Libro Vivo de Ciencia de Datos

Un enfoque intuitivo y práctico en machine learning, con análisis y preparación de datos, ¡apto para todas las edades!

por Pablo Casas

Libro Vivo de Ciencia de Datos

Pablo Casas

junio 2019

Libro Vivo de Ciencia de Datos

Prefacio

Acerca de esta edición

Finalmente disponible la versión en español del *Data Science Live Book*! El libro se abre sin barreras idiomáticas ante las personas de habla-hispana con ganas de aprender.

Esta publicación es una edición revisada tanto en gramática como en aspectos técnicos de la versión en inglés.

El *Data Science Live Book*, junto con dos artículos de como auto-publicar un libro usando bookdown, fueron premiados por RStudio en el [1st Bookdown Contest](#).

El objetivo es que puedan acercarse con un enfoque intutitivo y práctico al mundo de la ciencia con datos, el arte del descubrimiento.

Curso introductorio de R gratuito

Con el curso **[Desembarcando en R](#)**, podrán obtener una gentil introducción a los aspectos mas distintivos de R, mediante sus 11 laboratorios y 19 videos de programación cortos.

Es un curso complementario a este libro. Solo se requieren conocimientos básicos de programación.

Modalidad 100% práctica, se ven introducciones a las librerías R Base, `tidyverse`, `funModeling`, `Hmisc`, `randomForest`. Al final de cada laboratorio se plantean ejercicios (con su resolución).

Al finalizar, pueden calificarlo, compartir y hacer sus comentarios :)

¿Por qué leer este libro?

Este libro facilitará el entendimiento de problemas comunes en el análisis de datos y machine learning.

Construir un modelo predictivo es tan complejo como una línea de código en R:

```
mi_super_modelo=randomForest(target ~ var_1 + var_2, mis_datos_compl
```

Eso es todo.

Pero, en la práctica los datos tienen su suciedad. Debemos esculpirlos, como hace un artista, para exponer su información y encontrar respuestas (y nuevas preguntas).

Hay muchos desafíos para resolver, algunos conjuntos de datos requieren más *esculpido* que otros. Para dar un ejemplo, random forest no acepta valores vacíos, ¿qué hacemos entonces? ¿Quitamos las filas que tienen conflictos? ¿O transformamos los valores vacíos en otros valores? **¿Cuál es la implicancia**, en cualquier caso, para *mis* datos?

Además del problema de los valores vacíos, debemos enfrentar otras situaciones, como los valores extremos (outliers) que suelen sesgar no solamente el modelo en sí mismo, sino también la interpretación de los resultados finales. Es común "intentar adivinar" *cómo* interpreta el modelo predictivo cada variable (ordenando las mejores variables), y cuáles son los valores que aumentan (o disminuyen) la probabilidad de que ocurra algún evento (análisis numérico de variables).

Decidir el **tipo de datos** de las variables puede no ser algo menor. Una variable categórica *podría ser* numérica y viceversa, dependiendo del contexto, los datos, y el algoritmo mismo (algunos sólo pueden manejar un tipo de datos). Esta conversión también tiene sus propias implicancias en *cómo el modelo ve las variables*.

Este es un libro sobre la preparación de los datos, el análisis de los datos y

machine learning. Generalmente, en la literatura, la preparación de los datos no es un tema tan popular como la creación de modelos de machine learning.

El camino del aprendizaje

El libro tiene un enfoque altamente práctico, e intenta demostrar lo que dice. Por ejemplo, dice: *"Las variables trabajan en grupos."*, y luego encontrarán el código que apoya esa idea.

Prácticamente todos los capítulos pueden ser copiados y pegados y replicados por el lector para que extraiga sus propias conclusiones. Incluso, en las ocasiones que lo permitieron, el código o script propuesto (en lenguaje R) fue pensado genéricamente, para que pueda ser utilizado en escenarios reales, ya sea con fines de investigación o laborales.

La semilla de este libro fue la biblioteca de *R* `funModeling` que comenzó a tener una documentación didáctica que rápidamente se convirtió en este libro. Es didáctica porque hay una diferencia entre usar una simple función que grafica histogramas para analizar numéricamente la variable objetivo (`cross_plot`), y la explicación sobre cómo llegar a las conclusiones semánticas. La intención es aprender el concepto interno, para que puedan *exportar ese conocimiento* a otros lenguajes, como Python, Julia, etc.

Este libro, al igual que el desarrollo de un proyecto de datos, no es lineal. Los capítulos están relacionados entre sí. Por ejemplo, el capítulo sobre **valores faltantes** puede llevar al de **reducción de la cardinalidad en variables categóricas**. O pueden leer el capítuo sobre **tipos de datos** y luego cambiar la forma en la que lidian con valores faltantes.

Encontrarán referencias a otros sitios web para que puedan expandir su estudio, *este libro es sólo otro paso en el camino del aprendizaje.*

¿Este libro es para mí? ¿Podré entenderlo?

Si ya están en el campo de la ciencia de datos, probablemente no crean que es

para ustedes. Tomarán el código que necesiten, lo copiarán y pegarán si así lo desean, y listo.

Pero, si están empezando una carrera en la ciencia de datos, enfrentarán un problema común de la educación: *Tener respuestas a preguntas que aún no han sido formuladas.*

Definitivamente se acercarán al mundo de la ciencia de datos. Todo el código está bien comentado, por lo que no es necesario que sean programadores para entenderlo. Ese es el desafío de este libro, tratar de que su lectura resulte amistosa, usando la lógica, el sentido común y la intuición.

Lenguaje de programación

Podrían aprender un poco de R, pero puede resultar difícil aprenderlo directamente de este libro. Si quieren aprender programación en R, hay otros libros o cursos especializados en programación.

Es hora de la siguiente sección.

¿Las máquinas y la inteligencia artificial dominarán el mundo? 😱

Aunque es cierto que el poder de la computación está aumentando exponencialmente, la rebelión de las máquinas está lejos de ocurrir hoy en día.

Este libro trata de exponer problemas comunes al crear y manejar modelos predictivos, mostrando que cada decisión tiene su implicancia. También hay una relación con *soluciones de 1 solo click* y ¡voilà! El sistema de predicción está instalado y ejecutándose. Toda la preparación de datos, transformaciones, uniones de tablas, consideraciones de tiempo, ajustes finos, *etc* se resuelve en un solo paso.

Tal vez lo sea. De hecho, a medida que pasa el tiempo, existen técnicas más robustas que nos ayudan a automatizar tareas en el modelado predictivo. Pero, por si acaso, sería una buena práctica no confiar ciegamente en las soluciones de caja negra sin saber, por ejemplo, cómo el sistema *elige las mejores variables*,

cuál es el procedimiento interno para validar el modelo, cómo trata los valores extremos o raros, entre otros temas tratados en este libro.

Si están evaluando alguna plataforma de machine learning, algunos de los temas que se mencionan en este libro pueden ayudarlos a decidir cuál es la mejor opción. Intentando *abrir la caja negra*.

Es difícil tener una solución que se adapte a todos los casos. La intervención humana **es crucial** para tener un proyecto exitoso. En lugar de preocuparse por las máquinas, el punto es *cuál será el uso de esta tecnología*. La tecnología es *inocente*. Es el científico de datos quien establece los datos de entrada y da al modelo el objetivo necesario para aprender. Surgirán patrones, y algunos de ellos podrían ser perjudiciales para muchas personas. Tenemos que ser conscientes del objetivo final, como con cualquier otra tecnología.

> La máquina la hace el hombre, y es lo que el hombre hace con ella.

Por Jorge Drexler (músico, actor y doctor). Citada de la canción "Guitarra y vos".

Quizás, ¿podría esta ser la diferencia entre **machine learning** y **ciencia de datos**? ¿Una máquina que aprende vs. un ser humano haciendo ciencia con los datos?

Una pregunta abierta.

¿Qué necesito para empezar?

En términos generales, tiempo y paciencia. La mayoría de los conceptos son independientes del lenguaje, pero cuando vemos un ejemplo técnico lo probamos en **lenguaje R**, (R version 3.5.1 (2018-07-02)).

El libro utiliza las siguientes bibliotecas, (entre paréntesis se encuentra la versión del paquete):

```
## funModeling (1.7), dplyr (0.8.1), Hmisc (4.2.0)
```

```
## reshape2 (1.4.3), ggplot2 (3.1.1), caret (6.0.84)
## minerva (1.5.8), missForest (1.4), gridExtra (2.3)
## mice (3.5.0), Lock5Data (2.8), corrplot (0.84)
## RColorBrewer (1.1.2), infotheo (1.2.0)
```

El paquete `funModeling` fue el origen de este libro: comenzó como un conjunto de funciones para ayudar al científico de datos en sus tareas *diarias*. Ahora su documentación evolucionó y se convirtió en este libro ❤!

Instalen cualquiera de estas utilizando: `install.packages("PACKAGE_NAME")`.

El Entorno de Desarrollo Integrado (IDE por Integrated Development Environment en inglés) recomendado es **Rstudio**.

Este libro, tanto en PDF como formato web, fue creado en Rstudio, usando el increíble Bookdown.

Todo es gratis y de código abierto, Bookdown, R, Rstudio y este libro

Pueden revisar el detrás de escena de como fue generado el libro en bookdown, y como publicarlo en Amazon en: How to self-publish a book: A handy list of resources y How to self publish a book: customizing Bookdown.

¡Ojalá lo disfruten!

Comunidad de R en español

Foro de preguntas y respuestas de R en español: Datos en R.

argentinaR.org | twitter - Una de las comunidades de R en español donde se organizan encuentros sobre IA, machine learning y cualquier tópico relacionado en R. Además posee un blog de R en español.

¿Cómo puedo contactar al autor?

Si quieren decir *hola*, contribuir comentando que alguna sección no está bien

explicada, sugerir un nuevo tema o compartir una buena experiencia que tuvieron al aplicar algún concepto explicado aquí, pueden enviarme un e-mail a:

pcasas.biz (arroba) gmail.com. Estoy aprendiendo constantemente, así que es lindo intercambiar conocimiento y estar en contacto con colegas.

- Twitter
- Linkedin
- Github
- Blog Data Science Heroes

Además, pueden referirse a los repositorios de **Github** para ambos, el libro y `funModeling`, para reportar bugs, enviar sugerencias, nuevas ideas, etc:

- funModeling
- Libro Vivo de Ciencia de Datos

Agradecimientos

Agradecimientos especiales a mis mentores en este mundo de los datos, Miguel Spindiak y Marcelo Ferreyra.

Revisión técnica del libro: Pablo Seibelt (aka The Sicarul) . Gracias por tu ayuda sincera y desinteresada.

El arte de la tapa fue hecho por: Bárbara Muñoz.

Este libro está dedicado a *Los nadies*, un cuento corto escrito por Eduardo Galeano, y a mis padres.

Información del libro

Traducción del inglés al español por: Valentina Varas. Revisión por Pablo Casas.

Publicado original en inglés: Data Science Live Book

Publicación en español: LibroVivoDeCienciaDeDatos.ai.

ISBN: 978-987-783-715-5 (versión eBook).

1 Introduction

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter 1. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter 5.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

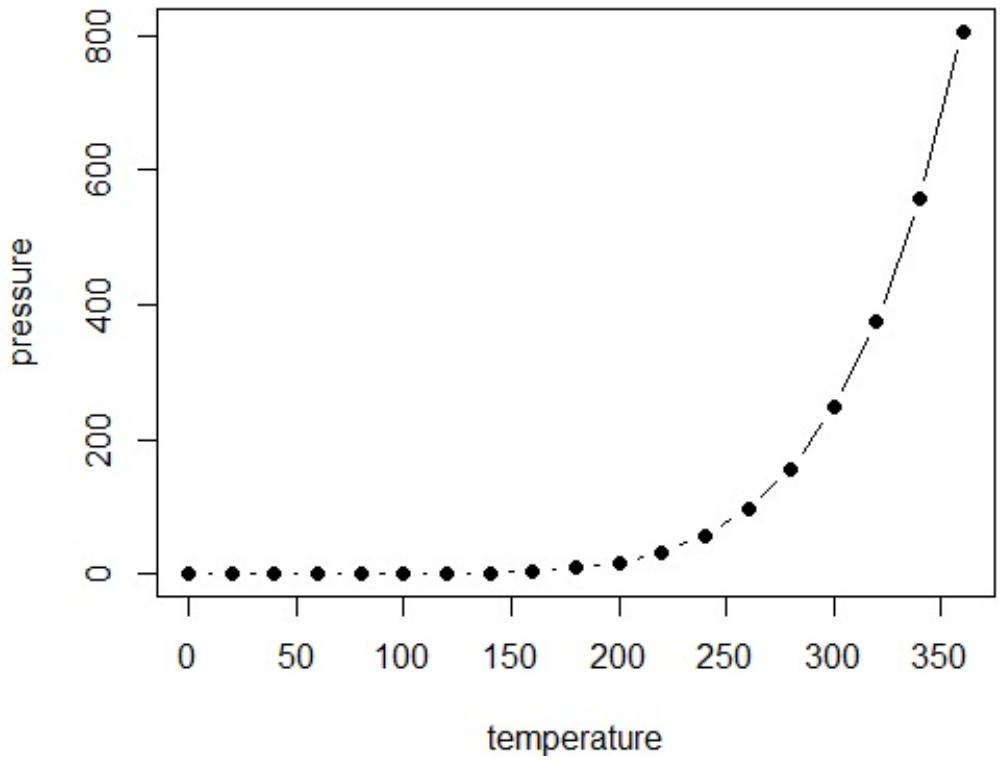


Figure 1.1: Here is a nice figure!

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 1.1. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table 1.1.

```
knitr::kable(
  head(iris, 20), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

Table 1.1: Here is a nice table!

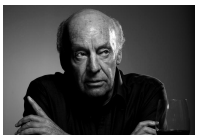
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

You can write citations, too. For example, we are using the **bookdown** package (Xie 2019) in this sample book, which was built on top of R Markdown and **knitr** (**???**).

2 Análisis exploratorio de datos

Escuchando a los números :)

2.1 Análisis numérico, La voz de los números



> "La voz de los números" -- una metáfora de Eduardo Galeano. Escritor y novelista.

Los datos que exploramos podrían ser como jeroglíficos egipcios sin una interpretación correcta. El análisis numérico es el primer paso de una serie de etapas iterativas en la búsqueda de lo que los datos nos quieren decir, si somos lo suficientemente pacientes para escucharlos.

Este capitulo abarcará, con unas pocas funciones, el análisis numérico completo de datos. Esto debería ser el primer paso en cualquier proyecto de datos, donde comenzamos sabiendo cuáles son los tipos correctos de datos y exploramos las distribuciones de variables numéricas y categóricas.

También se enfocará en la extracción de conclusiones semánticas, que será útil a la hora de escribir informes para audiencias no especializadas.

¿Qué vamos a repasar en este capítulo?

- **Estado de salud de un conjunto de datos**:
  - Obtener métricas como el número total de filas, columnas, tipos de datos, ceros, y valores faltantes
  - Cómo cada uno de los ítems mencionados impactan sobre diferentes análisis
  - Cómo filtrar y operar rápidamente sobre (y con) ellos para limpiar los datos
- **Análisis univariado en una variable categórica**:
  - Frecuencia, porcentaje, valor acumulativo, y gráficos coloridos

- **Análisis univariado con variables numéricas**:
  - Percentil, dispersión, desvío estándar, promedio, valores máximos y mínimos
  - Percentil vs. cuantil vs. cuartil
  - Curtosis, asimetría, rango intercuartil, coeficiente de variación
  - Gráficos de distribuciones
  - **Estudio de caso completo** basado en *"Data World"*, preparación de datos, y análisis de datos

Repaso de funciones utilizadas en el capítulo:

- `df_status(data)`: Análisis de la estructura del conjunto de datos
- `describe(data)`: Análisis numérico y categórico (cuantitativo)
- `freq(data)`: Análisis categórico (cuantitativo y gráfico)
- `profiling_num(data)`: Análisis para variables numéricas (cuantitativas)
- `plot_num(data)`: Análisis para variables numéricas (gráficos)

Note: `describe` se encuentra en el paquete `Hmisc` mientras que el resto de las funciones se encuentran en `funModeling`.

2.1.1 Estado de salud de un conjunto de datos

La cantidad de ceros, NA, Inf, valores únicos al igual que el tipo de datos puede llevar a un buen o mal modelo. Aquí, un acercamiento al primer paso del modelado de datos.

Primero, vamos a cargar las bibliotecas `funModeling` y `dplyr`.

```
# ¡Cargar funModeling!
library(funModeling)
library(dplyr)
data(heart_disease)
```

2.1.1.1 Buscar valores faltantes, ceros, tipos de datos y valores únicos

Probablemente uno de los primeros pasos, cuando recibimos un nuevo conjunto de datos para analizar, es verificar si existen valores faltantes (`NA` en **R**) y el tipo de datos.

La función `df_status` incluida en `funModeling` nos puede ayudar mostrándonos estas cifras en valores relativos y porcentuales. También obtiene las estadísticas de infinitos y ceros.

```
# Analizar los datos ingresados
df_status(heart_disease)
```

	variable	q_zeros	p_zeros	q_na	p_na	q_inf	p_inf	type	unique
1	age	0	0.00	0	0.00	0	0	integer	41
2	gender	0	0.00	0	0.00	0	0	factor	2
3	chest_pain	0	0.00	0	0.00	0	0	factor	4
4	resting_blood_pressure	0	0.00	0	0.00	0	0	integer	50
5	serum_cholestoral	0	0.00	0	0.00	0	0	integer	152
6	fasting_blood_sugar	258	85.15	0	0.00	0	0	factor	2
7	resting_electro	151	49.83	0	0.00	0	0	factor	3
8	max_heart_rate	0	0.00	0	0.00	0	0	integer	91
9	exer_angina	204	67.33	0	0.00	0	0	integer	2
10	oldpeak	99	32.67	0	0.00	0	0	numeric	40
11	slope	0	0.00	0	0.00	0	0	integer	3
12	num_vessels_flour	176	58.09	4	1.32	0	0	integer	4
13	thal	0	0.00	2	0.66	0	0	factor	3
14	heart_disease_severity	164	54.13	0	0.00	0	0	integer	5
15	exter_angina	204	67.33	0	0.00	0	0	factor	2
16	has_heart_disease	0	0.00	0	0.00	0	0	factor	2

Figure 2.1: Estado de salud de un conjunto de datos

- `q_zeros`: cantidad de ceros (`p_zeros`: en porcentaje)
- `q_inf`: cantidad de valores infinitos (`p_inf`: en porcentaje)
- `q_na`: cantidad de NA (`p_na`: en porcentaje)
- `type`: factor o numérico
- `unique`: cantidad de valores únicos

2.1.1.2 ¿Por qué son importantes estas métricas?

- **Ceros**: Las variables con **muchos ceros** pueden no ser útiles para modelar y, en algunos casos, pueden sesgar dramáticamente el modelo.
- **NA**: Varios modelos excluyen automáticamente las filas que tienen valores NA (**random forest** por ejemplo). En consecuencia, el modelo final puede resultar sesgado debido a varias filas faltantes por una sola variable. Por ejemplo, si los datos contienen tan sólo una de las 100 variables con el 90% de datos NA, el modelo se ejecutará con sólo el 10% de las filas originales.
- **Inf**: Los valores infinitos pueden ocasionar un comportamiento inesperado en algunas funciones en R.
- **Type**: Algunas variables están codificadas como números, pero son códigos o categorías y los modelos **no las manejan** de la misma manera.
- **Unique**: Las variables factor/categóricas con un alto número de valores diferentes (~30) tienden a sobreajustar si las categorías tienen una baja

cardinalidad (**árboles de decisión,** por ejemplo).

2.1.1.3 Filtrar casos no deseados

La función `df_status` toma un data frame y genera una *tabla de estado* que nos puede ayudar a quitar rápidamente atributos (o variables) en base a todas las métricas descriptas en la sección anterior. Por ejemplo:

Quitar variables con un *alto número* de ceros

```
# Analizar los datos ingresados
my_data_status=df_status(heart_disease, print_results = F)

# Quitar las variables que tienen un 60% de valores cero
vars_to_remove=filter(my_data_status, p_zeros > 60)  %>% .$variable
vars_to_remove
```

```
## [1] "fasting_blood_sugar" "exer_angina"         "exter_angina"
```

```
# Conservar todas las columnas excepto aquellas presentes en el vect
heart_disease_2=select(heart_disease, -one_of(vars_to_remove))
```

Ordenar datos según el porcentaje de ceros

```
arrange(my_data_status, -p_zeros) %>% select(variable, q_zeros, p_ze
```

```
##                  variable q_zeros p_zeros
## 1     fasting_blood_sugar     258   85.15
## 2             exer_angina     204   67.33
## 3            exter_angina     204   67.33
## 4       num_vessels_flour     176   58.09
## 5  heart_disease_severity     164   54.13
## 6         resting_electro     151   49.83
## 7                 oldpeak      99   32.67
## 8                     age       0    0.00
## 9                  gender       0    0.00
```

```
## 10             chest_pain       0    0.00
## 11 resting_blood_pressure       0    0.00
## 12      serum_cholestoral       0    0.00
## 13         max_heart_rate       0    0.00
## 14                  slope       0    0.00
## 15                   thal       0    0.00
## 16      has_heart_disease       0    0.00
```

El mismo razonamiento aplica cuando queremos quitar (o conservar) aquellas variables que estén por encima o por debajo de determinado umbral. Por favor revisen el capítulo de valores faltantes para obtener más informaación sobre las implicancias de trabajar con variables que contienen valores faltantes.

2.1.1.4 Profundizar en estos temas

Los valores que devuelve la función `df_status` son abordados en profundidad en otros capítulos:

- **Valores faltantes** el tratamiento, análisis e imputación de (NA) son abordados en el capítulo Datos faltantes.
- **Tipos de datos**, las conversiones e implicancias de trabajar con distintos tipos de datos y más temas son abordados en el capítulo Tipos de datos.
- Un alto número de **valores únicos** es sinónimo de variables de alta cardinalidad. Estudiamos esta situación en los siguientes capítulos:
  - Variables de alta cardinalidad en estadística descriptiva
  - Variables de alta cardinalidad en modelado predictivo

2.1.1.5 Obtener otras estadísticas comunes: cantidad total de filas, cantidad total de columnas y nombres de columnas:

```
# Cantidad total de filas
nrow(heart_disease)
```

```
## [1] 303
```

```
# Cantidad total de columnas
ncol(heart_disease)
```

```
## [1] 16
```

```
# Nombres de columnas
colnames(heart_disease)
```

```
##  [1] "age"                    "gender"
##  [3] "chest_pain"             "resting_blood_pressure"
##  [5] "serum_cholestoral"      "fasting_blood_sugar"
##  [7] "resting_electro"        "max_heart_rate"
##  [9] "exer_angina"            "oldpeak"
## [11] "slope"                  "num_vessels_flour"
## [13] "thal"                   "heart_disease_severity"
## [15] "exter_angina"           "has_heart_disease"
```

2.1.2 Análisis de variables categóricas

Asegúrense de tener la última version de 'funModeling' (>= 1.6).

La función `freq` simplifica el análisis de frecuencia o distribución. Dicha función vuelca la distribución en una tabla y en un gráfico (por defecto) y muestra la distribución de números absolutos y relativos.

Si desean obtener la distribución de dos variables:

```
freq(data=heart_disease, input = c('thal','chest_pain'))
```

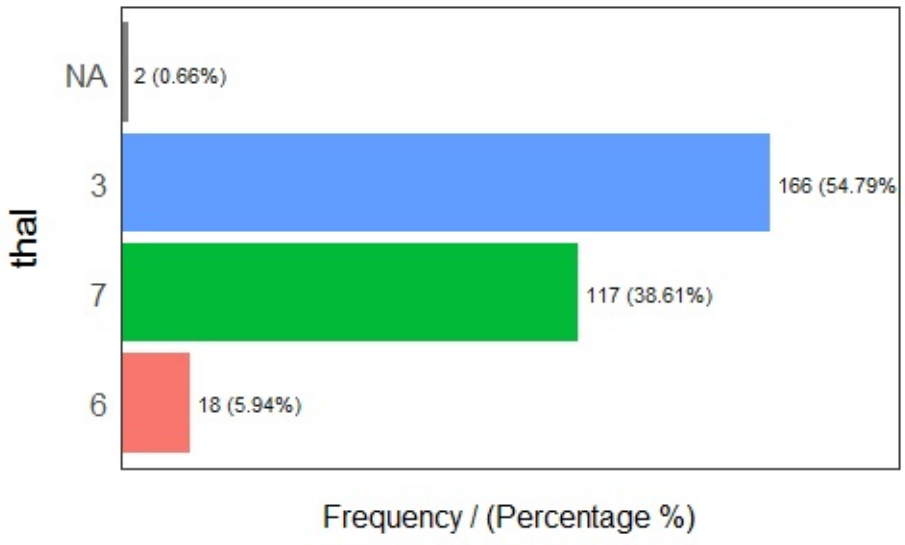


Figure 2.2: Análisis de frecuencia 1

```
##   thal frequency percentage cumulative_perc
## 1    3       166      54.79           54.79
## 2    7       117      38.61           93.40
## 3    6        18       5.94           99.34
## 4 <NA>         2       0.66          100.00
```

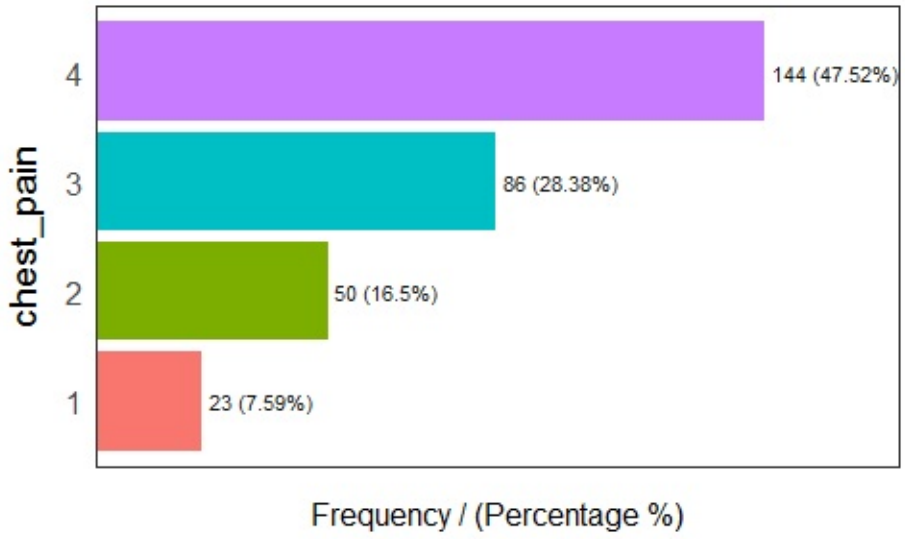


Figure 2.3: Análisis de frecuencia 2

```
##   chest_pain frequency percentage cumulative_perc
## 1          4       144      47.52           47.52
## 2          3        86      28.38           75.90
## 3          2        50      16.50           92.40
```

```
## 4          1        23       7.59          100.00
```

```
## [1] "Variables processed: thal, chest_pain"
```

Al igual que en las demás funciones de `funModeling`, si falta `input`, entonces la función se ejecutará para todas las variables factor o de carácter que estén presentes en un data frame dado:

```
freq(data=heart_disease)
```

Si sólo queremos obtener la tabla sin el gráfico, entonces configuramos el parámetro `plot` como `FALSE`. El ejemplo `freq` también puede trabajar con una **variable singular**. Por *defecto*, los valores `NA` **son considerados** tanto en la tabla como en el gráfico. Si es necesario excluir los valores `NA`, entonces ingresen `na.rm = TRUE`. Verán ambos ejemplos a continuación:

```
freq(data=heart_disease$thal, plot = FALSE, na.rm = TRUE)
```

Si sólo se brinda una variable, entonces `freq` generará la tabla; por lo tanto, es fácil realizar algunos cálculos en base a las variables que brinda.

- Por ejemplo, para visualizar las categorías que representan la mayor parte del 80% (en base a `cumulative_perc < 80`).
- Para obtener las categorías que pertenecen a la **cola larga**, es decir, filtrando por `percentage < 1` e identificando aquellas categorías que aparecen menos del 1% de las veces.

Además, al igual que con las demás funciones para graficar que están incluidas en el paquete, si es necesario exportar gráficos, agreguen el parámetro `path_out`, que creará la carpeta si aún no ha sido creada.

```
freq(data=heart_disease, path_out='my_folder')
```

2.1.2.0.1 Análisis

Los resultados están ordenados según la variable `frequency`, que rápidamente analiza las categorías con mayor frecuencia y qué porcentaje representan

(variable `cummulative_perc`). En términos generales, a los seres humanos nos gusta el orden. Si las variables no están ordenadas, nuestros ojos empiezan a moverse por todas las barras para compararlas y nuestros cerebros ubican cada barra en relación a las demás.

Comprueben la diferencia entre los mismos datos ingresados, primero sin ordenar y luego ordenados:

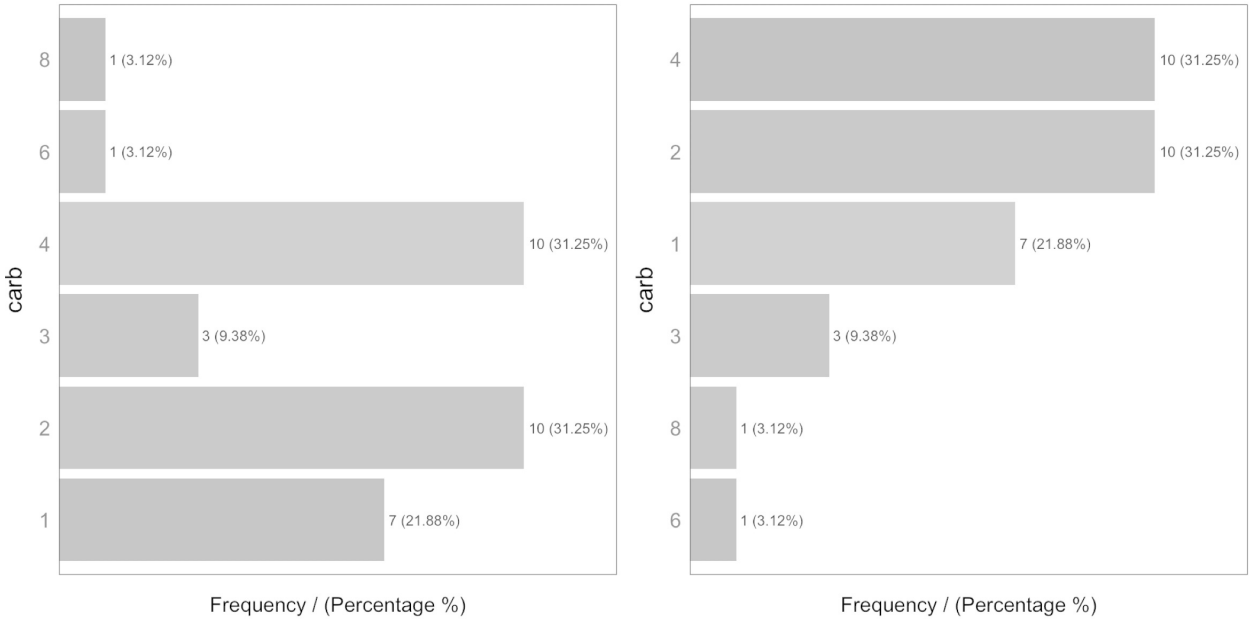


Figure 2.4: Orden y belleza

En general, suele haber unas pocas categorías que aparecen la mayor parte del tiempo.

Pueden encontrar un análisis más completo en Variables de alta cardinalidad en estadística descriptiva

2.1.2.1 Introduciendo la función `describe`

Esta función está incluida en el paquete `Hmisc` y nos permite analizar rápidamente un conjunto de datos completo para variables númericas y categóricas. En este caso, seleccionaremos sólo dos variables y analizaremos el resultado.

```
# Conservar solamente las dos variables que utilizaremos en este eje
heart_disease_3=select(heart_disease, thal, chest_pain)

# ¡Analizar los datos!
describe(heart_disease_3)
```

```
## heart_disease_3
##
##  2  Variables      303  Observations
## ---------------------------------------------------------------------
## thal
##        n  missing distinct
##      301        2        3
##
## Value          3     6     7
## Frequency    166    18   117
## Proportion 0.551 0.060 0.389
## ---------------------------------------------------------------------
## chest_pain
##        n  missing distinct
##      303        0        4
##
## Value          1     2     3     4
## Frequency     23    50    86   144
## Proportion 0.076 0.165 0.284 0.475
## ---------------------------------------------------------------------
```

Donde:

- `n`: cantidad de filas que no son `NA`. En este caso, indica que hay `301` pacientes que contienen un número.
- `missing`: cantidad de valores faltantes. Al sumar este indicador y `n`, obtenemos la cantidad total de filas.
- `unique`: cantidad de valores únicos (o distintos).

El resto de la información es bastante similar a la función `freq` e indica entre paréntesis el número total en valores relativos y absolutos para cada categoría diferente.

2.1.3 Análisis de variables numéricas

Esta sección se divide en dos partes:

- Parte 1: Introducción al caso de estudio "World Data"
- Parte 2: Hacer el análisis numérico en R

Si no desean saber cómo se calcula la etapa de preparación de datos de Data World, pueden saltar a "Parte 2: Haciendo el análisis numérico en R", cuando el análisis comenzó.

2.1.3.1 Parte 1: Introducción al caso de estudio "World Data"

Este estudio contiene muchos indicadores sobre el desarrollo mundial. Independientemente del ejemplo de análisis numérico, la idea es proveer una tabla lista para usar para sociólogos, investigadores, etc. interesados en analizar este tipo de datos.

La fuente original de los datos es: http://databank.worldbank.org. Allí encontrarán un diccionario de datos que explica todas las variables.

Primero, tenemos que hacer una limpieza de datos. Vamos a conservar el valor más reciente de cada indicador.

```
library(Hmisc)

# Cargar datos desde el repositorio de libros sin alterar el formato
data_world=read.csv(file = "https://goo.gl/2TrDgN", header = T, stri

# Excluir los valores faltantes en Series.Code. Los datos descargado
data_world=filter(data_world, Series.Code!="")

# La función mágica que conserva los valores más recientes de cada m
max_ix<-function(d)
{
  ix=which(!is.na(d))
  res=ifelse(length(ix)==0, NA, d[max(ix)])
  return(res)
}

data_world$newest_value=apply(data_world[,5:ncol(data_world)], 1, FU

# Visualizar las primeras tres filas
head(data_world, 3)
```

```
##                                          Series.Name       Series
## 1 Population living in slums (% of urban population) EN.POP.SLUM.
## 2 Population living in slums (% of urban population) EN.POP.SLUM.
```

```
## 3 Population living in slums (% of urban population) EN.POP.SLUM.
##   Country.Name Country.Code X1990..YR1990. X2000..YR2000. X2007..
## 1  Afghanistan          AFG             NA             NA
## 2      Albania          ALB             NA             NA
## 3      Algeria          DZA           11.8             NA
##   X2008..YR2008. X2009..YR2009. X2010..YR2010. X2011..YR2011.
## 1             NA             NA             NA             NA
## 2             NA             NA             NA             NA
## 3             NA             NA             NA             NA
##   X2012..YR2012. X2013..YR2013. X2014..YR2014. X2015..YR2015.
## 1             NA             NA           62.7             NA
## 2             NA             NA             NA             NA
## 3             NA             NA             NA             NA
##   X2016..YR2016. newest_value
## 1             NA         62.7
## 2             NA           NA
## 3             NA         11.8
```

Las columnas `Series.Name` y `Series.Code` son los indicadores a analizar. `Country.Name` y `Country.Code` son los países. Cada fila representa una combinación única de país e indicador. Las columnas restantes, `X1990..YR1990.` (año 1990),`X2000..YR2000.` (año 2000), `X2007..YR2007.` (año 2007), y sucesivamente indican el valor de cada métrica para ese año, cada columna corresponde a un año.

2.1.3.2 Tomar decisiones como científicos de datos

Hay muchas celdas `NAs` porque algunos países no cuentan con las mediciones de los indicadores en esos años. En este punto, debemos **tomar una decisión** como científicos de datos. Probablemente no tomemos la decisión óptima si no consultamos con un experto, por ejemplo, un sociólogo.

¿Qué hacemos con los valores `NA`? En este caso, vamos a conservar el **valor más reciente** para todos los indicadores. Quizás esta no sea la mejor manera de extraer conclusiones para un *paper* académico, ya que vamos a comparar algunos países que cuentan con información actualizada al 2016 con países cuyos datos fueron actualizados por última vez en el 2009. Comparar todos los indicadores con los datos más recientes es un enfoque apropiado para un primer análisis.

Otra solución podría haber sido conservar el valor más reciente solamente si este es de los últimos cinco años. Esto reduciría la cantidad de países a analizar.

Estas preguntas son imposibles de responder para un *sistema de inteligencia artificial*, no obstante la decisión puede influir drásticamente en los resultados.

La última transformación

El próximo paso convertirá la última tabla de formato *largo* a *ancho*. En otras palabras, cada fila representará un país y cada columna un indicador (gracias a la última transformación que tiene el *valor más reciente* para cada combinación de indicador-país).

Los nombres de los indicadores son poco claros, por lo que "traduciremos" algunos de ellos.

```
# Obtener la lista de descripciones de indicadores.
names=unique(select(data_world, Series.Name, Series.Code))
head(names, 5)
```

```
##                                                Series.Name       Seri
## 1   Population living in slums (% of urban population) EN.POP.SLU
## 218                     Income share held by second 20%    SI.DST.
## 435                      Income share held by third 20%    SI.DST.
## 652                     Income share held by fourth 20%    SI.DST.
## 869                    Income share held by highest 20%    SI.DST.
```

```
# Convertir algunos
df_conv_world=data.frame(
  new_name=c("urban_poverty_headcount",
             "rural_poverty_headcount",
             "gini_index",
             "pop_living_slums",
             "poverty_headcount_1.9"),
  Series.Code=c("SI.POV.URHC",
                "SI.POV.RUHC",
                "SI.POV.GINI",
                "EN.POP.SLUM.UR.ZS",
                "SI.POV.DDAY"),
  stringsAsFactors = F)

# Agregar el nuevo valor del indicador
```

```
data_world_2 = left_join(data_world,
                         df_conv_world,
                         by="Series.Code",
                         all.x=T)

data_world_2 =
  mutate(data_world_2, Series.Code_2=
           ifelse(!is.na(new_name),
                  as.character(data_world_2$new_name),
                  data_world_2$Series.Code)
         )
```

El significado de cualquiera de los indicadores puede cotejarse en data.worldbank.org. Por ejemplo, si queremos saber qué significa `EN.POP.SLUM.UR.ZS`, ingresamos a: http://data.worldbank.org/indicator/EN.POP.SLUM.UR.ZS

```
# El paquete 'reshape2' contiene tanto la función 'dcast' como 'melt
library(reshape2)

data_world_wide=dcast(data_world_2, Country.Name  ~ Series.Code_2, v
```

Nota: Para entender más acerca de los formatos `largo` y `ancho` utilizando el paquete `reshape2`, y cómo convertir de uno a otro, por favor diríjanse a: http://seananderson.ca/2013/10/19/reshape.html.

Ahora tenemos la tabla final para analizar:

```
# Visualizar las primeras tres filas
head(data_world_wide, 3)
```

```
##   Country.Name gini_index pop_living_slums poverty_headcount_1.9
## 1  Afghanistan         NA              62.7                    NA
## 2      Albania      28.96                NA                  1.06
## 3      Algeria         NA              11.8                    NA
##   rural_poverty_headcount SI.DST.02ND.20 SI.DST.03RD.20 SI.DST.04
## 1                    38.3             NA             NA
## 2                    15.3          13.17          17.34
## 3                     4.8             NA             NA
##   SI.DST.05TH.20 SI.DST.10TH.10 SI.DST.FRST.10 SI.DST.FRST.20 SI.
```

```
## 1             NA             NA             NA             NA
## 2          37.82          22.93           3.66           8.85
## 3             NA             NA             NA             NA
##   SI.POV.GAP2 SI.POV.GAPS SI.POV.NAGP SI.POV.NAHC SI.POV.RUGP SI.
## 1          NA          NA         8.4         35.8         9.3
## 2        1.43        0.22         2.9         14.3         3.0
## 3          NA          NA          NA          5.5         0.8
##   SI.SPR.PC40 SI.SPR.PC40.ZG SI.SPR.PCAP SI.SPR.PCAP.ZG
## 1          NA             NA          NA             NA
## 2        4.08        -1.2205        7.41        -1.3143
## 3          NA             NA          NA             NA
##   urban_poverty_headcount
## 1                    27.6
## 2                    13.6
## 3                     5.8
```

2.1.3.3 Parte 2: Hacer el análisis numérico en R

Utilizaremos las siguientes funciones:

- `describe` de `Hmisc`
- `profiling_num` (análisis univariado completo), y `plot_num` (histogramas) de `funModeling`

Seleccionaremos solamente dos variables como ejemplo:

```
library(Hmisc) # contiene la función `describe`

vars_to_profile=c("gini_index", "poverty_headcount_1.9")
data_subset=select(data_world_wide, one_of(vars_to_profile))

# Utilizar la función `describe` en con junto de datos completo.
# Puede ejecutarse con una variable; por ejemplo, describe(data_subs

describe(data_subset)
```

```
## data_subset
##
##  2  Variables      217  Observations
## ---------------------------------------------------------------
```

```
## gini_index
##        n  missing distinct     Info     Mean      Gmd      .05
##      140       77      136        1     38.8    9.594    26.81
##      .25      .50      .75      .90      .95
##    32.35    37.69    43.92    50.47    53.53
##
## lowest : 24.09 25.59 25.90 26.12 26.13, highest: 56.24 60.46 60.7
## ------------------------------------------------------------------
## poverty_headcount_1.9
##        n  missing distinct     Info     Mean      Gmd      .05
##      116      101      107        1    18.33    23.56    0.025
##      .25      .50      .75      .90      .95
##    1.052    6.000   33.815   54.045   67.328
##
## lowest :  0.00  0.01  0.03  0.04  0.06, highest: 68.64 68.74 70.9
## ------------------------------------------------------------------
```

Tomando `poverty_headcount_1.9` (*Índice de pobreza de US$1.90 por día es el porcentaje de la población que vive con menos de US$1.90 por día a precios internacionales de 2011*.), lo podemos describir como:

- `n`: cantidad de filas que no son `NA`. En este caso, indica que hay `116` países que contienen un número.
- `missing`: cantidad de valores faltantes. Al sumar este indicador y `n`, obtenemos la cantidad total de filas. Casi la mitad de los países no tienen datos.
- `unique`: cantidad de valores únicos (o distintos).
- `Info`: un estimador de la cantidad de información presente en la variable y que no es importante en este punto.
- `Mean`: el clásico promedio o media.
- Números: `.05`, `.10`, `.25`, `.50`, `.75`, `.90` y `.95` son percentiles. Estos valores son muy útiles ya que nos ayudan a describir la distribución. Cubriremos este tema en profundidad más adelante, pero, por ejemplo, `.05` es el 5to percentil.
- `lowest` y `highest`: los cinco valores mínimos/máximos. Aquí podemos detectar valores atípicos y errores de datos. Por ejemplo, si la variable representa un porcentaje, entonces no puede contener valores negativos.

La siguiente función es `profiling_num`, que toma un data frame y devuelve una *gran* tabla en la que es fácil sentirse abrumado por un *mar de métricas*. Es

similar a lo que vemos en la película *Matrix*.

Figure 2.5: La matrix de datos

Imagen de la película "Matrix" (1999), dirigida por las hermanas Wachowski.

La idea de la siguiente tabla es brindarle al usuario un **conjunto completo de métricas** para que él o ella pueda decidir cuáles utilizar para el estudio.

Nota: Detrás de cada métrica hay mucha teoría estadística. Aquí cubriremos solamente un enfoque acotado y **muy simplificado** para introducir los conceptos.

```
library(funModeling)

# El análisis numérico completo de una función automáticamente exclu
profiling_num(data_world_wide)
```

```
##                   variable mean std_dev variation_coef   p_01   p
## 1               gini_index 38.8    8.49           0.22 25.711 26.
## 2         pop_living_slums 45.7   23.66           0.52  6.830 10.
## 3    poverty_headcount_1.9 18.3   22.74           1.24  0.000  0.
## 4  rural_poverty_headcount 41.2   21.91           0.53  2.902  6.
## 5           SI.DST.02ND.20 10.9    2.17           0.20  5.568  7.
## 6           SI.DST.03RD.20 15.2    2.03           0.13  9.137 11.
## 7           SI.DST.04TH.20 21.5    1.49           0.07 16.286 18.
## 8           SI.DST.05TH.20 45.9    7.14           0.16 35.004 36.
## 9           SI.DST.10TH.10 30.5    6.75           0.22 20.729 21.
## 10          SI.DST.FRST.10  2.5    0.87           0.34  0.916  1.
## 11          SI.DST.FRST.20  6.5    1.87           0.29  2.614  3.
## 12             SI.POV.2DAY 32.4   30.64           0.95  0.061  0.
## 13             SI.POV.GAP2 14.2   16.40           1.16  0.012  0.
## 14             SI.POV.GAPS  6.9   10.10           1.46  0.000  0.
## 15             SI.POV.NAGP 12.2   10.12           0.83  0.421  1.
## 16             SI.POV.NAHC 30.7   17.88           0.58  1.842  6.
## 17             SI.POV.RUGP 15.9   11.83           0.75  0.740  1.
## 18             SI.POV.URGP  8.3    8.24           0.99  0.300  0.
```

```
## 19             SI.SPR.PC40 10.3    9.75           0.95  0.857  1.
## 20          SI.SPR.PC40.ZG  2.0    3.62           1.85 -6.232 -3.
## 21             SI.SPR.PCAP 21.1   17.44           0.83  2.426  3.
## 22          SI.SPR.PCAP.ZG  1.5    3.21           2.20 -5.897 -3.
## 23 urban_poverty_headcount 23.3   15.06           0.65  0.579  3.
##      p_25 p_50 p_75 p_95 p_99 skewness kurtosis  iqr       range_
## 1  32.348 37.7 43.9 53.5 60.9    0.552      2.9 11.6  [25.71, 60.
## 2  25.175 46.2 65.6 83.4 93.4    0.087      2.0 40.5  [6.83, 93.4
## 3   1.052  6.0 33.8 67.3 76.2    1.125      2.9 32.8     [0, 76.1
## 4  25.250 38.1 57.6 75.8 81.7    0.051      2.0 32.3    [2.9, 81.
## 5   9.527 11.1 12.6 14.2 14.6   -0.411      2.7  3.1  [5.57, 14.5
## 6  13.877 15.5 16.7 17.9 18.1   -0.876      3.8  2.8  [9.14, 18.1
## 7  20.758 21.9 22.5 23.0 23.4   -1.537      5.6  1.8 [16.29, 23.3
## 8  40.495 44.8 49.8 58.1 65.9    0.738      3.3  9.3    [35, 65.8
## 9  25.710 29.5 34.1 42.2 50.6    0.905      3.6  8.4 [20.73, 50.6
## 10  1.885  2.5  3.2  3.9  4.3    0.043      2.2  1.4   [0.92, 4.3
## 11  5.092  6.5  8.0  9.4 10.0   -0.119      2.2  2.9     [2.61, 1
## 12  3.828 20.3 63.2 84.6 90.3    0.536      1.8 59.4  [0.06, 90.3
## 13  1.305  5.5 26.3 48.5 56.2    1.063      2.9 25.0  [0.01, 56.1
## 14  0.287  1.4 10.3 31.5 38.3    1.654      4.7 10.0     [0, 38.2
## 15  4.500  8.6 16.9 32.4 36.7    1.129      4.0 12.4  [0.42, 36.7
## 16 16.350 26.6 44.2 63.0 71.6    0.529      2.4 27.9  [1.84, 71.6
## 17  5.950 13.6 22.5 37.7 45.3    0.801      3.0 16.5  [0.74, 45.2
## 18  2.900  6.3  9.9 25.1 35.2    2.316      9.8  7.0   [0.3, 35.1
## 19  3.475  6.9 12.7 28.9 35.4    1.251      3.4  9.2  [0.86, 35.3
## 20 -0.084  1.7  4.6  7.9  9.0   -0.294      3.3  4.7     [-6.23,
## 21  8.003 15.3 25.4 52.8 67.2    1.132      3.3 17.4  [2.43, 67.1
## 22 -0.486  1.3  3.6  7.0  8.5   -0.018      3.6  4.0   [-5.9, 8.4
## 23 12.708 20.1 31.2 51.0 61.8    0.730      3.0 18.5  [0.58, 61.7
##          range_80
## 1  [27.58, 50.47]
## 2    [12.5, 75.2]
## 3   [0.08, 54.05]
## 4  [13.99, 71.99]
## 5    [8.28, 13.8]
## 6   [12.67, 17.5]
## 7  [19.73, 22.81]
## 8  [36.99, 55.24]
## 9  [22.57, 39.89]
## 10   [1.48, 3.67]
## 11   [3.99, 8.89]
## 12  [0.79, 78.29]
## 13  [0.16, 40.85]
## 14  [0.02, 23.45]
## 15     [1.85, 27]
## 16  [9.86, 58.22]
## 17    [3.3, 32.2]
```

```
## 18    [1.3, 19.1]
## 19  [1.81, 27.63]
## 20  [-2.64, 6.48]
## 21  [4.25, 49.22]
## 22  [-2.07, 5.17]
## 23  [5.98, 46.11]
```

Cada indicador tiene su *raison d'être*:

- `variable`: nombre de la variable
- `mean`: el famoso promedio o media
- `std_dev`: desvío estándar, una medida de **dispersión** o **extensión** con respecto al valor promedio. Un valor cerca de `0` indica que casi no hay variación (por lo que parece más una constante); por otro lado, es más difícil definir qué sería un valor *alto*, pero podemos decir que a mayor variación, mayor dispersión. *El caos se asemeja a un desvío estándar de valor infinito*. La unidad es la misma que la del promedio para que puedan compararse.
- `variation_coef`: coeficiente de variación=`std_dev`/`mean`. Dado que el `std_dev` es un valor absoluto, es bueno tener un indicador que lo exprese en un valor relativo, comparando el `std_dev` contra el `mean` Un valor de `0.22` indica que el `std_dev` es el 22% del `mean`. Si estuviera cerca de `0` entonces la variable estaría más centrada cerca del promedio. Si comparamos dos clasificadores, entonces es probable que prefiramos el que tenga menores `std_dev` y `variation_coef` por su precisión.
- `p_01`, `p_05`, `p_25`, `p_50`, `p_75`, `p_95`, `p_99`: **Percentiles** en 1%, 5%, 25%, y así sucesivamente. Encontrarán un repaso completo sobre percentiles más adelante en este capítulo.

Para leer una explicación completa sobre percentiles, por favor diríjanse a: Anexo 1: La magia de los percentiles.

- `skewness`: es una medida de *asimetría*. Un valor cercano a **0** indica que la distribución de los datos es *igual* hacia ambos lados (o simétrica) con respecto al promedio. Un **valor positivo** implica una larga cola hacia la derecha, mientras que un **valor negativo** significa lo opuesto.

Después de esta sección, revisen la asimetría en los gráficos. La variable `pop_living_slums` está cerca de 0 ("igualmente" distribuida), `poverty_headcount_1.9` es positiva (cola hacia la derecha), y `SI.DST.04TH.20` es negativa (cola hacia la izquierda). Cuanto más lejos esté la asimetría de 0, más probable es que la distribución tenga **valores atípicos**.

- `kurtosis`: describe las **colas** de la distribución; dicho en términos simples, un número alto puede indicar la presencia de valores atípicos (tal como veremos más adelante para la variable `SI.POV.URGP` que tiene un valor atípico cerca de `50`). Para leer un repaso completo de asimetría y curtosis, diríjanse a las Referencias (McNeese 2016) y (Handbook 2013).

- `iqr`: el rango intercuartil es el resultado de observar los percentiles `0.25` y `0.75`, e indica, en la misma unidad de la variable, el largo de dispersión del 50% de los valores. Cuanto mayor sea el valor, más dispersa es la variable.

- `range_98` y `range_80`: indican el rango en el que el se encuentra el `98%` de los valores. Quita el 1% inferior y superior (ergo, el número `98%`). Es bueno saber cuál es el rango de la variable sin valores atípicos potenciales. Por ejemplo, `pop_living_slums` va de `0` a `76.15`. Es **más robusto** que comparar los valores **mínimos** y **máximos**. `range_80` es igual que `range_98` pero sin el `10%` inferior y superior.

`iqr`, `range_98` y `range_80` están basados en percentiles, que cubriremos más adelante en este capítulo.

Importante: Todas las métricas se calculan quitando los valores `NA`. De lo contrario, la tabla estaría llena de NAs.

2.1.3.3.1 Consejos para utilizar `profiling_num`

La idea de la función `profiling_num` es proveer al científico de datos un conjunto completo de métricas para que pueda seleccionar las más relevantes. Esto se puede hacer fácilmente utilizando la función `select` del paquete `dplyr`.

Además, debemos configurar en `profiling_num` el parámetro `print_results = FALSE`. De esta forma, evitamos visualizar en la consola.

Por ejemplo, probemos con `mean`, `p_01`, `p_99` y `range_80`:

```
my_profiling_table=profiling_num(data_world_wide, print_results = FA

# Visualizar sólo las primeras tres filas
head(my_profiling_table, 3)
```

```
##                 variable mean p_01 p_99       range_80
## 1             gini_index   39 25.7   61 [27.58, 50.47]
## 2      pop_living_slums   46  6.8   93   [12.5, 75.2]
## 3 poverty_headcount_1.9   18  0.0   76  [0.08, 54.05]
```

Noten que `profiling_num` devuelve una tabla, por lo que podemos filtrar rápidamente los casos de acuerdo a las condiciones que definamos.

2.1.3.3.2 Analizar variables numéricas utilizando gráficos

Otra función de `funModeling` es `plot_num`, que toma un conjunto de datos y grafica la distribución de cada variable numérica excluyendo automáticamente las variables no numéricas:

```
plot_num(data_world_wide)
```

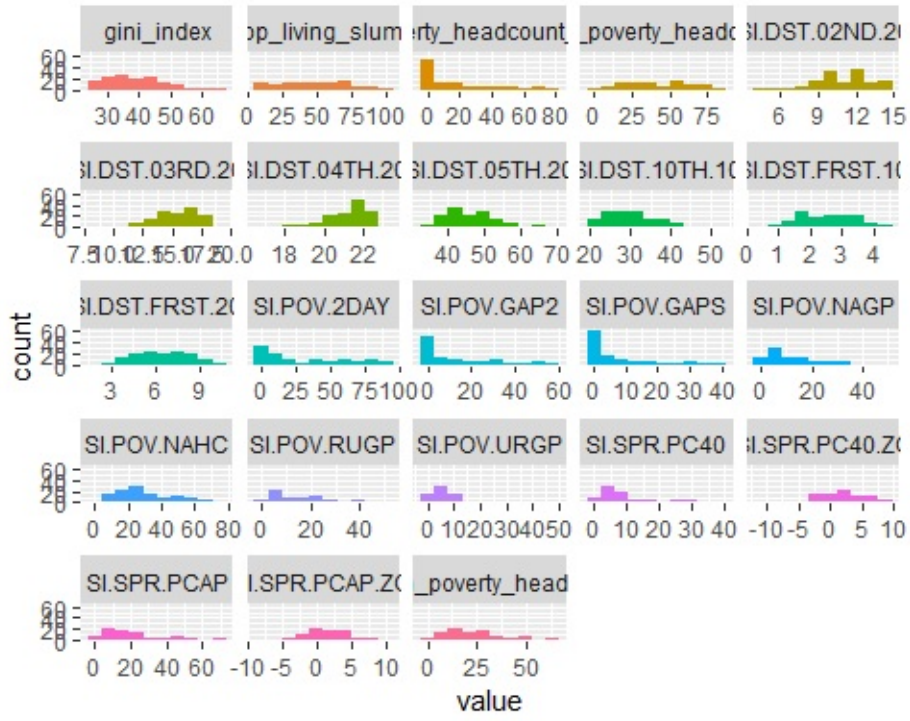


Figure 2.6: Analizando datos numéricos

Podemos ajustar la cantidad de barras del gráfico cambiando el parámetro `bins` (por defecto está configurado en 10). Por ejemplo: `plot_num(data_world_wide, bins = 20)`.

2.1.4 Reflexiones finales

Hasta aquí, han aparecido muchos números, *y aún más en el apéndice de percentiles*. Lo importante es que encuentren el enfoque adecuado para explorar sus datos. Puede estar relacionado con las métricas o con otros criterios.

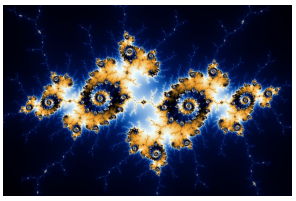
Las funciones `df_status`, `describe`, `freq`, `profiling_num` y `plot_num` pueden ejecutarse al principio de un proyecto de datos.

Con respecto al **comportamiento normal y anormal** de los datos, es importante estudiar ambos. Para describir un conjunto de datos en términos generales, deberíamos excluir los valores extremos: por ejemplo, con la variable `range_98`.

El promedio debería disminuir después de la exclusión.

Estos análisis son **univariados**; es decir, no tienen en cuenta otras variables (análisis **multivariado**). Esto estará incluido en este libro más adelante. Mientras tanto, para encontrar la correlación entre variables ingresadas (y de resultados), pueden dirigirse al capítulo de Correlación.

2.2 Correlación y relación



Fractal de Mandelbrot, donde el caos expresa su belleza; fuente de la imagen: Wikipedia.

2.2.1 ¿De qué se trata esto?

Este capítulo contiene aspectos metodológicos y prácticos de la medición de correlación en variables. Veremos que la palabra *correlación* puede traducirse en "**relación funcional**".

En metodología encontrarán el Cuarteto de Anscombe, un conjunto de cuatro gráficos con distribuciones espaciales diferentes, pero que comparten la misma

medida de correlación. Iremos un paso más allá recalculando su relación a través de una métrica más robusta (Coeficiente de Información Máxima o MIC, por sus siglas en inglés).

Mencionaremos **Teoría de la Información** varias veces; aunque por ahora la cubriremos a nivel matemático, está previsto hacerlo. Muchos algoritmos se basan en ella, incluso el aprendizaje profundo (deep learning).

Entender estos conceptos en dimensiones bajas (dos variables) y datos pequeños (un grupo de filas) nos permite entender mejor los datos de alta dimensión. No obstante, algunos casos reales son sólo datos *pequeños*.

Desde el punto de vista práctico, podrán replicar el análisis con sus propios datos, analizando numéricamente y exponiendo sus relaciones en gráficos sofisticados.

Empecemos por cargar todas las bibliotecas que necesitaremos.

```
# Cargar las bibliotecas necesarias
library(funModeling) # contiene datos de heart_disease
library(minerva) # contiene estadístico MIC
library(ggplot2)
library(dplyr)
library(reshape2)
library(gridExtra) # nos permite realizar
# dos gráficos en una fila
options(scipen=999) # desactiva la notación científica
```

2.2.2 Correlación lineal

Quizás la medida de correlación más estándar para las variables numéricas es la `R statistic` (o coeficiente de Pearson) que va desde `1` *correlación positiva* hasta `-1` *correlación negativa*. Un valor alrededor de `0` implica que no hay correlación.

Consideren el ejemplo siguiente, que calcula la medida R basada en una variable

de destino (por ejemplo, para realizar la ingeniería de factores). La función `correlation_table` obtiene la métrica R de todas las variables numéricas omitiendo las categóricas/nominales.

```
correlation_table(data=heart_disease, target="has_heart_disease")
```

```
##                  Variable has_heart_disease
## 1      has_heart_disease              1.00
## 2 heart_disease_severity              0.83
## 3      num_vessels_flour              0.46
## 4                oldpeak              0.42
## 5                  slope              0.34
## 6                    age              0.23
## 7 resting_blood_pressure              0.15
## 8      serum_cholestoral              0.08
## 9         max_heart_rate             -0.42
```

La variable `heart_disease_severity` es la variable -numérica- más importante; cuanto mayor sea su valor, mayores serán las probabilidades de padecer una enfermedad cardíaca (correlación positiva). Lo contrario de `max_heart_rate`, que tiene una correlación negativa.

Al elevar este número al cuadrado obtenemos la estadística `R-squared` (también conocida como R cuadrado o `R2`), que va desde `0` *no hay correlación* hasta `1` *alta correlación*.

El estadístico R está profundamente influenciado por los **valores atípicos** y las relaciones **no lineales**.

2.2.2.1 Correlación en el Cuarteto de Anscombe

Observen el **Cuarteto de Anscombe**, citando a Wikipedia:

> Fueron construidos en 1973 por el estadístico Francis Anscombe para demostrar tanto la importancia de graficar los datos antes de analizarlos como el efecto de los valores atípicos sobre las propiedades estadísticas.

1973 y sigue vigente hoy, es fantástico.

Estas cuatro relaciones son diferentes, pero todas tienen la misma R2: `0.816`.

El siguiente ejemplo calcula el R2 y grafica cada par.

