Notas de clase del curso de introducción a Data Science

Diego Kozlowski y Natsumi Shokida

2019-08-29

Contents

Chapter 1

Introducción

1.1 Presentación

En los últimos años se han difundido muchas herramientas estadísticas novedosas para el análisis de información socioeconómica y geográfica. En particular el software denominado "R", por tratarse de un software libre, se extiende cada vez más en diferentes disciplinas y recibe el aporte de investigadores e investigadoras en todo el mundo, multiplicando sistemáticamente sus capacidades.

Este programa se destaca, entre otras cosas, por su capacidad de trabajar con grandes volúmenes de información, utilizar múltiples bases de datos en simultáneo, generar reportes, realizar gráficos a nivel de publicación y por su comunidad de usuarios que publican sus sintaxis y comparten sus problemas, hecho que potencia la capacidad de consulta y de crecimiento. A su vez, la expresividad del lenguaje permite diseñar funciones específicas que permiten optimizar de forma personalizada el trabajo cotidiano con R.

1.2 Objetivos del curso

El presente Taller tiene como objetivo principal introducir a los participantes en la ciencia de datos, sobre la base de la utilización del lenguaje R aplicado procesamiento de diferentes bases de datos provistas por el programa de Gobierno Abierto y la Encuesta Permanente de Hogares (EPH) - INDEC. Se apunta a brindar las herramientas necesarias para la gestión de la información, presentación de resultados y algunas técnicas de modelado de datos, de forma tal que los participantes puedan luego avanzar por su cuenta a técnicas más avanzadas.

1.3 Temario

1.3.1 clase 1: Introducción al entorno R:

- Descripción del programa "R". Lógica sintáctica del lenguaje y comandos básicos
- Presentación de la plataforma RStudio para trabajar en "R"
- Caracteres especiales en "R"
- Operadores lógicos y aritméticos
- Definición de Objetos: Valores, Vectores y DataFrames
- Tipos de variable (numérica, de caracteres, lógicas)
- Lectura y Escritura de Archivos

1.3.2 clase 2: Tidyverse:

- Limpieza de Base de datos: Renombrar y recodificar variables, tratamiento de valores faltantes (missing values/ NA´s)
- Seleccionar variables, ordenar y agrupar la base de datos para realizar cálculos

- Creación de nuevas variables
- Aplicar filtros sobre la base de datos
- Construir medidas de resumen de la información
- Tratamiento de variables numéricas (edad, ingresos, horas de trabajo, cantidad de hijos / componentes del hogar, entre otras).

1.3.3 clase 3: Estadística descriptiva

- Introducción a probabilidad
- Introducción a distribuciones
- El problema de la inversión
- Estadística
- Población y muestra
- Estimadores puntuales, tests de hipótesis
- Boxplots, histogramas y kernels

1.3.4 clase 4: Visualización de la información

- Gráficos básicos de R (función "plot"): Comandos para la visualización ágil de la información
- Gráficos elaborados en R (función "ggplot"):
- Gráficos de línea, barras, Boxplots y distribuciones de densidad
- Parámetros de los gráficos: Leyendas, ejes, títulos, notas, colores
- Gráficos con múltiples cruces de variables.

1.3.5 clase 5: Documentación en R

- Manejo de las extensiones del software "Rmarkdown" y "RNotebook" para elaborar documentos de trabajo, presentaciones interactivas e informes:
- Opciones para mostrar u ocultar código en los reportes
- Definición de tamaño, títulos y formato con el cual se despliegan los gráficos y tablas en el informe
- Caracteres especiales para incluir múltiples recursos en el texto del informe: Links a páginas web, notas al pie, enumeraciones, cambios en el formato de letra (tamaño, negrita, cursiva)
- Código embebido en el texto para automatización de reportes

1.3.6 clase 6: Análisis de encuestas

- Introducción al diseño de encuestas
- Presentación de la Encuesta Permanente de Hogares
- Generación de estadísticos de resumen en muestras estratificadas
- Utilización de los ponderadores

1.3.7 clase 7: Programación funcional

- Estructuras de código condicionales
- Loops
- Creación de funciones a medida del usuario
- Librería purrr para programación funcional

1.3.8 clase 8: Mapas

- Utilización de información geográfica en R
- Elaboración de mapas
- gestión de shapefiles

1.3.9 clase 9: Shiny

- Shiny como reportes dinámicos
- Su utilidad para el análisis exploratorio
- Lógica de servidor- interfaz de usuario
- Extensiones del mundo shiny
- Publicación de resultados

1.3.10 clase 10: Correlación y Modelo Lineal

- Análisis de correlación.
- Presentación conceptual del modelo lineal
- El modelo lineal desde una perspectiva computacional
- Supuestos del modelo lineal
- Modelo lineal en R
- Modelo lineal en el tidyverse

1.3.11 clase 11: Text Mining

- Introducción al análisis de textos
- Limpieza
- Preprocesamiento
- BoW
- Stopwords
- TF-IDF
- Wordcloud
- Escrapeo de Twitter

1.4 Bibliografía de consulta

- GWickham, H., & Grolemund, G. (2016). R for data science: import, tidy, transform, visualize, and model data. " O'Reilly Media, Inc.". https://es.r4ds.hadley.nz/

- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning. New York: springer. http://faculty.marshall.usc.edu/gareth-james/ISL/
- Wickham, Hadley. ggplot2: elegant graphics for data analysis. Springer, 2016. https://ggplot2-book.org/

1.4.1 Librerias a instalar

```
install.packages(c("tidyverse","openxlsx","xlsx",'ggplot2','GGally','ggridges','treema
```

Chapter 2

Introducción a R

En esta primera clase revisaremos los fundamentos de R base y el entorno de RStudio. El objetivo es poder comenzar a utilizar el programa, abrir archivos y empezar a experimentar para ganar confianza.

- Descripción del programa *R*. Lógica sintáctica del lenguaje y comandos básicos
- Presentación de la plataforma RStudio para trabajar en *R*
- Caracteres especiales en *R*
- Operadores lógicos y aritméticos
- Definición de objetos: valores, vectores y DataFrames
- Tipos de variable (numéricas, de caracteres, lógicas)
- Lectura y escritura de archivos

2.1 Explicación

2.1.1 ¿Qué es R?

- Lenguaje para el procesamiento y análisis estadístico de datos
- Software Libre
- Sintaxis Básica: R base
- Sintaxis incremental[1]: El lenguaje se va ampliando por aportes de Universidades, investigadores/as, usuarios/as y empresas privadas, organizados en librerías (o paquetes)
- Comunidad web muy grande para realizar preguntas y despejar dudas. Por ejemplo, en el caso de Buenos Aires contamos con https://www.meetup.com/es-ES/rladies-buenos-aires/ y https://www.meetup.com/es-ES/renbaires/.

[1] Más allá de los comandos elementales, comandos más sofisticados tienen muchas versiones, y algunas quedan en desuso en el tiempo.

Figure 2.1: https://cran.r-project.org/

- Gráficos con calidad de publicación

Uno de los *entornos* más cómodos para utilizar el *lenguaje* **R** es el *programa* **R studio**.

- Rstudio es una empresa que produce productos asociados al lenguaje R, como el programa sobre el que corremos los comandos, y extensiones del lenguaje (librerías).
- El programa es *gratuito* y se puede bajar de la página oficial

2.1.2 Lógica sintáctica.

2.1.2.1 Definición de objetos

Los **Objetos/Elementos** constituyen la categoría esencial del R. De hecho, todo en R es un objeto, y se almacena con un nombre específico que **no debe poseer espacios**. Un número, un vector, una función, la progresión de letras del abecedario, una base de datos, un gráfico, constituyen para R objetos de distinto tipo. Los objetos que vamos creando a medida que trabajamos pueden visualizarse en el panel derecho superior de la pantalla (el *Environment*).

El operador `<-` (**Alt + Guión**) sirve para definir un objeto. **A la izquierda** del `<-` debe ubicarse el nombre que tomará el elemento a crear. **Del lado derecho** debe ir la definición del mismo.

```
A <- 1
```

Por ejemplo, podemos crear el elemento **A**, cuyo valor será 1. Para esto, debemos *correr* el código presionando **Ctrl + Enter**, con el cursor ubicado en cualquier parte de la línea. Al definir un elemento, el mismo queda guardado en el ambiente del programa, y podrá ser utilizado posteriormente para observar su contenido o para realizar una operación con el mismo.

```
A
```

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

```
A+6
```

```
## [1] 7
```

Al correr una linea con el nombre del objeto, la consola del programa nos muestra su contenido. Entre corchetes observamos el número de orden del elemento en cuestión. Si corremos una operación, la consola nos muestra el resultado de la misma.

El operador `=` es **equivalente** a `<-`, pero en la práctica no se utiliza para la definición de objetos.

```
B = 2
B
```

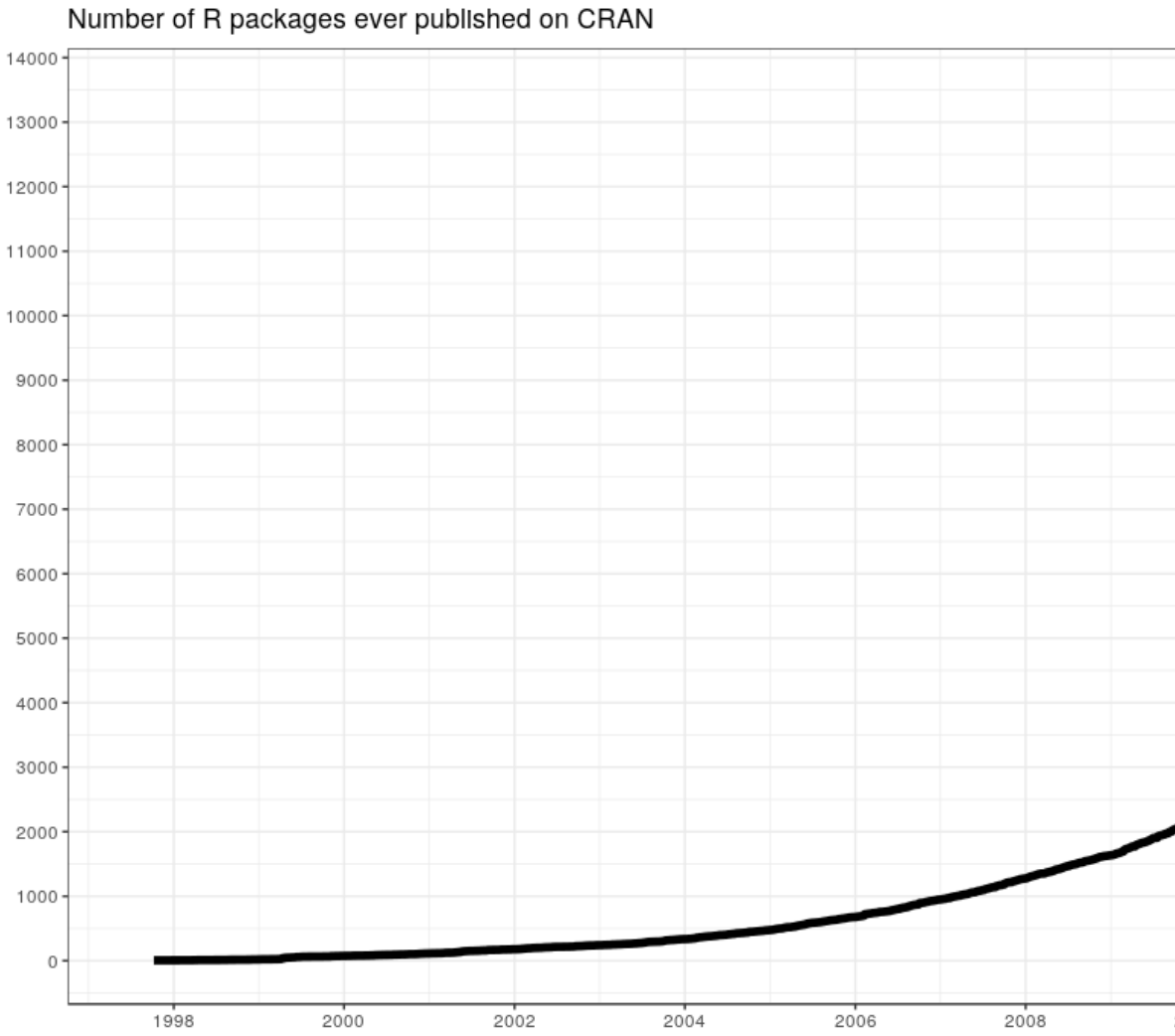


Figure 2.2: fuente: https://gist.github.com/daroczig/3cf06d6db4be2bbe3368

Figure 2.3: https://www.rstudio.com/

Rstudio.bb

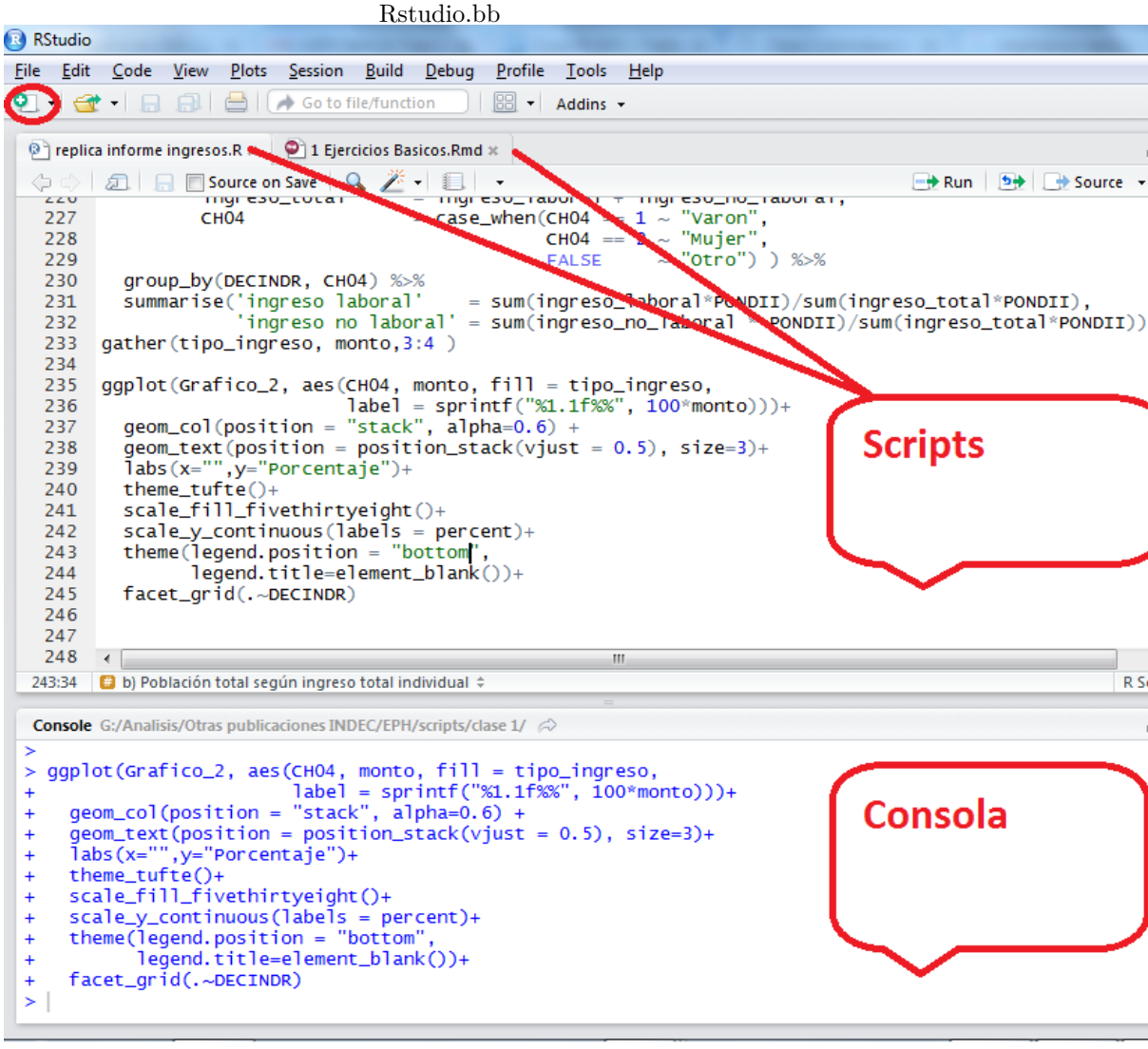


Figure 2.4: Pantalla Rstudio

```
## [1] 2
```

<- es un operador **Unidireccional**, es decir que:
`A <- B` implica que **A** va tomar como valor el contenido del objeto **B**, y no al revés.

```
A <- B
A      # Ahora A toma el valor de B, y B continúa conservando el mismo valor
```

```
## [1] 2
```

```
B
```

```
## [1] 2
```

También podemos utilizar el # para realizar comentarios entre el código, en lo que resta de la línea en cuestión.

2.1.3 R base

Con *R base* nos referimos a los comandos básicos que vienen incorporados en el R, sin necesidad de cargar librerías.

2.1.3.1 Operadores lógicos:

- > (mayor a-)
- >= (mayor o igual a-)
- < (menor a-)
- <= (menor o igual a-)
- == (igual a-)
- ! = (distinto a-)

```
# Redefinimos los valores A y B
A <- 10
B <- 20

# Realizamos comparaciones lógicas
A >  B
```

```
## [1] FALSE
```

```
A >= B
```

```
## [1] FALSE
```

```
A <  B
```

```
## [1] TRUE
```

```
A <= B
```

```
## [1] TRUE
```

```
A == B
```

```
## [1] FALSE
```

```
A != B
```

```
## [1] TRUE
```

```
C <- A != B
C
```

```
## [1] TRUE
```

Como muestra el último ejemplo, el resultado de una operación lógica puede almacenarse como el valor de un objeto.

2.1.3.2 Operadores aritméticos:

```
#suma
A <- 5+6
A
```

```
## [1] 11
```

```
#Resta
B <- 6-8
B
```

```
## [1] -2
```

```
#cociente
C <- 6/2.5
C
```

```
## [1] 2.4
```

```
#multiplicacion
D <- 6*2.5
D
```

```
## [1] 15
```

2.1.3.3 Funciones:

Las funciones son series de procedimientos estandarizados, que toman como imput determinados argumentos a fijar por el usuario, y devuelven un resultado acorde a la aplicación de dichos procedimientos. Su lógica de funcionamiento es:
`funcion(argumento1 = arg1, argumento2 = arg2)`

A lo largo del curso iremos viendo numerosas funciones, según lo requieran los

distintos ejercicios. Sin embargo, veamos ahora algunos ejemplos para comprender su funcionamiento:

- paste() : concatena una serie de caracteres, indicando por última instancia como separar a cada uno de ellos

- paste0(): concatena una serie de caracteres sin separar
- sum(): suma de todos los elementos de un vector

- mean() promedio aritmético de todos los elementos de un vector

```
paste("Pega","estas",4,"palabras", sep = " ")
```

```
## [1] "Pega estas 4 palabras"
```

```
#Puedo concatenar caracteres almacenados en objetos
paste(A,B,C,sep = "**")
```

```
## [1] "11**-2**2.4"
```

```
# Paste0 pega los caracteres sin separador
paste0(A,B,C)
```

```
## [1] "11-22.4"
```

```
1:5
```

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

```
sum(1:5)
```

```
## [1] 15
```

```
mean(1:5,na.rm = TRUE)
```

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

2.1.3.4 Caracteres especiales

- R es sensible a mayúsculas y minúsculas, tanto para los nombres de las variables, como para las funciones y parámetros.
- Los **espacios en blanco** y los **carriage return** (*enter*) no son considerados por el lenguaje. Los podemos aprovechar para emprolijar el código y que la lectura sea más simple[2].

- El **numeral #** se utiliza para hacer comentarios. Todo lo que se escribe después del # no es interpretado por R. Se debe utilizar un # por cada línea de código que se desea anular

- Los **corchetes** `[]` se utilizan para acceder a un objeto:

[2] veremos que existen ciertas excepciones con algunos paquetes más adelante.

  - en un vector[n° orden]
  - en una tabla[fila, columna]
  - en una lista[n° elemento]

- el signo **$** también es un método de acceso. Particularmente, en los dataframes, nos permitira acceder a una determinada columna de una tabla
- Los **paréntesis**`()` se utilizan en las funciones para definir los parámetros.
- Las **comas** **,** se utilizan para separar los parametros al interior de una función.

