Data Science en RStudio

Capacitación para el uso del software RStudio

Contents

En este curso damos inicio al manejo del software RStudio, en el cuál aprenderemos, creación e importación de dataset, análisis univariado, análisis bivariado, modelamiento predictivo e interpretación de resultados.

1 Introducción a RStudio

Primero te enseñaremos algunos comandos básicos que podrían ayudarte durante el curso. Para manejar mejor RStudio y hacer que esta herramienta sea aún más amigable.

- rm(list=ls()) : remueve objetos creados.
- ?matrix : te llevara a CRAN, la ayuda de Rstudio.
- getwd() : Te indica la ruta donde se encuentra Rstudio.
- setwd() : Le entregas la ruta donde guardaras tus script.
- R.version: Obtiene la versión de R.
- sessionInfo(): Obtiene información de R.

1.1 Instalación de librerías

Las librerías son una colección de funciones que nos ayudarán a realizar diferentes análisis descriptivos y predictivos. La sentencia encargada de ejecutar la instación del packages y la ejecució de las librerías es:

```
install.packages("nombre_packages")
library(nombre_packages)
```

1.2 Operaciones básicas

1.2.1 Tipos de variables

La clasificación de variables permitirá, posteriormente, seleccionar las medidas descriptivas y análisis estadísticos adecuados. Las variables se clasifican de acuerdo al **nivel de medición** y **tamaño del recorrido.**

El recorrido de una variable estadística es el conjunto de todas los valores o categorías que ésta pueda tomar.

Nivel de Medición.

- **Nominales:** toman valores cualitativos o categóricos, es decir, sirven para distinguir entre diferentes categorías.

Por ejemplo: nacionalidad, género, raza, color de un producto.

- **Ordinales:** variables categóricas que presentan un orden intrínseco, jerárquico y/o secuencial, que presentan relaciones entre los niveles.

Por ejemplo: cargo en la Administración Pública, nivel educacional, nivel socioeconómico.

- **Intervalares:**: variables continuas numéricas que presentan el hecho de que las distancias entre dos puntos cualquiera de la escala son de tamaño conocido. Por lo tanto, existe una unidad de medida constante y común a cada par de sujetos del conjunto ordenado. Es decir, la razón de dos intervalos cualesquiera es independiente de la unidad de medida y del punto cero elegido.

Por ejemplo, la temperatura, que se puede medir en grados Celsius o Farenheit.

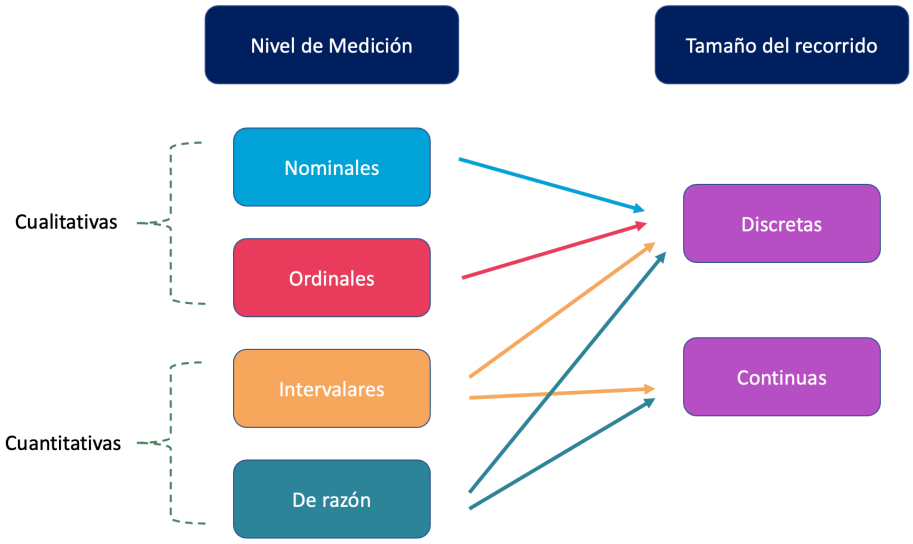
- **De Razón:** Son variables con características como las de intervalos, pero se agrega el hecho de tener un cero absoluto como origen. Además, la razón entre dos pares de puntos cualquiera es independiente de la unidad de medida.

Por ejemplo, la edad, peso de un animal, estatura, distancia entre dos puntos.

Tamaño de Recorrido

- **Discretas:** presentan un recorrido de un conjunto finito o infinito numerable. Por ejemplo, aquellas que toman valores en los números enteros.

- **Continuas:** el conjunto recorrido es infinito no numerable. Toman valores en algún intervalos dentro de los número reales.



Tenemos diferentes tipos de variables en R, que serán analizadas de diferentes maneras:

- **Numeric:** Valores numéricos, incluye decimales.
- **Integer:** Números enteros, no incluye decimales.
- **Character:** Valores alfanuméricos, es decir, letras, números y signos mezclados.

- **Logical:** Valores lógicos, TRUE o FALSE.