```
# Leer los datos del Cuarteto de Anscombe
anscombe_data =
  read.delim(file="https://goo.gl/mVLz5L", header = T)

# Calcular la correlación (R cuadrado o R2) para
#cada par, los valores son el mismo: 0.86.
cor_1 = cor(anscombe_data$x1, anscombe_data$y1)
cor_2 = cor(anscombe_data$x2, anscombe_data$y2)
cor_3 = cor(anscombe_data$x3, anscombe_data$y3)
cor_4 = cor(anscombe_data$x4, anscombe_data$y4)

# Definir la función
plot_anscombe <- function(x, y, value, type)
{
  # 'anscombe_data' es una variable global, esto es
  # una mala práctica de programación ;)
  p=ggplot(anscombe_data, aes_string(x,y))  +
    geom_smooth(method='lm', fill=NA) +
    geom_point(aes(colour=factor(1),
                   fill = factor(1)),
               shape=21, size = 2
               ) +
    ylim(2, 13) +
    xlim(4, 19) +
    theme_minimal() +
    theme(legend.position="none") +
    annotate("text",
             x = 12,
             y =4.5,
             label =
               sprintf("%s: %s",
                       type,
                       round(value,2)
                       )
             )

  return(p)
}

# Graficar en una cuadrícula de 2x2
grid.arrange(plot_anscombe("x1", "y1", cor_1, "R2"),
```

```
plot_anscombe("x2", "y2", cor_2, "R2"),
plot_anscombe("x3", "y3", cor_3, "R2"),
plot_anscombe("x4", "y4", cor_4, "R2"),
ncol=2,
nrow=2)
```

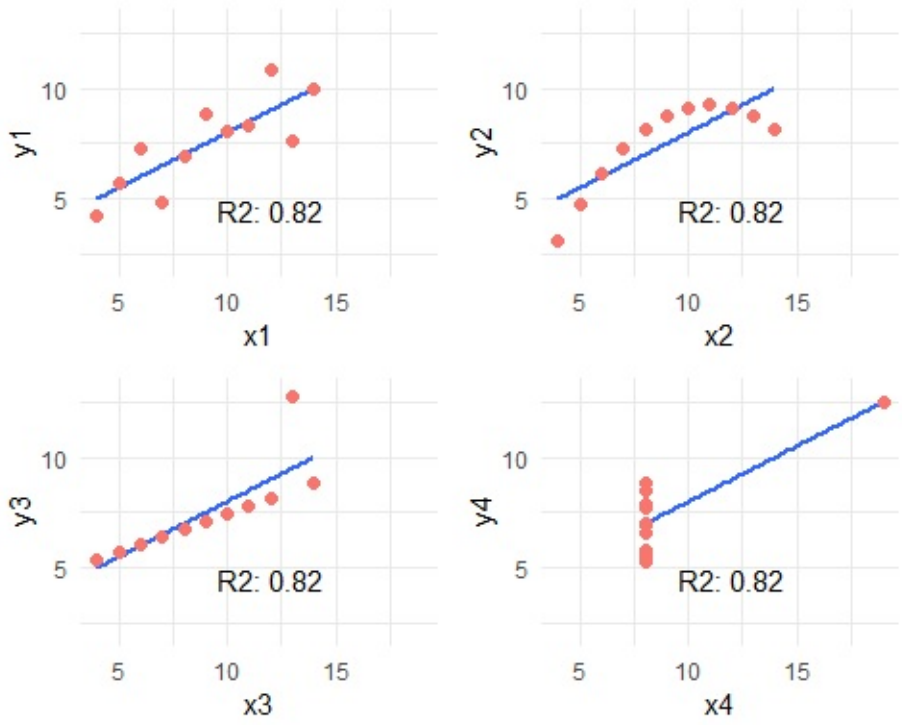


Figure 2.7: Conjunto de Anscombe

4 gráficos diferentes, con el mismo `mean` para cada variable `x` y `y` (9 y 7.501, respectivamente), y el mismo grado de correlación. Pueden comprobar todas las medidas ingresando `summary(anscombe_data)`.

Por esto es tan importante graficar relaciones cuando se analizan las correlaciones.

Volveremos sobre estos datos más tarde. ¡Se pueden mejorar! Primero, introduciremos algunos conceptos de la Teoría de la Información.

2.2.3 Correlación basada en la Teoría de la Información

Estas relaciones pueden medirse mejor con conceptos de la Teoría de la

Información. Uno de los muchos algoritmos que se utilizan para medir la correlación basada en esto es **MINE**, una sigla que significa Maximal Information-based nonparametric exploration, en inglés.

Pueden encontrar la implementación en R en el paquete minerva. También está disponible en otros lenguajes, como Python.

2.2.3.1 Ejemplo en R: Una relación perfecta

Grafiquemos una relación no lineal, basada directamente en una función (exponencial negativa), y visualicemos el valor MIC.

```
x=seq(0, 20, length.out=500)
df_exp=data.frame(x=x, y=dexp(x, rate=0.65))
ggplot(df_exp, aes(x=x, y=y)) + geom_line(color='steelblue') + theme
```

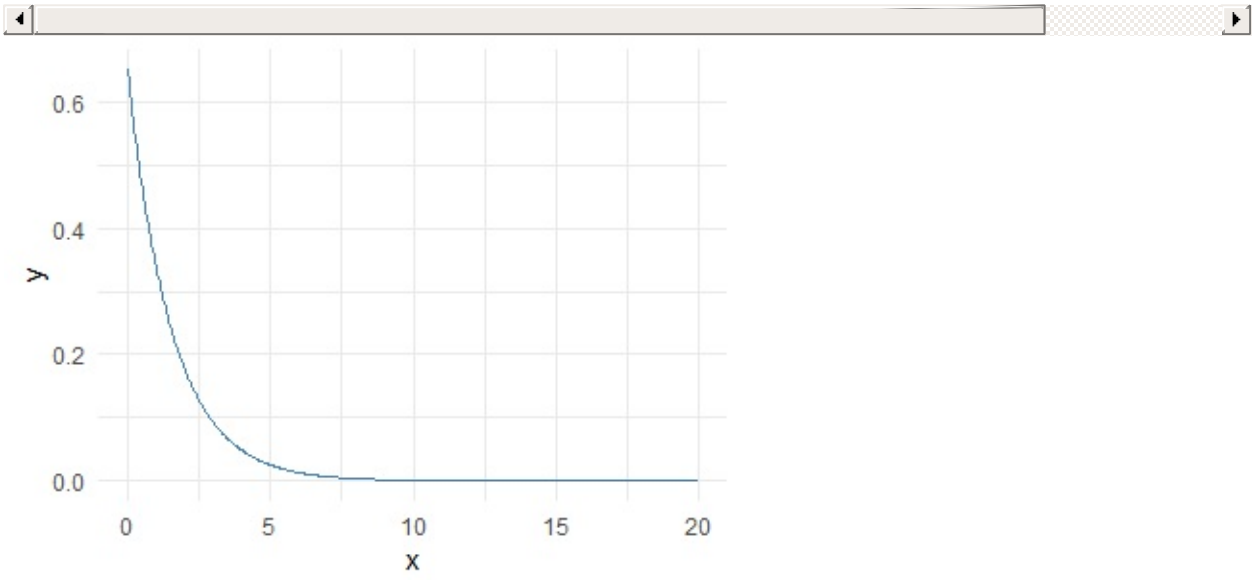


Figure 2.8: Una relación perfecta

```
# La posición [1,2] contiene la correlación de ambas variables, excl

# Calcular la correlación lineal
res_cor_R2=cor(df_exp)[1,2]^2
sprintf("R2: %s", round(res_cor_R2,2))
```

```
## [1] "R2: 0.39"
```

```
# Ahora computamos la métrica MIC
res_mine=mine(df_exp)
sprintf("MIC: %s", res_mine$MIC[1,2])
```

```
## [1] "MIC: 1"
```

Los valores de **MIC** van de 0 a 1. Cuando es 0, implica que no hay correlación y 1 implica la mayor correlación posible. La interpretación de los valores es la misma que para R cuadrado.

2.2.3.2 Análisis de los resultados

`MIC=1` indica que hay una correlación perfecta entre dos variables. Si estamos haciendo **ingeniería de factores** debemos incluir esta variable.

Más que una simple correlación, lo que dice el MIC es: "Hey, estas dos variables muestran una relación funcional".

En términos de *machine learning* (y simplificando demasiado): "la variable `y` es dependiente de la variable `x` y una función -no sabemos cuál- puede ser un modelo de la relación entre ellas."

Esto es delicado porque esa relación fue efectivamente creada a partir de una función, una exponencial.

Pero continuemos con otros ejemplos...

2.2.4 Agregar ruido

El ruido es una señal no deseada que se suma a la original. En machine learning, el ruido contribuye a que el modelo se confunda. En concreto: dos casos idénticos son ingresados -por ejemplo, clientes- y tienen resultados diferentes -

uno compra y el otro no.

Ahora vamos a añadir algo de ruido creando la variable `y_noise_1`.

```
df_exp$y_noise_1=jitter(df_exp$y, factor = 1000, amount = NULL)
ggplot(df_exp, aes(x=x, y=y_noise_1)) +
  geom_line(color='steelblue') + theme_minimal()
```

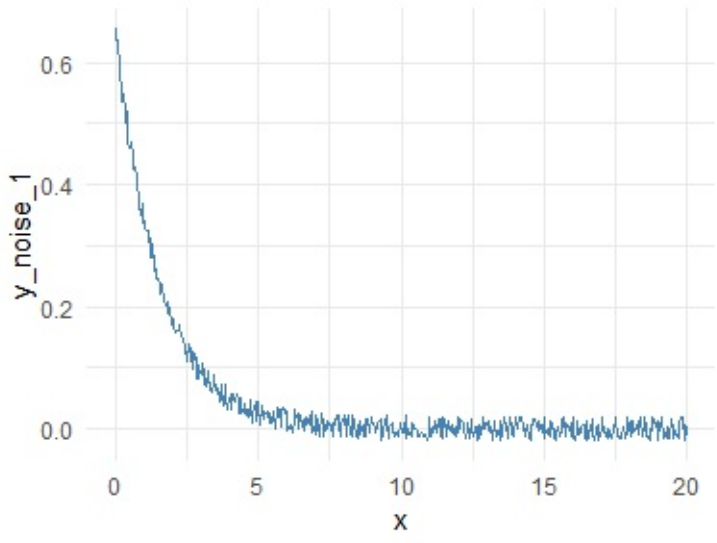


Figure 2.9: Agregando un poco de ruido

Calculando de nuevo la correlación y el MIC, visualizando en ambos casos la matriz completa, que muestra la métrica de correlación/MIC de cada variable de entrada con respecto a todas las demás, incluidas ellas mismas.

```
# Calcular R cuadrado
res_R2=cor(df_exp)^2
res_R2
```

```
##                x    y y_noise_1
## x           1.00 0.39      0.39
## y           0.39 1.00      0.99
## y_noise_1 0.39 0.99      1.00
```

```
# Calcular MINE
res_mine_2=mine(df_exp)
```

```
# Visualizar MIC
res_mine_2$MIC
```

```
##                x    y y_noise_1
## x          1.00 1.00      0.75
## y          1.00 1.00      0.74
## y_noise_1 0.75 0.74      1.00
```

Al agregar ruido a los datos, el valor de MIC disminuye de 1 a 0.7226365 (-27%), ¡y eso es genial!

R2 también disminuyó, pero sólo apenas, de 0.3899148 a 0.3866319 (-0.8%).

Conclusión: El MIC refleja una relación ruidosa mucho mejor que R2, y nos ayuda a encontrar asociaciones correlacionadas.

Sobre el último ejemplo: Generar datos basándonos en una función solamente sirve para fines educativos. Pero el concepto de ruido en variables es bastante común *casi* **todos los conjuntos de datos**, sin importar su fuente. No tienen que hacer nada para agregar ruido a las variables, ya está ahí. Los modelos de machine learning lidian con este ruido aproximándose a la forma *real* de los datos.

Es bastante útil usar la medición del MIC para tener una idea de la información presente en la relación entre dos variables.

2.2.5 Midiendo la no linealidad (MIC-R2)

La función `mine` devuelve varias métricas, sólo observamos **MIC**, pero dada la naturaleza del algoritmo (pueden referirse al paper original (Reshef et al. 2011)), puede computar indicadores mucho más interesantes. Pueden examinarlos todos en el objeto `res_mine_2`.

Uno de ellos es `MICR2`, que se utiliza como una medida de **no linealidad**. Se calcula haciendo: MIC - R2. Como R2 mide la linealidad, un alto `MICR2` indicaría una relación no lineal.

Podemos verificarlo calculando el MICR2 manualmente, las dos matrices a

continuación devuelven el mismo resultado:

```
# MIC r2: métrica de no linealidad
round(res_mine_2$MICR2, 3)
# Calcular MIC r2 manualmente
round(res_mine_2$MIC-res_R2, 3)
```

Las relaciones no lineales son más complejas para construir modelos, sobre todo utilizando un algoritmo lineal como árboles de decisión o regresión lineal.

Imaginen que debemos explicar la relación a otra persona, necesitaremos "más palabras" para hacerlo. Es más fácil decir: *"A aumenta mientras B disminuye y el coeficiente siempre es 3x"* (si A=1 entonces B=3, lineal).

En comparación con: *"A aumenta mientras B disminuye, pero A es casi 0 hasta que B alcanza el valor 10, entonces A aumenta a 300; y cuando B alcanza 15, A llega a 1000."*

```
# Crear un ejemplo con datos
df_example=data.frame(x=df_exp$x,
                      y_exp=df_exp$y,
                      y_linear=3*df_exp$x+2)

# Obtener métricas de mine
res_mine_3=mine(df_example)

# Generar etiquetas para visualizar los resultados
results_linear =
  sprintf("MIC: %s \n MIC-R2 (non-linearity): %s",
          res_mine_3$MIC[1,3],
          round(res_mine_3$MICR2[1,3],2)
          )

results_exp =
  sprintf("MIC: %s \n MIC-R2 (non-linearity): %s",
          res_mine_3$MIC[1,2],
          round(res_mine_3$MICR2[1,2],4)
          )

# Graficar los resultados
# Crear el gráfico de la variable exponencial
p_exp=ggplot(df_example, aes(x=x, y=y_exp)) +
```

```
  geom_line(color='steelblue') +
  annotate("text", x = 11, y =0.4, label = results_exp) +
  theme_minimal()

# Crear el gráfico de la variable lineal
p_linear=ggplot(df_example, aes(x=x, y=y_linear)) +
  geom_line(color='steelblue') +
  annotate("text", x = 8, y = 55,
           label = results_linear) +
  theme_minimal()

grid.arrange(p_exp,p_linear,ncol=2)
```

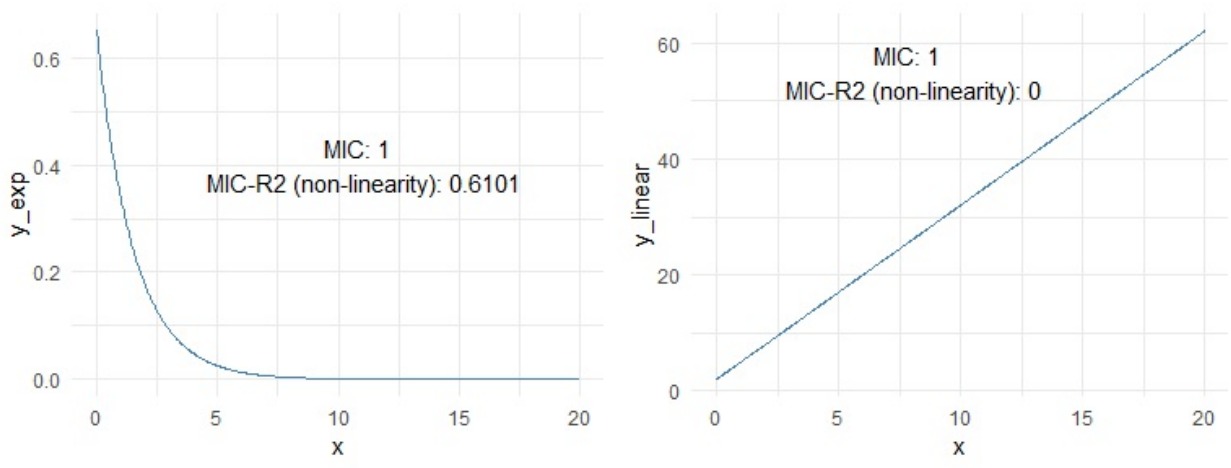


Figure 2.10: Comparando relaciones

Ambos gráficos muestran una correlación (o relación) perfecta, con MIC=1. Con respecto a la no linealidad, MICR2 se comporta como era esperado, en `y_exp`=0.6101, y en `y_linear`=0.

Este punto es importante dado que **MIC se comporta como R2 en relaciones lineales**, además se adapta bastante bien a relaciones **no lineales** como vimos antes, obteniendo una métrica de puntuación particular (`MICR2`) para analizar la relación.

2.2.6 Medir información en el Cuarteto de Anscombe

¿Recuerdan el ejemplo que revisamos al principio? Cada par del Cuarteto de Anscombe devuelve un **R2 de 0.86**. Pero basándonos en los gráficos estaba claro que no todos los pares exhiben ni una buena correlación ni una distribución similar de x y y.

Pero, ¿qué pasa si medimos la relación con una métrica basada en la Teoría de la Información? Sí, otra vez MIC.

```
# Calcular el MIC para cada par
mic_1=mine(anscombe_data$x1, anscombe_data$y1, alpha=0.8)$MIC
mic_2=mine(anscombe_data$x2, anscombe_data$y2, alpha=0.8)$MIC
mic_3=mine(anscombe_data$x3, anscombe_data$y3, alpha=0.8)$MIC
mic_4=mine(anscombe_data$x4, anscombe_data$y4, alpha=0.8)$MIC

# Graficar el MIC en una cuadrícula 2x2
grid.arrange(plot_anscombe("x1", "y1", mic_1, "MIC"), plot_anscombe(
```

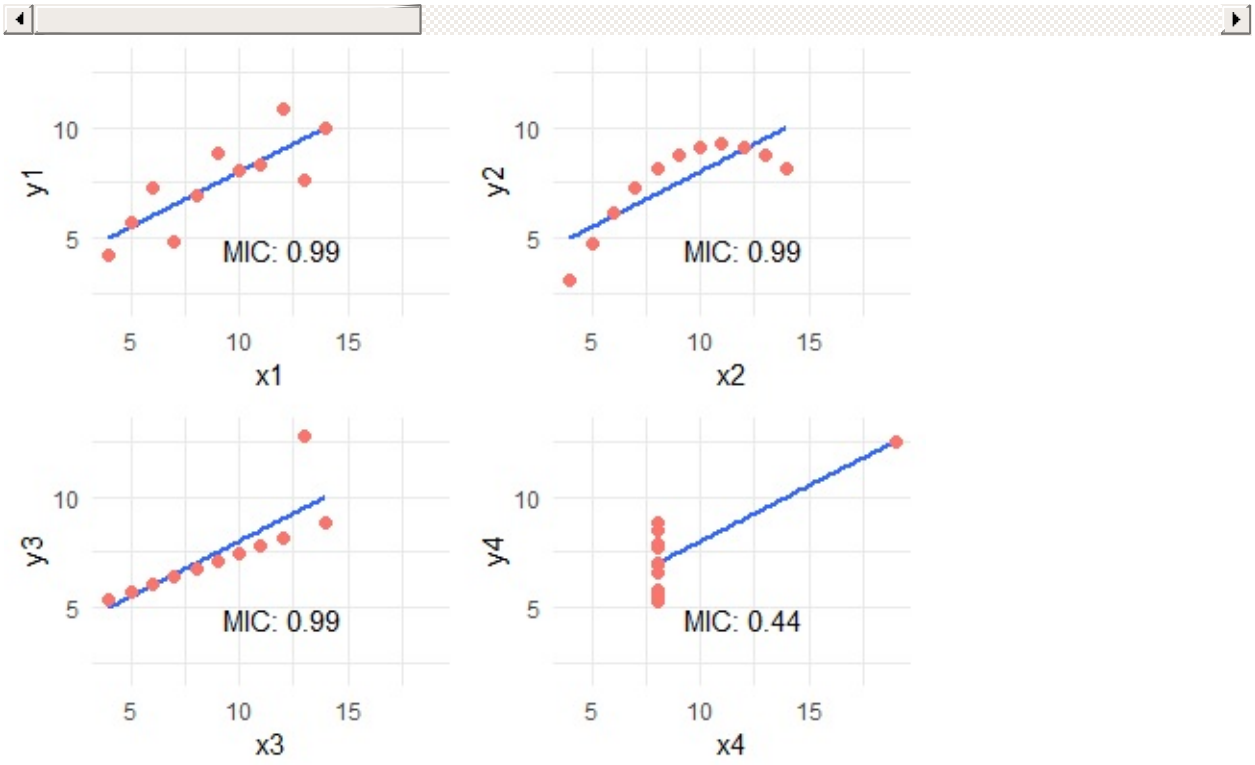


Figure 2.11: Estadístico MIC

Como habrán notado, aumentamos el valor de `alpha` a 0.8, esto es una buena práctica -de acuerdo a la documentación- cuando analizamos muestras pequeñas. El valor por defecto es 0.6 y el máximo es 1.

En este caso, el valor de MIC detectó la relación más espuria en el par `x4 - y4`. Probablemente debido a unos pocos casos por gráfico (11 filas) el MIC fue el mismo para todos los otros pares. Tener más casos se reflejará en diferentes valores de MIC.

Pero cuando se combina el MIC con **MIC-R2** (medida de no linealidad) aparecen nuevas perspectivas:

```
# Calcular el MIC para cada par, noten que el objeto "MIC-R2" lleva
mic_r2_1=mine(anscombe_data$x1, anscombe_data$y1, alpha = 0.8)$`MIC-
mic_r2_2=mine(anscombe_data$x2, anscombe_data$y2, alpha = 0.8)$`MIC-
mic_r2_3=mine(anscombe_data$x3, anscombe_data$y3, alpha = 0.8)$`MIC-
mic_r2_4=mine(anscombe_data$x4, anscombe_data$y4, alpha = 0.8)$`MIC-

# Ordenar por mic_r2
df_mic_r2=data.frame(pair=c(1,2,3,4), mic_r2=c(mic_r2_1,mic_r2_2,mic
df_mic_r2
```

```
##   pair mic_r2
## 1    2   0.33
## 2    3   0.33
## 3    1   0.33
## 4    4  -0.23
```

Al ordenarlos de manera decreciente según su **no linealidad** los resultados son consistentes con los gráficos: 2 > 3 > 1 > 4. Ocurre algo llamativo en el par 4, un número negativo. Esto se debe a que el MIC es inferior al R2. Una relación que merece ser graficada.

2.2.7 Midiendo la no-monotonicidad: medida MAS

MINE también nos puede ayudar a análizar numéricamente series temporales con respecto a su no-monotonicidad con **MAS** (puntaje de asimetría máxima).

Una serie monótona es aquella que nunca cambia su tendencia, siempre es creciente o decreciente. Encontrarán más información sobre esto en (Wikipedia 2017b).

El siguiente ejemplo simula dos series temporales, una no-monótona `y_1` y una monótona `y_2`.

```
# Crear datos de muestra (simulando una serie temporal)
time_x=sort(runif(n=1000, min=0, max=1))
y_1=4*(time_x-0.5)^2
y_2=4*(time_x-0.5)^3

# Calcular el MAS para ambas series
mas_y1=round(mine(time_x,y_1)$MAS,2)
mas_y2=mine(time_x,y_2)$MAS

# Unir todo
df_mono=data.frame(time_x=time_x, y_1=y_1, y_2=y_2)

# Graficar
label_p_y_1 =
  sprintf("MAS=%s (goes down \n and up => not-monotonic)",
          mas_y1)

p_y_1=ggplot(df_mono, aes(x=time_x, y=y_1)) +
  geom_line(color='steelblue') +
  theme_minimal()  +
  annotate("text", x = 0.45, y =0.75,
           label = label_p_y_1)

label_p_y_2=
  sprintf("MAS=%s (goes up => monotonic)", mas_y2)

p_y_2=ggplot(df_mono, aes(x=time_x, y=y_2)) +
  geom_line(color='steelblue') +
  theme_minimal() +
  annotate("text", x = 0.43, y =0.35,
           label = label_p_y_2)

grid.arrange(p_y_1,p_y_2,ncol=2)
```

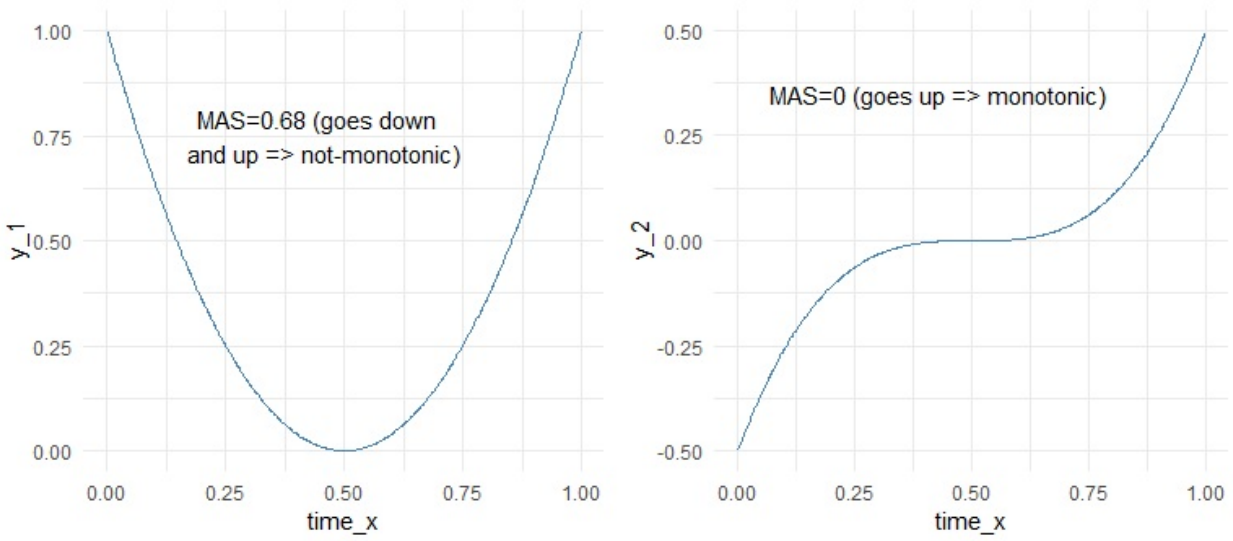


Figure 2.12: Monotonicidad en funciones

Desde otra perspectiva, el MAS también es útil para detectar relaciones periódicas. Ilustremos esto con un ejemplo:

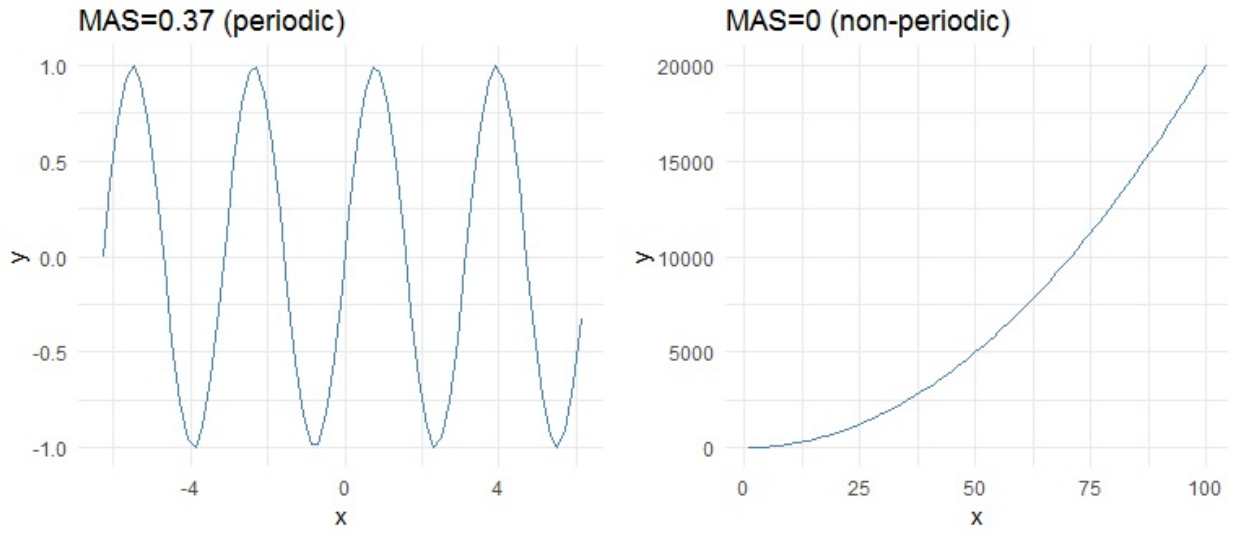


Figure 2.13: Periodicidad en funciones

2.2.7.1 Un ejemplo más real: Series Temporales

Consideren el siguiente caso que contiene tres series temporales: "y1", "y2" y

"y3". Se pueden analizar numéricamente en cuanto a su no-monotonicidad o a la tendencia general de crecimiento.

```
# Leer los datos
df_time_series =
  read.delim(file="https://goo.gl/QDUjfd")

# Convertirlos a formato largo para poder graficarlos
df_time_series_long=melt(df_time_series, id="time")

# Graficar
plot_time_series =
  ggplot(data=df_time_series_long,
         aes(x=time, y=value, colour=variable)) +
  geom_line() +
  theme_minimal() +
  scale_color_brewer(palette="Set2")

plot_time_series
```

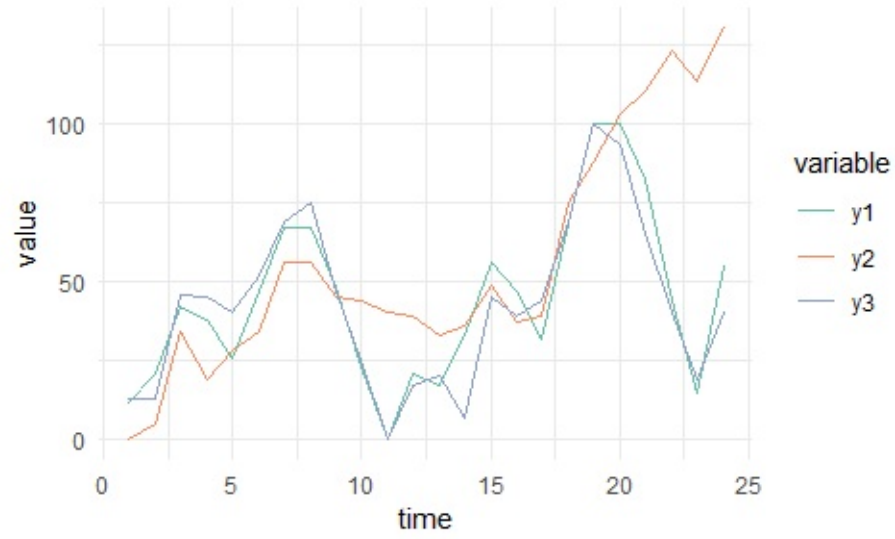


Figure 2.14: Ejemplo de series temporales

```
# Calcular y visualizar los valores de MAS para datos de series temp
mine_ts=mine(df_time_series)
mine_ts$MAS
```

```
##       time     y1     y2     y3
```

```
## time 0.00 0.120 0.105 0.191
## y1   0.12 0.000 0.068 0.081
## y2   0.11 0.068 0.000 0.057
## y3   0.19 0.081 0.057 0.000
```

Necesitamos mirar la columna de "tiempo", así tendremos el valor de MAS de cada serie con respecto al tiempo. `y2` es la serie más monótona (y menos periódica), y podemos confirmarlo mirándola. Parece que siempre es ascendente.

Resumen de MAS:

- MAS ~ 0 indica una función monótona o no periódica ("siempre" ascendente o descendente)
- MAS ~ 1 indica una función no-monótona o periódica

2.2.8 Correlación entre series temporales

La métrica MIC también puede medir la **correlación en series temporales**, no es una herramienta de uso general pero puede ser útil para comparar diferentes series rápidamente.

Esta sección se basa en los mismos datos que usamos en el ejemplo de MAS.

```
# Visualizar otra vez las 3 series temporales
plot_time_series
```

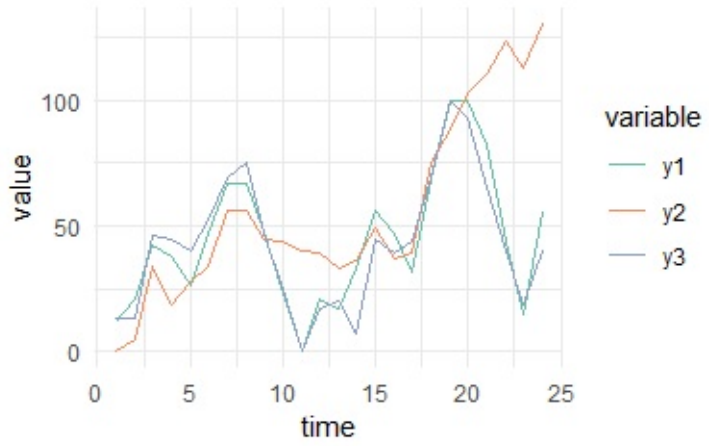


Figure 2.15: Ejemplo de series temporales

```
# Visualizar los valores de MIC
mine_ts$MIC
```

```
##      time   y1   y2   y3
## time 1.00 0.38 0.69 0.34
## y1   0.38 1.00 0.62 0.71
## y2   0.69 0.62 1.00 0.52
## y3   0.34 0.71 0.52 1.00
```

Ahora tenemos que mirar la columna `y1`. De acuerdo con la medición del MIC, podemos confirmar lo mismo que se muestra en el último gráfico:

`y1` es más parecido a `y3` (MIC=0.709) que a `y2` (MIC=0.61).

2.2.8.1 Yendo más allá: Deformación dinámica del tiempo

El MIC no servirá en escenarios más complejos que tengan series temporales que varíen en velocidad, utilizaremos la técnica de [deformación dinámica del tiempo] (https://en.wikipedia.org/wiki/Dynamic_time_warping) (**DTW**, en inglés).

Usemos una imagen para acercarnos al concepto visualmente:

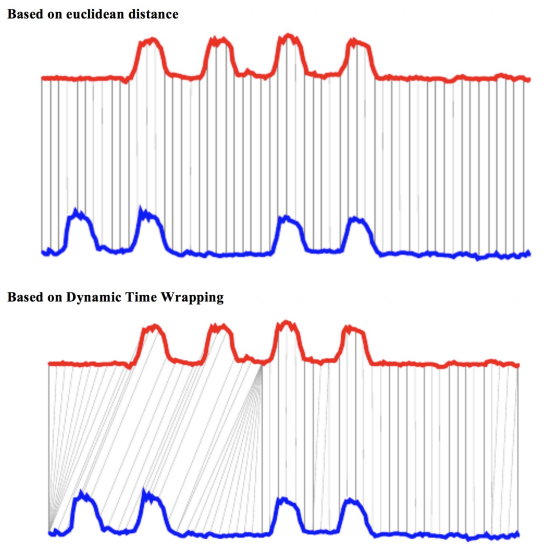


Figure 2.16: Deformación dinámica del tiempo

Fuente de la imagen: *Deformación dinámica del tiempo Convirtiendo imágenes en series temporales para data mining* (Izbicki 2011).

La última imagen muestra dos enfoques diferentes para comparar series temporales, y el **euclídeo** es más similar a la medición del MIC. Mientras que la Deformación dinámica del tiempo puede rastrear las similitudes que ocurren en diferentes momentos.

Una linda implementación en **R**: dtw package.

Encontrar correlaciones entre series temporales es otra forma de **agrupar series temporales**.

2.2.9 Correlación en variables categóricas

MINE -y muchos otros algoritmos- sólo funcionan con datos numéricos. Tenemos que hacer un truco de **preparación de datos**, convirtiendo cada variable categórica en un flag (o variable dummy).

Si la variable categórica original tiene 30 valores posibles, como resultado tendremos 30 nuevas columnas con valor `0` o `1`, donde `1` representa la presencia de esa categoría en la fila.

Si usamos el paquete `caret` de R, esta conversión sólo requiere dos líneas de código:

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.5.3
```

```
# Seleccionar sólo algunas variables
heart_disease_2 =
  select(heart_disease, max_heart_rate, oldpeak,
         thal, chest_pain,exer_angina, has_heart_disease)

# Esta conversión de categórica a numérica es meramente para
# tener un gráfico más limpio
heart_disease_2$has_heart_disease=
  ifelse(heart_disease_2$has_heart_disease=="yes", 1, 0)

# Convierte todas las variables categóricas (factor y
# carácter para R) en variables numéricas
# saltando las originales para que los datos estén
# listos para usar
dmy = dummyVars(" ~ .", data = heart_disease_2)

heart_disease_3 =
  data.frame(predict(dmy, newdata = heart_disease_2))

# Importante: Si ven este mensaje
# `Error: Missing values present in input variable 'x'.
# Consideren usar = 'pairwise.complete.obs'.`
# Es porque los datos tienen valores faltantes.
# Por favor no omitan valores NA sin analizar el
# impacto antes, en este caso no es importante.
heart_disease_4=na.omit(heart_disease_3)

# ¡Computen el mic!
mine_res_hd=mine(heart_disease_4)
```

Visualizando una muestra...

```
mine_res_hd$MIC[1:5,1:5]
```

```
##                max_heart_rate oldpeak thal.3 thal.6 thal.7
```

```
## max_heart_rate           1.00    0.24  0.244  0.120  0.184
## oldpeak                  0.24    1.00  0.175  0.111  0.157
## thal.3                   0.24    0.18  0.992  0.073  0.710
## thal.6                   0.12    0.11  0.073  0.327  0.044
## thal.7                   0.18    0.16  0.710  0.044  0.964
```

Donde la columna `thal.3` toma un valor de 1 cuando `thal=3`.

2.2.9.1 ¡Visualizamos unos sofisticados gráficos!

Utilizaremos el paquete `corrplot` de R que puede graficar un objeto `cor` (matriz de correlación clásica), o cualquier otra matriz. Graficaremos la matriz **MIC** en este caso, pero también se puede usar cualquier otra, por ejemplo, **MAS** o cualquier otra métrica que devuelva una matriz cuadrada de correlaciones.

Los dos gráficos se basan en los mismos datos pero muestran la correlación de distintas formas.

```
# Biblioteca para graficar esta matriz
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 3.5.3
```

```
## corrplot 0.84 loaded
```

```
# Para usar la paleta de color brewer.pal
library(RColorBrewer)
```

```
## Warning: package 'RColorBrewer' was built under R version 3.5.2
```

```
# Truco para visualizar el valor máximo de la escala
# excluyendo la diagonal (var. con respecto a sí misma)
diag(mine_res_hd$MIC)=0

# Gráfico de correlación con círculos.
corrplot(mine_res_hd$MIC,
         method="circle",
         col=brewer.pal(n=10, name="PuOr"),
         # Mostrar sólo la diagonal superior
```

```
          type="lower",
          #color, tamaño y rotación de las etiquetas
          tl.col="red",
          tl.cex = 0.9,
          tl.srt=90,
          # no visualizar la diagonal,
          # (var con respecto a sí misma)
          diag=FALSE,
          # aceptar cualquier matriz, mic en este caso
          #(no es un elemento de correlación)
          is.corr = F

)
```

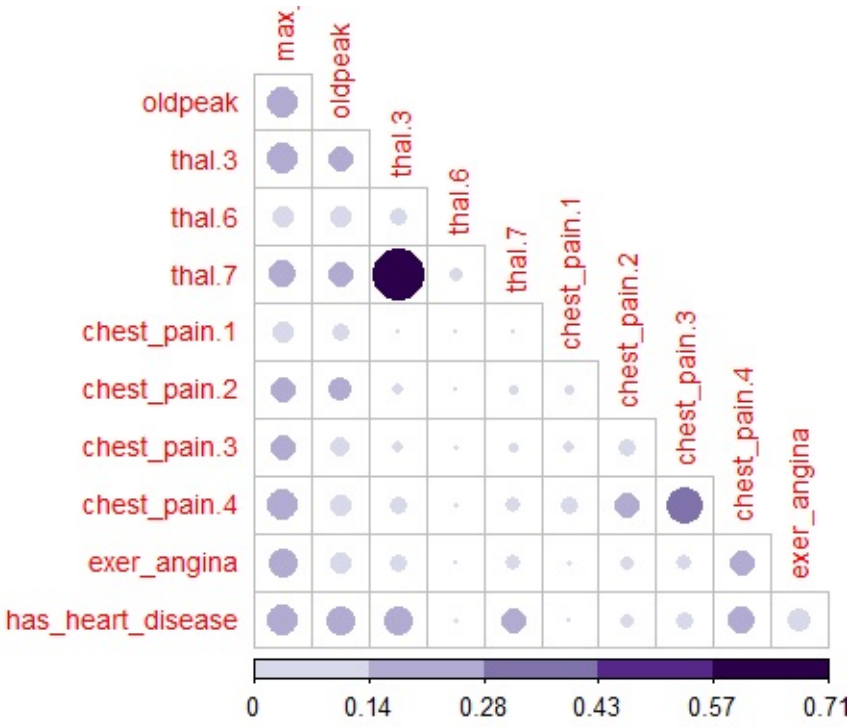


Figure 2.17: Gráfico de correlación

```
# Gráfico de correlación con color y correlación MIC
corrplot(mine_res_hd$MIC,
          method="color",
          type="lower",
          number.cex=0.7,
          # Agregar coeficiente de correlación
          addCoef.col = "black",
          tl.col="red",
          tl.srt=90,
```

```
        tl.cex = 0.9,
        diag=FALSE,
        is.corr = F
)
```

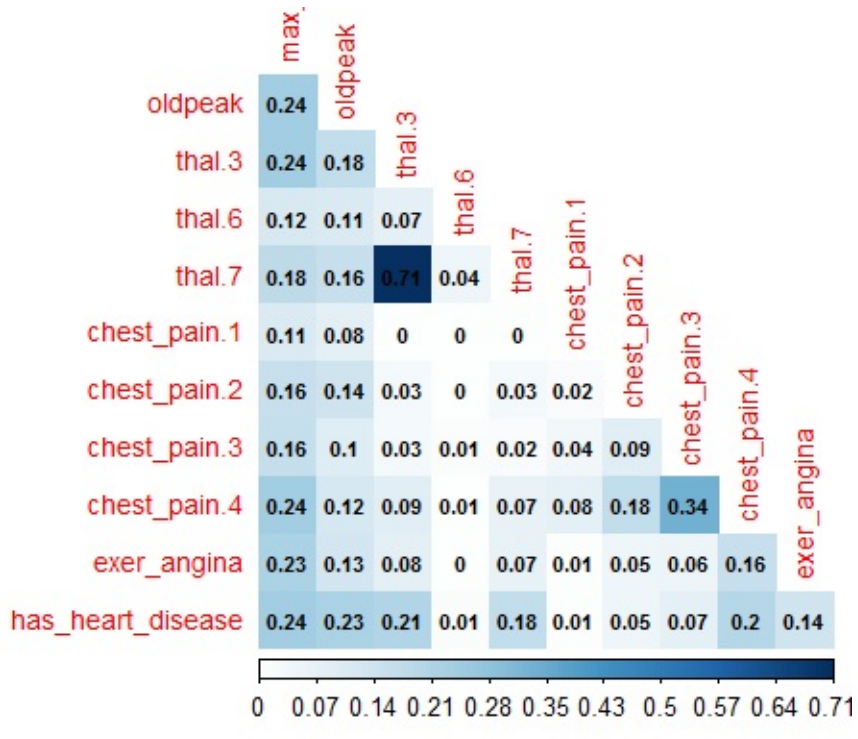


Figure 2.18: Gráfico de correlación

Simplemente cambien el primer parámetro `-mine_res_hd$MIC-` a la matriz que quieran y reutilicen el código con sus datos.

2.2.9.2 Un comentario sobre este tipo de gráficos

Sólo son útiles cuando no hay un gran número de variables. O si primero hacen una selección de variables, teniendo en cuenta que todas deben ser numéricas.

Si hay alguna variable categórica en la selección, pueden convertirla primero en numérica e inspeccionar la relación entre las variables, para ver cómo ciertos valores de las variables categóricas están más relacionados con ciertos resultados, como en este caso.

2.2.9.3 ¿Qué tal si sacamos algunas conclusiones de los gráficos?

Como la variable a predecir es `has_heart_disease`, aparece algo interesante: tener una enfermedad cardíaca está más correlacionado con `thal=3` que con el valor `thal=6`.

El mismo análisis aplica para la variable `chest_pain', un valor de 4 es más peligroso que un valor de 1.

Y podemos comprobarlo con otro gráfico:

```
cross_plot(heart_disease, input = "chest_pain", target = "has_heart_
```

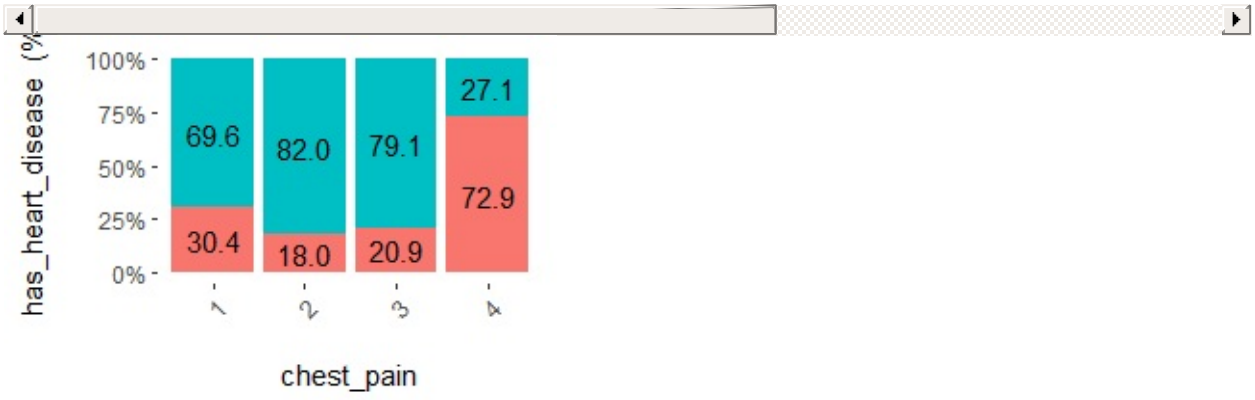


Figure 2.19: Análisis visual utilizando cross-plot

La probabilidad de tener una enfermedad cardíaca es del 72,9% si el paciente tiene `dolor en el pecho=4`. Más de 2 veces más probable que si tiene `dolor_en_el_pecho=1` (72.9 vs 30.4%).

Algunas reflexiones...

Los datos son los mismos, pero el enfoque para explorarlos es diferente. Lo mismo ocurre cuando estamos creando un modelo predictivo, los datos ingresados en el espacio *N-dimensional* pueden ser abordados mediante diferentes modelos, como: Máquinas de vectores soporte (support vector

machine o SVM, en inglés), Bosques aleatorios (random forests), etc.

Como un fotógrafo que dispara desde diferentes ángulos o desde diferentes cámaras. El objeto es siempre el mismo, pero la perspectiva brinda información diferente.

La combinación de tablas en crudo y diferentes gráficos nos da una perspectiva de objeto más real y complementaria.

2.2.10 Análisis de correlación basado en la Teoría de la Información

Basándose en la medición del MIC, la función mine puede recibir el índice de la columna a predecir (o para obtener todas las correlaciones contra una sola variable).

```
# Obtener el índice de la variable a
# predecir: has_heart_disease
target="has_heart_disease"
index_target=grep(target, colnames(heart_disease_4))

# master toma el número de la columna índice para
# calcular todas las correlaciones
mic_predictive=mine(heart_disease_4,
                    master = index_target)$MIC

# Crear el data frame que contiene los resultados,
# ordenándolos por correlación decreciente y excluyendo
# la correlación del objetivo con respecto a sí mismo
df_predictive =
  data.frame(variable=rownames(mic_predictive),
                          mic=mic_predictive[,1],
                          stringsAsFactors = F) %>%
  arrange(-mic) %>%
  filter(variable!=target)

# Crear un colorido gráfico que muestre la
# importancia de las variables basándonos en
# la medición del MIC
ggplot(df_predictive,
       aes(x=reorder(variable, mic),y=mic, fill=variable)
```

```
    ) +
geom_bar(stat='identity') +
coord_flip() +
theme_bw() +
xlab("") +
ylab("Variable Importance (based on MIC)") +
guides(fill=FALSE)
```

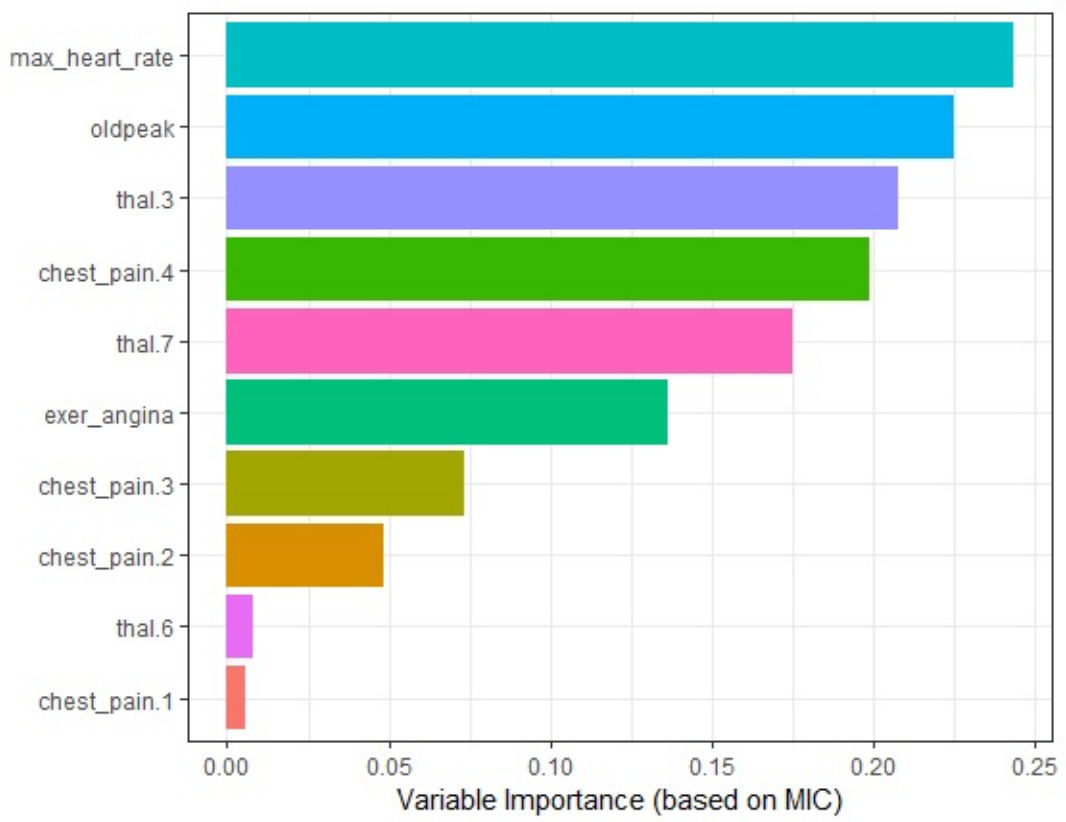


Figure 2.20: Correlación utilizando la Teoría de la Información

Aunque es recomendable ejecutar las correlaciones entre todas las variables para excluir factores ingresados correlacionados.

2.2.10.1 Consejos prácticos al utilizar `mine`

Si lleva demasiado tiempo, consideren tomar una muestra. Si hay muy pocos datos, consideren utilizar un valor más alto en el parámetro `alpha`, 0.6 es el valor por defecto. También se puede ejecutar en paralelo, configurando el parámetro

`n.cores=3` si tienen 4 cores. Una buena práctica en general cuando se ejecutan procesos paralelos, el core extra será utilizado por el sistema operativo.

2.2.11 ¿Sólo MINE cubre esto?

No. Sólo usamos la suite de MINE, pero hay otros algoritmos relacionados con información mutua. En R algunos de los paquetes son: entropy y infotheo.

El paquete `funModeling` (desde la versión 1.6.6) introduce la función `var_rank_info`, que calcula varias métricas de la Teoría de la Información, como vimos en la sección Análisis de correlación basado en la Teoría de la Información.

En **Python** se puede calcular información mutua mediante scikit-learn, aquí un ejemplo.

El concepto trasciende la herramienta.

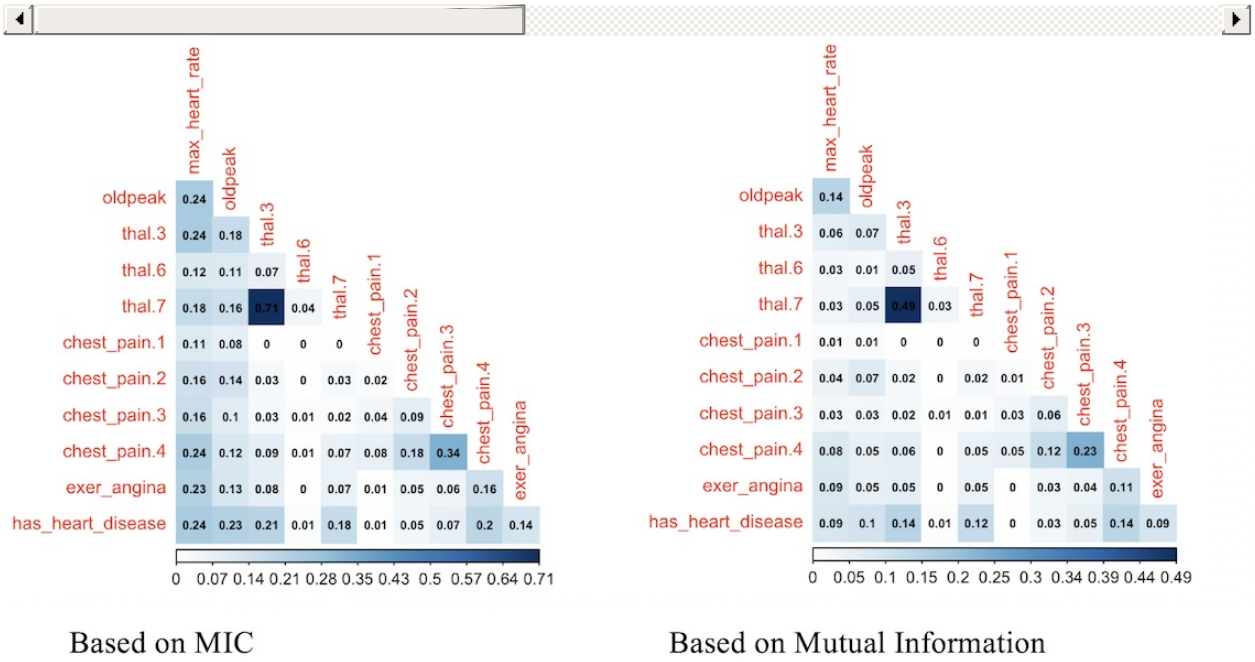
2.2.11.1 Otro ejemplo de correlación (información mutua)

Esta vez utilizaremos el paquete `infotheo`. Primero tenemos que hacer un un paso de **preparación de datos**, aplicando una función `discretize` (o de binning) que está incluida en el paquete. Convierte todas las variables numéricas a categóricas basándose en criterios de igual frecuencia.

El siguiente código creará la matriz de correlación como hemos visto antes, pero basándose en el índice de información mutua:

```
library(infotheo)
# Discretizar cada variable
heart_disease_4_disc=discretize(heart_disease_4)

# Calcular la "correlación" basándonos en información mutua
heart_info=mutinformation(heart_disease_4_disc, method= "emp")

# Truco para visualizar el valor máximo de la escala
```

```
# excluyendo la diagonal (variable con respecto a sí misma)
diag(heart_info)=0

# Gráfico de correlación con color y correlación con información mut
corrplot(heart_info, method="color",type="lower", number.cex=0.6,add
```



Based on MIC

Based on Mutual Information

Figure 2.21: Comparando la importancia de las variables

El puntaje de correlación basado en información mutua ordena las relaciones de una forma bastante similar al MIC, ¿no es cierto?

2.2.12 Medidas de información: Una perspectiva general

Más allá de la correlación, MIC y otras medidas de información identifican si hay una *relación funcional*.

Un alto MIC indica que la relación entre las dos variables puede explicarse por una función. Es nuestro trabajo encontrar esa función o modelo predictivo.

Este análisis se extiende a n-variables, este libro introduce otro algoritmo en el capítulo de Selección de las mejores variables.

Algunos modelos predictivos funcionan mejor que otros, pero si la relación es

absolutamente ruidosa, sin importar cuán avanzado sea el algoritmo, los resultados serán malos.

Hay mucho más sobre la **Teoría de la Información** más adelante. Por ahora, pueden investigar estas lecturas didácticas:

- Video introductorio de 7 minutos https://www.youtube.com/watch?v=2s3aJfRr9gE
- http://alex.smola.org/teaching/cmu2013-10-701x/slides/R8-information_theory.pdf
- http://www.scholarpedia.org/article/Mutual_information

2.2.13 Conclusiones

El Cuarteto de Anscombe nos enseñó la buena práctica de combinar la *estadística cruda* con un gráfico.

Pudimos ver cómo el **ruido** puede afectar la relación entre dos variables, y este fenómeno siempre aparece en los datos. El ruido en los datos confunde al modelo predictivo.

El ruido está relacionado con el error, y puede ser estudiado con medidas basadas en la Teoría de la Información, tales como **información mutua** y **MIC**, que van un paso más allá de la típica R cuadrado. Existe un estudio clínico que utiliza MINE como selector de características en(Caban et al. 2012).

Estos métodos son aplicables en **ingeniería de factores** como un método que no se basa en un modelo predictivo para clasificar las variables más importantes. También se aplica a las series temporales agrupadas.

Siguiente capítulo recomendado: Selección de las mejores variables

Libro Vivo de Ciencia de Datos

3 Literature

Here is a review of existing methods.

4 Preparación de datos

4.1 Manejando tipos de datos

4.1.1 ¿De qué se trata esto?

Una de las primeras cosas que hay que hacer cuando empezamos un proyecto de datos es asignar el tipo de datos correcto para cada variable. Aunque esto parece una tarea sencilla, algunos algoritmos funcionan con ciertos tipos de datos. Aquí trataremos de cubrir estas conversiones mientras explicamos con ejemplos las implicaciones en cada caso.

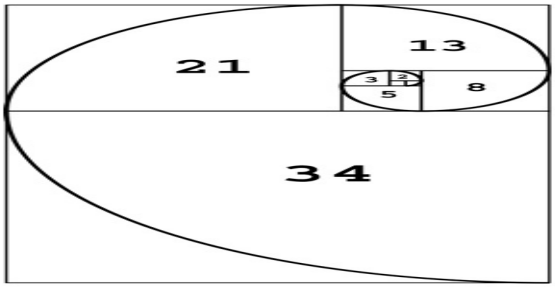


Figure 4.1: Espiral de Fibonacci

La sucesión de Fibonacci. Una secuencia de números presente en la naturaleza y en los cuerpos humanos.

¿Qué vamos a repasar en este capítulo?

- Detección del tipo de datos correcto
- Cómo convertir de categórico a numérico
- Cómo convertir de numérico a categórico (métodos de discretización)
- Aspectos teóricos y prácticos (ejemplos en R)
- Cómo observa las variables numéricas un modelo predictivo

4.1.2 El universo de los tipos de datos

Hay dos tipos principales de datos, **numérico** y **categórico**. Otros nombres para categóricos son **string** y **nominal**.

Un subconjunto de categórico es el ordinal o, como se lo llama en R, un factor **ordenado**. Al menos en R, este tipo sólo es relevante cuando se grafican categorías en un orden determinado. Un ejemplo en R:

```
# Crear un factor ordinal u ordenado
var_factor=factor(c("3_high", "2_mid", "1_low"))
var_ordered=factor(var_factor, ordered = T)
var_ordered
```

```
## [1] 3_high 2_mid  1_low
## Levels: 1_low < 2_mid < 3_high
```

No presten demasiada atención a este tipo de datos, ya que los numéricos y categóricos son los más necesarios.

4.1.2.1 Variable binaria: ¿numérica o categórica?

Este libro sugiere utilizar las variables binarias como numéricas cuando `0` es `FALSE` y `1` es `TRUE`. Simplifica el análisis matemático de los datos.

4.1.3 Tipos de datos por algoritmo

Algunos algoritmos funcionan de la siguiente manera:

- Sólo con datos categóricos
- Sólo con datos numéricos
- Con ambos tipos

Además, no todos los modelos predictivos pueden manejar **valores faltantes**.

El **Data Science Live Book** busca cubrir todas estas situaciones.

4.1.4 Convirtiendo variables categóricas en numéricas

Usar el paquete `caret` en R es una tarea sencilla que convierte cada variable categórica en una variable **flag**, también conocida como variable *dummy*.

Si la variable categórica original tiene treinta valores posibles, entonces resultará en 30 nuevas columnas que contengan el valor `0` o `1`, donde `1` representa la presencia de esa categoría en la fila.

Si usamos el paquete caret de R, entonces para esta conversión sólo se necesitan dos líneas de código:

```
library(caret) # contiene la función dummyVars
library(dplyr) # librería de data munging
library(funModeling) # df_status function

# Comprobar variables categóricas
status=df_status(heart_disease, print_results = F)
filter(status,  type %in% c("factor", "character")) %>% select(varia
```

```
##              variable
## 1              gender
## 2          chest_pain
## 3 fasting_blood_sugar
## 4     resting_electro
## 5                thal
## 6        exter_angina
## 7   has_heart_disease
```

```
# Convierte todas las variables categóricas (factor y carácter) en v
# Se salta la variable original, por lo que no es necesario eliminar
dmy = dummyVars(" ~ .", data = heart_disease)
heart_disease_2 = data.frame(predict(dmy, newdata = heart_disease))

# Comprobar el nuevo conjuntos de datos numéricos:
colnames(heart_disease_2)
```

```
##  [1] "age"                    "gender.female"
##  [3] "gender.male"            "chest_pain.1"
##  [5] "chest_pain.2"           "chest_pain.3"
##  [7] "chest_pain.4"           "resting_blood_pressure"
##  [9] "serum_cholestoral"      "fasting_blood_sugar.0"
## [11] "fasting_blood_sugar.1"  "resting_electro.0"
## [13] "resting_electro.1"      "resting_electro.2"
## [15] "max_heart_rate"         "exer_angina"
## [17] "oldpeak"                "slope"
## [19] "num_vessels_flour"      "thal.3"
## [21] "thal.6"                 "thal.7"
## [23] "heart_disease_severity" "exter_angina.0"
## [25] "exter_angina.1"         "has_heart_disease.no"
## [27] "has_heart_disease.yes"
```

Los datos originales `heart_disease` han sido convertidos a `heart_disease_2` que no tiene variables categóricas, sólo numéricas y dummy. Observe que cada nueva variable tiene un *punto* seguido por el *valor*.

Si comprobamos el antes y el después para el séptimo paciente (fila) en la variable `chest_pain` que puede tomar los valores `1`, `2`, `3` o `4`, entonces

```
# Antes
as.numeric(heart_disease[7, "chest_pain"])
```

```
## [1] 4
```

```
# Después
heart_disease_2[7, c("chest_pain.1", "chest_pain.2", "chest_pain.3",
```

```
##   chest_pain.1 chest_pain.2 chest_pain.3 chest_pain.4
## 7            0            0            0            1
```

Habiendo conservado y transformado sólo variables numéricas excluyendo las nominales, los datos `heart_disease_2` están listos para ser utilizados.