2.1.4 Objetos:

Existen un gran cantidad de objetos distintos en R, en lo que resepcta al curso trabajaremos principalmente con 3 de ellos:

- Valores
- Vectores
- Data Frames
- Listas

2.1.4.1 Valores

Los valores y vectores pueden ser a su vez de distintas *clases*:

Numeric

```
A <-  1
class(A)
```

```
## [1] "numeric"
```

Character

```
A <-  paste('Soy', 'una', 'concatenación', 'de', 'caracteres', sep = " ")
A
```

```
## [1] "Soy una concatenación de caracteres"
```

```
class(A)
```

```
## [1] "character"
```

Factor

```
A <- factor("Soy un factor, con niveles fijos")
class(A)
```

```
## [1] "factor"
```

La diferencia entre un *character* y un *factor* es que el último tiene solo algunos valores permitidos (levels), con un orden interno predefinido (el cual ,por ejemplo, se respetará a la hora de realizar un gráfico)

2.1.4.2 Vectores

Para crear un **vector** utilizamos el comando `c()`, de combinar.

```
C <- c(1, 3, 4)
C
```

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

sumarle 2 a cada elemento del **vector** anterior

```
C <- C + 2
C
```

```
## [1] 3 5 6
```

sumarle 1 al primer elemento, 2 al segundo, y 3 al tercer elemento del **vector** anterior

```
D <- C + 1:3 #esto es equivalente a hacer 3+1, 5+2, 6+9
D
```

```
## [1] 4 7 9
```

`1:3` significa que queremos todos los números enteros desde 1 hasta 3.

crear un **vector** que contenga las palabras: "Carlos","Federico","Pedro"

```
E <- c("Carlos","Federico","Pedro")
E
```

```
## [1] "Carlos"   "Federico" "Pedro"
```

para acceder a algún elemento del vector, podemos buscarlo por su número de orden, entre `[ ]`

```
 E[2]
```

```
## [1] "Federico"
```

Si nos interesa almacenar dicho valor, al buscarlo lo asignamos a un nuevo objeto, dandole el nombre que deseemos

```
elemento2 <-  E[2]
```

```
elemento2
```

```
## [1] "Federico"
```

para **borrar** un objeto del ambiente de trabajo, utilizamos el comando *rm()*

```
rm(elemento2)
elemento2
```

```
## Error in eval(expr, envir, enclos): object 'elemento2' not found
```

También podemos cambiar el texto del segundo elemento de E, por el texto "Pablo"

```
E[2] <- "Pablo"
E
```

```
## [1] "Carlos" "Pablo"  "Pedro"
```

2.1.5 Data Frames

Un Data Frame es una tabla de datos, donde cada columna representa una variable, y cada fila una observación.

Este objeto suele ser central en el proceso de trabajo, y suele ser la forma en que se cargan datos externos para trabajar en el ambiente de R, y en que se exportan los resultados de nuestros trabajo.

También Se puede crear como la combinación de N vectores de igual tamaño. Por ejemplo, tomamos algunos valores del Indice de salarios

```
INDICE  <- c(100,   100,   100,
             101.8, 101.2, 100.73,
             102.9, 102.4, 103.2)

FECHA  <-  c("Oct-16", "Oct-16", "Oct-16",
             "Nov-16", "Nov-16", "Nov-16",
             "Dic-16", "Dic-16", "Dic-16")


GRUPO  <-  c("Privado_Registrado","Público","Privado_No_Registrado",
             "Privado_Registrado","Público","Privado_No_Registrado",
             "Privado_Registrado","Público","Privado_No_Registrado")


Datos <- data.frame(INDICE, FECHA, GRUPO)
Datos
```

```
##   INDICE  FECHA                 GRUPO
## 1 100.00 Oct-16    Privado_Registrado
## 2 100.00 Oct-16               Público
## 3 100.00 Oct-16 Privado_No_Registrado
## 4 101.80 Nov-16    Privado_Registrado
## 5 101.20 Nov-16               Público
## 6 100.73 Nov-16 Privado_No_Registrado
```

```
## 7 102.90 Dic-16    Privado_Registrado
## 8 102.40 Dic-16               Público
## 9 103.20 Dic-16 Privado_No_Registrado
```

Tal como en un **vector** se ubica a los elementos mediante `[ ]`, en un **dataframe** se obtienen sus elementos de la forma **`[fila, columna]`**.

Otra opción es especificar la columna, mediante el operador **`$`**, y luego seleccionar dentro de esa columna el registro deseado mediante el número de orden.

```
Datos$FECHA
```

```
## [1] Oct-16 Oct-16 Oct-16 Nov-16 Nov-16 Nov-16 Dic-16 Dic-16 Dic-16
## Levels: Dic-16 Nov-16 Oct-16
```

```
Datos[3,2]
```

```
## [1] Oct-16
## Levels: Dic-16 Nov-16 Oct-16
```

```
Datos$FECHA[3]
```

```
## [1] Oct-16
## Levels: Dic-16 Nov-16 Oct-16
```

¿que pasa si hacemos `Datos$FECHA[3,2]` ?

```
Datos$FECHA[3,2]
```

```
## Error in `[.default`(Datos$FECHA, 3, 2): incorrect number of dimensions
```

Nótese que el último comando tiene un número incorrecto de dimensiones, porque estamos refiriendonos 2 veces a la columna FECHA.

Acorde a lo visto anteriormente, el acceso a los **dataframes** mediante `[ ]`, puede utilizarse para realizar filtros sobre la base, especificando una condición para las filas. Por ejemplo, puedo utilizar los `[ ]` para conservar del **dataframe** `Datos` unicamente los registros con fecha de Diciembre 2016:

```
Datos[Datos$FECHA=="Dic-16",]
```

```
##   INDICE  FECHA                 GRUPO
## 7  102.9 Dic-16    Privado_Registrado
## 8  102.4 Dic-16               Público
## 9  103.2 Dic-16 Privado_No_Registrado
```

La lógica del paso anterior sería: Accedo al dataframe `Datos`, pidiendo únicamente conservar las filas (por eso la condición se ubica a la *izquierda* de la `,`) que cumplan el requisito de pertenecer a la categoría **"Dic-16"** de la variable **FECHA**.
Aún más, podría aplicar el filtro y al mismo tiempo identificar una variable de interés para luego realizar un cálculo sobre aquella. Por ejemplo, podría calcular la media de los indices en el mes de Diciembre.

```
###Por separado
Indices_Dic <- Datos$INDICE[Datos$FECHA=="Dic-16"]
Indices_Dic
```

```
## [1] 102.9 102.4 103.2
```

```
mean(Indices_Dic)
```

```
## [1] 102.8333
```

```
### Todo junto
mean(Datos$INDICE[Datos$FECHA=="Dic-16"])
```

```
## [1] 102.8333
```

La lógica de esta sintaxis sería: "Me quedó con la variable **INDICE**, cuando la variable FECHA sea igual a **"Dic-16"**, luego calculo la media de dichos valores"

2.1.6 Listas

Contienen una concatenación de objetos de cualquier tipo. Así como un vector contiene valores, un dataframe contiene vectores, una lista puede contener dataframes, pero también vectores, o valores, y *todo ello a la vez*

```
superlista <- list(A,B,C,D,E,FECHA, DF = Datos, INDICE, GRUPO)
superlista
```

```
## [[1]]
## [1] Soy un factor, con niveles fijos
## Levels: Soy un factor, con niveles fijos
##
## [[2]]
## [1] -2
##
## [[3]]
## [1] 3 5 6
##
## [[4]]
## [1] 4 7 9
##
## [[5]]
## [1] "Carlos" "Pablo"  "Pedro"
##
## [[6]]
## [1] "Oct-16" "Oct-16" "Oct-16" "Nov-16" "Nov-16" "Nov-16" "Dic-16" "Dic-16"
## [9] "Dic-16"
##
## $DF
```

```
##   INDICE  FECHA                 GRUPO
## 1 100.00 Oct-16    Privado_Registrado
## 2 100.00 Oct-16               Público
## 3 100.00 Oct-16 Privado_No_Registrado
## 4 101.80 Nov-16    Privado_Registrado
## 5 101.20 Nov-16               Público
## 6 100.73 Nov-16 Privado_No_Registrado
## 7 102.90 Dic-16    Privado_Registrado
## 8 102.40 Dic-16               Público
## 9 103.20 Dic-16 Privado_No_Registrado
##
## [[8]]
## [1] 100.00 100.00 100.00 101.80 101.20 100.73 102.90 102.40 103.20
##
## [[9]]
## [1] "Privado_Registrado"    "Público"               "Privado_No_Registrado"
## [4] "Privado_Registrado"    "Público"               "Privado_No_Registrado"
## [7] "Privado_Registrado"    "Público"               "Privado_No_Registrado"
```

Para acceder un elemento de una lista, podemos utilizar el operador **$**, que se puede usar a su vez de forma iterativa

```
superlista$DF$FECHA[2]
```

```
## [1] Oct-16
## Levels: Dic-16 Nov-16 Oct-16
```

2.1.7 Ambientes de trabajo

Hay algunas cosas que tenemos que tener en cuenta respecto del orden del ambiente en el que trabajamos:

- Working Directory: El directorio de trabajo, pueden ver el suyo con `getwd()`, es *hacia donde apunta el código*, por ejemplo, si quieren leer un archivo, la ruta del archivo tiene que estar explicitada como el recorrido desde el Working Directory.
- Environment: Esto engloba tanto la información que tenemos cargada en *Data* y *Values*, como las librerías que tenemos cargadas mientras trabajamos.

Es importante que mantengamos bien delimitadas estas cosas entre diferentes trabajos, sino:

1. El directorio queda referido a un lugar específico en nuestra computadora.

- Si se lo compartimos a otro **se rompe**
- Si cambiamos de computadora **se rompe**
- Si lo cambiamos de lugar **se rompe**
- Si primero abrimos otro script **se rompe**

2. Tenemos mezclados resultados de diferentes trabajos:

- Nunca sabemos si esa variable/tabla/lista se creo en ese script y no otro
- Perdemos espacio de la memoria
- No estamos seguros de que el script cargue todas las librerías que necesita

Rstudio tiene una herramienta muy útil de trabajo que son los **proyectos**. Estos permiten mantener un ambiente de trabajo delimitado por cada uno de nuestros trabajos. Es decir:

- El directorio de trabajo se refiere a donde esta ubicado el archivo .Rproj
- El Environment es específico de nuestro proyecto.

Un proyecto no es un sólo script, sino toda una carpeta de trabajo.

Para crearlo, vamos al logo de nuevo projecto (Arriba a la izquierda de la panatalla), y elegimos la carpeta de trabajo.

2.1.8 Tipos de archivos de R

- **Script**: Es un archivo de texto plano, donde podemos poner el código que utilizamos para preservarlo
- **Rnotebook**: También sirve para guardar el código, pero a diferencia de los scripts, se puede compilar, e intercalar código con resultados (este archivo es un rnotebook)
- **Rproject**: Es un archivo que define la metadata del proyecto
- **RDS y Rdata**: Dos formatos de archivos propios de R para guardar datos.

2.2 Práctica Guiada

2.2.1 Instalación de paquetes complementarios al R Base

Hasta aquí hemos visto múltiples funciones que están contenidas dentro del lenguaje básico de R. Ahora bien, al tratarse de un software libre, los usuarios de R con más experiencia contribuyen sistemáticamente a expandir este lenguaje mediante la creación y actualización de **paquetes** complementarios. Lógicamente, los mismos no están incluidos en la instalación inicial del programa, pero podemos descargarlos e instalarlos al mismo tiempo con el siguiente comando:

```
install.packages("nombre_del_paquete")
```

Resulta recomendable **ejecutar este comando desde la consola** ya que solo necesitaremos correrlo una vez en nuestra computadora. Al ejecutar el mismo, se descargarán de la pagina de CRAN los archivos correspondientes al paquete hacia el directorio en donde hayamos instalado el programa. Típicamente los archivos se encontrarán en `C:\Program Files\R\R-3.5.0\library\`, siempre con la versión del programa correspondiente.
Una vez instalado el paquete, cada vez que abramos una nueva sesión de R

Figure 2.5: logo Rpoject

y querramos utilizar el mismo debemos **cargarlo al ambiente de trabajo** mediante la siguiente función:

```
library(nombre_del_paquete)
```

Nótese que al cargar/activar el paquete no son necesarias las comillas.

2.2.2 Lectura y escritura de archivos

2.2.2.1 .csv y .txt

Hay **muchas** funciones para leer archivos de tipo *.txt* y *.csv*. La mayoría sólo cambia los parámetros que vienen por default.

Es importante tener en cuenta que una base de datos que proviene de archivos *.txt*, o *.csv* puede presentar diferencias en cuanto a los siguientes parametros:

- encabezado
- delimitador (`,`, tab, `;`)
- separador decimal

```
dataframe <- read.delim(file, header = TRUE, sep = "\t", quote = "\"", dec = ".", fill
```

Ejemplo. Levantar la base de sueldos de funcionarios

En el parametro `file` tengo que especificar el nombre completo del archivo, incluyendo el directorio donde se encuentra. Lo más sencillo es abrir comillas, apretar `Tab` y se despliega el menú de las cosas que tenemos en el directorio de trabajo. Si queremos movernos hacia arriba, agregamos `../`

```
sueldos_funcionarios <- read.table(file = '../fuentes/sueldo_funcionarios_2019.csv',se
sueldos_funcionarios[1:10,]
```

```
##             cuil anio mes funcionario_apellido funcionario_nombre
## 1  20-17692128-6 2019   1    RODRIGUEZ LARRETA    HORACIO ANTONIO
## 2  20-17735449-0 2019   1              SANTILLI        DIEGO CESAR
## 3  27-24483014-0 2019   1                 ACUÑA      MARIA SOLEDAD
## 4  20-13872301-2 2019   1              ASTARLOA      GABRIEL MARIA
## 5  20-25641207-2 2019   1              AVOGADRO       ENRIQUE LUIS
## 6  27-13221055-7 2019   1             BOU PEREZ          ANA MARIA
## 7  27-13092400-5 2019   1                 FREDA     MONICA BEATRIZ
## 8  20-17110752-1 2019   1          MACCHIAVELLI    EDUARDO ALBERTO
## 9  20-22293873-3 2019   1                MIGUEL       FELIPE OSCAR
## 10 20-14699669-9 2019   1                MOCCIA             FRANCO
##                                       repartición asignacion_por_cargo_i
## 1                                 Jefe de Gobierno               197745.8
## 2                           Vicejefatura de Gobierno               197745.8
## 3               Ministerio de Educación e Innovación               224516.6
## 4  Procuración General de la Ciudad de Buenos Aires               224516.6
## 5                              Ministerio de Cultura               224516.6
```

```
## 6                              Ministerio de Salud              224516.6
## 7  Sindicatura General de la Ciudad de Buenos Aires              224516.6
## 8          Ministerio de Ambiente y Espacio Público              224516.6
## 9                 Jefatura de Gabinete de Ministros              224516.6
## 10     Ministerio de Desarrollo Urbano y Transporte              224516.6
##    aguinaldo_ii total_salario_bruto_i_._ii observaciones
## 1             0                   197745.8
## 2             0                   197745.8
## 3             0                   224516.6
## 4             0                   224516.6
## 5             0                   224516.6
## 6             0                   224516.6
## 7             0                   224516.6
## 8             0                   224516.6
## 9             0                   224516.6
## 10            0                   224516.6
```

Como puede observarse aquí, las bases individuales de la EPH cuentan con más de 58.000 registros y 177 variables. Al trabajar con bases de microdatos, resulta conveniente contar con algunos comandos para tener una mirada rápida de la base, antes de comenzar a realizar los procesamientos que deseemos.

Veamos algunos de ellos:

```
#View(individual_t117)
names(sueldos_funcionarios)
```

```
##  [1] "cuil"                      "anio"
##  [3] "mes"                       "funcionario_apellido"
##  [5] "funcionario_nombre"        "repartición"
##  [7] "asignacion_por_cargo_i"    "aguinaldo_ii"
##  [9] "total_salario_bruto_i_._ii" "observaciones"
```

```
summary(sueldos_funcionarios)
```

```
##             cuil         anio           mes       funcionario_apellido
##  20-13872301-2: 3   Min.   :2019   Min.   :1.00   ACUÑA     : 3
##  20-14699669-9: 3   1st Qu.:2019   1st Qu.:2.00   ASTARLOA  : 3
##  20-16891528-5: 3   Median :2019   Median :3.00   AVELLANEDA: 3
##  20-16891539-0: 3   Mean   :2019   Mean   :3.34   AVOGADRO  : 3
##  20-17110752-1: 3   3rd Qu.:2019   3rd Qu.:5.00   BENEGAS   : 3
##  20-17692128-6: 3   Max.   :2019   Max.   :6.00   BOU PEREZ : 3
##  (Other)      :76                                 (Other)   :76
##        funcionario_nombre
##   ANA MARIA      : 3
##   BRUNO GUIDO    : 3
##   CHRISTIAN      : 3
##   DIEGO CESAR    : 3
```

```
##   DIEGO HERNAN   : 3
##   EDUARDO ALBERTO: 3
##  (Other)         :76
##                                                          repartición
##  Consejo de los Derechos de Niñas, Niños y Adoles - Presidencia: 3
##  Ente de Turismo Ley Nº 2627                                    : 3
##  Jefatura de Gabinete de Ministros                              : 3
##  Jefe de Gobierno                                               : 3
##  Ministerio de Ambiente y Espacio Público                       : 3
##  Ministerio de Cultura                                          : 3
##  (Other)                                                        :76
##  asignacion_por_cargo_i  aguinaldo_ii    total_salario_bruto_i_._ii
##  Min.   :197746         Min.   :     0   Min.   :197746
##  1st Qu.:217520         1st Qu.:     0   1st Qu.:217805
##  Median :226866         Median :     0   Median :226866
##  Mean   :224718         Mean   : 14843   Mean   :239560
##  3rd Qu.:231168         3rd Qu.:     0   3rd Qu.:248033
##  Max.   :249662         Max.   :113433   Max.   :340300
##
##         observaciones
##                 :93
##  baja 28/2/2019: 1
##
##
##
##
##
```

```
head(sueldos_funcionarios)[,1:5]
```

```
##            cuil anio mes funcionario_apellido funcionario_nombre
## 1 20-17692128-6 2019   1    RODRIGUEZ LARRETA    HORACIO ANTONIO
## 2 20-17735449-0 2019   1             SANTILLI        DIEGO CESAR
## 3 27-24483014-0 2019   1                ACUÑA      MARIA SOLEDAD
## 4 20-13872301-2 2019   1             ASTARLOA      GABRIEL MARIA
## 5 20-25641207-2 2019   1             AVOGADRO       ENRIQUE LUIS
## 6 27-13221055-7 2019   1            BOU PEREZ          ANA MARIA
```

2.2.2.2 Excel

Para leer y escribir archivos excel debemos utilizar los comandos que vienen con la librería openxlsx

```
# install.packages("openxlsx") # por única vez
library(openxlsx) #activamos la librería

#creamos una tabla cualquiera de prueba
```

```
x <- 1:10
y <- 11:20
tabla_de_R <- data.frame(x,y)

# escribimos el archivo
write.xlsx( x = tabla_de_R, file = "../resultados/archivo.xlsx",row.names = FALSE)
#Donde lo guardó? Hay un directorio por default en caso de que no hayamos definido alguno.

#getwd()

#Si queremos exportar multiples dataframes a un Excel, debemos armar previamente una lista de el
Lista_a_exportar <- list("sueldos funcionarios" = sueldos_funcionarios,
                         "Tabla Numeros" = tabla_de_R)

write.xlsx( x = Lista_a_exportar, file = "../resultados/archivo_2_hojas.xlsx",row.names = FALSE)

#leemos el archivo especificando la ruta (o el directorio por default) y el nombre de la hoja que
Indices_Salario <- read.xlsx(xlsxFile = "../resultados/archivo_2_hojas.xlsx",sheet = "sueldos fun
#alternativamente podemos especificar el número de orden de la hoja que deseamos levantar
Indices_Salario <- read.xlsx(xlsxFile = "../resultados/archivo_2_hojas.xlsx",sheet = 1)
Indices_Salario[1:10,]
```

```
##             cuil anio mes funcionario_apellido funcionario_nombre
## 1  20-17692128-6 2019   1    RODRIGUEZ LARRETA    HORACIO ANTONIO
## 2  20-17735449-0 2019   1             SANTILLI        DIEGO CESAR
## 3  27-24483014-0 2019   1                ACUÑA      MARIA SOLEDAD
## 4  20-13872301-2 2019   1             ASTARLOA      GABRIEL MARIA
## 5  20-25641207-2 2019   1             AVOGADRO       ENRIQUE LUIS
## 6  27-13221055-7 2019   1            BOU PEREZ          ANA MARIA
## 7  27-13092400-5 2019   1                FREDA     MONICA BEATRIZ
## 8  20-17110752-1 2019   1         MACCHIAVELLI    EDUARDO ALBERTO
## 9  20-22293873-3 2019   1               MIGUEL       FELIPE OSCAR
## 10 20-14699669-9 2019   1               MOCCIA             FRANCO
##                                      repartición asignacion_por_cargo_i
## 1                              Jefe de Gobierno               197745.8
## 2                         Vicejefatura de Gobierno               197745.8
## 3             Ministerio de Educación e Innovación               224516.6
## 4  Procuración General de la Ciudad de Buenos Aires               224516.6
## 5                           Ministerio de Cultura               224516.6
## 6                             Ministerio de Salud               224516.6
## 7  Sindicatura General de la Ciudad de Buenos Aires               224516.6
## 8          Ministerio de Ambiente y Espacio Público               224516.6
## 9                 Jefatura de Gabinete de Ministros               224516.6
## 10     Ministerio de Desarrollo Urbano y Transporte               224516.6
##    aguinaldo_ii total_salario_bruto_i_._ii observaciones
```

```
## 1           0               197745.8
## 2           0               197745.8
## 3           0               224516.6
## 4           0               224516.6
## 5           0               224516.6
## 6           0               224516.6
## 7           0               224516.6
## 8           0               224516.6
## 9           0               224516.6
## 10          0               224516.6
```

Chapter 3

Probabilidad y Estadística

Esta clase es un repaso de los rudimentos de probabilidad y estadística. El objetivo es obtener las herramientas básicas para la interpretación de resultados estadísticos.

