

Taller Tidyverse

Néstor Montaña

Sociedad Ecuatoriana de Estadística

Octubre 2023

**Nota:**

Con *Alt + F* o *Option + F* puede hacer que estas diapositivas ocupen todo el navegador (es decir que se ignore el aspecto de diapositiva que tiene por default la presentación)

Diapositivas y Set de Datos

https://github.com/nestormontano/2023_DS_Month__Taller_Tidyverse

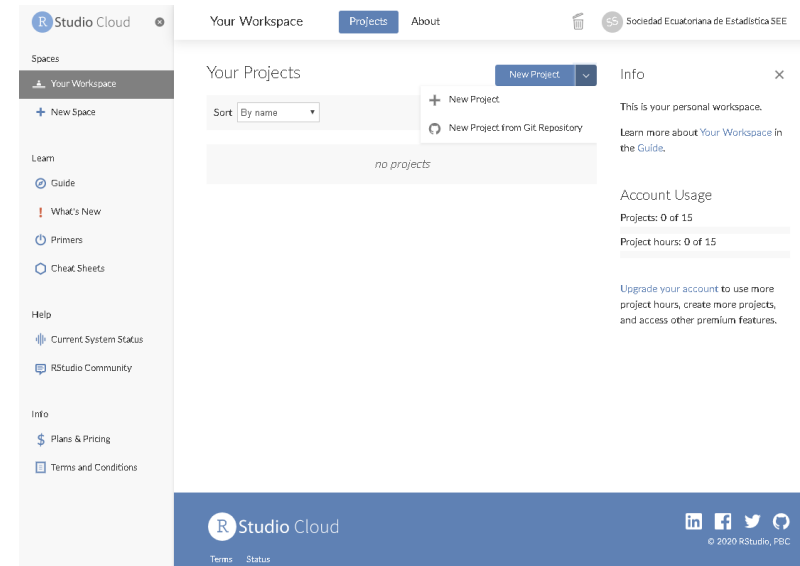
Introducción a R y RStudio

Taller Tidyverse

Néstor Montaña P.

RStudio

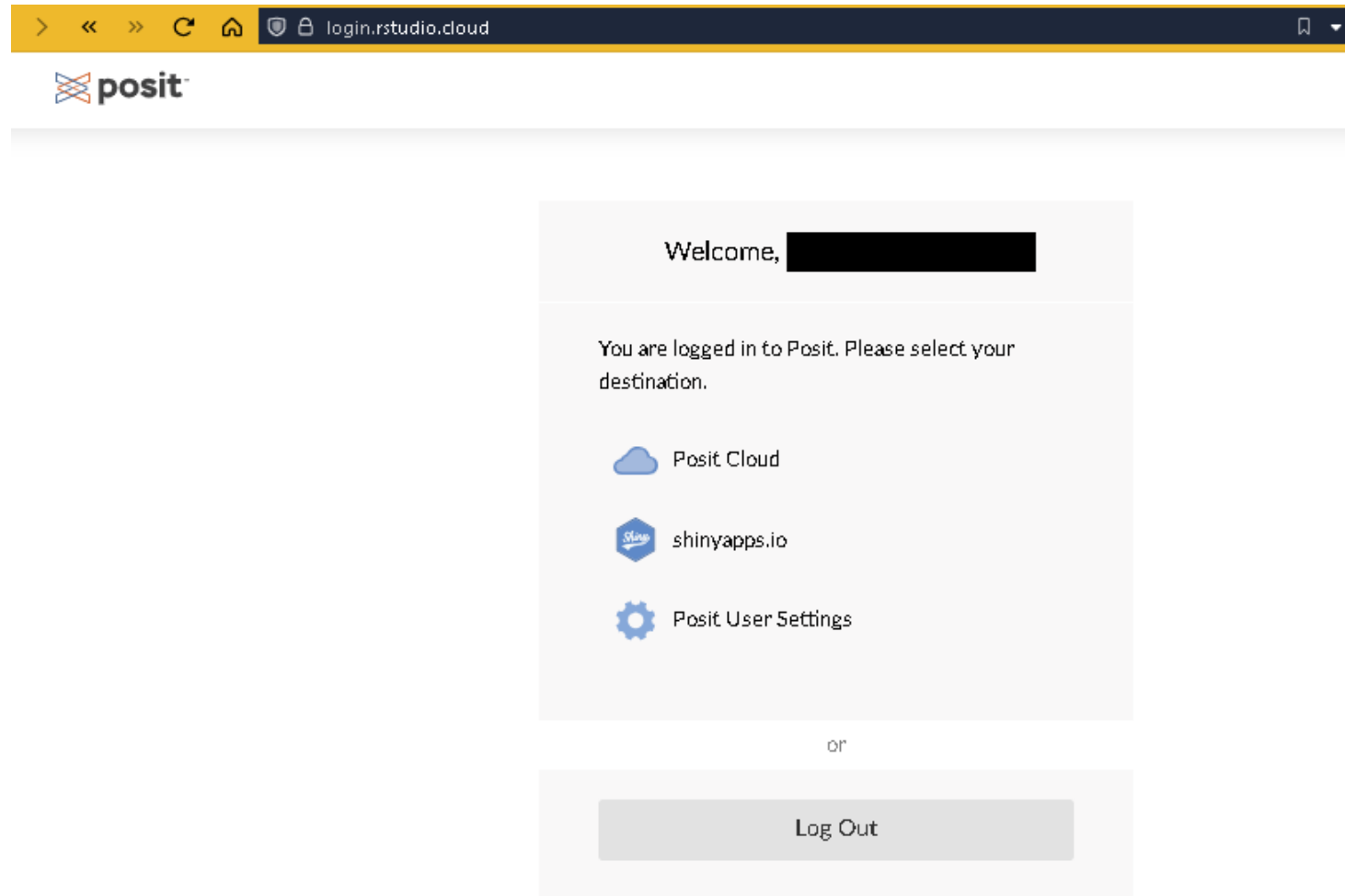
Para interactuar con R, usaremos RStudio; los que no lo tengan instalado pueden usar RStudio Cloud desde <https://login.rstudio.cloud>