Hay más información sobre `dummyVars` en: http://amunategui.github.io/dummyVar-Walkthrough/

4.1.5 ¿Es categórica o numérica? Piénsenlo.

Consideren la variable `chest_pain`, que puede tomar los valores `1`, `2`, `3`, o `4`. ¿Es esta variable categórica o numérica?

Si los valores están ordenados, entonces se la puede considerar tan numérica como si exhibiera un **orden**, es decir, 1 es menos de 2, 2 es menos de 3, y 3 es menos de 4.

Si creamos un modelo de árbol de decisión, entonces podemos encontrar reglas como: "Si el dolor de pecho es > 2.5, entonces...". ¿Tiene sentido? El algoritmo divide la variable por un valor que no está presente (`2.5`); sin embargo, la interpretación que hacemos es "si `dolor de pecho` es igual o superior a 3, entonces...".

4.1.6 Pensar como un algoritmo

Considere dos variables numéricas de entrada y una variable binaria de destino. El algoritmo *verá* ambas variables de entrada como puntos en un rectángulo, considerando que hay valores infinitos entre cada número.

Por ejemplo, una **Máquina de Soporte Vectorial** (SVM) creará *varios* vectores para separar la clase de la variable de destino. Encontrará regiones basadas en estos vectores. ¿Cómo sería posible encontrar estas regiones basándose en variables categóricas? No es posible y es por eso que el SVM sólo funciona con variables numéricas como en las redes neuronales artificiales.

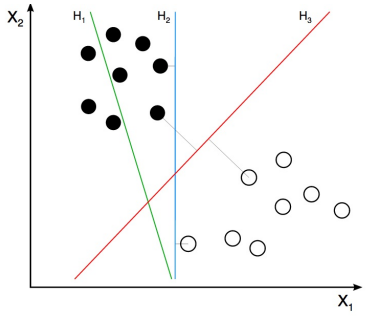


Figure 4.2: Máquina de vectores soporte

Image credit: ZackWeinberg

La última imagen muestra tres líneas, que representan tres límites de decisión o regiones diferentes.

Para una rápida introducción a este concepto de SVM, por favor vean este corto video: Demo SVM.

Sin embargo, si el modelo está basado en árboles, como *decision trees*, *random forest o gradient boosting machine*, entonces manejan ambos tipos porque su espacio de búsqueda puede ser regiones (igual que SVM) y categorías. Como la regla `"si postal_code es AX441AG y tiene más de 55 años, entonces..."`.

Volviendo al ejemplo de la enfermedad cardíaca, la variable `chest_pain` exhibe orden. Debemos aprovechar esto porque si lo convertimos en una variable categórica, entonces **estamos perdiendo información** y este es un punto importante a la hora de manejar los tipos de datos.

4.1.6.1 ¿Es la solución tratar a todas las variables como categóricas?

No.... Una variable numérica contiene más información que una nominal debido a su orden. En las variables categóricas, los valores no se pueden comparar. Digamos que no es posible hacer una regla como `Si el código postal es superior a "AX2004-P"`.

Los valores de una variable nominal pueden ser comparados si tenemos otra variable para usar como referencia (normalmente un resultado a predecir).

Por ejemplo, el código postal "AX2004-P" es *más alto* que "MA3942-H" porque hay más personas interesadas en asistir a clases de fotografía.

Además, **la alta cardinalidad** es un problema en las variables categóricas, por ejemplo, una variable `postal code` que contiene cientos de valores diferentes. Este libro ha tratado este tema en ambos capítulos: el manejo de variables de alta categorización para estadísticas descriptivas y cuando hacemos modelado predictivo.

De todos modos, pueden hacer *la prueba gratis* de convertir todas las variables en categóricas y ver qué pasa. Comparen los resultados con las variables numéricas. Recuerden usar alguna buena medida de error para la prueba, como el estadístico Kappa o ROC, y validar los resultados.

4.1.6.2 Tengan cuidado al convertir variables categóricas en numéricas

Imaginemos que tenemos una variable categórica que necesitamos convertir a numérica. Como en el caso anterior, pero intentando una diferente **transformación**, asignen un número diferente a cada categoría.

Tenemos que tener cuidado al hacer tales transformaciones porque estamos **introduciendo orden** a la variable.

Considere el siguiente ejemplo de datos con cuatro filas. Las dos primeras variables son `visits` y `postal_code` (esto funciona como dos variables de entrada o `visits` como entrada y `postal_code` como salida).

El siguiente código mostrará las `visits` dependiendo de `postal_code` transformadas según dos criterios:

- `transformation_1`: Asigna un número de secuencia basado en el orden dado.
- `transformation_2`: Asigna un número basado en la cantidad de `visits`.

```
# Crear una muestra de datos de juguete
df_pc=data.frame(visits=c(10, 59, 27, 33), postal_code=c("AA1", "BA5

# Visualizar la tabla
knitr::kable(df_pc)
```

visits	postal_code	transformation_1	transformation_2
10	AA1	1	1
59	BA5	2	4
27	CG3	3	2
33	HJ1	4	3

```
library(gridExtra)

# Transformación 1
plot_1=ggplot(df_pc, aes(x=transformation_1, y=visits, label=postal_


# Transformación 2
plot_2=ggplot(df_pc, aes(x=transformation_2, y=visits, label=postal_

# Disponer los gráficos uno al lado del otro
grid.arrange(plot_1, plot_2, ncol=2)
```

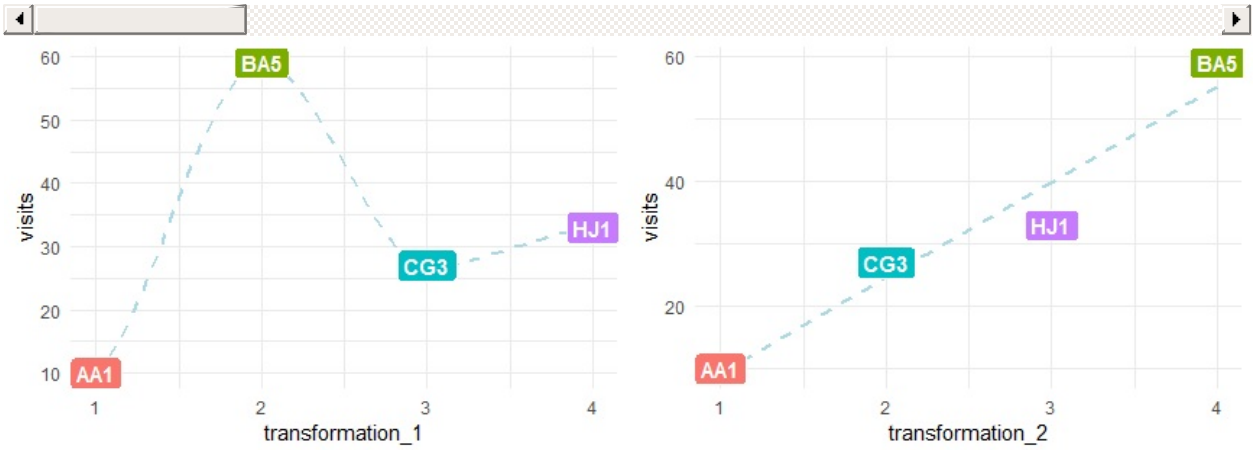


Figure 4.3: Comparación entre transformaciones de datos

Para estar seguros, nadie construye un modelo predictivo usando sólo cuatro filas; sin embargo, la intención de este ejemplo es mostrar cómo la relación cambia de no lineal (`transformation_1`) a lineal (`transformation_2`). Esto hace las cosas más fáciles para el modelo predictivo y explica la relación.

El efecto es el mismo cuando manejamos millones de filas de datos y el número de variables escala a cientos. Aprender de datos pequeños es un enfoque adecuado en estos casos.

4.1.7 Discretizando variables numéricas

Este proceso convierte los datos en una categoría dividiéndolos en segmentos. Para una definición más sofisticada, podemos citar a *Wikipedia*: *La discretización refiere al proceso de transferir funciones, modelos y ecuaciones*

continuas a contrapartes discretas.

Los segmentos también se conocen como *bins* o *buckets*. Continuemos con los ejemplos.

4.1.7.1 Sobre los datos

Los datos contienen información sobre el porcentaje de niños con retraso en el crecimiento. El valor ideal es cero.

> El indicador refleja la proporción de niños menores de 5 años que presentan retraso en el crecimiento. Los niños con retraso en el crecimiento tienen mayor riesgo de enfermedad y muerte.

Fuente: [ourworldindata.org, hunger and undernourishment]().

En primer lugar, tenemos que hacer una rápida preparación de datos. Cada fila representa un par país-año, por lo que tenemos que obtener el indicador más reciente por país.

```
data_stunting=read.csv(file = "https://goo.gl/hFEUfN",
                       header = T,
                       stringsAsFactors = F)

# Renombrar la métrica
data_stunting=
  dplyr::rename(
    data_stunting,
    share_stunted_child=
      Share.of.stunted.children.under.5
    )

# Realizar la agrupación previamente mencionada
d_stunt_grp = group_by(data_stunting, Entity) %>%
  filter(Year == max(Year)) %>%
  dplyr::summarise(share_stunted_child=
                     max(share_stunted_child)
                   )
```

Los criterios de segmentación más comunes son:

- Igual rango
- Igual frecuencia
- Segmentos personalizados

Todos están explicados a continuación.

4.1.7.2 Igual rango

El rango se suele encuentrar en los histogramas que estudian la distribución, pero es altamente susceptible a los valores atípicos. Para crear, por ejemplo, cuatro segmentos, dividimos por 4 los valores mínimos y máximos.

```
# funModeling contiene equal_freq (discretización)
library(funModeling)

# ggplot2 brinda la función 'cut_interval' que se
# utiliza para dividir las variables en base al
# criterio de igual rango
library(ggplot2)

# Al crear una variable de igual rango, agreguen
# el parámetro `dig.lab=9`para desactivar la
# notación científica al igual que con la
# función `cut`.
d_stunt_grp$share_stunted_child_eq_range=
  cut_interval(d_stunt_grp$share_stunted_child, n = 4)

# La función `describe` del paquete Hmisc package es
# extremadamente útil para analizar datos
describe(d_stunt_grp$share_stunted_child_eq_range)
```

```
## d_stunt_grp$share_stunted_child_eq_range
##        n  missing distinct
##      154        0        4
##
## Value       [1.3,15.8] (15.8,30.3] (30.3,44.8] (44.8,59.3]
## Frequency           62          45          37          10
## Proportion       0.403       0.292       0.240       0.065
```

```
# Graficar la variable
```

```
p2=ggplot(d_stunt_grp,
          aes(share_stunted_child_eq_range)
          ) +
  geom_bar(fill="#009E73") +
  theme_bw()
p2
```

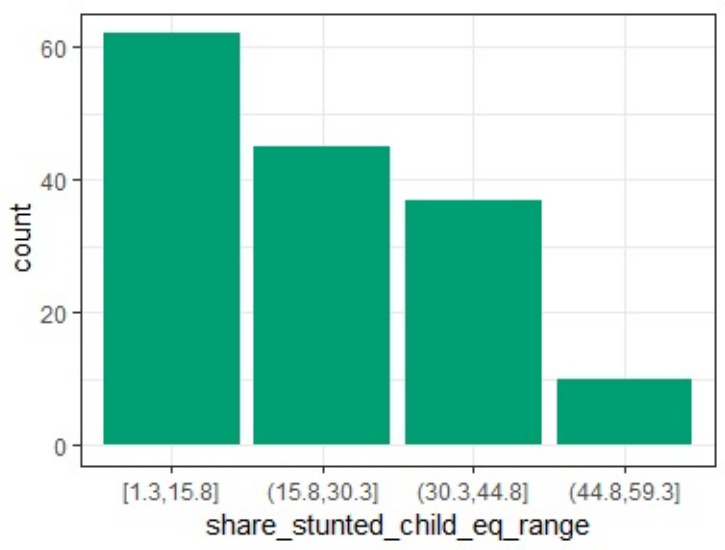


Figure 4.4: Discretización por igual frecuencia

El resultado de `describe` nos dice que hay cuatro categorías en la variable y, entre paréntesis y corchetes, el número total de casos por categoría tanto en valores absolutos como relativos, respectivamente. Por ejemplo, la categoría `(15.8,30.3]` contiene todos los casos que tienen `share_stunted_child` desde `15.8` (no inclusive) hasta `30.3` (inclusive). Aparece `45` veces y representa `29%` del total de casos.

4.1.7.3 Igual frecuencia

Esta técnica agrupa la misma cantidad de observaciones utilizando criterios basados en percentiles. Pueden encontrar más información sobre percentiles en el capítulo: Anexo 1: La magia de los percentiles.

El paquete `funModeling` incluye la función `equal_freq` para crear segmentos basándonos en este criterio:

```
d_stunt_grp$stunt_child_ef=
  equal_freq(var = d_stunt_grp$share_stunted_child,
             n_bins = 4
             )

# Analizar la variable
describe(d_stunt_grp$stunt_child_ef)
```

```
## d_stunt_grp$stunt_child_ef
##        n  missing distinct
##      154        0        4
##
## Value      [ 1.3, 9.5) [ 9.5,20.8) [20.8,32.9) [32.9,59.3]
## Frequency           40          37          39          38
## Proportion       0.260       0.240       0.253       0.247
```

```
p3=ggplot(d_stunt_grp, aes(stunt_child_ef)) +
  geom_bar(fill="#CC79A7") + theme_bw()
p3
```

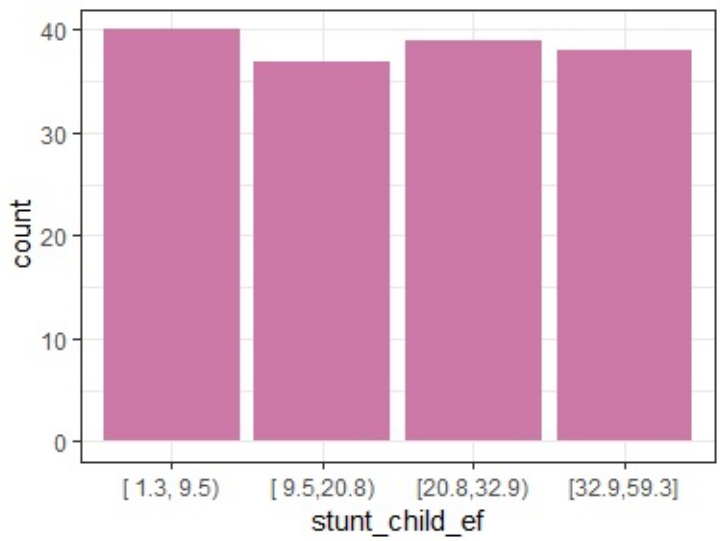


Figure 4.5: Ejemplo de igual frecuencia

En este caso, seleccionamos cuatro segmentos, por lo que cada uno contendrá aproximadamente un 25% del total.

4.1.7.4 Segmentos personalizados

Si ya tenemos los puntos de los cuales queremos los segmentos, podemos usar la función `cut`.

```
# El parámetro dig.lab desactiva la notación científica
d_stunt_grp$share_stunted_child_custom=
  cut(d_stunt_grp$share_stunted_child,
      breaks = c(0, 2, 9.4, 29, 100)
      )

describe(d_stunt_grp$share_stunted_child_custom)
```

```
## d_stunt_grp$share_stunted_child_custom
##        n  missing distinct
##      154        0        4
##
## Value         (0,2]  (2,9.4] (9.4,29] (29,100]
## Frequency         5       35       65       49
## Proportion    0.032    0.227    0.422    0.318
```

```
p4=ggplot(d_stunt_grp, aes(share_stunted_child_custom)) +
  geom_bar(fill="#0072B2") +
  theme_bw()
p4
```

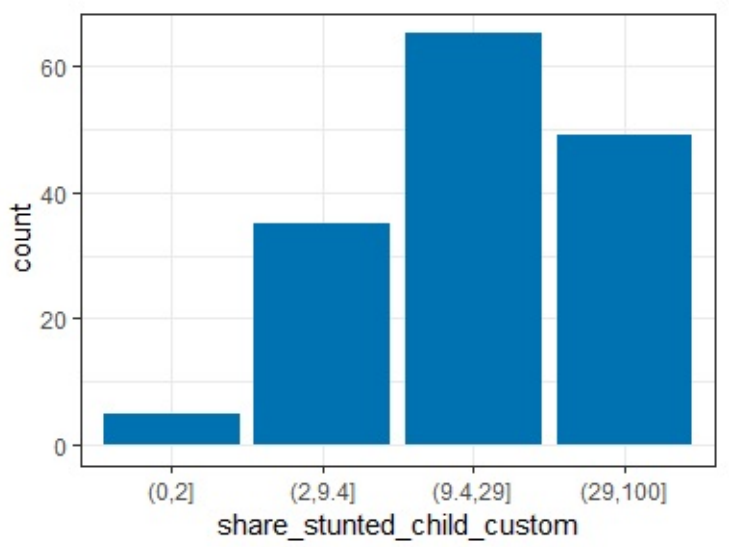


Figure 4.6: Discretización manual

Noten que solo es necesario definir el valor máximo de cada segmento.

Por lo general, no sabemos cuál es el valor mínimo o máximo. En esos casos, podemos utilizar los valores `-Inf` e `Inf`. De lo contrario, si definimos un valor que está fuera del rango, `cut` le asignará el valor `NA`.

Es una buena práctica asignar el valor mínimo y máximo usando una función. En este caso, la variable es un porcentaje, por lo que sabemos de antemano que su escala es de 0 a 100; sin embargo, ⚠ *¿qué pasaría si no conociéramos el rango?*

La función devolverá `NA` para aquellos valores por debajo o por encima de los puntos de corte. Una solución es obtener valores mínimos y máximos variables:

```
# Obtener el valor mínimo y máximo
min_value=min(d_stunt_grp$share_stunted_child)
max_value=max(d_stunt_grp$share_stunted_child)

# Configuren `include.lowest=T` para incluir el valor
# mínimo, de lo contrario será asignado como NA.
d_stunt_grp$share_stunted_child_custom_2=
  cut(d_stunt_grp$share_stunted_child,
      breaks = c(min_value, 2, 9.4, 29, max_value),
      include.lowest = T)

describe(d_stunt_grp$share_stunted_child_custom_2)
```

```
## d_stunt_grp$share_stunted_child_custom_2
##        n  missing distinct
##      154        0        4
##
## Value        [1.3,2]   (2,9.4]  (9.4,29] (29,59.3]
## Frequency          5        35        65        49
## Proportion     0.032     0.227     0.422     0.318
```

4.1.8 Discretizacación con nuevos datos

Todas estas transformaciones se realizan con un conjunto de datos de práctica basado en las distribuciones de las variables. Tal es el caso de la discretización de igual frecuencia y de igual rango. Pero, ¿qué pasaría si llegaran nuevos datos?

Si aparece un nuevo valor mínimo o máximo, afectará el rango de ubicaciones

en el método **igual rango**. Si llega algún nuevo valor, entonces moverá los puntos basados en percentiles como vimos en el método **igual frecuencia**.

Para ver qué pasa, imaginemos que añadimos cuatro casos más en el ejemplo propuesto, con los valores `88`, `2`, `7` y `3`:

```
# Simular que se agregan cuatro valores nuevos
updated_data=c(d_stunt_grp$share_stunted_child, 88, 2, 7, 3)

# Discretizar por igual frecuencia
updated_data_eq_freq=equal_freq(updated_data,4)

# Resultados en...
describe(updated_data_eq_freq)
```

```
## updated_data_eq_freq
##        n  missing distinct
##      158        0        4
##
## Value      [ 1.3, 9.3) [ 9.3,20.6) [20.6,32.9) [32.9,88.0]
## Frequency           40          39          40          39
## Proportion       0.253       0.247       0.253       0.247
```

Ahora comparemos con los segmentos que creamos anteriormente:

```
describe(d_stunt_grp$stunt_child_ef)
```

```
## d_stunt_grp$stunt_child_ef
##        n  missing distinct
##      154        0        4
##
## Value      [ 1.3, 9.5) [ 9.5,20.8) [20.8,32.9) [32.9,59.3]
## Frequency           40          37          39          38
## Proportion       0.260       0.240       0.253       0.247
```

¡Todos los segmentos cambiaron! 😱 Dado que estas son nuevas categorías, el modelo predictivo fallará a la hora de procesarlas porque son todos valores nuevos.

La solución es conservar los puntos de corte cuando preparamos los datos. Luego, ejecutamos el modelo en producción, utilizamos el segmento de

discretización manual y, así, forzamos que cada caso quede en la categoría correspondiente. De esta manera, el modelo predictivo siempre *ve* lo mismo.

La solución será detallada en la siguiente sección.

4.1.9 Discretización automática de data frames

El paquete `funModeling` (desde la versión > 1.6.6) introduce dos funciones: `discretize_get_bins` y `discretize_df` que operan juntas para ayudarnos en la tarea de discretización.

```
# Primero cargamos las bibliotecas
# install.packages("funModeling")
library(funModeling)
library(dplyr)
```

Veamos un ejemplo. Primero, corroboramos los tipos de datos actuales:

```
df_status(heart_disease, print_results = F) %>% select(variable, typ
```

```
##                   variable    type unique q_na
## 1                   gender  factor      2    0
## 2               chest_pain  factor      4    0
## 3     fasting_blood_sugar  factor      2    0
## 4         resting_electro  factor      3    0
## 5                    thal  factor      3    2
## 6            exter_angina  factor      2    0
## 7       has_heart_disease  factor      2    0
## 8                     age integer     41    0
## 9  resting_blood_pressure integer     50    0
## 10      serum_cholestoral integer    152    0
## 11         max_heart_rate integer     91    0
## 12            exer_angina integer      2    0
## 13                  slope integer      3    0
## 14      num_vessels_flour integer      4    4
## 15 heart_disease_severity integer      5    0
## 16                oldpeak numeric     40    0
```

Tenemos variables factor, enteras, y numéricas: ¡una buena mezcla! La transformación tiene dos pasos. Primero, obtiene los valores de corte o de umbral donde comienza cada segmento. El segundo paso es utilizar el umbral para obtener las variables como categóricas.

Discretizaremos dos variables en el siguiente ejemplo: `max_heart_rate` y `oldpeak`. Además, agregaremos algunos valores `NA` a `oldpeak` para evaluar cómo opera la función con datos faltantes.

```
# Crear una copia para conservar los datos originales intactos
heart_disease_2=heart_disease

# Introducir algunos valores faltantes en las primeras 30 filas de l
heart_disease_2$oldpeak[1:30]=NA
```

Paso 1) Obtener los umbrales de segmento para cada variable de entrada:

`discretize_get_bins` devuelve un data frame que necesitaremos para la función `discretize_df`, que genera el data frame final procesado.

```
d_bins=discretize_get_bins(data=heart_disease_2, input=c("max_heart_
```

```
## Variables processed: max_heart_rate, oldpeak
```

```
# Verificar el objeto `d_bins`:
d_bins
```

```
##         variable                cuts
## 1 max_heart_rate 131|147|160|171|Inf
## 2        oldpeak   0.1|0.3|1.1|2|Inf
```

Parámetros:

- `data`: el data frame que contiene las variables a procesar.
- `input`: vector de strings que contienen los nombres de las variables.
- `n_bins`: la cantidad de segmentos que tendremos en los datos discretizados.

Podemos ver el punto del umbral (o límite superior) para cada variable.

Nota: Cambios de la versión 1.6.6 a 1.6.7:

- `discretize_get_bins` no crea el umbral `-Inf` dado que dicho valor siempre fue considerado el mínimo.
- La categoría de un valor ahora se representa como un rango, por ejemplo, lo que era `"5"`, ahora es `"[5, 6)"`.
- El formato de los segmentos puede haber cambiado, si utilizaron esta función en producción, tienen que verificar los nuevos valores.

¡Es hora de continuar con el siguiente paso!

Paso 2) Aplicar los umbrales para cada variable:

```
# Ahora se puede aplicar en el mismo data frame o
# en una nueva (por ejemplo, en un modelo predictivo
# en el que los datos cambian con el tiempo)
heart_disease_discretized =
  discretize_df(data=heart_disease_2,
                data_bins=d_bins,
                stringsAsFactors=T)
```

```
## Variables processed: max_heart_rate, oldpeak
```

Parámetros:

- `data`: data frame que contiene las variables a procesar.
- `data_bins`: data frame resultado dem`discretize_get_bins`. Si el usuario la modifica, entonces cada límite superior deberá estar separado por un carácter pipe (`|`) como se ve en el ejemplo.
- `stringsAsFactors`: `TRUE` por defecto, las variables finales serán factor (en lugar de un carácter) y útiles para graficar.

4.1.9.1 Resultados finales y sus gráficos

Antes y después:

```
##   max_heart_rate_before max_heart_rate_after oldpeak_before oldpe
## 1                   171          [ 171, Inf]             NA
```

```
## 2                   114          [-Inf, 131)            NA
## 3                   151          [ 147, 160)           1.8   [ 1
## 4                   160          [ 160, 171)           1.4   [ 1
## 5                   158          [ 147, 160)           0.0   [-I
## 6                   161          [ 160, 171)           0.5   [ 0
```

Distribución final:

```
freq(heart_disease_discretized %>%
           select(max_heart_rate,oldpeak),
     plot = F)
```

```
##   max_heart_rate frequency percentage cumulative_perc
## 1    [-Inf, 131)        63      20.79           20.79
## 2    [ 147, 160)        62      20.46           41.25
## 3    [ 160, 171)        62      20.46           61.71
## 4    [ 131, 147)        59      19.47           81.18
## 5    [ 171, Inf]        57      18.81          100.00
##
##       oldpeak frequency percentage cumulative_perc
## 1 [-Inf, 0.1)        97      32.01           32.01
## 2 [ 0.3, 1.1)        54      17.82           49.83
## 3 [ 1.1, 2.0)        54      17.82           67.65
## 4 [ 2.0, Inf]        50      16.50           84.15
## 5         NA.        30       9.90           94.05
## 6 [ 0.1, 0.3)        18       5.94          100.00

## [1] "Variables processed: max_heart_rate, oldpeak"
```

```
p5=ggplot(heart_disease_discretized,
          aes(max_heart_rate)) +
  geom_bar(fill="#0072B2") +
  theme_bw() +
  theme(axis.text.x =
          element_text(angle = 45, vjust = 1, hjust=1)
        )

p6=ggplot(heart_disease_discretized,
          aes(oldpeak)) +
  geom_bar(fill="#CC79A7") +
  theme_bw() +
  theme(axis.text.x =
          element_text(angle = 45, vjust = 1, hjust=1)
        )
```

```
gridExtra::grid.arrange(p5, p6, ncol=2)
```

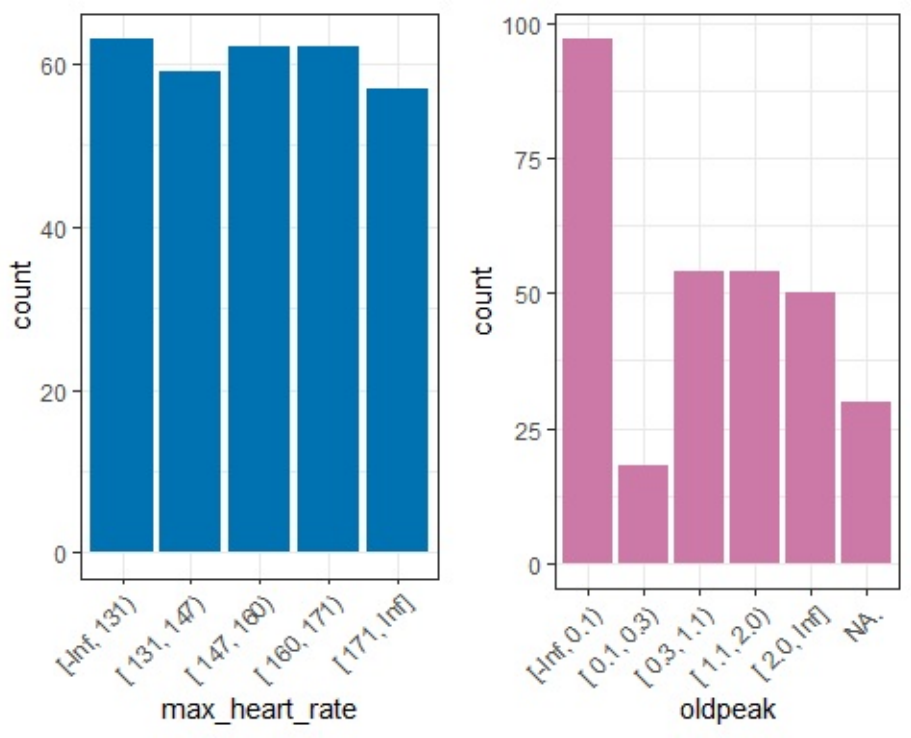


Figure 4.7: Resultados de la discretización automática

A veces, no es posible obtener la misma cantidad de casos por segmento al computar por **igual frecuencia**, como en el caso de la variable `oldpeak`.

4.1.9.2 Manejo de valores NA

Con respecto a los valores `NA`, la nueva variable `oldpeak` tiene seis categorías: cinco categorías definidas en `n_bins=5` más el valor `NA`. Noten el punto al final que indica la presencia de valores faltantes.

4.1.9.3 Más información

- `discretize_df` nunca devolverá un valor `NA` sin transformarlo en el string `NA.`.
- `n_bins` configura la cantidad de segmentos para todas las variables.
- Si falta `input`, entonces correrá para todas las variables numéricas o enteras en las que la cantidad de valores únicos sea mayor que la cantidad de segmentos (`n_bins`).

- Solo las variables definidas en `input` serán procesadas, mientras que las restantes **no serán modificadas en absoluto**.
- `discretize_get_bins` devuelve un data frame que puede ser modificado a mano como sea necesario, ya sea en un archivo de texto o en la sesión de R.

4.1.9.4 Discretización con nuevos datos

En nuestros datos, el valor mínimo para `max_heart_rate` es 71. La preparación de datos debe ser robusta cuando incorporamos nuevos casos; por ejemplo, si llega un nuevo paciente cuya `max_heart_rate` es 68, entonces el proceso actual lo asignará a la categoría más baja.

En otras funciones de otros paquetes, esta preparación puede devolver un `NA` porque está fuera del segmento.

Como señalamos anteriormente, si los nuevos datos llegan a lo largo del tiempo, es probable que obtengan nuevos valores mínimos/máximos. Esto puede romper nuestro proceso. Para resolver esto, `discretize_df` siempre tendrá como mínimo/máximo los valores `-Inf`/`Inf`; por lo tanto, cualquier nuevo valor que caiga por debajo o por encima del mínimo/máximo se añadirá al segmento más bajo o más alto según corresponda.

El data frame devuelto por `discretize_get_bins` debe ser guardado para poder aplicarlo a nuevos datos. Si la discretización no está pensada para funcionar con nuevos datos, entonces no tiene sentido tener dos funciones: puede ser solo una. Además, no habría necesidad de guardar los resultados de `discretize_get_bins`.

Con este enfoque de dos pasos, podemos manejar ambos casos.

4.1.9.5 Conclusiones sobre la discretización de dos pasos

El uso de `discretize_get_bins` + `discretize_df` permite una rápida preparación de datos, con un data frame limpio y listo para usar. Dado que muestra claramente dónde empieza y termina cada segmento, resulta indispensable a la hora de realizar informes estadísticos.

La decisión de *no fallar* a la hora de manejar un nuevo valor mínimo o máximo cuando incorporamos nuevos datos es **solo una decisión**. En algunos contextos,

fracasar puede ser el comportamiento deseado.

La intervención humana: La manera más fácil de discretizar un data frame es seleccionar la misma cantidad de segmentos para aplicar a cada variable, igual que en el ejemplo que vimos. Sin embargo, si es necesario realizar ajustes, entonces algunas variables pueden necesitar un **número diferente de segmentos.** Por ejemplo, una variable con menos dispersión puede funcionar bien con pocos segmentos.

Los valores más comunes para el número de segmentos suelen ser 3, 5, 10 ó 20 (pero no más). Esta decisión corre por cuenta del científico de datos.

4.1.9.6 Bonus track: El arte del equilibrio ⚖

- Alta cantidad de segmentos => Más ruido capturado
- Baja cantidad de segmentos => Demasiada simplificación, menos varianza.

¿Estos términos les suenan parecidos a otros empleados en el ámbito de *machine learning*?

La respuesta: *¡Sí!* Solo por mencionar un ejemplo: buscar el equilibrio a la hora de agregar o quitar variables en un modelo predictivo.

- Más variables: Alerta de sobreajuste (el modelo predictivo es demasiado detallado).
- Menos variables: Peligro de subajuste (no hay suficiente información para captar los patrones generales).

Como la filosofía oriental ha señalado durante miles de años, hay un arte en encontrar el equilibrio justo entre un valor y su opuesto.

4.1.10 Reflexiones finales

Como podemos ver, **cada decisión tiene su costo** en la discretización o preparación de datos. ¿Cómo creen que un *sistema automático o inteligente*

resolverá todas estas situaciones sin la intervención o el análisis humano?

Para estar seguros, podemos delegar algunas tareas a procesos automáticos; sin embargo, **los humanos son indispensables en la etapa de preparación de datos**, brindando los datos de entrada correctos para procesar.

La asignación de variables como categóricas o numéricas, los dos tipos de datos más utilizados, varía según la naturaleza de los datos y los algoritmos seleccionados, ya que algunos sólo soportan un tipo de datos.

La conversión **introduce algún sesgo** al análisis. Un caso similar existe cuando se trata de valores faltantes: Manejo e imputación de datos faltantes.

Cuando trabajamos con variables categóricas, podemos cambiar su distribución reorganizando las categorías según una variable objetivo para **exponer mejor su relación**. Convertir una relación variable no lineal en una lineal.

4.1.11 Bonus track

Volvamos a la sección sobre discretización de variables y grafiquemos todas las transformaciones que hemos visto hasta ahora:

```
grid.arrange(p2, p3, p4, ncol = 3)
```

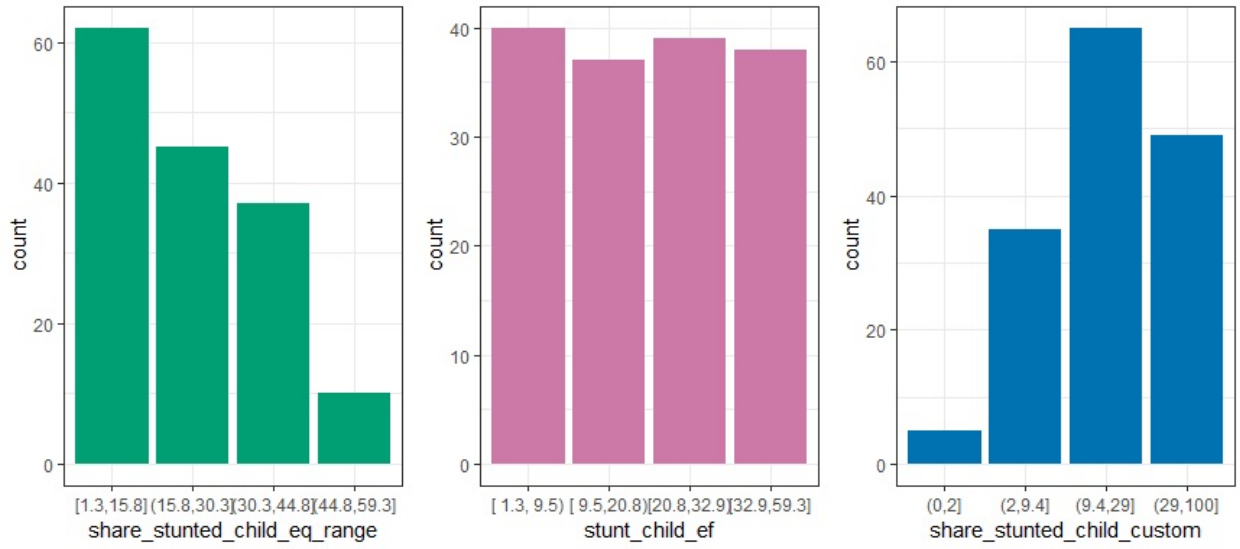


Figure 4.8: Mismos datos, diferentes visualizaciones

Los datos ingresados son siempre los mismos. Sin embargo, todos estos métodos **exhiben diferentes perspectivas de la misma *cosa***.

Algunas perspectivas son más adecuadas que otras para ciertas situaciones, como el uso de **igual frecuencia** para **modelos predictivos**.

Aunque este caso solo considera una variable, el razonamiento es el mismo si tenemos más variables a la vez, es decir, un espacio "N-dimensional".

Cuando construimos modelos predictivos, describimos el mismo grupo de puntos de diferentes maneras, al igual que cuando distintas personas dan su opinión sobre un objeto.

4.2 Variables de alta cardinalidad en estadística descriptiva

4.2.1 ¿De qué se trata esto?

Una variable de **alta cardinalidad** es aquella que puede tomar *muchos* valores diferentes. Por ejemplo, la variable país.

Este capítulo cubrirá la reducción de la cardinalidad basada en la regla de Pareto, usando la función `freq` que da una visión rápida sobre dónde se concentran la mayoría de los valores y la distribución de la variable.

4.2.2 Alta cardinalidad en estadística descriptiva

El siguiente ejemplo contiene una encuesta de 910 casos, con 3 columnas: `person`, `country` y `has_flu`, que indica haber tenido gripe en el último mes.

```
library(funModeling)
```

Los datos de `data_country` están incluidos en el paquete `funModeling` (por favor actualicen a la versión 1.6).

Rápido análisis numérico de `data_country` (primeras 10 filas)

```
# Graficar las primeras 10 filas
head(data_country, 10)
```

```
##     person     country has_flu
## 478    478      France      no
## 990    990      Brazil      no
## 606    606      France      no
## 575    575 Philippines      no
## 806    806      France      no
## 232    232      France      no
## 422    422      Poland      no
## 347    347     Romania      no
## 858    858     Finland      no
## 704    704      France      no
```

```
# Explorar los datos, visualizando solamente las primeras 10 filas
head(freq(data_country, "country"), 10)
```

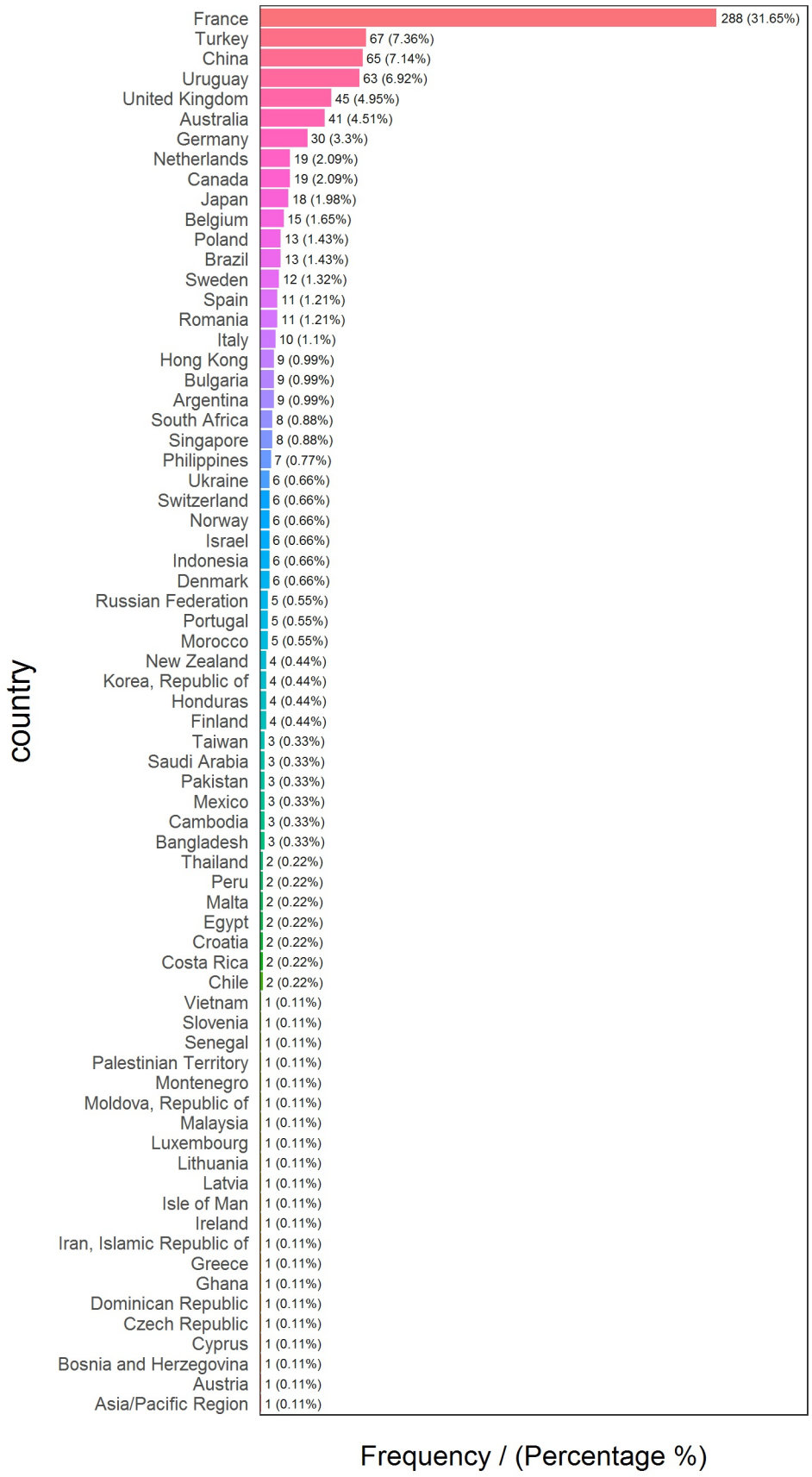

country
France 288 (31.65%)
Turkey 67 (7.36%)
China 65 (7.14%)
Uruguay 63 (6.92%)
United Kingdom 45 (4.95%)
Australia 41 (4.51%)
Germany 30 (3.3%)
Netherlands 19 (2.09%)
Canada 19 (2.09%)
Japan 18 (1.98%)
Belgium 15 (1.65%)
Poland 13 (1.43%)
Brazil 13 (1.43%)
Sweden 12 (1.32%)
Spain 11 (1.21%)
Romania 11 (1.21%)
Italy 10 (1.1%)
Hong Kong 9 (0.99%)
Bulgaria 9 (0.99%)
Argentina 9 (0.99%)
South Africa 8 (0.88%)
Singapore 8 (0.88%)
Philippines 7 (0.77%)
Ukraine 6 (0.66%)
Switzerland 6 (0.66%)
Norway 6 (0.66%)
Israel 6 (0.66%)
Indonesia 6 (0.66%)
Denmark 6 (0.66%)
Russian Federation 5 (0.55%)
Portugal 5 (0.55%)
Morocco 5 (0.55%)
New Zealand 4 (0.44%)
Korea, Republic of 4 (0.44%)
Honduras 4 (0.44%)
Finland 4 (0.44%)
Taiwan 3 (0.33%)
Saudi Arabia 3 (0.33%)
Pakistan 3 (0.33%)
Mexico 3 (0.33%)
Cambodia 3 (0.33%)
Bangladesh 3 (0.33%)
Thailand 2 (0.22%)
Peru 2 (0.22%)
Malta 2 (0.22%)
Egypt 2 (0.22%)
Croatia 2 (0.22%)
Costa Rica 2 (0.22%)
Chile 2 (0.22%)
Vietnam 1 (0.11%)
Slovenia 1 (0.11%)
Senegal 1 (0.11%)
Palestinian Territory 1 (0.11%)
Montenegro 1 (0.11%)
Moldova, Republic of 1 (0.11%)
Malaysia 1 (0.11%)
Luxembourg 1 (0.11%)
Lithuania 1 (0.11%)
Latvia 1 (0.11%)
Isle of Man 1 (0.11%)
Ireland 1 (0.11%)
Iran, Islamic Republic of 1 (0.11%)
Greece 1 (0.11%)
Ghana 1 (0.11%)
Dominican Republic 1 (0.11%)
Czech Republic 1 (0.11%)
Cyprus 1 (0.11%)
Bosnia and Herzegovina 1 (0.11%)
Austria 1 (0.11%)
Asia/Pacific Region 1 (0.11%)
Frequency / (Percentage %)

Figure 4.9: Análisis de frecuencia por país

```
##           country frequency percentage cumulative_perc
## 1          France       288      31.65           31.65
## 2          Turkey        67       7.36           39.01
## 3           China        65       7.14           46.15
## 4         Uruguay        63       6.92           53.07
## 5  United Kingdom        45       4.95           58.02
## 6       Australia        41       4.51           62.53
## 7         Germany        30       3.30           65.83
## 8          Canada        19       2.09           67.92
## 9     Netherlands        19       2.09           70.01
## 10          Japan        18       1.98           71.99
```

```
# Explorar los datos
freq(data_country, "has_flu")
```

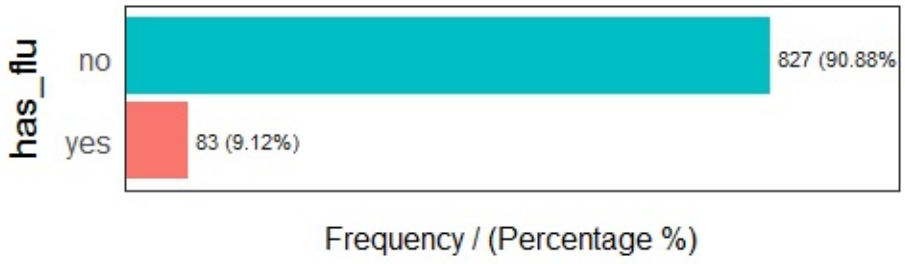


Figure 4.10: Análisis de frecuencia de casos con gripe

```
##   has_flu frequency percentage cumulative_perc
## 1      no       827      90.88           90.88
## 2     yes        83       9.12          100.00
```

La última tabla muestra que hay **sólo 83 filas** en las que `has_flu="yes"`, lo que representa cerca del 9% del total de personas (que tuvieron gripe).

Pero muchos de ellos casi no tienen participación en los datos. Esta es la *cola larga*, por lo que una técnica para reducir la cardinalidad es conservar aquellas categorías que están presentes en un alto porcentaje de los datos, por ejemplo 70, 80 o 90%, el principio de Pareto.

```
# La función 'freq', del paquete 'funModeling', recupera el porcenta
```

```
country_freq=freq(data_country, 'country', plot = F)

# Dado que 'country_freq' es una tabla ordenada por frecuencia, insp
country_freq[1:10,]
```

```
##           country frequency percentage cumulative_perc
## 1          France       288      31.65           31.65
## 2          Turkey        67       7.36           39.01
## 3           China        65       7.14           46.15
## 4         Uruguay        63       6.92           53.07
## 5  United Kingdom        45       4.95           58.02
## 6       Australia        41       4.51           62.53
## 7         Germany        30       3.30           65.83
## 8          Canada        19       2.09           67.92
## 9     Netherlands        19       2.09           70.01
## 10          Japan        18       1.98           71.99
```

Vemos que 10 representan más del 70% de los casos. Podemos asignar la categoría `other` a los casos restantes y graficar:

```
data_country$country_2=ifelse(data_country$country %in% country_freq
freq(data_country, 'country_2')
```

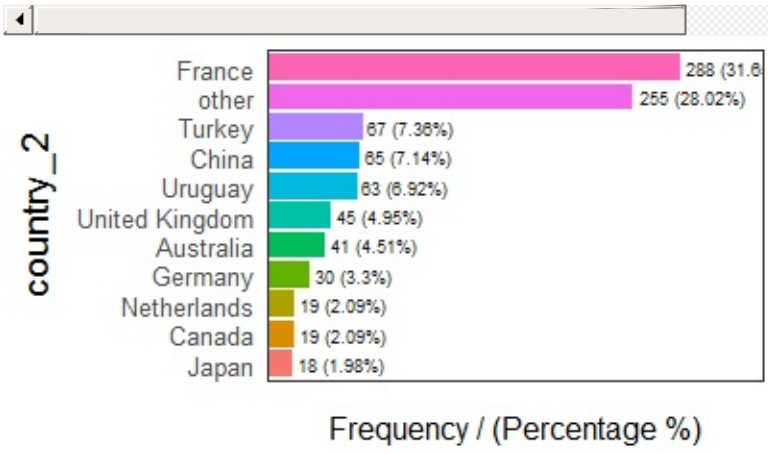


Figure 4.11: Variable país modificada - análisis de frecuencia

```
##         country_2 frequency percentage cumulative_perc
## 1          France       288      31.65           31.65
## 2           other       255      28.02           59.67
```

```
## 3          Turkey        67       7.36           67.03
## 4           China        65       7.14           74.17
## 5         Uruguay        63       6.92           81.09
## 6  United Kingdom        45       4.95           86.04
## 7       Australia        41       4.51           90.55
## 8         Germany        30       3.30           93.85
## 9          Canada        19       2.09           95.94
## 10    Netherlands        19       2.09           98.03
## 11          Japan        18       1.98          100.00
```

4.2.3 Comentarios finales

Las categorías poco representativas a veces son errores en los datos, como tener: "Egipto", "Eggipto.", y pueden dar alguna evidencia de malos hábitos de recolección de datos y/o posibles errores en la recolección de la fuente.

No existe una regla general para reducir los datos, depende de cada caso particular.

Próximo capítulo recomendado: Variables de alta cardinalidad en modelado predictivo

---

4.3 Variables de alta cardinalidad en modelado predictivo

4.3.1 ¿De qué se trata esto?

Como hemos visto en el capítulo anterior, *Alta cardinalidad en estadística descriptiva*, conservamos las categorías con la mayor representatividad, pero ¿qué tal si podemos tener otra variable para predecir con ella? Es decir, predecir `has_flu` basándonos en `country`.

Utilizar el último método puede destruir la información de la variable, por lo que **pierde poder predictivo**. En este capítulo iremos más allá en el método descrito anteriormente, utilizando una función de agrupación automática -`auto_grouping`- y navegando a través de la estructura de la variable para dar algunas ideas sobre cómo optimizar una variable categórica, pero lo más importante: animar al lector a realizar sus propias optimizaciones.

Otros autores han nombrado este reagrupamiento como reducción de la cardinalidad o **encoding**.

¿Qué vamos a repasar en este capítulo?

- Concepto de representatividad de los datos (tamaño de muestra).
- Tamaño de muestra con una variable objetivo o de resultado.
- De R: Presentar un método para ayudar a reducir la cardinalidad y analizar numéricamente variables categóricas.
- Un ejemplo práctico de antes y después que reduce la cardinalidad y facilita la extracción de ideas.
- Cómo diferentes modelos, como un random forest o *gradient boosting machine* (GBM, en inglés), manejan las variables categóricas.

4.3.2 Pero, ¿es necesario reagrupar la variable?

Depende del caso, pero la respuesta más rápida es sí. En este capítulo veremos un caso en el que esta preparación de datos aumenta la precisión general (medida por área debajo de la curva ROC).

Existe un equilibrio entre la **representación de los datos** (cuántas filas tiene

cada categoría) y cómo se relaciona cada categoría con la variable de resultado. Por ejemplo: algunos países son más propensos a los casos de gripe que otros.

```
# Cargar funModeling >=1.6 que contiene todas las funciones para lid
library(funModeling)
library(dplyr)
```

Analizamos numéricamente `data_country`, que viene en el paquete `funModeling` (por favor actualicen a la versión > 1.6.5).

Análisis rápido de `data_country` (primeras 10 filas)

```
# Graficar las primeras 10 filas
head(data_country, 10)
```

```
##     person     country has_flu country_2
## 478    478      France      no    France
## 990    990      Brazil      no     other
## 606    606      France      no    France
## 575    575 Philippines      no     other
## 806    806      France      no    France
## 232    232      France      no    France
## 422    422      Poland      no     other
## 347    347     Romania      no     other
## 858    858     Finland      no     other
## 704    704      France      no    France
```

```
# Explorar los datos, visualizando solamente las primeras 10 filas
head(freq(data_country, "country"), 10)
```

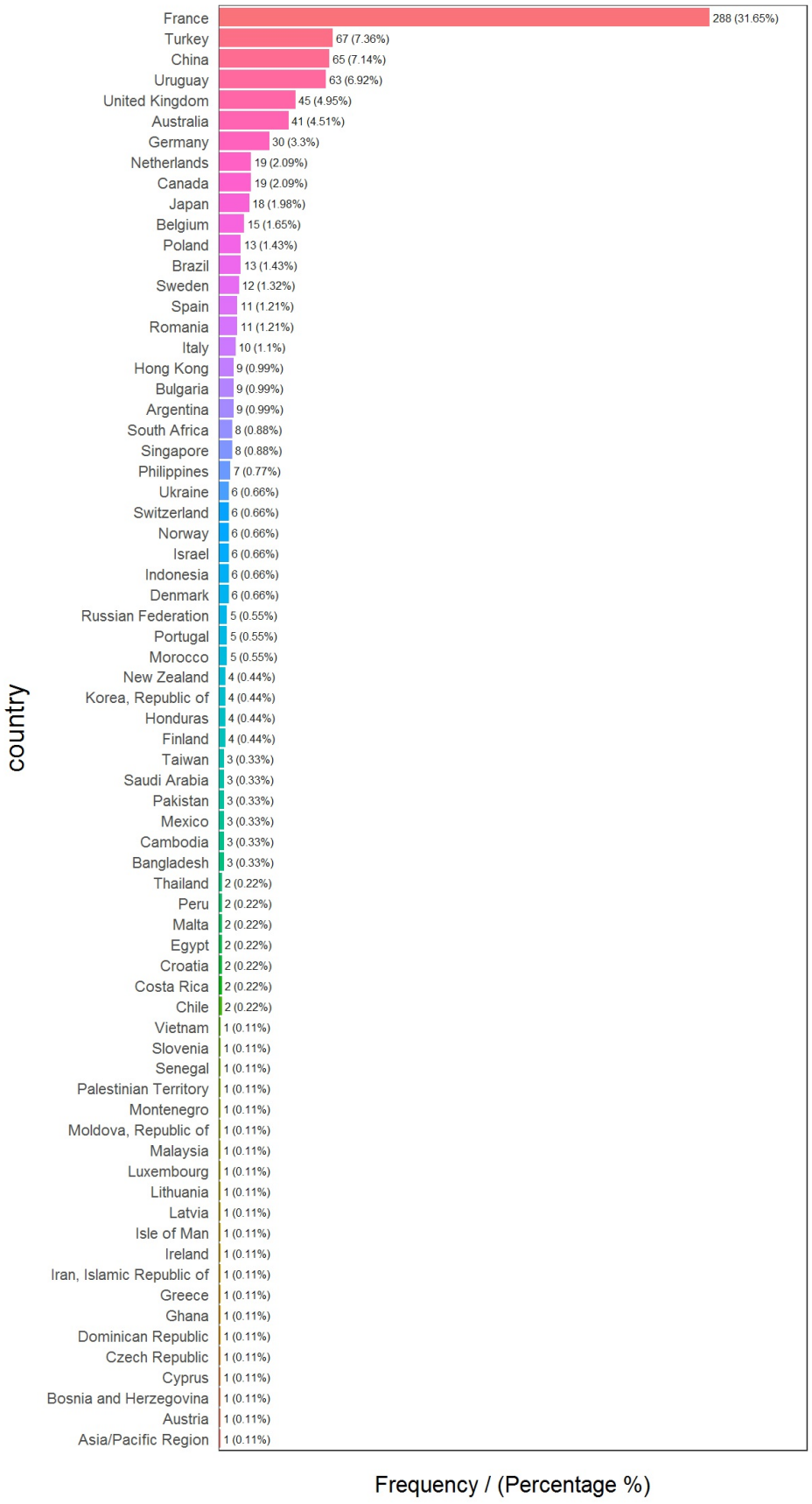

country
France 288 (31.65%)
Turkey 67 (7.36%)
China 65 (7.14%)
Uruguay 63 (6.92%)
United Kingdom 45 (4.95%)
Australia 41 (4.51%)
Germany 30 (3.3%)
Netherlands 19 (2.09%)
Canada 19 (2.09%)
Japan 18 (1.98%)
Belgium 15 (1.65%)
Poland 13 (1.43%)
Brazil 13 (1.43%)
Sweden 12 (1.32%)
Spain 11 (1.21%)
Romania 11 (1.21%)
Italy 10 (1.1%)
Hong Kong 9 (0.99%)
Bulgaria 9 (0.99%)
Argentina 9 (0.99%)
South Africa 8 (0.88%)
Singapore 8 (0.88%)
Philippines 7 (0.77%)
Ukraine 6 (0.66%)
Switzerland 6 (0.66%)
Norway 6 (0.66%)
Israel 6 (0.66%)
Indonesia 6 (0.66%)
Denmark 6 (0.66%)
Russian Federation 5 (0.55%)
Portugal 5 (0.55%)
Morocco 5 (0.55%)
New Zealand 4 (0.44%)
Korea, Republic of 4 (0.44%)
Honduras 4 (0.44%)
Finland 4 (0.44%)
Taiwan 3 (0.33%)
Saudi Arabia 3 (0.33%)
Pakistan 3 (0.33%)
Mexico 3 (0.33%)
Cambodia 3 (0.33%)
Bangladesh 3 (0.33%)
Thailand 2 (0.22%)
Peru 2 (0.22%)
Malta 2 (0.22%)
Egypt 2 (0.22%)
Croatia 2 (0.22%)
Costa Rica 2 (0.22%)
Chile 2 (0.22%)
Vietnam 1 (0.11%)
Slovenia 1 (0.11%)
Senegal 1 (0.11%)
Palestinian Territory 1 (0.11%)
Montenegro 1 (0.11%)
Moldova, Republic of 1 (0.11%)
Malaysia 1 (0.11%)
Luxembourg 1 (0.11%)
Lithuania 1 (0.11%)
Latvia 1 (0.11%)
Isle of Man 1 (0.11%)
Ireland 1 (0.11%)
Iran, Islamic Republic of 1 (0.11%)
Greece 1 (0.11%)
Ghana 1 (0.11%)
Dominican Republic 1 (0.11%)
Czech Republic 1 (0.11%)
Cyprus 1 (0.11%)
Bosnia and Herzegovina 1 (0.11%)
Austria 1 (0.11%)
Asia/Pacific Region 1 (0.11%)
Frequency / (Percentage %)

Figure 4.12: Primeros 10 países

```
##           country frequency percentage cumulative_perc
## 1          France       288      31.65           31.65
## 2          Turkey        67       7.36           39.01
## 3           China        65       7.14           46.15
## 4         Uruguay        63       6.92           53.07
## 5  United Kingdom        45       4.95           58.02
## 6       Australia        41       4.51           62.53
## 7         Germany        30       3.30           65.83
## 8          Canada        19       2.09           67.92
## 9     Netherlands        19       2.09           70.01
## 10          Japan        18       1.98           71.99
```

```
# Explorar los datos
freq(data_country, "has_flu")
```

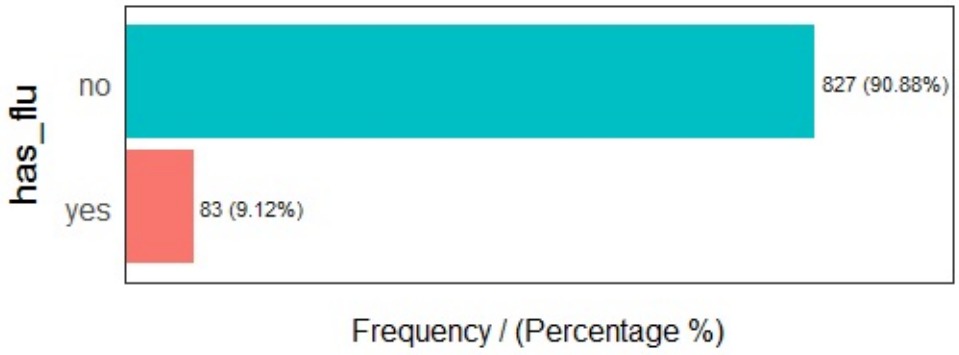


Figure 4.13: Distribucíón de la variable has flu

```
##   has_flu frequency percentage cumulative_perc
## 1      no       827      90.88           90.88
## 2     yes        83       9.12          100.00
```

4.3.3 El caso

El modelo predictivo intentará mapear ciertos valores con ciertos resultados, en nuestro caso la variable objetivo es binaria.

Calcularemos un análisis numérico completo de `country` con respecto a la variable objetivo `has_flu` basado en `categ_analysis`.

Cada fila representa una categoría única de variables `input`. Y en cada fila podemos encontrar atributos que definen cada categoría en términos de representatividad y probabilidad.

```
# `categ_analysis` está disponible en "funModeling" >= v1.6, por fav
country_profiling=categ_analysis(data=data_country, input="country",

# Visualizar las primeras 15 filas (países) de 70.
head(country_profiling, 15)
```

	country	mean_target	sum_target	perc_target	q_rows	perc_rows
1	Malaysia	1.000	1	0.012	1	0.001
2	Mexico	0.667	2	0.024	3	0.003
3	Portugal	0.200	1	0.012	5	0.005
4	United Kingdom	0.178	8	0.096	45	0.049
5	Uruguay	0.175	11	0.133	63	0.069
6	Israel	0.167	1	0.012	6	0.007
7	Switzerland	0.167	1	0.012	6	0.007
8	Canada	0.158	3	0.036	19	0.021
9	France	0.142	41	0.494	288	0.316
10	Argentina	0.111	1	0.012	9	0.010
11	Germany	0.100	3	0.036	30	0.033
12	Australia	0.098	4	0.048	41	0.045
13	Romania	0.091	1	0.012	11	0.012
14	Spain	0.091	1	0.012	11	0.012
15	Sweden	0.083	1	0.012	12	0.013

Figure 4.14: Analizando el objetivo vs. los datos ingresados

- Nota 1: *La primera columna ajusta automáticamente su nombre en base a la variable `input`*
- Nota 2: La variable *`has_flu` tiene valores `yes` y `no`, `categ_analysis` asigna internamente el número **1** a la clase menos representativa, `yes` en este caso, para calcular el promedio, suma y porcentaje.*

Estas son las métricas que devuelve `categ_analysis`:

- `country`: nombre de cada categoría en la variable `input`.
- `mean_target`: `sum_target/q_rows`, número promedio de `has_flu="yes"` para una categoría. Esta es la probabilidad.
- `sum_target`: cantidad de valores `has_flu="yes"` en cada categoría.
- `perc_target`: lo mismo que `sum_target` pero expresado como porcentaje, `sum_target of each category / total sum_target`. Esta columna suma

`1.00`.