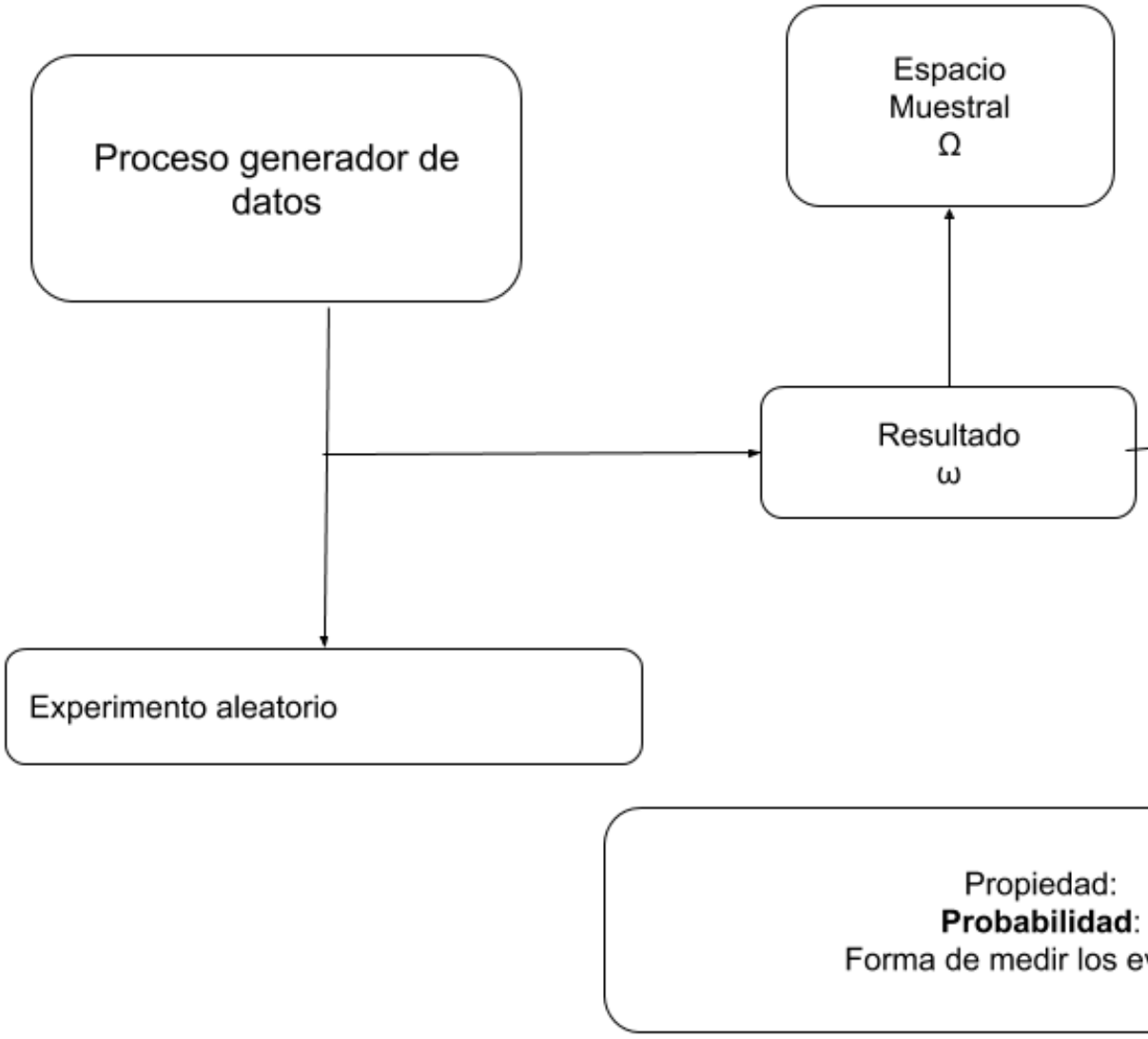
- Introducción a probabilidad
- Introducción a distribuciones
- El problema de la inversión
- Estadística
- Población y muestra
- Estimadores puntuales, tests de hipótesis
- Boxplots, histogramas y kernels

3.1 Explicación

3.1.1 Probabilidad

Previo a estudiar las herramientas de la estadística descriptiva, es necesario hacer un breve resumen de algunos conceptos fundamentales de probabilidad

3.1.1.1 Marco conceptual



- El análisis de las probabilidades parte de un **proceso generador de datos** entendido como cualquier fenómeno que produce algún tipo de información de forma sistemática.
- Cada iteración de este proceso produce información, que podemos interpretar como un **resultado**.
- Existe un conjunto de posibles resultados, que definimos como **espacio muestral**.
- Un **evento** es el conjunto de resultados ocurridos.

- En este marco, la **probabilidad** es un atributo de los eventos. Es la forma de medir los eventos tal que, siguiendo la definición moderna de probabilidad:

A) $P(A) \geq 0 \forall\ A \subseteq \Omega$
B) $P(\Omega) = 1$
C) $P(A \cup B) = P(A) + P(B)\ si\ A \cap B = \emptyset$

ejemplo, tiramos un dado y sale tres

- Espacio muestral: 1,2,3,4,5,6
- Resultado: 3
- Evento: impar (el conjunto 1,3,5)

3.1.1.2 Distribución de probabilidad

- La distribución de probabilidad hace referencia a los posibles valores teóricos de cada uno de los resultados pertenecientes al espacio muestral.
- Existen dos tipos de distribuciones, dependiendo si el espacio muestral es o no numerable.

3.1.1.2.1 Distribuciones discretas

Sigamos con el ejemplo de dado.

Podríamos definir la distribución de probabilidad, si no esta cargado, cómo:

```
## # A tibble: 6 x 2
##   valor probabilidad
##   <int> <chr>
## 1     1 1/6
## 2     2 1/6
## 3     3 1/6
## 4     4 1/6
## 5     5 1/6
## 6     6 1/6
```

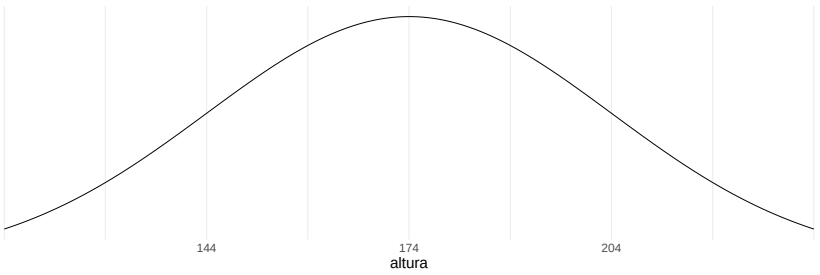
Cómo el conjunto de resultados posibles es acotado, podemos definirlo en una tabla, esta es una distribución *discreta*

3.1.1.2.2 Distribuciones continuas

¿Qué pasa cuando el conjunto de resultados posibles es tan grande que no se puede enumerar la probabilidad de cada caso?

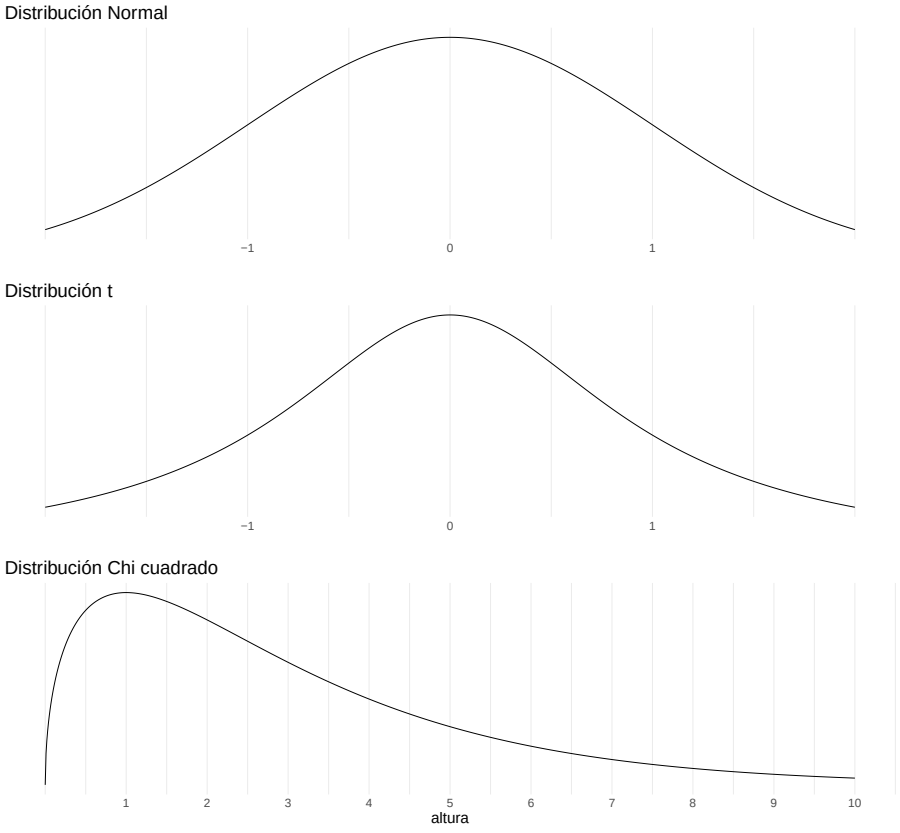
Si, por definición o por practicidad, no se puede enumerar cada caso, lo que tenemos es una **distribución continua**

Por ejemplo, la altura de la población



- En este caso, no podemos definir en una tabla la probabilidad de cada uno de los posibles valores. *de hecho, la probabilidad puntual es 0.*
- Sin embargo, sí podemos definir una *función de probabilidad*, la *densidad.*
- Según qué función utilicemos, cambiara la forma de la curva.

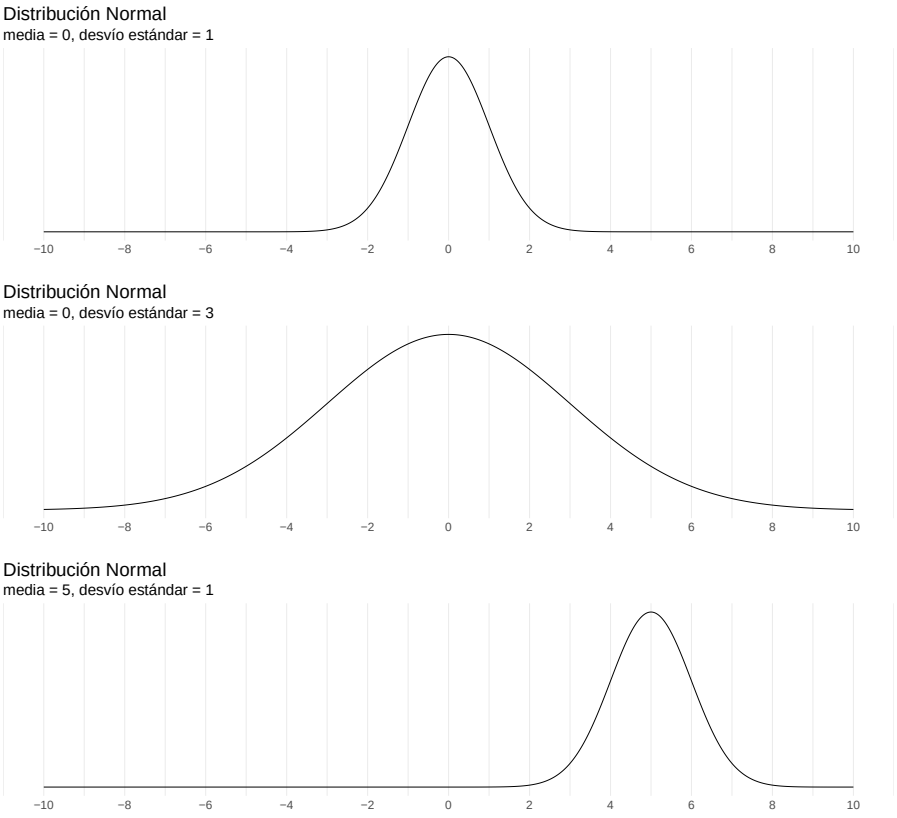
Por ejemplo:



Una distribución de probabilidad se **caracteriza** por sus *parámetros.*

- Por ejemplo, la distribución normal se caracteriza por su *esperanza* y su

varianza (o desvío estándar)

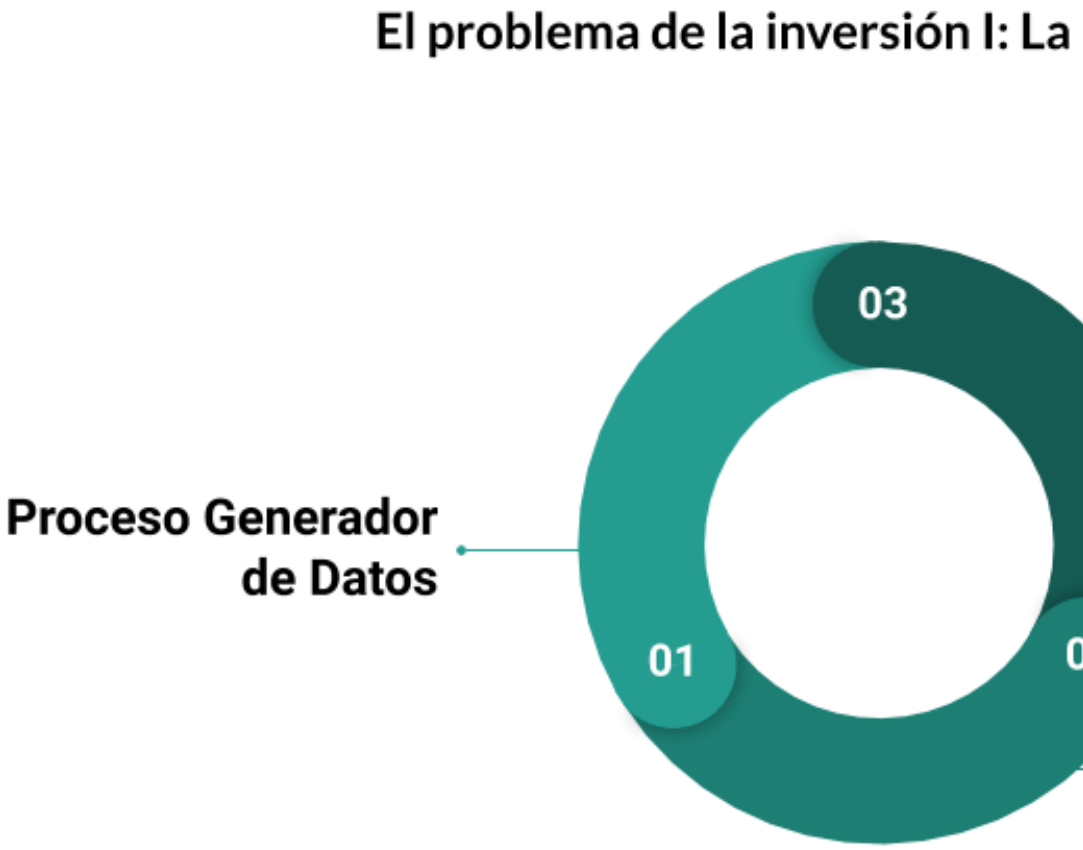


3.1.2 Estadística

3.1.2.1 El problema de la inversión

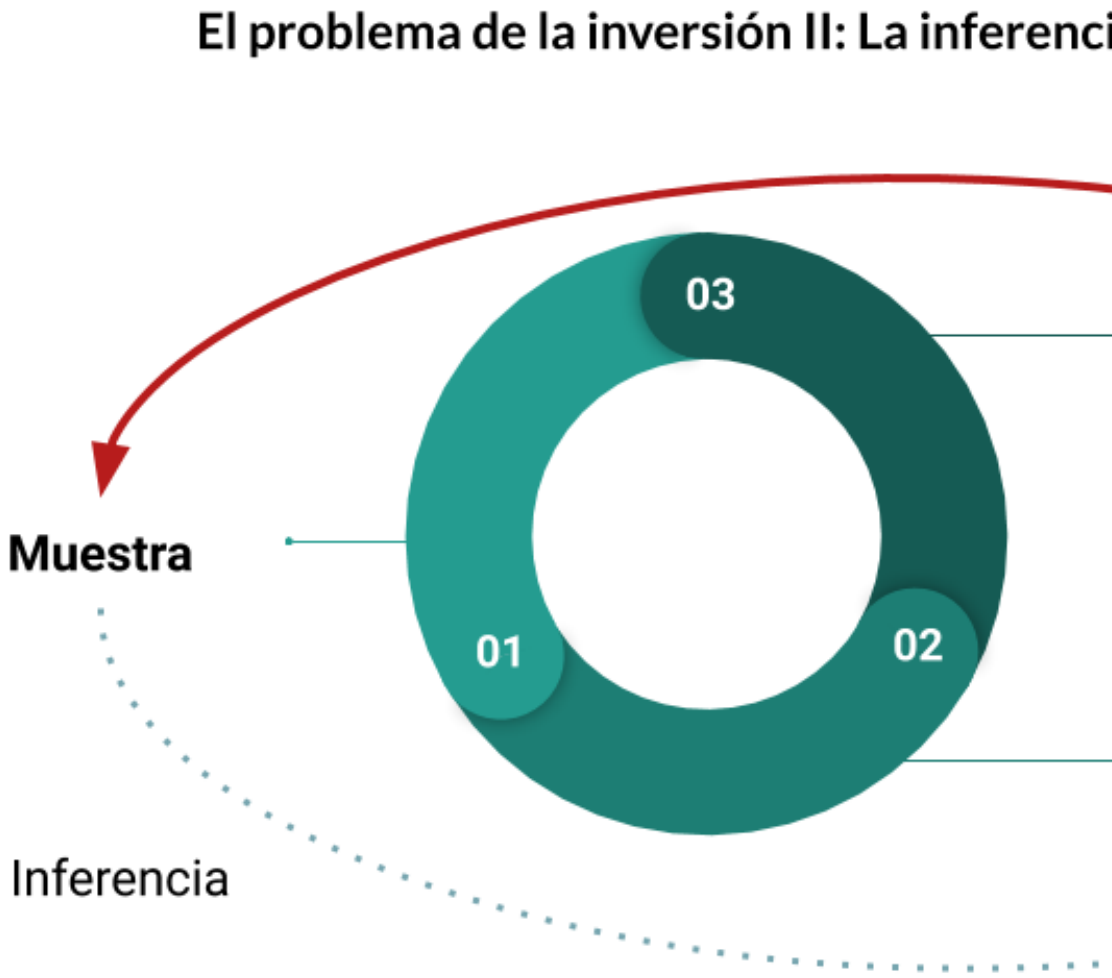
El problema de la probabilidad se podría pensar de la siguiente forma:

1. Vamos a partir de un **proceso generador de datos**
2. para calcular su **distribución de probabilidad**, los **parámetros** que caracterizan a ésta, y a partir de allí,
3. calcular la probabilidad de que, al tomar una **muestra**, tenga ciertos eventos.