RStudio Cloud

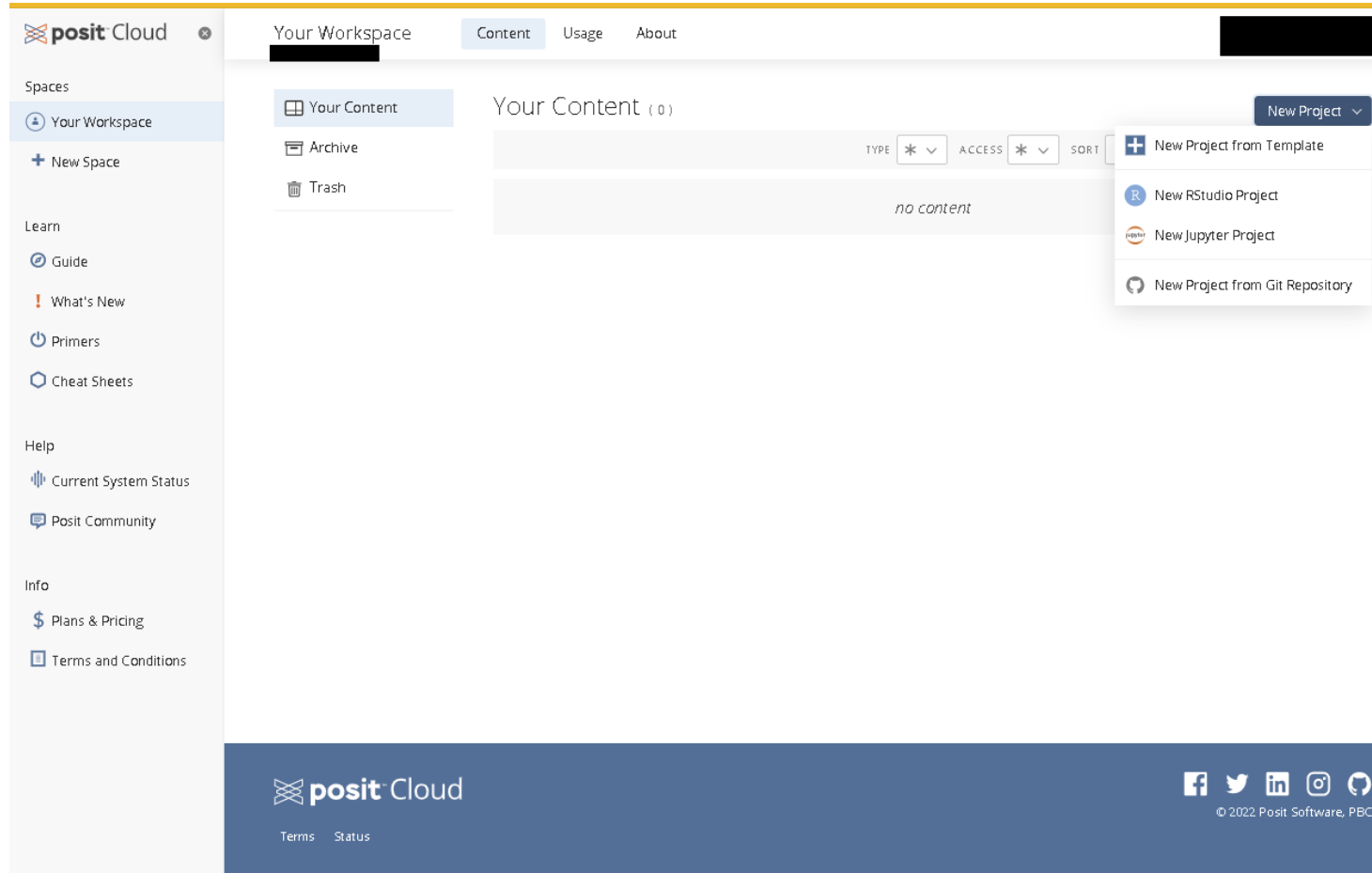
RStudio Cloud

Iniciar sesion en Posit Cloud



RStudio Cloud

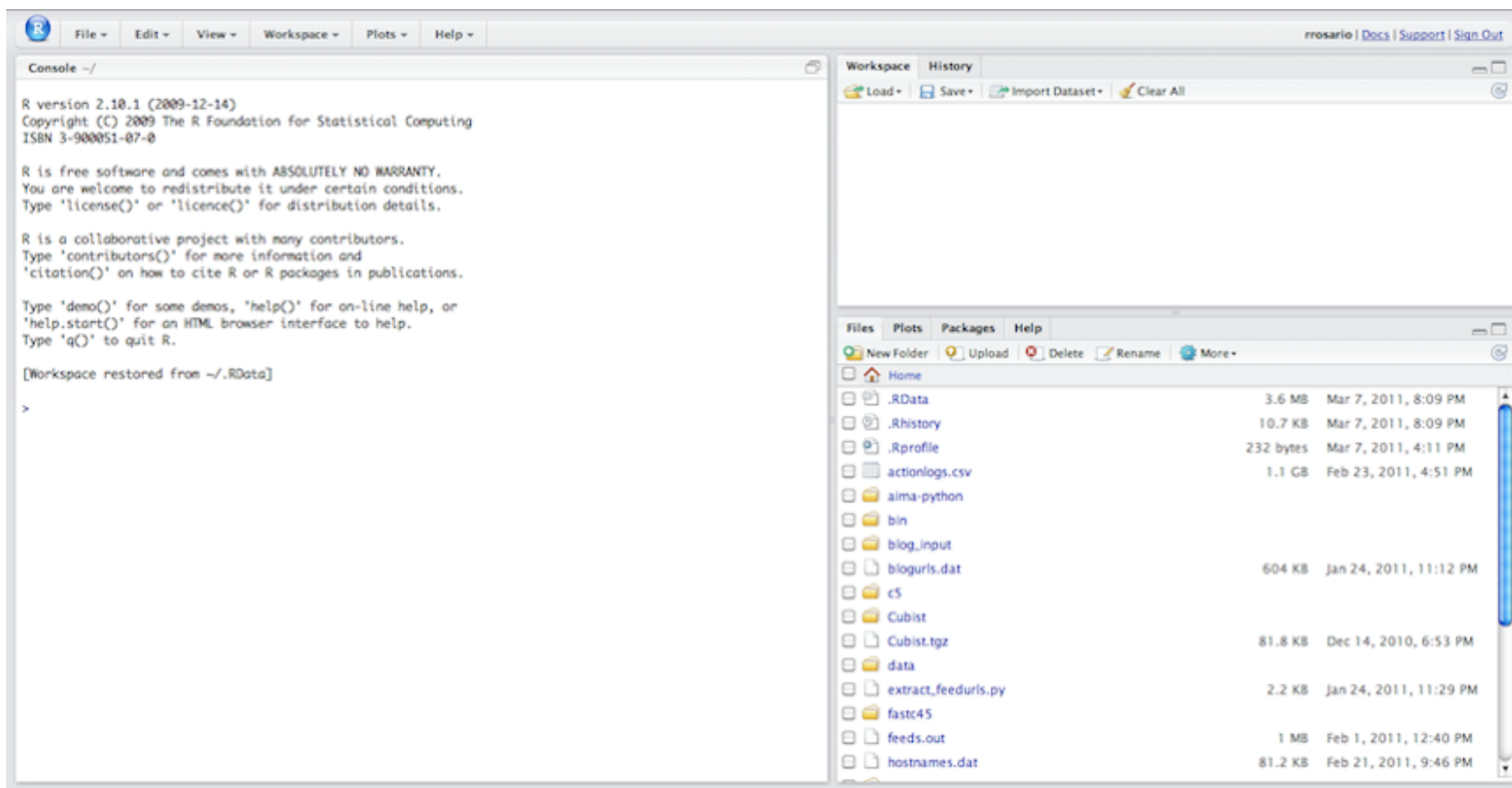
Iniciar un Proyecto en RStudio Cloud



The screenshot displays the RStudio Cloud web interface. On the left is a sidebar with navigation links under 'Spaces' (Your Workspace, New Space), 'Learn' (Guide, What's New, Primers, Cheat Sheets), 'Help' (Current System Status, Posit Community), and 'Info' (Plans & Pricing, Terms and Conditions). The main area is titled 'Your Workspace' and contains a 'Content' tab. Under 'Your Content', there are links for 'Your Content', 'Archive', and 'Trash'. A 'New Project' button is visible in the top right of the content area, which has opened a dropdown menu with the following options: 'New Project from Template', 'New RStudio Project', 'New Jupyter Project', and 'New Project from Git Repository'. The bottom of the interface features a dark blue footer with the Posit Cloud logo, social media icons, and copyright information: '© 2022 Posit Software, PBC'.