- `q_rows`: cantidad de filas que, más allá de la variable `has_flu`, cayeron en una categoría. Es la distribución de `input`. Esta columna suma la cantidad total de filas analizadas.
- `perc_rows`: relacionado con `q_rows`, representa la porción o porcentaje de cada categoría. Esta columna suma `1.00`.

4.3.3.1 ¿Qué conclusiones podemos extraer de esto?

Leyendo como ejemplo la primera fila de `France`:

- 41 personas tienen gripe (`sum_target=41`). Estas 41 personas representan casi el 50% del total de personas con gripe (`perc_target=0.494`).
- La probabilidad de tener gripe en Francia es 14.2% (`mean_target=0.142`)
- Total de filas de Francia=288 -de 910-. Esta es la variable `q_rows`; `perc_rows` es el mismo número pero en porcentaje.

Sin considerar el filtro por país, tenemos:

- La columna `sum_target` suma el total de personas con gripe en los datos actuales.
- La columna `perc_target` suma `1.00` -o 100%
- La columna `q_rows` suma el total de filas presentes en el data frame `data_country`.
- La columna `perc_rows` suma `1.00` o 100%.

4.3.4 Análisis para el modelado predictivo

Cuando desarrollamos modelos predictivos, puede que nos interesen aquellos valores que aumentan la probabilidad de un determinado evento. En nuestro caso:

¿Cuáles son los países que maximizan la probabilidad de encontrar personas con gripe?

Fácil, tomemos `country_profiling` en orden descendiente según `mean_target`:

```
# Ordenar country_profiling por mean_target y luego tomar los primer
arrange(country_profiling, -mean_target) %>%  head(.)
```

```
##          country mean_target sum_target perc_target q_rows perc_r
## 1       Malaysia       1.000          1       0.012      1     0.
## 2         Mexico       0.667          2       0.024      3     0.
## 3       Portugal       0.200          1       0.012      5     0.
## 4 United Kingdom       0.178          8       0.096     45     0.
## 5        Uruguay       0.175         11       0.133     63     0.
## 6         Israel       0.167          1       0.012      6     0.
```

¡Genial! Tenemos a `Malasyia` como el país con mayor probabilidad de tener gripe! El 100% de las personas ahí tienen gripe (`mean_has_flu=1.000`).

Pero nuestro sentido común nos aconseja que *quizás* algo anda mal....

¿Cuántas filas tiene Malasyia? Respuesta: 1. -columna: `q_rows=1` ¿Cuántos casos positivos tiene Malasyia? Respuesta: 1 -columna: `sum_target=1`.

Dado que no se puede aumentar la muestra vean que si esta proporción se mantiene alta, contribuirá a un **sobreajuste** y creará un sesgo en el modelo predictivo.

¿Y qué pasa con `Mexico`? 2 de cada 3 tienen gripe.... todavía parece baja. Sin embargo, `Uruguay` tiene un 17,3% de probabilidad -11 de 63 casos- y estos 63 casos representan casi el 7% de la población total (`perc_row=0,069`), esta proporción parece más creíble.

A continuación se presentan algunas ideas para tratar esto:

4.3.4.1 Caso 1: Reducción mediante la recategorización de valores menos representativos

Mantengamos todos los casos que tengan al menos un determinado porcentaje de

representación en los datos. Supongamos que cambiamos el nombre de los países que tienen menos del 1% de presencia en los datos a `others`.

```
country_profiling=categ_analysis(data=data_country, input="country",

countries_high_rep=filter(country_profiling, perc_rows>0.01) %>% .$c

# Si no pertenece a countries_high_rep entonces lo asignamos a la ca
data_country$country_new=ifelse(data_country$country %in% countries_
```

Volvemos a chequear la probabilidad:

```
country_profiling_new=categ_analysis(data=data_country, input="count
country_profiling_new
```

```
##       country_new mean_target sum_target perc_target q_rows perc_
## 1  United Kingdom       0.178          8       0.096     45     0
## 2         Uruguay       0.175         11       0.133     63     0
## 3          Canada       0.158          3       0.036     19     0
## 4          France       0.142         41       0.494    288     0
## 5         Germany       0.100          3       0.036     30     0
## 6       Australia       0.098          4       0.048     41     0
## 7         Romania       0.091          1       0.012     11     0
## 8           Spain       0.091          1       0.012     11     0
## 9          Sweden       0.083          1       0.012     12     0
## 10    Netherlands       0.053          1       0.012     19     0
## 11          other       0.041          7       0.084    170     0
## 12         Turkey       0.030          2       0.024     67     0
## 13        Belgium       0.000          0       0.000     15     0
## 14         Brazil       0.000          0       0.000     13     0
## 15          China       0.000          0       0.000     65     0
## 16          Italy       0.000          0       0.000     10     0
## 17          Japan       0.000          0       0.000     18     0
## 18         Poland       0.000          0       0.000     13     0
```

Hemos reducido drásticamente la cantidad de países **-74% menos**- sólo reduciendo la cantidad de países al recategorizar al 1% menos representativo. Quedaron 18 de los 70 países.

La probabilidad de una variable objetivo se ha estabilizado un poco más en la

categoría "otra". Ahora cuando el modelo predictivo *vea* `Malasyia` **no asignará el 100% de la probabilidad, sino el 4.1%** (`mean_has_flu=0.041`).

Consejo sobre este último método

Tengan cuidado al aplicar esta técnica a ciegas. A veces, en una predicción objetivo **altamente desequilibrada** -por ejemplo, **detección de anomalías**- el comportamiento anormal está presente en menos del 1% de los casos.

```
# Replicar los datos
d_abnormal=data_country

# Simular comportamiento anormal en algunos países
d_abnormal$abnormal=ifelse(d_abnormal$country %in% c("Brazil", "Chil

# Análisis categórico
ab_analysis=categ_analysis(d_abnormal, input = "country", target = "

# Visualizar sólo los primeros 6 elementos
head(ab_analysis)
```

```
##               country mean_target sum_target perc_target q_rows p
## 1              Brazil           1         13       0.867     13
## 2               Chile           1          2       0.133      2
## 3           Argentina           0          0       0.000      9
## 4 Asia/Pacific Region           0          0       0.000      1
## 5           Australia           0          0       0.000     41
## 6             Austria           0          0       0.000      1
```

```
# Inspeccionar la distribución, sólo unos pocos pertenecen a la cate
freq(d_abnormal, "abnormal", plot = F)
```

```
##   abnormal frequency percentage cumulative_perc
## 1       no       895      98.35           98.35
## 2      yes        15       1.65          100.00
```

¿Cuántos valores anormales hay?

Sólo 15, y representan el 1,65% de los valores totales.

Comprobando la tabla devuelta por `categ_analysis`, podemos ver que este comportamiento *anormal* ocurre **sólo** en categorías con una participación realmente baja: `Brazil` que está presente en sólo 1,4% de los casos, y `Chile` con 0,2%.

En este caso, crear una categoría `other` basada en la distribución no es una buena idea.

Conclusión:

A pesar de que este es un ejemplo preparado, hay algunas técnicas de preparación de datos que pueden ser realmente útiles en términos de precisión, pero necesitan cierta supervisión. Esta supervisión puede ser con ayuda de algoritmos.

4.3.4.2 Caso 2: Reducción mediante agrupación automática

Este procedimiento utiliza la técnica de *clustering* o agrupamiento `kmeans` y la tabla devuelta por `categ_analysis` para crear grupos -clusters- que contienen categorías que muestran un comportamiento similar en términos de:

- `perc_rows`
- `perc_target`

La combinación de ambos nos llevará a encontrar grupos en base a la probabilidad y la representatividad.

Manos a la obra en R:

Definimos el parámetro `n_groups`, es el número de grupos deseados. El número es relativo a los datos y a la cantidad de categorías totales. Pero un número general estaría entre 3 y 10.

La función `auto_grouping` está incluida en `funModeling` >=1.6. Por favor noten que el parámetro `target` sólo funciona para variables no binarias.

Note: el parámetro `seed` es opcional, pero al asignarle un número siempre obtendrá los mismos resultados.

```r
# Reducir la cardinalidad
country_groups=auto_grouping(data = data_country, input = "country",
country_groups$df_equivalence
```

```
##                       country country_rec
## 1                   Australia     group_1
## 2                      Canada     group_1
## 3                     Germany     group_1
## 4                      France     group_2
## 5                       China     group_3
## 6                      Turkey     group_3
## 7          Asia/Pacific Region     group_4
## 8                     Austria     group_4
## 9                  Bangladesh     group_4
## 10    Bosnia and Herzegovina     group_4
## 11                  Cambodia     group_4
## 12                     Chile     group_4
## 13                Costa Rica     group_4
## 14                   Croatia     group_4
## 15                    Cyprus     group_4
## 16            Czech Republic     group_4
## 17        Dominican Republic     group_4
## 18                     Egypt     group_4
## 19                     Ghana     group_4
## 20                    Greece     group_4
## 21 Iran, Islamic Republic of     group_4
## 22                   Ireland     group_4
## 23               Isle of Man     group_4
## 24                    Latvia     group_4
## 25                 Lithuania     group_4
## 26                Luxembourg     group_4
## 27                     Malta     group_4
## 28      Moldova, Republic of     group_4
## 29                Montenegro     group_4
## 30                  Pakistan     group_4
## 31     Palestinian Territory     group_4
## 32                      Peru     group_4
## 33              Saudi Arabia     group_4
## 34                   Senegal     group_4
## 35                  Slovenia     group_4
## 36                    Taiwan     group_4
## 37                  Thailand     group_4
## 38                   Vietnam     group_4
## 39                   Belgium     group_5
## 40                    Brazil     group_5
```

```
## 41                  Bulgaria     group_5
## 42                Hong Kong      group_5
## 43                    Italy      group_5
## 44                   Poland      group_5
## 45                Singapore      group_5
## 46             South Africa      group_5
## 47                Argentina      group_6
## 48                   Israel      group_6
## 49                 Malaysia      group_6
## 50                   Mexico      group_6
## 51                 Portugal      group_6
## 52                  Romania      group_6
## 53                    Spain      group_6
## 54                   Sweden      group_6
## 55              Switzerland      group_6
## 56                    Japan      group_7
## 57              Netherlands      group_7
## 58           United Kingdom      group_8
## 59                  Uruguay      group_8
## 60                  Denmark      group_9
## 61                  Finland      group_9
## 62                 Honduras      group_9
## 63                Indonesia      group_9
## 64       Korea, Republic of      group_9
## 65                  Morocco      group_9
## 66              New Zealand      group_9
## 67                   Norway      group_9
## 68              Philippines      group_9
## 69       Russian Federation      group_9
## 70                  Ukraine      group_9
```

`auto_grouping` devuelve una lista que contiene 3 objetos:

- `df_equivalence`: data frame que contiene una tabla para encontrar las equivalencias entre datos viejos y nuevos.
- `fit_cluster`: modelo k-means que se utiliza para reducir la cardinalidad (los valores se escalan).
- `recateg_results`: data frame que contiene el análisis numérico de cada grupo con respecto a la variable objetivo. La primera columna ajusta su nombre a la variable de entrada. En este caso tenemos: `country_rec`. Cada grupo corresponde a una o varias categorías de la variable de entrada (como vimos en `df_equivalence`).

Exploremos cómo se comportan los nuevos grupos, esto es lo que *verá* el

modelo predictivo:

```
country_groups$recateg_results
```

```
##   country_rec mean_target sum_target perc_target q_rows perc_rows
## 1     group_8       0.176         19       0.229    108     0.119
## 2     group_6       0.156         10       0.120     64     0.070
## 3     group_2       0.142         41       0.494    288     0.316
## 4     group_1       0.111         10       0.120     90     0.099
## 5     group_7       0.027          1       0.012     37     0.041
## 6     group_3       0.015          2       0.024    132     0.145
## 7     group_4       0.000          0       0.000     49     0.054
## 8     group_5       0.000          0       0.000     85     0.093
## 9     group_9       0.000          0       0.000     57     0.063
```

La última tabla está ordenada por `mean_target`, por lo que podemos ver rápidamente los grupos según probabilidad máxima o mínima:

- `group_2` es el más común, está presente en el 31.6% de los casos y el valor de `mean_target` (probabilidad) es 14.2%.
- `group_8` tiene la probabilidad más alta (17.6%), seguido por `group_6` que tiene una probabilidad de 15.6% de tener un caso positivo (`has_flu="yes"`).
- `group_4`, `group_5` y `group_9` se ven iguales. Pueden ser un mismo grupo, dado que la probabiliad es 0 en todos los casos.
- `group_7` y `group_3` tienen 1 y 2 países con casos positivos. Podríamos considerar estos números como uno solo, juntándolos en un solo grupo, que eventualmente representará a los países con la probabilidad más baja.

Primero debemos agregar la columna de la nueva categoría al conjunto de datos original.

```
data_country_2=data_country %>% inner_join(country_groups$df_equival
```

Ahora hacemos las transformaciones adicionales reemplazando:

- `group_4`, `group_5` y `group_9` serán `low_likelihood`, (países sin casos positivos o con bajo porcentaje de la variable objetivo).

- group_7 y group_3 serán low_target_share.

```
data_country_2$country_rec=
  ifelse(data_country_2$country_rec %in%
           c("group_4", "group_5", "group_9"),
         "low_likelihood",
         data_country_2$country_rec
         )

data_country_2$country_rec=
  ifelse(data_country_2$country_rec %in%
           c("group_7", "group_3"),
         "low_target_share",
         data_country_2$country_rec
         )
```

Verificando la agrupación final (variable country_rec):

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.5.3
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:survival':
##
##     cluster
```

```
categ_analysis(data=data_country_2, input="country_rec", target = "h
```

```
##        country_rec mean_target sum_target perc_target q_rows perc
## 1          group_8       0.176         19       0.229    108
## 2          group_6       0.156         10       0.120     64
## 3          group_2       0.142         41       0.494    288
## 4          group_1       0.111         10       0.120     90
## 5 low_target_share       0.018          3       0.036    169
## 6   low_likelihood       0.000          0       0.000    191
```

Cada grupo parece tener un buen tamaño de muestra con respecto a la distribución de sum_target. Nuestra transformación dejó a low_likelihood con

una representación del 21% del total de casos, aún con 0 casos positivos (`sum_target`=0). Y `low_target_share` con 3 casos positivos, lo que representa el 3.6% de los casos positivos.

Todos los grupos parecen tener una buena representación. Esto se puede comprobar en la variable `perc_rows`. Todos los casos están por encima del 7%.

Intentar con un número menor de clusters puede ayudar a reducir un poco esta tarea manual. Esto fue sólo una demostración de cómo optimizar una variable que tiene muchas categorías diferentes.

4.3.5 Manejo de nuevas categorías cuando el modelo predictivo está en producción

Imaginemos que aparece un nuevo país, `new_country_hello_world`, los modelos predictivos fallarán ya que fueron entrenados con valores fijos. Una técnica es asignar un grupo que tenga `mean_target=0`.

Es similar al caso del último ejemplo. Pero la diferencia está en `group_5`: esta categoría encajaría mejor en un grupo de probabilidad media que en un valor completamente nuevo.

Después de un tiempo deberíamos reconstruir el modelo con todos los nuevos valores, de lo contrario estaríamos penalizando a `new_country_hello_world` si tiene una buena probabilidad.

En otras palabras:

¿Aparece una nueva categoría? Envíenla al grupo menos significativo. Después de un tiempo, vuelvan a analizar su impacto. ¿Tiene una probabilidad media o alta? Cámbienla al grupo más adecuado.

4.3.6 ¿Los modelos predictivos pueden manejan la alta cardinalidad? Parte 1

Sí, y no. Algunos modelos tratan mejor que otros este asunto de la alta cardinalidad. En algunos escenarios, esta preparación de datos puede no ser necesaria. Este libro trata de exponer este tema que, a veces, puede llevar a un mejor modelo.

Ahora, vamos a atravesar este tema construyendo dos modelos predictivos: Máquina de potenciación del gradiente - bastante robusta para muchas entradas de datos diferentes.

El primer modelo no tiene datos tratados, y el segundo ha sido tratado por la función en el paquete `funModeling`.

Estamos midiendo la precisión basándonos en el área ROC, que oscila entre 0.5 y 1; cuanto más alto sea el número, mejor será el modelo. Vamos a utilizar la validación cruzada para estar *seguros* del valor. La importancia de la validación cruzada de los resultados se trata en el capítulo Conociendo el error.

```
library(caret)
# Construir el primer modelo, sin reducir la cardinalidad.
fitControl <- trainControl(method = "cv",
                           number = 4,
                           classProbs = TRUE,
                           summaryFunction = twoClassSummary)


fit_gbm_1 <- train(has_flu ~ country,
                   data = data_country_2,
                   method = "gbm",
                   trControl = fitControl,
                   verbose = FALSE,
                   metric = "ROC")


# Obtener el mejor valor de ROC
roc=round(max(fit_gbm_1$results$ROC),2)
```

El área debajo de la curva ROC es (`roc`): 0.65.

Ahora hacemos el mismo modelo con los mismos parámetros, pero aplicando la preparación de datos que hicimos antes.

```
# Construir el segundo modelo, basándonos en la variable country_rec
fit_gbm_2 <- train(has_flu ~ country_rec,
                   data = data_country_2,
                   method = "gbm",
                   trControl = fitControl,
                   verbose = FALSE,
                   metric = "ROC")

# Obtener el nuevo mejor valor de ROC
new_roc=round(max(fit_gbm_2$results$ROC),2)
```

La nueva curva ROC es (`new_roc`): 0.71.

Luego calculamos el procentaje de mejora con respecto al primer valor de ROC:

Mejora: ~ 9.23%.

Nada mal, ¿no?

Un breve comentario sobre la última prueba:

Hemos utilizado uno de los modelos más robustos, **máquina de potenciación del gradiente**, y hemos aumentado el rendimiento. Si probamos otro modelo, por ejemplo regresión logística, que es más sensible a los datos sucios, obtendremos una mayor diferencia entre reducir y no reducir la cardinalidad. Esto se puede comprobar borrando el parámetro `verbose=FALSE` y cambiando `method=glm` (`glm` implica regresión logística).

En *lecturas adicionales* hay un punto de referencia de diferentes tratamientos para variables categóricas y cómo cada una aumenta o disminuye la precisión.

4.3.7 ¿Los modelos predictivos pueden manejan la alta cardinalidad? Parte 2

Revisemos cómo algunos modelos lidian con esto:

Árboles de decisión: Tienden a seleccionar variables con alta cardinalidad en la parte superior, dándoles más importancia que a otras, en función de la ganancia de información. En la práctica, es una prueba de que está sobreajustado. Este modelo es bueno para ver la diferencia entre reducir o no una variable de alta cardinalidad.

Random forest: al menos en la implementación de R, maneja sólo variables categóricas con por lo menos 52 categorías diferentes. Es muy probable que esta limitación sea para evitar el sobreajuste. Este punto, en conjunción con la naturaleza del algoritmo -crea muchos árboles-, reduce el efecto de un único árbol de decisión al elegir una variable de alta cardinalidad.

Gradient Boosting Machine y **Regresión logística**: convierten variables categóricas internas en variables *flag* o *dummy*. En el ejemplo que vimos sobre los países, implica la creación -interna- de 70 variables flag (así es como `caret` maneja la fórmula, si queremos mantener la variable original sin los dummies no tenemos que usar una fórmula).

Comprobemos el modelo que creamos antes:

```
# Verificar el primer modelo...
fit_gbm_1$finalModel
```

```
## A gradient boosted model with bernoulli loss function.
## 50 iterations were performed.
## There were 69 predictors of which 8 had non-zero influence.
```

Eso es: 69 variables de entrada representan a los países, pero las columnas flag fueron reportadas como no relevantes para la predicción.

Esto está relacionado con **Ingeniería de variables**. Además, está relacionado con Selección de las mejores variables. Es una práctica muy recomendable seleccionar primero las variables que contienen más información y luego crear el modelo predictivo.

Conclusión: la reducción de la cardinalidad reducirá la cantidad de variables en estos modelos.

4.3.8 Variable objetivo numérica o multinomial

Hasta ahora, el libro sólo cubrió casos donde la variable objetivo era una variable binaria. Está previsto que en el futuro abarque también variables objetivo numéricas y multi-valor.

Sin embargo, si leyeron hasta aquí, puede que quieran explorar por su cuenta teniendo en mente la misma idea. En las variables numéricas, por ejemplo la previsión de `page visits` en un sitio web, habrá ciertas categorías de la variable de entrada que estarán más relacionadas con un valor alto en las visitas, mientras que hay otras que están más correlacionadas con valores bajos.

Lo mismo ocurre con la variable de salida multinomial, habrá algunas categorías más relacionadas con ciertos valores. Por ejemplo, prediciendo el grado de epidemia: `high`, `mid` o `low` según la ciudad. Habrá algunas ciudades que se correlacionarán más con un alto nivel epidémico que otras.

4.3.9 ¿Qué beneficio "extra" obtuvimos con la agrupación?

Saber cómo las categorías fueron asignadas a los grupos nos brinda información que -en algunos casos- es bueno registrar. Las categorías que pertenezcan a un mismo grupo van a tener un comportamiento similar -en términos de representatividad y poder predictivo.

Si `Argentina` y `Chile` están en el `group_1`, entonces son iguales, y así es cómo las *verá* el modelo.

4.3.10 Representatividad o tamaño de muestra

Este concepto aplica al análisis de cualquier variable categórica, pero es un tema muy común en la ciencia de datos y las estadísticas: **tamaño de muestra**. ¿Cuántos datos necesitamos para ver el patrón *bien desarrollado*?

En una variable categórica: ¿Cuántos casos de la categoría "`X`" necesitamos para

confiar en la correlación entre el valor "x" y un valor objetivo? Esto es lo que hemos analizado.

En términos generales: cuanto más difícil sea predecir un evento, más casos vamos a necesitar...

Más adelante en este libro abarcaremos este tema desde otros puntos de vista refiriéndonos de vuelta a esta página.

4.3.11 Reflexiones finales

- Vimos dos casos para reducir la cardinalidad, al primero no le importa la variable objetivo, lo que puede ser peligroso en un modelo predictivo, mientras que al segundo sí. Crea una nueva variable basada en la afinidad -y representatividad- de cada categoría de entrada con la variable objetivo.

- Concepto clave: **representatividad** de cada categoría respecto a sí misma, y respecto al evento que se va a predecir. Un buen punto a explorar es analizarlo basándonos en pruebas estadísticas.

- Lo que se mencionó al principio con respecto a **destruir la información en la variable de entrada** implica que la agrupación resultante tiene las mismas proporciones entre grupos (en una variable binaria de entrada).

- ¿Siempre debemos reducir la cardinalidad? Depende, dos pruebas con un simple dato no son suficientes para extrapolar a todos los casos. Esperamos que sea un buen comienzo para que el lector empiece a hacer sus propias optimizaciones cuando lo considere relevante para el proyecto.

4.3.12 Lecturas adicionales

- El siguiente enlace contiene muchos resultados de precisión diferentes basados en distintos tratamientos para variables categóricas: Beyond One-Hot: an exploration of categorical variables.

Libro Vivo de Ciencia de Datos

4.4 Tratamiento de valores atípicos

4.4.1 ¿De qué se trata esto?

El concepto de valores extremos, al igual que otros temas en *machine learning*, no es un concepto exclusivo de esta área. Lo que hoy es un valor atípico puede que mañana no lo sea. Los límites entre el comportamiento normal y el anormal son difusos; por otro lado, pararse en los extremos es fácil.

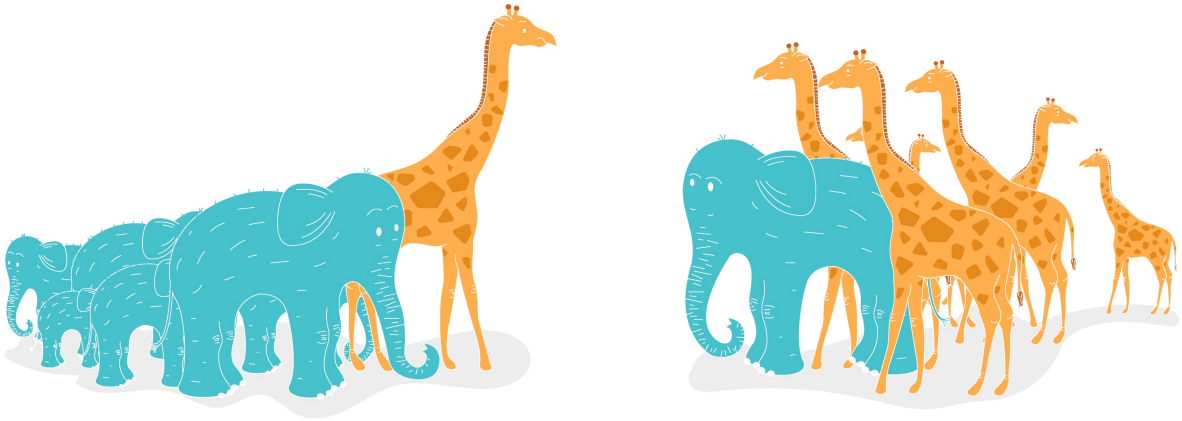


Imagen creada por: [Guillermo Mesyngier]

¿Qué vamos a repasar en este capítulo?

- ¿Qué es un valor atípico? Enfoques filosóficos y prácticos
- Valores atípicos por dimensionalidad y tipo de datos (numéricos o categóricos)
- Cómo detectar valores atípicos en R (bottom/top X%, Tukey y Hampel)
- Preparación de valores atípicos para análisis numérico en R
- Preparación de valores atípicos para modelado predictivo en R

4.4.2 La intuición detrás de los valores atípicos

Por ejemplo, consideren la siguiente distribución:

```
# Cargar ggplot2 para visualizar la distribución
library(ggplot2)

# Crear un conjunto de datos de muestra
set.seed(31415)
df_1=data.frame(var=round(10000*rbeta(1000,0.15,2.5)))

# Graficar
ggplot(df_1, aes(var, fill=var)) + geom_histogram(bins=20) + theme_l
```

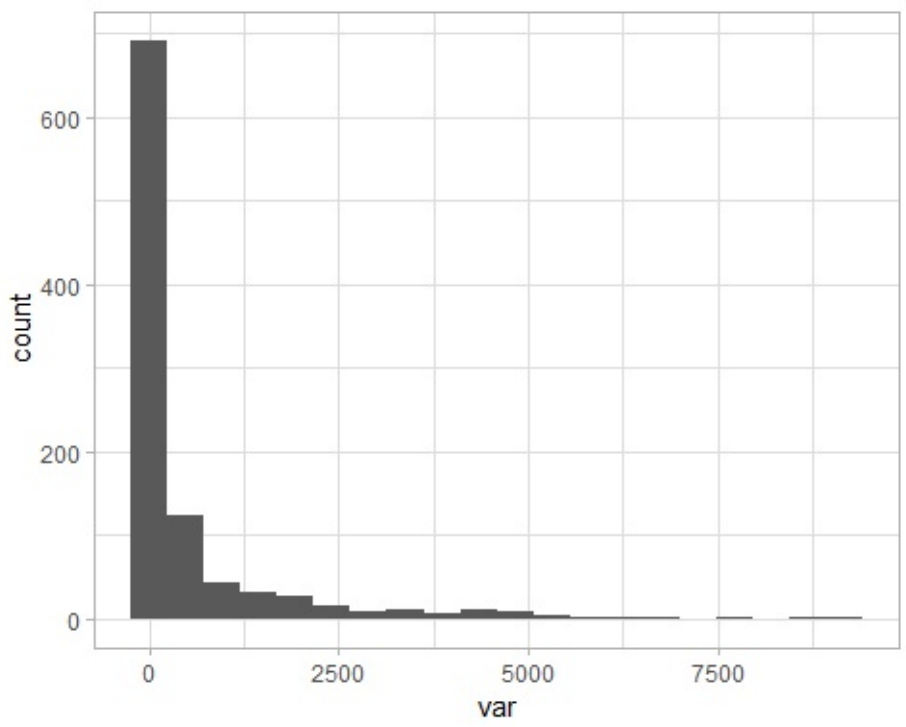


Figure 4.15: Distribución de muestra con cola larga

La variable está sesgada hacia la izquierda, mostrando algunos puntos atípicos a la derecha. Queremos *lidiar con ellos*. (😎). Entonces, surge la pregunta: *¿Dónde definimos el umbral de lo extremo?* Basándonos en la intuición, puede ser el 1% más alto, o podemos analizar cómo cambia el promedio si quitamos el 1% más alto.

Ambos casos podrían estar bien. De hecho, tomar otro número como el umbral (es decir, 2% o 0,1%), también puede ser correcto. Vamos a visualizarlos:

```
# Calcular los percentiles del 3% y 1% superior
percentile_var=quantile(df_1$var, c(0.98, 0.99, 0.999), na.rm = T)
df_p=data.frame(value=percentile_var, percentile=c("a_98th", "b_99th

# Graficar la misma distribución más los percentiles
ggplot(df_1, aes(var)) + geom_histogram(bins=20) + geom_vline(data=d
```

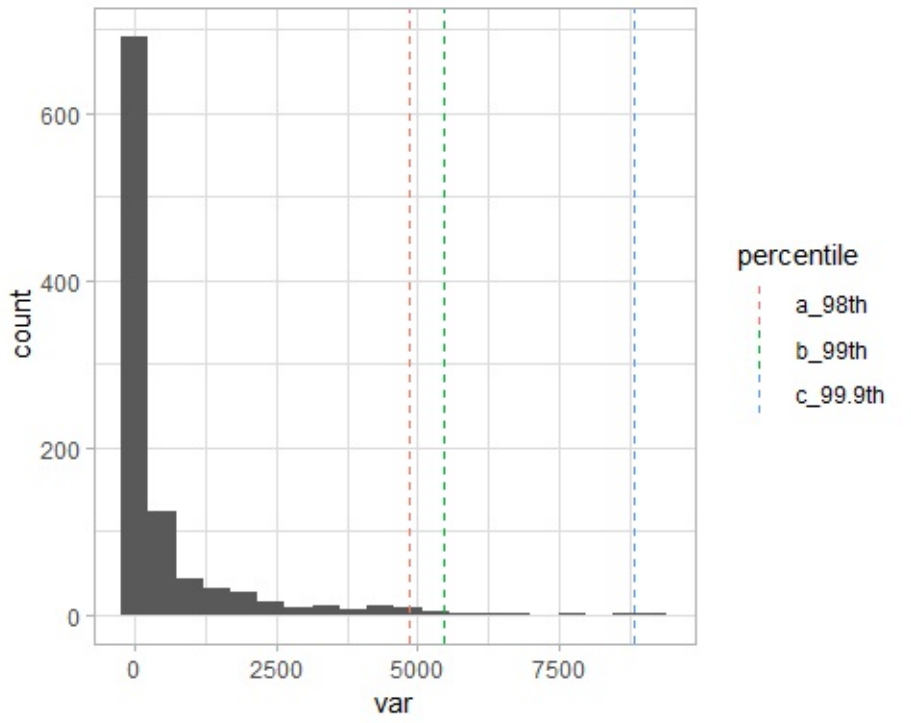


Figure 4.16: Diferentes umbrales para valores atípicos

Para entender los percentiles en mayor profundidad, por favor diríanse al capítulo Anexo 1: La magia de los percentiles.

Por ahora, seguiremos con el 1% superior (percentil 99) como el umbral para marcar todos los puntos que estén más allá como valores atípicos.

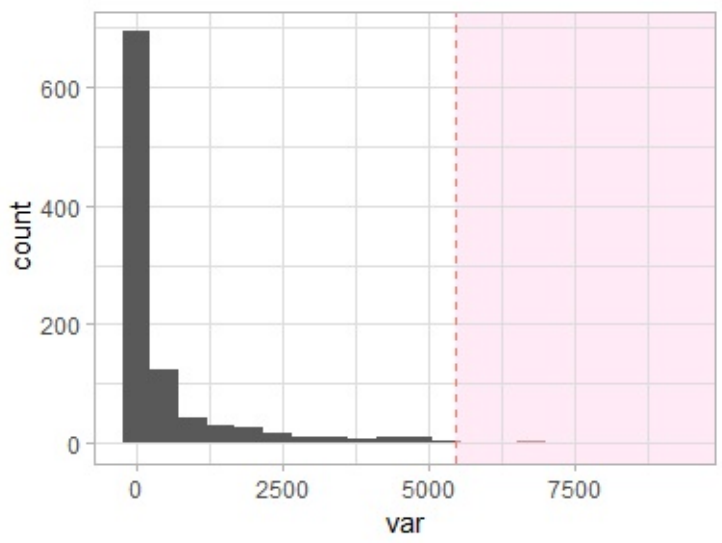


Figure 4.17: Marcando el 1 porciento superior como atípico

Aquí surge un elemento conceptual interesante: cuando definimos lo **anormal** (o una anomalía), el **concepto de normal emerge como su opuesto**.

Este comportamiento "normal" está representado en el área verde:

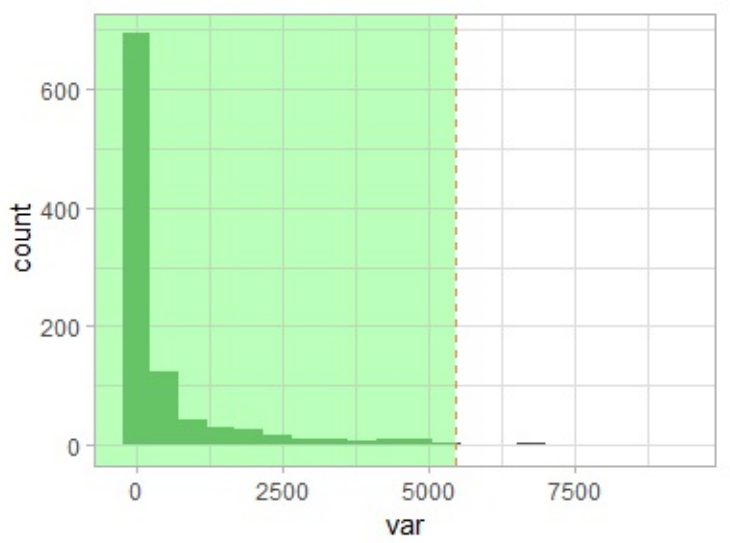


Figure 4.18: Mismo umbral, diferente perspectiva

Lo difícil es determinar dónde se separa lo normal de lo anormal. Hay varios enfoques para lidiar con esto. Vamos a repasar algunos de ellos.

4.4.3 ¿Cuál es el límite entre clima cálido y clima frío?

Hagamos esta sección más filosófica. Algunos buenos matemáticos también fueron filósofos, como es el caso de Pitágoras e Isaac Newton.

¿Dónde podemos poner el umbral para indicar que comienza el clima cálido o, a la inversa, que termina el clima frío?

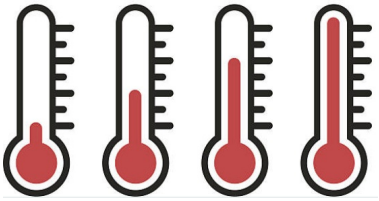


Figure 4.19: ¿Cuál es el punto de corte?

Cerca del Ecuador, una temperatura cerca de los 10ºC (50ºF) probablemente sea un valor extremadamente bajo; sin embargo, en la Antártida, ¡sería un día de playa! ☃

: *"¡Oh! ¡Pero eso sería tomar un ejemplo extremo con dos locaciones diferentes!"*

¡No hay problema! Hagamos zoom a una ciudad, como un fractal, el límite donde una empieza (y otra termina) no tendrá un único valor para determinar lo siguiente: *"Ok, el clima cálido empieza en los 25.5ºC (78ºF)."*

Es relativo.

Sin embargo, es bastante fácil pararse en los extremos, donde la incertidumbre disminuye a casi cero. Por ejemplo, cuando consideramos una temperatura de 60ºC (140ºF).

: *"Ok. Pero, ¿cómo se relacionan estos conceptos con machine learning?"*

Estamos exponiendo aquí la relatividad que existe al considerar una etiqueta (cálido/frío) como una variable numérica (temperatura). Esto puede ser considerado para cualquier otra variable numérica, como los ingresos económicos y las etiquetas "normal" y "anormal".

Entender los **valores extremos** es una de las primeras tareas en **análisis exploratorio de datos**. Entonces podremos ver cuáles son los valores normales. Esto se trata en el capítulo Análisis numérico, La voz de los números.

Existen varios métodos para marcar valores como valores atípicos. Así como podríamos analizar la temperatura, esta marca es *relativa* y todos los métodos pueden ser correctos. El método más rápido puede ser tratar el X% superior e inferior como valores atípicos.

Los métodos más robustos consideran las variables de distribución utilizando cuantiles (método de Tukey) o la dispersión de los valores a través de la desviación estándar (método de Hampel).

La definición de estos límites es una de las tareas más comunes en *machine learning*. *¿Por qué? ¿Cuándo?* Señalemos dos ejemplos:

- Ejemplo 1: Cuando desarrollamos un modelo predictivo que devuelve una probabilidad de llamar o no llamar a un determinado cliente, necesitamos configurar el umbral para asignar la etiqueta final: "¡sí, llamar!"/"no llamar". Hay más información sobre esto en el capítulo de Scoring de datos.

- Ejemplo 2: Otro ejemplo se da cuando necesitamos discretizar una variable numérica porque necesitamos que sea categórica. Los límites en cada segmento afectarán al resultado general. Hay más información sobre esto en la sección Discretizando variables numéricas

Volviendo al problema original (*¿Dónde termina el clima frío?*), no todas las preguntas necesitan una respuesta: algunas solamente nos ayudan a pensar.

4.4.4 El impacto de los valores atípicos

4.4.4.1 Construcción de modelos

Algunos modelos, como el bosque aleatorio y las máquinas de potenciación del gradiente, tienden a lidiar mejor con los valores atípicos; sin embargo, el "ruido" puede afectar los resultados de todos modos. El impacto de los valores atípicos en estos modelos es menor que en otros, como las regresiones lineales, las regresiones logísticas, los kmeans y los árboles de decisión.

Un aspecto que contribuye a la disminución del impacto es que ambos modelos crean *muchos* sub-modelos. Si cualquiera de los modelos toma un valor atípico como información, entonces otros sub-modelos probablemente no lo harán; por lo tanto, el error se cancela. El equilibrio yace en la pluralidad de voces.

4.4.4.2 Comunicar los resultados

Si debemos informar cuáles fueron las variables utilizadas en el modelo, terminaremos quitando los valores atípicos para no mostrar un histograma con una sola barra y/o un sesgo en el promedio.

Es mejor mostrar un número no sesgado que justificar que el modelo *podrá lidiar* con valores extremos.

4.4.4.3 Tipos de valores atípicos según el tipo de datos

- **Numéricos** : como los que vimos antes:

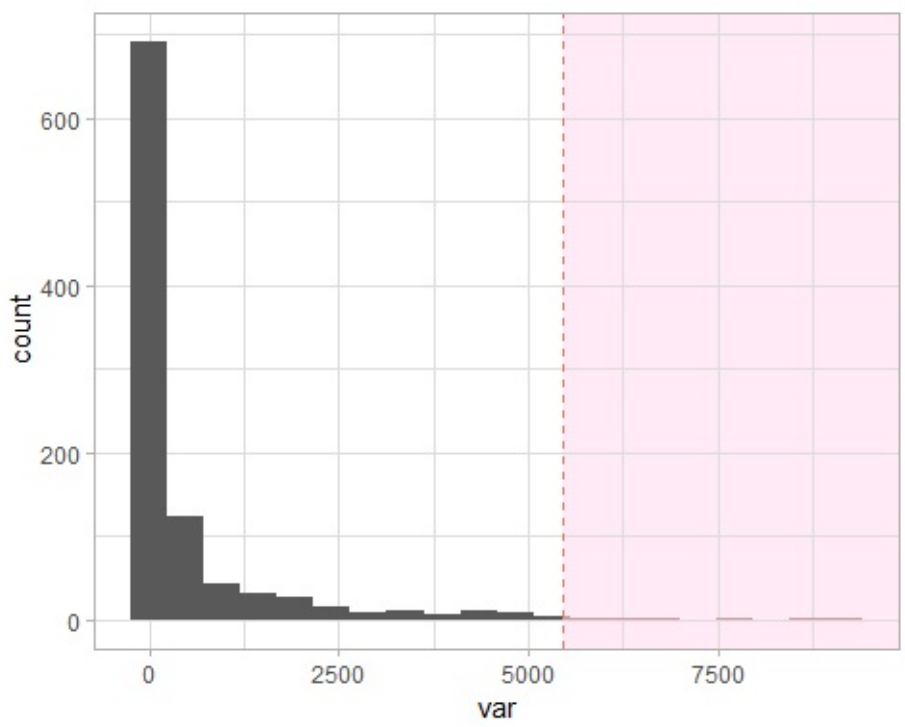


Figure 4.20: Variable numérica con valores atípicos

- **Categóricos** : Tener una variable en la que la dispersión de la categorías es bastante alta (alta cardinalidad): por ejemplo, código postal. Hay más información sobre cómo lidiar con valores atípicos en variables categóricas en el capítulo Variables de alta cardinalidad en estadística descriptiva.

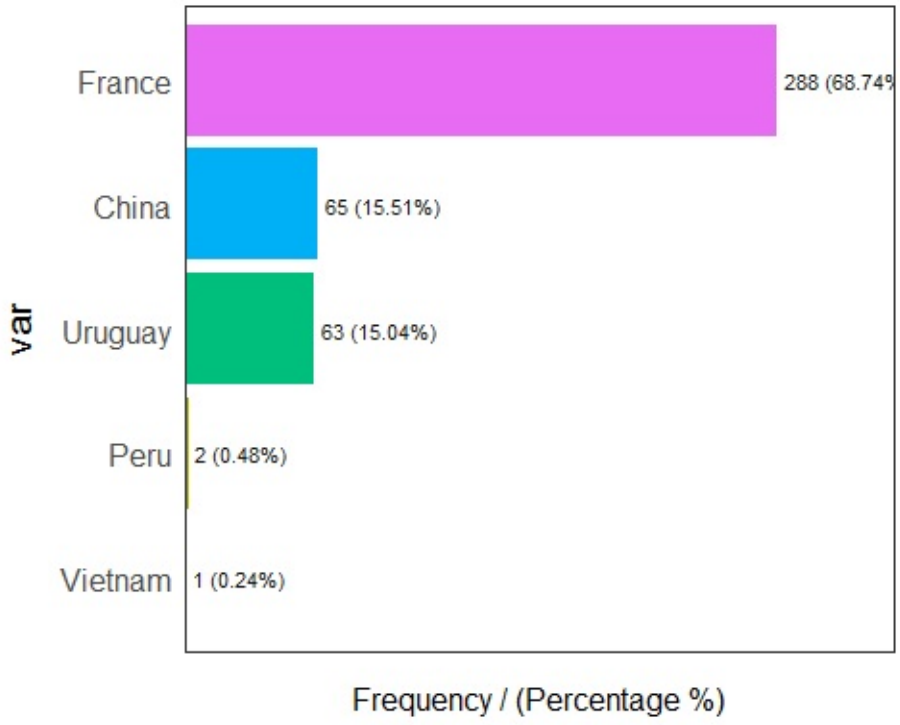


Figure 4.21: Variable categórica con valores atípicos

```
##       var frequency percentage cumulative_perc
## 1  France       288      68.74           68.74
## 2   China        65      15.51           84.25
## 3 Uruguay        63      15.04           99.29
## 4    Peru         2       0.48           99.77
## 5 Vietnam         1       0.24          100.00
```

`Peru` y `Vietnam` son valores atípicos en este ejemplo dado que su participación en los datos es inferior al 1%.

4.4.4.4 Tipos de valores atípicos según dimensionalidad

Hasta ahora, hemos observado valores atípicos unidimensionales y univariados. También podemos considerar dos o más variables en simultáneo.

Por ejemplo, tenemos el siguiente conjunto de datos, `df_hello_world`, con dos variables: `v1` y `v2`. Haciendo el mismo análisis que antes:

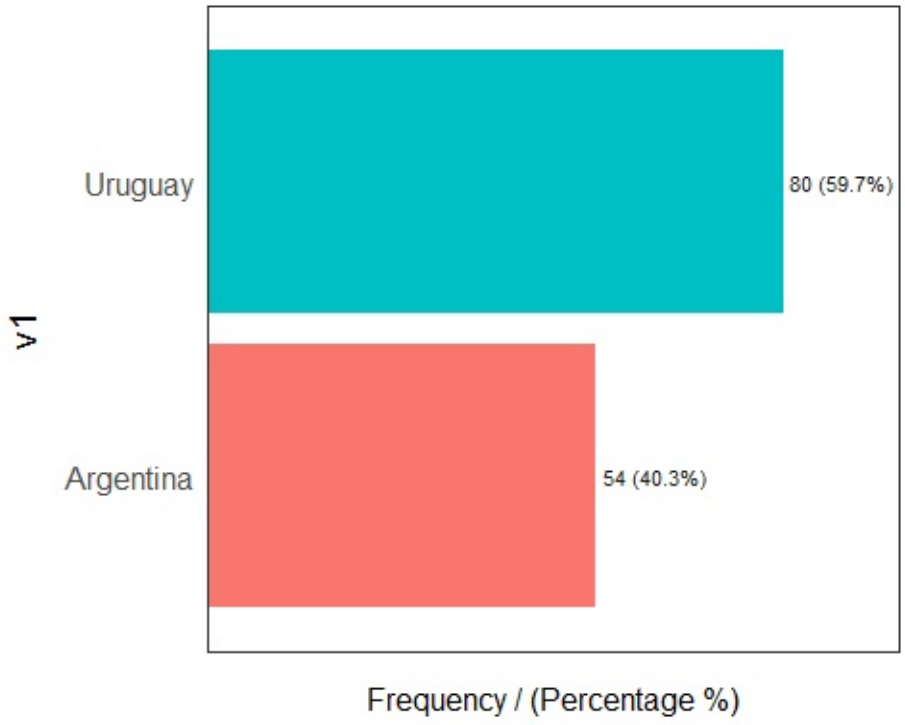


Figure 4.22: Valores atípicos según dimensionalidad

```
##          v1 frequency percentage cumulative_perc
## 1   Uruguay        80       59.7             59.7
## 2 Argentina        54       40.3            100.0
```

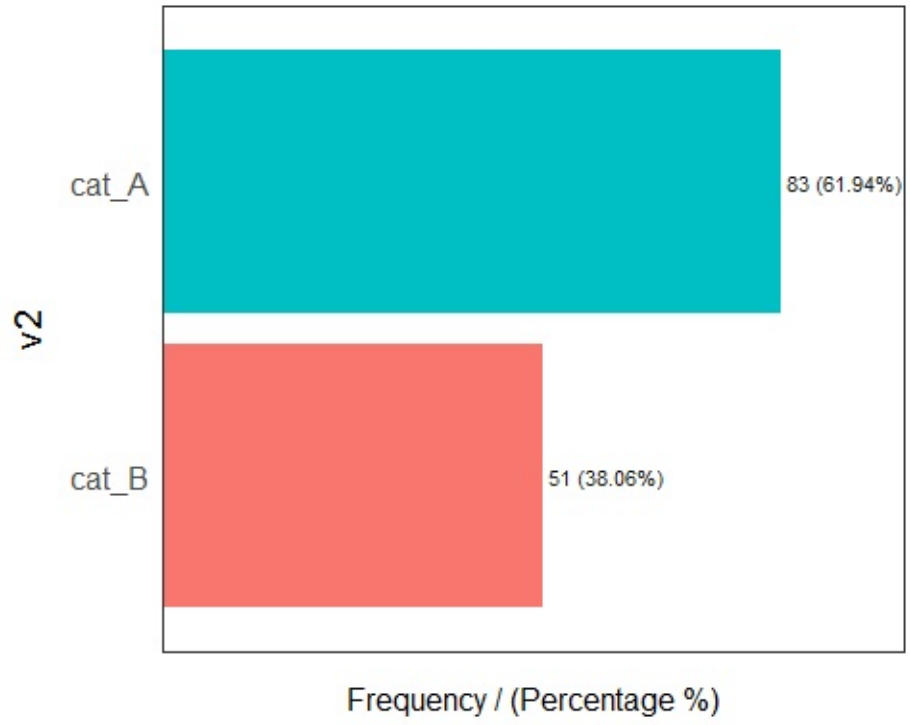


Figure 4.23: Valores atípicos según dimensionalidad

```
##      v2 frequency percentage cumulative_perc
## 1 cat_A        83      61.94           61.94
## 2 cat_B        51      38.06          100.00

## [1] "Variables processed: v1, v2"
```

Por ahora no hay valores atípicos, ¿correcto?

Ahora creamos una tabla de contingencia que nos diga la distribución de ambas variables, una contra la otra:

```
##           v2
## v1         cat_A cat_B
##   Argentina 39.55  0.75
##   Uruguay   22.39 37.31
```

¡Oh 😱! La combinación de `Argentina` y `cat_B` es *realmente baja* (0.75%) en comparación con los otros valores (menos del 1%), mientras que las otras intersecciones están por encima del 22%.

4.4.4.5 Algunas reflexiones...

Los últimos ejemplos muestran el *potencial* de los valores extremos o atípicos y están presentados como consideraciones que tenemos que tener en cuenta con un nuevo conjunto de datos.

Mencionamos **1%** como un posible umbral para marcar un valor como atípico. Este número podría ser 0.5% o 3%, dependiendo del caso.

Además, la presencia de este tipo de valores atípicos podría no traer problemas.

4.4.5 Cómo lidiar con valores atípicos en R

La función `prep_outliers` que viene incluida en el paquete funModeling puede ayudarnos con esta tarea. Puede manejar de una a 'N' variables en simultáneo (especificando el parámetro `input`).

El núcleo es el siguiente:

- Soporta tres métodos diferentes (parámetro `method`) para considerar un valor como un outlier: bottom_top, Tukey, y Hampel.
- Funciona en dos modos (parámetro `type`) al establecer un valor `NA` o al *frenar la variable* en un valor particular.
- Además de la explicación a continuación, `prep_outliers` es una función bien documentada: `help("prep_outliers")`.

4.4.6 Paso 1: Cómo detectar valores atípicos

Los siguientes métodos se implementan en la función `prep_outliers`. Obtienen diferentes resultados para que el usuario pueda seleccionar los que mejor se ajustan a sus necesidades.

4.4.6.0.1 Método de valores 'bottom' y 'top'

Esto considera valores atípicos tomando los valores del X% inferior y superior, basados en el percentil. Los puntos de corte más utilizados son 0.5%, 1%, 1.5%, 3%, entre otros.

Configurando el parámetro `top_percent` en `0.01` se tratarán todos los valores del 1% superior.

La misma lógica aplica a los valores más bajos: si se establece el parámetro `bottom_percent` en `0.01` se marcará como valores atípicos al 1% más bajo de todos los valores.

La función interna utilizada es `quantile`; si queremos marcar el 1% inferior y el superior, escribimos:

```
quantile(heart_disease$age, probs = c(0.01, 0.99), na.rm = T)
```

```
##  1% 99%
##  35  71
```

Todos los valores para aquellos casos que tengan menos de 35 años o más de 71 serán considerados atípicos.

Para leer más sobre percentiles, diríjanse al capítulo: Anexo 1: La magia de los percentiles.

4.4.6.0.2 Método de Tukey

Este método marca valores atípicos utilizando los valores cuartiles, Q1, Q2, y Q3, donde Q1 es esquivalente al percentil 25, Q2 al percentil 50 (también conocido como la mediana), y Q3 es el percentil 75.

El rango intercuartil (IQR por sus siglas en inglés) se calcula haciendo Q3 - Q1.

La fórmula:

- El umbral inferior es: Q1 - 3*IQR. Todos los valores que queden por debajo son considerados atípicos.

- El umbral superior es: Q1 + 3*IQR. Todos los valores que queden por encima son considerados atípicos.

El valor 3 es para detectar el límite "extremo". Este método viene del diagrama de caja, donde el multiplicador es 1.5 (no 3). Esto hace que muchos más valores sean marcados como atípicos, lo veremos en la siguiente imagen.

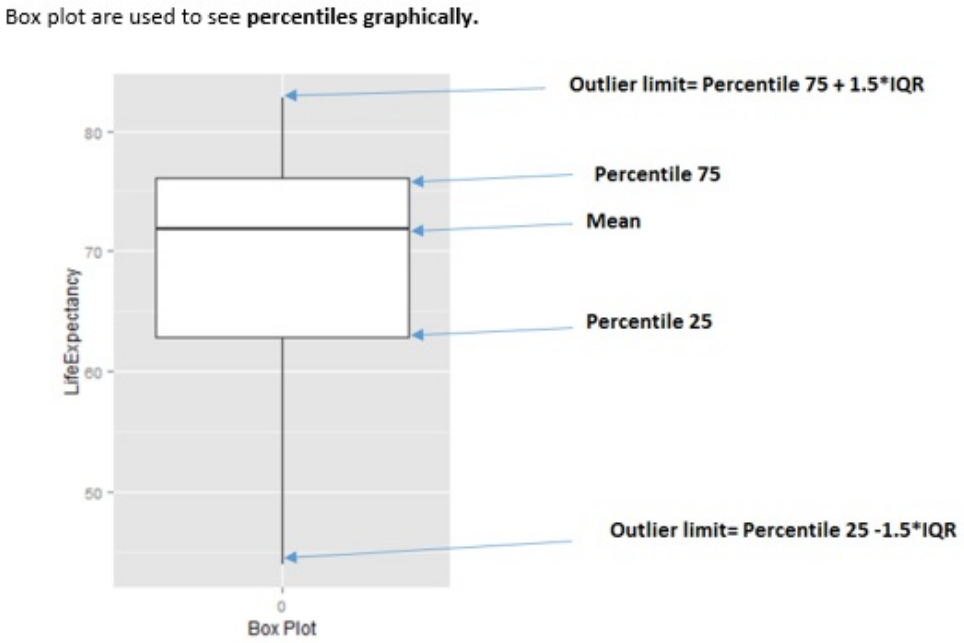


Figure 4.24: Cómo interpretar un diagrama de caja

Podemos acceder a la función interna utilizada en `prep_outliers` para calcular el límite de Tukey:

```
tukey_outlier(heart_disease$age)
```

```
## bottom_threshold    top_threshold
##                9              100
```

Devuelve un vector de dos valores; por lo tanto, tenemos el umbral inferior y el superior: todos los valores que estén por debajo de nueve y por encima de 100 serán considerados atípicos.

Encontrarán un ejemplo visual, simple y paso a paso en [tukey_outliers].

4.4.6.0.3 Método de Hampel

La fórmula:

- El umbral inferior es: `median_value - 3*mad_value`. Todos los valores que queden por debajo son considerados atípicos.
- El umbral superior es: `median_value + 3*mad_value`. Todos los valores que queden por encima son considerados atípicos.

Podemos acceder a la función interna utilizada en `prep_outliers` para calcular el límite de Hampel:

```
hampel_outlier(heart_disease$age)
```

```
## bottom_threshold    top_threshold
##          29.3132          82.6868
```

Devuelve un vector de dos valores; por lo tanto, tenemos el umbral inferior y el superior: todos los valores que estén por debajo de 29.31 y por encima de 82.68 serán considerados atípicos.

Tiene un parámetro llamado `k_mad_value`, y su valor por defecto es `3`. El valor `k_mad_value` puede ser modificado, pero no en la función `prep_outliers` por ahora.

Cuanto más alto sea el valor `k_mad_value`, más altos serán los límites de los umbrales.

```
hampel_outlier(heart_disease$age, k_mad_value = 6)
```

```
## bottom_threshold    top_threshold
##          2.6264         109.3736
```

4.4.7 Paso 2: ¿Qué hacemos con los valores atípicos?

Ya detectamos qué puntos son los atípicos. Ahora, la pregunta es *¿Qué hacemos con ellos?*

Hay dos escenarios posibles:

- Escenario 1: Preparar los valores atípicos para el análisis numérico
- Escenario 2: Preparar los valores atípicos para modelado predictivo

Hay un tercer escenario en el que no hacemos nada con los valores atípicos detectados. Simplemente los dejamos ser.

Proponemos recurrir a la función `prep_outliers` del paquete `funModeling` que nos dará una mano con esta tarea.

Más allá de la función en sí, lo importante aquí es el concepto subyacente y la posibilidad de desarrollar un método superador.

La función `prep_outliers` abarca estos dos escenarios con el parámetro `type`:

- `type = "set_na"`, para el escenario 1
- `type = "stop"`, para el escenario 2

4.4.7.1 Escenario 1: Preparar los valores atípicos para el análisis numérico

El análisis inicial:

En este caso, todos los valores atípicos son convertidos a `NA`, por lo que, al aplicar la mayoría de las funciones características (máx, mín, promedio, etc.) obtendremos un **valor menos sesgado**. Recuerden configurar el parámetro `na.rm=TRUE` en dichas funciones. De lo contrario, el resultado será `NA`.

Por ejemplo, consideremos la siguiente variable (la que vimos al principio con algunos valores atípicos):

```
# Para entender todas estas métricas, por favor diríjanse al capítul
profiling_num(df_1$var)
```

```
##   variable mean std_dev variation_coef p_01 p_05 p_25 p_50 p_75 p
## 1      var  548    1226            2.2    0    0    0   24  370 3
##   skewness kurtosis iqr     range_98    range_80
## 1      3.3       16 370 [0, 5467.33] [0, 1791.1]
```

Aquí podemos ver varios indicadores que nos dan algunas pistas. El desvío estándar `std_dev` es realmente alto comparado con el promedio `mean`, y eso se refleja en el coeficiente de variación `variation_coef`. Además, la curtosis es alta (16) y el valor de `p_99` es casi el doble que el de `p_95` (5767 vs. 3382).

Esta última tarea de mirar algunos números y visualizar la distribución de la variable es como imaginar una fotografía por lo que otra persona nos dice: convertimos la voz (que es una señal) en una imagen en nuestro cerebro. ...
=>

4.4.7.1.1 Utilizar `prep_outliers` para el análisis numérico

Debemos configurar `type="set_na"`. Esto implica que cada punto marcado como un valor atípico será convertido a `NA`.

Usaremos los tres métodos: Tukey, Hampel, y bottom/top X%.

Usando el método de Tukey:

```
df_1$var_tukey=prep_outliers(df_1$var, type = "set_na", method = "tu
```

Ahora verificamos cuántos valores `NA` había antes (la variable original) y después de la transformación basada en Tukey.

```
# Antes
df_status(df_1$var, print_results = F) %>% select(variable, q_na, p_
```

```
##   variable q_na p_na
## 1      var    0    0
```

```
# Después
df_status(df_1$var_tukey, print_results = F) %>% select(variable, q_
```

```
##   variable q_na p_na
## 1      var  120   12
```

Antes de la transformación, había 0 valores `NA`, mientras que después 120 valores (cerca del 12%) fueron marcados como atípicos de acuerdo a la prueba de Tukey y reemplazados por `NA`.

Podemos comparar el antes y el después:

```
profiling_num(df_1, print_results = F) %>% select(variable, mean, st
```

```
##    variable mean std_dev variation_coef kurtosis     range_98
## 1       var  548    1226            2.2     15.6 [0, 5467.33]
## 2 var_tukey  163     307            1.9      8.4 [0, 1358.46]
```

El promedio disminuyó casi en una tercera parte, y todas las demás métricas también disminuyeron.

Método de Hampel:

Veamos qué pasa con el método de Hampel (`method="hampel"`):

```
df_1$var_hampel=prep_outliers(df_1$var, type = "set_na", method="ham
```

Verificando...

```
df_status(df_1, print_results = F) %>% select(variable, q_na, p_na)
```

```
##     variable q_na p_na
## 1        var    0    0
## 2  var_tukey  120   12
## 3 var_hampel  364   36
```

Este último método es mucho más severo al identificar valores atípicos, marcando el 36% de los valores como atípicos. Es probable que esto se deba a que la variable está *bastante* sesgada hacia la izquierda.

Método del 'bottom' y 'top' X%

Por último, podemos probar el método más fácil: quitar el 2% superior.

```
df_1$var_top2=prep_outliers(df_1$var, type = "set_na", method="botto
```

Por favor noten que el valor de 2% fue asignado arbitrariamente. También pueden intentar con otros valores, como 3% o 0.5%.

¡Es ahora de comparar todos los métodos!

4.4.7.1.2 Uniendo todo lo que vimos

Tomaremos algunos indicadores para realizar la comparación cuantitativa.