1. Creación de vector

El vector se crea en una variable x e y. Este vector puede tomar más de un valor, y se define de la siguiente manera:

```
x <- c(3,4,6)
y <-c(2,1,4)
```

2. Extraer elementos de un vector

Si quisieramos un valor exacto del vector que generamos. Tenemos que extraer ese valor conociendo su posición dentro de este, de la siguiente manera.

```
x[1] # Extraer el  1er elemento de un vector
```

```
## [1] 3
```

```
x[1:3] # Extraer los primeros 3 elementos de un vector
```

```
## [1] 3 4 6
```

3. Operaciones entre vectores

También podemos generar operaciones básicas entre distintas variables.

```
x+2 # Sumar 2 a cada elemento de un vector
x*2 # Multiplicar por 2 a cada elemento de un ve ctor
x/2 # Dividir por 2 a cada elemento de un vector
x^2 # Elevar 2 a cada elemento de un vector

x+y # Sumar dos vectores
x*y # Multiplicar dos vectores x/y # Dividir dos vectores
x^y # Elevar dos vectores
```

4. Creación de Matrices

Generamos una matriz (nxm), es decir, n filas y m columnas:

```
M <- matrix(1:12, nrow = 3, ncol = 4)
M
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

Extraer elementos de una matriz y realizar operaciones con ella.

```r
M[1,2] # Extraer fila 1 columna 2 M[1,] # Extraer fila 1
```

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

```r
M[,2] # Extraer columna 1
```

```
## [1] 4 5 6
```

Generamos las siguientes matrices:

```r
M1 <- matrix(1:12, nrow = 3, ncol = 4)
M2 <- matrix(2:25, nrow = 3, ncol = 4)
M3 <- matrix(2:25, nrow = 4, ncol = 3)
```

Y realizamos operaciones con matrices, suma de matrices, producto matricial y producto elemento a elemento,

```r
M1+M2  #Suma de matrices
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    3    9   15   21
## [2,]    5   11   17   23
## [3,]    7   13   19   25
```

```r
M1%*%M3 #Producto matricial
```

```
##      [,1] [,2] [,3]
## [1,]   92  180  268
## [2,]  106  210  314
## [3,]  120  240  360
```

```r
M1*M2 #Producto elemento a elemento
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2   20   56  110
## [2,]    6   30   72  132
## [3,]   12   42   90  156
```

5. Creación de secuencias

Si quisieramos detallar una variable con varios valores numéricos, sin la necesitar de escribir uno a uno podemos generar un secuencia de valores con la función **seq**.

```r
seq(from = 1, to = 5)
```

```
## [1] 1 2 3 4 5
```

```r
seq(from = 2, by = -0.1, length.out = 4)
```

```
## [1] 2.0 1.9 1.8 1.7
```

6. Funciones de interés

Al generar un vector o asignar un valor a una variable, se pueden realizar diferentes tipos de funciones matemáticas, que nos ayudan a extraer información, de la siguiente manera.

```r
length(x) #largo de x
exp(x)    #función exponencial
sqrt(x)   #raiz cuadrada
log(x)    #funcion logaritmo
```

7. Crear nuevas funciones

Si se necesitan generar nuevas funciones, se tiene el comando **function**, en el cuál, se les puede entregar un argumento o input (x) y este retornará un valor (y). Este comando tiene la siguiente estructura:

```r
new_funcion <- function(x){
  y=funcion
  y
}

new_funcion(x)
```

Podemos tener funciones de lo que tu estimes conveniente, pero siempre con un objetivo en común, aligerar trabajo y aumentar rapidez. Algunos ejemplos de funciones son:

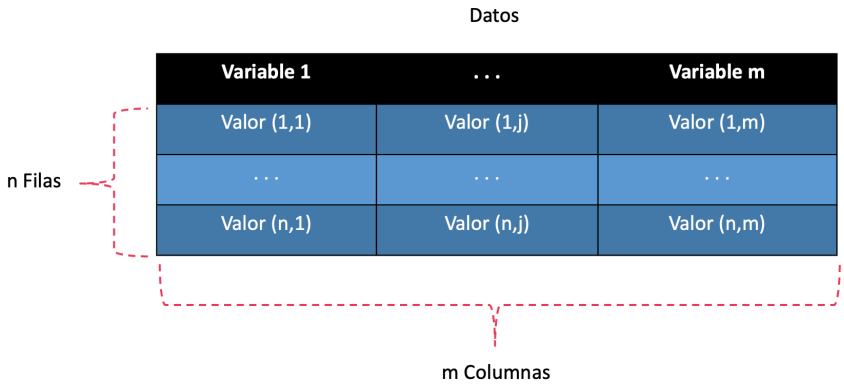
```r
coef_variacion <- function(x, na.rm = FALSE) {
  sd(x, na.rm=na.rm) / mean(x, na.rm=na.rm)
}

x<-c(124,537,289,267,784,382)
coef_variacion(x)
```

```
## [1] 0.5879959
```

2 Proceso Data Mining

Para la generación del proceso data mining conoceremos algunos comando previos que nos ayudarán a trabajar nuestros datos. Los datos se encuentran de la siguiente manera. Cada base de datos es construida por n filas y m columnas, en donde cada columna tendrá el nombre de una variable. Y este conjunto de objetos nos entregarán valores especificos que podremos analizar.



En la siguiente sesión revisaremos el tratamiento y análisis de dataset, y también la creación de dataframe, a través de valores ingresados por el usuario de la siguiente manera, generamos las columnas para luego crear mi dataset df.