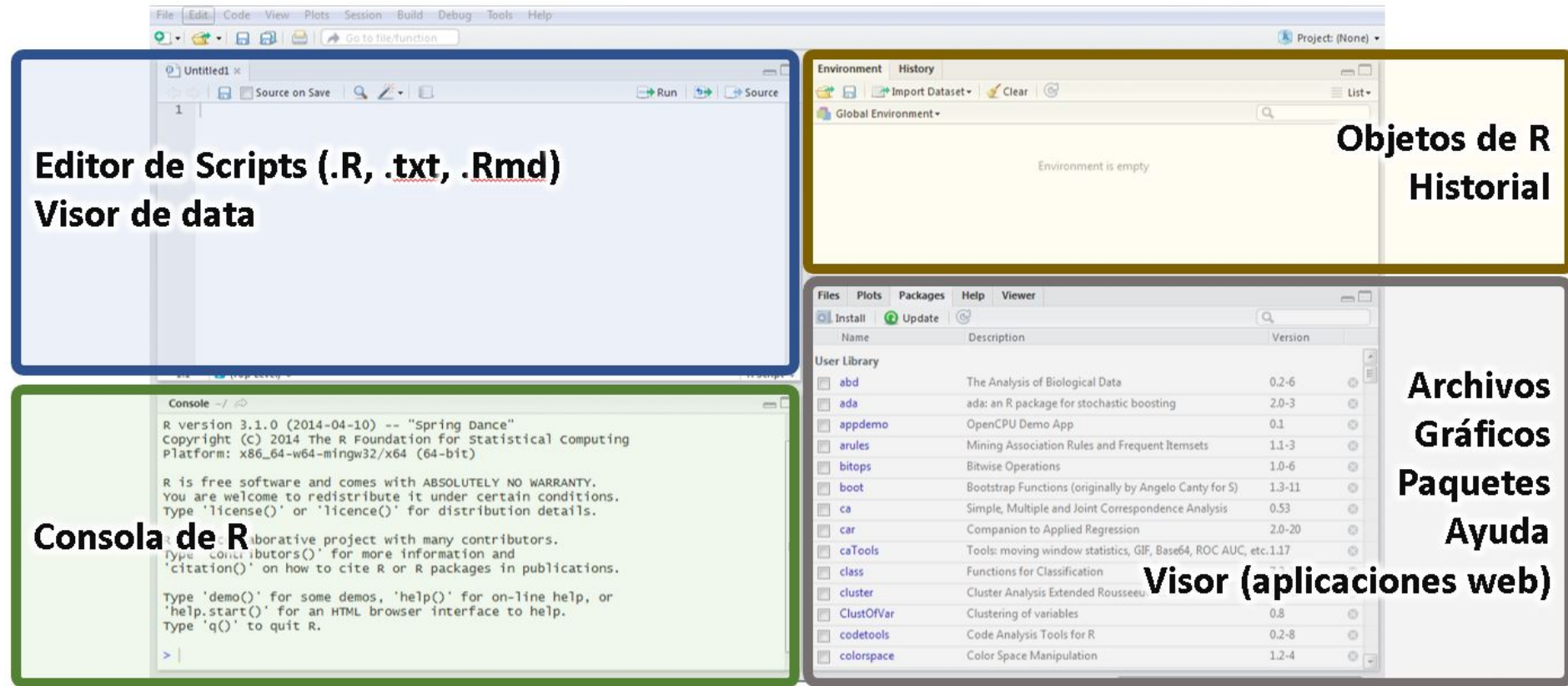
RStudio

Pantalla inicial de RStudio



RStudio

Interfaz de RStudio se divide en 4 paneles:



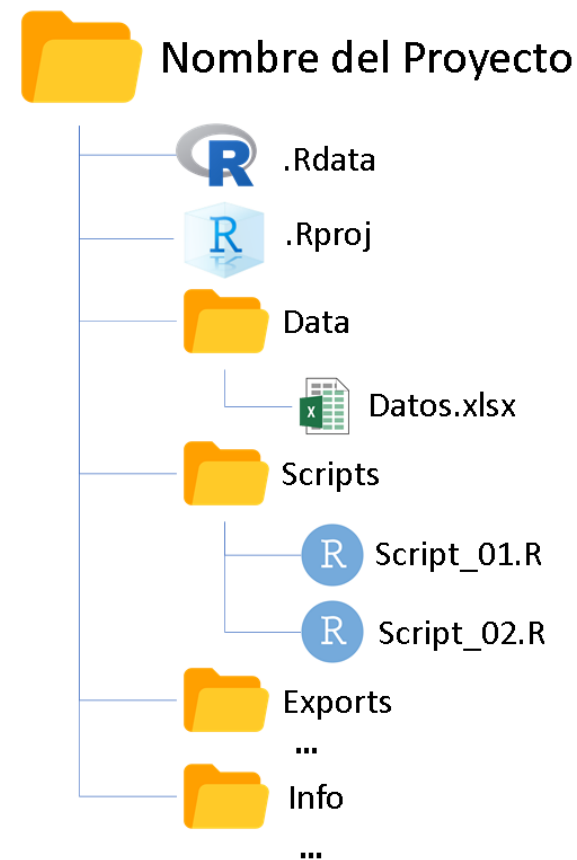
The screenshot shows the RStudio interface with four panels highlighted by colored boxes and labels:

- Editor de Scripts (.R, .txt, .Rmd)** (Blue box): The top-left panel, labeled "Untitled1", contains a code editor with a single line of code: `1`. It includes a toolbar with icons for saving, running, and other functions.
- Objetos de R** (Yellow box): The top-right panel, labeled "Environment", shows the "Global Environment" with the message "Environment is empty".
- Consola de R** (Green box): The bottom-left panel, labeled "Console", displays the R startup message: "R version 3.1.0 (2014-04-10) -- 'Spring Dance' Copyright (C) 2014 The R Foundation for Statistical Computing Platform: x86_64-w64-mingw32/x64 (64-bit)". It also includes instructions on how to use R, such as "Type 'demo()' for some demos, 'help()' for on-line help, or 'help.start()' for an HTML browser interface to help. Type 'q()' to quit R."
- Archivos Gráficos Paquetes Ayuda** (Grey box): The bottom-right panel, labeled "Files", shows a list of installed packages in the "User Library". The list includes columns for Name, Description, and Version. The packages listed are: `abd` (The Analysis of Biological Data, 0.2-6), `ada` (ada: an R package for stochastic boosting, 2.0-3), `appdemo` (OpenCPU Demo App, 0.1), `arules` (Mining Association Rules and Frequent Itemsets, 1.1-3), `bitops` (Bitwise Operations, 1.0-6), `boot` (Bootstrap Functions (originally by Angelo Canty for S), 1.3-11), `ca` (Simple, Multiple and Joint Correspondence Analysis, 0.53), `car` (Companion to Applied Regression, 2.0-20), `caTools` (Tools: moving window statistics, GIF, Base64, ROC AUC, etc., 1.17), `class` (Functions for Classification, 7.3-15), `cluster` (Cluster Analysis Extended Rousseeu, 2.0.6), `ClustOfVar` (Clustering of variables, 0.8), `codetools` (Code Analysis Tools for R, 0.2-8), and `colorspace` (Color Space Manipulation, 1.2-4).

Proyectos en R, RStudio

- Un **Proyecto** es una carpeta que contiene todos los scripts, los archivos desde donde se importan los datos y los archivos de proyecto como el .Rdata (que contiene los objetos con los que se está trabajando) y .Rhistory (que contiene la historia de comandos ejecutados)
- Permite tener nuestros análisis ordenados,
- Al abrir un proyecto antiguo RStudio lo abre con las pestañas que se tenía activas,
- Permite colaboración utilizando GIT o Subversion,
- Se sugiere tener una estructura interior, por ejemplo:
 - Scripts, Data, Exports, Info
- En R podemos hacer:

```
dir.create('Scripts')  
dir.create('Data')  
dir.create('Exports')
```





Paquetes a usar

Instalar los paquetes (esto se ejecuta una sola vez)

```
install.packages("openxlsx") # Para importar/exportar excel
install.packages("magrittr") # pipe
install.packages("tidyverse") # Manipulación de datos y ggplot2
```

Activar los paquetes (esto se ejecutará cada vez que se abra el proyecto o R)

```
# Cargar la libreria a utilizar
library(openxlsx)
library(magrittr)
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.0.5
```

```
## -- Attaching packages ----- tidyverse 1.3.1 --
```

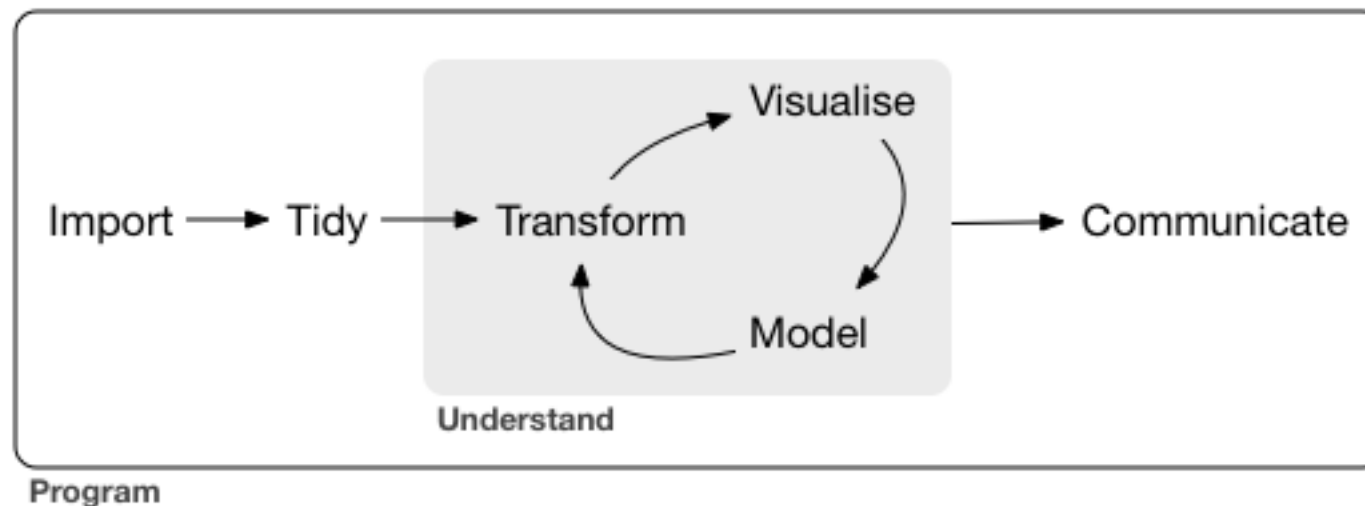
```
## v ggplot2 3.3.3      v purrr    0.3.4
## v tibble  3.1.3      v dplyr    1.0.7
## v tidyr   1.1.3      v stringr  1.4.0
## v readr   1.4.0      v forcats  0.5.1
```

Desarrollo de un caso de análisis de datos

Taller Tidyverse

Néstor Montaña P.