```
df_status(df_1, print_results = F) %>% select(variable, q_na, p_na)
```

```
##     variable q_na p_na
## 1        var    0    0
## 2  var_tukey  120   12
## 3 var_hampel  364   36
## 4   var_top2   20    2
```

```
prof_num=profiling_num(df_1, print_results = F) %>% select(variable,
prof_num
```

```
##     variable mean std_dev variation_coef kurtosis     range_98
## 1        var  548    1226            2.2     15.6 [0, 5467.33]
## 2  var_tukey  163     307            1.9      8.4 [0, 1358.46]
```

```
## 3 var_hampel   17      31            1.8      6.0   [0, 118.3]
## 4   var_top2  432     908            2.1     10.9 [0, 4364.29]
```

Graficar

```
# Primero debemos convertir el conjunto de datos a formato ancho
df_1_m=reshape2::melt(df_1)
plotar(df_1_m,  target= "variable", input = "value", plot_type = "bo
```

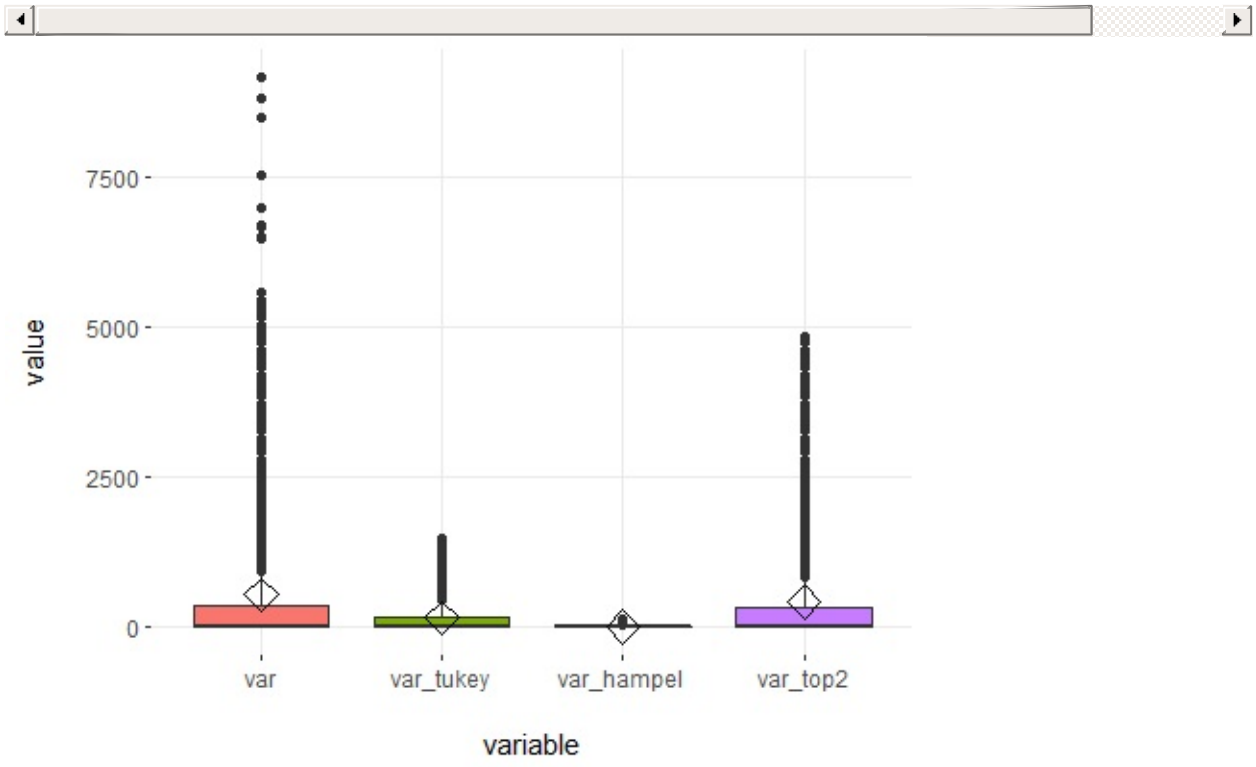


Figure 4.25: Comparación de métodos para identificar valores atípicos

Al seleccionar el bottom/top X%, siempre tendremos algunos valores que cumplan con esa condición, mientras que en los otros dos métodos puede que esto no suceda.

4.4.7.1.3 Conclusiones sobre el manejo de valores atípicos en el análisis numérico

La idea es modificar los valores atípicos lo menos posible (por ejemplo, si estamos interesados solamente en describir el comportamiento general).

Para lograr eso -a la hora de crear un informe ad hoc, por ejemplo- podemos usar el promedio. Podríamos elegir el método del 2% superior porque solo afecta al 2% de todos los valores y provoca una disminución drástica en el promedio: de 548 a 432, o **21% menos**.

> "Modificar o no modificar el conjunto de datos, esa es la cuestión." William Shakespeare como científico de datos.

El método de Hampel modificó demasiado el promedio, ¡de 584 a 17! Eso fue tomando el valor *estándar* de este método, que es 3-MAD (un desvío estándar un tanto robusto).

Por favor tengan en cuenta que esta demostración no significa que Hampel o Tukey sean una mala elección. De hecho, son métodos más robustos porque el umbral puede ser más alto que el valor actual; de hecho, ningun valor es tratado como atípico.

En el otro extremo, podemos considerar, por ejemplo, la variable `age` de los datos `heart_disease`. Analicemos sus valores atípicos:

```
# Obtener el umbral de valores atípicos
tukey_outlier(heart_disease$age)
```

```
## bottom_threshold    top_threshold
##                9              100
```

```
# Obtener los valores mínimos y máximos
min(heart_disease$age)
```

```
## [1] 29
```

```
max(heart_disease$age)
```

```
## [1] 77
```

- El umbral inferior es 9, y el valor mínimo es 29.
- El umbral superior es 100, y el valor máximo es 77.

Ergo: la variable `age` no tiene valores atípicos.

Si hubiéramos utilizado el método bottom/top X%, entonces los datos dentro de esos porcentajes hubieran sido detectados como valores atípicos.

Todos los ejemplos que vimos hasta ahora tomaron una sola variable a la vez; no obstante, `prep_outliers` puede manejar varias en simultáneo usando el parámetro `input` como veremos en la siguiente sección. Todo lo que vimos hasta aquí será equivalente, excepto lo que hacemos una vez que detectamos los valores atípicos, es decir, el método de imputación.

4.4.7.2 Escenario 2: Preparar los valores atípicos para modelado predictivo

El caso anterior da como resultado que los valores atípicos observados se convierten a valores `NA`. Esto es un gran problema si estamos construyendo un modelo de machine learning, ya que muchos de ellos no funcionan con valores `NA`. Hay más información sobre el manejo de datos faltantes en el capítulo Datos faltantes.

Para lidiar con valores atípicos y poder usar un modelo predictivo, una buena idea es configurar el parámetro `type='stop'`, para que todos los valores marcados como atípicos sean convertidos al valor del umbral.

Algunas cosas a tener en cuenta:

Traten de pensar en el tratamiento (y creación) de las variables como si se lo estuvieran explicando al modelo. Al frenar las variables en un determinado valor, 1% por ejemplo, le estamos diciendo al modelo: *Hey, modelo, por favor considera todos los valores extremos como si estuvieran en el percentil 99, dado que este valor ya es lo suficientemente alto. Gracias.*

Algunos modelos predictivos son más **tolerantes al ruido** que otros. Podemos ayudarlos tratando algunos de los valores atípicos. En la práctica, pre-procesar datos tratando los atípicos tiende a producir resultados más precisos cuando estamos en presencia de datos nunca vistos.

4.4.7.3 Imputar valores atípicos para modelado predictivo

Primero, creamos un conjunto de datos con algunos valores atípicos. Ahora el ejemplo tiene dos variables.

```
# Crear data frame con valores atípicos

# Desactivar la notación científica
options(scipen=999)
# Configurar seed para que tenga un ejemplo reproducible
set.seed(10)
# Crear las variables
df_2=data.frame(var1=rchisq(1000,df = 1), var2=rnorm(1000))
# Forzar los valores atípicos
df_2=rbind(df_2, 135, rep(400, 30), 245, 300, 303, 200)
```

Lidiar con los valores atípicos en ambas variables (`var1` y `var2`) usando el método de Tukey:

```
df_2_tukey=prep_outliers(data = df_2, input = c("var1", "var2"), typ
```

Verificar algunas métricas antes y después de la imputación:

```
profiling_num(df_2, print_results = F) %>% select(variable, mean, st
```

```
##   variable mean std_dev variation_coef
## 1     var1  2.6      21            8.3
## 2     var2  1.6      21           13.6
```

```
profiling_num(df_2_tukey, print_results = F) %>% select(variable, me
```

```
##   variable  mean std_dev variation_coef
## 1     var1 0.997     1.3            1.3
## 2     var2 0.018     1.0           57.5
```

Tukey funcionó perfectamente esta vez, exponiendo un promedio más preciso

para ambas variables: 1 para `var1` y 0 para `var2`.

Observen que esta vez no hay ni un valor `NA`. Lo que hizo la función esta vez fue **frenar la variable** en los valores umbral. Ahora, los valores mínimos y máximos serán los mismos que informó el método de Tukey.

Verificar el umbral para `var1`:

```
tukey_outlier(df_2$var1)
```

```
## bottom_threshold    top_threshold
##             -3.8              5.3
```

Ahora verificamos los valores min/max antes de la transformación:

```
# Antes:
min(df_2$var1)
```

```
## [1] 0.0000031
```

```
max(df_2$var1)
```

```
## [1] 400
```

y después de la transformación...

```
# Después
min(df_2_tukey$var1)
```

```
## [1] 0.0000031
```

```
max(df_2_tukey$var1)
```

```
## [1] 5.3
```

El mínimo sigue siendo el mismo (0.0000031), pero el máximo fue ajustado al valor de Tukey de ~5.3.

Los cinco valores más altos antes de la pre-partición eran:

```
# Antes
tail(df_2$var1[order(df_2$var1)], 5)
```

```
## [1] 200 245 300 303 400
```

pero después...

```
# Después:
tail(df_2_tukey$var1[order(df_2_tukey$var1)], 5)
```

```
## [1] 5.3 5.3 5.3 5.3 5.3
```

Y verificamos que no haya ningún `NA`:

```
df_status(df_2_tukey, print_results = F) %>% select(variable, q_na,
```

```
##   variable q_na p_na
## 1     var1    0    0
## 2     var2    0    0
```

Bastante claro, ¿no?

Ahora repliquemos el ejemplo que vimos en la última sección con una sola variable para comparar los tres métodos.

```
df_2$tukey_var2=prep_outliers(data=df_2$var2, type='stop', method =
df_2$hampel_var2=prep_outliers(data=df_2$var2, type='stop', method =
df_2$bot_top_var2=prep_outliers(data=df_2$var2, type='stop', method
```

4.4.7.3.1 Uniendo todo lo que vimos

```
# Excluir var1
df_2_b=select(df_2, -var1)

# Análisis numérico
profiling_num(df_2_b, print_results = F) %>% select(variable, mean,
```

```
##       variable   mean std_dev variation_coef kurtosis     range_9
## 1         var2 1.5649   21.36             14    223.8 [-2.32, 2.4
## 2   tukey_var2 0.0178    1.02             58      4.6 [-2.32, 2.4
## 3  hampel_var2 0.0083    0.98            118      3.2 [-2.32, 2.4
## 4 bot_top_var2 0.0083    0.97            116      2.9 [-2.32, 2.4
```

Los tres métodos muestran resultados muy similares con estos datos.

Graficar

```
# Primero debemos convertir el conjunto de datos a formato ancho
df_2_m=reshape2::melt(df_2_b) %>% filter(value<100)
plotar(df_2_m,  target= "variable", input = "value", plot_type = "bo
```

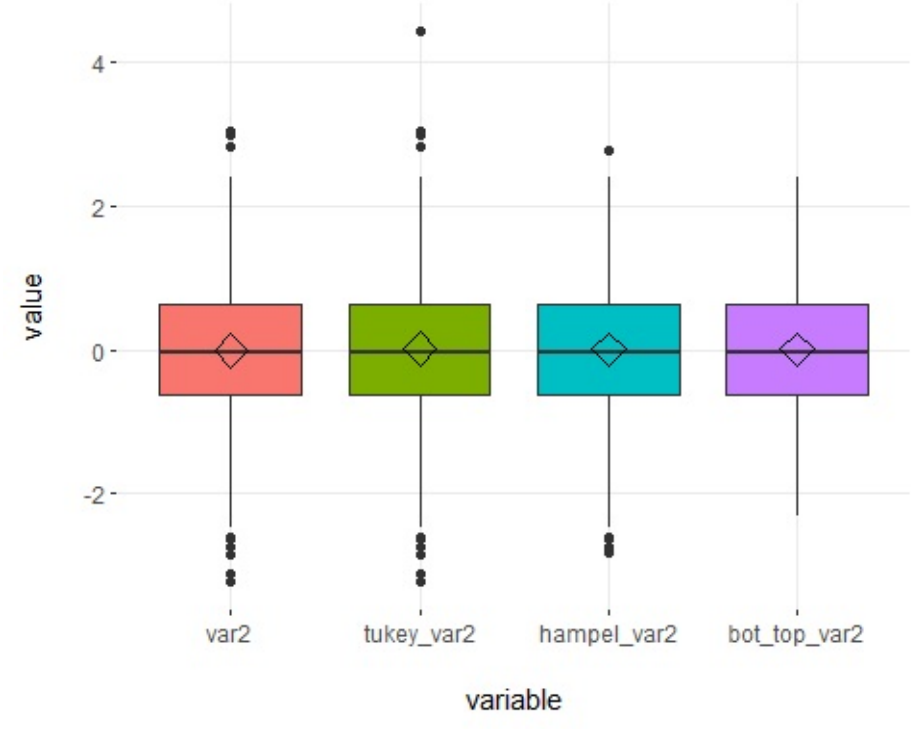


Figure 4.26: Comparación de métodos para identificar valores atípicos

Importante: Los dos puntos que están por encima del valor 100 (sólo para `var1`) fueron excluidos, de lo contrario, hubiera sido imposible notar la diferencia entre los métodos.

4.4.8 Reflexiones finales

Hemos abordado el tema de los valores atípicos tanto desde una perspectiva filosófica como técnica, invitando al lector a mejorar sus habilidades de pensamiento crítico a la hora de definir los límites (umbrales). Es fácil pararse en los extremos, pero encontrar el equilibrio es una tarea difícil.

En términos técnicos, cubrimos tres métodos con diferentes bases para identificar valores atípicos:

- **Bottom/Top X%**: Este método siempre marcará valores como atípicos dado que en todas las variables hay un X% inferior y superior.
- **Tukey**: Se basa en el clásico boxplot, que usa cuartiles.
- **Hampel**: Es bastante restrictivo si no modificamos el parámetro por defecto. Se basa en la mediana y el valor del MAD (similar al desvío estándar pero menos sensible a los valores atípicos).

Una vez que identificamos los valores atípicos, el siguiente paso es decidir qué hacer con ellos. En algunos casos no es necesario hacer ningún tratamiento. En conjuntos de datos muy pequeños, podemos identificarlos a simple vista.

La regla de: ***"Sólo modificar lo que es necesario"*** (que también puede aplicar a la relación entre el *ser humano* y la naturaleza), nos dice que no tratemos o excluyamos ciegamente todos los valores atípicos extremos. **Con cada acción que hicimos, introducimos algún sesgo**. Por eso es tan importante saber cuáles son las implicaciones de cada método. Si es una buena decisión o no depende de la naturaleza de los datos que estamos analizando.

En **modelado predictivo**, aquellos que tienen algún tipo de técnica de remuestreo interno, o crean *varios modelos pequeños* para llegar a una predicción final, son más estables con los valores extremos. Hay más información sobre remuestreo y error en el capítulo Conociendo el error.

En algunos casos, cuando el modelo predictivo está **ejecutándose en producción**, es recomendable reportar o considerar la preparación de cualquier valor extremo nuevo, es decir, un valor que no estaba presente durante la construcción del modelo. Hay más información sobre este tema, pero con una variable categórica, en Variables de alta cardinalidad en modelado predictivo, sección: *Manejo de nuevas categorías cuando el modelo predictivo está en producción*.

Una buena prueba para que haga el lector es tomar un conjunto de datos, tratar los valores atípicos, y luego comparar algunas métricas de desempeño como Kappa, ROC, Precisión (*Accuracy*), etc.; **¿La preparación de los datos mejoró alguna de ellas?** O, en los reportes, ver cuánto cambia el promedio. Incluso graficando, ¿ahora el gráfico nos dice algo? De esta manera, el lector creará nuevo conocimiento basándose en su experiencia.☺.

4.5 Datos faltantes: Análisis, manejo e imputación

4.5.1 ¿De qué se trata esto?

El análisis de los valores faltantes es la estimación del vacío mismo. Los valores faltantes presentan un obstáculo a la hora de crear modelos predictivos, análisis de clusters, reportes, etc.

En este capítulo, ahondaremos en el concepto y tratamiento de valores nulos. Realizaremos análisis utilizando diferentes enfoques e interpretaremos los distintos resultados.

Si todo sale bien, después de estudiar el capítulo entero, el lector podrá entender conceptos clave del manejo de valores faltantes y podrá tomar mejores abordajes que los que proponemos aquí.

¿Qué vamos a repasar en este capítulo?

- ¿Qué es un valor nulo, conceptualmente?
- Cuándo excluir filas o columnas.
- Análisis numérico de valores faltantes.
- Transformación e imputación de variables numéricas y categóricas.
- Imputar valores: desde enfoques simples a algunos más complejos.

Ejemplificaremos estos temas con un enfoque práctico en R. Este código busca ser lo suficientemente genérico para que lo puedan aplicar en sus proyectos .

4.5.2 Cuando el valor nulo representa información

Los valores vacíos también aparecen como "NULL" en bases de datos, `NA` en R, o simplemente el string "empty" en programas de hojas de cálculo. También pueden estar representados con algún número, como: `0`, `-1` o `-999`.

Por ejemplo, imaginen una agencia de viajes que une dos tablas, una de personas y otra de países. El resultado muestra la cantidad de viajes por persona:

```
##    person South_Africa Brazil Costa_Rica
## 1  Fotero            1      5          5
## 2   Herno           NA     NA         NA
## 3 Mamarul           34     40         NA
```

En este resultado, `Mamarul` viajó a `South Africa` `34` veces.

¿Qué representa el valor `NA` (o NULL)?

En este caso, NA debería ser reemplazado por 0, indicando cero viajes en esa intersección persona-país. Después de la conversación, la tabla está lista para

usar.

Ejemplo: Reemplazar todos los valores NA por 0

```
# Hacer una copia
df_travel_2=df_travel

# Reemplazar todos los valores NA con 0
df_travel_2[is.na(df_travel_2)]=0
df_travel_2
```

```
##    person South_Africa Brazil Costa_Rica
## 1  Fotero            1      5          5
## 2   Herno            0      0          0
## 3 Mamarul           34     40          0
```

El último ejemplo transforma **todos** los valores `NA` en `0`. No obstante, en otros escenarios, esta transformación podría no aplicar para todas las columnas.

Ejemplo: Reemplazar los valores NA por 0 sólo en ciertas columnas

Probablemente el escenario más común sea reemplazar NA por algún valor -cero en este caso- sólo en algunas columnas. Definimos un vector que contiene todas las variables a reemplazar y luego aplicamos la función `mutate_at` del paquete `dplyr`.

```
library(dplyr) # vers 0.7.1

# Reemplazar valores NA con 0 solo en las columnas seleccionadas
vars_to_replace=c("Brazil", "Costa_Rica")

df_travel_3=df_travel %>% mutate_at(.vars=vars_to_replace, .funs = f
```

```
## Warning: funs() is soft deprecated as of dplyr 0.8.0
## please use list() instead
##
##   # Before:
##   funs(name = f(.))
##
##   # After:
##   list(name = ~ f(.))
```

```
## This warning is displayed once per session.
```

```
df_travel_3
```

```
##    person South_Africa Brazil Costa_Rica
## 1  Fotero            1      5          5
## 2   Herno           NA      0          0
## 3 Mamarul           34     40          0
```

Tengan a mano la última función ya que es muy común enfrentarnos a la situación de aplicar una función especificada a un subconjunto de variables y volver a incorporar las variables transformadas y no transformadas al mismo conjunto de datos.

Vamos a un ejemplo más complejo.

4.5.3 Cuando el valor nulo es un valor nulo

En otras ocasiones, tener un valor nulo es correcto, está expresando la ausencia de algo. Debemos tratarlos para poder usar la tabla. Muchos modelos predictivos no pueden manejar tablas de entrada con valores faltantes.

En algunos casos, una variable es medida *después* de un período de tiempo, por lo que tenemos datos a partir de este momento y NA en las instancias previas.

A veces hay casos aleatorios, como una máquina que falla al recoletar datos o un usuario que se olvidó de completar algún campo en un formulario, entre otros.

Aquí aparece una pregunta importante: *¿Qué hacemos?* 😱

Las siguientes recomendaciones son simplemente eso, recomendaciones. Pueden probar diferentes enfoques para descubrir cuál es la mejor estrategia para los datos que están analizando. **No existe un "talle único y universal" en esto**.

4.5.4 Excluir la fila entera

Si al menos una columna tiene un valor `NA`, excluyan la fila.

Es un método fácil y rápido, ¿no? Lo recomendamos cuando la cantidad de filas con valores faltantes sea *baja*. Pero, ¿cuán baja es baja? Eso depende de ustedes. Diez casos en 1,000 filas *pueden no* tener un gran impacto, a menos que esos 10 casos estén vinculados con la predicción de una anomalía; en esta instancia, representan información. Señalamos este tema en Caso 1: Reducción mediante la recategorización de valores menos representativos.

Ejemplo en R:

Inspeccionemos el conjunto de datos `heart_disease` con la función `df_status`, dado que uno de sus objetivos principales es ayudarnos con este tipo de decisiones.

```
library(dplyr)
library(funModeling)
df_status(heart_disease, print_results = F) %>% select(variable, q_n
```

```
##                   variable q_na p_na
## 1       num_vessels_flour    4 1.32
## 2                    thal    2 0.66
## 3                     age    0 0.00
## 4                  gender    0 0.00
## 5              chest_pain    0 0.00
## 6  resting_blood_pressure    0 0.00
## 7       serum_cholestoral    0 0.00
## 8     fasting_blood_sugar    0 0.00
## 9          resting_electro    0 0.00
## 10          max_heart_rate    0 0.00
## 11             exer_angina    0 0.00
## 12                 oldpeak    0 0.00
## 13                   slope    0 0.00
## 14 heart_disease_severity    0 0.00
## 15           exter_angina    0 0.00
## 16      has_heart_disease    0 0.00
```

`q_na` indica la cantidad de valores `NA` y `p_na` es el porcentaje. Pueden encontrar toda la información sobre la función `df_status` en el capítulo Análisis

numérico, La voz de los números.

Dos variables tienen 4 y 2 filas con valores NA, entonces excluimos estas filas:

```
# na.omit devuelve el mismo data frame habiendo excluido todas las f
heart_disease_clean=na.omit(heart_disease)

# número de filas antes de la exclusión:
nrow(heart_disease)
```

```
## [1] 303
```

```
# número de filas después de la exclusión:
nrow(heart_disease_clean)
```

```
## [1] 297
```

Después de la exclusión, seis filas de 303 fueron eliminadas. Este enfoque parece adecuado para este conjunto de datos.

Sin embargo, existen otros escenarios en los que casi todos los casos son valores vacíos, por lo que ¡esta exclusión eliminaría todo el conjunto de datos!

4.5.5 Excluir la columna

En una operación similar al último caso, excluimos la columna. Si aplicamos el mismo razonamiento y la eliminación es sólo de unas *pocas* columnas y las restantes proveen un resultado final confiable, entonces es aceptable.

Ejemplo en R:

Estas exclusiones se pueden manejar fácilmente con la función `df_status`. El siguiente código va a conservar todos los nombres de las variables cuyo porcentaje de valores NA es mayor que 0.

```
# Obtener nombres de variables que contienen valores NA
vars_to_exclude=df_status(heart_disease, print_results = F) %>% filt

# Verificar las variables a excluir
vars_to_exclude
```

```
## [1] "num_vessels_flour" "thal"
```

```
# Excluir las variables del conjunto de datos original
heart_disease_clean_2=select(heart_disease, -one_of(vars_to_exclude)
```

4.5.6 Tratamiento de valores vacíos en variables categóricas

Abarcaremos diferentes perspectivas tanto para convertir como para tratar valores vacíos en variables nominales.

Los datos del siguiente ejemplo fueron derivados de `web_navigation_data`, que contiene información estándar sobre cómo llegan los usuarios a un determinado sitio web. Contiene `source_page` (la página desde la que proviene el usuario), `landing_page` (primera página visitada en el sitio), y `country`.

```
# Cuando lean los datos del ejemplo, presten atención
# al parámetro na.strings
web_navigation_data=
  read.delim(file="https://goo.gl/dz7zNx",
             sep="\t",
             header = T,
             stringsAsFactors=F,
             na.strings="")
```

4.5.6.1 Análisis numérico de los datos

```
stat_nav_data=df_status(web_navigation_data)
```

```
##       variable q_zeros p_zeros q_na p_na q_inf p_inf      type un
## 1  source_page       0       0   50 51.5     0     0 character
## 2 landing_page       0       0    5  5.2     0     0 character
## 3      country       0       0    3  3.1     0     0 character
```

Las tres variables tienen valores vacíos (`NA`). Falta casi la mitad de los valores en `source_page`, mientras que las otras dos variables tienen 5% y 3% de valores `NA`.

4.5.6.2 Caso A: Convertir el valor nulo en un string

En variables categóricas o nominales, el tratamiento más rápido es convertir el valor nulo en el string `unknown`. Así, el modelo de machine learning va a tomar los valores "vacíos" como otra categoría. Piénsenlo como una regla: "Si variable_X = unknown, entonces el resultado = sí".

A continuación, proponemos dos métodos para cubrir los escenarios típicos:

Ejemplo en R:

```
library(tidyr)
```

```
## Warning: package 'tidyr' was built under R version 3.5.3
```

```
# Método 1: Convertir una sola variable
web_navigation_data_1=web_navigation_data %>%
  mutate(source_page =
           replace_na(source_page,
                      "unknown_source")
         )

# Método 2: Es una situación típica la de aplicar una
# función sólo a variables específicas y luego
# reingresarlas al data frame original

# Imaginen que queremos convertir todas las variables
# que tengan menos de 6% de valores NA:
vars_to_process =
  filter(stat_nav_data, p_na<6)[,"variable"]

vars_to_process
```

```
## [1] "landing_page" "country"
```

```
# Crear un nuevo data frame con las variables
# transformadas
web_navigation_data_2=web_navigation_data %>%
  mutate_at(.vars=vars(vars_to_process),
            .funs=funs(replace_na(.,"other"))
            )
```

Verificar los resultados:

```
df_status(web_navigation_data_1)
```

```
##       variable q_zeros p_zeros q_na p_na q_inf p_inf      type un
## 1  source_page       0       0    0  0.0     0     0 character
## 2 landing_page       0       0    5  5.2     0     0 character
## 3      country       0       0    3  3.1     0     0 character
```

```
df_status(web_navigation_data_2)
```

```
##       variable q_zeros p_zeros q_na p_na q_inf p_inf      type un
## 1  source_page       0       0   50   52     0     0 character
## 2 landing_page       0       0    0    0     0     0 character
## 3      country       0       0    0    0     0     0 character
```

Nota: Aplicar una función a ciertas columnas es una tarea muy común en cualquier proyecto de datos. Hay más información sobre cómo utilizar la función `mutate_at` de `dplyr` aquí: [How do I select certain columns and give new names to mutated columns?]

4.5.6.3 Caso B: Asignar la categoría más frecuente

La intuición detrás de este método es *agregar más de lo mismo para no afectar la variable*. Sin embargo, a veces la afecta. No tendrá el mismo impacto si el valor más común aparece el 90% de las ocasiones que si aparece el 10%; es decir, depende de la distribución.

Hay otros escenarios en los que podemos incorporar nuevos valores faltantes basándonos en modelos predictivos como k-NN. Este enfoque es más apropiado que reemplazar por el valor más frecuente. No obstante, la técnica recomendada es la que vimos en *Caso A: Convertir el valor nulo en un string*.

4.5.6.4 Caso C: Excluir algunas columnas y transformar otras

El caso sencillo sería que la columna contenga, digamos, 50% de casos `NA`, dado que sería altamente probable que los datos no sean confiables.

En el caso que vimos antes, `source_page` tiene más de la mitad de los valores vacíos. Podríamos excluir esta variable y transformar -como lo hicimos- las otras dos.

El ejemplo está preparado como genérico:

```
# Configurar el umbral
threshold_to_exclude=50 # 50 Represents 50%
vars_to_exclude=filter(stat_nav_data, p_na>=threshold_to_exclude)
vars_to_keep=filter(stat_nav_data, p_na<threshold_to_exclude)

# Finalmente...
vars_to_exclude$variable
```

```
## [1] "source_page"
```

```
vars_to_keep$variable
```

```
## [1] "landing_page" "country"
```

```
# La próxima línea excluirá las variables que queden por encima del
web_navigation_data_3=select(web_navigation_data, -one_of(vars_to_ex
    mutate_at(.vars=vars(vars_to_keep$variable),
              .funs=funs(replace_na(.,"unknown"))
              )

# Verificar que no haya valores NA y que la variable que estaba por
df_status(web_navigation_data_3)
```

```
##       variable q_zeros p_zeros q_na p_na q_inf p_inf      type un
## 1 landing_page       0       0    0    0     0     0 character
## 2      country       0       0    0    0     0     0 character
```

4.5.6.5 Resumiendo

¿Qué pasa si los datos tienen 40% de valores `NA`? Depende del objetivo del análisis y de la naturaleza de los datos.

Lo importante aquí es "salvar" la variable para poder usarla. Es común encontrar muchas variables con valores faltantes. Puede que esas *variables incompletas* contengan información útil para la predicción cuando tienen un valor, por lo tanto, debemos tratarlas y luego construir un modelo predictivo.

De todas maneras, debemos minimizar el sesgo que estamos introduciendo porque el valor faltante es un valor que "no está ahí".

- Cuando estamos haciendo un informe, la sugerencia es reemplazar `NA` con el string `empty`,
- Cuando estamos haciendo un modelo predictivo que se está corriendo en el momento, la recomendación es asignar la categoría que más se repite.

4.5.7 ¿Hay algún patrón en los valores faltantes?

Primero, carguemos los datos de ejemplo de películas y hagamos un análisis rápido.

```
# Lock5Data contiene muchos data frames para practicar
# install.packages("Lock5Data")
library(Lock5Data)
```

```
## Warning: package 'Lock5Data' was built under R version 3.5.2
```

```r
# Cargar datos
data("HollywoodMovies2011")

# Análisis numérico
df_status(HollywoodMovies2011)
```

```
##              variable q_zeros p_zeros q_na  p_na q_inf p_inf    ty
## 1               Movie       0    0.00    0  0.00     0     0  fact
## 2         LeadStudio       0    0.00    0  0.00     0     0  fact
## 3     RottenTomatoes       0    0.00    2  1.47     0     0 integ
## 4      AudienceScore       0    0.00    1  0.74     0     0 integ
## 5               Story       0    0.00    0  0.00     0     0  fact
## 6               Genre       0    0.00    0  0.00     0     0  fact
## 7   TheatersOpenWeek       0    0.00   16 11.76     0     0 integ
## 8  BOAverageOpenWeek       0    0.00   16 11.76     0     0 integ
## 9      DomesticGross       0    0.00    2  1.47     0     0 numer
## 10      ForeignGross       0    0.00   15 11.03     0     0 numer
## 11        WorldGross       0    0.00    2  1.47     0     0 numer
## 12            Budget       0    0.00    2  1.47     0     0 numer
## 13     Profitability       1    0.74    2  1.47     0     0 numer
## 14    OpeningWeekend       1    0.74    3  2.21     0     0 numer
```

Observemos los valores presentes en la columna `p_na`. Hay un patrón en los valores faltantes: cuatro variables tienen 1.47% de valores NA y otras cuatro tienen cerca de 11.7%. En este caso, no podemos chequear la fuente de los datos; sin embargo, es una buena idea verificar si esos casos tienen un problema en común.

4.5.8 Tratar valores faltantes en variables numéricas

Nuestro primer acercamiento a este punto al principio del capítulo fue convertir todos los valores de `NA` a 0.

Una solución es reemplazar los valores vacíos por la media, la mediana u otros criterios. Sin embargo, tenemos que ser conscientes del cambio que esto genera en la distribución.

Si vemos que la variable parece estar correlacionada cuando no está vacía (igual

que la categórica), entonces un método alternativo es crear segmentos, convirtiéndola así en categórica.

4.5.8.1 Método 1: Convertir la variable a categórica

La función `equal_freq` divide la variable en los segmentos deseados. Toma una variable numérica (`TheatersOpenWeek`) y devuelve una categórica (`TheatersOpenWeek_cat`), basándose en el criterio de igual frecuencia.

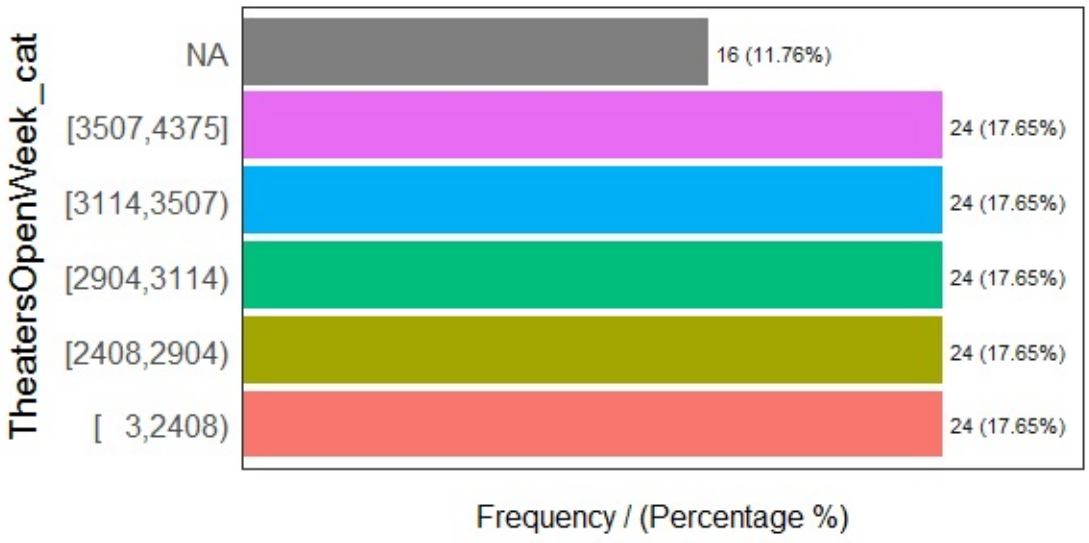
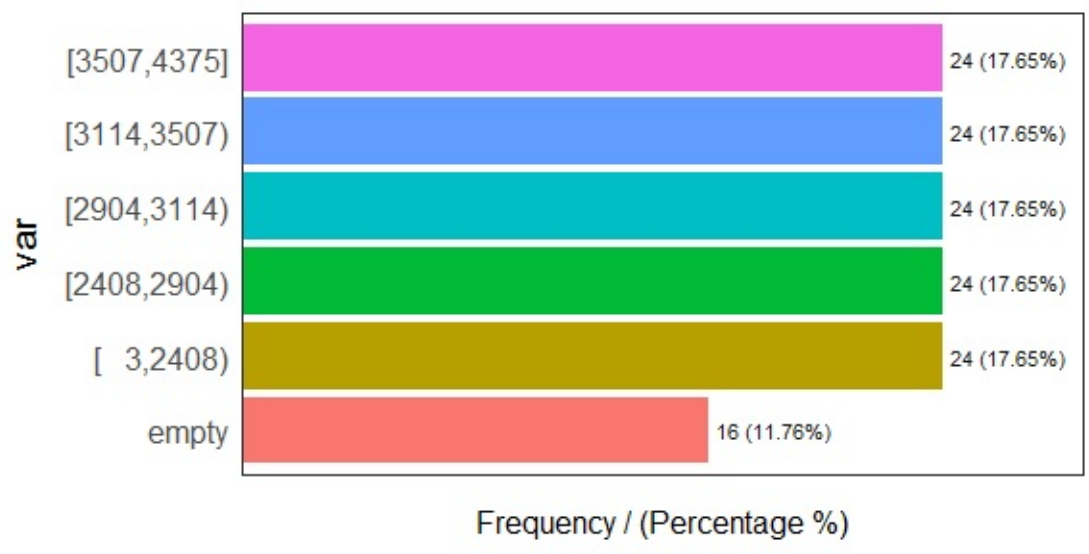


Figure 4.27: Valores faltantes en datos categóricos

```
##   TheatersOpenWeek_cat frequency percentage cumulative_perc
## 1           [   3,2408)        24         18              18
## 2           [2408,2904)        24         18              35
## 3           [2904,3114)        24         18              53
## 4           [3114,3507)        24         18              71
## 5           [3507,4375]        24         18              88
## 6                 <NA>        16         12             100
```

Como podemos ver, `TheatersOpenWeek_cat` contiene cinco segmentos de 24 casos cada uno, cada uno representa el ~18% de los casos totales. Pero, los valores `NA` siguen ahí.

Por último, debemos convertir los NA en el string `empty`.



Y eso es todo: la variable está lista para usar.

Cortes personalizados:

Si queremos usar tamaños personalizados de segmentos en lugar de los que vienen dados por la igual frecuencia, podemos usar la función `cut`. En este caso toma la variable numérica `TheatersOpenWeek` y devuelve `TheatersOpenWeek_cat_cust`.

```
# Desactivar la notación científica en la sesión actual
# de R
options(scipen=999)

# Crear segmentos personalizados, con límites en 1,000,
# 2,300, y un máx de 4,100. Los valores por encima de
# 4,100  serán asignados a NA.

HollywoodMovies2011$TheatersOpenWeek_cat_cust=
  cut(HollywoodMovies2011$TheatersOpenWeek,
      breaks = c(0, 1000, 2300, 4100),
      include.lowest = T,
      dig.lab = 10)

freq(HollywoodMovies2011$TheatersOpenWeek_cat_cust,
     plot = F)


##           var frequency percentage cumulative_perc
## 1 (2300,4100]        94       69.1              69
```

```
## 2        <NA>        19       14.0              83
## 3 (1000,2300]        14       10.3              93
## 4    [0,1000]         9        6.6             100
```

Debemos destacar que la **segmentación por igual frecuencia** tiende a ser más robusta que la igual distancia que divide la variable, que se basa en tomar el mínimo y el máximo, y la distancia entre cada segmento, sin considerar cuántos casos caen en cada segmento.

La igual frecuencia ubica a los valores atípicos en el primer o último segmento según corresponda. Los valores normales pueden ir desde 3 hasta 20 segmentos. Un alto número de segmentos suele significar más ruido. Para leer más, diríjanse al capítulo de cross_plot.

4.5.8.2 Método 2: Completar el NA con algún valor

Al igual que con las variables categóricas, podemos reemplazar los valores con un número como el promedio o la mediana.

En este caso, reemplazaremos los valores `NA` por el promedio y graficar los resultados del antes y el después lado a lado.

```
# Completar todos los valores NA con el promedio de
# la variable
HollywoodMovies2011$TheatersOpenWeek_mean=
  ifelse(is.na(HollywoodMovies2011$TheatersOpenWeek),
         mean(HollywoodMovies2011$TheatersOpenWeek,
              na.rm = T),
         HollywoodMovies2011$TheatersOpenWeek
         )

# Graficar la variable original
p1=ggplot(HollywoodMovies2011, aes(x=TheatersOpenWeek)) +
  geom_histogram(colour="black", fill="white") +
  ylim(0, 30)

# Graficar la variable transformada
p2=ggplot(HollywoodMovies2011,
          aes(x=TheatersOpenWeek_mean)
          ) +
```

```
  geom_histogram(colour="black", fill="white") +
  ylim(0, 30)

# Ubicar los gráficos lado a lado
library(gridExtra)
grid.arrange(p1, p2, ncol=2)
```

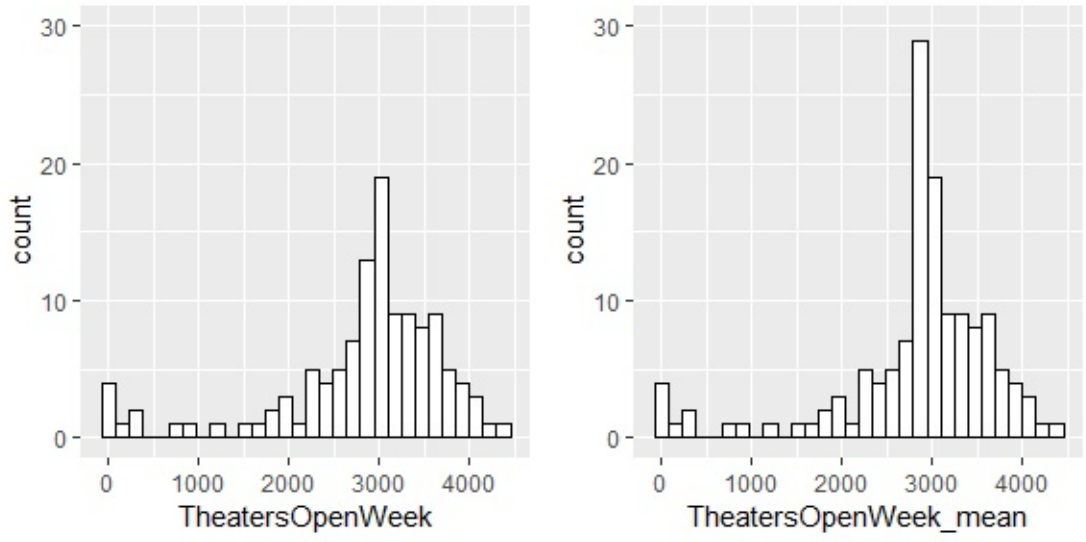


Figure 4.28: Completando los NA con el valor promedio

Podemos ver un pico en `2828`, que es producto de la transformación. Se introduce un sesgo alrededor de este punto. Si estamos prediciendo algún evento, entonces sería más seguro no tener ningún evento especial cerca de este punto.

Por ejemplo, si estamos prediciendo un evento binario y el evento menos representativo está correlacionado con tener un promedio de `3000` en `TheatersOpenWeek`, entonces las probabilidades de tener una **Tasa de falsos positivos** pueden ser más altas. De nuevo, esto se relaciona con el capítulo de Variables de alta cardinalidad en estadística descriptiva.

Como un comentario extra con respecto a la ultima visualización, fue importante configurar el máximo del eje-y en 30 para que los dos gráficos fueran comparables entre sí.

Como pueden ver, existe una interrelación entre todos los conceptos. ☺.

4.5.8.3 Eligiendo el valor adecuado para completar

En el último ejemplo remplazamos los valores `NA` con el promedio, ¿pero qué pasa si usamos otros valores? Depende de la distribución de la variable.

La variable que usamos (`TheatersOpenWeek`) parece tener una distribución normal, que es la razón por la cual utilizamos el promedio. No obstante, si la variable está más sesgada, entonces otra métrica sería más adecuada; por ejemplo, la mediana es menos sensible a los valores atípicos.

4.5.9 Métodos avanzados de imputación

Ahora vamos a hacer un repaso rápido de métodos de imputación más sofisticados en los que creamos un modelo predictivo, con todo lo que eso implica.

4.5.9.1 Método 1: Usando random forest (missForest)

La funcionalidad del paquete missForest se basa en random forest para completar cada valor faltante en un proceso iterativo, manejando variables categóricas y numéricas simultáneamente.

Más allá de la imputación de valores faltantes, el modelo de bosque aleatorio tiene uno de los mejores desempeños con muchos tipos distintos de datos.

En el siguiente ejemplo, completaremos los datos de `HollywoodMovies2011` con los que estábamos trabajando antes. Estos datos contienen valores `NA` tanto en variables numéricas como categóricas.

```
# install.packages("missForest")
library(missForest)
```

```
## Warning: package 'missForest' was built under R version 3.5.3
```

```
## Warning: package 'randomForest' was built under R version 3.5.3

## Warning: package 'foreach' was built under R version 3.5.3

## Warning: package 'itertools' was built under R version 3.5.3

## Warning: package 'iterators' was built under R version 3.5.3
```

```
# Copiar los datos
df_holly=Lock5Data::HollywoodMovies2011

# Volver a crear la TheatersOpenWeek_cat_cust
df_holly$TheatersOpenWeek_cat_cust=
  cut(HollywoodMovies2011$TheatersOpenWeek,
      breaks = c(0, 1000, 2300, 4100),
      include.lowest = T,
      dig.lab = 10)

# Vamos a introducir 15% más de valores NA en
# TheatersOpenWeek_3 para producir un mejor ejemplo.
# La función prodNA en missForest nos va a ayudar.

# Configurar seed para obtener siempre la misma
# cantidad de valores NA
set.seed(31415)

df_holly$TheatersOpenWeek_cat_cust=
  prodNA(
    select(df_holly, TheatersOpenWeek_cat_cust),
    0.15
    )[,1]

# Excluir las variables que no son útiles
df_holly=select(df_holly, -Movie)

# ¡Ahora la magia! Imputar el data frame
# xmis parameter=los datos con valores faltantes
imputation_res=missForest(xmis = df_holly)
```

```
##   missForest iteration 1 in progress...done!
##   missForest iteration 2 in progress...done!
##   missForest iteration 3 in progress...done!
##   missForest iteration 4 in progress...done!
```

```
# data frame final completado
```

```
df_imputed=imputation_res$ximp
```

Nota: `missForest` fallará si tiene alguna variable carácter.

Ahora es momento de comparar las distribuciones de algunas de las variables imputadas, usaremos la variable original previa a la discretización: `TheatersOpenWeek`. Si todo sale bien, se verán parecidas en un análisis visual.

```
# Crear otra imputación basada en na.rougfix del paquete randomFores
df_rough=na.roughfix(df_holly)

# Comparar distribuciones antes y después de la imputación
df_holly$imputation="original"
df_rough$imputation="na.roughfix"
df_imputed$imputation="missForest"

# Poner los dos data frames en uno solo, pero dividido por la variab
df_all=rbind(df_holly, df_imputed, df_rough)

# Convertir a factor para utilizarla en un gráfico
df_all$imputation=factor(df_all$imputation, levels=unique(df_all$imp

# Graficar
ggplot(df_all, aes(TheatersOpenWeek, colour=imputation)) + geom_dens
```

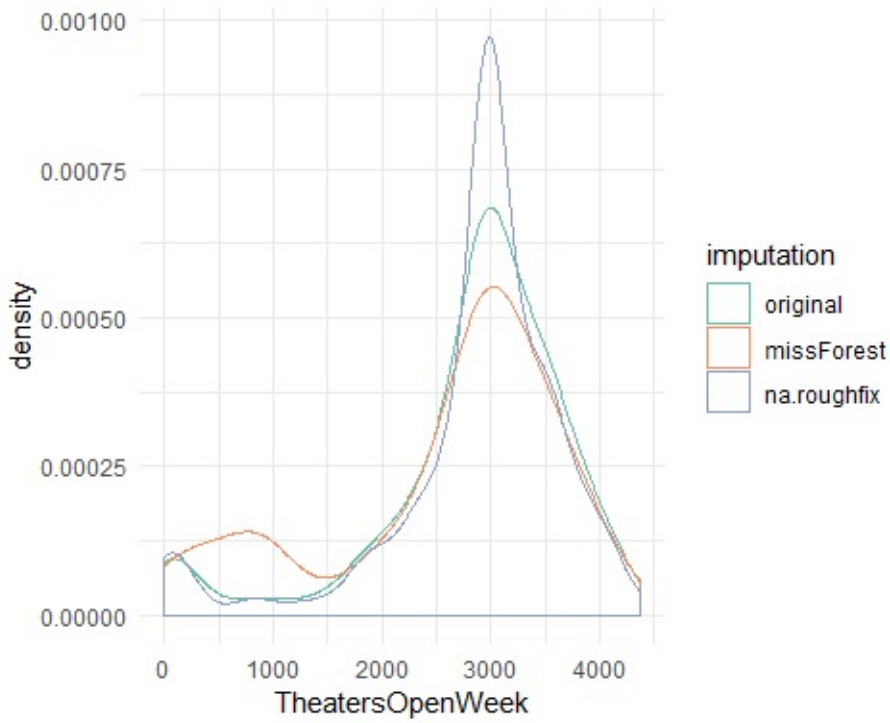


Figure 4.29: Comparación de métodos de imputación (variable numérica)

- La curva naranja muestra la distribución después de la imputación basada en el paquete `missForest`.
- La azul muestra el método de imputación que vimos al principio, que reemplaza todos los valores `NA` por la mediana usando la función `na.roughfix` del paquete `randomForest`.
- La verde muestra la distribución sin ninguna imputación (por supuesto, los valores NA no están expuestos).

Análisis:

Reemplazar los valores `NA` por la mediana tiende a concentrar, como era de esperarse, todos los valores cerca de 3000. Por otro lado, la imputación realizada por el paquete missForest provee una **distribución más natural** porque no concentra los valores cerca de un solo valor. Es por eso que el pico alrededor de 3000 es más bajo que el original.

¡La curva naranja y la verde son bastante parecidas!

Si deseamos tomar un punto de vista analítico, podemos realizar una prueba estadística para comparar, por ejemplo, los promedios o las varianzas.

A continuación vamos a visualizar la variable discretizada y personalizada de `TheatersOpenWeek` (`TheatersOpenWeek_cat_cust`).

```
# Un truco feo para graficar NA como una categoría
levels(df_all$TheatersOpenWeek_cat_cust)=
  c(levels(df_all$TheatersOpenWeek_cat_cust), "NA")

flag_na=is.na(df_all$TheatersOpenWeek_cat_cust)

df_all$TheatersOpenWeek_cat_cust[flag_na]="NA"

# ¡Ahora el gráfico!
ggplot(df_all, aes(x = TheatersOpenWeek_cat_cust,
                   fill = TheatersOpenWeek_cat_cust)
       ) +
  geom_bar(na.rm=T) +
  facet_wrap(~imputation) +
  geom_text(stat='count',
            aes(label=..count..),
            vjust=-1) +
  ylim(0, 125) +
  scale_fill_brewer(palette="Set2") +
  theme_minimal() +
  theme(axis.text.x=
          element_text(angle = 45, hjust = 0.7)
        ) +
  theme(legend.position="bottom")
```

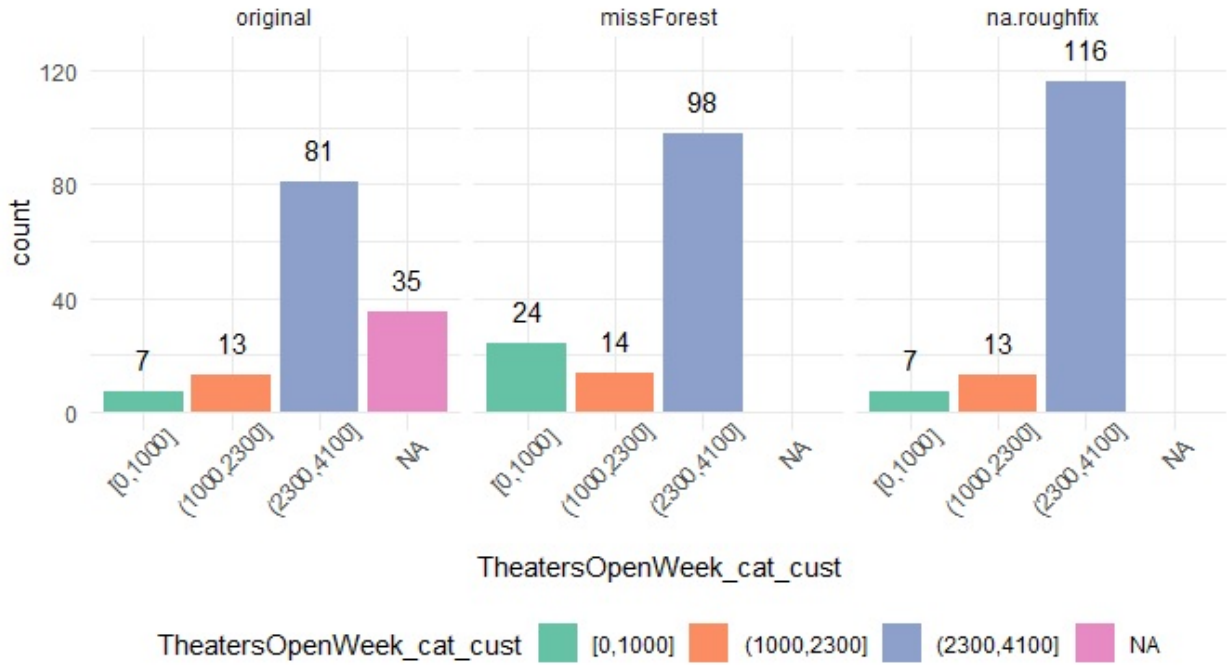


Figure 4.30: Comparación de métodos de imputación

Análisis:

La variable original contiene 35 valores `NA` que fueron reemplazados con la moda o el valor más frecuente en `na.roughfix`: `(2300, 4100]`. Por otro lado, obtuvimos un resultado apenas diferente al utilizar `missForest`, que completó los valores basándose en otras variables.

`missForest` agregó 15 filas en la categoría `[0, 1000]`, 3 en `[1000, 2300]`, y 17 en la categoría `[2300, 4100]`.

4.5.9.2 Método 2: Usar el enfoque MICE

Consejo: Como primer abordaje en la imputación de los valores faltantes, este método es sumamente complejo. 😵.

MICE significa "Imputación Multivariada por ecuaciones encadenadas"

(*"Multivariate Imputation by Chained Equations"* en inglés), también se la conoce como "Especificación totalmente condicional" (*"Fully Conditional Specification"*). Este libro cubre este tema debido a su popularidad.

MICE implica un marco completo para analizar y lidiar con los valores faltantes. Considera las interacciones entre **todas las variables** al mismo tiempo (multivariado y no una sola) y basa su funcionalidad en un proceso **iterativo** que utiliza diferentes modelos predictivos para completar cada variable.

Internamente, completa la variable A, basada en B y C. Luego, llena B basada en A y C (A se predice previamente) y la iteración continúa. El nombre "ecuaciones encadenadas" viene del hecho de que podemos especificar el algoritmo por variable para imputar los casos.

Esto crea réplicas M de los datos originales sin que falten valores. Pero, ¿por qué crear réplicas M?

En cada réplica, la decisión de qué valor insertar en el *espacio vacío* se basa en la distribución.

Muchas demostraciones de MICE se enfocan en validar la imputación y en utilizar los modelos predictivos que apoyan el paquete, que son pocos. Esto es genial si no queremos usar otros modelos predictivos (random forest, gradient boosting machine, etc.), o una técnica de validación cruzada (p.e.,'caret').

La técnica MICE pone el resultado final al establecer una función `pool()` que promedia los parámetros (o betas) de los modelos predictivos M proporcionando facilidades para medir la varianza debido a valores faltantes.

Sí, un modelo por cada data frame generado. Suena como bagging, ¿no? Pero no tenemos esta posibilidad con los modelos mencionados.

MICE tiene muchas funciones que nos ayudan a procesar y validar los resultados de la compleción. Pero, para mantenerlo muy simple, cubriremos sólo una pequeña parte de ellas. El siguiente ejemplo se centrará en la extracción de un data frame **sin valores faltantes, listo para ser utilizado** con otros programas o modelos predictivos.

Ejemplo en R:

Esto imputará los datos del data frame `nhanes` que viene incluido en mice package. Veámoslo:

```
# install.packages("mice")
library(mice)
```

```
## Warning: package 'mice' was built under R version 3.5.3
```

```
df_status(nhanes)
```

```
##   variable q_zeros p_zeros q_na p_na q_inf p_inf    type unique
## 1      age       0       0    0    0     0     0 numeric      3
## 2      bmi       0       0    9   36     0     0 numeric     16
## 3      hyp       0       0    8   32     0     0 numeric      2
## 4      chl       0       0   10   40     0     0 numeric     13
```

Tres variables tienen valores faltantes. Vamos a completarlos:

```
# La imputación por defecto crea cinco conjuntos de datos completos
imp_data=mice(nhanes, m = 5, printFlag = FALSE)

# Obtener el conjunto de datos final que contiene los cinco data fra
data_all=complete(imp_data, "long")

# data_all contiene las mismas columnas que nhanes más dos adicional
# .id=filas de la 1 a la 25
# .imp=imputar el data frame .id 1 a 5 (parámetro m)
```

En los datos originales, `nhanes` tiene 25 filas y data_all tiene 125 filas, que es el resultado de crear 5 (`m=5`) data frames completos de 25 filas cada uno.

Es hora de chequear los resultados:

```
densityplot(imp_data)
```

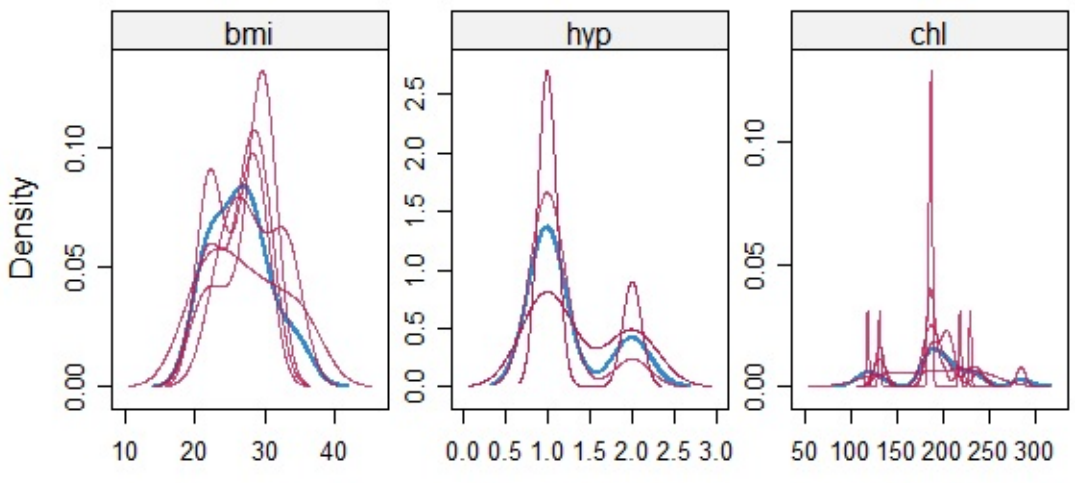


Figure 4.31: Analizar los valores faltantes usando MICE

Cada línea roja muestra la distribución de cada data frame imputado y la línea azul contiene la distribución original. La idea detrás de esto es que si se ven similares, entonces la imputación sigue la distribución original.

Por ejemplo, chl contiene un data frame completado; por lo tanto, sólo una línea roja con dos picos alrededor de dos valores muy superiores a los originales.

El inconveniente es que se trata de un proceso lento que puede requerir ciertos ajustes finos para funcionar. Por ejemplo: `mice_hollywood=mice(HollywoodMovies2011, m=5)` fallará después de algún tiempo de procesarlo y es un data frame pequeño.

Más información sobre el paquete `MICE`:

- Paper original sobre MICE: Multivariate Imputation by Chained Equations in R
- Handling missing data with MICE package; a simple approach

4.5.10 Conclusiones

Habiendo cubierto todas las opciones, podríamos preguntar: ¿cuál es la mejor

estrategia? Bueno, depende de cuánto queramos intervenir para manejar los valores faltantes.

Un pequeño repaso de las estrategias posibles:

A. Excluir las filas y columnas con valores faltantes. Sólo es aplicable si hay *pocas* filas (o columnas) con valores faltantes, **y** los datos restantes son suficientes para alcanzar la meta del proyecto. No obstante, cuando excluimos filas con valores faltantes y creamos un modelo predictivo que va a ejecutarse en producción, si **llega un caso nuevo** que contiene valores faltantes, debemos asignar un valor para procesarlos.

B. Las estrategias de **convertir variables numéricas a categóricas** y luego crear el valor "empty" (también aplicable a las variables categóricas), es la opción más rápida -y recomendada- para lidiar con valores nulos. De esta manera incorporamos los valores faltantes al modelo para que pueda manejar la incertidumbre.

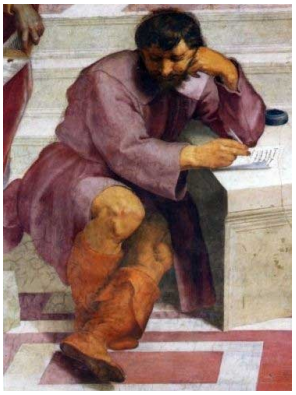
C. Los **métodos de imputación** como los que cubrimos con MICE y missForest son considerablemente más complejos. Con estos métodos, introducimos un **sesgo controlado** para no tener que excluir filas o columnas.

Es un arte encontrar el equilibrio correcto entre profundizar en estas transformaciones y mantenerlo simple. El tiempo invertido puede no reflejarse en la precisión global.

Independientemente del método, es muy importante analizar el impacto de cada decisión. Hay mucho de prueba y error, así como análisis exploratorio de datos, que conduce al descubrimiento del método más adecuado para sus datos y proyecto.

4.6 Consideraciones que involucran al tiempo

4.6.1 ¿De qué se trata esto?



> Todo cambia y nada permanece. - Heráclito, (535 - 475 AC), filósofo griego presocrático.

Lo mismo ocurre con las variables.

Con el paso del tiempo, las variables pueden cambiar sus valores, haciendo que el análisis del tiempo sea crucial a la hora de crear un modelo predictivo. Así evitamos tomar los **efectos** como **causas**.

¿Qué vamos a repasar en este capítulo?

- Conceptos de filtrado de información antes del evento a predecir.
- Cómo analizar y preparar las variables que aumentan -o disminuyen- su valor hasta el infinito (y más allá).

4.6.1.1 No utilicen información del futuro

Imagen de la película: "Volver al futuro" (1985). Robert Zemeckis (Director).

Usar una variable que contiene información **después** del evento que se está prediciendo es un error común cuando se comienza un nuevo proyecto de modelo predictivo, como jugar a la lotería hoy usando el periódico de mañana.

Imaginemos que necesitamos construir un modelo predictivo para saber qué usuarios es probable que adquieran una suscripción completa en una aplicación web, y este software tiene una funcionalidad ficticia llamada `feature_A`:

```
##    user_id feature_A full_subscription
## 1        1       yes               yes
## 2        2       yes               yes
## 3        3       yes               yes
## 4        4        no                no
## 5        5       yes               yes
## 6        6        no                no
## 7        7        no                no
## 8        8        no                no
## 9        9        no                no
## 10      10        no                no
```

Creamos el modelo predictivo, obtuvimos una precisión perfecta, y una inspección arroja el siguiente mensaje: *"El 100% de los usuarios que tienen una suscripción completa utiliza la característica Feature A"*. Algunos algoritmos predictivos reportan la importancia de las variables; por lo que `feature_A` estará por encima del resto.

El problema es: `feature_A` solo está disponible **después de que el usuario adquiere la suscripción completa**. Por lo tanto, no puede ser utilizada.

El mensaje clave es: No confíen en variables perfectas, ni modelos perfectos.

4.6.1.2 Sean justos con los datos, dejen que desarrollen su comportamiento

Como en la naturaleza, las cosas tienen un tiempo mínimo y máximo para empezar a mostrar cierto comportamiento. Este tiempo oscila de 0 a infinito. En la práctica se recomienda estudiar cuál es el mejor período para analizar, es decir, podemos excluir todo el comportamiento antes y después de este período de observación. Establecer rangos en las variables no es sencillo, ya que puede ser un poco subjetivo.

Imaginen que tenemos una **variable numérica** que aumenta a medida que pasa el tiempo. Es posible que necesitemos definir una **ventana de tiempo de observación** para filtrar los datos y alimentar el modelo predictivo.

- Configurar el tiempo **mínimo**: ¿Cuánto tiempo es necesario para empezar a ver el comportamiento?
- Configurar el tiempo **máximo**: ¿Cuánto tiempo es necesario para ver el final del comportamiento?

La solución más fácil es: configurar el mínimo en el principio y el máximo en toda la historia.

Estudio de caso:

Dos personas, `Ouro` y `Borus`, son usuarios de una aplicación web que tiene una determinada funcionalidad llamada `feature_A`, y necesitamos crear un modelo predictivo que pronostique basándose en el uso de `feature_A` usage -medido en clicks- si una persona va a adquirir `full_subscription`.

Los datos actuales dicen: Borus tiene `full_subscription`, mientras que Ouro no.

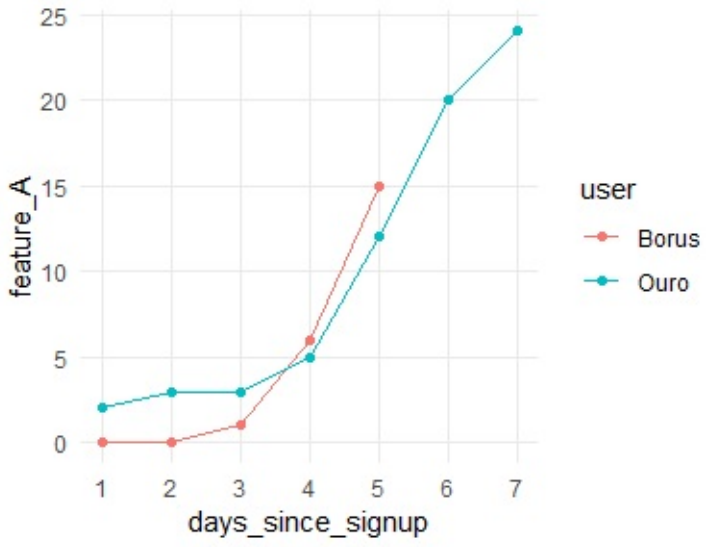


Figure 4.32: Tengan cuidado con las consideraciones que involucran el tiempo

El usuario `Borus` comienza a usar `feature_A` a partir del día 3, y después de 5 días ella le da más uso -15 clicks vs. 12- a esta función que `Ouro`, que comenzó a usarla desde el día 0.

Si `Borus` adquiere `full subscription` y `Ouro` no, *¿qué aprenderá el modelo?*

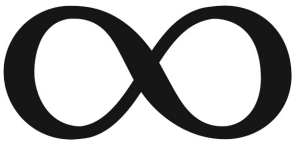
Cuando modelamos con la historia completa -`days_since_signup = all`-, cuanto más alto sea `days_since_signup` mayor será probabilidad, dado que `Borus` tiene el número más alto.

Sin embargo, si sólo tomamos la historia de los usuarios de los primeros 5 días desde la suscripción, la conclusión será la opuesta.

¿Por qué conservar los primeros 5 días de la historia?

El comportamiento durante este período inicial (*kick-off*) puede ser más relevante -con respecto a la precisión de la predicción- que analizar toda la historia. Como dijimos antes, depende de cada caso.

4.6.1.3 Luchar contra el infinito



El número de ejemplos sobre este tema es muy amplio. Mantengamos la esencia de este capítulo en **cómo cambian los datos a lo largo del tiempo**. A veces es sencillo, como una variable que alcanza su mínimo (o máximo) después de un tiempo fijo. Este caso es fácilmente alcanzable.

Por otro lado, requiere que el ser humano luche contra el infinito.

Consideren el siguiente ejemplo. *¿Cuántas horas se necesitan para alcanzar el valor 0?*

¿Qué tal 100 horas?

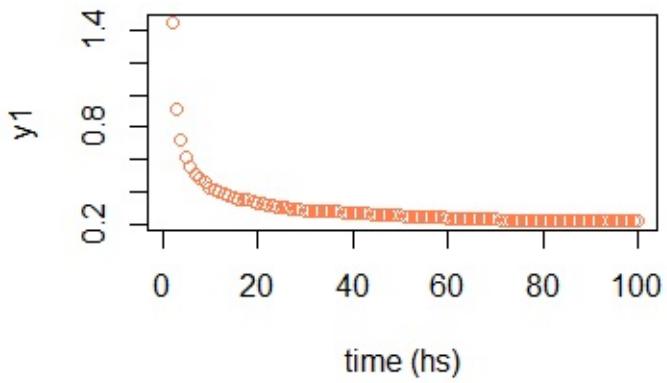


Figure 4.33: 100 horas

Hmm chequeemos el valor mínimo...