```
columa1 <- c(value_1, value_2, ...)
columa2 <- c(value_1, value_, ...)
df <- data.frame(colunma1,columna2)
```

Luego, lo primero que realizamos para entender nuestra base de datos son un set de comandos que deben ser utilizados:

```
View(df)     #Ver en una pestañna nueva los datos
attach(df)   #Cargar los nombdres del datset
head(df)     #extraer 6 primeras filas
tail(df)     #ultimas 6 filas
names(df)    #nombres de las columnas
dim(df)      #filas por columnas
nrow(df)     #numero de filas
ncol(df)     #numero de columnas
```

Si necesitamos cargar base de datos que provengan de otras extensiones, tenemos diferentes sentencias dependiendo del archivo a cargar:

1. Importar archivos excel, utilizando la librería readxl

Se le puede indicar también el tipo de columna que es cada variables (numeric, character, double, etc) con la sentensia **col_types**.

```
read_excel("path/file.xlsx", col_types=c())
```

2. Importar archivos csv, utilizando la librería readr

```
 read_csv("path/file.csv")
```

3. Importar archivos .sas, utilizando la librería heaven

```
read_sas("path/file.sas")
```

4. Leer archivos .rds

```
readRDS("path/file.rds")
```

5. Leer tablas txt

Tenemos dos rutas para cargar un dataset, en donde podemos declarar:

- col.names=c() : Los nombres de las columnas.
- header=T : Indica que tienen los nombres de las variables.
- sep="" : Que tiene de separación tiene cada columna en el archivo.

```
#Selecionando archivo:
read.table(file.choose(),col.names=c(), header=T, sep=";")

#Ingresando ruta:
read.table("path/file.txt",col.names=c(), header=T, sep=";")
```

6. Crear data frame

Otra manera de generar un dataframe es a través de la siguiente sentencia:

```
x <- c(1,2,3,4)
y <- c("A","B","C","D")
df <- as.data.frame(cbind(x,y))
```

El análisis estadístico al igual que el proceso de Data Mining, nos ayuda a realizar diferentes etapas para llegar al conocimiento e interpretación de los datos. De las cuales podemos destacar las siguientes:

1. Análisis Descriptivo: El análisis descriptivo o análisis univariado nos permite describir la naturaleza de los datos, es decir, obtener información que nos ayude a la interpretación a partir de valor cuantificados, es aqui donde encontramos medidas de tendencia central, dispersión y graficos de densidades.

2. Inferencia Estadística:: Nos ayuda a explorar la relación de los datos con la población, generar conclusiones poblacionales a través de estimaciones muéstrales.

3. Modelamiento Estadístico: Se realiza un modelo para predecir una variable de interés o variable respuesta, planteando correctamente los supuestos o variables independientes que nos ayuden a predecir nuestro objetivo.

4. Validación del modelo: Obetenemos pruebas de bondad de ajuste y discriminación para validar que nuestros valores estimados sean lo más cercanos a los valores reales. Estas pruebas se realizan en muestras de validación y holdout, como también en procesos de backtesting.

2.1 Análisis Descriptivo

2.1.1 Medidas de tendencia central y dispersión

Para los análisis a continuación usaremos la data Iris. Dentro de las medidas principales podemos encontrar medidas de tendencia central como la media, moda, y mediana, lo que nos entregará la primera distribución de nuestros datos. Encontrando curvas que con cierta asimetría.

```
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
attach(iris)
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
mean(Sepal.Width)    #promedio
```

```
## [1] 3.057333
```

```
median(Sepal.Width)  #mediana
```

```
## [1] 3
```

```r
sd(Sepal.Width)      #desviacion estandar
```

```
## [1] 0.4358663
```

```r
var(Sepal.Width)     #varianza
```

```
## [1] 0.1899794
```

Para obtener el valor de la moda, necesitamos cargar la librería modeest, pero para no cargar el packages completo podemos ejecutar la siguiente sentendia que nos ejecutará la función especifica que necesitemos.

```r
modeest::mfv(Sepal.Width)  #Libreria :: funcion()
```

```
## [1] 3
```

También, tenemos estadísticos de posición, que nos entregarán valores de cuantiles. Usados para saber en que porcentaje de nuestra data se encuentran centrados nuestros valores.

```r
quantile(Sepal.Width, prob=seq(0,1,1/3))  #probabilidad con secuencia
```

```
##        0% 33.33333% 66.66667%      100%
##       2.0       2.9       3.2       4.4
```

```r
quantile(Sepal.Width)  #probabilidad con vector
```

```
##   0%  25%  50%  75% 100%
##  2.0  2.8  3.0  3.3  4.4
```

2.1.1.1 Estadísticos en base de datos R cuenta con funciones rapidas de análisis descriptivos, los que nos ayudan a trabajar con más variables al mismo tiempo, pero siempre hay que tener cuidado con los diferentes tipos de variables que existen.
Cargaremos la base iris para ir desarrollando los calculos:

```r
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
attach(iris)
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

La función summary, como su nombre nos indica, entrega un resumen de estadísticos más comunes, para todas las variables que esten dentro de nuestro dataset "iris".