Workflow de un análisis estadístico



- Import: Obtener y entender los datos
- Tidy: Ordenar los datos de tal manera que sea sencillo transformarlos, sumarlos, visualizarlos o realizar un modelo con ellos
- Transform: Manipular los datos hasta obtener el input que el análisis o técnica estadística necesita
- Visualise: Realizar el análisis exploratorio de datos
- Model: Aplicar técnicas estadísticas para el entendimiento del problema o tomar decisiones
- Communicate: Tratar de mostrar los resultados de tal forma que el resto del mundo los entienda, usando reportes, gráficos, visualizaciones interactivas, integración con herramientas de BI, web apps, etc.



Ejemplo: Data de transacciones bancarias

El Banco del Pacífico requiere mejorar los tiempos de atención al cliente en ventanilla, para ello ha recolectado esta información anónimamente para cada cajero y transacción realizada.

Le suministran un excel con dos hojas:

1. Tiene los datos de las transacciones, columnas: Sucursal, Cajero, ID_Transaccion, Transaccion, Tiempo_Servicio_seg, Nivel de satisfacción, Monto de la transaccion.
2. Otra hoja que indica si en la sucursal se ha puesto o no el nuevo sistema.
3. Datos demográficos de los cajeros



Importar desde excel

- Copiando desde un archivo de excel abierto
`read.table("clipboard", sep="\t", header=TRUE)`
- Desde RStudio
Rstudio > Import Dataset > From Excel > Escoger archivo > Abrir > Escribir nombre a la variable > Import
- Usando el paquete `openxlsx`
`read.xlsx(xlsxFile , sheet , startRow , colNames , skipEmptyRows, rowNames)`
`dataframe <- read.xlsx(xlsxFile = 'Data/Data_Banco.xlsx')`
- Usando el paquete `readxl`
`dataframe <- read_excel('Data/Data_Banco.xlsx')`
- Otros paquetes
`excel.link`, `XLConnect`, `xlsx`, `rio`



Ejemplo - Importar

```
# Leer el archivo de excel y asignarlo al objeto data_banco  
data_banco <- read.xlsx(xlsxFile = "Data/Data_Banco.xlsx", sheet = "Data")  
data_sucursal <- read.xlsx(xlsxFile = "Data/Data_Banco.xlsx", sheet = "Data_Sucursal")
```


Ejemplo - Importar

Bien, se han creado dos objetos en nuestro 'environment', **¿qué tipo de estructura hemos importado?**

R. Un data.frame

- Se puede ver la estructura de un objeto con `str()`

```
str(data_banco)
```

```
## 'data.frame':    24299 obs. of  7 variables:
## $ Sucursal      : num  62 62 62 62 62 62 62 62 62 62 ...
## $ Cajero        : num  4820 4820 4820 4820 4820 4820 4820 4820 4820 4820 ...
## $ ID_Transaccion : chr   "2" "2" "2" "2" ...
## $ Transaccion    : chr   "Cobro/Pago (Cta externa)" "Cobro/Pago (Cta externa)" "Cobro/Pago (Cta externa)" ...
## $ Tiempo_Servicio_seg: num  622 329 643 422 378 487 384 910 694 650 ...
## $ Satisfaccion   : chr   "Muy Bueno" "Malo" "Regular" "Regular" ...
## $ Monto          : chr   "2889,3" "1670,69" "3172,49" "1764.92" ...
```



Estructuras de datos | Objetos

R cuenta con un sinnúmero de estructuras de datos (clases de objetos), los más usados son:

- Vector
- Matriz
- Data.frame
- Litas
- Serie de Tiempo
- Data.table



Vectores

- En R no existen escalares, sino vectores de $\text{dim} = 1$

```
x <- 1  
is.vector(x)
```

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

- Los vectores se crean con la función `c()`

```
x <- c(11, 12, 13, 14) # crea x  
x # presenta x
```

```
## [1] 11 12 13 14
```

- Ver la estructura interna con `str()`

```
str(x) # presenta x
```

```
##  num [1:4] 11 12 13 14
```



Data.frames

- Data.frame es una lista de vectores, cumple:
 - Las componentes son vectores
 - Cada vector puede ser de un tipo de dato distinto
 - Cada elemento, columna es una variable
 - Las columnas tienen el mismo largo
- Se podría decir que un data.frame es como una tabla en una hoja de excel

Data.frames

Visualizar el dataframe (primeras 4 filas)

```
head(data_banco)
```

##	Sucursal	Cajero	ID_Transaccion	Transaccion	Tiempo_Servicio_seg
## 1	62	4820	2 Cobro/Pago	(Cta externa)	622
## 2	62	4820	2 Cobro/Pago	(Cta externa)	329
## 3	62	4820	2 Cobro/Pago	(Cta externa)	643
## 4	62	4820	2 Cobro/Pago	(Cta externa)	422
## 5	62	4820	2 Cobro/Pago	(Cta externa)	378
## 6	62	4820	2 Cobro/Pago	(Cta externa)	487

##	Satisfaccion	Monto
## 1	Muy Bueno	2889,3
## 2	Malo	1670,69
## 3	Regular	3172,49
## 4	Regular	1764.92
## 5	Muy Bueno	1835.69
## 6	Bueno	2165.42

Estructuras de datos | Objetos

VECTOR

<i>names</i>			
[1]	[2]	[...]	[n]

SERIE DE TIEMPO

tiempo	$t1$	$t2$	tn	
data	[1]	[2]	[...]	[n]

MATRIZ

<i>rownames</i>	<i>colnames</i>			
	[1,1]	[1,2]	[...]	[1,m]
	[2,1]	[2,2]	[...]	[2,m]
	[...]	[...]	[...]	[...]
	[n,1]	[n,2]	[...]	[n,m]

DATAFRAME

		names		
rownames	Var_1	Var_2		Var_n
	[1,1]	[1,2]	[...]	[1,m]
	[2,1]	[2,2]	[...]	[2,m]
	[...]	[...]	[...]	[...]
	[n,1]	[n,2]	[...]	[n,m]

LISTA

[1,1] [1,2]		[1]	[1,1] [1,2] [...]		[1,m]
[2,1] [2,2]		[2]	[2,1] [2,2] [...]		[2,m]
[3,1] [3,2]			[...]		[...]



Entender los datos

Luego de importar se debe entender los datos

- ¿Qué representa cada columna?
- ¿Qué tipo de dato debería tener cada columna?
- ¿Qué granularidad o atomicidad tiene la data?
- Si es que se tiene varios conjuntos de datos ¿Cómo se relacionan los datos?
- A qué periodo de tiempo corresponde la data
- Muchas veces se obtiene la información desde una base de datos y por tanto toca entender la base y el query que genera los datos

Ejemplo - Entender los datos

Podríamos ver las primeras filas

```
# ver las primeras 5 filas  
head(data_sucursal, n = 5)
```

##	ID_Sucursal	Sucursal	Nuevo_Sistema
## 1	62	Riocentro Sur	No
## 2	85	Centro	Si
## 3	267	Alborada	Si
## 4	443	Mall del Sol	Si
## 5	586	Via Daule	No