El problema de la estadística es exactamente el contrario:

1. Partimos de una **muestra** para
2. inferir cuál es la **distribución de probabilidad**, y los **parámetros** que la caracterizan
3. para finalmente poder sacar conclusiones sobre el **proceso generador de datos**



3.1.2.1.1 Población y muestra

En este punto podemos hacer la distinción entre **población** y **muestra**

- **Población**: El universo en estudio. Puede ser:
  - finita: Los votantes en una elección.
  - infinita: El lanzamiento de una moneda.
- **Muestra**: subconjunto de n observaciones de una población.

Solemos utilizar las mayúsculas (N) para la población y las minúsculas (n) para

las muestras

3.1.2.1.2 Parámetros y Estimadores

- Como dijimos, los **parámetros** describen a la función de probabilidad. Por lo tanto hacen referencia a los atributos de la **población**. Podemos suponer que son *constantes*
- Un **estimador** es un estadístico (esto es, una función de la muestra) usado para estimar un parámetro desconocido de la población.

3.1.2.1.3 Ejemplo. La media

Esperanza o Media Poblacional:

$$\mu = E(x) = \sum_{i=1}^{N} x_i p(x_i)$$

Media muestral:

$$\bar{X} = \sum_{i=1}^{n} \frac{Xi}{n}$$

Como no puedo conocer μ, lo estimo mediante $\bar{X}$

3.1.2.2 Estimación puntual, Intervalos de confianza y Tests de hipótesis

- El estimador $\bar{X}$ nos devuelve un número. Esto es una inferencia de cuál creemos que es la media. Pero no es seguro de que esa sea realmente la media. Esto es lo que denominamos estimación puntual
- También podemos estimar un intervalo, dentro del cual consideramos que se encuentra la media poblacional. La ventaja de esta metodología es que podemos definir la probabilidad de que el parámetro poblacional realmente este dentro de este intervalo. Esto se conoce como **intervalos de confianza**
- Por su parte, también podemos calcular la probabilidad de que el parámetro poblacional sea mayor, menor o igual a un cierto valor. Esto es lo que se conoce como **test de hipótesis**.
- En el fondo, los intervalos de confianza y los tests de hipótesis se construyen de igual manera. Son funciones que se construyen a partir de los datos, que se comparan con distribuciones conocidas, *teóricas*.

3.1.2.2.1 Definición de los tests

- Los tests se construyen con dos hipótesis: La hipótesis nula H_0, y la hipótesis alternativa, H_1. Lo que buscamos es ver si *hay evidencia suficiente para rechazar la hipótesis nula.*

Por ejemplo, si queremos comprobar si la media poblacional, μ de una distribución es mayor a X_i, haremos un test con las siguientes hipótesis:

- $H_0 : \mu = X_i$
- $H_1 : \mu > X_i$

Si la evidencia es lo suficientemente fuerte, podremos rechazar la hipótesis H_0, *pero no afirmar la hipótesis* H_1

3.1.2.2.2 Significatividad en los tests

- Muchas veces decimos que algo es **"estadística mente significativo"**. Detrás de esto se encuentra un test de hipótesis que indica que hay una suficiente *significativdad estadística.*

- La *significatividad estadística*,representada con α, es la probabilidad de rechazar H_0 cuando en realidad es cierta. Por eso, cuanto más bajo el valor de α, más seguros estamos de no equivocarnos. Por lo general testeamos con valores de alpha de 1%, 5% y 10%, dependiendo del área de estudio

- El **p-valor** es _la mínima significatividad para la que rechazo el test. Es decir, cuanto más bajo es el p-valor, más seguros estamos de rechazar H_0

- El resultado de un test esta determinado por

  1. **La fuerza evidencia empírica**: Si nuestra duda es si la media poblacional es mayor a, digamos, 10. Y la media muestral es 11, no es es lo mismo que si es 100, 1000 o 10000.
  2. **El tamaño de la muestra**: En las fórmulas que definen los test siempre juega el tamaño de la muestra: cuanto más grande es, más seguros estamos de que el resultado no es producto del mero azar.
  3. **La veracidad de los supuestos**: Otra cosa importante es que los test asumen ciertas cosas:

  – Normalidad en los datos.
  – Que conocemos algún otro parámetro de la distribución, como la varianza.
  – Que los datos son independientes entre sí,
  – Etc.
  Cada Test tiene sus propios supuestos. Por eso a veces luego de hacer un test, hay que hacer otros tests para validar que los supuestos se cumplen.

- Lo primero, la fuerza de la evidencia, es lo que más nos importa, y no hay mucho por hacer.

- El tamaño de la muestra es un problema, porque si la muestra es muy chica, entonces podemos no llegar a conclusiones significativas aunque sí ocurra aquello que queríamos probar.

- Sin embargo, el verdadero problema en *La era del big data* es que tenemos muestras demasiado grandes, por lo que cualquier test, por más mínima que sea la diferencia, puede dar significativo.

  Por ejemplo, podemos decir que la altura promedio en Argentina es 1,74. Pero si hacemos un test, utilizando como muestra 40 millones de personas, vamos a rechazar que ese es el valor, porque en realidad es 1,74010010. En términos de lo que nos puede interesar, 1,74 sería válido, pero estadísticamente rechazaríamos.

- Finalmente, según la información que tengamos de la población y cual es el problema que queremos resolver, vamos a tener que utilizar distintos tipos de tests. La cantidad de tests posibles es ENORME, y escapa al contenido de este curso, así como sus fórmulas. A modo de ejemplo, les dejamos el siguiente machete:

3.1.3 Algunos estimadores importantes

3.1.3.1 Medidas de centralidad

- **Media**

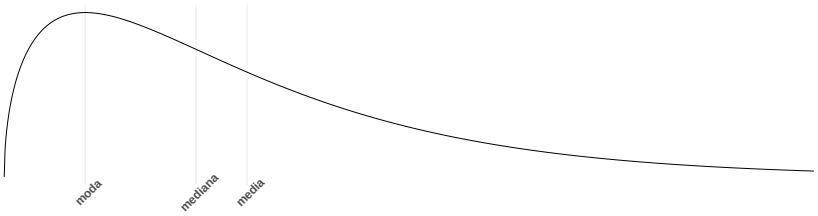
$$\bar{X} = \sum_{i=1}^{n} \frac{Xi}{n}$$

- **Mediana**:

Es el valor que parte la distribución a la mitad

- **Moda**

La moda es el valor más frecuente de la distribución



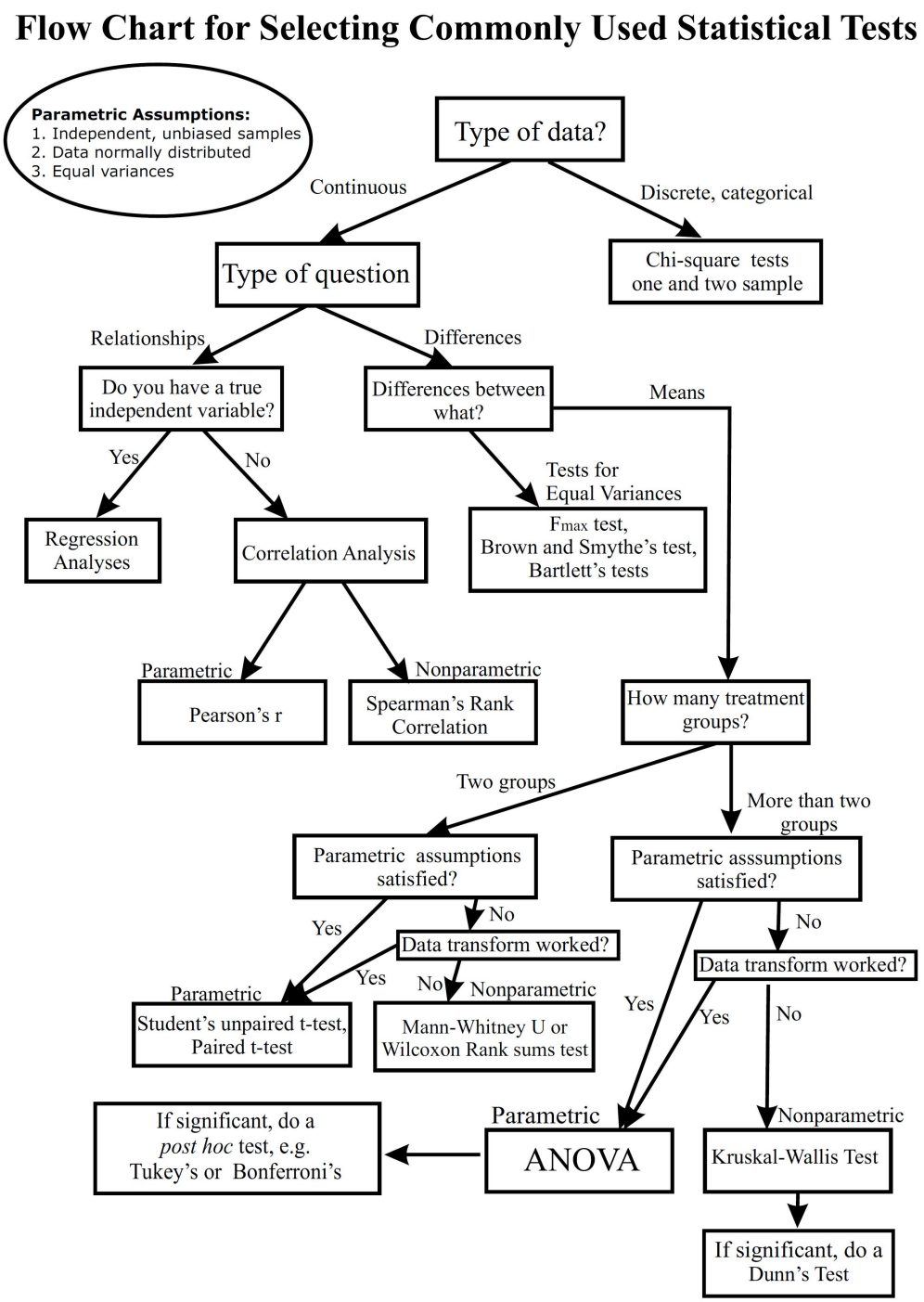
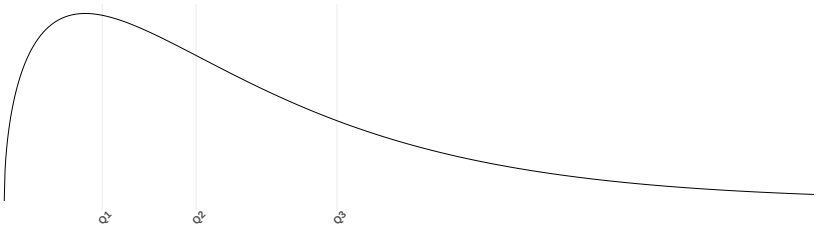


Figure 3.1: fuente: http://abacus.bates.edu/~ganderso/biology/resources/statistics.html

3.1.3.2 Cuantiles

Así como dijimos que la mediana es el valor que deja al 50% de los datos de un lado y al 50% del otro, podemos generalizar este concepto a cualquier X%. Esto son los cuantiles. El cuantil x, es el valor tal que queda un x% de la distribución a izquierda, y 1-x a derecha.

Algunos de los más utilizados son el del 25%, también conocido como Q_1 (el *cuartil* 1), el Q_2 (la mediana) y el Q_3 (el *cuartil* 3), que deja el 75% de los datos a su derecha. Veamos como se ven en la distribución de arriba



3.1.3.3 desvío estándar

- El *desvío estándar* es una medida de dispersión de los datos, que indica cuánto se suelen alejar de la media.

3.1.4 Gráficos estadísticos

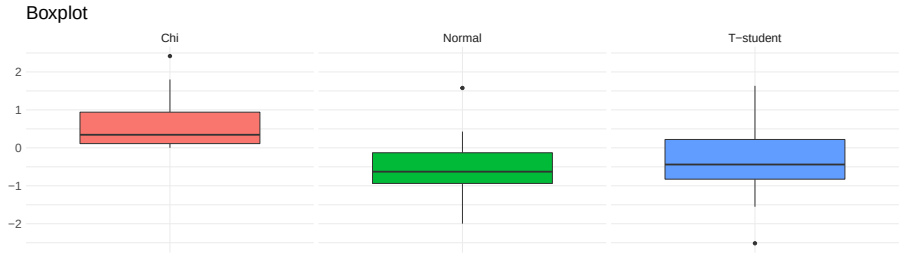
Cerramos la explicación con algunos gráficos que resultan útiles para entender las propiedades estadísticas de los datos.

3.1.4.1 Boxplot

El Boxplot es muy útil para describir una distribución y para detectar outliers. Reúne los principales valores que caracterizan a una distribución:

- Q_1
- Q_2 (la mediana)
- Q_3
- el *rango intercuarítlico* $Q_3 - Q_1$, que define el centro de la distribución
- Outliers, definidos como aquellos puntos que se encuentran a más de 1,5 veces el rango intercuartílico del centro de la distribución.

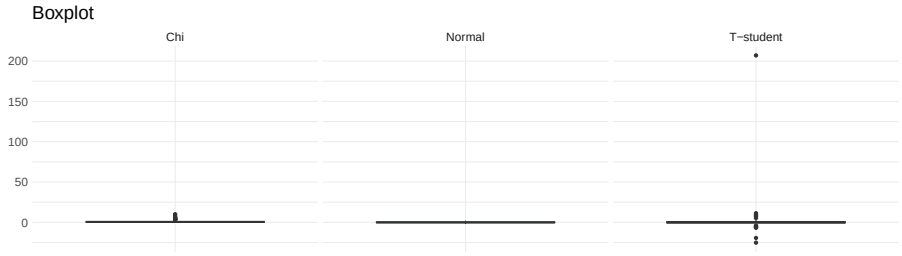
veamos qué pinta tienen los boxplot de números generados aleatoriamente a partir de tres distribuciones que ya vimos. En este caso, sólo tomaremos 15 valores de cada distribución



Algunas cosas que resaltan:

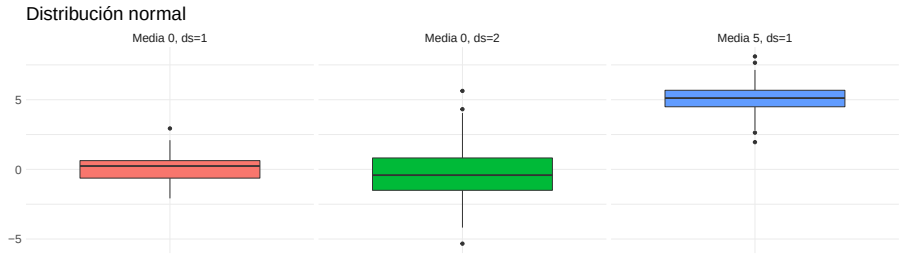
- la distribución χ^2 no toma valores en los negativos.
- La normal esta más concentrada en el centro de la distribución

Podemos generar 100 números aleatorios en lugar de 15:



Cuando generamos 100 valores en lugar de 15, tenemos más chances de agarrar un punto alejado en la distribución. De esta forma podemos apreciar las diferencias entre la distribución normal y la T-student.

También podemos volver a repasar qué efecto generan los distintos parámetros. Por ejemplo



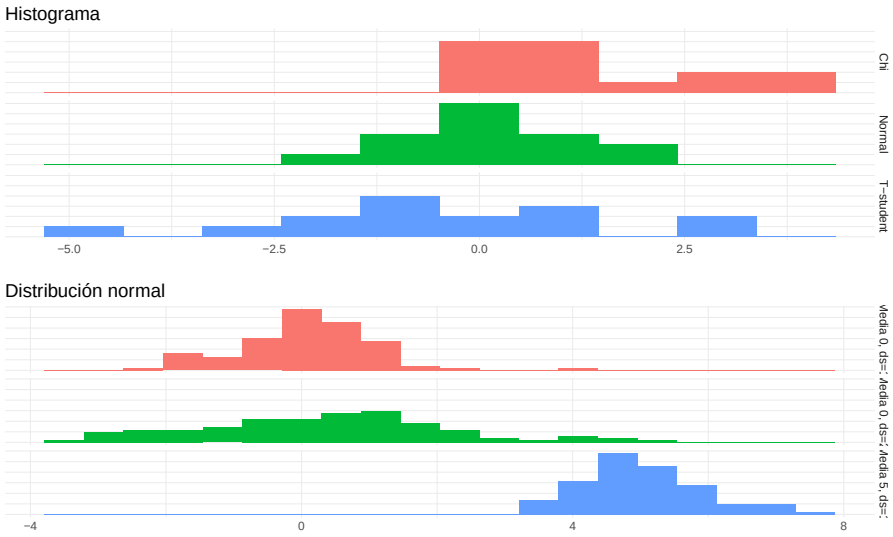
3.1.4.2 Histograma

Otra forma de analizar una distribución es mediante los histogramas:

- En un histograma agrupamos las observaciones en rangos fijos de la variable y contamos la cantidad de ocurrencias.

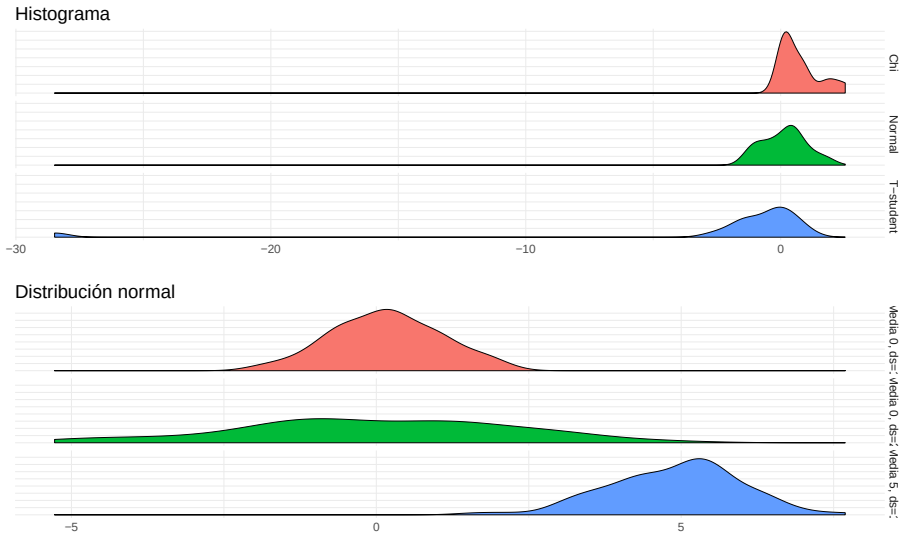
- Cuanto más alta es una barra, es porque más observaciones se encuentran en dicho rango

Veamos el mismo ejemplo que arriba, pero con histogramas



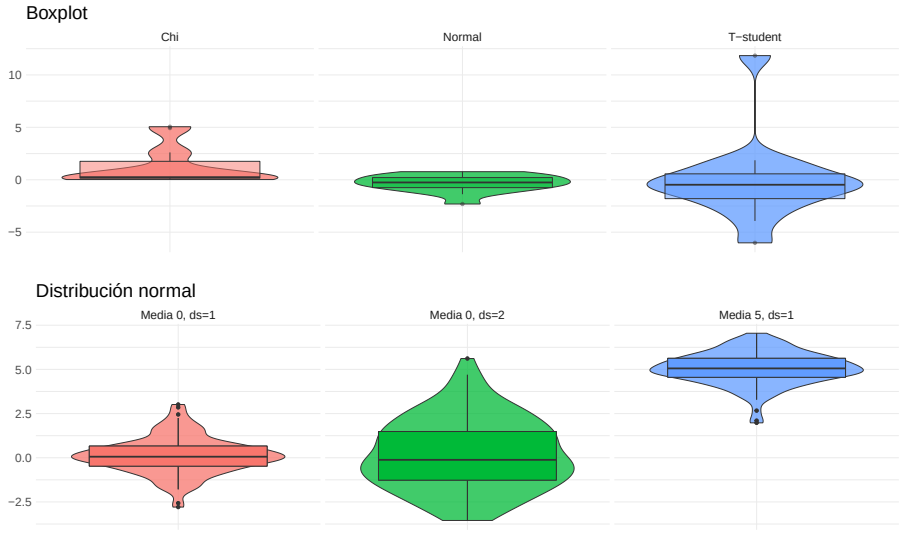
3.1.4.3 Kernel

Los Kernels son simplemente un suavizados sobre los histogramas



3.1.4.4 Violin plots

Combinando la idea de Kernels y boxplots, se crearon los violin plots, que simplemente muestran a los kernels duplicados



3.1.5 Bibliografía de consulta

Quién quiera profundizar en estos temas, puede ver los siguientes materiales:

- https://seeing-theory.brown.edu/
- https://lagunita.stanford.edu/courses/course-v1:OLI+ProbStat+Open_Jan2017/about
- Jay L. Devore, "Probabilidad y Estadística para Ingeniería y Ciencias", International Thomson Editores. https://inferencialitm.files.wordpress.com/2018/04/probabilidad-y-estadistica-para-ingenieria-y-ciencias-devore-7th.pdf

3.2 Práctica Guiada

```
library(tidyverse)
```

3.2.1 Generación de datos aleatorios

Para generar datos aleatorios, usamos las funciones

`-rnorm` para generar datos que surgen de una distribución normal `-rt` para generar datos que surgen de una distribución T-student `-rchisq` para generar datos que surgen de una distribución Chi cuadrado

pero antes, tenemos que fijar la *semilla* para que los datos sean reproducibles

```
set.seed(1234)
rnorm(n = 15,mean = 0, sd = 1 )
```

```
##  [1] -1.20706575  0.27742924  1.08444118 -2.34569770  0.42912469
##  [6]  0.50605589 -0.57473996 -0.54663186 -0.56445200 -0.89003783
## [11] -0.47719270 -0.99838644 -0.77625389  0.06445882  0.95949406
```

```
rt(n = 15,df=1 )
```

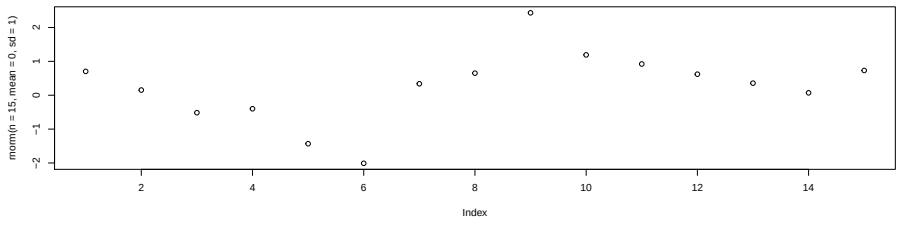
```
##  [1] -0.363717710 -1.603466805 -0.388596796 -0.588007490  0.007839245
##  [6] 14.690527710 -1.863488555  0.022667470 -2.084247299 -0.249237745
## [11] -1.311594174 -3.569055208 -2.490838240 -3.848779244 -4.271087169
```

```
rchisq(n = 15,df=1)
```

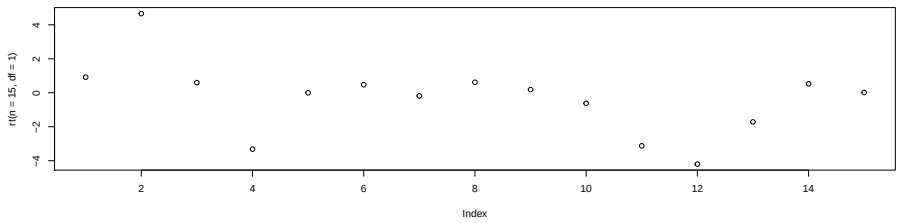
```
##  [1] 0.5317744 1.4263809 4.2797098 0.2184660 0.6923773 0.0455256 3.1902100
##  [8] 0.2949942 0.5403827 0.1543732 0.8639196 0.1417290 1.1386091 0.2966193
## [15] 0.5110879
```

Para poder ver rápidamente de qué se tratan los valores, podemos usar el comando plot

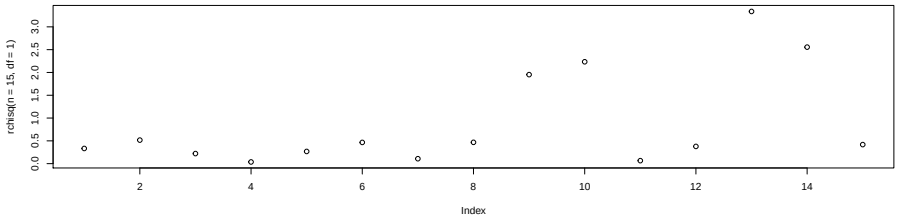
```
plot(rnorm(n = 15,mean = 0, sd = 1 ))
```




```
plot(rt(n = 15,df=1 ))
```




```
plot(rchisq(n = 15,df=1))
```



Noten que el eje X es el índice de los valores, es decir que no agrega información.

3.2.2 Tests

Utilicemos ahora datos reales.

los datos salen de https://data.buenosaires.gob.ar/dataset/femicidios

> Vamos a ver ahora las estadisticas de Buenos Aires sobre la cantidad de femicidios por grupo etario. Es interesante preguntarse si hay más femicidios para cierto rango etario.

```
femicidios <- read_csv(file = '../fuentes/vict_fem_annio__g_edad_limpio.csv')
femicidios
```

```
## # A tibble: 19 x 3
##     anio cantidad_femicidios grupo_edad
##    <dbl> <chr>               <chr>
##  1  2015 1                   0 - 15
##  2  2015 2                   16 - 20
##  3  2015 5                   21 - 40
##  4  2015 3                   41 - 60
##  5  2015 -                   61 y más
##  6  2015 1                   Ignorado
##  7  2016 2                   0 - 15
##  8  2016 3                   16 - 20
##  9  2016 4                   21 - 40
## 10  2016 1                   41 - 60
## 11  2016 2                   61 y más
## 12  2016 2                   Ignorado
## 13  2017 …                   0 - 15
## 14  2017 …                   16 - 20
## 15  2017 …                   21 - 40
## 16  2017 …                   41 - 60
## 17  2017 …                   61 y más
## 18  2017 …                   Ignorado
## 19  2017 9                   TOTAL
```

fijense que las estadísitcas no estan desagregadas por rango etario para 2017,

que en caso de que haya 0 femicidios pusieron '-' en lugar de 0. Además, como tenemos pocos datos, es mejor hacer un test que compare sólamente dos grupos.

Vamos a reorganizar la información para corregir todas estas cosas

```
femicidios <- femicidios %>%
  filter(anio!=2017, grupo_edad !='Ignorado') %>%  #Sacamos al 2017 y los casos donde
  mutate(cantidad_femicidios = case_when(cantidad_femicidios=='-' ~ 0, # reemplazamos
                                         TRUE ~as.numeric(cantidad_femicidios)), # y c
         grupo_edad = case_when(grupo_edad %in% c('0 - 15','16 - 20','21 - 40') ~ '0-40
                                grupo_edad %in% c('41 - 60','61 y más') ~ '41 y más'))
  group_by(grupo_edad) %>%
  summarise(cantidad_femicidios= sum(cantidad_femicidios)) # sumamos los años y grupos
femicidios
```

```
## # A tibble: 2 x 2
##   grupo_edad cantidad_femicidios
##   <chr>                    <dbl>
## 1 0-40                        17
## 2 41 y más                     6
```

Con esta tabla de contingencia podemos hacer un test de hipótesis.

¿Cuál usamos? Nos fijamos en el machete, o googleamos, y vemos que como queremos comparar la cantidad de casos por grupos categóricos, tenemos que usar el test Chi.

- H_0 No hay asociación entre las variables
- H_1 Hay asociación entre las variables

La idea es que tenemos dos variables: El rango etario y la cantidad de femicidios

```
chisq.test(femicidios$cantidad_femicidios)
```

```
##
##  Chi-squared test for given probabilities
##
## data:  femicidios$cantidad_femicidios
## X-squared = 5.2609, df = 1, p-value = 0.02181
```

noten que el resultado lo dan en términos del p-valor. Como el valor es bajo, menor a 0.05, entonces podemos rechazar que no existe relación. O en otros términos, pareciera que la diferencia es significativa estadísticamente.

3.2.3 Descripción estadística de los datos

Volveremos a ver los datos de sueldos de funcionarios

```
sueldos <- read_csv('../fuentes/sueldo_funcionarios_2019.csv')
```

Con el comando `summary` podemos ver algunos de los principales estadísticos de resumen

```
summary(sueldos$asignacion_por_cargo_i)
```

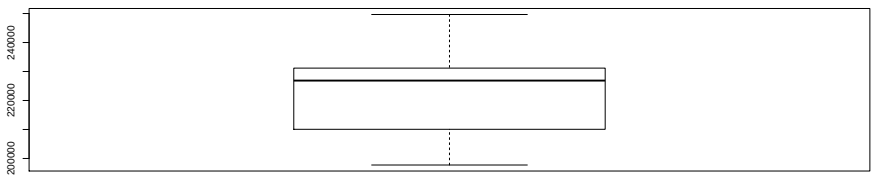
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  197746  210061  226866  225401  231168  249662
```

3.2.4 Gráficos estadísticos

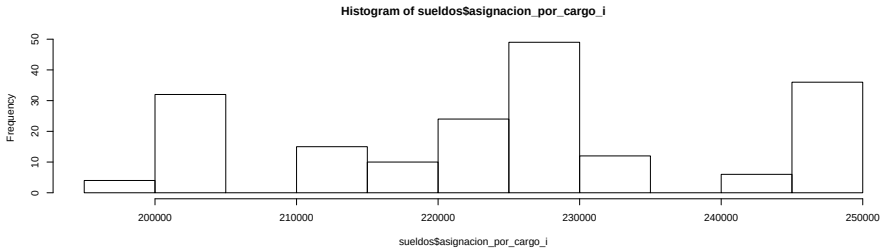
No nos vamos a detener demasiado a ver cómo hacer los gráficos de resumen, porque la próxima clase veremos como realizar gráficos de mejor calidad. Como los presentados en las notas de clase

A modo de ejemplo, dejamos los comandos de R base para realizar gráficos

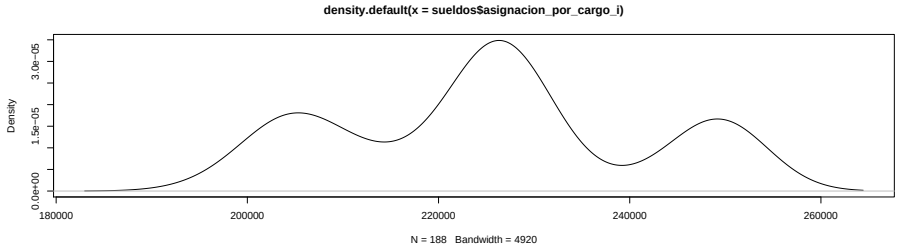
```
boxplot(sueldos$asignacion_por_cargo_i)
```




```
hist(sueldos$asignacion_por_cargo_i)
```




```
plot(density(sueldos$asignacion_por_cargo_i))
```


density.default(x = sueldos$asignacion_por_cargo_i)
Density
0.0e+00
1.5e−05
3.0e−05
180000
200000
220000
240000
260000
N = 188 Bandwidth = 4920

Chapter 4

Visualización de la información

En esta clase veremos como realizar gráficos en R, tanto los comandos básicos como utilizando la librería GGPLOT.

- Gráficos básicos de R (función "plot"): Comandos para la visualización ágil de la información
- Gráficos elaborados en R (función "ggplot"):
- Gráficos de línea, barras, Boxplots y distribuciones de densidad
- Parámetros de los gráficos: Leyendas, ejes, títulos, notas, colores
- Gráficos con múltiples cruces de variables.

4.1 Explicación

4.1.1 Gráficos Básicos en R

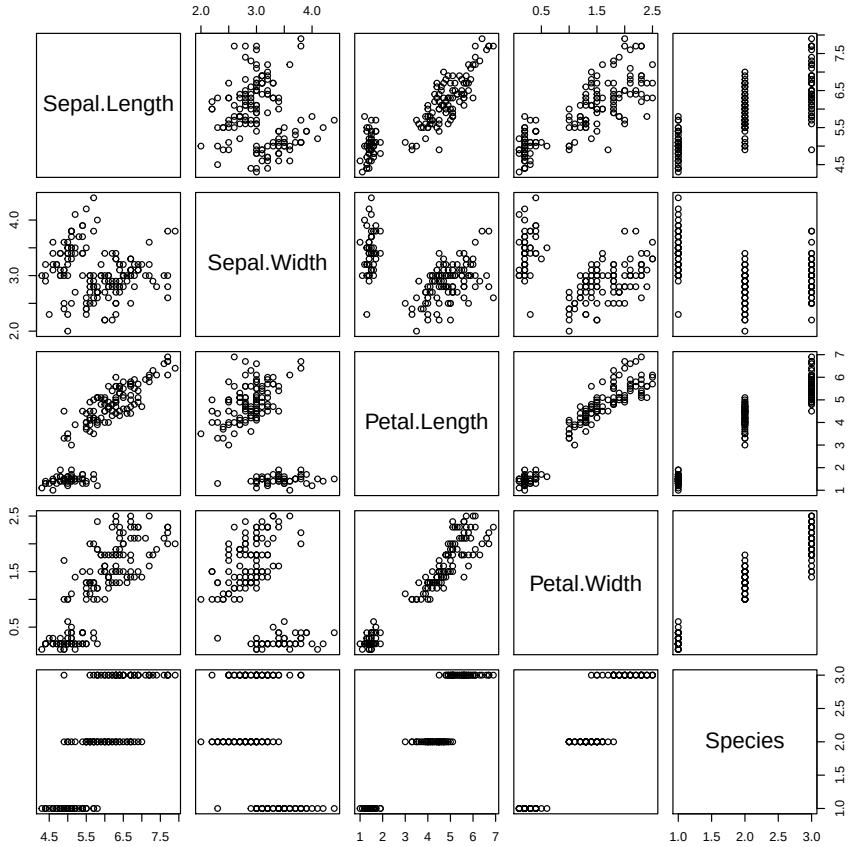
Rbase tiene algunos comandos genéricos para realizar gráficos, que se adaptan al tipo de información que se le pide graficar, por ejemplo:

- plot()
- hist()

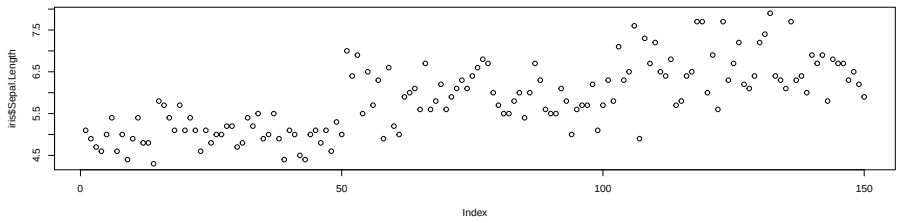
```
# iris es un set de datos clásico, que ya viene incorporado en R
iris[10,]
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 10          4.9         3.1          1.5         0.1  setosa
```

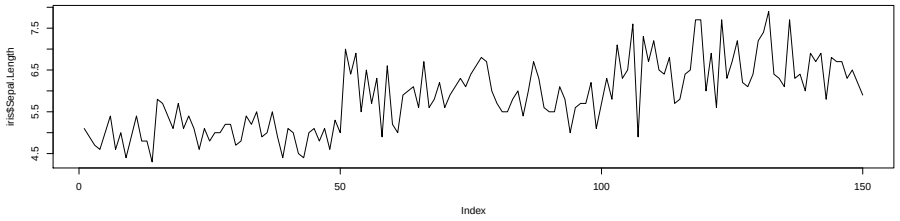
```
plot(iris)
```



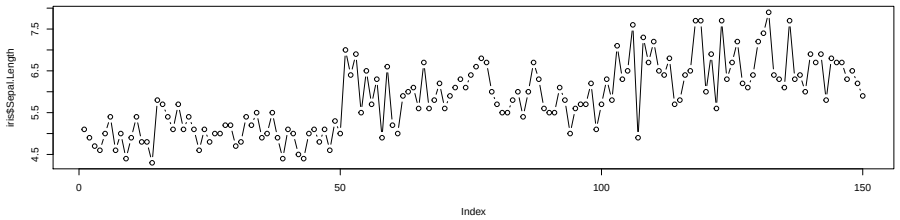

```r
#Al especificar una variable, puedo ver el valor que toma cada uno de sus registros (I
plot(iris$Sepal.Length,type = "p") # Un punto por cada valor
```



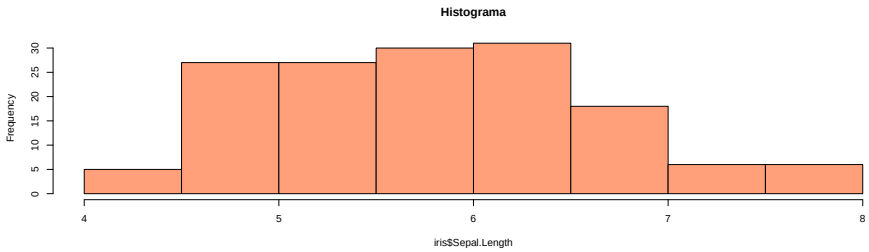

```r
plot(iris$Sepal.Length,type = "l") # Una linea que una cada valor
```




```
plot(iris$Sepal.Length,type = "b") #Ambas
```




```
hist(iris$Sepal.Length, col = "lightsalmon1", main = "Histograma")
```



4.1.1.1 png

La función `png()` nos permite grabar una imagen en el disco. Lleva como argumento principal la ruta completa a donde se desea guardar la misma, incluyendo el nombre que queremos dar al archivo. A su vez pueden especificarse otros argumetnos como el ancho y largo de la imagen, entre otros.

```
ruta_archivo <- "../resultados/grafico1.PNG"
ruta_archivo
```

```
## [1] "../resultados/grafico1.PNG"
```

```
png(ruta_archivo)
plot(iris$Sepal.Length,type = "b")
dev.off()
```

```
## pdf
##   2
```

La función `png()` *abre el dispositivo de imagen* en el directorio especificado. Luego creamos el gráfico que deseamos (o llamamos a uno previamente construido), el cual se desplegará en la ventana inferior derecha de la pantalla de Rstudio. Finalmente con `dev.off()` se *cierra el dispositivo* y se graban los gráficos.

Los gráficos del R base son útiles para escribir de forma rápida y obtener alguna información mientras trabajamos. Muchos paquetes estadísticos permiten mostrar los resultados de forma gráfica con el comando plot (por ejemplo, las regresiones lineales `lm()`).

Sin embargo, existen librerías mucho mejores para crear gráficos de nivel de publicación. La más importante es **ggplot2**, que a su vez tiene extensiones mediante otras librerías.

4.1.2 Ggplot2

ggplot tiene su sintaxis propia. La idea central es pensar los gráficos como una sucesión de capas, que se construyen una a la vez.

- El operador `+` nos permite incorporar nuevas capas al gráfico.
- El comando `ggplot()` nos permite definir la fuente de **datos** y las **variables** que determinaran los ejes del grafico (x,y), así como el color y la forma de las líneas o puntos,etc.
- Las sucesivas capas nos permiten definir:
  - Uno o más tipos de gráficos (de columnas, `geom_col()`, de línea, `geom_line()`, de puntos,`geom_point()`, boxplot, `geom_boxplot()`)
  - Títulos `labs()`
  - Estilo del gráfico `theme()`
  - Escalas de los ejes `scale_y_continuous`,`scale_x_discrete`
  - División en subconjuntos `facet_wrap()`,`facet_grid()`

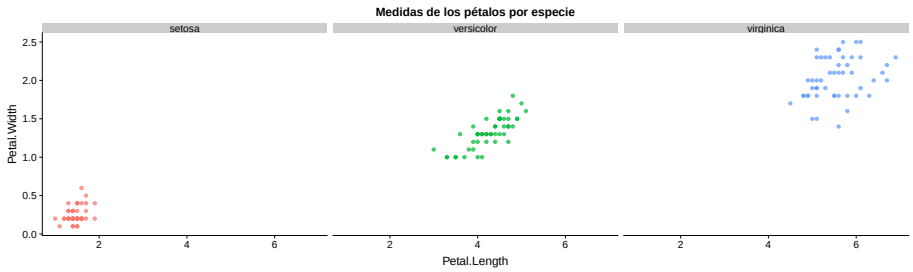
ggplot tiene **muchos** comandos, y no tiene sentido saberlos de memoria, es siempre útil reutilizar gráficos viejos y tener a mano el machete.

4.1.2.1 Gráfico de Puntos

A continuación se muestra un gráfico de varias capas de construcción, con su correspondiente porción de código. En el mismo se buscará visualizar, a partir de la base de datos **iris** la relación entre el ancho y el largo de los petalos, mediante un gráfico de puntos.