Siempre hay que tener en consideración los tipos de variables que tenemos dentro del dataset, ya que los resultados del summary, harán sentido para variables numeric y factor.

```
class(iris)  # nos indica el tipo base
```

```
## [1] "data.frame"
```

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

Otra función que puede ser de utilidad, siempre y cuandos la cantidad de filas con valores missing sea no significativo, podrían omitirse valores NA o NULL.

```
na.omit(iris)    #elimina filas con valores NA
```

Tenemos diferentes funciones para indicar o preguntar si una variables es tipo factor, numrica, character, o string.

- **is.numeric:** es o no numerico.

- **is.character:** es o no character.
- **is.factor:** es o no categorico.
- **as.factor:** Indicar que una variable es categórica.
- **class:**: tipo de dataset.

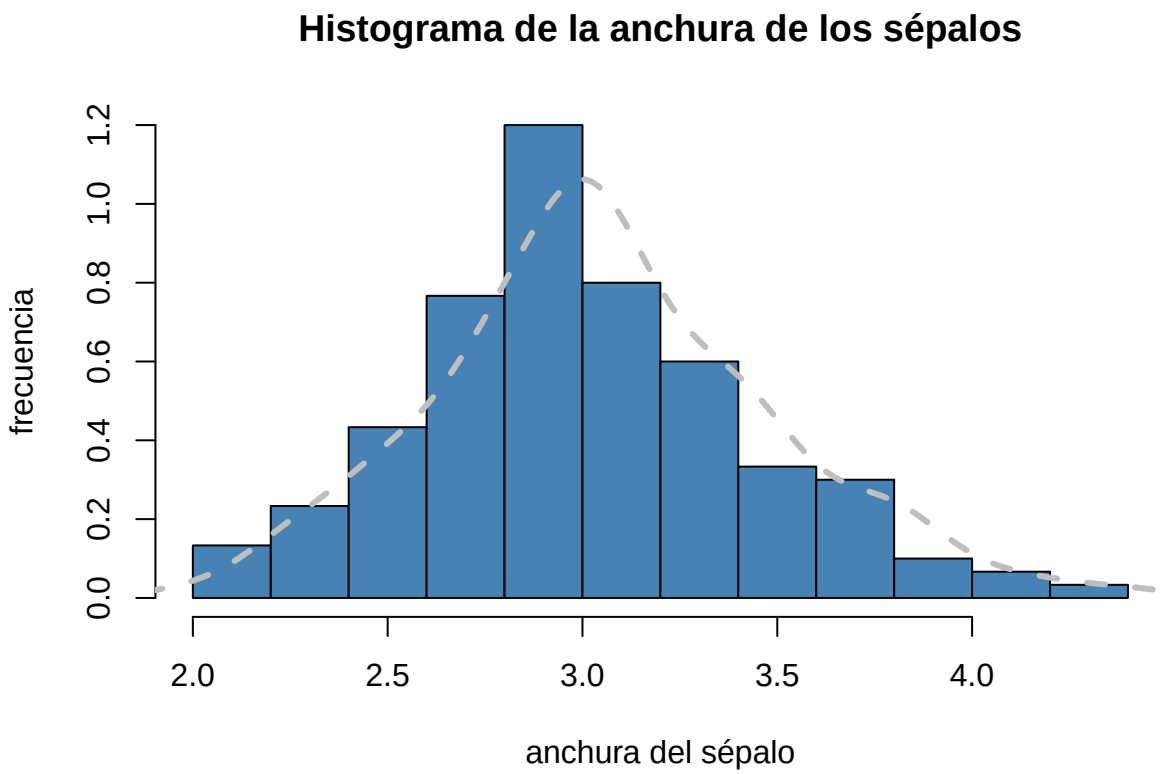
```
Species=as.factor(Species)
```

Otras medidas que nos ayudan a entender el comportamiento de nuestras variables son los gráficos de diferentes estilos que se generan a través de sentencias simples