Ejemplo - Entender los datos

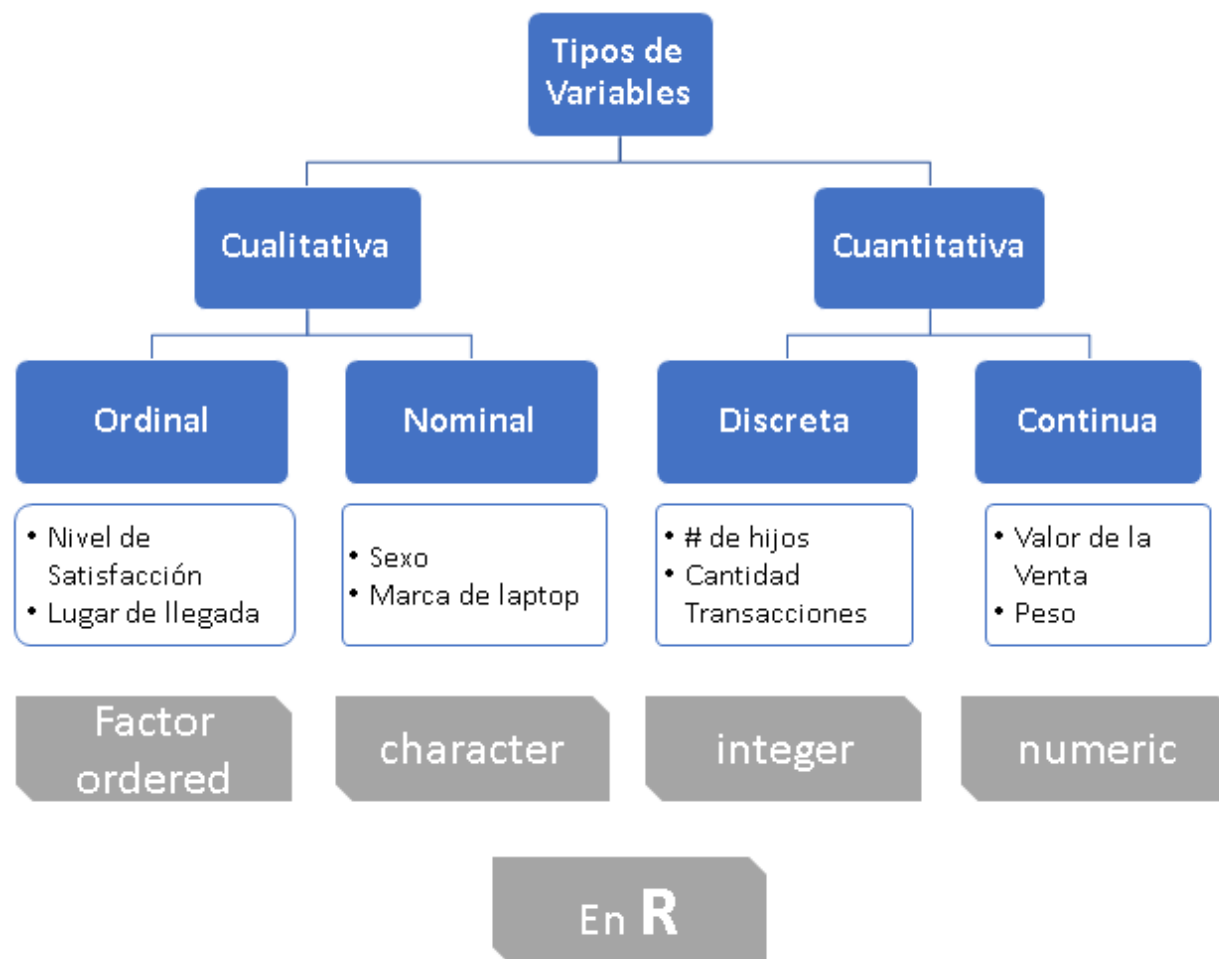
`glimpse` permite dar un vistazo rápido al dataframe, nos da los nombres de las columnas, los tipos de datos y una muestra de sus valores

```
glimpse(data_banco)
```

```
## Rows: 24,299
## Columns: 7
## $ Sucursal      <dbl> 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62~
## $ Cajero        <dbl> 4820, 4820, 4820, 4820, 4820, 4820, 4820, 4820, 48~
## $ ID_Transaccion <chr> "2", "2", "2", "2", "2", "2", "2", "2", "2", "2", ~
## $ Transaccion    <chr> "Cobro/Pago (Cta externa)", "Cobro/Pago (Cta exter~
## $ Tiempo_Servicio_seg <dbl> 622, 329, 643, 422, 378, 487, 384, 910, 694, 650, ~
## $ Satisfaccion   <chr> "Muy Bueno", "Malo", "Regular", "Regular", "Muy Bu~
## $ Monto          <chr> "2889,3", "1670,69", "3172,49", "1764.92", "1835.6~
```

Tipos de variables

Tipos de variables y su correspondencia en R



Entender los datos - Ejemplo

¿Están bien nuestros tipos de datos?

...

```
# Ver la estructura del data.frame  
glimpse(data_banco)
```

```
## Rows: 24,299  
## Columns: 7  
## $ Sucursal      <dbl> 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62~  
## $ Cajero        <dbl> 4820, 4820, 4820, 4820, 4820, 4820, 4820, 4820, 48~  
## $ ID_Transaccion <chr> "2", "2", "2", "2", "2", "2", "2", "2", "2", "2", ~  
## $ Transaccion   <chr> "Cobro/Pago (Cta externa)", "Cobro/Pago (Cta exter~  
## $ Tiempo_Servicio_seg <dbl> 622, 329, 643, 422, 378, 487, 384, 910, 694, 650, ~  
## $ Satisfaccion  <chr> "Muy Bueno", "Malo", "Regular", "Regular", "Muy Bu~  
## $ Monto         <chr> "2889,3", "1670,69", "3172,49", "1764.92", "1835.6~
```

Entender los datos - Ejemplo

¿Está bien nuestros tipos de datos?

Si no lo están entonces debemos transformarlos, para esto aprenderemos sobre manipulación de datos.

```
# Ver la estructura del data.frame  
glimpse(data_banco)
```

```
## Rows: 24,299  
## Columns: 7  
## $ Sucursal      <dbl> 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62~  
## $ Cajero        <dbl> 4820, 4820, 4820, 4820, 4820, 4820, 4820, 4820, 48~  
## $ ID_Transaccion <chr> "2", "2", "2", "2", "2", "2", "2", "2", "2", "2", ~  
## $ Transaccion   <chr> "Cobro/Pago (Cta externa)", "Cobro/Pago (Cta exter~  
## $ Tiempo_Servicio_seg <dbl> 622, 329, 643, 422, 378, 487, 384, 910, 694, 650, ~  
## $ Satisfaccion  <chr> "Muy Bueno", "Malo", "Regular", "Regular", "Muy Bu~  
## $ Monto         <chr> "2889,3", "1670,69", "3172,49", "1764.92", "1835.6~
```

Manipulacion de datos - Basico

Taller Tidyverse

Néstor Montaña P.

Manipulacion de datos

R tiene sus comandos predeterminados para manipular datos, esto se conoce como *R Base*, sin embargo existen varios paquetes que simplifican esta tarea, en este curso veremos como hacerlo con el paquete *dplyr* (y *magrittr*) que están dentro del conjunto de paquetes llamado **tidyverse**

Lo primero que haremos es convertir los objetos a **Tibble**, este es un objeto del tidyverse que tiene ciertas mejoras al dataframe, por ejemplo cuando se quiere ver el objeto no imprime todo el objeto en pantalla, sino un resumen del mismo. (más información tipeando ?tibble)

```
# Convertir el data_banco a un tibble
data_banco <- as_tibble( data_banco)
# Muestra data_banco
data_banco
```

```
## # A tibble: 24,299 x 7
##   Sucursal Cajero ID_Transaccion Transaccion      Tiempo_Servicio~ Satisfaccion
##   <dbl>   <dbl> <chr>          <chr>          <dbl> <chr>
## 1      62    4820 2             Cobro/Pago (Cta~      622 Muy Bueno
## 2      62    4820 2             Cobro/Pago (Cta~      329 Malo
## 3      62    4820 2             Cobro/Pago (Cta~      643 Regular
## 4      62    4820 2             Cobro/Pago (Cta~      422 Regular
## 5      62    4820 2             Cobro/Pago (Cta~      378 Muy Bueno
## 6      62    4820 2             Cobro/Pago (Cta~      487 Bueno
## 7      62    4820 2             Cobro/Pago (Cta~      384 Regular
```



Seleccionar columnas: select()

Seleccionar las columnas Transaccion, Tiempo_Servicio_seg del data.frame data_banco

```
# Seleccionar las columnas Transaccion, Tiempo_Servicio_seg del data.frame data_banco
# Note que como no se asignó, R evalúa la expresión y presenta el resultado
select( data_banco, Transaccion, Tiempo_Servicio_seg)
```

```
## # A tibble: 24,299 x 2
##   Transaccion      Tiempo_Servicio_seg
##   <chr>          <dbl>
## 1 Cobro/Pago (Cta externa)      622
## 2 Cobro/Pago (Cta externa)      329
## 3 Cobro/Pago (Cta externa)      643
## 4 Cobro/Pago (Cta externa)      422
## 5 Cobro/Pago (Cta externa)      378
## 6 Cobro/Pago (Cta externa)      487
## 7 Cobro/Pago (Cta externa)      384
## 8 Cobro/Pago (Cta externa)      910
## 9 Cobro/Pago (Cta externa)      694
## 10 Cobro/Pago (Cta externa)      650
## # ... with 24,289 more rows
```