```
library(ggplot2) # cargamos la librería
ggplot(data = iris, aes(x = Petal.Length, Petal.Width, color = Species))+
```

```
  geom_point(alpha=0.75)+
  labs(title = "Medidas de los pétalos por especie")+
  theme(legend.position = 'none')+
  facet_wrap(~Species)
```

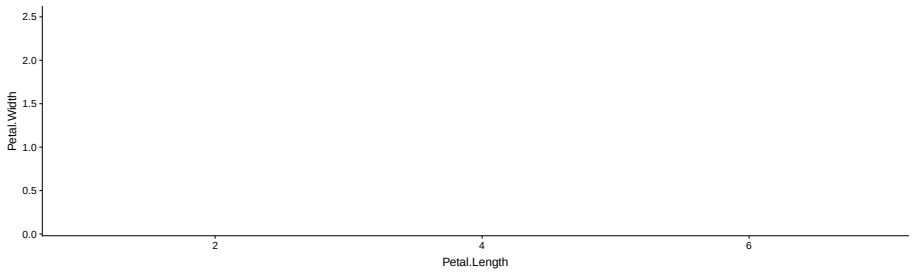


4.1.2.2 Capas del Gráfico

Veamos ahora, el "paso a paso" del armado del mismo.

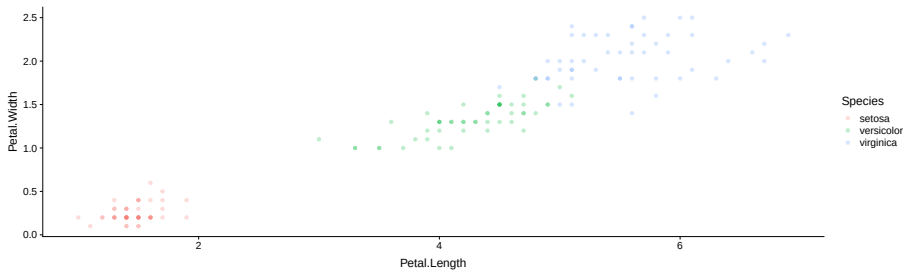
En primera instancia solo defino los ejes. Y en este caso un color particular para cada Especie.

```
g <- ggplot(data = iris, aes(x = Petal.Length, Petal.Width, color = Species))
g
```



Luego, defino el tipo de gráfico. El *alpha* me permite definir la intensidad de los puntos

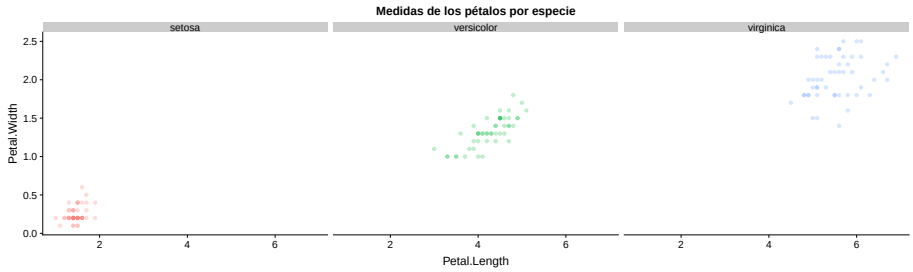
```
g <- g +  geom_point(alpha=0.25)
g
```



Las siguientes tres capas me permiten respectivamente:

- Definir el título del gráfico
- Quitar la leyenda
- Abrir el gráfico en tres fragmentos, uno para cada especie

```
g <- g +
  labs(title = "Medidas de los pétalos por especie")+
  theme(legend.position = 'none')+
  facet_wrap(~Species)
g
```

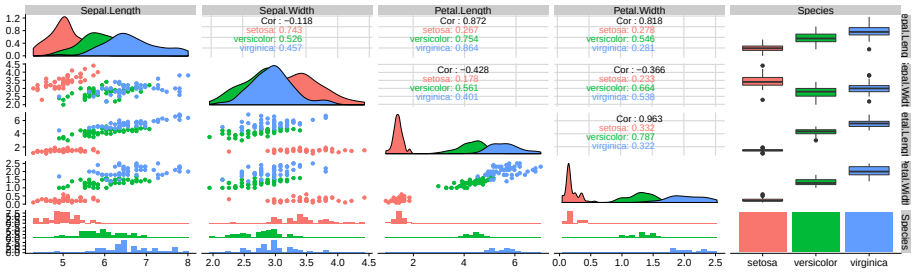


4.1.2.3 Extensiones de GGplot.

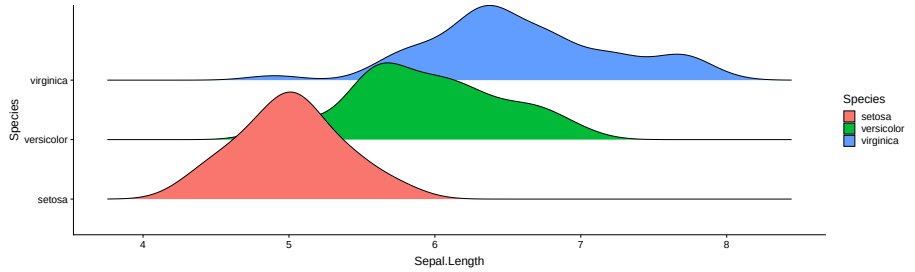
La librería GGplot tiene a su vez muchas otras librerías que extienden sus potencialidades. Entre nuestras favoritas están:

- gganimate: Para hacer gráficos animados.
- ggridge: Para hacer gráficos de densidad faceteados
- ggally: Para hacer varios gráficos juntos. ^

```
library(GGally)

ggpairs(iris,  mapping = aes(color = Species))
```




```
library(ggridges)
ggplot(iris, aes(x = Sepal.Length, y = Species, fill=Species)) +
  geom_density_ridges()
```



También hay extensiones que te ayudan a escribir el código, como esquisse

```
iris <- iris
#Correr en la consola
esquisse::esquisser()
```

4.1.3 Dimensiones del gráfico

Esta forma de pensar los gráficos nos permite repenser los distintos atributos como potenciales aliados a la hora de mostrar información multidimensional. Por ejemplo:

- **color** `color =`
- **rellenofill** `=`
- **forma** `shape =`
- **tamaño** `size =`
- **transparencia** `alpha =`
- Abrir un mismo gráfico según alguna variable discreta: `facet_wrap()`
- Los atributos que queremos que *mapeen* una variable, deben ir **dentro del aes()**, `aes(... color = variable)`

- Cuando queremos simplemente mejorar el diseño (es fijo), se asigna por fuera, o dentro de cada tipo de gráficos, `geom_col(color = 'green')`.

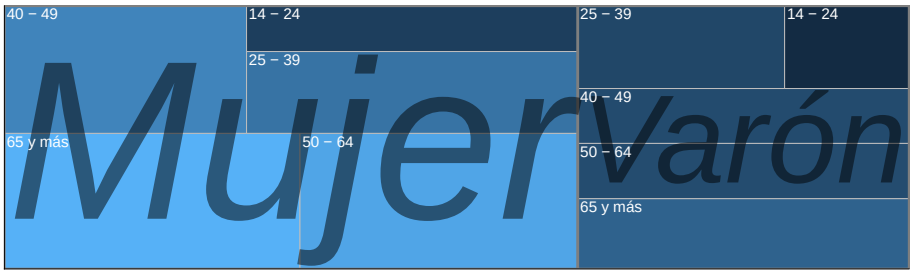
4.1.4 Treemaps

```
library(treemapify)
```

Trabajo doméstico no remunerado

```
trabajo_no_remunerado <- read_csv('../fuentes/prom_t_simul_dom_16_sexo__annio__g_edad_l
```

```
 trabajo_no_remunerado %>%
  filter(sexo != 'TOTAL', grupo_edad != 'TOTAL') %>%
   mutate(promedio_hs_diarias = as.numeric(promedio_hs_diarias),
          sexo = case_when(sexo=='m'~'Mujer',
                           sexo=='v'~'Varón')) %>%
  ggplot(., aes(area = promedio_hs_diarias, fill = promedio_hs_diarias, label = grupo_e
                subgroup = sexo)) +
  geom_treemap() +
  geom_treemap_subgroup_border() +
  geom_treemap_subgroup_text(place = "centre", grow = T, alpha = 0.5, colour =
                               "black", fontface = "italic", min.size = 0) +
  geom_treemap_text(colour = "white", place = "topleft", reflow = T)+
  theme(legend.position = 'none')
```




```
 trabajo_no_remunerado %>%
  filter(sexo != 'TOTAL', grupo_edad != 'TOTAL') %>%
   mutate(promedio_hs_diarias = as.numeric(promedio_hs_diarias)) %>%
ggplot(., aes(area=promedio_hs_diarias, fill=sexo, label=sexo))+
  geom_treemap() +
  geom_treemap_text(colour = "white", place = "topleft", reflow = T)+
  facet_wrap(.~grupo_edad, ncol = 1)
```

4.2 Práctica Guiada

4.2.1 Graficos Ingresos - EPH

Para esta práctica utilizaremos las variables de ingresos captadas por la Encuesta Permanente de Hogares

A continuación utilzaremos los conceptos abordados, para realizar gráficos a partir de las variables de ingresos.

```
#Cargamos las librerías a utilizar

library(tidyverse) # tiene ggplot, dplyr, tidyr, y otros
library(ggthemes)  # estilos de gráficos
library(ggrepel)   # etiquetas de texto más prolijas que las de ggplot
```

```
Individual_t119 <- read.table("../fuentes/usu_individual_t119.txt",
                              sep=";", dec=",", header = TRUE, fill = TRUE)
```

4.2.1.1 Boxplot de ingresos de la ocupación principal, según nivel educativo

Hacemos un procesamiento simple: Sacamos los ingresos iguales a cero y las no respuestas de nivel educativo.
Es importante que las variables sean del tipo que conceptualmente les corresponde (el nivel educativo es una variable categórica, no continua), para que el ggplot pueda graficarlo correctamente.

```
# Las variables sexo( CH04 ) y Nivel educativo están codificadas como números, y el R las entiend
class(Individual_t119$NIVEL_ED)
```

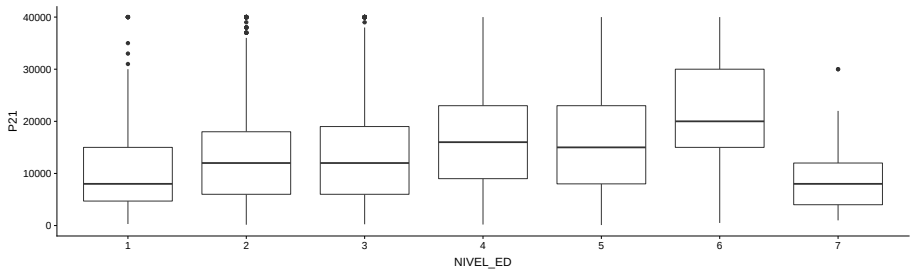
```
## [1] "integer"
```

```
class(Individual_t119$CH04)
```

```
## [1] "integer"
```

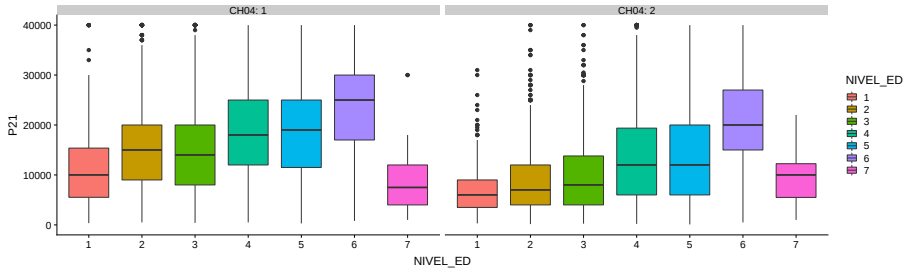
```
ggdata <- Individual_t119 %>%
  filter(P21>0, !is.na(NIVEL_ED)) %>%
  mutate(NIVEL_ED = as.factor(NIVEL_ED),
         CH04     = as.factor(CH04))
```

```
ggplot(ggdata, aes(x = NIVEL_ED, y = P21)) +
  geom_boxplot()+
  scale_y_continuous(limits = c(0, 40000))#Restrinjo el gráfico hasta ingresos de $400
```



Si queremos agregar la dimensión *sexo*, podemos hacer un `facet_wrap()`

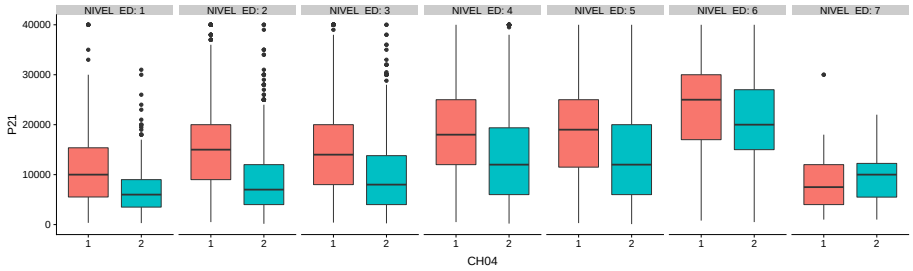
```
ggplot(ggdata, aes(x= NIVEL_ED, y = P21, group = NIVEL_ED, fill = NIVEL_ED )) +
  geom_boxplot()+
  scale_y_continuous(limits = c(0, 40000))+
  facet_wrap(~ CH04, labeller = "label_both")
```



Por la forma en que está presentado el gráfico, el foco de atención sigue puesto en las diferencias de ingresos entre niveles educativo. Simplemente se agrega un corte por la variable de sexo.

Si lo que queremos hacer es poner el foco de atención en las diferencias por sexo, simplemente basta con invertir la variable x especificada con la variable utilizada en el `facet_wrap`

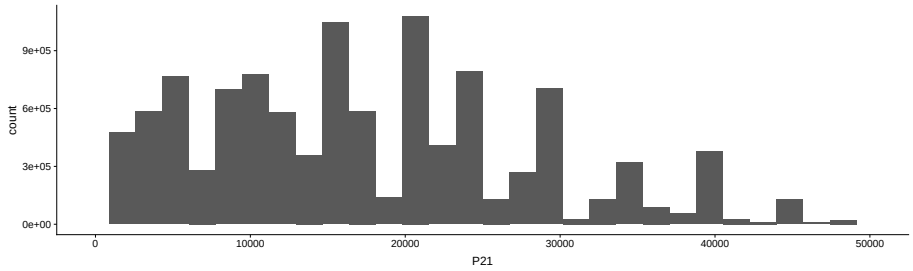
```
ggplot(ggdata, aes(x= CH04, y = P21, group = CH04, fill = CH04 )) +
  geom_boxplot()+
  scale_y_continuous(limits = c(0, 40000))+
  facet_grid(~ NIVEL_ED, labeller = "label_both") +
  theme(legend.position = "none")
```



4.2.2 Histogramas

Por ejemplo, si observamos el ingreso de la ocupación principal:

```
hist_data <-Individual_t119 %>%
  filter(P21>0)

ggplot(hist_data, aes(x = P21,weights = PONDIIO))+
geom_histogram()+
scale_x_continuous(limits = c(0,50000))
```



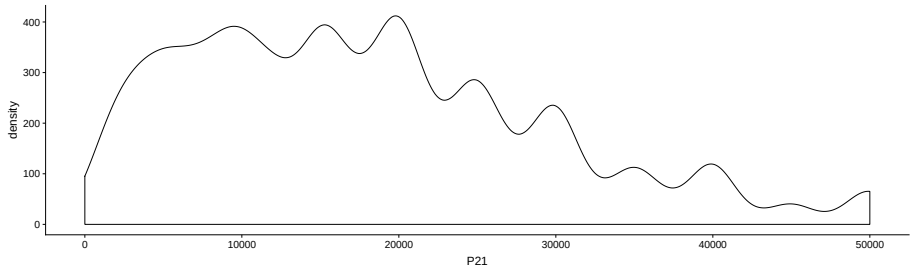
En este gráfico, los posibles valores de p21 se dividen en 30 **bins** consecutivos y el gráfico muestra cuantas observaciones caen en cada uno de ellos

4.2.3 Kernels

La función `geom_density()` nos permite construir **kernels** de la distribución. Es particularmente útil cuando tenemos una variable continua, dado que los histogramas rompen esa sensación de continuidad.

Veamos un ejemplo sencillo con los ingresos de la ocupación principal. Luego iremos complejizandolo

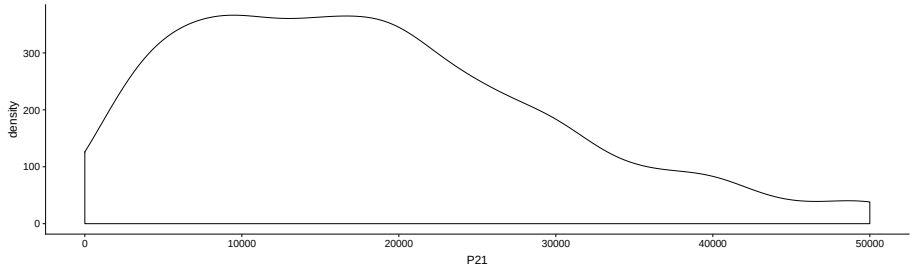
```
kernel_data <-Individual_t119 %>%
  filter(P21>0)

ggplot(kernel_data, aes(x = P21,weights = PONDIIO))+
geom_density()+
scale_x_continuous(limits = c(0,50000))
```



El eje y no tiene demasiada interpretabilidad en los Kernel, porque hace a la forma en que se construyen las distribuciones.

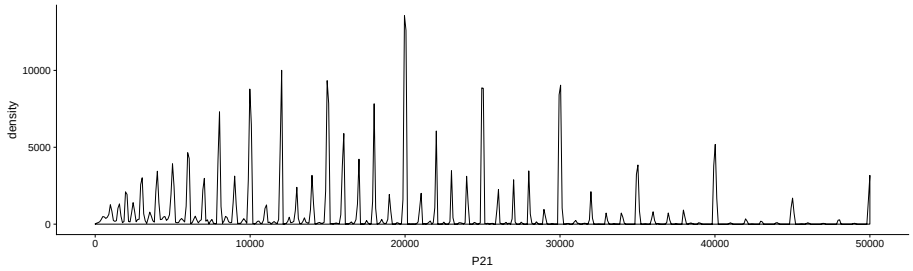
El parametro adjust, dentro de la función `geom_density`nos permite reducir o ampliar el rango de suavizado de la distribución. Su valor por default es 1. Veamos que sucede si lo seteamos en 2

```
ggplot(kernel_data, aes(x = P21,weights = PONDIIO))+
geom_density(adjust = 2)+
scale_x_continuous(limits = c(0,50000))
```



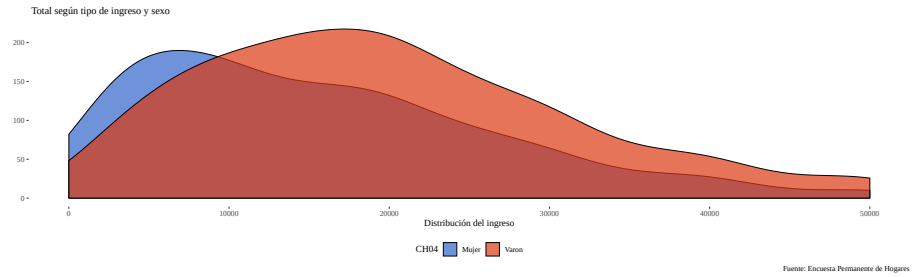
Como es esperable, la distribución del ingreso tiene "picos" en los valores redondos, ya que la gente suele declarar un valor aproximado al ingreso efectivo que percibe. Nadie declara ingresos de 30001. Al suavizar la serie con un kernel, eliminamos ese efecto.Si seteamos el rango para el suavizado en valores menores a 1, podemos observar estos picos.

```
ggplot(kernel_data, aes(x = P21,weights = PONDIIO))+
geom_density(adjust = 0.01)+
scale_x_continuous(limits = c(0,50000))
```



Ahora bien, como en todo grafico de R, podemos seguir agregando dimensiones para enriquecer el análisis.