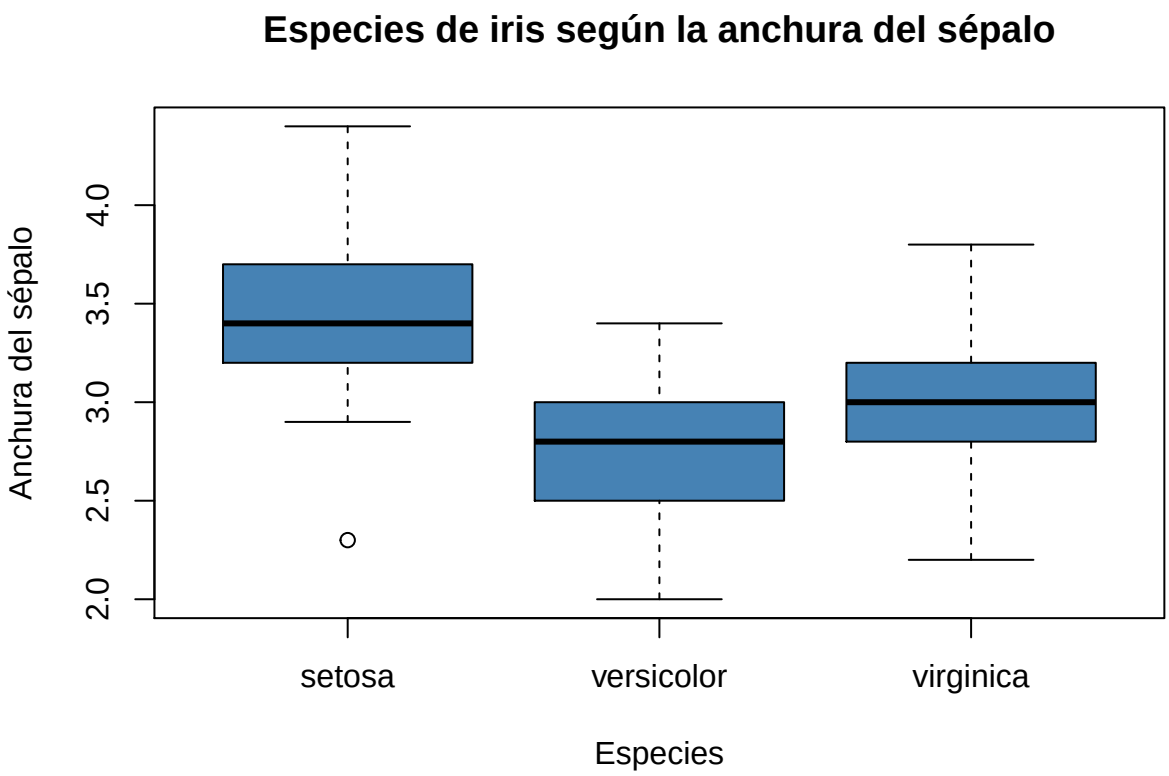
Estas gráficas varían según las diferentes variables que podemos tener:

Variables continuas: Realizaremos histogramas o boxplot.

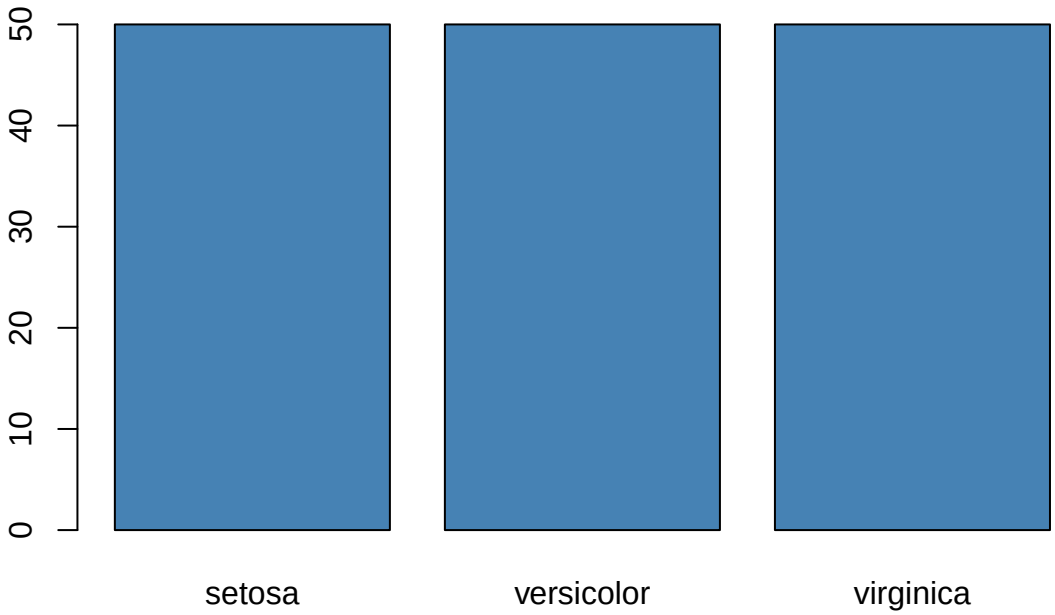
A la variables tipo continuas, se les puede aplicar la función **hist**, la que nos entregará la frecuencia en intervalos en cada barra, en donde se le puede agregar la curva de la densidad.



Como también se pueden crear boxplot, donde nos entregará valores outliers, minimo, primer cuartil (Q_1), mediana (Q_2), tercer cuartil (Q_3) y máximo. El cuál, nos mostrará si los datos varían en cada categoría.