Operador Pipe: %>%

El operador Pipe %>% del paquete magrittr (y del tidyverse) permiten que el código sea más legible porque:

- Permite secuencias estructurantes de operaciones de datos de izquierda a derecha (a diferencia de dentro y fuera),

```
# Con el operador pipe  
data_banco %>% names
```

```
## [1] "Sucursal"          "Cajero"             "ID_Transaccion"  
## [4] "Transaccion"       "Tiempo_Servicio_seg" "Satisfaccion"  
## [7] "Monto"
```

```
# Sin el operador pipe  
names(data_banco)
```

```
## [1] "Sucursal"          "Cajero"             "ID_Transaccion"  
## [4] "Transaccion"       "Tiempo_Servicio_seg" "Satisfaccion"  
## [7] "Monto"
```




Operador Pipe: %>%

El operador Pipe %>% del paquete magrittr (y del tidyverse) permiten que el código sea más legible porque:

- Permite secuencias estructurantes de operaciones de datos de izquierda a derecha (a diferencia de dentro y fuera),
- Evita llamadas a funciones anidadas,

```
# Con Pipe
```

```
data_banco %>% names %>% length
```

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

```
# Sin Pipe
```

```
length(names(data_banco))
```

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

Operador Pipe: %>%

El operador Pipe %>% reemplaza el primer argumento del comando siguiente, es decir $x \%>\% f(y)$ es equivalente a $f(x, y)$.

```
head(data_banco, n= 3) # Sin Pipe
```

```
## # A tibble: 3 x 7
##   Sucursal Cajero ID_Transaccion Transaccion Tiempo_Servicio~ Satisfaccion Monto
##   <dbl> <dbl> <chr>          <chr>          <dbl> <chr>      <chr>
## 1      62   4820 2          Cobro/Pago~      622 Muy Bueno  2889~
## 2      62   4820 2          Cobro/Pago~      329 Malo     1670~
## 3      62   4820 2          Cobro/Pago~      643 Regular  3172~
```

```
data_banco %>% head(n= 3) # Con Pipe
```

```
## # A tibble: 3 x 7
##   Sucursal Cajero ID_Transaccion Transaccion Tiempo_Servicio~ Satisfaccion Monto
##   <dbl> <dbl> <chr>          <chr>          <dbl> <chr>      <chr>
## 1      62   4820 2          Cobro/Pago~      622 Muy Bueno  2889~
## 2      62   4820 2          Cobro/Pago~      329 Malo     1670~
## 3      62   4820 2          Cobro/Pago~      643 Regular  3172~
```

Luego retornaremos a ver más beneficios de este operador.



Seleccionar columnas: select()

Seleccionar las columnas Transaccion, Tiempo_Servicio_seg del data.frame data_banco pero usando %>%, lo que permite programar como si se escribiese "del data_banco, selecciona las columnas Transaccion y Tiempo_Servicio_seg"

```
# Seleccionar las columnas Transaccion, Tiempo_Servicio_seg del data.frame data_banco
# Note que como no se asignó, R evalúa la expresión y presenta el resultado
# Se lee, del data_banco, selecciona las columnas Transaccion y Tiempo_Servicio_seg
# data_banco[ , c("Transaccion", "Tiempo_Servicio_seg") ] ## Base de R
data_banco %>% select( Transaccion, Tiempo_Servicio_seg)
```

```
## # A tibble: 24,299 x 2
##   Transaccion      Tiempo_Servicio_seg
##   <chr>          <dbl>
## 1 Cobro/Pago (Cta externa)      622
## 2 Cobro/Pago (Cta externa)      329
## 3 Cobro/Pago (Cta externa)      643
## 4 Cobro/Pago (Cta externa)      422
## 5 Cobro/Pago (Cta externa)      378
## 6 Cobro/Pago (Cta externa)      487
## 7 Cobro/Pago (Cta externa)      384
## 8 Cobro/Pago (Cta externa)      910
## 9 Cobro/Pago (Cta externa)      694
## 10 Cobro/Pago (Cta externa)      650
```



Seleccionar columnas: select()

Seleccionar todas las columnas menos Cajero

```
# Seleccionar todas las columnas menos Cajero
data_banco %>% select( -Cajero)
```

```
## # A tibble: 24,299 x 6
##   Sucursal ID_Transaccion Transaccion Tiempo_Servicio_~ Satisfaccion Monto
##   <dbl> <chr>          <chr>          <dbl> <chr>          <chr>
## 1      62 2          Cobro/Pago (C~      622 Muy Bueno    2889,3
## 2      62 2          Cobro/Pago (C~      329 Malo        1670,69
## 3      62 2          Cobro/Pago (C~      643 Regular     3172,49
## 4      62 2          Cobro/Pago (C~      422 Regular     1764.92
## 5      62 2          Cobro/Pago (C~      378 Muy Bueno    1835.69
## 6      62 2          Cobro/Pago (C~      487 Bueno       2165.42
## 7      62 2          Cobro/Pago (C~      384 Regular     1304.9~
## 8      62 2          Cobro/Pago (C~      910 Bueno       4080.05
## 9      62 2          Cobro/Pago (C~      694 Muy Bueno    2541.27
## 10     62 2          Cobro/Pago (C~      650 Muy Bueno    2218.77
## # ... with 24,289 more rows
```



Seleccionar columnas: select()

Seleccionar según nombre de la columna/variable.

```
# Seleccionar todas las columnas cuyo nombre contenga el texto "Tra"
data_banco %>% select( contains("Tra"))
# Mostrar el resultado en el visor de RStudio
data_banco %>% select( contains("Tra")) %>% View
# Seleccionar todas las columnas cuyo nombre inicie con "S"
data_banco %>% select( starts_with("S")) %>% View
# Seleccionar todas las columnas cuyo nombre finalice con "on"
data_banco %>% select( ends_with("on")) %>% View
# Seleccionar todas las columnas cuyo nombre contenga una "r" o un "sa"
data_banco %>% select( matches("r?sa")) %>% View
# Más información sobre expresiones regulares usando: ?base::regex
```



Filtrar/Seleccionar filas: `filter()`

Filtrar las filas según las condiciones dadas en `filter()`, pero para esto debemos entender los operadores de relación y lógicos en R

Operadores de relación

```
3 == 4 # Igualdad
```

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

```
3 != 4 # Desigualdad
```

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

```
3 > 4 # Mayor que
```

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

```
3 <= 4 # Menor igual que
```

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



Operadores lógicos

```
! FALSE # No
```

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

```
TRUE & FALSE # Y
```

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

```
TRUE | FALSE # O
```

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

```
xor(TRUE, TRUE) # Ó excluyente
```

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

```
TRUE & NA # Cuidado especial con los NA
```

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




Operadores lógicos

```
xor(TRUE,TRUE) # Ó excluyente
```

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

```
TRUE & NA # Cuidado especial con los NA
```

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

Filtrar/Seleccionar filas: filter()

Filtrar las filas según las condiciones dadas en filter()

```
# Filtrar las filas correspondientes a la sucursal 62
data_banco %>% filter( Sucursal== 62 )
# ver el resultado en el visor de RStudio
data_banco %>% filter( Sucursal== 62 ) %>% View
# Filtrar las filas correspondientes a la sucursal 62 y hayan durado más de 120 seg
data_banco %>% filter( Sucursal== 62 & Tiempo_Servicio_seg > 120 ) %>% View
# Filtrar las filas correspondientes a la sucursal 62, hayan durado
# más de 120 segundos y la evaluación a la satisfacción sea Bueno
data_banco %>% filter( Sucursal== 62 & Tiempo_Servicio_seg > 120 &
                      Satisfaccion== "Muy Bueno") %>% View
```



Operador Pipe: %>%

Regresamos al operador Pipe %>%, recordemos que lo encontramos en el paquete magrittr y tidyverse, y su principal utilidad es que permite que el código sea más legible porque:

- Permite secuencias estructurantes de operaciones de datos de izquierda a derecha (a diferencia de dentro y fuera),
- Evitando llamadas a funciones anidadas,
- Minimiza la necesidad de variables locales y definiciones de funciones
- Facilita agregar pasos en cualquier lugar de la programación



Operador Pipe: %>%

Hay varios tipos de Pipe, %>%, %<>% %\$%; sólo el %>% viene en el tidyverse, el resto están en el paquete magrittr. Este operador funciona así:

- `x %>% f` es equivalente a `f(x)`
- `x %>% f(y)` es equivalente a `f(x, y)`
- `x %>% f %>% g %>% h` es equivalente a `h(g(f(x)))`
- `x %>% f(y = nrow(.), z = ncol(.))` es equivalente a `f(x, y = nrow(x), z = ncol(x))`
- `x %>% {f(y = nrow(.), z = ncol(.))}` es equivalente a `f(y = nrow(x), z = ncol(x))`
- `x %<>% f %>% g` es equivalente a `x <- g(f(x))`
- `%$%` permite seleccionar columnas `x %$% f(col1, col2)` es equivalente a `f(x$col1, x$col2)`

Sólo veremos el funcionamiento del caso principal, pero pueden notar lo potente del operador.