```
kernel_data_2 <- kernel_data %>%
  mutate(CH04= case_when(CH04 == 1 ~ "Varon",
                         CH04 == 2 ~ "Mujer"))

ggplot(kernel_data_2, aes(x = P21,
  weights = PONDIIO,
  group = CH04,
  fill = CH04)) +
  geom_density(alpha=0.7,adjust =2)+
  labs(x="Distribución del ingreso", y="",
       title=" Total según tipo de ingreso y sexo",
       caption = "Fuente: Encuesta Permanente de Hogares")+
  scale_x_continuous(limits = c(0,50000))+
  theme_tufte()+
  scale_fill_gdocs()+
  theme(legend.position = "bottom",
        plot.title      = element_text(size=12))
```




```
ggsave(filename = "../resultados/Kernel_1.png",scale = 2)
```

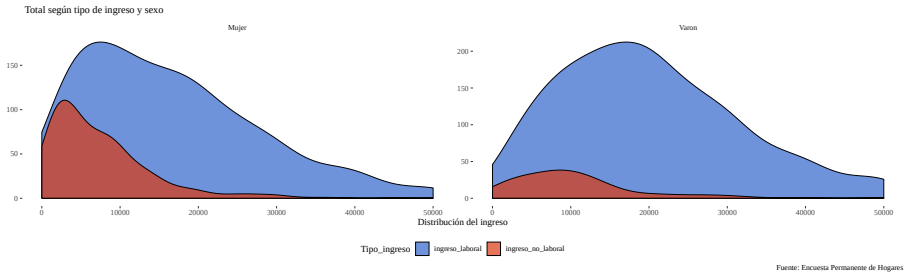
Podemos agregar aún la dimensión de ingreso laboral respecto del no laboral

```
kernel_data_3 <-kernel_data_2 %>%
  select(REGION,P47T,T_VI, TOT_P12, P21 , PONDII, CH04) %>%
  filter(!is.na(P47T), P47T > 0 ) %>%
```

```
  mutate(ingreso_laboral    = TOT_P12 + P21,
         ingreso_no_laboral = T_VI) %>%
  gather(., key = Tipo_ingreso, Ingreso, c((ncol(.)-1):ncol(.))) %>%
  filter( Ingreso !=0)# Para este gráfico, quiero eliminar los ingresos = 0

kernel_data_3 %>%
  sample_n(10)
```

```
##    REGION  P47T T_VI TOT_P12   P21 PONDII  CH04       Tipo_ingreso Ingreso
## 1      44 20000    0       0 20000    180 Mujer    ingreso_laboral   20000
## 2      40  7400 5600       0  1800    404 Varon ingreso_no_laboral    5600
## 3      42 10000    0       0 10000     96 Mujer    ingreso_laboral   10000
## 4       1 41000 8000       0 33000   3451 Mujer ingreso_no_laboral    8000
## 5      40 27000    0       0 27000    208 Mujer    ingreso_laboral   27000
## 6      43 45000    0       0 30000    799 Mujer    ingreso_laboral   30000
## 7       1 11500 9500       0  2000    874 Mujer ingreso_no_laboral    9500
## 8      42 35000    0       0 35000    618 Varon    ingreso_laboral   35000
## 9      40 30000    0       0 20000    344 Mujer    ingreso_laboral   20000
## 10     41  8500    0       0  8500    328 Mujer    ingreso_laboral    8500
```

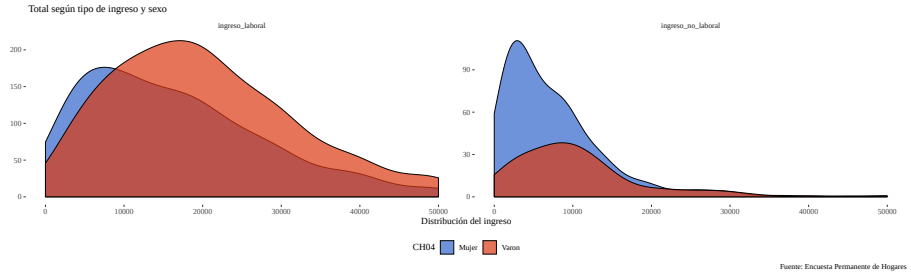
```
  ggplot(kernel_data_3, aes(
  x = Ingreso,
  weights = PONDII,
  group = Tipo_ingreso,
  fill = Tipo_ingreso)) +
  geom_density(alpha=0.7,adjust =2)+
  labs(x="Distribución del ingreso", y="",
       title=" Total según tipo de ingreso y sexo",
       caption = "Fuente: Encuesta Permanente de Hogares")+
  scale_x_continuous(limits = c(0,50000))+
  theme_tufte()+
  scale_fill_gdocs()+
  theme(legend.position = "bottom",
        plot.title      = element_text(size=12))+
  facet_wrap(~ CH04, scales = "free")
```



```
ggsave(filename = "../resultados/Kernel_2.png",scale = 2)
```

En este tipo de gráficos, importa mucho qué variable se utiliza para *facetear* y qué variable para agrupar, ya que la construcción de la distribución es diferente.

```
ggplot(kernel_data_3, aes(
  x = Ingreso,
  weights = PONDII,
  group = CH04,
  fill = CH04)) +
  geom_density(alpha=0.7,adjust =2)+
  labs(x="Distribución del ingreso", y="",
       title=" Total según tipo de ingreso y sexo",
       caption = "Fuente: Encuesta Permanente de Hogares")+
  scale_x_continuous(limits = c(0,50000))+
  theme_tufte()+
  scale_fill_gdocs()+
  theme(legend.position = "bottom",
        plot.title      = element_text(size=12))+
  facet_wrap(~Tipo_ingreso, scales = "free")
```




```
ggsave(filename = "../resultados/Kernel_3.png",scale = 2)
```

Chapter 5

Visualización de la información

- Manejo de las extensiones del software "Rmarkdown" y "RNotebook" para elaborar documentos de trabajo, presentaciones interactivas e informes:
- Opciones para mostrar u ocultar código en los reportes
- Definición de tamaño, títulos y formato con el cual se despliegan los gráficos y tablas en el informe
- Caracteres especiales para incluir múltiples recursos en el texto del informe: Links a páginas web, notas al pie, enumeraciones, cambios en el formato de letra (tamaño, negrita, cursiva)
- Código embebido en el texto para automatización de reportes

5.1 Explicación

5.2 Práctica Guiada

Chapter 6

Visualización de la información

- Introducción al diseño de encuestas
- Presentación de la Encuesta Permanente de Hogares
- Generación de estadísticos de resumen en muestras estratificadas
- Utilización de los ponderadores

6.1 Explicación

6.2 Práctica Guiada

Chapter 7

Programacion Funcional

El objetivo de esta clase es introducir a los alumnos en el uso de la programación funcional. Es decir, en la utilización de funciones y el uso de controles de flujo de la información para la organización de su código.

- Estructuras de código condicionales
- Loops
- Creación de funciones a medida del usuario
- Librería purrr para programación funcional

7.1 Explicación

```
library(tidyverse)
```

7.1.1 Loops

Un **loop** es una estructura de código que nos permite aplicar iterativamente un mismo conjunto de comandos, variando el valor de una variable. Por ejemplo:

```
for(i in 1:10){
   print(i^2)
}
```

```
## [1] 1
## [1] 4
## [1] 9
## [1] 16
## [1] 25
## [1] 36
## [1] 49
```

```
## [1] 64
## [1] 81
## [1] 100
```

Esto se lee como : "Recorre cada uno de los valores (i) del vector numérico 1 a 10, y para cada uno de ellos imprimí el cuadrado (i^2)".
Uno puede especificar la palabra que desee que tomé cada uno de los valores que debe tomar. En el ejemplo anterior fue **i**, pero bien podría ser la "**Valores**"

```
for(Valores in 1:10){
   print(Valores^2)

}
```

```
## [1] 1
## [1] 4
## [1] 9
## [1] 16
## [1] 25
## [1] 36
## [1] 49
## [1] 64
## [1] 81
## [1] 100
```

Un loop puede iterar sobre cualquier tipo de vector, independientemente de lo que contenga.

> Los loops son una estructura básica que existen en cualquier lenguaje de programación. En R no recomendamos abusar de ellos porque hacen que el código sea más lento.

7.1.2 Estructuras Condicionales

Las **estructuras condiconales** nos permiten ejecutar una porción de código en caso de que cumplan una condición lógica

7.1.2.1 if

Su funcionamiento es el siguiente:
`if(condicion){codigo a ejecutar si se cumple la condición}`

```
if( 2+2 == 4){
  print("Menos Mal")
}
```

```
## [1] "Menos Mal"
```

```
if( 2+2 == 148.24){
  print("R, tenemos un problema")
}
```

7.1.2.2 ifelse

La función `if_else()` sirve para crear o modificar dicotómicamente un objeto/variable/vector a partir del cumplimiento de una o más condiciones lógicas.
Su funcionamiento es el siguiente:
`if_else(condicion,función a aplicar si se cumple la condición,función a aplicar si no se cumple la condición)`

```
if_else(2+2==4, true = "Joya",false = "Error")
```

```
## [1] "Joya"
```

7.1.3 Funciones

La creación de **funciones** propias nos permite automatizar todas aquellas partes del código que se repiten mucho. Una vez diseñadas, funcionan igual que cualquier comando.

Por ejemplo, podemos definir la suma de dos elementos como

```
suma <- function(valor1, valor2) {
  valor1+valor2
}

suma(5,6)
```

```
## [1] 11
```

Obviamente las funciones no son sólo para variables numéricas. Por ejemplo, podemos pegar dos strings con una flecha en el medio

```
funcion_prueba <- function(parametro1,parametro2) {
  paste(parametro1, parametro2, sep = " <--> ")
}

funcion_prueba(parametro1 = "A ver", parametro2 = "Que pasa")
```

```
## [1] "A ver <--> Que pasa"
```

También podemos asignar un valor por default para los parametros en caso de que el usuario no defina su valor al utilizar la función.

```
Otra_funcion_prueba <- function(parametro1 ,parametro2 = "String default") {
  paste(parametro1, parametro2, sep = " <--> ")
```

```
}
Otra_funcion_prueba(parametro1 = "Valor 1 ")
```

```
## [1] "Valor 1  <--> String default"
```

Las funciones que creamos nosotros permanecen en el ambiente de R temporariamente. Cuando removemos los objetos del ambiente, la función deja de existir. Por ende, debemos incorporarla en cada uno de los scripts en la cual la necesitemos. Una buena práctica, es incorporar nuestras funciones útiles al comienzo de cada script junto a la carga de las librerías.

Vale mencionar que **lo que ocurre en una función, queda en la función** excepto que explícitamente pidamos que devuelva el resultado, con el comando `print()`.

Las funciones siempre devuelven el último objeto que se crea en ellas, o si explicitamente se utiliza el comando `return()`

7.1.4 PURRR[1]

MAP es la forma *tidy* de hacer loops. Además de ser más prolijo el código, es mucho más eficiente.

La función **map** toma un input, una función para aplicar, y alguna otra cosa (por ejemplo parametros que necesite la función)

- map(.x, .f, ...)
- map(VECTOR_O_LIST_INPUT, FUNCTION_A_APLICAR, OTROS_OPCIONALES)

Usamos **map2** cuando tenemos que pasar dos input, que se aplican sobre una función:

- map2(.x, .y, .f, ...)
- map2(INPUT_UNO, INPUT_DOS, FUNCTION_A_APLICAR, OTROS_OPCIONALES)

Si tenemos más de dos...

- pmap(.l, .f, ...)
- pmap(VECTOR_O_LIST_INPUT, FUNCTION_A_APLICAR, OTROS_OPCIONALES)

Por ejemplo. Si queremos utilizar la función prueba sobre los datos del dataframe ABC_123

```
ABC_123 <- data.frame(Letras = LETTERS[1:20],Num = 1:20)
funcion_prueba
```

[1] basado en https://jennybc.github.io/purrr-tutorial/ls03_map-function-syntax.html

```
## function(parametro1,parametro2) {
##   paste(parametro1, parametro2, sep = " <--> ")
## }
```

Si el resultado que queremos es que junte cada fila, necesitamos pasarle dos parámetros: utilizamos `map2()`

```
resultado <- map2(ABC_123$Letras,ABC_123$Num,funcion_prueba)
resultado[1:3]
```

```
## [[1]]
## [1] "A <--> 1"
##
## [[2]]
## [1] "B <--> 2"
##
## [[3]]
## [1] "C <--> 3"
```

La salida de los `map()` es una **lista**, no un vector, por lo que si lo metemos dentro de un dataframe se vería así:

```
ABC_123 %>%
  mutate(resultado= map2(Letras,Num,funcion_prueba))
```

```
##    Letras Num resultado
## 1       A   1  A <--> 1
## 2       B   2  B <--> 2
## 3       C   3  C <--> 3
## 4       D   4  D <--> 4
## 5       E   5  E <--> 5
## 6       F   6  F <--> 6
## 7       G   7  G <--> 7
## 8       H   8  H <--> 8
## 9       I   9  I <--> 9
## 10      J  10 J <--> 10
## 11      K  11 K <--> 11
## 12      L  12 L <--> 12
## 13      M  13 M <--> 13
## 14      N  14 N <--> 14
## 15      O  15 O <--> 15
## 16      P  16 P <--> 16
## 17      Q  17 Q <--> 17
## 18      R  18 R <--> 18
## 19      S  19 S <--> 19
## 20      T  20 T <--> 20
```

al ponerlo dentro del dataframe desarma la lista y guarda cada elemento por separado. La magia de eso es que podemos **guardar cualquier cosa en el**

dataframe no sólo valores, sino también listas, funciones, dataframes, etc.

Si queremos recuperar los valores originales en este caso podemos usar `unlist()`

```
resultado[1:3] %>% unlist()
```

```
## [1] "A <--> 1" "B <--> 2" "C <--> 3"
```

```
ABC_123 %>%
  mutate(resultado= unlist(map2(Letras,Num,funcion_prueba)))
```

```
##    Letras Num resultado
## 1       A   1  A <--> 1
## 2       B   2  B <--> 2
## 3       C   3  C <--> 3
## 4       D   4  D <--> 4
## 5       E   5  E <--> 5
## 6       F   6  F <--> 6
## 7       G   7  G <--> 7
## 8       H   8  H <--> 8
## 9       I   9  I <--> 9
## 10      J  10 J <--> 10
## 11      K  11 K <--> 11
## 12      L  12 L <--> 12
## 13      M  13 M <--> 13
## 14      N  14 N <--> 14
## 15      O  15 O <--> 15
## 16      P  16 P <--> 16
## 17      Q  17 Q <--> 17
## 18      R  18 R <--> 18
## 19      S  19 S <--> 19
## 20      T  20 T <--> 20
```

Si lo que queríamos era que la función nos haga todas las combinaciones de letras y número, entonces lo que necesitamos es pasarle el segúndo parametro como algo *fijo*, poniendolo después de la función.

```
map(ABC_123$Letras,funcion_prueba,ABC_123$Num)[1:2]
```

```
## [[1]]
##  [1] "A <--> 1"  "A <--> 2"  "A <--> 3"  "A <--> 4"  "A <--> 5"
##  [6] "A <--> 6"  "A <--> 7"  "A <--> 8"  "A <--> 9"  "A <--> 10"
## [11] "A <--> 11" "A <--> 12" "A <--> 13" "A <--> 14" "A <--> 15"
## [16] "A <--> 16" "A <--> 17" "A <--> 18" "A <--> 19" "A <--> 20"
##
## [[2]]
##  [1] "B <--> 1"  "B <--> 2"  "B <--> 3"  "B <--> 4"  "B <--> 5"
##  [6] "B <--> 6"  "B <--> 7"  "B <--> 8"  "B <--> 9"  "B <--> 10"
## [11] "B <--> 11" "B <--> 12" "B <--> 13" "B <--> 14" "B <--> 15"
```

```
## [16] "B <--> 16" "B <--> 17" "B <--> 18" "B <--> 19" "B <--> 20"
```

En este caso, el map itera sobre cada elemento de `letras`, y para cada elemento i hace `funcion_prueba(i,ABC$Num)` y guarda el resultado en la lista

si lo queremos meter en el dataframe

```
ABC_123 %>%
  mutate(resultado= map(Letras,funcion_prueba,Num))
```

```
##    Letras Num
## 1       A   1
## 2       B   2
## 3       C   3
## 4       D   4
## 5       E   5
## 6       F   6
## 7       G   7
## 8       H   8
## 9       I   9
## 10      J  10
## 11      K  11
## 12      L  12
## 13      M  13
## 14      N  14
## 15      O  15
## 16      P  16
## 17      Q  17
## 18      R  18
## 19      S  19
## 20      T  20
##
## 1  A <--> 1, A <--> 2, A <--> 3, A <--> 4, A <--> 5, A <--> 6, A <--> 7, A <--> 8, A <--> 9, A
## 2  B <--> 1, B <--> 2, B <--> 3, B <--> 4, B <--> 5, B <--> 6, B <--> 7, B <--> 8, B <--> 9, B
## 3  C <--> 1, C <--> 2, C <--> 3, C <--> 4, C <--> 5, C <--> 6, C <--> 7, C <--> 8, C <--> 9, C
## 4  D <--> 1, D <--> 2, D <--> 3, D <--> 4, D <--> 5, D <--> 6, D <--> 7, D <--> 8, D <--> 9, D
## 5  E <--> 1, E <--> 2, E <--> 3, E <--> 4, E <--> 5, E <--> 6, E <--> 7, E <--> 8, E <--> 9, E
## 6  F <--> 1, F <--> 2, F <--> 3, F <--> 4, F <--> 5, F <--> 6, F <--> 7, F <--> 8, F <--> 9, F
## 7  G <--> 1, G <--> 2, G <--> 3, G <--> 4, G <--> 5, G <--> 6, G <--> 7, G <--> 8, G <--> 9, G
## 8  H <--> 1, H <--> 2, H <--> 3, H <--> 4, H <--> 5, H <--> 6, H <--> 7, H <--> 8, H <--> 9, H
## 9  I <--> 1, I <--> 2, I <--> 3, I <--> 4, I <--> 5, I <--> 6, I <--> 7, I <--> 8, I <--> 9, I
## 10 J <--> 1, J <--> 2, J <--> 3, J <--> 4, J <--> 5, J <--> 6, J <--> 7, J <--> 8, J <--> 9, J
## 11 K <--> 1, K <--> 2, K <--> 3, K <--> 4, K <--> 5, K <--> 6, K <--> 7, K <--> 8, K <--> 9, K
## 12 L <--> 1, L <--> 2, L <--> 3, L <--> 4, L <--> 5, L <--> 6, L <--> 7, L <--> 8, L <--> 9, L
## 13 M <--> 1, M <--> 2, M <--> 3, M <--> 4, M <--> 5, M <--> 6, M <--> 7, M <--> 8, M <--> 9, M
## 14 N <--> 1, N <--> 2, N <--> 3, N <--> 4, N <--> 5, N <--> 6, N <--> 7, N <--> 8, N <--> 9, N
## 15 O <--> 1, O <--> 2, O <--> 3, O <--> 4, O <--> 5, O <--> 6, O <--> 7, O <--> 8, O <--> 9, O
```

```
## 16 P <--> 1, P <--> 2, P <--> 3, P <--> 4, P <--> 5, P <--> 6, P <--> 7, P <--> 8, P
## 17 Q <--> 1, Q <--> 2, Q <--> 3, Q <--> 4, Q <--> 5, Q <--> 6, Q <--> 7, Q <--> 8, Q
## 18 R <--> 1, R <--> 2, R <--> 3, R <--> 4, R <--> 5, R <--> 6, R <--> 7, R <--> 8, R
## 19 S <--> 1, S <--> 2, S <--> 3, S <--> 4, S <--> 5, S <--> 6, S <--> 7, S <--> 8, S
## 20 T <--> 1, T <--> 2, T <--> 3, T <--> 4, T <--> 5, T <--> 6, T <--> 7, T <--> 8, T
```

Ahora cada fila tiene un vector de 20 elementos guardado en la columna resultado

7.1.5 Funciones implícitas

no es necesario que definamos la función de antemano. Podemos usar *funciones implícitas*

```
map_dbl(c(1:10), function(x) x^2)
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

```
map2_dbl(c(1:10),c(11:20), function(x,y) x*y)
```

```
##  [1]  11  24  39  56  75  96 119 144 171 200
```

7.1.6 Funciones lambda

incluso más conciso que las funciones implíictas son las **funciones lambda** donde definimos las variables como *.x .y*, etc. La flexibilidad de estas expresiones es limitada, pero puede ser útil en algunos casos.

```
map_dbl(c(1:10),~.x^2)
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

```
map2_dbl(c(1:10),c(11:20),~.x*.y)
```

```
##  [1]  11  24  39  56  75  96 119 144 171 200
```

7.1.7 Walk

Las funciones `Walk` Tienen la misma forma que los `map`, pero se usan cuando lo que queremos iterar no genera una salida, sino que nos interesan los efectos secundarios que generan.