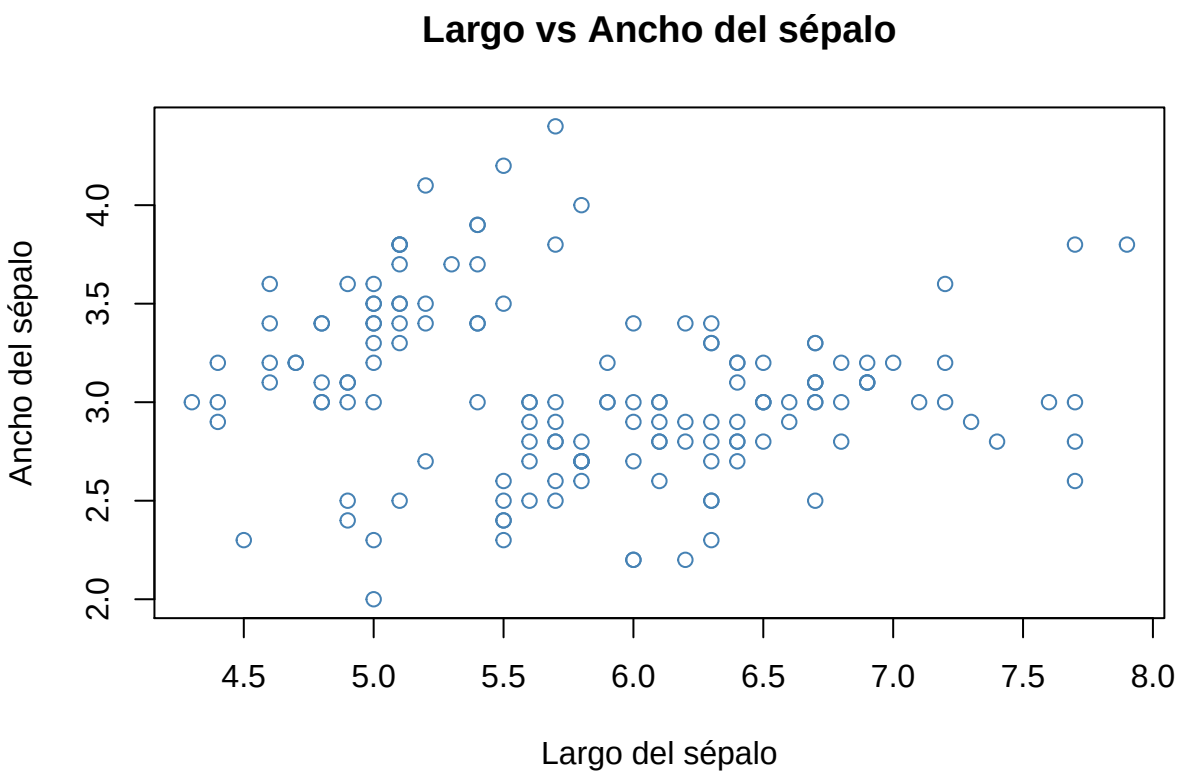


Variables categóricas:: Realizaremos gráfico de barras.



Gráficos de dispersión

Se realizan gráficos de dispersión de dos variables continuas.



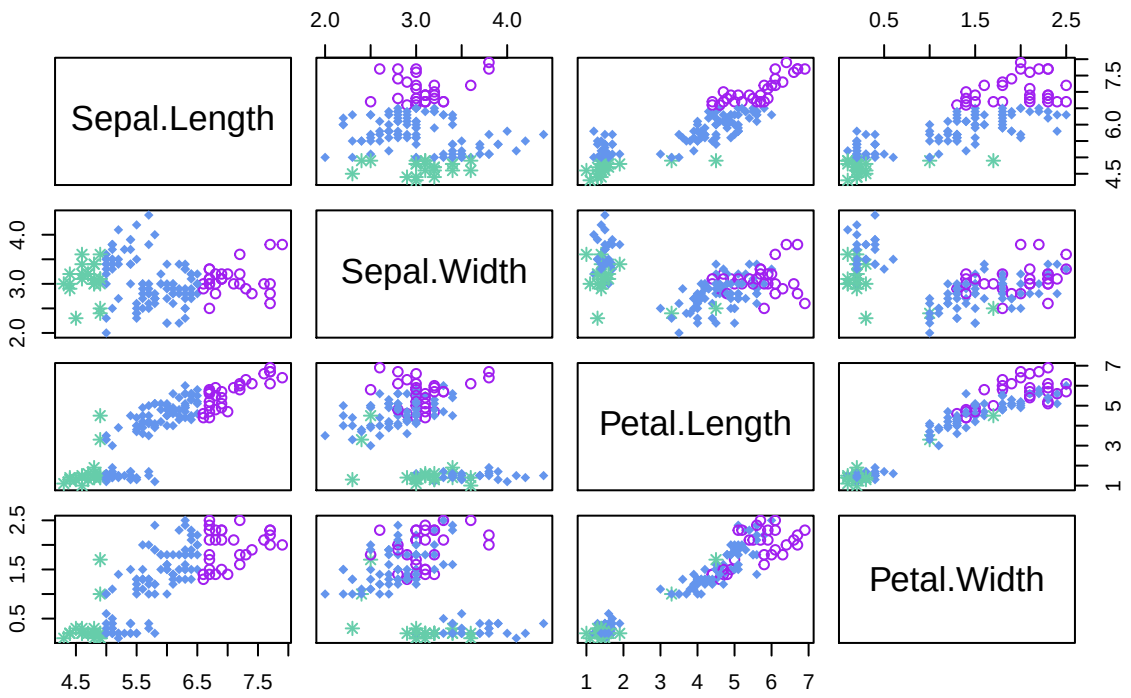
Si quisieramos representar una variable que está ordenada temporalmente podemos agregar la opción type="l".

```
plot(x, type = "l")
```

Si necesitamos graficar la correlación de todas las variables de un dataset, podemos realizarlo a través de la función pairs.

```
#se realiza agrupación de la variable Sepal.Length
group <- NA
group[iris$Sepal.Length < 5.5] <- 1
group[iris$Sepal.Length >= 5 & iris$Sepal.Length <= 6.5] <- 2
group[iris$Sepal.Length > 6.5] <- 3

pairs(iris[,1:4],
      col = c("aquamarine3", "cornflowerblue", "purple")[group],#Cambiar color x group
      pch = c(8, 18, 1)[group],                  #Cambiar punto x group
      labels = c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width"),
      main = "Correlacion de variables Iris")
```



2.1.2 Manipulación base de datos

Una de las librerías más usadas actualmente para la manipulación de datos, es **dplyr**, esta librería cuenta con numerosas funciones que nos ayudan a trabajar los dataset de manera rapida. Si necesitamos encontrar dataset o funciones que nos ayuden a los diferentes procesos, podemos consultar la página https://github.com .