Utilidad del Operador Pipe: %>%

Filtrar las filas correspondientes a la sucursal 62 y hayan durado más de 500 seg pero mostrando sólo la Sucursal, Transaccion y Tiempo. **Con Pipe**

```
# Filtrar las filas correspondientes a la sucursal 62 y
# hayan durado más de 500 seg pero
# muestra sólo la Sucursal, Transaccion y Tiempo
data_banco %>%
  filter( Sucursal== 62 & Tiempo_Servicio_seg > 120 ) %>%
  select(Sucursal, Transaccion, Tiempo_Servicio_seg)
```

```
## # A tibble: 2,723 x 3
##   Sucursal Transaccion      Tiempo_Servicio_seg
##   <dbl> <chr>          <dbl>
## 1      62 Cobro/Pago (Cta externa)      622
## 2      62 Cobro/Pago (Cta externa)      329
## 3      62 Cobro/Pago (Cta externa)      643
## 4      62 Cobro/Pago (Cta externa)      422
## 5      62 Cobro/Pago (Cta externa)      378
## 6      62 Cobro/Pago (Cta externa)      487
## 7      62 Cobro/Pago (Cta externa)      384
## 8      62 Cobro/Pago (Cta externa)      910
## 9      62 Cobro/Pago (Cta externa)      694
## 10     62 Cobro/Pago (Cta externa)      650
## # ... with 2,713 more rows
```

Utilidad del Operador Pipe: %>%

Filtrar las filas correspondientes a la sucursal 62 y hayan durado más de 500 seg pero mostrando sólo la Sucursal, Transaccion y Tiempo. **Sin Pipe**

```
# Filtrar las filas correspondientes a la sucursal 62 y
# hayan durado más de 500 seg pero
# muestra sólo la Sucursal, Transaccion y Tiempo
select( filter( data_banco, Sucursal== 62 & Tiempo_Servicio_seg > 120 ),
        Sucursal, Transaccion, Tiempo_Servicio_seg)
```

```
## # A tibble: 2,723 x 3
##   Sucursal Transaccion      Tiempo_Servicio_seg
##   <dbl> <chr>          <dbl>
## 1      62 Cobro/Pago (Cta externa)      622
## 2      62 Cobro/Pago (Cta externa)      329
## 3      62 Cobro/Pago (Cta externa)      643
## 4      62 Cobro/Pago (Cta externa)      422
## 5      62 Cobro/Pago (Cta externa)      378
## 6      62 Cobro/Pago (Cta externa)      487
## 7      62 Cobro/Pago (Cta externa)      384
## 8      62 Cobro/Pago (Cta externa)      910
## 9      62 Cobro/Pago (Cta externa)      694
## 10     62 Cobro/Pago (Cta externa)      650
## # ... with 2,713 more rows
```



Utilidad del Operador Pipe: %>%

```
# Filtrar las filas correspondientes a la sucursal 62 y
# hayan durado más de 500 seg pero
# muestra sólo la Sucursal, Transaccion y Tiempo
# --- Con Pipe
data_banco %>%
  filter( Sucursal== 62 & Tiempo_Servicio_seg > 120 ) %>%
  select(Sucursal, Transaccion, Tiempo_Servicio_seg)
# --- Sin Pipe
select( filter( data_banco, Sucursal== 62 & Tiempo_Servicio_seg > 120 ),
        Sucursal, Transaccion, Tiempo_Servicio_seg)
```



Ordenar las filas: arrange()

Ordenar las filas según lo expresado en arrange()

```
# Ordenar por la satisfaccion  
data_banco %>% arrange( Satisfaccion ) %>% View  
# Ordenar cada Transaccion y dentro de cada transaccion  
# de mayor a menor por tiempo de servicio  
data_banco %>% arrange( Transaccion, desc(Tiempo_Servicio_seg) ) %>% View
```


Crear o modificar columnas/variables mutate()

Crear una nueva columna con el tiempo en minutos

```
# Crear una nueva columna con el tiempo en minutos
data_banco %>% mutate(Tiempo_Servicio_Min= Tiempo_Servicio_seg/60)
```

```
## # A tibble: 24,299 x 8
##   Sucursal Cajero ID_Transaccion Transaccion      Tiempo_Servicio~ Satisfaccion
##   <dbl>   <dbl> <chr>          <chr>          <dbl> <chr>
## 1      62    4820 2      Cobro/Pago (Cta~      622 Muy Bueno
## 2      62    4820 2      Cobro/Pago (Cta~      329 Malo
## 3      62    4820 2      Cobro/Pago (Cta~      643 Regular
## 4      62    4820 2      Cobro/Pago (Cta~      422 Regular
## 5      62    4820 2      Cobro/Pago (Cta~      378 Muy Bueno
## 6      62    4820 2      Cobro/Pago (Cta~      487 Bueno
## 7      62    4820 2      Cobro/Pago (Cta~      384 Regular
## 8      62    4820 2      Cobro/Pago (Cta~      910 Bueno
## 9      62    4820 2      Cobro/Pago (Cta~      694 Muy Bueno
## 10     62    4820 2      Cobro/Pago (Cta~      650 Muy Bueno
## # ... with 24,289 more rows, and 2 more variables: Monto <chr>,
## #   Tiempo_Servicio_Min <dbl>
```

Crear o modificar columnas/variables mutate()

Crear una nueva columna con el tiempo en minutos

```
# Crear una nueva columna con el tiempo en minutos
data_banco %>% mutate(Tiempo_Servicio_Min= Tiempo_Servicio_seg/60)
```

```
## # A tibble: 24,299 x 8
##   Sucursal Cajero ID_Transaccion Transaccion      Tiempo_Servicio~ Satisfaccion
##   <dbl>   <dbl> <chr>          <chr>          <dbl> <chr>
## 1      62    4820 2      Cobro/Pago (Cta~      622 Muy Bueno
## 2      62    4820 2      Cobro/Pago (Cta~      329 Malo
## 3      62    4820 2      Cobro/Pago (Cta~      643 Regular
## 4      62    4820 2      Cobro/Pago (Cta~      422 Regular
## 5      62    4820 2      Cobro/Pago (Cta~      378 Muy Bueno
## 6      62    4820 2      Cobro/Pago (Cta~      487 Bueno
## 7      62    4820 2      Cobro/Pago (Cta~      384 Regular
## 8      62    4820 2      Cobro/Pago (Cta~      910 Bueno
## 9      62    4820 2      Cobro/Pago (Cta~      694 Muy Bueno
## 10     62    4820 2      Cobro/Pago (Cta~      650 Muy Bueno
## # ... with 24,289 more rows, and 2 more variables: Monto <chr>,
## #   Tiempo_Servicio_Min <dbl>
```



Crear o modificar columnas/variables mutate()

Crear una nueva columna con el tiempo en minutos

```
# Crear una nueva columna con el tiempo en minutos  
data_banco %>% mutate(Tiempo_Servicio_Min= Tiempo_Servicio_seg/60)
```

Nótese que **no se asignó**, el objeto data_banco no tiene la columna Tiempo_Servicio_Min

Crear o modificar columnas/variables mutate()

Crear una nueva columna con el tiempo en minutos

```
# Crear una nueva columna con el tiempo en minutos
data_banco <- data_banco %>%
  mutate(Tiempo_Servicio_Min= Tiempo_Servicio_seg/60)
# Mostrar
data_banco
```

```
## # A tibble: 24,299 x 8
##   Sucursal Cajero ID_Transaccion Transaccion      Tiempo_Servicio~ Satisfaccion
##   <dbl>   <dbl> <chr>          <chr>          <dbl> <chr>
## 1      62    4820 2          Cobro/Pago (Cta~      622 Muy Bueno
## 2      62    4820 2          Cobro/Pago (Cta~      329 Malo
## 3      62    4820 2          Cobro/Pago (Cta~      643 Regular
## 4      62    4820 2          Cobro/Pago (Cta~      422 Regular
## 5      62    4820 2          Cobro/Pago (Cta~      378 Muy Bueno
## 6      62    4820 2          Cobro/Pago (Cta~      487 Bueno
## 7      62    4820 2          Cobro/Pago (Cta~      384 Regular
## 8      62    4820 2          Cobro/Pago (Cta~      910 Bueno
## 9      62    4820 2          Cobro/Pago (Cta~      694 Muy Bueno
## 10     62    4820 2          Cobro/Pago (Cta~      650 Muy Bueno
## # ... with 24,289 more rows, and 2 more variables: Monto <chr>,
## #   Tiempo_Servicio_Min <dbl>
```

Entender los datos - Ejemplo

¿Está bien nuestros tipos de datos?