```
## [1] "Min value after 100 hours: 0.22"
```

Está cerca de cero, pero *¿qué pasa si esperamos 1000 horas?*

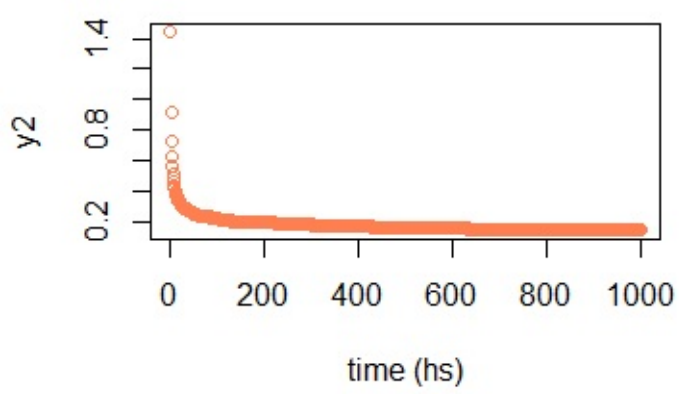


Figure 4.34: 1,000 horas

```
## [1] "Min value after 1,000 hours: 0.14"
```

¡Hurra! ¡Nos estamos acercando! De `0.21` a `0.14`. Pero ¿qué pasa si esperamos 10 veces más? (10,000 horas)

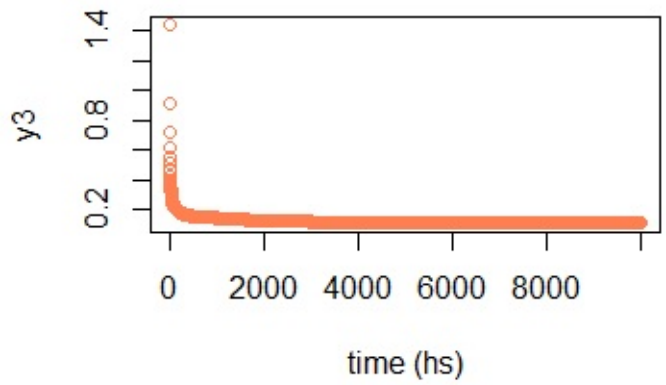


Figure 4.35: 10,000 horas

```
## [1] "Min value after 10,000 hours: 0.11"
```

¡Seguimos sin llegar a cero! ¡¿Cuánto tiempo necesitamos?! 😱

Como habrán notado, es probable que lleguemos a cero en el infinito... Estamos ante una Asíntota.

¿Qué debemos hacer? Es momento de pasar a la siguiente sección.

4.6.1.4 Amigarnos con el infinito

En el último ejemplo vimos el análisis de la antigüedad de un consumidor en una compañía. *Este valor puede ser infinito.*

Por ejemplo, si el objetivo del proyecto es predecir un resultado binario, como `buy/don't buy`, un análisis útil es calcular la tasa de `buy` de acuerdo a la antigüedad del usuario. Llegaremos a conclusiones como: *En promedio, un consumidor necesita cerca de 6 meses para comprar este producto.*

Esta respuesta puede alcanzarse gracias al trabajo conjunto del científico de datos y un experto en la materia.

En este caso, un cero puede considerarse al igual que el valor que tenga el 95% de la población. En términos estadísticos, es el **percentil** `0.95`. Este libro abarca extensamente este tema en Anexo 1: La magia de los percentiles. Es un tema clave en el análisis exploratorio de datos.

Un caso relacionado es el de **lidiar con valores atípicos**, cuando aplicamos percentiles como criterio de corte, como vimos en el capítulo Tratamiento de valores atípicos.

4.6.1.5 Ejemplos en otras áreas

Es muy común encontrar este tipo de variables en muchos conjuntos o proyectos de datos.

En medicina, en los proyectos de análisis de supervivencia, los médicos suelen definir un umbral de, por ejemplo, 3 años para considerar que un paciente *sobrevive* al tratamiento.

En proyectos de marketing, si un usuario disminuye su actividad dentro de un cierto umbral, digamos: * 10-clicks en el sitio web de la compañía en el último mes * No abrir un email después de 1 semana * Él o ella no compra por 30 días

Se lo puede definir como la pérdida de un cliente u oportunidad.

En atención al cliente, un problema puede marcarse como resuelto una vez que la persona pasó 1 semana sin reportar nuevas quejas.

En el **análisis de señales cerebrales**, si estas señales provienen de la corteza visual en un proyecto en el que, por ejemplo, necesitamos predecir qué tipo de imagen está mirando el paciente, entonces los primeros 40ms de valores no sirven porque es el tiempo que el cerebro necesita para empezar a procesar la señal.

Pero esto también ocurre en *"la vida real"*, como cuando escribimos un libro de análisis de datos para todas las edades, ¿cuánto tiempo necesitamos para terminarlo? ¿Una cantidad infinita? Probablemente no. 😄.

4.6.1.6 Reflexiones finales

Definir un periodo de tiempo para crear un conjunto de entrenamiento y validación, implica un sesgo importante en nuestros datos. Al igual que decidir cómo manejar las variables que cambian con el tiempo. Es por eso que el **Análisis exploratorio de datos** es importante para entrar en contacto con los datos que estamos analizando.

Los temas están interconectados. Ahora es momento de mencionar la relación de este capítulo con la Validación out-of-time. Cuando predecimos eventos en el futuro, debemos analizar cuánto tiempo es necesario para que la variable objetivo cambie.

El concepto clave aquí es: **cómo manejar el tiempo en modelado predictivo**. Es una buena oportunidad para preguntar: *¿Cómo sería posible abordar estos problemas del tiempo con sistemas automáticos?*

El conocimiento humano es crucial en estos contextos para definir umbrales

basándonos en la experiencia, intuición y algunos cálculos.

5 Methods

We describe our methods in this chapter.

6 Selección de las mejores variables

6.1 Aspectos generales de la selección de las mejores variables

6.1.1 ¿De qué se trata esto?

Este capítulo abarca los siguientes temas:

- El ranking de mejores variables según los algoritmos convencionales de machine learning, ya sean predictivos o de agrupamiento.
- La naturaleza de la selección de variables con y sin modelos predictivos.
- El efecto de las variables trabajando en grupos (intuición y Teoría de la Información).
- Explorar el mejor subconjunto de variables en la práctica usando R.

La selección de las mejores variables también se conoce como selección de los predictores más importantes, selección de los mejores predictores, entre otros.

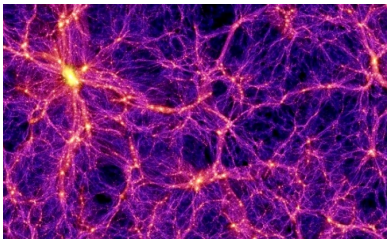


Figure 6.1: Como es arriba, es abajo

Image: ¿Es una red neuronal? Nop. Materia oscura, de "The Millennium Simulation Project".

6.2 Intuición

Seleccionar las mejores variables es como hacer un resumen de una historia,

queremos concentrarnos en unos pocos detalles que mejor describen lo que estamos contando. El equilibrio está entre hablar *demasiado* sobre detalles innecesarios (sobreajustar/overfitting) y hablar *muy poco* sobre la esencia de la historia (subajustar/underfitting).

Otro ejemplo puede ser la decisión de comprar una nueva computadora portátil: *¿cuáles son los factores que más nos importan? ¿Precio, color y forma de envío? ¿Color y duración de la batería? ¿O sólo precio?*

Desde el punto de vista de la **Teoría de la Información** -un punto clave en machine learning-, los datos con los que estamos trabajando tienen **entropía** (caos). Cuando seleccionamos variables, estamos disminuyendo la entropía en nuestro sistema agregando información.

6.3 ¿La "mejor" selección?

Este capítulo dice "las mejores", pero conviene que mencionemos un punto conceptual: en términos generales, *no hay una única selección de mejores variables.*

Partir de esta perspectiva es importante dado que, en la exploración de muchos algoritmos que *clasifican* las variables según su poder predictivo, podemos encontrar resultados diferentes -y similares-. Por ejemplo:

- El algoritmo 1 seleccionó `var_1` como la mejor variable, seguida de `var_5` y `var_14`.
- El algoritmo 2 hizo este ranking: `var_1`, `var_5` y `var_3`.

Imaginemos, basándonos en el algoritmo 1, que la precisión es del 80%, mientras que la precisión al basarnos en el algoritmo 2 es del 78%. Considerando que cada modelo tiene su propia varianza interna, los resultados pueden ser vistos como iguales.

Esta perspectiva nos puede ayudar a reducir el tiempo que nos lleva buscar la selección perfecta de variables.

No obstante, yendo a los extremos, sí habrá un conjunto de variables que tendrá

una alta clasificación en muchos algoritmos, y lo mismo aplica para aquellas con bajo poder predictivo. Después de varias ejecuciones, las variables más confiables saldrán rápidamente a la luz, entonces:

Conclusión: Si los resultados no son buenos, deberíamos enfocarnos en mejorar y revisar la instancia de **preparación de datos**. *La siguiente sección dará un ejemplo de esto.*

6.3.1 Profundizando en la clasificación de variables

Es bastante común encontrar en literatura y algoritmos que cubren este tema un análisis univariado, que es un ranking de variables según una métrica particular.

Vamos a crear dos modelos: random forest y gradient boosting machine (GBM) usando el paquete `caret` en R para hacer una verificación cruzada de los datos. Luego, vamos a comparar el ranking de mejores variables que devuelva cada modelo.

```
library(caret)
library(funModeling)
library(dplyr)

# Excluir todas las filas NA de los datos, en este caso, los NA no s
heart_disease=na.omit(heart_disease)

# Configurar una validación cruzada cuádruple
fitControl = trainControl(method = "cv",
                          number = 4,
                          classProbs = TRUE,
                          summaryFunction = twoClassSummary)

# Crear el modelo de random forest, encontrando el mejor conjunto de
set.seed(999)
fit_rf = train(x=select(heart_disease, -has_heart_disease, -heart_di
               y = heart_disease$has_heart_disease,
               method = "rf",
               trControl = fitControl,
               verbose = FALSE,
               metric = "ROC")
```

```
# Crear el modelo de gradient boosting machine, encontrando el mejor
fit_gbm = train(x=select(heart_disease, -has_heart_disease, -heart_d
             y = heart_disease$has_heart_disease,
             method = "gbm",
             trControl = fitControl,
             verbose = FALSE,
             metric = "ROC")
```

Ahora podemos proceder a la comparación.

Las columnas `importance_rf` y `importance_gbm` representan la importancia medida por cada algoritmo. De acuerdo a cada métrica, existen `rank_rf` y `rank_gbm`, que representan el orden de importancia. Finalmente, `rank_diff` (`rank_rf - rank_gbm`) representa cuán diferentes son los rankings de los algoritmos.

```
# Aquí manipulamos para mostrar una linda tabla con lo que describim
var_imp_rf=data.frame(varImp(fit_rf, scale=T)["importance"]) %>%
  dplyr::mutate(variable=rownames(.)) %>% dplyr::rename(importance_r
  dplyr::arrange(-importance_rf) %>%
  dplyr::mutate(rank_rf=seq(1:nrow(.)))

var_imp_gbm=as.data.frame(varImp(fit_gbm, scale=T)["importance"])  %
  dplyr::mutate(variable=rownames(.)) %>% dplyr::rename(importance_g
  dplyr::arrange(-importance_gbm) %>%
  dplyr::mutate(rank_gbm=seq(1:nrow(.)))
final_res=merge(var_imp_rf, var_imp_gbm, by="variable")

final_res$rank_diff=final_res$rank_rf-final_res$rank_gbm

# ¡Visualizar los resultados!
final_res
```

variable	importance_rf	rank_rf	importance_gbm	rank_gbm	rank_diff
exter_angina	29.15468	11	11.544326	6	5
slope	37.44874	10	20.799135	5	5
gender	21.72545	12	5.557935	9	3
chest_pain	97.85192	3	100.000000	1	2
fasting_blood_sugar	0.00000	14	0.000000	13	1
oldpeak	89.31555	5	51.556053	4	1
resting_blood_pressure	57.47780	8	5.690607	8	0
thal	98.15852	2	91.395284	2	0
exer_angina	37.68362	9	5.106130	10	-1
resting_electro	9.09967	13	0.000000	14	-1
num_vessels_flour	100.00000	1	88.478330	3	-2
max_heart_rate	96.89898	4	6.697837	7	-3
serum_cholestoral	62.03931	7	3.344057	11	-4
age	76.10440	6	3.140659	12	-6

Figure 6.2: Comparación de diferentes clasificaciones de variables

Podemos ver que hay variables que no son para nada importantes en ambos modelos (`fasting_blood_sugar`). Hay otras que mantienen una posición en lo más alto del ranking de importancia, como `chest_pain` y `thal`.

Las implementaciones de diferentes modelos predictivos tienen sus propios criterios para informar cuáles son los mejores factores, de acuerdo a ese modelo particular. Esto resulta en diferentes clasificaciones en los diferentes algoritmos. *Hay más información sobre la importancia de las métricas internas en la documentación de caret*.

Además, en los modelos basados en árboles, como GBM y random forest, hay un componente aleatorio en la selección de variables, y la importancia se basa en una selección anterior -y automática- a la hora de construir los árboles. La importancia de cada variable depende de las otras, no solamente en su contribución aislada: **Las variables trabajan en grupos**. Regresaremos a este punto más adelante en este capítulo.

Si bien el ranking variará de algoritmo a algoritmo, en términos generales suele

haber una correlación entre todos estos resultados, como mencionamos previamente.

Conclusión: Cada lista de clasificación de variables no es la *"verdad final"*, nos orienta sobre dónde está la información.

6.4 La naturaleza de la selección

Hay dos principales enfoques a la hora de seleccionar variables:

Dependiente de los modelos predictivos:

Como los ejemplos que vimos antes, este es el más común. El modelo ordenará las variables de acuerdo a una medida intrínseca de precisión. En los modelos basados en árboles, métricas como ganancia de información, índice de Gini, impureza de nodos. Hay más información sobre esto en (stackoverflow.com 2017) y (stats.stackexchange.com 2017a).

No dependiente de los modelos predictivos:

Los criterios de este tipo son interesantes dado que no son tan populares como los otros, pero se ha comprobado que funcionan muy bien en áreas relacionadas con datos genómicos. Deben encontrar aquellos genes *relevantes* (variable de entrada) que están correlacionados con alguna enfermedad, como cáncer (variable objetivo).

Los datos de esta área de estudio se caracterizan por tener una enorme cantidad de variables (miles), que es mucho mayor a los problemas que abordan otras áreas.

Un algoritmo para lograr esto es mRMR, acrónimo de Selección de Factores con Mínima Redundancia y Máxima Relevancia (*Minimum Redundancy Maximum Relevance Feature Selection* en inglés). Tiene su propia implementación en R en el paquete mRMRe.

Otro algoritmo no dependiente de modelos es `var_rank_info`, una función incluida en el paquete funModeling. Clasifica las variables usando varias

métricas de **Teoría de la Información**. Presentaremos un ejemplo más adelante.

6.5 Mejorar variables

Podemos aumentar el poder predictivo de las variables tratándolas.

Hasta aquí, este libro ha cubierto:

- Mejora de variables categóricas.
- Reducción de ruido en variables numéricas mediante agrupación en el capítulo: Discretizando variables numéricas.
- Cómo lidiar con valores atípicos en R.
- Análisis, manejo e imputación

6.6 Limpiar por conocimiento del dominio

Esto no está relacionado con procedimientos algorítmicos, sino con el área de donde vienen los datos.

Imaginen que los datos vienen de una encuesta. Esta encuesta tiene un año de historia, y durante los primeros tres meses no hubo un buen control del proceso. Al ingresar los datos, los usuarios podían escribir lo que quisieran. Las variables durante este período probablemente serán espurias.

Es fácil reconocerlo cuando durante un período de tiempo dado la variable viene vacía, nula o con valores extremos.

Entonces deberíamos hacer una pregunta:

¿Estos datos son confiables? Tengan en cuenta que el modelo predictivo aprenderá *como un niño*, no juzgará los datos, solo aprenderá de ellos. Si los datos son espurios en un período de tiempo determinado, entonces podemos eliminar estos casos ingresados.

Para avanzar más en este punto, deberíamos hacer un análisis exploratorio más

profundo de los datos. Tanto numérica como gráficamente.

6.7 Las variables trabajan en grupos

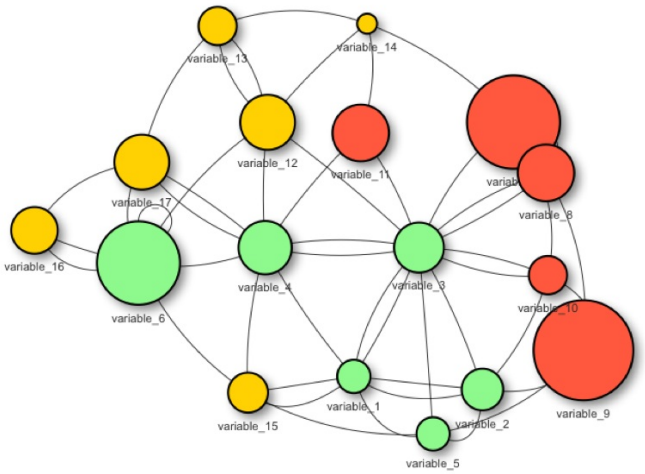


Figure 6.3: Las variables trabajan en grupos

Cuando seleccionamos las *mejores* variables, el principal objetivo es obtener aquellas variables que contienen la mayor información con respecto a una variable objetivo, de resultado o dependiente.

Un modelo predictivo encontrará sus pesos o parámetros basándose en sus 1 a 'N' variables de ingreso.

Las variables aisladas no suelen funcionar para explicar un evento. Citando a Aristóteles:

> “El todo es más que la suma de las partes.”

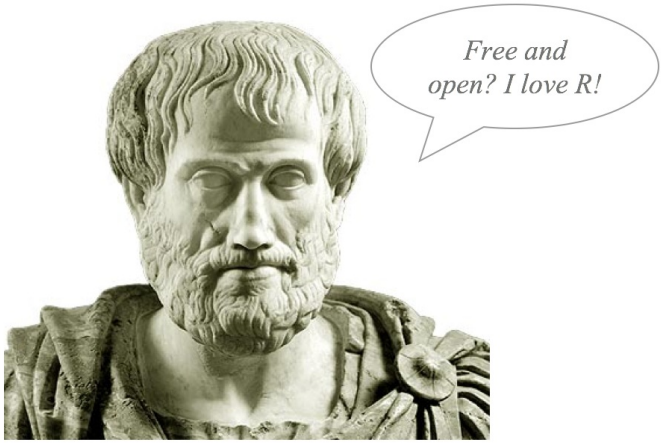
Esto también aplica cuando queremos seleccionar las *mejores* variables:

Construir un modelo predictivo con dos variables puede brindar una mayor precisión que los modelos construidos con una sola variable.

Por ejemplo: Construir un modelo basado en la variable `var_1` podría llevarnos a una precisión global del 60%. Por otro lado, un modelo basado en `var_2` podría

alcanzar una precisión del 72%. Pero al combinar estas dos variables, `var_1` y `var_2` variables, podríamos lograr una precisión que supere el 80%.

6.7.1 Ejemplo en R: Variables trabajando en grupos



Aristotle (384 - 322 BC).
Philosopher and Data Scientist.

Figure 6.4: Aristóteles (384 a.C.–322 a.C.)

El siguiente código ilustra lo que Aristóteles dijo hace *algunos* años.

Crea 3 modelos basados en diferentes subconjuntos de variables:

- El modelo 1 está basado en la variable de entrada `max_heart_rate`
- El modelo 2 está basado en la variable de entrada `chest_pain`
- El modelo 3 está basado en las variables de entrada `max_heart_rate` **y** `chest_pain`

Cada modelo devuelve la métrica ROC, y el resultado contiene la mejora de considerar dos variables al mismo tiempo vs. tomar cada variable por separado.

```
library(caret)
library(funModeling)
library(dplyr)

# Configurar la validación cruzada 4-fold
fitControl =
```

```r
  trainControl(method = "cv",
               number = 4,
               classProbs = TRUE,
               summaryFunction = twoClassSummary
               )

create_model<-function(input_variables)
{
  # Crear el modelo de gradient boosting machine
  # basado en las variables de entrada
  fit_model = train(x=select(heart_disease,
                             one_of(input_variables)
                             ),
                    y = heart_disease$has_heart_disease,
                    method = "gbm",
                    trControl = fitControl,
                    verbose = FALSE,
                    metric = "ROC")

  # Devolver el ROC como una métrica de desempeño
  max_roc_value=max(fit_model$results$ROC)
  return(max_roc_value)
}

roc_1=create_model("max_heart_rate")
roc_2=create_model("chest_pain")
roc_3=create_model(c("max_heart_rate", "chest_pain"))

avg_improvement=round(100*(((roc_3-roc_1)/roc_1)+
                             ((roc_3-roc_2)/roc_2))/2,
                      2)
avg_improvement_text=sprintf("Average improvement: %s%%",
                             avg_improvement)

results =
  sprintf("ROC model based on 'max_heart_rate': %s.;
  based on 'chest_pain': %s; and based on both: %s",
  round(roc_1,2),
  round(roc_2,2),
  round(roc_3, 2)
  )

# ¡Visualizar los resultados!
cat(c(results, avg_improvement_text), sep="\n\n")


## ROC model based on 'max_heart_rate': 0.74.;
```

```
##   based on 'chest_pain': 0.77; and based on both: 0.82
##
## Average improvement: 9.03%
```

6.7.2 Pequeño ejemplo (basado en la Teoría de la información)

Consideren la siguiente tabla de *big data* 😜 4 filas, 2 variables de entrada (`var_1`, `var_2`) y un resultado (`target`):

var_1	var_2	target
a	x	red
a	z	blue
b	x	blue
b	z	red

Figure 6.5: La unión hace a la fuerza: Combinar variables

Si creamos un modelo predictivo basado solamente en `var_1`, ¿qué *verá* este modelo?, el valor `a` está correlacionado con el resultado `blue` y `red` en la misma proporción (50%):

- Si `var_1='a'` entonces la probabilidad del objetivo='red' es 50% (fila 1)
- Si `var_1='b'` entonces la probabilidad del objetivo='blue' es 50% (fila 2)

El mismo análisis aplica para `var_2`

Cuando los mismos datos ingresados están relacionados con diferentes resultados lo definimos como **ruido**. La intuición es lo mismo que una persona diciéndonos: *"Hey, ¡mañana va a llover!"*, y otra persona diciéndonos *"Es seguro que mañana no va a llover"*. Pensaríamos... *"¡Dios mío! ¿Necesito el paraguas o no 😱?"*

Volviendo al ejemplo, al tomar las dos variables en simultáneo, la correspondencia entre los datos ingresados y el resultado es única: "Si `var_1='a'` y `var_2='x'` entonces la probabilidad de `target='red'` es 100%". Pueden probar otras combinaciones.

Resumiendo:

Ese fue un ejemplo de **variables trabajando en grupos**, considerar `var_1` y

`var_2` al mismo tiempo aumenta el poder predictivo.

No obstante, es un tema más profundo para tratar, teniendo en cuenta el último análisis; ¿qué pasa si tomamos una columna `Id` (cada valor es único) para predecir algo? La correspondencia entre datos ingresados y resultado también será única... ¿pero es un modelo útil? Hablaremos más sobre Teoría de la Información en este libro.

6.7.3 Conclusiones

- El ejemplo en R que propusimos, basado en los datos de `heart_disease`, muestra en promedio una **mejora del 9%** al considerar dos variables al mismo tiempo, nada mal. Este porcentaje de mejora es el resultado de las **variables trabajando en grupos**.
- Este efecto aparece si las variables contienen información, como es el caso de `max_heart_rate` y `chest_pain` (o `var_1` y `var_2`).
- Colocar **variables ruidosas** junto a variables buenas **usualmente afectará** el desempeño global.
- Además el efecto del **trabajo en grupos** es mayor si las variables de entrada **no están correlacionadas entre sí**. Es difícil optimizar esto en la práctica. Seguiremos hablando de esto en la siguiente sección...

6.7.4 Rankear las mejores variables usando la Teoría de la Información

Tal como anticipamos al principio de este capítulo, podemos saber la importancia de una variable sin recurrir a un modelo predictivo si utilizamos la Teoría de la Información.

A partir de la versión 1.6.6 el paquete `funModeling` introduce la función `var_rank_info` que toma dos argumentos, los datos y la variable objetivo, porque sigue:

```
variable_importance =
  var_rank_info(heart_disease, "has_heart_disease")
```

```r
# Visualizar resultados
variable_importance
```

```
##                        var    en    mi           ig           gr
## 1  heart_disease_severity 1.846 0.995 0.9950837595 0.5390655068
## 2                     thal 2.032 0.209 0.2094550580 0.1680456709
## 3             exer_angina 1.767 0.139 0.1391389302 0.1526393841
## 4            exter_angina 1.767 0.139 0.1391389302 0.1526393841
## 5              chest_pain 2.527 0.205 0.2050188327 0.1180286190
## 6       num_vessels_flour 2.381 0.182 0.1815217813 0.1157736478
## 7                   slope 2.177 0.112 0.1124219069 0.0868799615
## 8       serum_cholestoral 7.481 0.561 0.5605556771 0.0795557228
## 9                  gender 1.842 0.057 0.0572537665 0.0632970555
## 10                oldpeak 4.874 0.249 0.2491668741 0.0603576874
## 11         max_heart_rate 6.832 0.334 0.3336174096 0.0540697329
## 12 resting_blood_pressure 5.567 0.143 0.1425548155 0.0302394591
## 13                    age 5.928 0.137 0.1371752885 0.0270548944
## 14        resting_electro 2.059 0.024 0.0241482908 0.0221938072
## 15    fasting_blood_sugar 1.601 0.000 0.0004593775 0.0007579095
```

```r
# Graficar
ggplot(variable_importance,
       aes(x = reorder(var, gr),
           y = gr, fill = var)
       ) +
  geom_bar(stat = "identity") +
  coord_flip() +
  theme_bw() +
  xlab("") +
  ylab("Variable Importance
       (based on Information Gain)"
       ) +
  guides(fill = FALSE)
```

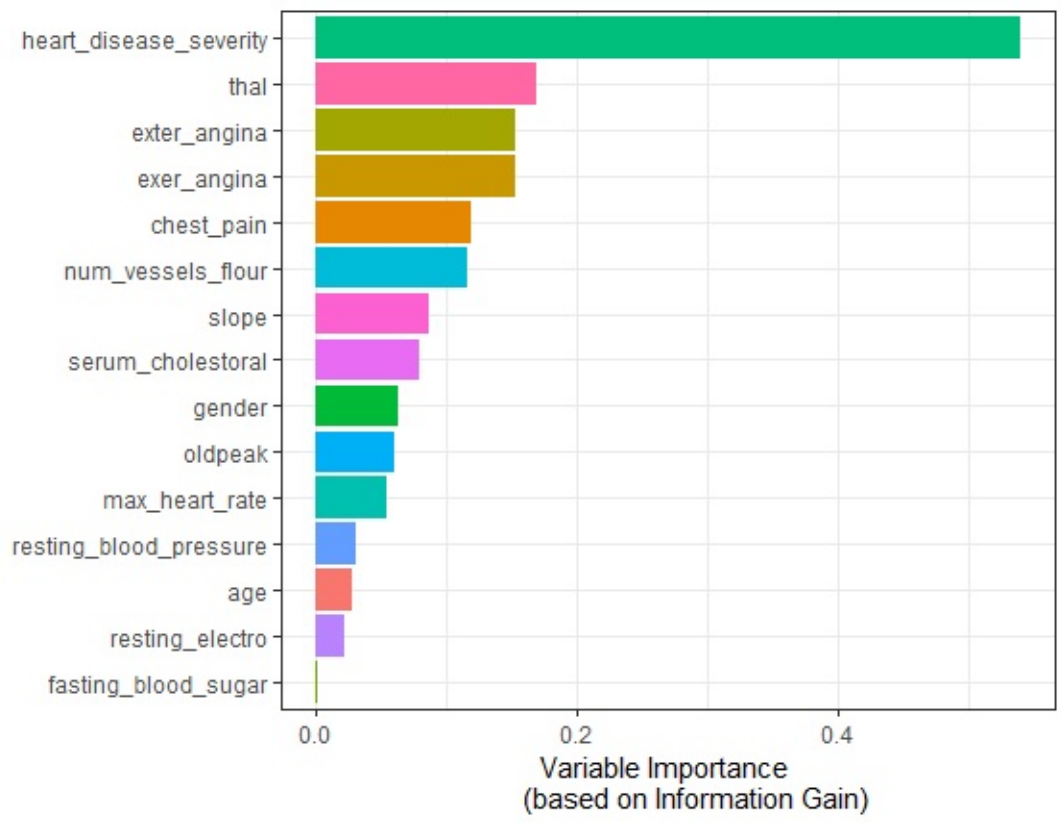


Figure 6.6: Importancia de las variables (basada en la proporción de ganancia)

¿Es `heart_disease_severity` el factor que explica en mayor medida el objetivo?

No, esta variable fue utilizada para generar el objetivo, por lo que debemos excluirla. Es un error típico cuando desarrollamos un modelo predictivo tener o una variable de entrada que fue construida de la misma manera que el objetivo (como en este caso) o agregar variables del futuro como explicamos en Consideraciones con respecto al tiempo.

Volviendo al resultado de `var_rank_info`, las métricas resultantes vienen de la Teoría de la información:

- `en`: entropía medida en bits
- `mi`: información mutual (mutual information)
- `ig`: ganancia de información (information gain)
- `gr`: proporción de ganancia (gain ratio)

En este punto no vamos a detallar qué hay detrás de estas métricas dado que lo

cubriremos exclusivamente en un capítulo más adelante. Sin embargo, `gain ratio` es la métrica más importante aquí, cuyo valor puede ser entre 0 y 1, donde un valor más alto es mejor.

Límites difusos

Acabamos de ver cómo calcular la importancia basándonos en métricas de Teoría de la información. Este tema no es exclusivo de este capítulo; este concepto también aparece en la sección Análisis exploratorio de datos - Correlación y relación.

Seleccionar los mejores factores se relaciona con el *análisis exploratorio de datos* y viceversa.

6.8 Correlación entre variables de entrada

El escenario ideal es construir un modelo predictivo sólo con variables que no estén correlacionadas entre sí. En la práctica, es complicado mantener este escenario para todas las variables.

Generalmente habrá un conjunto de variables que no están correlacionadas entre sí, pero también habrá otras que tengan al menos alguna correlación.

En la práctica una solución adecuada sería excluir aquellas variables que tengan una correlación **notablemente alta**.

Sobre cómo medir la correlación. Los resultados pueden ser muy diferentes según se trate de procedimientos lineales o no lineales. Encontrarán más información en la sección de Correlación.

¿Cuál es el problema de agregar variables correlacionadas?

El problema es que estamos agregando complejidad al modelo: normalmente lleva más tiempo, es más difícil de entender, menos explicable, menos preciso, etc. Este es un efecto que repasamos en ¿Los modelos predictivos pueden manejan la alta cardinalidad? Parte 2. La regla general sería: Intenten agregar las N variables principales que están correlacionadas con el resultado pero no entre

ellas. Esto nos lleva a la siguiente sección.

6.9 Manténganlo simple

Figure 6.7: Fractales en la naturaleza

> La naturaleza opera de la manera más corta posible. -Aristóteles.

El principio de **la Navaja de Ockham**: Entre distintas hipótesis, deberíamos elegir la que tenga menos supuestos.

Si reinterpretamos esta oración para aplicarla a machine learning, esas hipótesis pueden ser vistas como variables, por lo que tenemos:

Entre distintos modelos predictivos, deberíamos elegir el que tenga menos variables. (Wikipedia 2017c)

Por supuesto, también está el equilibrio entre agregar-quitar variables y la precisión del modelo.

Un modelo predictivo con un *alto* número de variables tenderá a **sobreajustar**. Mientras que, por otro lado, un modelo con un *bajo* número de variables tenderá a **subajustar**.

El concepto de *alto* y *bajo* es **altamente subjetivo** a los datos que están siendo analizados. En la práctica, podemos tener alguna métrica de precisión, como por ejemplo, el valor de ROC. Es decir, veríamos algo como:

Variables	ROC
5	0.6188
10	0.8194
20	0.9324
30	0.9500
58	0.9392

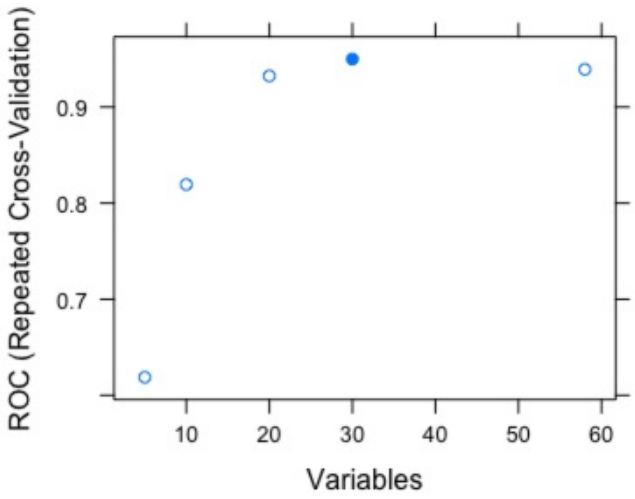


Figure 6.8: Valores de ROC para diferentes subconjuntos de variables

El último gráfico muestra la métrica de precisión ROC de distintos subconjuntos de variables (5, 10, 20, 30 y 58). Cada punto representa el valor ROC dado un determinado número de variables utilizadas para construir el modelo.

Podemos verificar que el valor más alto de ROC aparece cuando el modelo es construido con 30 variables. Si basáramos la selección solamente en un proceso automatizado, podríamos terminar eligiendo un subconjunto que tienda a sobreajustar. Este informe fue producido por la biblioteca `caret` en R ((Kuhn 2017) pero es análogo para cualquier software.

Observando más de cerca la diferencia entre el subconjunto de 20 y el de 30; hay sólo una mejora de **1.8%** -de 0.9324 a 0.95- tomando **10 variables más**. En otras palabras: *Tomar 50% más variables significará menos de 2% de mejora.*

Incluso, este 2% podría ser el margen de error debido a la varianza en la predicción que todo modelo predictivo tiene, como veremos en el capítulo Conociendo el error.

Conclusión:

En este caso, y siendo consecuentes con el principio de la Navaja de Ockham, la mejor solución es construir el modelo con el subconjunto de 20 variables.

Explicar a terceros -y entender- un modelo con 20 variables es más fácil que uno

similar pero con 30 variables.

6.10 ¿Selección de variables en agrupamiento (clustering)?

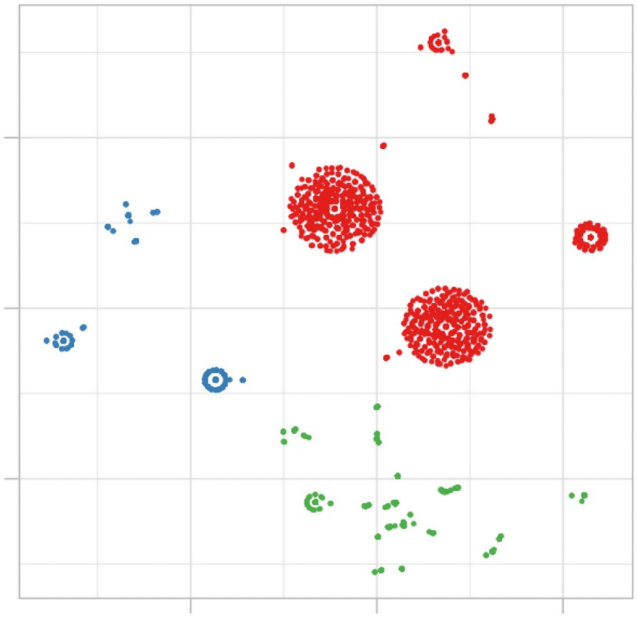


Figure 6.9: Ejemplo de agrupación en segmentos

Este concepto suele aparecer sólo en el modelado predictivo, es decir, tener algunas variables para predecir un objetivo. En la agrupación no hay una variable objetivo, dejamos que los datos hablen, y los grupos naturales surgen de acuerdo a alguna métrica de distancia.

Sin embargo, **no todas las variables contribuyen de la misma manera a la diferencia en el modelo de clusters**. Siendo breves, si tenemos 3 clusters como resultado, y medimos el promedio de cada variable, esperamos que estos promedios sean *bastante* diferentes entre sí, ¿cierto?

Habiendo construido 2 modelos de clusters, en el primero los promedios de la variable `age` son 24, 33 y 26 años; mientras que en el segundo tenemos: 23, 31 y 46. En el segundo modelo la variable `age` está teniendo más variabilidad, por lo

que es más relevante para el modelo.

Este fue sólo un ejemplo considerando dos modelos, pero es lo mismo considerando sólo uno. Aquellas variables con **más distancia** entre promedios tenderán a **definir mejor** el cluster que las otras.

A diferencia del modelado predictivo, en el agrupamiento las variables *menos importantes* no deben ser eliminadas, esas variables no son importantes en ese modelo en particular, pero podrían serlo si construimos otro con otros parámetros. La calidad de los modelos de clusters es muy subjetiva.

Finalmente, podríamos ejecutar, por ejemplo, un modelo de bosque aleatorio con el cluster como variable objetivo y de esta manera identificar rápidamente las variables más importantes.

6.11 Seleccionar las mejores variables en la práctica

6.11.1 La respuesta corta

Tomen las *N* variables superiores del algoritmo que están usando y luego reconstruyan el modelo con este subconjunto. No todos los modelos predictivos extraen las clasificaciones de las variables, pero si lo hace, utilicen el mismo modelo (por ejemplo, gradient boosting machine) para obtener la clasificación y construir el modelo final.

Para aquellos modelos como k-nearest neighbors, que no tienen incorporado un procedimiento de selección de mejores factores, es válido utilizar la selección de otro algoritmo. Esto conducirá a mejores resultados que el uso de todas las variables.

6.11.2 La respuesta larga

- Cuando sea posible, **validen** la lista con alguien que conozca el contexto, la industria o el origen de los datos. Ya sea para las *N* variables superiores o las *M* variables inferiores. Con respecto a aquellas *malas* variables, puede

que se nos esté escapando algo en el procesamiento de datos que esté destruyendo su poder predictivo.

- Entiendan cada variable, su significado en el contexto (industria, medicina, otros).
- Hagan un **análisis exploratorio de datos** para ver las distribuciones de las variables más importantes con respecto a la variable objetivo, *¿la selección tiene sentido?* Si el objetivo es binario, entonces pueden utilizar la función cross_plot.
- ¿Hay algún promedio de alguna variable que cambie *significativamente* a lo largo del tiempo? Revisen si hay cambios abruptos en las distribuciones.
- Sospechen de las variables con alta clasificación y alta cardinalidad (como código postal, digamos con más de 100 categorías). Hay más información en Alta cardinalidad en modelado predictivo.
- Cuando estén haciendo la selección -al igual que el modelado predictivo-, traten de usar métodos que tengan algún mecanismo de remuestreo (como *bootstrapping*), y validación cruzada. Hay más información en el capítulo Conociendo el error.
- Prueben otros métodos para encontrar **grupos de variables**, como el que mencionamos antes: mRMR.
- Si la selección no se ajusta a las necesidades, prueben creando nuevas variables. Pueden referirse al capítulo de **preparación de datos**. Muy pronto: un capítulo sobre ingeniería de factores.

6.11.3 Generen su propio conocimiento

Es difícil generalizar cuando la naturaleza de los datos es tan diferente, desde la **genética** en la que hay miles de variables y unas pocas filas, hasta la navegación web donde llegan nuevos datos todo el tiempo.

Lo mismo aplica al objetivo del análisis. ¿Se utilizará en una competición en la que la precisión es muy necesaria? Tal vez la solución pueda incluir más variables correlacionadas en comparación con un estudio ad-hoc en el que el objetivo principal sea una explicación simple.

No hay una respuesta única para hacer frente a todos los desafíos posibles; ustedes encontrarán poderosas ideas y revelaciones usando su experiencia. Es sólo cuestión de práctica.

6.12 Análisis del objetivo

6.12.1 Usando `cross_plot` (dataViz)

6.12.1.1 ¿De qué se trata esto?

Este gráfico busca mostrar en escenarios reales si una variable es importante o no, haciendo un resumen visual de la misma, *(agrupando variables numéricas en segmentos)*.

6.12.1.2 Ejemplo 1: ¿El género está correlacionado con las enfermedades cardíacas?

```
cross_plot(heart_disease, input="gender", target="has_heart_disease"
```

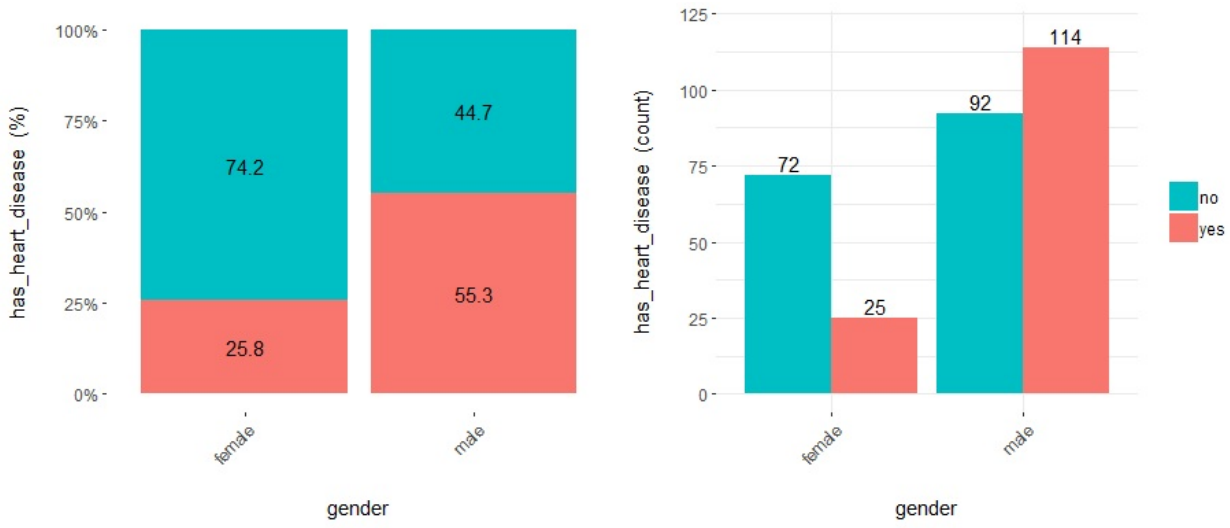


Figure 6.10: Usando cross-plot para analizar y reportar la importancia de las variables

Los últimos dos gráficos comparten la misma fuente de datos, y muestran la distribución de `has_heart_disease` con respecto a `gender`. El gráfico de la izquierda la muestra en valor porcentual, mientras el de la derecha la muestra en valor absoluto.

6.12.1.2.1 ¿Cómo extraer conclusiones de los gráficos? (Versión corta)

La variable `Gender` parece ser un **buen predictor**, dado que la probabilidad de tener una enfermedad cardíaca es diferente para los grupos mujer/hombre. **Le da un orden a los datos**.

6.12.1.3 ¿Cómo extraer conclusiones de los gráficos? (Versión larga)

Del 1er gráfico (%):

1. La **probabilidad** de tener una enfermedad cardíaca para los hombres es de 55.3%, mientras para las mujeres es: 25.8%.
2. La tasa de enfermedad cardíaca en los hombres es el **doble** de la tasa en mujeres (55.3 vs. 25.8, respectivamente).

Del 2do gráfico (cantidad):

1. Hay un total de **97 mujeres**:

    - 25 de ellas tienen una enfermedad cardíaca (25/97=25.8%, que es la proporción del 1er gráfico).
    - las 72 restantes no tienen una enfermedad cardíaca (74.2%)
2. Hay un total de **206 hombres**:
    - 114 de ellos tienen una enfermedad cardíaca (55.3%)
    - los 92 restantes no tienen una enfermedad cardíaca (44.7%)
3. Casos totales: Sumar los valores de las cuatro barras: 25+72+114+92=**303**.

Nota: ¿Qué hubiera pasado si en lugar de tener tasas de 25.8% vs. 55.3% (mujer vs. hombre), hubieran tenido tasas más similares, como 30.2% vs. 30.6%? En ese caso, la variable `gender` *hubiera sido mucho menos relevante, dado que no separa el evento* `has_heart_disease`.

6.12.1.4 Ejemplo 2: Cruzando con variables numéricas

Las variables numéricas deberían estar **segmentadas** para graficarlas con un histograma, si no, el gráfico no muestra información, como podemos ver aquí:

6.12.1.4.1 Segmentación por igual frecuencia

Hay una función incluida en el paquete (heredada del paquete Hmisc): `equal_freq`, que devuelve los segmentos basándose en el **criterio de igual frecuencia**, que tiene -*o trata de tener*- la misma cantidad de filas por segmento.

Para variables numéricas, `cross_plot` tiene por defecto `auto_binning=T`, que automáticamente ejecuta la función `equal_freq` con `n_bins=10` (o el número más cercano).

```
cross_plot(heart_disease, input="max_heart_rate", target="has_heart_
```

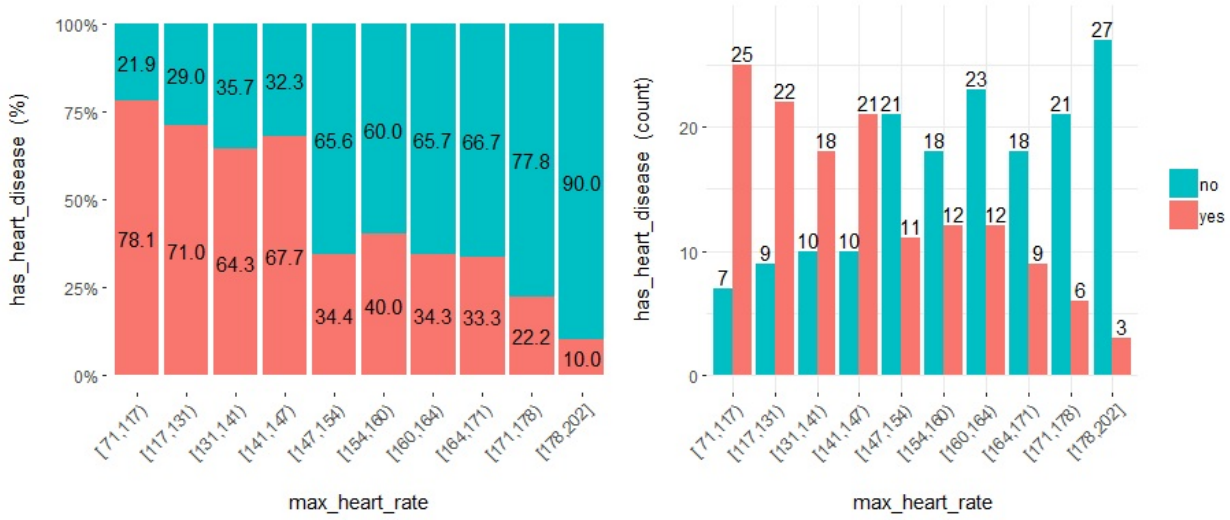


Figure 6.11: Variable numérica como dato de entrada (segmentación automática)

6.12.1.5 Ejemplo 3: Segmentación manual

Si no quieren la segmentación automática, entonces configuren `auto_binning=F` en la función `cross_plot`.

Por ejemplo, crear `oldpeak_2` basada en igual frecuencia, con tres segmentos.

```
heart_disease$oldpeak_2 =
  equal_freq(var=heart_disease$oldpeak, n_bins = 3)
summary(heart_disease$oldpeak_2)
```

```
## [0.0,0.2) [0.2,1.5) [1.5,6.2]
##       106       107        90
```

Graficar la variable segmentada (`auto_binning = F`):

```
cross_oldpeak_2=cross_plot(heart_disease, input="oldpeak_2", target=
```

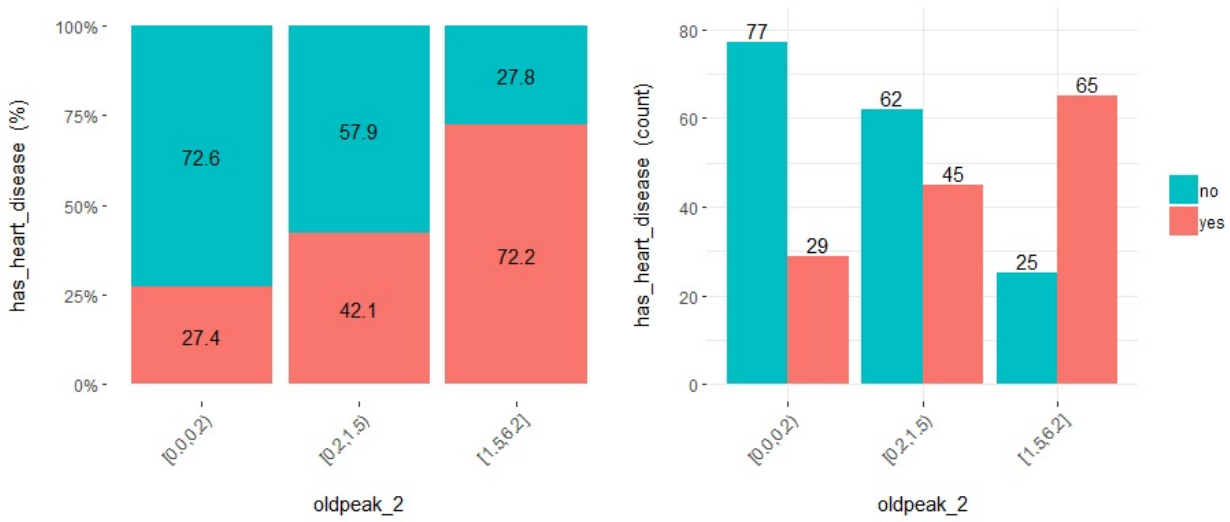


Figure 6.12: Al desactivar la segmentación automática vemos la variable original

6.12.1.5.1 Conclusión

Este nuevo gráfico basado en `oldpeak_2` muestra claramente cómo la probabilidad de **tener una enfermedad cardíaca aumenta** mientras que **oldpeak_2 aumenta** también. *Nuevamente, da un orden a los datos.*

6.12.1.6 Ejemplo 4: Reducir el ruido

Convertir la variable `max_heart_rate` en una de 10 segmentos:

```
heart_disease$max_heart_rate_2 =
  equal_freq(var=heart_disease$max_heart_rate, n_bins = 10)

cross_plot(heart_disease,
           input="max_heart_rate_2",
           target="has_heart_disease"
           )
```

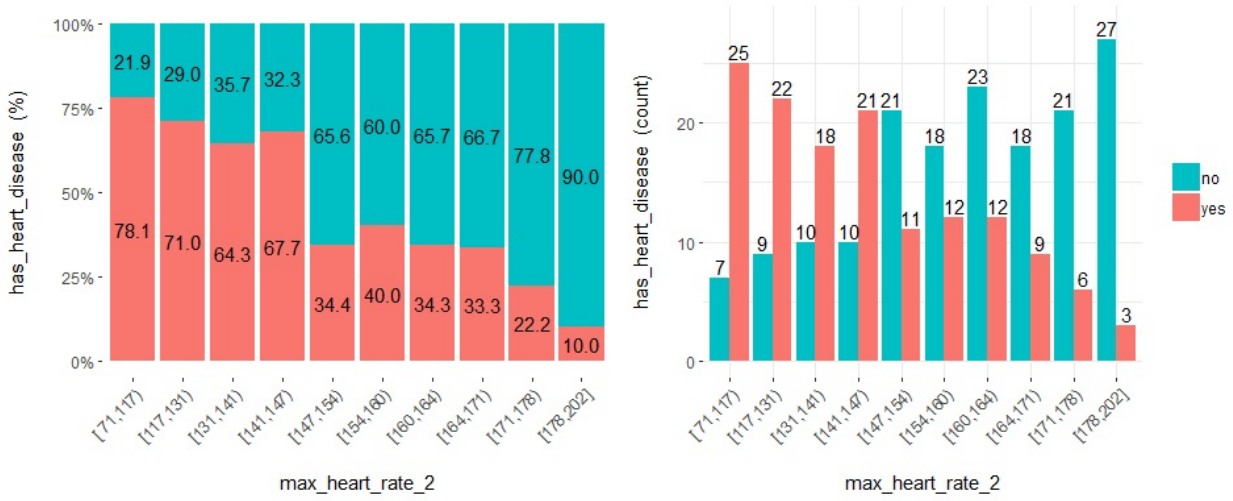


Figure 6.13: Graficar usando segmentación personalizada

A simple vista, `max_heart_rate_2` muestra una relación negativa y lineal. Sin embargo, hay algunos segmentos que agregan ruido a la relación. Por ejemplo, el segmento `(141, 147]` tiene una tasa de enfermedad cardíaca más alta que el segmento anterior, y se esperaba que tuviera una más baja. *Esto podría ser ruido en los datos.*

Nota clave: Una forma de reducir el **ruido** (a costa de **perder** información), es dividir en menos segmentos:

```
heart_disease$max_heart_rate_3 =
  equal_freq(var=heart_disease$max_heart_rate, n_bins = 5)

cross_plot(heart_disease,
           input="max_heart_rate_3",
           target="has_heart_disease"
           )
```

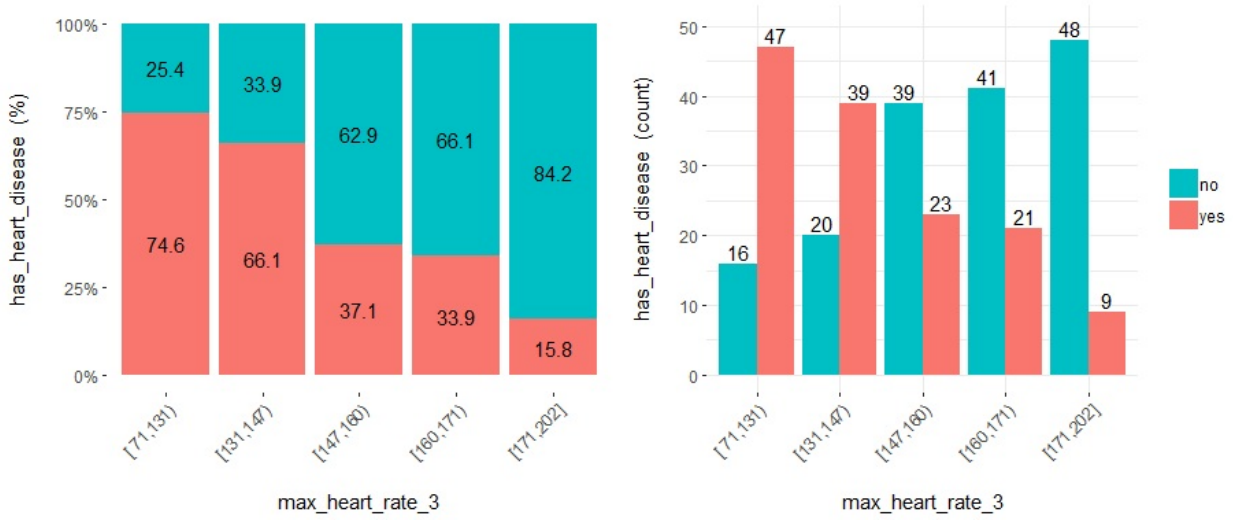


Figure 6.14: Reducir la cantidad de segmentos puede ayudar a exponer mejor la relación

Conclusión: Como podemos ver, ahora la relación es mucho más limpia y clara. El segmento *'N'* tiene una tasa más alta que *'N+1'*, lo que implica una correlación negativa.

¿Y si guardamos el resultado de cross_plot en una carpeta?

Simplemente creen el parámetro `path_out` con la carpeta que quieran -Crea una nueva si no existe la que ingresaron-.

```
cross_plot(heart_disease, input="max_heart_rate_3", target="has_hear
```

Este comando crea la carpeta `my_plots` en el directorio en uso.

6.12.1.7 Ejemplo 5: `cross_plot` en múltiples variables

Imaginen que quieren ejecutar cross_plot en una cantidad de variables al mismo tiempo. Para lograr esto sólo es necesario definir un vector que contenga los nombres de las variables.

Si desean analizar estas 3 variables:

```
vars_to_analyze=c("age", "oldpeak", "max_heart_rate")

cross_plot(data=heart_disease, target="has_heart_disease", input=var
```

6.12.1.8 Exportar gráficos

Las funciones `plotar` y `cross_plot` pueden manejar de 1 a N variables de entrada, y los gráficos que se generan a partir de ellas pueden exportarse fácilmente en alta calidad con el parámetro `path_out`.

```
plotar(data=heart_disease, input=c('max_heart_rate', 'resting_blood_
```

6.12.2 Usando boxplots

6.12.2.1 ¿De qué se trata esto?

El uso de boxplots en el análisis de importancia de variables brinda una vistazo rápido de cuán diferentes son los cuartiles entre los distintos valores de una variable objetivo binaria.