Para usar esta librería requerimos importarla:

```
library(dplyr)
```

Luego, antes de empezar con las funciones aprenderemos el operador pipe "%>%", el cuál se encarga de optimizar los codigos en R. Este operador indica hacia la derecha la función que se le aplicara al dataset que se encuentra al izquierda de el.

```
data %>% funcion(x)
```

2.1.2.1 Función select Nos permite obtener las columnas que se indiquen de un dataset y puede usarse de dos maneras.

```
select(data,x,y)
data %>% select(x,y)
```

```
df<-select(iris,Sepal.Length,Sepal.Width)
df<- iris %>% select(Sepal.Length,Sepal.Width)
```

Si quisieramos seleccionar todo menos la variable **sepal.length**:

```
iris%>% select(-Sepal.Length) %>% head()
```

```
##   Sepal.Width Petal.Length Petal.Width Species
## 1         3.5          1.4         0.2  setosa
## 2         3.0          1.4         0.2  setosa
## 3         3.2          1.3         0.2  setosa
## 4         3.1          1.5         0.2  setosa
## 5         3.6          1.4         0.2  setosa
## 6         3.9          1.7         0.4  setosa
```

2.1.2.2 Función Summary y summarice Realiza estadísticos de resumen, que debes ir agregando uno a uno en la función summarice, con un respectivo nombre.

```
iris%>% select(Sepal.Length,Sepal.Width)  %>% summary()
```

```
##   Sepal.Length    Sepal.Width
##  Min.   :4.300   Min.   :2.000
##  1st Qu.:5.100   1st Qu.:2.800
##  Median :5.800   Median :3.000
##  Mean   :5.843   Mean   :3.057
##  3rd Qu.:6.400   3rd Qu.:3.300
##  Max.   :7.900   Max.   :4.400
```

```
iris  %>% summarise(media=mean(Sepal.Width), mediana=median(Sepal.Width), desv_stan=sd(Sepal.Width))
```

```
##      media mediana desv_stan
## 1 3.057333       3 0.4358663
```

2.1.2.3 Función filter Nos sirve para filtrar observaciones que queremos excluir o incluir dentro de nuestro análisis.

```
iris %>% filter(Sepal.Width>0.2) %>% head()
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
iris %>% filter(Species=="setosa") %>% head()
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Algunos operadores que podrías usar dentro de la función filter son:

- **Igual:** ==
- **Distinto:** !=
- **Menor/mayor:** < o >
- **Menor igual/mayor igual:** <= o >=
- **En:** %in%

2.1.2.4 Función mutate Nos sirve para generar nuevas variables a través de transformaciones que se necesiten realizar.

```
iris %>% mutate(new_var=Sepal.Width*3) %>% head()
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species new_var
## 1          5.1         3.5          1.4         0.2  setosa    10.5
## 2          4.9         3.0          1.4         0.2  setosa     9.0
```

```
## 3          4.7         3.2          1.3         0.2  setosa     9.6
## 4          4.6         3.1          1.5         0.2  setosa     9.3
## 5          5.0         3.6          1.4         0.2  setosa    10.8
## 6          5.4         3.9          1.7         0.4  setosa    11.7
```

2.1.2.5 Función group by Cuando se necesitan realizar análisis de variables continuas agrupadas por alguna variables categórica.

```
iris %>% group_by(Species) %>% summarise(media=mean(Sepal.Length))
```

```
## # A tibble: 3 x 2
##   Species    media
##   <fct>      <dbl>
## 1 setosa      5.01
## 2 versicolor  5.94
## 3 virginica   6.59
```

Ejemplo 1.

En este ejemplo podemos revisar que se usan las funciones **filter** para filtrar solo las especies setosa, y luego se realiza una transformación con la función **mutate** a la variable Sepal.lenght, y por último, una media a la nueva variable creada.

```
iris %>% filter(Species=="setosa") %>% mutate(var_new=Sepal.Length^2) %>% summarise(media=mean(var_new))
```

```
##     media
## 1 25.1818
```

Ejemplo 2.

Cargamos la base, y realizamos transformaciones,y usamos las funciones **factor**, para convertir a factor y ordenar los nivles de una variable.