```
map2(ABC_123$Letras,ABC_123$Num,funcion_prueba)[1:3]
```

```
## [[1]]
## [1] "A <--> 1"
##
## [[2]]
## [1] "B <--> 2"
##
```

```
## [[3]]
## [1] "C <--> 3"
```

```
walk2(ABC_123$Letras,ABC_123$Num,funcion_prueba)
```

```
imprimir_salida <- function(x,y){
  print(funcion_prueba(x,y))
}

walk2(ABC_123$Letras,ABC_123$Num,imprimir_salida)
```

```
## [1] "A <--> 1"
## [1] "B <--> 2"
## [1] "C <--> 3"
## [1] "D <--> 4"
## [1] "E <--> 5"
## [1] "F <--> 6"
## [1] "G <--> 7"
## [1] "H <--> 8"
## [1] "I <--> 9"
## [1] "J <--> 10"
## [1] "K <--> 11"
## [1] "L <--> 12"
## [1] "M <--> 13"
## [1] "N <--> 14"
## [1] "O <--> 15"
## [1] "P <--> 16"
## [1] "Q <--> 17"
## [1] "R <--> 18"
## [1] "S <--> 19"
## [1] "T <--> 20"
```

Eso que vemos es el efecto secundario dentro de la función (imprimir)

7.1.8 Cuando usar estas herramientas?

A lo largo del curso vimos diferentes técnicas para manipulación de datos. En particular, la librería dplyr nos permitía fácilmente modificar y crear nuevas variables, agrupando. Cuando usamos `dplyr` y cuando usamos `purrr`.

- Si trabajamos sobre un DF simple, sin variables anidadas (lo que conocíamos hasta hoy) podemos usar `dplyr`
- Si queremos trabajar con DF anidados, con cosas que no son DF, o si el resultado de la operación que vamos a realizar a nivel file es algo distinto a un valor único, nos conviene usar `map` y `purrr`
- Las funciones `walk` son útiles por ejemplo para escribir archivos en disco de forma iterativa. Algo que no genera una salida

7.2 Práctica Guiada

```
library(fs)
library(tidyverse)
library(openxlsx)
library(glue)
```

7.2.1 Ejemplo 1: Iterando en la EPH

Lo primero que necesitamos es definir un vector o lista sobre el que iterar.

Por ejemplo, podemos armar un vector con los path a las bases individuales, con el comando `fs::dir_ls`

```
bases_individuales_path <- dir_ls(path = '../fuentes/', regexp= 'individual')
bases_individuales_path
```

```
## ../fuentes/usu_individual_t119.txt ../fuentes/usu_individual_t418.txt
```

Luego, como en la función que usamos para leer las bases definimos muchos parametros, nos podemos armar una función *wrapper* que sólo necesite un parámetro, y que simplifique la escritura del map

```
leer_base_eph <- function(path) {
  read.table(path,sep=";", dec=",", header = TRUE, fill = TRUE) %>%
    select(ANO4,TRIMESTRE,REGION,P21,CH04, CH06)
}

bases_df <- tibble(bases_individuales_path) %>%
  mutate(base = map(bases_individuales_path, leer_base_eph))
```

```
bases_df
```

```
## # A tibble: 2 x 2
##   bases_individuales_path            base
##   <fs::path>                         <list>
## 1 ../fuentes/usu_individual_t119.txt <df[,6] [59,369 x 6]>
## 2 ../fuentes/usu_individual_t418.txt <df[,6] [57,418 x 6]>
```

El resultado es un DF donde la columna **base** tiene en cada fila, otro DF con la base de la EPH de ese período. Esto es lo que llamamos un *nested DF* o dataframe nesteado pa les pibes.

Si queremos juntar todo, podemos usar `unnest()`

```
bases_df <- bases_df %>% unnest()
bases_df
```

```
## # A tibble: 116,787 x 7
##    bases_individuales_path             ANO4 TRIMESTRE REGION   P21  CH04
```

```
##    <fs::path>                          <int>     <int>  <int> <int> <int>
##  1 ../fuentes/usu_individual_t119.txt  2019         1     41     0     2
##  2 ../fuentes/usu_individual_t119.txt  2019         1     41     0     2
##  3 ../fuentes/usu_individual_t119.txt  2019         1     41     0     1
##  4 ../fuentes/usu_individual_t119.txt  2019         1     41  5000     2
##  5 ../fuentes/usu_individual_t119.txt  2019         1     41     0     2
##  6 ../fuentes/usu_individual_t119.txt  2019         1     41  8000     1
##  7 ../fuentes/usu_individual_t119.txt  2019         1     41     0     1
##  8 ../fuentes/usu_individual_t119.txt  2019         1     41     0     2
##  9 ../fuentes/usu_individual_t119.txt  2019         1     41     0     2
## 10 ../fuentes/usu_individual_t119.txt  2019         1     41  3000     1
## # ... with 116,777 more rows, and 1 more variable: CH06 <int>
```

> ¿Qué pasa si los DF que tenemos nesteados no tienen la misma cantidad de columnas?

Esto mismo lo podemos usar para fragmentar el datastet por alguna variable, con el `group_by()`

```
bases_df %>%
  group_by(REGION) %>%
  nest()
```

```
## # A tibble: 6 x 2
##   REGION data
##    <int> <list>
## 1     41 <tibble [11,509 x 6]>
## 2     44 <tibble [14,204 x 6]>
## 3     42 <tibble [11,150 x 6]>
## 4     43 <tibble [34,702 x 6]>
## 5     40 <tibble [24,432 x 6]>
## 6      1 <tibble [20,790 x 6]>
```

Así, para cada región tenemos un DF.

> ¿ De qué sirve todo esto?

No todo en la vida es un Dataframe. Hay estucturas de datos que no se pueden normalizar a filas y columnas. En esos casos recurríamos tradicionalmente a los loops. Con MAP podemos tener los elementos agrupados en un sólo objeto y aún conservar sus formas diferentes.

7.2.2 Ejemplo 2. Regresión lineal

Si bien no nos vamos a meter en el detalle del modelo lineal hoy, es útil usarlo como ejemplo de lo que podemos hacer con MAP.

Planteamos el modelo

$$P21 = \beta_0 + \beta_1 * CH04 + \beta_2 * CH06$$

Osea, un modleo que explica el ingreso según sexo y edad

```
lmfit <- lm(P21~factor(CH04)+CH06,data = bases_df)

summary(lmfit)
```

```
##
## Call:
## lm(formula = P21 ~ factor(CH04) + CH06, data = bases_df)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -15472  -6606  -3367   2148 590198
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    4853.196     74.509   65.14   <2e-16 ***
## factor(CH04)2 -4063.112     72.200  -56.27   <2e-16 ***
## CH06            103.095      1.612   63.97   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12300 on 116784 degrees of freedom
## Multiple R-squared:  0.05511,    Adjusted R-squared:  0.0551
## F-statistic:  3406 on 2 and 116784 DF,  p-value: < 2.2e-16
```

(al final de la clase podemos charlar sobre los resultados, si hay interés :-))

De forma Tidy, la librería `broom` nos da los resultados en un DF.

```
broom::tidy(lmfit)
```

```
## # A tibble: 3 x 5
##   term          estimate std.error statistic p.value
##   <chr>            <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept)      4853.     74.5       65.1        0
## 2 factor(CH04)2   -4063.     72.2      -56.3        0
## 3 CH06              103.      1.61      64.0        0
```

Si lo queremos hacer por region

7.2.2.1 Loopeando

```
resultados <- tibble()

for (region in unique(bases_df$REGION)) {

  data <- bases_df %>%
```

```
    filter(REGION==region)

  lmfit <- lm(P21~factor(CH04)+CH06,data = data)

  lmtidy <- broom::tidy(lmfit)
  lmtidy$region <- region
  resultados <- bind_rows(resultados,lmtidy)

}

resultados
```

```
## # A tibble: 18 x 6
##    term          estimate std.error statistic   p.value region
##    <chr>            <dbl>     <dbl>     <dbl>     <dbl>  <int>
##  1 (Intercept)     3768.     185.        20.3 3.15e- 90     41
##  2 factor(CH04)2  -3814.     180.       -21.2 6.00e- 98     41
##  3 CH06             106.       4.18      25.3 1.12e-137     41
##  4 (Intercept)     7156.     291.        24.6 1.09e-130     44
##  5 factor(CH04)2  -5938.     278.       -21.4 1.42e- 99     44
##  6 CH06             145.       6.32      23.0 1.40e-114     44
##  7 (Intercept)     4930.     231.        21.4 2.15e- 99     42
##  8 factor(CH04)2  -4007.     224.       -17.9 1.71e- 70     42
##  9 CH06               97.8      4.95      19.7 2.68e- 85     42
## 10 (Intercept)     5107.     131.        39.0 0.            43
## 11 factor(CH04)2  -3949.     127.       -31.1 5.02e-209     43
## 12 CH06              83.5      2.78      30.0 3.87e-195     43
## 13 (Intercept)     3329.     128.        26.0 4.12e-147     40
## 14 factor(CH04)2  -3239.     125.       -25.9 3.74e-146     40
## 15 CH06             122.       2.89      42.2 0.            40
## 16 (Intercept)     5196.     197.        26.4 3.45e-151      1
## 17 factor(CH04)2  -4051.     189.       -21.4 1.80e-100      1
## 18 CH06             88.2      4.12      21.4 1.98e-100      1
```

7.2.2.2 Usando MAP

Primero me armo una funcion que me simplifica el codigo

```
fun<-function(porcion,grupo) {  broom::tidy(lm(P21~factor(CH04)+CH06,data = porcion))}
```

```
bases_df_lm <- bases_df %>%
  group_by(REGION) %>%
  nest() %>%
  mutate(lm = map(data,fun))
bases_df_lm
```

```
## # A tibble: 6 x 3
```

```
##   REGION data                  lm
##    <int> <list>                <list>
## 1     41 <tibble [11,509 x 6]> <tibble [3 x 5]>
## 2     44 <tibble [14,204 x 6]> <tibble [3 x 5]>
## 3     42 <tibble [11,150 x 6]> <tibble [3 x 5]>
## 4     43 <tibble [34,702 x 6]> <tibble [3 x 5]>
## 5     40 <tibble [24,432 x 6]> <tibble [3 x 5]>
## 6      1 <tibble [20,790 x 6]> <tibble [3 x 5]>
```

```
bases_df_lm %>%
  unnest(lm)
```

```
## # A tibble: 18 x 6
##    REGION term          estimate std.error statistic   p.value
##     <int> <chr>            <dbl>     <dbl>     <dbl>     <dbl>
##  1     41 (Intercept)     3768.     185.        20.3 3.15e- 90
##  2     41 factor(CH04)2  -3814.     180.       -21.2 6.00e- 98
##  3     41 CH06             106.       4.18      25.3 1.12e-137
##  4     44 (Intercept)     7156.     291.        24.6 1.09e-130
##  5     44 factor(CH04)2  -5938.     278.       -21.4 1.42e- 99
##  6     44 CH06             145.       6.32      23.0 1.40e-114
##  7     42 (Intercept)     4930.     231.        21.4 2.15e- 99
##  8     42 factor(CH04)2  -4007.     224.       -17.9 1.71e- 70
##  9     42 CH06              97.8      4.95      19.7 2.68e- 85
## 10     43 (Intercept)     5107.     131.        39.0 0.
## 11     43 factor(CH04)2  -3949.     127.       -31.1 5.02e-209
## 12     43 CH06             83.5      2.78      30.0 3.87e-195
## 13     40 (Intercept)     3329.     128.        26.0 4.12e-147
## 14     40 factor(CH04)2  -3239.     125.       -25.9 3.74e-146
## 15     40 CH06             122.       2.89      42.2 0.
## 16      1 (Intercept)     5196.     197.        26.4 3.45e-151
## 17      1 factor(CH04)2  -4051.     189.       -21.4 1.80e-100
## 18      1 CH06              88.2      4.12      21.4 1.98e-100
```

O incluso más facil, utilizando `group_modify` (que es un atajo que solo acepta DF)

```
bases_df %>%
  group_by(REGION) %>%
  group_modify(fun)
```

```
## # A tibble: 18 x 6
## # Groups:   REGION [6]
##    REGION term          estimate std.error statistic   p.value
##     <int> <chr>            <dbl>     <dbl>     <dbl>     <dbl>
##  1      1 (Intercept)     5196.     197.        26.4 3.45e-151
##  2      1 factor(CH04)2  -4051.     189.       -21.4 1.80e-100
##  3      1 CH06              88.2      4.12      21.4 1.98e-100
```

```
##  4     40 (Intercept)     3329.     128.        26.0 4.12e-147
##  5     40 factor(CH04)2  -3239.     125.       -25.9 3.74e-146
##  6     40 CH06             122.       2.89      42.2 0.
##  7     41 (Intercept)     3768.     185.        20.3 3.15e- 90
##  8     41 factor(CH04)2  -3814.     180.       -21.2 6.00e- 98
##  9     41 CH06             106.       4.18      25.3 1.12e-137
## 10     42 (Intercept)     4930.     231.        21.4 2.15e- 99
## 11     42 factor(CH04)2  -4007.     224.       -17.9 1.71e- 70
## 12     42 CH06             97.8      4.95      19.7 2.68e- 85
## 13     43 (Intercept)     5107.     131.        39.0 0.
## 14     43 factor(CH04)2  -3949.     127.       -31.1 5.02e-209
## 15     43 CH06             83.5      2.78      30.0 3.87e-195
## 16     44 (Intercept)     7156.     291.        24.6 1.09e-130
## 17     44 factor(CH04)2  -5938.     278.       -21.4 1.42e- 99
## 18     44 CH06             145.       6.32      23.0 1.40e-114
```

Pero MAP sirve para operar con cualquier objeto de R.

Por ejemplo podemos guardar el **objeto** `S3:lm` que es la regresion lineal entrenada. Ese objeto no es ni un vector, ni una lista, ni un DF. No es una estructura de datos, sino que es algo distinto, con *propiedades* como `predict()` para predecir, el `summary()` que vimos, etc.

```
fun<-function(porcion,grupo) {  lm(P21~factor(CH04)+CH06,data = porcion)}

bases_df %>%
  group_by(REGION) %>%
  nest() %>%
  mutate(lm = map(data,fun))
```

```
## # A tibble: 6 x 3
##   REGION data                lm
##    <int> <list>              <list>
## 1     41 <tibble [11,509 x 6]> <lm>
## 2     44 <tibble [14,204 x 6]> <lm>
## 3     42 <tibble [11,150 x 6]> <lm>
## 4     43 <tibble [34,702 x 6]> <lm>
## 5     40 <tibble [24,432 x 6]> <lm>
## 6      1 <tibble [20,790 x 6]> <lm>
```

7.2.3 Ejemplo 3: Gráficos en serie

Veamos un tercer ejemplo con otra base de datos que ya conocemos: Gapminder, que muestra algunos datos sobre la población de los países por año.

El objetivo de este ejercicio es hacer un gráfico por país de forma automática.

- Primero veamos los datos

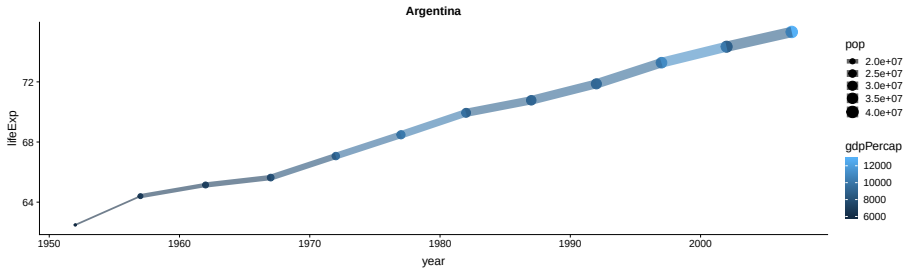
```
library(gapminder)


gapminder_unfiltered %>%
  sample_n(10)
```

```
## # A tibble: 10 x 6
##    country          continent  year lifeExp      pop gdpPercap
##    <fct>            <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Norway           Europe     1964    73.6  3694339    14440.
##  2 Togo             Africa     1962    43.9  1528098     1068.
##  3 Austria          Europe     1970    70.1  7467086    15079.
##  4 Congo, Dem. Rep. Africa     1957    40.7 15577932      906.
##  5 Qatar            Asia       1972    62.1   131794    81069.
##  6 Bulgaria         Europe     1986    71.6  8958770     8236.
##  7 Poland           Europe     1963    68.6 30662122     5597.
##  8 Zimbabwe         Africa     1957    50.5  3646340      519.
##  9 Uganda           Africa     2007    51.5 29170398     1056.
## 10 Mongolia         Asia       2007    66.8  2874127     3096.
```

la base tiene la siguiente info:

- country: Nombre del país
- continent: Nombre del continente
- year: año
- lifeExp: Esperanza de vida al nacer
- pop: Población
- gdpPercap
- Vamos a hacer un gráfico sencillo para Argentina

```
data_argentina <- gapminder_unfiltered %>%
  filter(country=='Argentina')

ggplot(data_argentina, aes(year, lifeExp, size= pop, color=gdpPercap))+
  geom_point()+
  geom_line(alpha=0.6)+
  labs(title = unique(data_argentina$country))
```

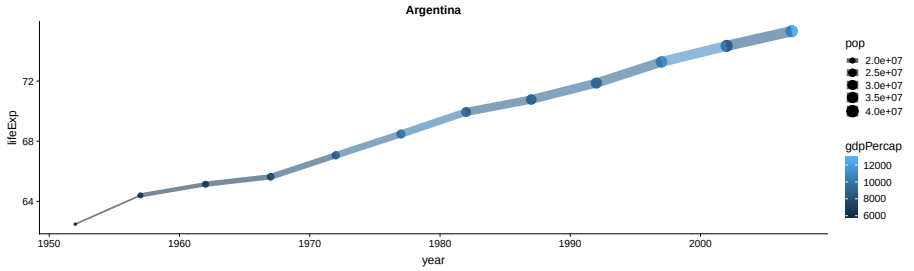


- Ahora que tenemos una idea de lo que queremos gráficar lo podemos poner adentro de una función que grafique.

```
# definimos la función
graficar_pais <- function(data, pais){

  ggplot(data, aes(year, lifeExp, size= pop, color=gdpPercap))+
    geom_point()+
    geom_line(alpha=0.6)+
    labs(title = pais)
}
```

probamos la función para un caso

```
graficar_pais(data_argentina, 'Argentina')
```



- Nos armamos un dataset nesteado

```
gapminder_nest <- gapminder_unfiltered %>%
  group_by(country) %>%
  nest()

gapminder_nest %>%
  sample_n(10)
```

```
## # A tibble: 10 x 2
##    country              data
##    <fct>                <list>
##  1 Puerto Rico          <tibble [13 x 5]>
```

```
##  2 Korea, Rep.          <tibble [12 x 5]>
##  3 Samoa                <tibble [7 x 5]>
##  4 Afghanistan          <tibble [12 x 5]>
##  5 Malaysia             <tibble [12 x 5]>
##  6 Italy                <tibble [56 x 5]>
##  7 French Polynesia     <tibble [9 x 5]>
##  8 Slovenia             <tibble [32 x 5]>
##  9 United Arab Emirates <tibble [8 x 5]>
## 10 Namibia              <tibble [12 x 5]>
```

- Ahora podemos crear una nueva columna que contenga los gráficos

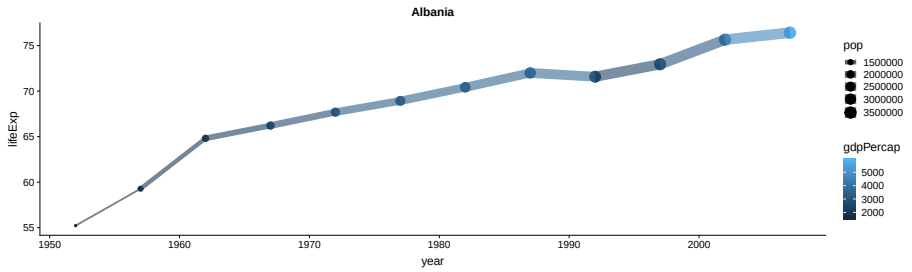
```
gapminder_nest <- gapminder_nest %>%
  mutate(grafico= map2(.x = data, .y = country,.f =  graficar_pais))

gapminder_nest %>%
  sample_n(10)
```

```
## # A tibble: 10 x 3
##    country       data              grafico
##    <fct>         <list>            <list>
##  1 Cyprus        <tibble [8 x 5]>  <gg>
##  2 Somalia       <tibble [12 x 5]> <gg>
##  3 Poland        <tibble [52 x 5]> <gg>
##  4 Uzbekistan    <tibble [4 x 5]>  <gg>
##  5 Ecuador       <tibble [12 x 5]> <gg>
##  6 Sweden        <tibble [58 x 5]> <gg>
##  7 Kenya         <tibble [12 x 5]> <gg>
##  8 Ethiopia      <tibble [12 x 5]> <gg>
##  9 Guinea-Bissau <tibble [12 x 5]> <gg>
## 10 Portugal      <tibble [58 x 5]> <gg>
```

Veamos un ejemplo

```
gapminder_nest$grafico[2]
```

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



Ahora podemos guardar todos los gráficos en un archivo PDF

```
pdf('../resultados/graficos_gapminder.pdf')
gapminder_nest$grafico
dev.off()
```