```r
# ¡Cargar funModeling!
library(funModeling)
data(heart_disease)

plotar(data=heart_disease, input="age", target="has_heart_disease",
```

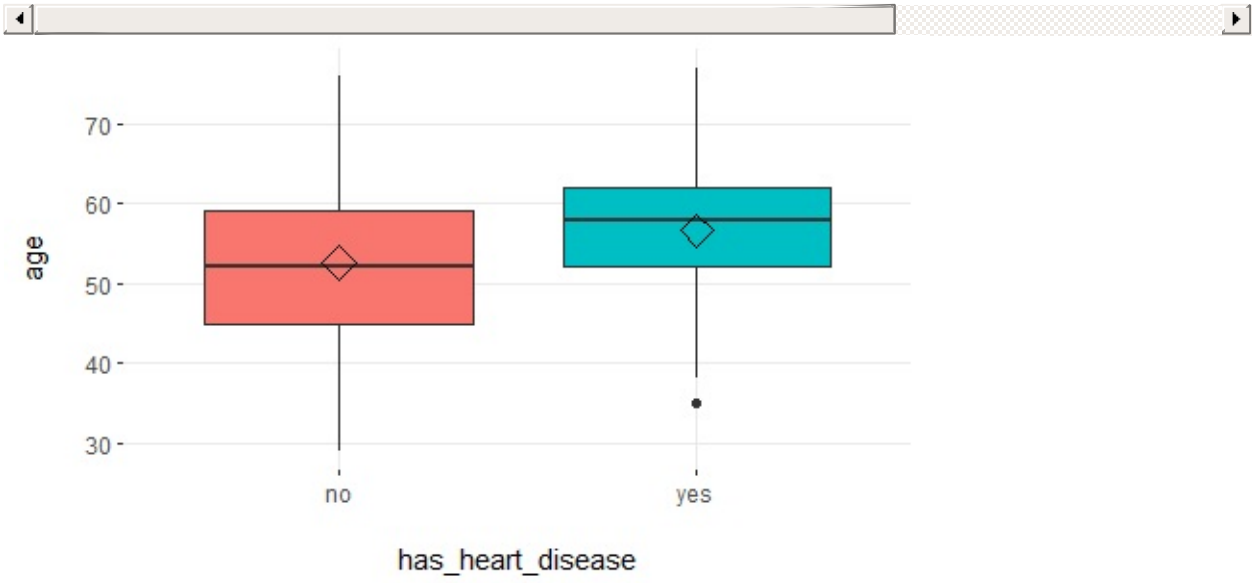


Figure 6.15: Análisis numérico de la variable objetivo usando boxplots

*El romboide cerca de la línea de promedio representa la **mediana**.*

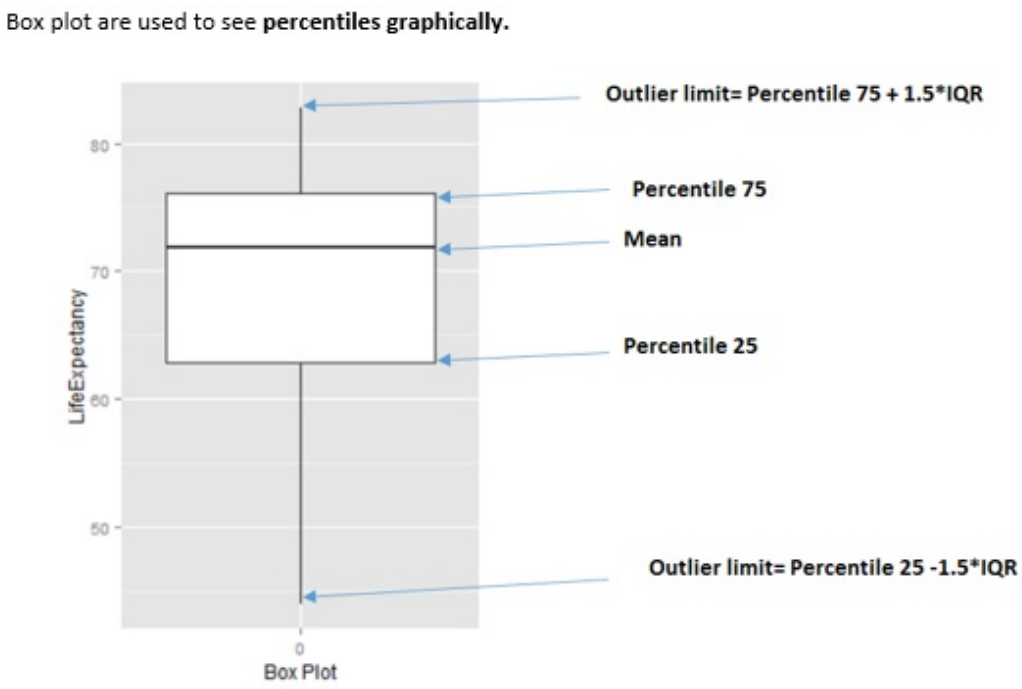


Figure 6.16: Cómo interpretar un diagrama de caja

¿Cuándo usar diagramas de caja?

Cuando necesitamos analizar diferentes percentiles entre las clases para predecir. Noten que esta es una técnica poderosa ya que el sesgo producido debido a los valores atípicos no afecta tanto como afecta al promedio.

6.12.2.2 Boxplot: Variable buena vs. variable mala

Usar más de una variable de entrada es útil para comparar rápidamente los diagramas de caja, y así obtener las mejores variables...

```
plotar(data=heart_disease, input=c('max_heart_rate', 'resting_blood_
```

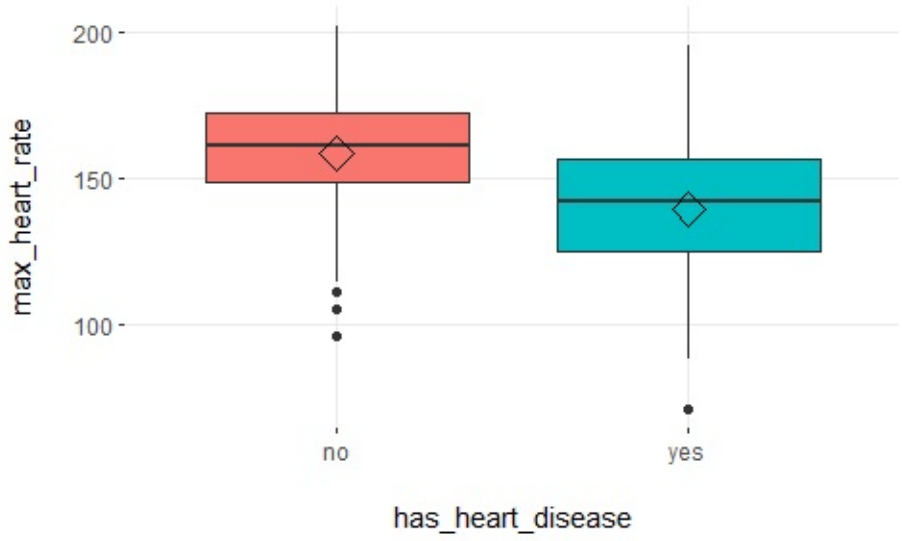


Figure 6.17: Función plotar para variables múltiples

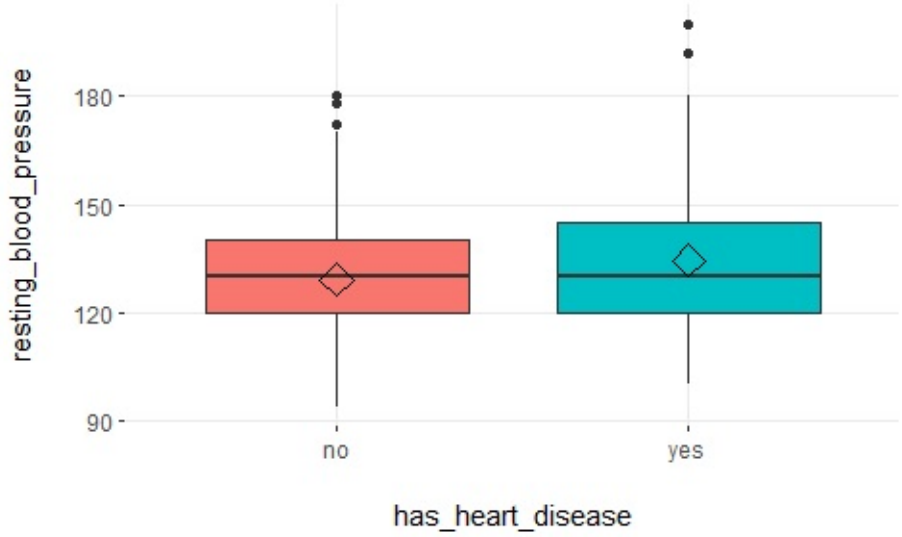


Figure 6.18: Función plotar para variables múltiples

Podemos concluir que `max_heart_rate` es un mejor predictor que `resting_blood_pressure`.

Como regla general, una variable se clasificará como **más importante** si los boxplots **no están alineados** horizontalmente.

Pruebas estadísticas: los percentiles son otra característica utilizada por ellos para determinar -por ejemplo- si los promedios entre grupos son o no los

mismos.

6.12.2.3 Exportando gráficos

`plotar` y `cross_plot` pueden manejar desde 1 hasta N variables de entrada, y los gráficos que generan pueden exportarse fácilmente en alta calidad con el parámetro `path_out`.

```
plotar(data=heart_disease, input=c('max_heart_rate', 'resting_blood_
```

- **Tengan esto en mente cuando usen diagramas de caja** Son agradables a la vista cuando la variable:
  - Tiene una buena dispersión -no está concentrada en *3*, *4..6*.. valores distintos, **y**
  - No tiene valores atípicos extremos... *(este punto se puede tratar con la función `prep_outliers`, incluida en este paquete)*

6.12.3 Usando histogramas de densidad

6.12.3.1 ¿De qué se trata esto?

Los histogramas de densidad son bastante estándar en cualquier libro/recurso cuando se grafican las distribuciones. Usarlas en la selección de variables brinda un vistazo rápido de lo bien que ciertas variables separan la clase.

```
plotar(data=heart_disease, input="age", target="has_heart_disease",
```

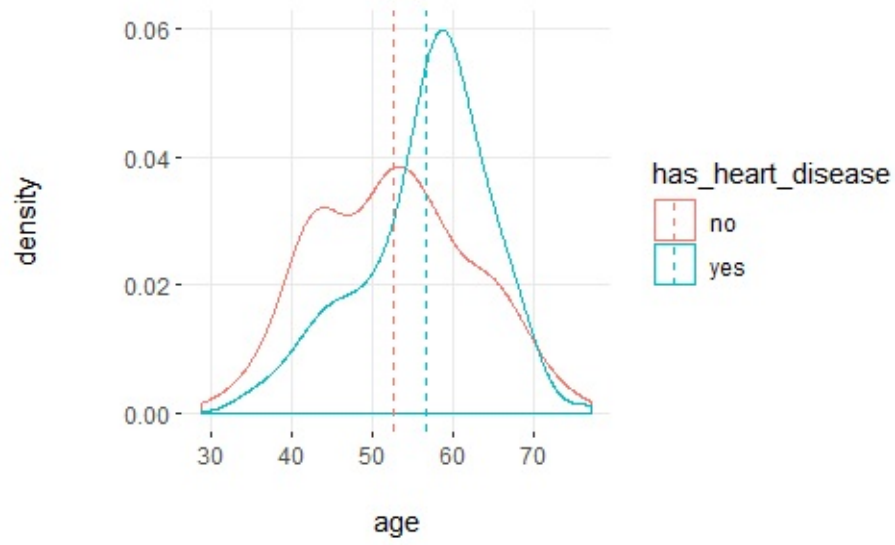


Figure 6.19: Análisis numérico de la variable objetivo usando histogramas de densidad

Nota: La línea punteada representa el promedio de la variable.

Los **histogramas de densidad** son útiles para visualizar la forma general de una distribución numérica.

Esta *forma general* se calcula en base a una técnica llamada **Kernel smoother (Suavizador Kernel)**, su idea general es reducir los picos altos/bajos (ruido) presentes en puntos/barras cercanos estimando la función que describe los puntos. Aquí hay algunas imágenes para ilustrar el concepto: https://en.wikipedia.org/wiki/Kernel_smoother

6.12.3.2 ¿Cuál es la relación con una prueba estadística?

Algo similar es lo que ve una **prueba estadística**: miden **cuán diferentes** son las curvas que la reflejan en algunas estadísticas como el p-value que se utiliza en el enfoque del frecuentista. Proporciona al analista información confiable para determinar si las curvas tienen -por ejemplo- el mismo promedio.

6.12.3.3 Variable buena vs. variable mala

```
plotar(data=heart_disease, input=c('resting_blood_pressure', 'max_he
```

Figure 6.20: Función plotar para variables múltiples

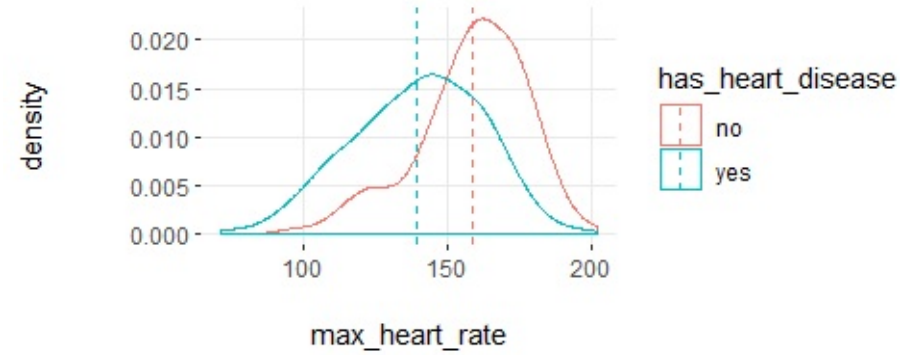


Figure 6.21: Función plotar para variables múltiples

Y el modelo verá lo mismo... si las curvas están bastante superpuestas, como están en `resting_blood_pressure`, entonces **no es un buen predictor**, como sí lo sería si estuvieran **más espaciadas** -como en `max_heart_rate`.

Tengan esto en mente cuando usen histogramas y diagramas de caja Son agradables a la vista cuando la variable: + Tiene una buena dispersión -no está concentrada en *3, 4..6..* valores distintos, **y** + No tiene valores atípicos extremos... *(este punto se puede tratar con la función `prep_outliers`, incluida en este paquete)*

Libro Vivo de Ciencia de Datos

7 Applications

Some *significant* applications are demonstrated in this chapter.

7.1 Example one

7.2 Example two

8 Evaluar el desempeño de un modelo

Este capítulo cubre **aspectos metodológicos del error** en los modelos predictivos, cómo medirlo a través de **validación cruzada** de los datos y su similitud con la técnica de **bootstrapping**. Y cómo estas estrategias son utilizadas internamente por algunos modelos predictivos como *random forests* o *gradient boosting machines*.

También hay un capítulo sobre cómo validar los modelos cuando el tiempo es un factor influyente, que es similar a la validación clásica de entrenamiento/prueba.

8.1 Conociendo el error

Aspectos metodológicos de la validación de modelos

8.1.1 ¿De qué se trata esto?

Una vez que construimos un modelo predictivo, ¿cómo sabemos si su calidad es buena? ¿Capturó -*información*- sobre patrones generales (excluyendo el -*ruido*-)?

8.1.1.1 ¿Qué tipo de datos?

Tiene un enfoque distinto del que veremos en la validación out-of-time. Este enfoque podría utilizarse incluso cuando no sea posible filtrar los casos por fecha, por ejemplo, teniendo un pantallazo de los datos en un momento determinado, cuando no se generará nueva información.

Por ejemplo, una investigación de datos de salud de una cantidad reducida de personas, una encuesta, o algunos datos disponibles en Internet para prácticas. Es caro, poco práctico, poco ético o incluso imposible agregar nuevos casos. Los datos de `heart_disease` que vienen en el paquete `funModeling` son un ejemplo.

8.1.2 Reducir comportamientos inesperados

Cuando un modelo es entrenado, solo ve una parte de la realidad. Es una muestra de una población que no se puede ver entera.

Hay muchas maneras de validar un modelo (Precisión / Curvas ROC / Lift / Ganancia / etc). Cualquiera de estas métricas están **ligadas a la varianza**, lo que implica **obtener valores diferentes**. Si eliminamos algunos casos y luego ajustamos un nuevo modelo, veremos un valor *ligeramente* diferente.

Imaginemos que construimos un modelo y logramos una precisión de `81`, ahora quitamos el 10% de los casos, y luego ajustamos uno nuevo, la precisión ahora es: `78.4`. ¿Cuál es la precisión real? ¿La que se obtiene con el 100% de los datos o la que se basa en el 90%? Por ejemplo, si el modelo se ejecuta en vivo en un entorno de producción, verá **otros casos** y el punto de precisión se moverá hacia uno nuevo.

Entonces, ¿cuál es el valor real, el que reportaremos? Las técnicas de **remuestreo** y **validación cruzada** se promedian -basándose en diferentes criterios de muestreo y prueba- con el fin de obtener una aproximación al valor más confiable.

Pero, ¿por qué quitar casos?

No tiene sentido eliminar casos como ese, pero da una idea de cuán sensible es la métrica de precisión, recuerden que estamos trabajando con una muestra de una población desconocida.

Si tuviéramos un modelo totalmente determinista, un modelo que contenga el 100% de todos los casos que estamos estudiando, y las predicciones fueran 100% exactas en todos los casos, no necesitaríamos todo esto.

En la medida en que siempre analizamos las muestras, sólo necesitamos acercarnos a la *verdad real y desconocida* de los datos a través de la repetición, el remuestreo, la validación cruzada, etc...

8.1.3 Ilustremos esto con Cross Validation (CV)

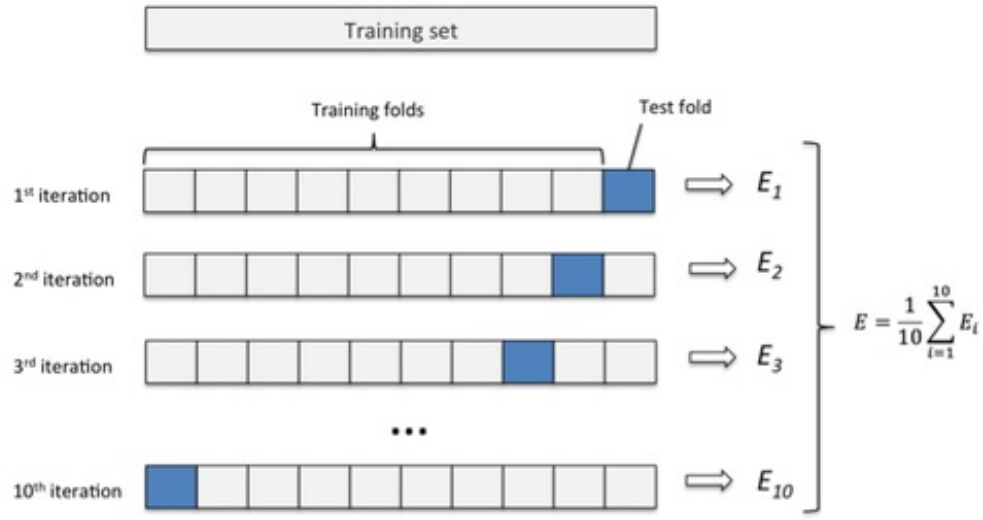


Figure 8.1: k-fold Cross Validation

Crédito de la imagen: Sebastian Raschka Ref. (Raschka 2017)

8.1.3.1 Breve resumen sobre CV

- Divide los datos en grupos aleatorios, digamos `10`, de igual tamaño. Estos

grupos son comúnmente llamados `folds` o pliegues, representados por la letra `'k'`.

- Tomen `9` pliegues, construyan un modelo, y luego apliquen el modelo al pliegue restante (el que dejaron fuera). Esto nos devolverá la métrica de precisión que queremos: precisión (accuracy), ROC, Kappa, etc. En este ejemplo estamos usando la precisión.
- Repitan esto `k` veces (`10` en nuestro ejemplo). Así conseguiremos `10` precisiones diferentes. El resultado final será el promedio de todos ellos.

Este promedio será el que evalúe si un modelo es bueno o no, y también si incluirlo o no en un informe.

8.1.3.2 Ejemplo práctico

Hay 150 filas en el data frame `iris`, usar el [paquete caret] para construir un `random forest` con `caret` usando `cross-validation` llevará a la construcción -interna- de 10 bosques aleatorios, cada uno basado en 135 filas (9/10 * 150), y reportando una precisión basada en los 15 casos restantes (1/10 * 150). Este procedimiento se repite 10 veces.

Esta parte del resultado:

```
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 135, 135, 135, 135, 135, 135, ...
Resampling results:

  Accuracy   Kappa
  0.9466667  0.92
```

Figure 8.2: Resultado de la validación cruzada de caret

`Summary of sample sizes: 135, 135, 135, 135, 135, 135, ...`, cada 135 representa una muestra de entrenamiento, 10 en total pero el resultado está truncado.

En lugar de un solo número -el promedio-, podemos ver una distribución:

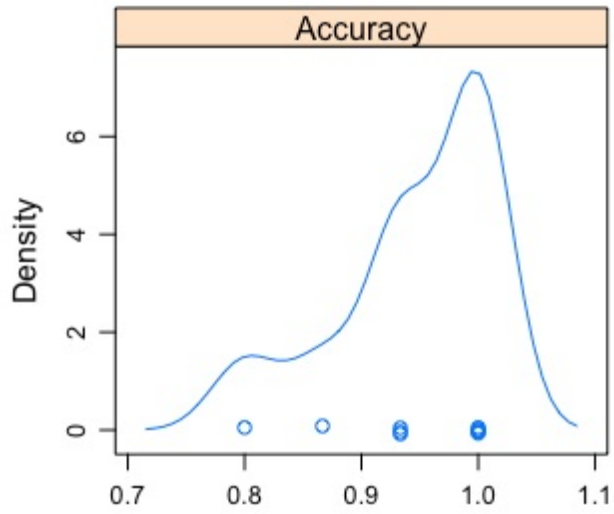


Figure 8.3: Análisis visual de la distribución de la precisión

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.8667  0.9333  0.9333  0.9467  1.0000  1.0000
```

Figure 8.4: Distribución de la precisión

- La precisión mín/máx estará entre `~0.8` y `~1`.
- El promedio es el que reportó `caret`.
- El 50% de las veces estará en el rango entre `~0.93 y ~1`.

Lectura recomendada por Rob Hyndman, creador del paquete `forecast`: *Why every statistician should know about cross-validation?* (Hyndman 2010)

8.1.4 Pero, ¿qué es el error?

La suma del **Sesgo**, **Varianza** y el ***error no explicado*** -ruido interno- en los datos, o el que el modelo jamás podrá reducir.

Estos tres elementos constituyen el error reportado.

8.1.4.1 ¿Cuál es la naturaleza del Sesgo (bias) y la Varianza?

Cuando el modelo no funciona bien, puede haber varias causas:

- **Modelo demasiado complicado**: Digamos que tenemos muchas variables de entrada, que se relaciona con una **alta varianza**. El modelo sobreajustará los datos de entrenamiento, teniendo una baja precisión sobre datos nunca vistos debido a su particularización.
- **Modelo demasiado sencillo**: Por otro lado, puede que el modelo no esté capturando toda la información que hay en los datos debido a su simpleza. Esto está relacionado con un **alto sesgo**.
- **Datos de entrada insuficientes**: Los datos crean formas en un espacio n-dimensional (donde `n` es todas las variables de entrada+objetivo). Si no hay suficientes puntos, esta forma no se desarrolla bien.

Hay más información sobre esto en *"In Machine Learning, What is Better: More Data or Better Algorithms"* (Amatriain 2015).

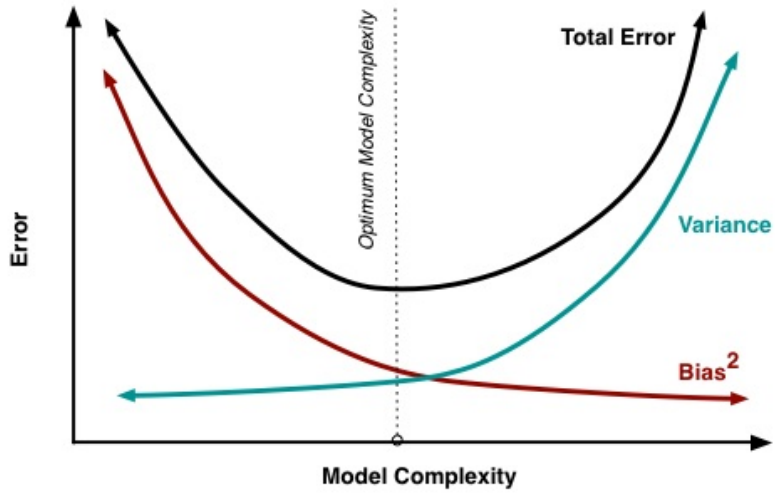


Figure 8.5: Equilibrio entre sesgo y varianza

Crédito de la imagen: Scott Fortmann-Roe (Fortmann 2012). También contiene una forma intuitiva de entender el error a través del sesgo y la varianza mediante una animación.

8.1.4.2 Equilibrio entre complejidad y precisión

El sesgo y la varianza están relacionados en el sentido de que si uno baja, el otro sube, por lo que hay un **equilibrio** entre ellos. Un ejemplo práctico de esto se ve en la medida de calidad de modelos llamada Criterio de Información de Akaike (AIC por Akaike Information Criterion, en inglés).

El **AIC** se utiliza como heurística para elegir el mejor **modelo de series temporales** en la función `auto.arima` dentro del paquete `forecast` en `R` (Hyndman 2017). Elige el modelo que tenga el AIC más bajo.

Cuanto más bajo, mejor: La precisión en la predicción reducirá el valor, mientras que el número de parámetros lo aumentará.

8.1.4.3 Bootstrapping vs Validación cruzada

- **Bootstrapping** suele utilizarse para estimar un parámetro.
- **Validación cruzada** se utiliza para elegir entre distintos modelos predictivos.

Nota: Para un abordaje más profundo sobre sesgo y varianza, por favor refiéranse a (Fortmann 2012) y (Amatriain 2015) al final de la página.

8.1.5 ¿Algún consejo para la práctica?

Depende de los datos, pero es común encontrar ejemplos como `10 fold CV`, más repetición: `10 fold CV, repeated 5 times`. Otras veces nos encontramos con: `5 fold CV, repeated 3 times`.

Y usando el promedio de la métrica deseada. También es recomendable usar el `ROC` porque está menos sesgado a las variables objetivo desequilibradas.

Dado que estas técnicas de validación **consumen mucho tiempo**, consideren la posibilidad de elegir un modelo que se ejecute rápidamente, permitiendo el ajuste del modelo, probando diferentes configuraciones, probando diferentes

variables en "poco" tiempo. Los random forests son una excelente opción que da resultados **rápidos** y **precisos**. Más información sobre el rendimiento general de los bosques aleatorios en (Fernandez-Delgado 2014).

Otra buena opción son las **gradient boosting machines**, tienen más parámetros para ajustar que los random forests, pero al menos en R su implementación funciona rápido.

8.1.5.1 Volviendo al sesgo y la varianza

- Los random forests se enfocan en disminuir el sesgo, mientras que...
- Las gradient boosting machines se enfocan en minimizar la varianza. Hay más información en *"Gradient boosting machine vs random forest"* (stats.stackexchange.com 2015).

8.1.6 No se olviden: Preparación de los datos

Ajustar los datos de entrada transformándolos y limpiándolos tendrá un impacto en la calidad del modelo. A veces más que optimizar el modelo a través de sus parámetros.

Expandan este punto con el capítulo Preparación de datos.

8.1.7 Reflexiones finales

- Validar los modelos mediante el remuestreo / la validación cruzada nos ayuda a estimar el error "real" que existe en los datos. Si el modelo se ejecuta en el futuro, ese es el error que se espera que tenga.
- Otra ventaja es el **ajuste del modelo**, evitando el sobreajuste al seleccionar los mejores parámetros para un determinado modelo, Ejemplo en caret. El equivalente en **Python** está incluido en Scikit Learn.
- La mejor prueba es la que hagan ustedes, adaptada a sus datos y necesidades. Prueben diferentes modelos y analicen el equilibrio entre el consumo de tiempo y cualquier métrica de precisión.

Estas técnicas de remuestreo podrían estar entre las poderosas herramientas detrás de sitios como stackoverflow.com o el software colaborativo de

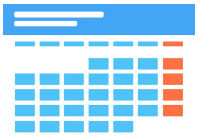
código abierto. Tener muchas opiniones para producir una solución menos sesgada.

Pero cada opinión debe ser confiable, imaginen pedirle un diagnóstico a diferentes médicos.

8.1.8 Lecturas adicionales

- Tutorial: Cross validation for predictive analytics using R
- Tutorial por Max Kahn (creador de caret): Comparing Different Species of Cross-Validation
- El enfoque de validación cruzada también se puede aplicar a los modelos dependientes del tiempo, consulten el otro capítulo: Validación out-of-time.

8.2 Validación out-of-time



8.2.1 ¿De qué se trata esto?

Una vez que hemos construido un modelo predictivo, ¿qué tan seguros estamos de que capturó los patrones generales y no sólo los datos que ha visto (sobreajuste)?

¿Funcionará bien cuando esté en producción / en vivo? ¿Cuál es el error esperado?

8.2.2 ¿Qué tipo de datos?

Si los datos son generados a lo largo del tiempo y -digamos- todos los días tenemos nuevos casos como *"visitas en un sitio web"*, o *"nuevos pacientes que llegan a un centro médico"*, una validación robusta es la del enfoque **Fuera de tiempo**.

8.2.3 Ejemplo de Validación Fuera de tiempo

¿Cómo se hace?

Imaginen que estamos construyendo el modelo un **Jan-01**, entonces para construirlo usamos todos los datos **hasta Oct-31**. Entre estas fechas hay 2 meses.

Al predecir una **variable binaria/de dos clases** (o multiclase), es bastante senscillo: con el modelo que construimos -con datos <= **Oct-31**- le asignamos un score (probabilidad) en ese día exacto, y después medimos cómo los usuarios/pacientes/personas/casos evolucionaron durante esos dos meses.

Dado que el resultado de un modelo binario debería ser un número que indique la probabilidad de cada caso de pertenecer a una determinada clase (capítulo de Scoring de datos, probamos lo que el **modelo "*dijo*" el "Oct-31" contra lo que realmente pasó el "Jan-01"**.

El siguiente **flujo de trabajo de validación** puede resultarles útil a la hora de construir un modelo predictivo que implique tiempo.

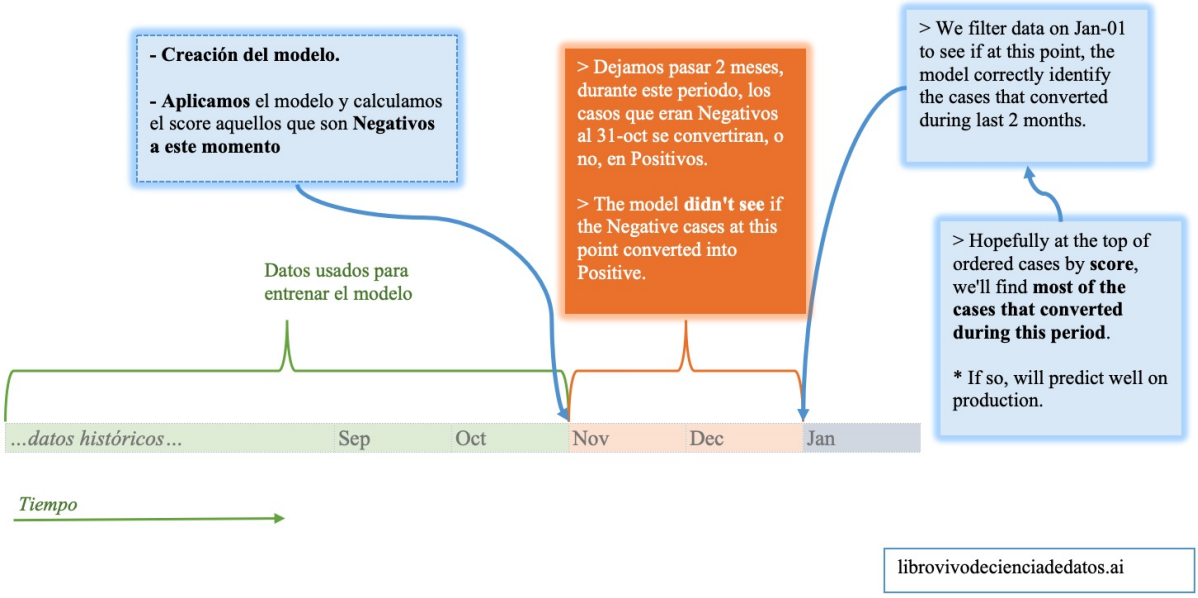


Figure 8.6: Un flujo de trabajo de validación para problemas que dependen del tiempo

Ampliar imagen.

8.2.4 Usando el análisis de Ganancia y Lift

Este análisis está explicado en otro capítulo (Ganancia y Lift) y se puede utilizar después de la validación fuera de tiempo.

Conservando solamente aquellos casos que fueron `negative` en `Oct-31`, obtenemos el `score` que devolvió el modelo en esa fecha, y la variable `target` es el valor que esos casos tuvieron en `Jan-1`.

8.2.5 ¿Qué pasa si la variable objetivo es numérica?

Ahora el sentido común y la necesidad del negocio están más presentes. Un resultado numérico puede tomar cualquier valor, puede aumentar o disminuir con el tiempo, por lo que puede que tengamos que considerar estos dos escenarios para ayudarnos a pensar en lo que consideramos éxito. Tal es el caso de la regresión lineal.

Escenario de ejemplo: Medimos el uso de alguna app (como la de homebanking), lo estándar sería que con el correr de los días los usuarios la utilizen más.

Ejemplos:

- Predecir la concentración de una determinada sustancia en la sangre.
- Predecir visitas en una página web.
- Análisis de series temporales.

En estos casos también tenemos la diferencia entre: **"lo que se esperaba" vs. "lo que es".**

Esta diferencia puede tomar cualquier valor. A este valor lo llamamos error o residuos.

User ID	Page visits on Oct-31	Prediction (on Oct-31) for Jan-01	Real value on Jan-01	Error / Difference / Residuals
1	23	50	63	-13
2	1	66	8	58
3	94	52	43	9
4	34	60	102	-42
...	...	...	...	and so on...

Figure 8.7: Predicción y análisis del error

Si el modelo es bueno, este error debería ser **ruido blanco**, hay más información sobre esto en la sección *"Análisis y regresión de series temporales"* en (Wikipedia 2017d). Sigue una curva normal cuando se cumplen algunas propiedades lógicas:

- El error debería estar **cerca de 0** -*el modelo deberá tender su error a 0*-.
- El desvío estándar de este error **debe ser finito** -para evitar valores atípicos impredecibles-.
- No tiene que haber una correlación entre los errores.
- **Distribución normal**: esperen la mayoría de los errores cerca de 0, con los errores más grandes en una **proporción menor** a medida que el error aumenta -la probabilidad de encontrar errores más grandes disminuye exponencialmente-.

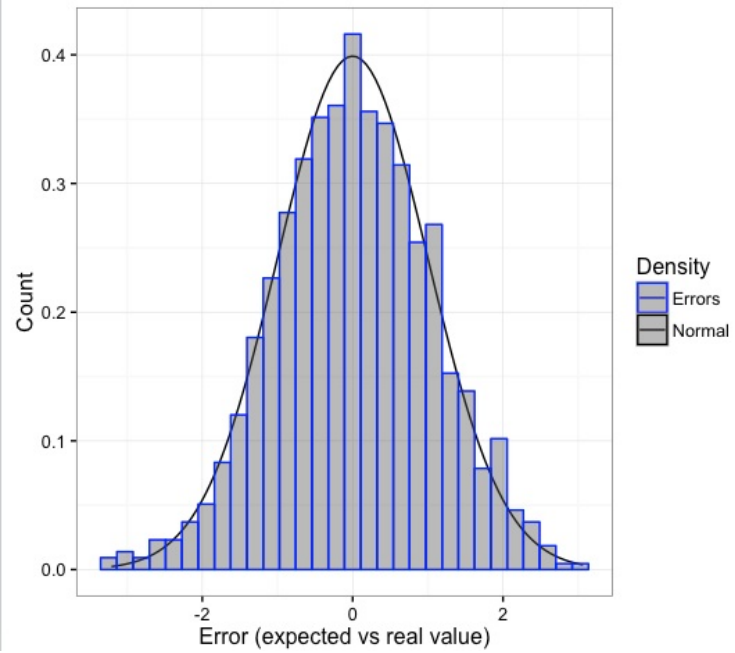


Figure 8.8: Una bella curva de error (distribución normal)

8.2.6 Reflexiones finales

- La **Validación fuera de tiempo** es una poderosa herramienta de validación para simular la ejecución de un modelo en producción con datos que pueden **no necesitar ni depender del muestreo**.

- El **análisis del error** es un capítulo importante en la ciencia de datos. Es hora de pasar al próximo capítulo, que intentará cubrir los conceptos clave de este tema: Conociendo el error.

8.3 Análisis de Ganancia y Lift

8.3.1 ¿De qué se trata esto?

Ambas métricas son extremadamente útiles para validar la calidad del modelo predictivo (resultado binario). Hay más información en Scoring de datos

Asegúrense de tener la última versión de `funModeling` (>= 1.3).

```
# Cargar funModeling
library(funModeling)
```

```
## Warning: package 'funModeling' was built under R version 3.5.3
```

```
## Warning: package 'Hmisc' was built under R version 3.5.3
```

```
## Warning: package 'Formula' was built under R version 3.5.2
```

```
## Warning: package 'ggplot2' was built under R version 3.5.3
```

```
# Crear un modelo GLM
fit_glm=glm(has_heart_disease ~ age + oldpeak, data=heart_disease, f

# Obtener los scores/probabilidades de cada fila
heart_disease$score=predict(fit_glm, newdata=heart_disease, type='re

# Graficar la curva de ganancia y lift
gain_lift(data=heart_disease, score='score', target='has_heart_disea
```

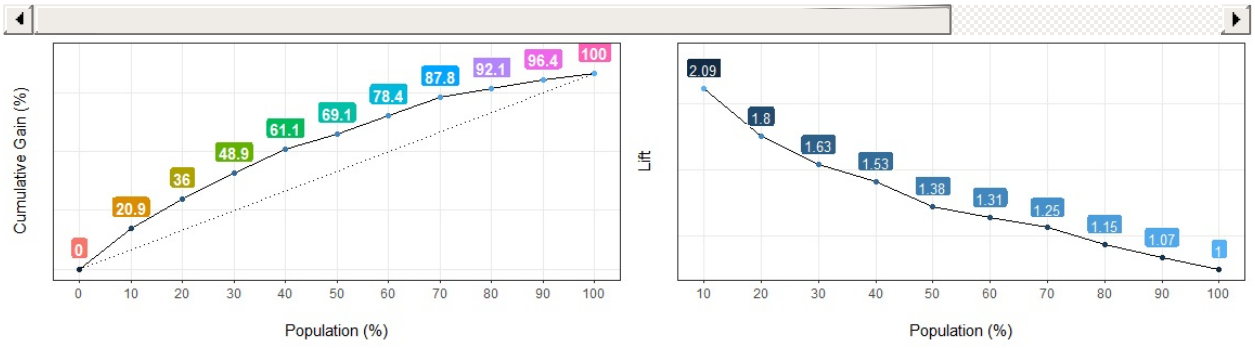


Figure 8.9: Curvas de ganancia y lift

```
##    Population   Gain Lift Score.Point
## 1          10  20.86 2.09   0.8185793
## 2          20  35.97 1.80   0.6967124
## 3          30  48.92 1.63   0.5657817
## 4          40  61.15 1.53   0.4901940
## 5          50  69.06 1.38   0.4033640
## 6          60  78.42 1.31   0.3344170
## 7          70  87.77 1.25   0.2939878
## 8          80  92.09 1.15   0.2473671
## 9          90  96.40 1.07   0.1980453
## 10        100 100.00 1.00   0.1195511
```

8.3.2 ¿Cómo interpretarlo?

Primero, cada caso está ordenado de acuerdo a la probabilidad de ser la clase menos representativa, es decir, según el valor de la puntuación.

Luego, la columna `Gain` acumula la clase positiva, por cada 10% de filas - columna `Population`.

Entonces el análisis de la primera fila sería:

"El primer 10 por ciento de la población, ordenado por score, acumula el 20.86% del total de casos positivos"

Por ejemplo, si estamos enviando e-mails basándonos en este modelo, y tenemos presupuesto para contactar solamente al **20%** de nuestros usuarios, ¿cuántas respuestas esperamos recibir? **Respuesta: 35.97%**

8.3.3 ¿Qué pasa si no usamos un modelo?

Si **no usamos un modelo**, y seleccionamos un 20% aleatoriamente, ¿cuántos usuarios tenemos que contactar? Bueno, 20%. Ese es el significado de la **línea punteada**, que empieza en 0% y termina en 100%. Con un poco de suerte, usando el modelo predictivo le vamos a ganar a la aleatoriedad.

La columna **Lift** representa la proporción entre `Gain` y la *ganancia por azar*. Tomando como ejemplo la Población=20%, el modelo es **1.8 veces mejor** que la aleatoriedad.

8.3.3.1 Usando el punto de corte ✂

¿Qué valor del score alcanza el 30% de la población? Respuesta: `0.56`

El punto de corte nos permite segmentar los datos.

8.3.3.2 Comparando modelos

En un buen modelo, la ganancia alcanzará el 100% "al principio" de la población, representando que separa las clases.

Al comparar modelos, una métrica rápida es ver si la ganancia al principio de la población (10-30%) es mayor.

Como resultado, el modelo con una mayor ganancia al principio habrá capturado más información de los datos.

Vamos a ilustrarlo....

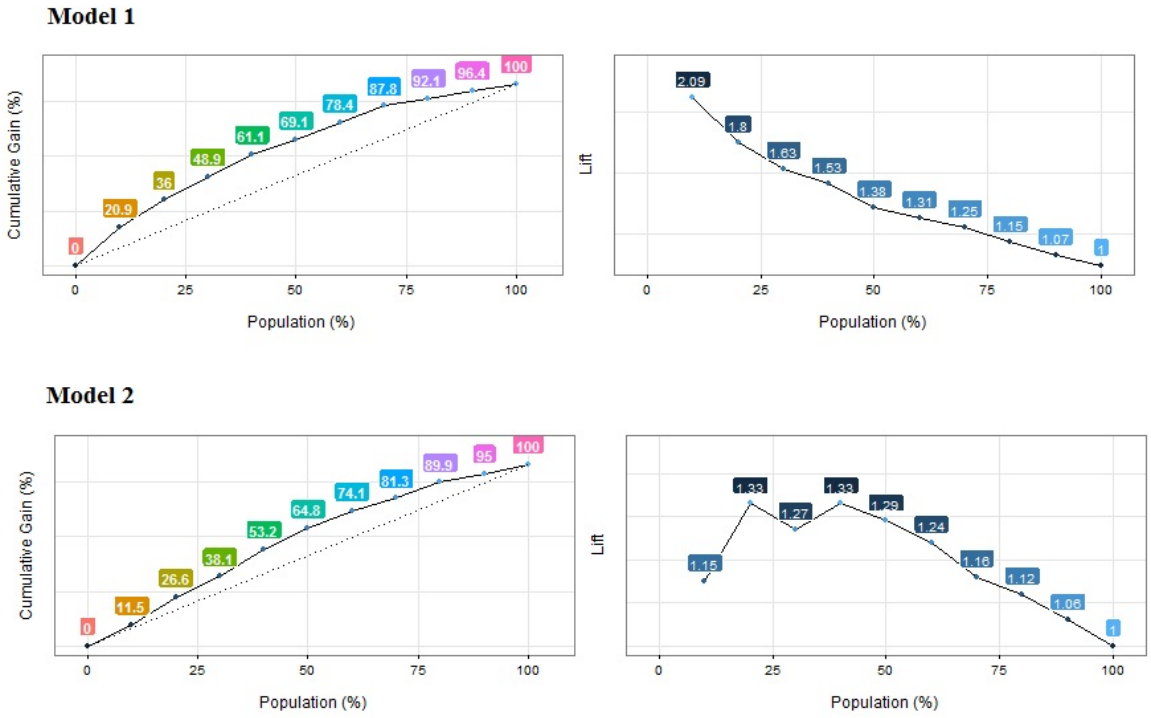


Figure 8.10: Comparando las curvas de ganancia y lift de dos modelos

Agrandar imagen.

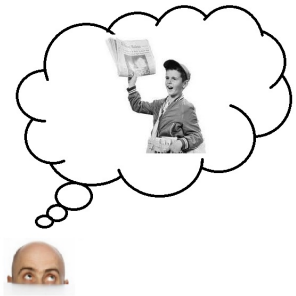
Análisis de la ganancia acumulada: Model 1 alcanza el ~20% de casos positivos cerca del 10% de la población, mientras que Model 2 alcanza una proporción similar cerca del 20% de la población. *Model 1 es mejor.*

Análisis de lift: Lo mismo que antes, pero también resulta sospechoso que no todos los números de lift siguen un patrón descendiente. Quizás el modelo no está ordenando los primeros percentiles de la población. Los mismos conceptos de orden que vimos en el capítulo Análisis numérico de la variable objetivo usando cross_plot.

8.4 Scoring de datos

8.4.1 La intuición detrás

Los eventos pueden ocurrir, o no... aunque no tenemos *el diario del lunes*, podemos hacer una buena suposición de cómo va a ser.



El futuro está indudablemente ligado a *la incertidumbre*, y esta incertidumbre puede ser estimada.

8.4.1.1 Y hay diferentes objetivos...

Por ahora, este libro va a tratar el clásico: objetivo de `Yes`/`No` -también conocido como predicción binaria o multiclase.

Entonces, esta estimación es el *valor de verdad* de que un evento suceda, por lo que es un valor probabilístico entre 0 y 1.

8.4.1.2 Resultados de dos categorías vs. multi-categoría

Por favor tengan en cuenta que este capítulo fue escrito para un resultado binario (dos categorías), pero un objetivo **multi-categoría** puede tomarse como un enfoque general de una clase binaria.

Por ejemplo, al tener una variable objetivo con 4 valores diferentes, puede haber 4 modelos que predicen la probabilidad de pertenecer a una determinada clase, o no. Y luego un modelo superior que tome los resultados de esos 4 modelos y prediga la clase final.

8.4.1.3 ¡¿Qué dijo?!

Algunos ejemplos: - ¿Este cliente va a comprar este producto? - ¿Este paciente va a mejorar? - ¿Cierto evento va a ocurrir en las próximas semanas?

Las respuestas a estas preguntas son Verdadero o Falso, pero **la esencia es tener un score**, o un número que indique la probabilidad de que ocurra un determinado evento.

8.4.1.4 Pero necesitamos más control...

Muchos recursos de machine learning muestran la versión simplificada -que es un buena para empezar- obteniendo la clase final como resultado. Digamos:

Enfoque simplificado:

- Pregunta: *¿Esta persona va a tener una enfermedad cardíaca?*
- Respuesta: "No"

Pero hay algo antes de la respuesta "Sí/No", y eso es el score:

- Pregunta: *¿Cuál es la probabilidad de que esta persona tenga una enfermedad cardíaca?*
- Respuesta: "25%"

Entonces primero obtenemos el puntaje, y luego, de acuerdo a nuestras necesidades, definimos el **punto de corte**. Y esto es **muy** importante.

8.4.2 Veamos un ejemplo

id	*x1*	*x2*	*x3*	target
1	*2.3*	*2.3*	*1.5*	yes
2	*2.1*	*2.4*	*0.2*	no
3	*2.6*	*1.6*	*1.0*	no
4	*2.3*	*2.5*	*0.7*	yes
5	*2.5*	*2.0*	*0.4*	yes
6	*2.0*	*2.4*	*0.5*	no
7	*1.9*	*2.4*	*0.1*	no
8	*1.7*	*2.4*	*0.1*	no
9	*1.9*	*1.9*	*0.2*	no
10	*2.4*	*2.0*	*0.9*	no
11	*2.4*	*1.7*	*1.3*	yes
12	*2.1*	*2.3*	*0.3*	no
13	*2.4*	*2.3*	*0.3*	no
14	*1.9*	*2.0*	*0.2*	no
15	*1.7*	*2.4*	*0.6*	yes

Figure 8.11: Ejemplo simple con un conjunto de datos

La tabla del ejemplo muestra lo siguiente

- `id`=identidad
- `x1`,`x2` y `x3` variables de entrada
- `target`=variable a predecir

id	target	score
1	yes	0.8
2	no	0.2
3	no	0.14
4	yes	0.98
5	yes	0.4
6	no	0.23
7	no	0.21
8	no	0.87
9	no	0.11
10	no	0.5
11	yes	0.89
12	no	0.11
13	no	0.34
14	no	0.23
15	yes	0.85

Figure 8.12: Obteniendo el score (resultado del modelo predictivo)

Dejando de lado la variables de entrada... Después de crear el modelo predictivo, como un modelo de random forest, nos interesan los **scores**. Aunque nuestro objetivo final es llegar a una variable predicha de `yes`/`no`.

Por ejemplo, las siguientes 2 oraciones expresan lo mismo: *La probabilidad de que sea `yes` es `0.8`* <=> *La probabilidad de que sea `no` es `0.2`.*

Tal vez ya se entendió, pero el score generalmente se refiere a la clase menos representativa: `yes`.

Sintaxis en R -*salteen esta sección si no quieren ver código*-

La siguiente oración devolverá el score:

```
score = predict(randomForestModel, data, type = "prob")[, 2]
```

Por favor tengan en cuenta que en otros modelos esta sintaxis puede variar un poco, pero el concepto **será el mismo**. Incluso en otros lenguajes de programación.

Donde `prob` indica que queremos las probabilidades (o scores).

La función `predict` + el parámetro `type="prob"` devuelve una matriz de 15 filas y 2 columnas: la primera indica la probabilidad de que sea `no` mientras la segunda columna muestra lo mismo para la clase `yes`.

Dado que la variable objetivo puede ser `no` o `yes`, entonces `[, 2]` devuelve la probabilidad de que sea -en este caso- `yes` (que es el complemento de la probabilidad de `no`).

8.4.3 Todo se trata del punto de corte

id	target	score *(ordered)*
4	yes	0.98
11	yes	0.89
8	no	0.87
15	yes	0.85
1	yes	0.8
10	no	0.5
5	yes	0.4
13	no	0.34
6	no	0.23
14	no	0.23
7	no	0.21
2	no	0.2
3	no	0.14
9	no	0.11
12	no	0.11

Figure 8.13: Casos ordenados por score más alto

Ahora la tabla está ordenada por score descendiente.

Esto sirve para ver cómo extraer la clase final teniendo por defecto el punto de corte en `0.5`. Ajustar el punto de corte conducirá a una mejor clasificación.

Las métricas de precisión o la matriz de confusión siempre están asociadas

a un determinado valor del punto de corte.

Después de asignar el punto de corte, podemos ver los resultados de la clasificación obteniendo los famosos:

- **Verdadero Positivo** (VP): Es *verdad* que la clasificación es *positiva*, o, "el modelo le acertó a la clase positiva (yes)".
- **Verdadero Negativo** (VN): Lo mismo que antes, pero con la clase negativa (no).
- **Falso Positivo** (FP): Es *falso* que la clasificación es *positiva*, o, "el modelo falló, predijo yes pero el resultado fue no"
- **Falso Negativo** (FN): Lo mismo que antes, pero con la clase negativa, "el modelo predijo un resultado negativo, pero fue positivo", o, "el modelo predijo no, pero la clase fue yes"

id	target	score (ordered)	predicted label (cutpoint @ 0.5)	Accuracy
4	yes	0.98	yes	TP
11	yes	0.89	yes	TP
8	no	0.87	yes	FP
15	yes	0.85	yes	TP
1	yes	0.8	yes	TP
10	no	0.5	yes	FP
5	yes	0.4	no	FN
13	no	0.34	no	TN
6	no	0.23	no	TN
14	no	0.23	no	TN
7	no	0.21	no	TN
2	no	0.2	no	TN
3	no	0.14	no	TN
9	no	0.11	no	TN
12	no	0.11	no	TN

Figure 8.14: Asignar la categoría predicha (cutoff=0.5)

8.4.4 El mejor y peor escenario

Al igual que la filosofía Zen, el análisis de los extremos nos ayudará a encontrar el punto medio.

El mejor escenario es aquel en el que las tasas de **VP** y **VN** son 100%. Eso

significa que el modelo predice correctamente todos los yes y todos los no; *(como resultado, las tasas de **FP** y **FN** son 0%).*

¡Pero esperen ! Si encontramos una clasificación perfecta, ¡probablemente se deba a un sobreajuste!

El peor escenario -lo opuesto al último ejemplo- es aquel en el que las tasas de **FP** y **FN** son 100%. Ni siquiera la aleatoriedad puede lograr un escenario tan horrible.

¿Por qué? Si las clases están balanceadas, 50/50, al tirar una moneda acertaremos alrededor de la mitad de los resultados. Esta es la línea de base común para probar si el modelo es mejor que la aleatoriedad.

En el ejemplo provisto, la distribución de clases es 5 para yes, y 10 para no; entonces: 33,3% (5/15) es yes.

8.4.5 Comparar clasificadores

8.4.5.1 Comparar resultados de clasificación

Trivia: Si un modelo predice correctamente este 33.3% (Tasa de VP=100%), ¿es un buen modelo?

Respuesta: Depende de cuántos 'yes' predijo el modelo.

Un clasificador que siempre predice yes, tendrá una tasa de VP de 100%, pero es absolutamente inútil dado que muchos de esos yes en realidad serán no. De hecho, la tasa de FP será alta.

8.4.5.2 Comparar la etiqueta de orden basándonos en el score

Un clasificador debe ser confiable, y esto es lo que mide la curva **ROC** al trazar

las tasas de VP vs FP. Cuanto mayor sea la proporción de VP sobre FP, mayor será el área debajo de la curva ROC (AUC por Area Under Curve, en inglés).

La intuición detrás de la curva ROC es obtener una **medida de sanidad** con respecto al **score**: qué tan bien ordena la etiqueta. Idealmente, todas las etiquetas positivas deben estar en la parte superior y las negativas en la parte inferior.

id	target	model 1 (good)
4	yes	0.98
11	yes	0.89
8	no	0.87
15	yes	0.85
1	yes	0.8
10	no	0.5
5	yes	0.4
13	no	0.34
6	no	0.23
14	no	0.23
7	no	0.21
2	no	0.2
3	no	0.14
9	no	0.11
12	no	0.11

id	target	model 2 (bad)
14	no	0.99
8	no	0.87
15	yes	0.85
2	no	0.8
10	no	0.7
11	yes	0.5
9	no	0.5
5	yes	0.4
7	no	0.36
13	no	0.34
4	yes	0.3
1	yes	0.3
3	no	0.3
6	no	0.23
12	no	0.11

Figure 8.15: Comparar scores de dos modelos predictivos

`model 1` tendrá una AUC mayor que `model 2`.

En Wikipedia hay artículo bueno y exhaustivo sobre este tema (en inglés): https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Está la comparación de los 4 modelos, con el punto de corte en 0.5:

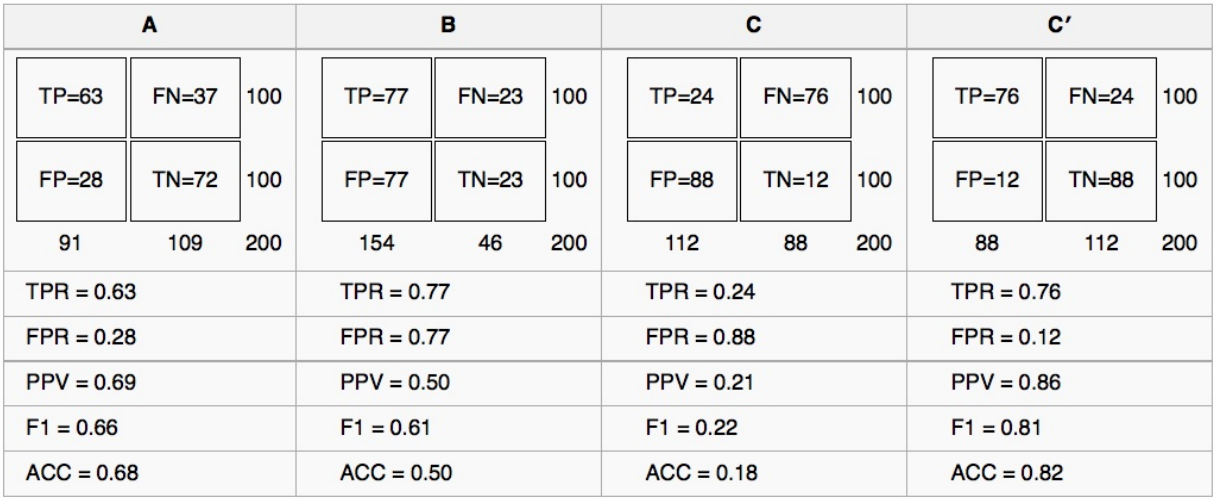


Figure 8.16: Comparando 4 modelos predictivos

8.4.6 ¡Manos a la obra en R!

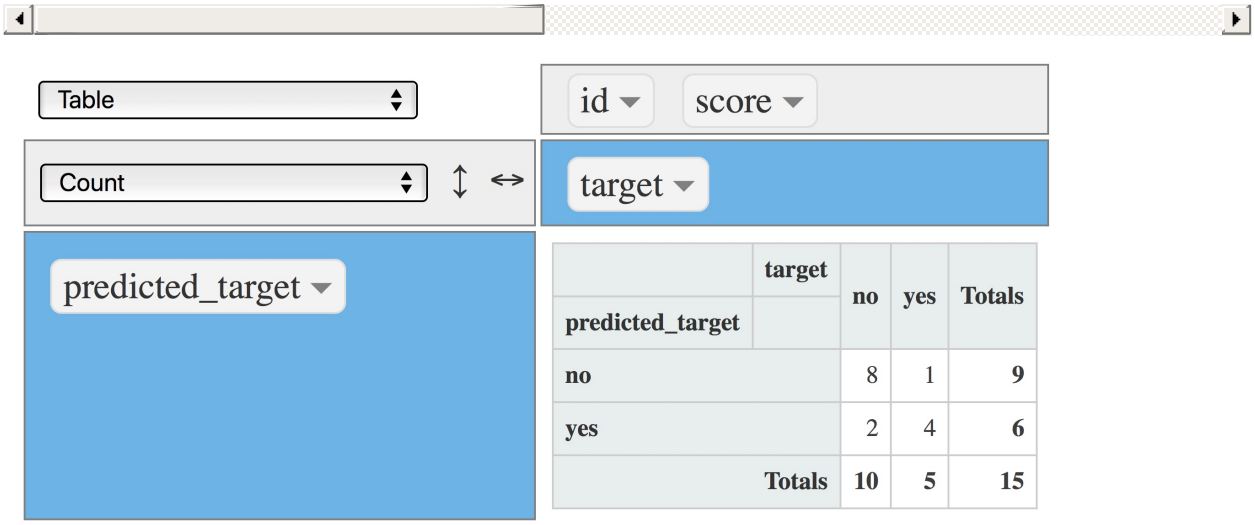
Analizaremos tres escenarios basándonos en diferentes puntos de corte.

```
# install.packages("rpivotTable")
# rpivotTable: crea una tabla pivote dinámicamente, también permite

library(rpivotTable)

## Leer los datos
data=read.delim(file="https://goo.gl/ac5AkG", sep="\t", header = T,
```

8.4.6.1 Escenario 1: punto de corte @ `0.5`

Matriz de confusión clásica, indica cuántos casos caen en la intersección de valor real vs. predicho:

```
data$predicted_target=ifelse(data$score>=0.5, "yes", "no")

rpivotTable(data = data, rows = "predicted_target", cols="target", a
```

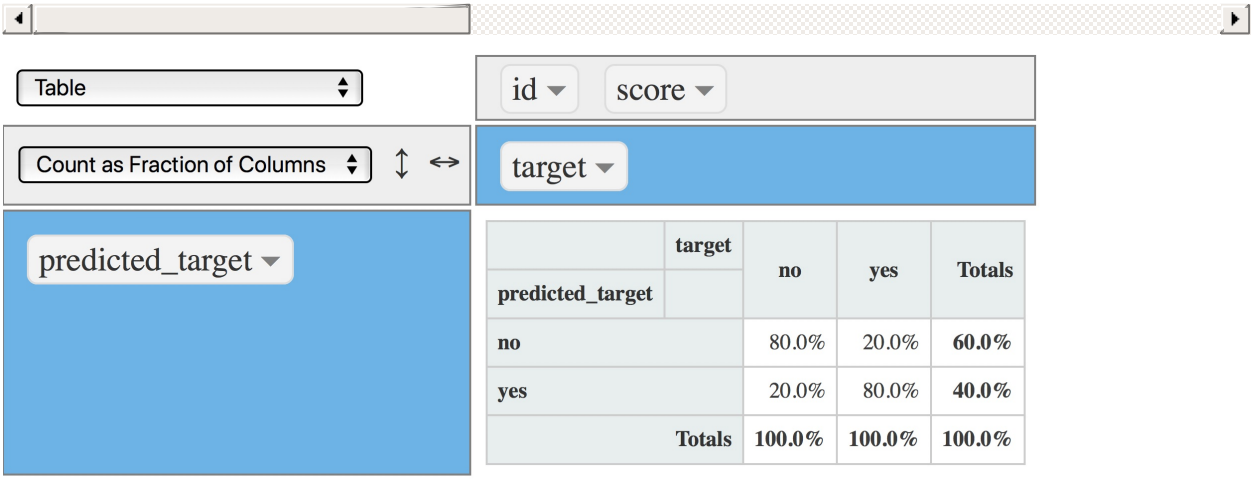


predicted_target	target	no	yes	Totals
no		8	1	9
yes		2	4	6
Totals		10	5	15

Figure 8.17: Matriz de confusión (métrica: conteo)

Otra vista, ahora cada columna suma **100%**. Es bueno responder las siguientes preguntas:

```
rpivotTable(data = data, rows = "predicted_target", cols="target", a
```



predicted_target	target	no	yes	Totals
no		80.0%	20.0%	60.0%
yes		20.0%	80.0%	40.0%
Totals		100.0%	100.0%	100.0%

Figure 8.18: Matriz de confusión (punto de corte en 0.5)

- *¿Cuál es el porcentaje de valores yes reales capturados por el modelo? Respuesta: 80%* También conocido como **Precisión** (PPV por Positive Predictive Value en inglés)
- *¿Cuál es el procentaje de yes que arrojó el modelo? 40%.*

Entonces, a partir de estas dos oraciones:

El modelo clasifica 4 de cada 10 predicciones como `yes`, y en este segmento -el `yes`- acierta en un 80%.