```
getwd()
```

```
## [1] "/Volumes/GoogleDrive/Mi unidad/Academia de estadística/Cursos/PERSONALIZADOS/R_clase1"
```

```r
#setwd("Google Drive/Mi unidad/Academia de estadística/Cursos/PERSONALIZADOS/R_clase1")
base=read.table("base_txt.txt", h=T)
head(base)
```

```
##   edad genero fuma hijos ingreso
## 1   22  lgtbi   si     2     368
## 2   20  lgtbi   si     5     339
## 3   25  lgtbi   si     0     588
## 4   18  lgtbi   si     4     292
## 5   15  lgtbi   si     5     699
## 6   20  lgtbi   si     4     347
```

```r
attach(base)
table(genero)
```

```
## genero
##     f lgtbi     m
##   232    62   206
```

```r
base_new=base %>%
mutate(genero2=factor(genero,levels = c("f","m", "lgtbi")),  #Ordena la variable genero por los niveles
edad2=factor(ifelse( edad >= 18, "Mayor de edad", "Menor de edad"), #Si es mayor a 18, asigna Mayor de
                     levels = c("Menor de edad", "Mayor de edad")),    #lo ordena en el orden indicado
    fuma2=factor(fuma, levels = c("si", "no")),
    estrato=factor(case_when(between(ingreso,0,324)~"E",       #Crea los agrupacion de ingreso
                             between(ingreso,324,562)~"D",
                             between(ingreso,562,899)~"C3",
                             between(ingreso,899,1360)~"C2",
                             between(ingreso,1360,1986)~"C1B",
                             between(ingreso,1986,2739)~"C1A",
                             TRUE~"AB"),
                       levels=c("E","D","C3","C2","C1B","C1A","AB")))

attach(base_new)
head(base_new)
```

```
##   edad genero fuma hijos ingreso genero2         edad2 fuma2 estrato
## 1   22  lgtbi   si     2     368   lgtbi Mayor de edad    si       D
## 2   20  lgtbi   si     5     339   lgtbi Mayor de edad    si       D
## 3   25  lgtbi   si     0     588   lgtbi Mayor de edad    si      C3
## 4   18  lgtbi   si     4     292   lgtbi Mayor de edad    si       E
## 5   15  lgtbi   si     5     699   lgtbi Menor de edad    si      C3
## 6   20  lgtbi   si     4     347   lgtbi Mayor de edad    si       D
```

```r
table(genero2)
```

```
## genero2
##     f     m lgtbi
##   232   206    62
```

Además, existen modificaciones de estas funciones en el cuál se les puede agregar al final de cada una de ellas la terminación "_if","_at" , "_all". algunos ejemplos son:

```
#toma variables que cumplan el criterio numerico
iris %>% select_if(is.numeric) %>% head()
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1          5.1         3.5          1.4         0.2
## 2          4.9         3.0          1.4         0.2
## 3          4.7         3.2          1.3         0.2
## 4          4.6         3.1          1.5         0.2
## 5          5.0         3.6          1.4         0.2
## 6          5.4         3.9          1.7         0.4
```

```
#_at: realizar una operacion a las variables que contengan el nombre en parentesis.
iris %>% select_at(c("Sepal.Length")) %>% head()
```

```
##   Sepal.Length
## 1          5.1
## 2          4.9
## 3          4.7
## 4          4.6
## 5          5.0
## 6          5.4
```

```
iris %>% select_at(vars(contains("Length"))) %>% head()    #Contengan
```

```
##   Sepal.Length Petal.Length
## 1          5.1          1.4
## 2          4.9          1.4
## 3          4.7          1.3
## 4          4.6          1.5
## 5          5.0          1.4
## 6          5.4          1.7
```

```
iris %>% select_at(vars(starts_with("Sepal"))) %>% head()  #Que empiecen con
```

```
##   Sepal.Length Sepal.Width
## 1          5.1         3.5
## 2          4.9         3.0
## 3          4.7         3.2
## 4          4.6         3.1
## 5          5.0         3.6
## 6          5.4         3.9
```

```
#que la variable tipo factor sea mayuscula.
iris %>% mutate_if(is.factor, toupper) %>% head()
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  SETOSA
## 2          4.9         3.0          1.4         0.2  SETOSA
## 3          4.7         3.2          1.3         0.2  SETOSA
## 4          4.6         3.1          1.5         0.2  SETOSA
## 5          5.0         3.6          1.4         0.2  SETOSA
## 6          5.4         3.9          1.7         0.4  SETOSA
```

```
#media en variables numericas que pertenezcan al grupo Species
iris %>% group_by(Species) %>% summarise_if(is.numeric,"mean")
```

```
## # A tibble: 3 x 5
##   Species    Sepal.Length Sepal.Width Petal.Length Petal.Width
##   <fct>             <dbl>       <dbl>        <dbl>       <dbl>
## 1 setosa             5.01        3.43         1.46       0.246
## 2 versicolor         5.94        2.77         4.26       1.33
## 3 virginica          6.59        2.97         5.55       2.03
```

Tenemos las librerías **expss** y **summarytools**, que con las funciones fre y freq respectivamente nos entregarán las frecuencias para la variable indicada.

```
#install.packages("expss")
library(expss)

#install.packages("summarytools")
#library(summarytools)

#base %>% base_new(genero) %>% expss::fre()
#base %>% base_new(genero) %>% summarytools::freq()

tabla=base_new %>% group_by(genero2) %>% summarise(n=n()) %>%
  mutate(porcentaje=(n/sum(n))*100,
  Porcent=paste0(round(porcentaje),"%"))  #print

tabla
```

```
## # A tibble: 3 x 4
##   genero2     n porcentaje Porcent
##   <fct>   <int>      <dbl> <chr>
## 1 f         232       46.4 46%
## 2 m         206       41.2 41%
## 3 lgtbi      62       12.4 12%
```