Otra vista: el modelo acierta 3 casos de cada 10 predicciones de `yes` *(0.4/0.8=3.2, o 3, redondeando para abajo).*

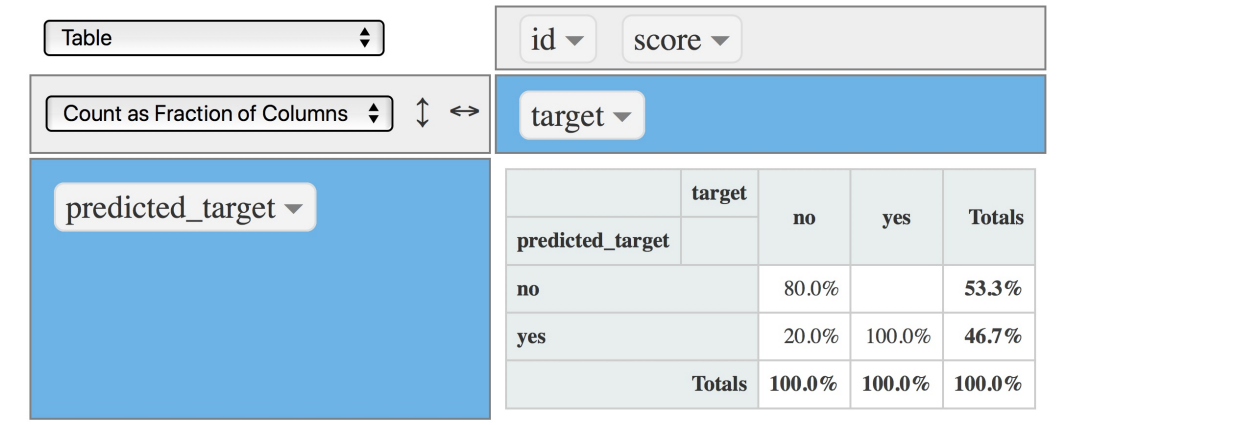
Nota: La última forma de análisis se puede encontrar cuando se construyen las reglas de una asociación (análisis de afinidad o de cesta de la compra), y un modelo de árbol de decisión.

8.4.6.2 Escenario 2: punto de corte @ `0.4`

Hora de cambiar el punto de corte a `0.4`, la cantidad de `yes` será mayor:

```
data$predicted_target=ifelse(data$score>=0.4, "yes", "no")
```

```
rpivotTable(data = data, rows = "predicted_target", cols="target", a
```



predicted_target \ target	no	yes	Totals
no	80.0%		53.3%
yes	20.0%	100.0%	46.7%
Totals	100.0%	100.0%	100.0%

Figure 8.19: Matriz de confusión (punto de corte en 0.4)

Ahora el modelo captura el `100%` de los `yes` (VP), por lo que la cantidad total de

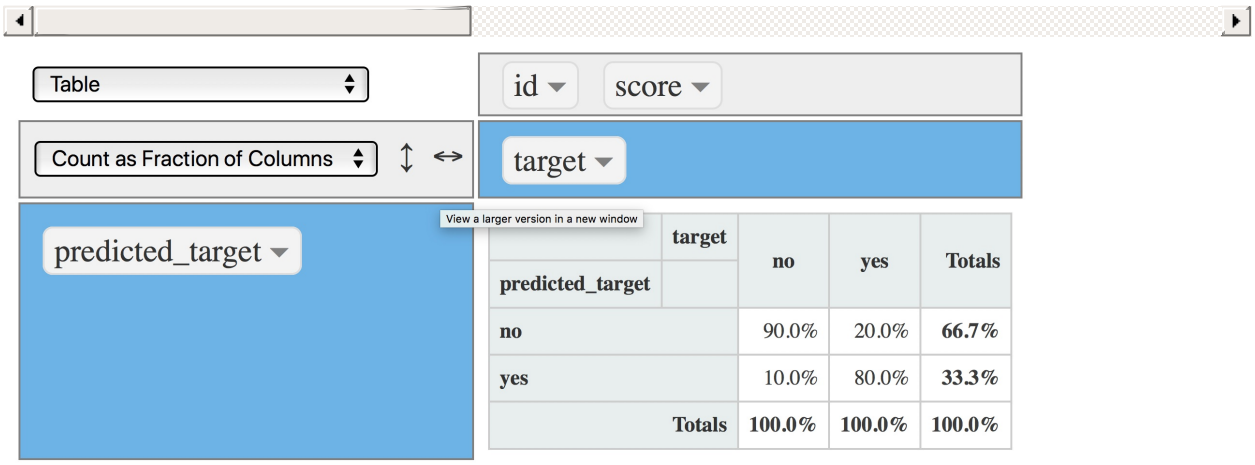
yes producida por el modelo aumentó a 46.7%, pero sin ningún costo, dado que los *VN y FP permanecieron iguales* .

8.4.6.3 Escenario 3: punto de corte @ 0.8

¿Quieren disminuir la tasa de FP? Configuren el punto de corte en un valor superior, por ejemplo: 0.8, que causará que disminuya la cantidad de yes producida por el modelo:

```
data$predicted_target=ifelse(data$score>=0.8, "yes", "no")
```

```
rpivotTable(data = data, rows = "predicted_target", cols="target", a
```



predicted_target \ target	no	yes	Totals
no	90.0%	20.0%	66.7%
yes	10.0%	80.0%	33.3%
Totals	100.0%	100.0%	100.0%

Figure 8.20: Matriz de confusión (punto de corte en 0.8)

Ahora la tasa de FP disminuyó a 10% (de 20%), y el modelo aún captura el 80% de VP que es la misma tasa que la que obtuvimos con el punto de corte en 0.5 .

Disminuir el punto de corte a 0.8 mejoró el modelo sin costo alguno.

8.4.7 Conclusiones

- Este capítulo se ha centrado en la esencia de la predicción de una variable binaria: Para producir un score o número de probabilidad que **ordena** la variable objetivo.

- Un modelo predictivo mapea la entrada con la salida.

- No hay un único y mejor **valor de punto de corte**, depende de las necesidades del proyecto, y está limitado por la tasa de `Falso Positivo` y `Falso Negativo` que podemos aceptar.

Este libro trata aspectos generales sobre el desempeño de los modelos en el capítulo Conociendo el error

---

---

9 Final Words

We have finished a nice book.

References

10 APÉNDICE

Lectura complementaria.

10.1 La magia de los percentiles

El percentil es un concepto tan crucial en el análisis de datos que vamos a cubrirlo ampliamente en este libro. Considera cada observación con respecto a las demás. Un número aislado puede no ser significativo, pero cuando se compara con otros, aparece el concepto de distribución.

Los percentiles se utilizan en el análisis numérico, así como en la evaluación del rendimiento de un modelo predictivo.

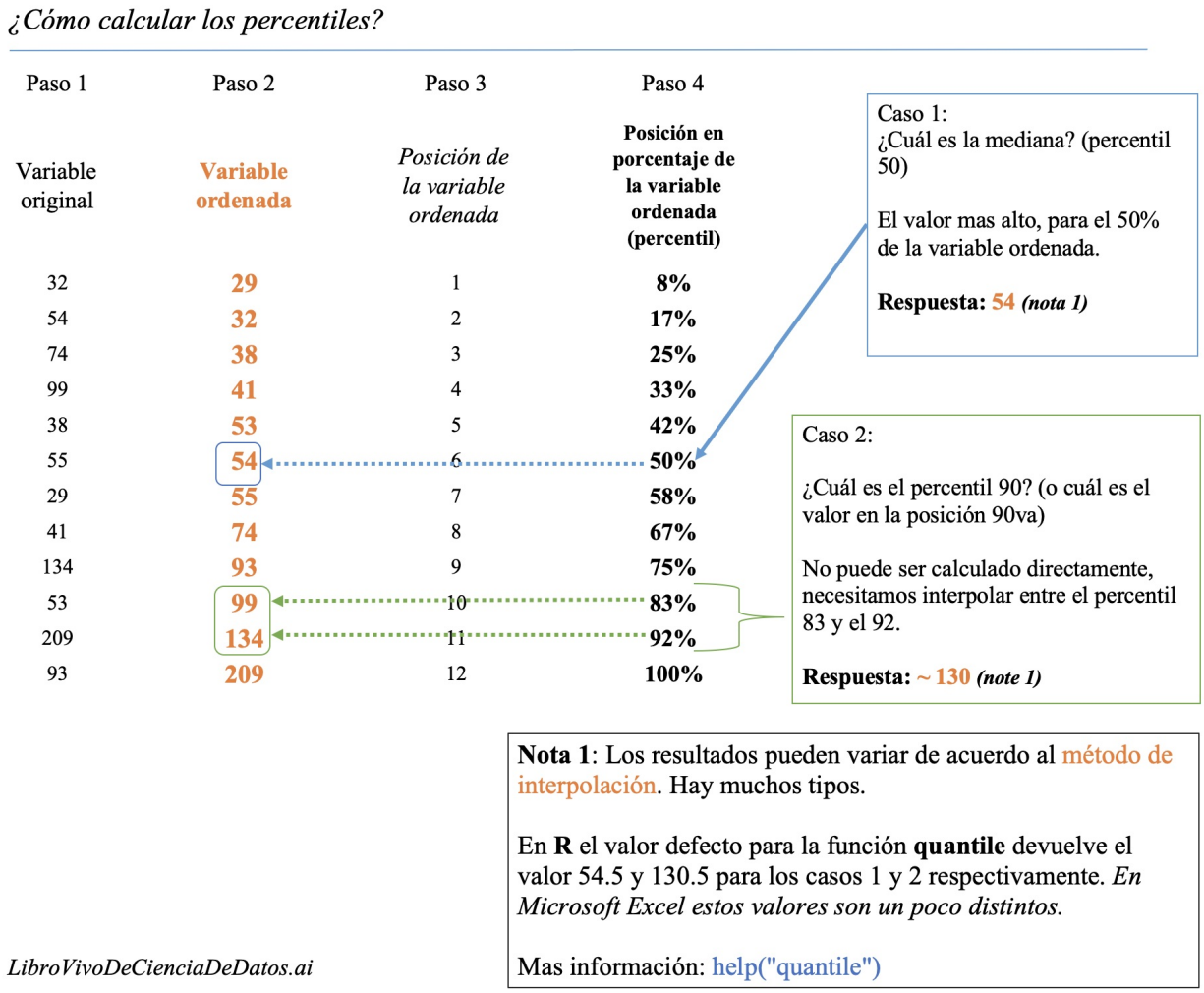


Figure 10.1: Cómo calcular percentiles

El conjunto de datos, un consejo antes de continuar:

Esto contiene muchos indicadores sobre el desarrollo mundial. Independientemente del ejemplo del análisis numérico, la idea es proporcionar una tabla lista para usar para sociólogos, investigadores, etc. interesados en analizar este tipo de datos.

La fuente de datos original es: http://databank.worldbank.org. Ahí encontrarán un diccionario de datos que explica todas las variables.

En esta sección utilizaremos una tabla que ya está preparada para el análisis. La

preparación completa de los datos paso a paso se encuentra en el capítulo Análisis numérico.

Pueden buscar el significado de cualquier indicador en data.worldbank.org. Por ejemplo, si queremos saber qué significa `EN.POP.SLUM.UR.ZS`, entonces escribimos: http://data.worldbank.org/indicator/EN.POP.SLUM.UR.ZS

10.1.1 Cómo calcular percentiles

Existen varios métodos para obtener el percentil. Basado en interpolaciones, la forma más fácil es ordenar la variable de forma ascendente, seleccionando el percentil que deseamos (por ejemplo, 75%), y luego observando *cuál es el valor máximo si queremos elegir el 75% de la población ordenada*.

Ahora vamos a usar la técnica de mantener la muestra pequeña para que podamos tener el máximo control sobre *lo que está sucediendo* detrás del cálculo.

Conservamos los 10 países aleatorios y visualizamos el vector de `rural_poverty_headcount`, que es la variable que vamos a utilizar.

```
library(dplyr)

data_world_wide =
  read.delim(file="https://goo.gl/NNYhCW",
             header = T
             )


data_sample=filter(data_world_wide, Country.Name %in% c("Kazakhstan"

select(data_sample, Country.Name, rural_poverty_headcount)
```

```
##             Country.Name rural_poverty_headcount
## 1               Malaysia                     1.6
## 2             Kazakhstan                     4.4
## 3                Morocco                    14.4
## 4               Colombia                    40.3
## 5                   Fiji                    44.0
```

```
## 6              Mauritania                    59.4
## 7  Sao Tome and Principe                    59.4
## 8           Sierra Leone                    66.1
## 9                  Haiti                    74.9
## 10               Zambia                     77.9
```

Tengan en cuenta que el vector se ordena sólo con fines didácticos. *Como dijimos en el capítulo de Análisis numérico, a nuestros ojos les gusta el orden.*

Ahora aplicamos la función "cuantitativa" a otra variable (el porcentaje de la población rural que vive por debajo de las líneas de pobreza):

```
quantile(data_sample$rural_poverty_headcount)
```

```
##     0%    25%    50%    75%   100%
##  1.600 20.875 51.700 64.425 77.900
```

Análisis

- **Percentil 50%**: el 50% de los países (cinco de ellos) tienen una `rural_poverty_headcount` debajo de `51.7`. Podemos comprobar esto en la última tabla: estos países son: Fiji, Colombia, Marruecos, Kazajistán, y Malasia.
- **Percentil 25%**: el 25% de los países están por debajo de 20.87. Aquí podemos ver una interpolación porque el 25% representa ~2.5 países. Si utilizamos este valor para filtrar los países, entonces tendremos tres países: Marruecos, Kazajistán, y Malasia.

Más información sobre los diferentes tipos de cuantiles y sus interpolaciones: `help("quantile")`.

10.1.1.1 Obtener las descripciones semánticas

Del último ejemplo podemos afirmar que:

- *"La mitad de los países tienen hasta un 51.7% de pobreza rural"*
- *"Tres cuartas partes de los países tienen un máximo de 64.4% en cuanto a su pobreza rural"* (basado en los países ordenados ascendentemente).

También podemos pensar en **usar lo contrario**:

- *"Una cuarta parte de los países que presentan los valores más altos de pobreza rural tienen un porcentaje de por lo menos 64.4%"*

10.1.2 Calcular cuantiles personalizados

Típicamente, queremos calcular ciertos cuantiles. La variable de ejemplo será el `gini_index`.

¿Qué es el Índice de Gini?

Es una medida de la desigualdad de ingresos o de riqueza.

- Un coeficiente de Gini de **cero** expresa **igualdad perfecta** donde todos los valores son iguales (por ejemplo, donde todos tienen los mismos ingresos).
- Un coeficiente de Gini de **1** (o 100%) expresa **desigualdad máxima** entre los valores (por ejemplo, en un gran número de personas, donde sólo una persona tiene todos los ingresos o el consumo mientras que todas las demás no tienen ninguno, el coeficiente de Gini será muy cercano a uno).

Fuente: https://en.wikipedia.org/wiki/Gini_coefficient

Ejemplo en R:

Si queremos obtener los cuantiles de 20, 40, 60, y 80 de la variable del índice de Gini, volvemos a usar la función `quantile`.

El parámetro `na.rm=TRUE` es necesario si tenemos valores vacíos como en este caso:

```
# También podemos obtener múltiples cuantiles en simultáneo
p_custom=quantile(data_world_wide$gini_index, probs = c(0.2, 0.4, 0.
p_custom
```

```
##    20%    40%    60%    80%
## 31.624 35.244 41.076 46.148
```

10.1.3 Indicar dónde están la mayoría de los valores

En estadística descriptiva, queremos describir la población en términos

generales. Podemos hablar de rangos usando dos percentiles. Tomemos los percentiles 10 y 90 para describir al 80% de la población.

La pobreza oscila entre el 0.075% y el 54.4% en el 80% de los países. (80% porque hicimos percentil 90 - percentil 10, centrándonos en la mitad de la población.)

Si consideramos al 80% como la mayoría de la población, entonces podríamos decir: *"Normalmente (o en términos generales), la pobreza pasa de 0.07% a 54.4%"*. Esta es una descripción semántica.

Observamos al 80% de la población, que parece ser un buen número para describir dónde están la mayoría de los casos. También podríamos haber usado el rango del 90% (percentil 95 - percentil 5).

10.1.3.1 ¿El percentil se relaciona con el cuartil?

Cuartil es el nombre formal para los percentiles 25, 50 y 75 (cuartos o 'Q'). Si queremos observar el 50% de la población, debemos restar el 3er cuartil (o percentil 75) del 1er cuartil (percentil 25) para saber dónde está concentrado el 50% de los datos, también conocido como el **rango intercuartil** o IQR.

Percentil vs. cuantil vs. cuartil

```
0 cuartil = 0 cuantil = 0 percentil
1 cuartil = 0.25 cuantil = 25 percentil
2 cuartil = .5 cuantil = 50 percentil (mediana)
3 cuartil = .75 cuantil = 75 percentil
4 cuartil = 1 cuantil = 100 percentil
```

Créditos: (stats.stackexchange.com 2017b).

10.1.4 Visualizar cuantiles

Graficar un histograma junto a los lugares donde se encuentra cada percentil puede ayudarnos a entender el concepto:

```
quantiles_var =
  quantile(data_world_wide$poverty_headcount_1.9,
           c(0.25, 0.5, 0.75),
```

```
            na.rm = T
            )

df_p = data.frame(value=quantiles_var,
                  quantile=c("25th", "50th", "75th")
                  )

library(ggplot2)
ggplot(data_world_wide, aes(poverty_headcount_1.9)) +
  geom_histogram() +
  geom_vline(data=df_p,
             aes(xintercept=value,
                 colour = quantile),
             show.legend = TRUE,
             linetype="dashed"
             ) +
  theme_light()
```

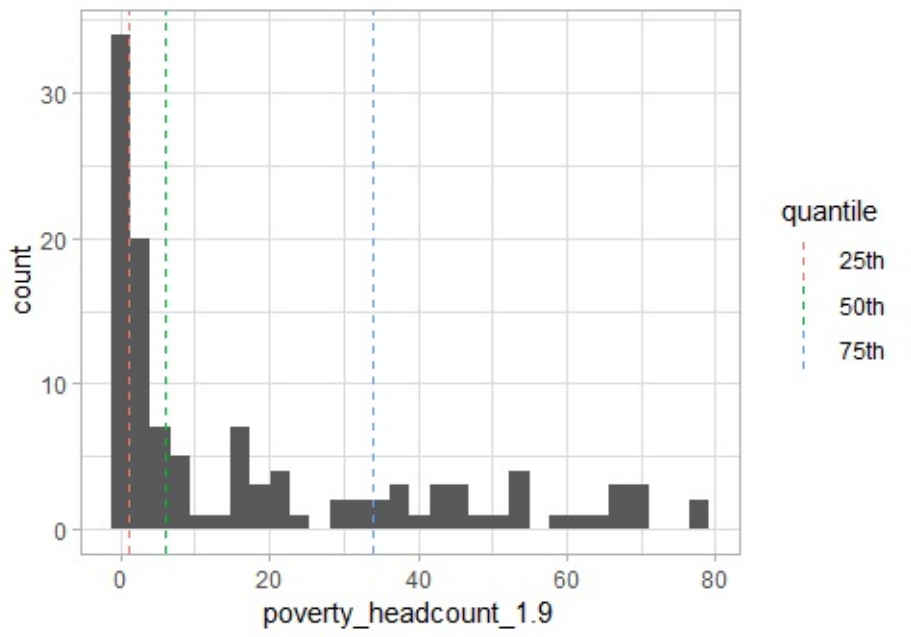


Figure 10.2: Visualizar cuantiles

Si sumamos todas las barras grises antes del percentil 25, tendrán aproximadamente la misma altura acumulada que la suma de las barras grises después del percentil 75.

En el último gráfico, el IQR aparece entre la primera y la última línea punteada y contiene el 50% de la población.

10.1.5 Conceptos de clasificación y top/bottom '*X%*'

El concepto de clasificación es el mismo que el de las competiciones. Nos permite responder *¿cuál es el país con la tasa más alta en la variable pop_living_slums?*

Usaremos la función `dense_rank` del paquete `ggplot2`. Asigna la posición (puesto) a cada país, pero las necesitamos en orden inverso; es decir, asignamos el `rank = 1` al valor más alto.

Ahora la variable será: *La población que vive en hogares marginales es la proporción de la población urbana que vive en hogares marginales. Un hogar marginal se define como un grupo de individuos que viven bajo el mismo techo y que carecen de una o más de las siguientes condiciones: acceso a agua potable, acceso a saneamiento mejorado, suficiente área habitable y durabilidad de la vivienda.*

La pregunta a responder: *¿Cuáles son los seis países con las tasas más altas de personas que viven en hogares marginales?*

```
# Crear la variable de clasificación
data_world_wide$rank_pop_living_slums =
  dense_rank(-data_world_wide$pop_living_slums)


# Ordenar los datos según la clasificación
data_world_wide=arrange(data_world_wide, rank_pop_living_slums)

# Visualizar los primeros seis resultados
select(data_world_wide, Country.Name, rank_pop_living_slums) %>% hea
```

```
##               Country.Name rank_pop_living_slums
## 1              South Sudan                     1
## 2 Central African Republic                     2
## 3                    Sudan                     3
## 4                     Chad                     4
## 5    Sao Tome and Principe                     5
## 6            Guinea-Bissau                     6
```

También podemos preguntar: *¿En qué posición está Ecuador?*

```
filter(data_world_wide, Country.Name=="Ecuador") %>% select(rank_pop
```

```
##   rank_pop_living_slums
## 1                    57
```

10.1.5.0.1 Concepto de top/bottom '*X%*'

Otra pregunta que podría interesarnos responder: *¿Cuál es el valor con el que obtengo el 10% superior de los valores más bajos?*

El percentil 10 es la respuesta:

```
quantile(data_world_wide$pop_living_slums, probs=.1, na.rm = T)
```

```
##  10%
## 12.5
```

Trabajando en lo opuesto: *¿Cuál es el valor con el que obtengo el 10% inferior de los valores más altos?*

El percentil 90 es la respuesta, podemos filtrar todos los casos por encima de este valor:

```
quantile(data_world_wide$pop_living_slums,
         probs=.9,
         na.rm = T
         )
```

```
##  90%
## 75.2
```

10.1.6 Uso de percentiles en el scoring de datos

Hay dos capítulos que usan este concepto:

- Scoring de datos
- Análisis de Ganancia y Lift

La idea básica es desarrollar un modelo predictivo que prediga una variable binaria (yes/no). Supongamos que necesitamos puntuar nuevos casos, por ejemplo, para utilizarlos en una campaña de marketing. La pregunta a responder es:

¿Cuál es el valor del puntaje que deberíamos sugerirle a los ejecutivos de ventas para capturar el 50% de las nuevas ventas potenciales? La respuesta proviene de una combinación de análisis de percentil sobre el valor del puntaje más el análisis acumulativo de la variable objetivo actual.

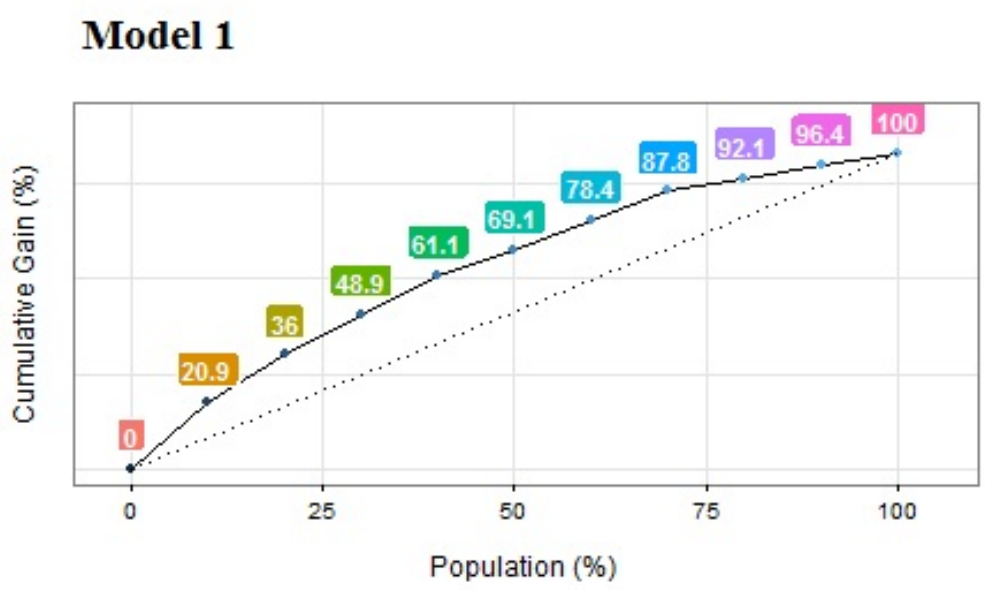


Figure 10.3: Curvas de ganancia y lift (desempeño del modelo)

10.1.6.1 Estudio de caso: Distribución de la riqueza

La distribución de la riqueza es similar a la del índice de Gini y se centra en la desigualdad. Mide los activos de los propietarios (que son diferentes de los ingresos), haciendo que la comparación entre países sea más uniforme con lo que las personas pueden adquirir según el lugar en el que viven. Para una mejor definición, consulten el artículo de *Wikipedia* y el *Informe sobre el patrimonio mundial 2013*. Referencias: (Wikipedia 2017a) y (Suisse 2013), respectivamente.

Citando a *Wikipedia* (Ref. (Wikipedia 2017a)):

la mitad de la riqueza mundial pertenece al 1% de la población;

el 10% superior de los adultos posee el 85%, mientras que el 90% inferior posee el 15% restante de la riqueza total del mundo; y

el 30% de los adultos más ricos poseen el 97% de la riqueza total.

Al igual que antes, a partir de la tercera frase podemos afirmarlo: *"El 3% de la riqueza total se distribuye entre el 70% de los adultos"*.

Las métricas `top 10%` y `top 30%` son los quantiles `0.1` y `0.3`. La riqueza es la variable numérica.

10.2 Quick-start `funModeling`

Este paquete contiene un conjunto de funciones relacionadas con el análisis exploratorio de datos, la preparación de datos y el desempeño del modelo. Es utilizado por personas procedentes de los negocios, la investigación y la docencia (profesores y estudiantes).

El paquete `funModeling` está íntimamente relacionado con este libro, en el sentido de que la mayor parte de su funcionalidad se utiliza para explicar los diferentes temas abordados por el libro.

10.2.1 Abriendo la caja negra

Algunas funciones tienen comentarios incluidos en las líneas para que el usuario

pueda abrir la caja negra y aprender cómo se desarrolló, o para afinar o mejorar cualquiera de ellas.

Todas las funciones están bien documentadas, explicando todos los parámetros con la ayuda de muchos ejemplos breves. Se puede acceder a la documentación en R a través de: `help("name_of_the_function")`.

Cambios importantes desde la última versión 1.6.7, (esto sólo es relevante si estaban usando versiones anteriores):

A partir de la última versión, 1.6.7 (21 de enero de 2018), los parámetros `str_input`, `str_target` y `str_score` se renombrarán a `input`, `target` y `score` respectivamente. La funcionalidad sigue siendo la misma. Si estaban utilizando estos nombres de parámetros en producción, seguirán funcionando hasta la próxima versión. Esto significa que, por ahora, pueden utilizar, por ejemplo, `str_input` o `input`.

El otro cambio importante fue en `discretize_get_bins`, que se detalla más adelante en este documento.

10.2.1.1 Sobre este quick-start

Este quick-start se centra sólo en las funciones. Pueden consultar todas las explicaciones en torno a ellas, así como el cómo y cuándo utilizarlas, siguiendo los enlaces "***Leer más aquí.***" que se encuentran debajo de cada sección, que los redirigirán al libro.

A continuación está la mayoría de las funciones de `funModeling` divididas por categoría.

10.2.2 Análisis exploratorio de datos

10.2.2.1 `df_status`: Estado de salud de un conjunto de datos

Caso de uso: analiza los ceros, los valores faltantes (`NA`), el infinito, el tipo de datos y el número de valores únicos para un conjunto de datos determinado.

```r
library(funModeling)

df_status(heart_disease)
```

```
##                   variable q_zeros p_zeros q_na p_na q_inf p_inf
## 1                      age       0    0.00    0 0.00     0     0 i
## 2                   gender       0    0.00    0 0.00     0     0
## 3               chest_pain       0    0.00    0 0.00     0     0
## 4  resting_blood_pressure       0    0.00    0 0.00     0     0 i
## 5       serum_cholestoral       0    0.00    0 0.00     0     0 i
## 6     fasting_blood_sugar     258   85.15    0 0.00     0     0
## 7         resting_electro     151   49.83    0 0.00     0     0
## 8          max_heart_rate       0    0.00    0 0.00     0     0 i
## 9              exer_angina     204   67.33    0 0.00     0     0 i
## 10                oldpeak      99   32.67    0 0.00     0     0 n
## 11                  slope       0    0.00    0 0.00     0     0 i
## 12      num_vessels_flour     176   58.09    4 1.32     0     0 i
## 13                   thal       0    0.00    2 0.66     0     0
## 14 heart_disease_severity     164   54.13    0 0.00     0     0 i
## 15           exter_angina     204   67.33    0 0.00     0     0
## 16      has_heart_disease       0    0.00    0 0.00     0     0
##    unique
## 1      41
## 2       2
## 3       4
## 4      50
## 5     152
## 6       2
## 7       3
## 8      91
## 9       2
## 10     40
## 11      3
## 12      4
## 13      3
## 14      5
## 15      2
## 16      2
```

[**Leer más aquí.**]

10.2.2.2 `plot_num`: Graficar las distribuciones de variables numéricas

Solamente grafica variables numéricas.

```
plot_num(heart_disease)
```

Figure 10.4: plot num: visualizar variables numéricas

Notas:

- `bins`: configura la cantidad de segmentos (10 por defecto).
- `path_out`: indica el directorio donde se guardará el archivo; si tiene un valor, entonces el gráfico se exportará en formato jpeg. Para guardarlo en el directorio actual debe ser un punto: "."

[**Leer más aquí.**]

10.2.2.3 `profiling_num`: Calcular varias estadísticas para variables numéricas

Obtiene varias estadísticas para variables numéricas.

```
profiling_num(heart_disease)
```

```
##                 variable   mean std_dev variation_coef p_01 p_05
```

```
## 1                    age  54.44    9.04          0.17   35   40
## 2 resting_blood_pressure 131.69   17.60          0.13  100  108
## 3      serum_cholestoral 246.69   51.78          0.21  149  175
## 4         max_heart_rate 149.61   22.88          0.15   95  108
## 5            exer_angina   0.33    0.47          1.44    0    0
## 6                oldpeak   1.04    1.16          1.12    0    0
## 7                  slope   1.60    0.62          0.38    1    1
## 8      num_vessels_flour   0.67    0.94          1.39    0    0
## 9 heart_disease_severity   0.94    1.23          1.31    0    0
##    p_50  p_75  p_95  p_99 skewness kurtosis  iqr        range_98
## 1  56.0  61.0  68.0  71.0    -0.21      2.5 13.0        [35, 71]
## 2 130.0 140.0 160.0 180.0     0.70      3.8 20.0      [100, 180]
## 3 241.0 275.0 326.9 406.7     1.13      7.4 64.0   [149, 406.74]
## 4 153.0 166.0 181.9 192.0    -0.53      2.9 32.5 [95.02, 191.96]
## 5   0.0   1.0   1.0   1.0     0.74      1.5  1.0          [0, 1]
## 6   0.8   1.6   3.4   4.2     1.26      4.5  1.6        [0, 4.2]
## 7   2.0   2.0   3.0   3.0     0.51      2.4  1.0          [1, 3]
## 8   0.0   1.0   3.0   3.0     1.18      3.2  1.0          [0, 3]
## 9   0.0   2.0   3.0   4.0     1.05      2.8  2.0          [0, 4]
##         range_80
## 1       [42, 66]
## 2     [110, 152]
## 3 [188.8, 308.8]
## 4   [116, 176.6]
## 5         [0, 1]
## 6       [0, 2.8]
## 7         [1, 2]
## 8         [0, 2]
## 9         [0, 3]
```

Nota:

- `plot_num` y `profiling_num` automáticamente excluyen variables no numéricas

[**Leer más aquí.**]

10.2.2.4 `freq`: Obtener las distribuciones de frecuencia de variables categóricas

```
library(dplyr)
```

```r
# Seleccionar sólo dos variables para este ejemplo
heart_disease_2=heart_disease %>% select(chest_pain, thal)

# Distribución de la frecuencia
freq(heart_disease_2)
```

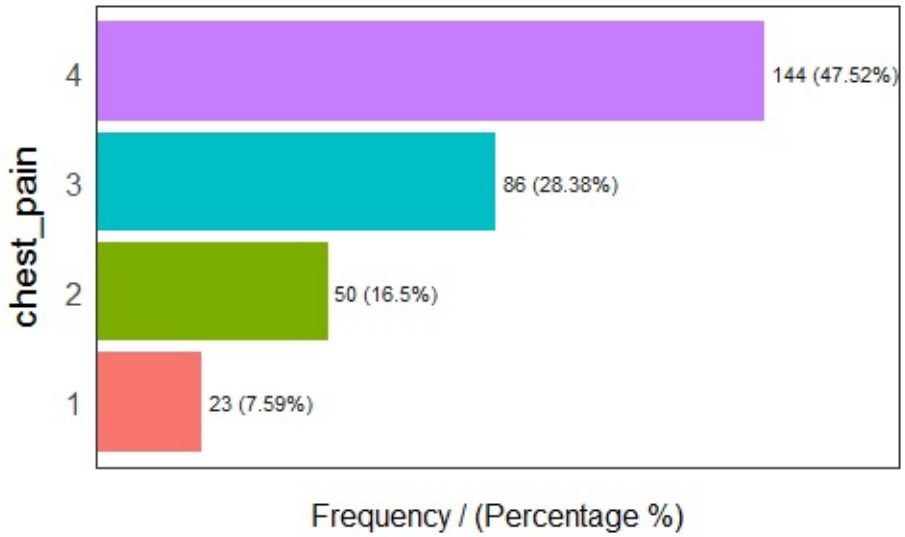


Figure 10.5: freq: visualizar variables categóricas

```
##   chest_pain frequency percentage cumulative_perc
## 1          4       144       47.5               48
## 2          3        86       28.4               76
## 3          2        50       16.5               92
## 4          1        23        7.6              100
```

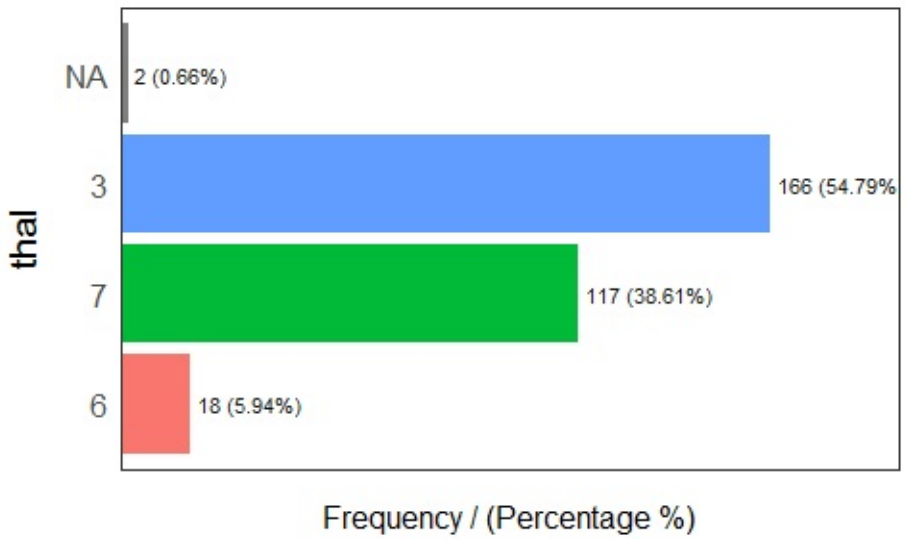


Figure 10.6: freq: visualizar variables categóricas

```
##   thal frequency percentage cumulative_perc
## 1    3       166      54.79              55
## 2    7       117      38.61              93
## 3    6        18       5.94              99
## 4 <NA>         2       0.66             100

## [1] "Variables processed: chest_pain, thal"
```

Notas:

- `freq` sólo procesa variables `factor` y `character`, excluyendo variables no categóricas.
- Devuelve la tabla de distribución como un data frame.
- Si `input` está vacío, entonces se ejecuta para todas las variables categóricas.
- `path_out` indica el directorio donde se guardará el archivo; si tiene un valor, entonces el gráfico se exportará en formato jpeg. Para guardarlo en el directorio actual debe ser un punto: "."
- `na.rm` indica si los valores `NA` deberían ser excluidos (`FALSE` por defecto).

[**Leer más aquí.**]

10.2.3 Correlaciones

10.2.3.1 `correlation_table`: Calcula el estadístico R

Obtiene la métrica R (o coeficiente de Pearson) para todas las variables numéricas, saltándose las categóricas.

```
correlation_table(heart_disease, "has_heart_disease")
```

```
##                 Variable has_heart_disease
## 1      has_heart_disease              1.00
## 2 heart_disease_severity              0.83
## 3      num_vessels_flour              0.46
## 4                oldpeak              0.42
## 5                  slope              0.34
## 6                    age              0.23
```

```
## 7 resting_blood_pressure              0.15
## 8      serum_cholestoral              0.08
## 9         max_heart_rate             -0.42
```

Notas:

- Sólo se analizan las variables numéricas. La variable objetivo debe ser numérica.
- Si la variable objetivo es categórica, entonces será convertida a numérica.

[**Leer más aquí.**]

10.2.3.2 `var_rank_info`: Correlación basada en Teoría de la Información

Calcula la correlación basándose en distintas métricas de la Teoría de la Información entre todas las variables de un data frame y una variable objetivo.

```
var_rank_info(heart_disease, "has_heart_disease")
```

```
##                         var  en    mi      ig      gr
## 1  heart_disease_severity 1.8 0.995 0.99508 0.53907
## 2                      thal 2.0 0.209 0.20946 0.16805
## 3              exer_angina 1.8 0.139 0.13914 0.15264
## 4             exter_angina 1.8 0.139 0.13914 0.15264
## 5               chest_pain 2.5 0.205 0.20502 0.11803
## 6       num_vessels_flour 2.4 0.182 0.18152 0.11577
## 7                    slope 2.2 0.112 0.11242 0.08688
## 8       serum_cholestoral 7.5 0.561 0.56056 0.07956
## 9                   gender 1.8 0.057 0.05725 0.06330
## 10                oldpeak 4.9 0.249 0.24917 0.06036
## 11         max_heart_rate 6.8 0.334 0.33362 0.05407
## 12 resting_blood_pressure 5.6 0.143 0.14255 0.03024
## 13                    age 5.9 0.137 0.13718 0.02705
## 14        resting_electro 2.1 0.024 0.02415 0.02219
## 15    fasting_blood_sugar 1.6 0.000 0.00046 0.00076
```

Nota: Analiza variables numéricas y categóricas. También se utiliza con el método de discretización numérica como antes, tal como `discretize_df`.

[**Leer más aquí.**]

10.2.3.3 `cross_plot`: Gráfico de distribución entre variable de entrada y variable objetivo

Obtiene la distribución relativa y absoluta entre una variable de entrada y una variable objetivo. Es útil para explicar y reportar si una variable es importante o no.

```
cross_plot(data=heart_disease, input=c("age", "oldpeak"), target="ha
```

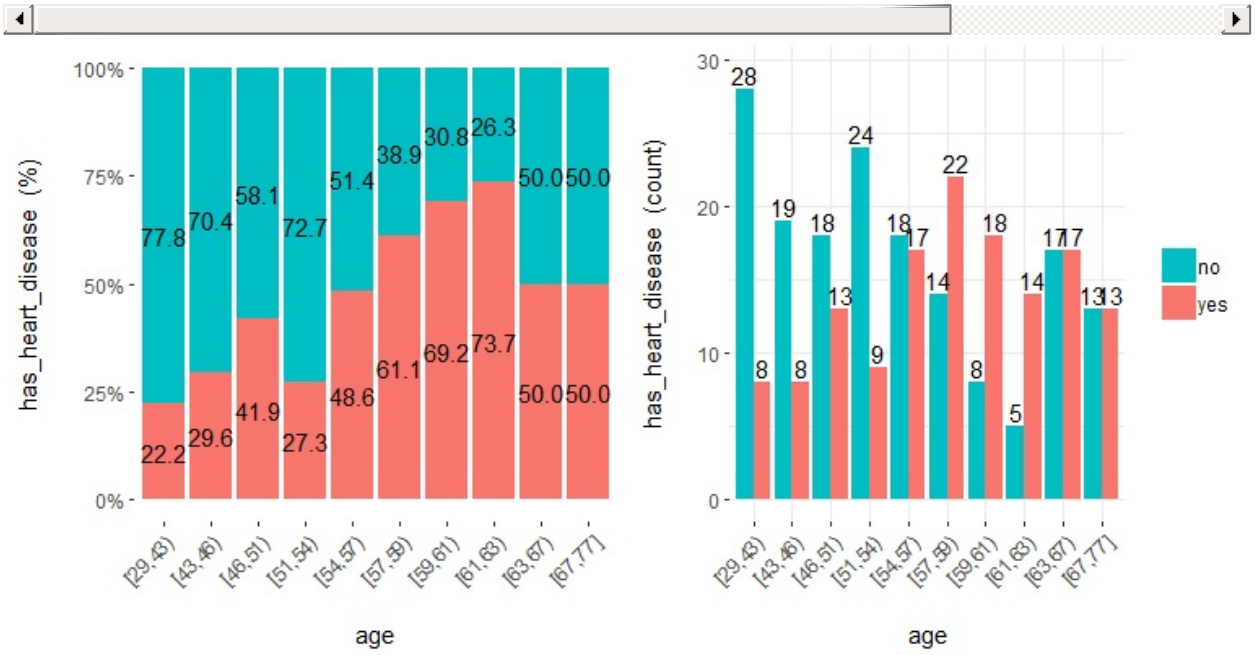


Figure 10.7: cross plot: Visualizar variable de entrada vs. variable objetivo

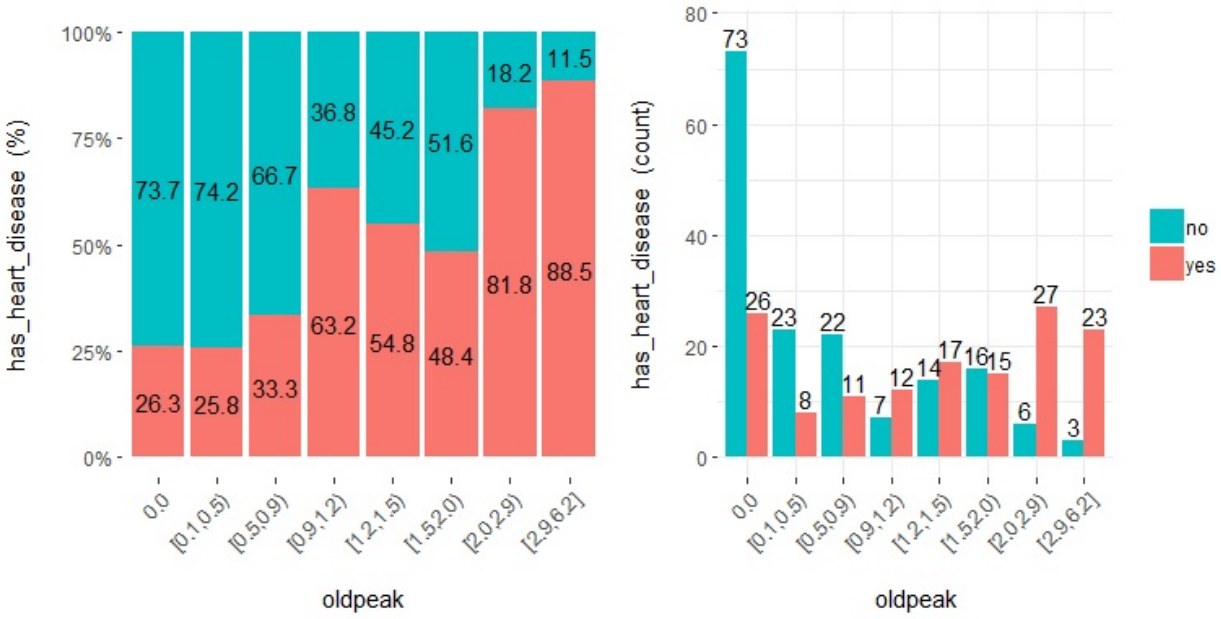


Figure 10.8: cross plot: Visualizar variable de entrada vs. variable objetivo

Notas:

- `auto_binning`: `TRUE` por defecto, muestra la variable numérica como categórica.
- `path_out` indica el directorio donde se guardará el archivo; si tiene un valor, entonces el gráfico se exportará en formato jpeg.
- `input` puede ser numérica o categórica, y `target` debe ser una variable binaria (dos clases).
- Si `input` está vacío, entonces se ejecutará para todas las variables.

[**Leer más aquí.**]

10.2.3.4 `plotar`: Diagramas de caja e histogramas de densidad entre variables de entrada y objetivo

Es útil para explicar y reportar si una variable es importante o no.

Diagrama de caja:

```
plotar(data=heart_disease, input = c("age", "oldpeak"), target="has_
```

Figure 10.9: plotar (1): visualizar un diagrama de caja

Figure 10.10: plotar (1): visualizar un diagrama de caja

[**Leer más aquí.**]

Histogramas de densidad:

```
plotar(data=mtcars, input = "gear", target="cyl", plot_type="histden
```

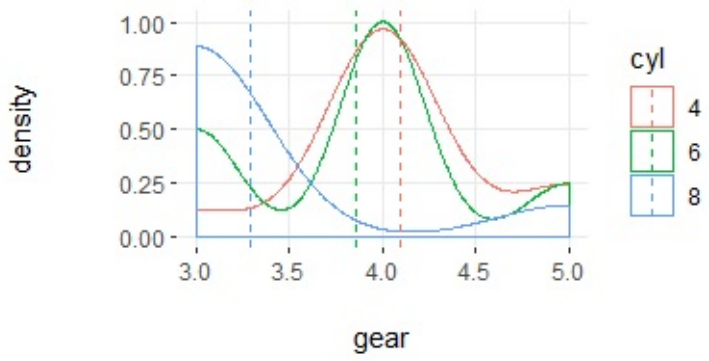


Figure 10.11: plotar (2): visualizar histograma de densidad

[**Leer más aquí.**]

Notes:

- `path_out` indica el directorio donde se guardará el archivo; si tiene un valor, entonces el gráfico se exportará en formato jpeg.
- Si `input` está vacío, entonces se ejecuta para todas la variables numéricas (saltéandose las categóricas).
- `input` debe ser numérico y la variable objetivo debe ser categórica.
- `target` puede ser multi-clase (no sólo binario).

10.2.3.5 `categ_analysis`: Análisis cuantitativo para un resultado binario

Hace un análisis numérico de una variable binaria objetivo basándose en una variable categórica de entrada, la representatividad (`perc_rows`) y la precisión (`perc_target`) de cada valor de la variable de entrada; por ejemplo, la tasa de infección por gripe por país.

```
df_ca=categ_analysis(data = data_country, input = "country", target

head(df_ca)
```

```
##          country mean_target sum_target perc_target q_rows perc_r
## 1       Malaysia        1.00          1       0.012      1     0.
## 2         Mexico        0.67          2       0.024      3     0.
```

```
## 3       Portugal        0.20          1       0.012      5     0.
## 4 United Kingdom        0.18          8       0.096     45     0.
## 5        Uruguay        0.17         11       0.133     63     0.
## 6         Israel        0.17          1       0.012      6     0.
```

Nota:

- La variable `input` debe ser categórica.
- La variable `target` debe ser binaria (dos valores posibles).

Esta función se utiliza para analizar datos cuando necesitamos reducir la cardinalidad de las variables en el modelado predictivo.

[**Leer más aquí.**]

10.2.4 Preparación de datos

10.2.4.1 Discretización de datos

10.2.4.1.1 `discretize_get_bins` + `discretize_df`: Convertir variables numéricas a categóricas

Necesitamos dos funciones: `discretize_get_bins`, que devuelve los umbrales para cada variable, y luego `discretize_df`, que toma el resultado de la primera función y convierte las variables deseadas. El criterio de segmentación es el de igual frecuencia.

Ejemplo: convertir sólo dos variables de un conjunto de datos.

```
# Paso 1: Obtener los umbrales de las variables deseadas: "max_heart
d_bins=discretize_get_bins(data=heart_disease, input=c("max_heart_ra
```

```
## Variables processed: max_heart_rate, oldpeak
```

```
# Paso 2: Aplicar el umbral para llegar al
# data frame final
heart_disease_discretized =
  discretize_df(data=heart_disease,
                data_bins=d_bins,
                stringsAsFactors=T
```

```
                  )

## Variables processed: max_heart_rate, oldpeak
```

La siguiente imagen ilustra el resultado. Por favor noten que el nombre de la variable no cambió.

max_heart_rate_before <int>	max_heart_rate_after <fctr>	oldpeak_before <dbl>	oldpeak_after <fctr>
171	[171, Inf]	*NA*	NA.
114	[-Inf, 131)	*NA*	NA.
151	[147, 160)	1.8	[1.1, 2.0)
160	[160, 171)	1.4	[1.1, 2.0)
158	[147, 160)	0.0	[-Inf, 0.1)
161	[160, 171)	0.5	[0.3, 1.1)

Figure 10.12: Resultados del proceso de discretización automática

Notas:

- Este procedimiento de dos pasos está pensado para ser utilizado en producción con nuevos datos.
- Los valores mín y máx de cada segmento serán `-Inf` e `Inf`, respectivamente.
- Un cambio en la última versión de `funModeling` (1.6.7) puede alterar el resultado en algunos escenarios. Por favor verifiquen los resultados si están usando la versión 1.6.6. Más información sobre este cambio aquí.

[**Leer más aquí.**]

10.2.4.2 `convert_df_to_categoric`: Convertir cada columna de un data frame a variables carácter

La segmentación, o el criterio de discretización para cualquier variable numérica es igual frecuencia. Las variables factor son convertidas directamente a variables carácter.

```
iris_char=convert_df_to_categoric(data = iris, n_bins = 5)
```

```r
# Verificar las primeras filas
head(iris_char)
```

10.2.4.3 `equal_freq`: Convertir variable numérica a categórica

Convierte un vector numérico en factor usando el criterio de igual frecuencia.

```r
new_age=equal_freq(heart_disease$age, n_bins = 5)

# Verificar los resultados
Hmisc::describe(new_age)
```

```
## new_age
##        n  missing distinct
##      303        0        5
##
## Value      [29,46) [46,54) [54,59) [59,63) [63,77]
## Frequency       63      64      71      45      60
## Proportion    0.21    0.21    0.23    0.15    0.20
```

[**Leer más aquí.**]

Notas:

- A diferencia de `discretize_get_bins`, esta función no inserta `-Inf` y `Inf` como los valores mín y máx, respectivamente.

10.2.4.4 `range01`: Escala la variable en el rango de 0 a 1

Convierte un vector numérico a una escala que va de 0 a 1, donde 0 es el mínimo y 1 es el máximo.

```r
age_scaled=range01(heart_disease$oldpeak)

# Verificar los resultados
summary(age_scaled)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    0.00    0.13    0.17    0.26    1.00
```

10.2.5 Preparación de datos con valores atípicos

10.2.5.1 `hampel_outlier` y `tukey_outlier`: Obtiene el umbral de los valores atípicos

Ambas funciones obtienen un vector de dos valores que indica el umbral en el que los valores son considerados atípicos. Las funciones `tukey_outlier` y `hampel_outlier` son utilizadas internamente en `prep_outliers`.

Usando el método de Tukey:

```
tukey_outlier(heart_disease$resting_blood_pressure)
```

```
## bottom_threshold    top_threshold
##               60              200
```

[**Leer más aquí.**]

Usando el método de Hampel:

```
hampel_outlier(heart_disease$resting_blood_pressure)
```

```
## bottom_threshold    top_threshold
##               86              174
```

[**Leer más aquí.**]

10.2.5.2 `prep_outliers`: Preparar los valores atípicos de un data frame

Toma un data frame y devuelve el mismo data frame más las transformaciones especificadas en el parámetro `input`. También funciona con un sólo vector.

Ejemplo tomando dos variables como datos de entrada:

```
# Obtener el umbral según el método de Hampel
hampel_outlier(heart_disease$max_heart_rate)
```

```
## bottom_threshold    top_threshold
##               86              220
```

```
# Aplicar la función para frenar los valores atípicos en los valores
data_prep=prep_outliers(data = heart_disease, input = c('max_heart_r
```

Verificar el antes y después de la variable `max_heart_rate`:

```
## [1] "Before transformation -> Min: 71; Max: 202"
```

```
## [1] "After transformation -> Min: 86.283; Max: 202"
```

El valor mínimo cambió de 71 a 86.23, mientras que el valor máximo queda igual, en 202.

Notas:

- `method` puede ser: `bottom_top`, `tukey` o `hampel`.
- `type` puede ser: `stop` o `set_na`. Si es `stop` todos los valores marcados como atípicos serán frenados en el umbral. Si es `set_na`, entonces los valores marcados serán tomados como `NA`.

[**Leer más aquí.**]

10.2.6 Desempeño de un modelo predictivo

10.2.6.1 `gain_lift`: Curva de desempeño de ganancia y lift

Después de calcular los scores o probabilidades de la clase que queremos predecir, los pasamos a la función `gain_lift`, que devuelve un data frame con métricas de desempeño.

```
# Crear un modelo de machine learning y obtener sus puntajes para ca
fit_glm=glm(has_heart_disease ~ age + oldpeak, data=heart_disease, f
heart_disease$score=predict(fit_glm, newdata=heart_disease, type='re

# Calcular las métricas de desempeño
gain_lift(data=heart_disease, score='score', target='has_heart_disea
```

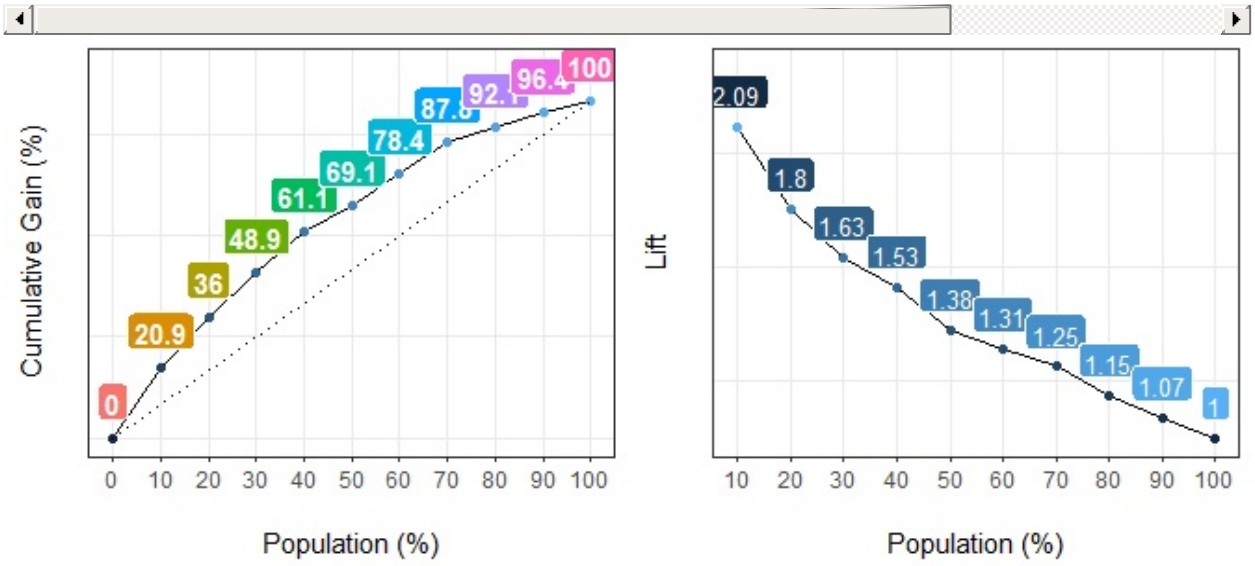


Figure 10.13: ganancia y lift: visualizando el desempeño de un modelo predictivo

```
##    Population Gain Lift Score.Point
## 1          10   21  2.1        0.82
## 2          20   36  1.8        0.70
## 3          30   49  1.6        0.57
## 4          40   61  1.5        0.49
## 5          50   69  1.4        0.40
## 6          60   78  1.3        0.33
## 7          70   88  1.2        0.29
## 8          80   92  1.1        0.25
## 9          90   96  1.1        0.20
## 10        100  100  1.0        0.12
```

[**Leer más aquí.**]

Libro Vivo de Ciencia de Datos

11 Descargar libro

Si les gustó el libro y quieren apoyar el proyecto, pueden adquirir para siempre y toda la eternidad el **Libro Vivo de Ciencia de Datos** portátil en versión PDF, mobi, y Kindle. ¡Las tres juntas! .

Compra segura por **Gumroad** :

Nota: Intenten desactivar la extensión 'Ghostery' durante la compra, se sabe que tiene algunos problemas.

Loading...

Si no apareció el formulario de arriba o desean comprarlo directamente del sitio de Gumroad, sigan este link: https://gumroad.com/l/AKxFW

¿Hay alguna diferencia entre las versiones online, PDF, mobi o Kindle?

No, son todas iguales. El **Libro Vivo de Ciencia de Datos** es y siempre será completamente accesible, de hecho es de código abierto (`Attribution-NonCommercial-ShareAlike 4.0 International`).

¿Qué pasa si sale una nueva versión? / ¿Puedo acceder a las versiones más nuevas?

¡Claro! Después de la compra, cada tanto serán notificados vía e-mail sobre la publicación de una nueva versión (pueden desactivar esta notificación).

La modalidad es que cada uno le asigna el precio que considere justo, con un mínimo de 5 dólares estadounidenses. ¡Feliz lectura!

¿Cómo puedo contactar al autor?

Si desean contactarme pueden hacerlo a pcasas.biz gmail.com, no necesitan un

motivo específico; ¡puede ser por cualquier cosa, desde sugerir un nuevo tema hasta simplemente compartir una buena experiencia que tuvieron al aplicar algún concepto que aprendieron aquí, es bienvenida!

: *No tengo suficiente dinero para contribuir, pero igual* realmente *quiero la versión portátil.* No hay problema, todos hemos pasado por malos momentos, simplemente escríbanme a pcasas.biz (arroba) gmail.com ☺.

Me gusta la idea de tener -cuando es posible- acceso gratuito a la educación, tal es el caso de la educación universitaria en Argentina, Finlandia (entre otros pocos países). Es por eso que cualquiera que desee estudiar y explorar nuevos horizontes puede hacerlo. El dinero no debería ser una barrera a la hora de adquirir conocimiento.

Manténganse al tanto en twitter.

Referencias

Amatriain, Xavier. 2015. “In Machine Learning, What Is Better: More Data or Better Algorithms.” http://www.kdnuggets.com/2015/06/machine-learning-more-data-better-algorithms.html.

Caban, Jesus J., Ulas Bagci, Alem Mehari, Shoaib Alam, Joseph R. Fontana, Gregory J. Kato, and Daniel J. Mollura. 2012. “Characterizing Non-Linear Dependencies Among Pairs of Clinical Variables and Imaging Data.” *Conf Proc IEEE Eng Med Biol Soc* 2012 (August): 2700–2703. doi:10.1109/EMBC.2012.6346521.

Fernandez-Delgado, Manuel. 2014. “Do We Need Hundreds of Classifiers to Solve Real World Classification Problems?” http://jmlr.csail.mit.edu/papers/volume15/delgado14a/delgado14a.pdf.

Fortmann, Scott. 2012. “Understanding the Bias-Variance Tradeoff.” http://scott.fortmann-roe.com/docs/BiasVariance.html.

Handbook, Engineering Statistics. 2013. “Measures of Skewness and Kurtosis.” http://www.itl.nist.gov/div898/handbook/eda/section3/eda35b.htm.

Hyndman, Rob J. 2010. “Why Every Statistician Should Know About Cross-Validation?” https://robjhyndman.com/hyndsight/crossvalidation/.

———. 2017. “ARIMA Modelling in R.” https://www.otexts.org/fpp/8/.

Izbicki, Mike. 2011. “Converting Images into Time Series for Data Mining.” https://izbicki.me/blog/converting-images-into-time-series-for-data-mining.html.

Kuhn, Max. 2017. “Recursive Feature Elimination in R Package Caret.” https://topepo.github.io/caret/recursive-feature-elimination.html.

McNeese, Bill. 2016. “Are the Skewness and Kurtosis Useful Statistics?” https://www.spcforexcel.com/knowledge/basic-statistics/are-skewness-and-kurtosis-useful-statistics.

Raschka, Sebastian. 2017. “Machine Learning Faq.”

http://sebastianraschka.com/faq/docs/evaluate-a-model.html.

Reshef, David N., Yakir A. Reshef, Hilary K. Finucane, Sharon R. Grossman, Gilean McVean, Peter J. Turnbaugh, Eric S. Lander, Michael Mitzenmacher, and Pardis C. Sabeti. 2011. “Detecting Novel Associations in Large Data Sets.” *Science* 334 (6062): 1518–24. doi:10.1126/science.1205438.

stackoverflow.com. 2017. “What Is Entropy and Information Gain?” http://stackoverflow.com/questions/1859554/what-is-entropy-and-information-gain.

stats.stackexchange.com. 2015. “Gradient Boosting Machine Vs Random Forest.” https://stats.stackexchange.com/questions/173390/gradient-boosting-tree-vs-random-forest.

———. 2017a. “How to Interpret Mean Decrease in Accuracy and Mean Decrease Gini in Random Forest Models.” http://stats.stackexchange.com/questions/197827/how-to-interpret-mean-decrease-in-accuracy-and-mean-decrease-gini-in-random-fore.

———. 2017b. “Percentile Vs Quantile Vs Quartile.” https://stats.stackexchange.com/questions/156778/percentile-vs-quantile-vs-quartile.

Suisse, Credit. 2013. “Global Wealth Report 2013.” https://publications.credit-suisse.com/tasks/render/file/?fileID=BCDB1364-A105-0560-1332EC9100FF5C83.

Wikipedia. 2017a. “Distribution of Wealth.” https://en.wikipedia.org/wiki/Distribution_of_wealth.

———. 2017b. “Monotonic Function.” https://en.wikipedia.org/wiki/Monotonic_function.

———. 2017c. “Occam’s Razor.” https://en.wikipedia.org/wiki/Occam's_razor#Probability_theory_and_statistics.

———. 2017d. “White Noise - Time Series Analysis and Regression.” https://en.wikipedia.org/wiki/White_noise.

Xie, Yihui. 2019. *Bookdown: Authoring Books and Technical Documents with R Markdown*. https://CRAN.R-project.org/package=bookdown.