Si no lo están entonces debemos transformarlos, para esto aprenderemos sobre manipulación de datos.

```
# Ver la estructura del data.frame  
data_banco %>% glimpse
```

```
## Rows: 24,299  
## Columns: 8  
## $ Sucursal      <dbl> 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62~  
## $ Cajero        <dbl> 4820, 4820, 4820, 4820, 4820, 4820, 4820, 4820, 48~  
## $ ID_Transaccion <chr> "2", "2", "2", "2", "2", "2", "2", "2", "2", "2", ~  
## $ Transaccion   <chr> "Cobro/Pago (Cta externa)", "Cobro/Pago (Cta exter~  
## $ Tiempo_Servicio_seg <dbl> 622, 329, 643, 422, 378, 487, 384, 910, 694, 650, ~  
## $ Satisfaccion  <chr> "Muy Bueno", "Malo", "Regular", "Regular", "Muy Bu~  
## $ Monto         <chr> "2889,3", "1670,69", "3172,49", "1764.92", "1835.6~  
## $ Tiempo_Servicio_Min <dbl> 10.366667, 5.483333, 10.716667, 7.033333, 6.300000~
```

Ejemplo - Manipulacion de datos

Lo primero que necesitamos es corregir los tipos de datos, nótese que

- **Monto** tiene una mezcla de "," y "."
- **Sucursal** y **Cajero** deberían ser de tipo character
- **Satisfaccion** debe ser factor ordenado

```
data_banco <- data_banco %>%  
  mutate( Monto= str_replace(Monto, pattern = ",", replacement = ".") ) %>%  
  mutate(Sucursal= as.character(Sucursal),  
         Cajero = as.character(Cajero),  
         Satisfaccion = parse_factor(Satisfaccion,  
                                     levels= c('Muy Malo', 'Malo', 'Regular',  
                                               'Bueno', 'Muy Bueno'), ordered = T),  
         Monto= parse_number(Monto))
```

Ejemplo - Manipulacion de datos

Tipos de Datos corregidos

```
# Ver la estructura del data.frame
data_banco %>% glimpse
```

```
## Rows: 24,299
## Columns: 8
## $ Sucursal      <chr> "62", "62", "62", "62", "62", "62", "62", "62", "6~
## $ Cajero        <chr> "4820", "4820", "4820", "4820", "4820", "4820", "4~
## $ ID_Transaccion <chr> "2", "2", "2", "2", "2", "2", "2", "2", "2", "2", ~
## $ Transaccion    <chr> "Cobro/Pago (Cta externa)", "Cobro/Pago (Cta exter~
## $ Tiempo_Servicio_seg <dbl> 622, 329, 643, 422, 378, 487, 384, 910, 694, 650, ~
## $ Satisfaccion   <ord> Muy Bueno, Malo, Regular, Regular, Muy Bueno, Buen~
## $ Monto          <dbl> 2889.30, 1670.69, 3172.49, 1764.92, 1835.69, 2165.~
## $ Tiempo_Servicio_Min <dbl> 10.366667, 5.483333, 10.716667, 7.033333, 6.300000~
```

Resumir/Agregar los Datos

Taller Tidyverse

Néstor Montaña P.

Crear resúmenes: summarise()

`summarise()` permite aplicar funciones a las columnas de nuestro `data.frame`. En R-base se usaría `tapply()` y otra opción muy conocida es `ddply()` del paquete `plyr`.

```
# Obtener la medidas descriptivas del tiempo de servicio
data_banco %>%
  summarise(
    MEDIA= mean(Tiempo_Servicio_seg, na.rm=TRUE),
    MEDIA_ACOT= mean(Tiempo_Servicio_seg, na.rm = TRUE, trim = 0.05),
    DESV= sd(Tiempo_Servicio_seg, na.rm=TRUE),
    RANGO= diff(range(Tiempo_Servicio_seg)),
    CANTIDAD= n() # n() permite contar el número de filas
  )
```

```
## # A tibble: 1 x 5
##   MEDIA MEDIA_ACOT  DESV RANGO CANTIDAD
##   <dbl>    <dbl> <dbl> <dbl>    <int>
## 1  166.      148.  145.  1598    24299
```

Crear resúmenes: summarise()

`summarise_at()` para escoger la(s) variable(s) a utilizar en los cálculos, la diferencia con lo anterior es que no se repite el nombre de la variable en cada línea.

```
# Obtener la media del tiempo de servicio
data_banco %>%
  summarise_at( vars(Tiempo_Servicio_seg),
    list(
      MEDIA= ~mean(., na.rm=TRUE),
      MEDIA_ACOT= ~mean(., na.rm = TRUE, trim = 0.05),
      DESV= ~sd(., na.rm=TRUE),
      RANGO= ~diff(range(.)),
      CANTIDAD= ~n()
    )
  )
```

```
## # A tibble: 1 x 5
##   MEDIA MEDIA_ACOT  DESV RANGO CANTIDAD
##   <dbl>    <dbl> <dbl> <dbl>    <int>
## 1  166.      148.  145.  1598    24299
```

Crear resúmenes: summarise()

`summarise_at()` escogiendo varias variables, nótese el nombre de las columnas resultantes

```
data_banco %>%  
  summarise_if( is.numeric,  
                list(  
                  MEDIA= ~mean(., na.rm=TRUE),  
                  MEDIA_ACOT= ~mean(., na.rm = TRUE, trim = 0.05)  
                )  
  )  
  
## # A tibble: 1 x 6  
##   Tiempo_Servicio_seg_MEDIA Monto_MEDIA Tiempo_Servicio_Mi~ Tiempo_Servicio_seg~  
##               <dbl>         <dbl>             <dbl>             <dbl>  
## 1             166.         1996.             2.76             148.  
## # ... with 2 more variables: Monto_MEDIA_ACOT <dbl>,  
## #   Tiempo_Servicio_Min_MEDIA_ACOT <dbl>
```

Crear resúmenes: summarise()

`summarise_if()` permite escoger varias variables que cumplan una condición

```
# Obtener la media del tiempo de servicio y el Monto
data_banco %>%
  summarise_if( is.numeric,
                list(
                  MEDIA= ~mean(., na.rm=TRUE),
                  MEDIA_ACOT= ~mean(., na.rm = TRUE, trim = 0.05)
                )
  )
```

```
## # A tibble: 1 x 6
##   Tiempo_Servicio_seg MEDIA Monto_MEDIA Tiempo_Servicio_Mi~ Tiempo_Servicio_seg~
##               <dbl>    <dbl>          <dbl>          <dbl>
## 1             166.    1996.          2.76          148.
## # ... with 2 more variables: Monto_MEDIA_ACOT <dbl>,
## #   Tiempo_Servicio_Min_MEDIA_ACOT <dbl>
```

Crear resúmenes para datos agrupados

`group_by()` permite aplicar funciones a nuestro `data.frame` separado por una o más variables, por ejemplo para obtener medidas de tendencia central para el tiempo de servicio según cada Transacción se haría:

```
# Obtener medidas de tendencia central para el tiempo de servicio para cada tipo de transacción
data_banco %>%
  group_by(Transaccion) %>%
  summarise(
    MEDIA= mean(Tiempo_Servicio_seg, na.rm=TRUE),
    MEDIA_ACOT= mean(Tiempo_Servicio_seg, na.rm = TRUE, trim = 0.05),
    DESV= sd(Tiempo_Servicio_seg, na.rm=TRUE),
    CANTIDAD= n()
  )
```

```
## # A tibble: 3 x 5
##   Transaccion      MEDIA MEDIA_ACOT  DESV CANTIDAD
##   <chr>          <dbl>      <dbl> <dbl>    <int>
## 1 Cobrar cheque (Cta del Bco) 212.        200. 124.     5407
## 2 Cobro/Pago (Cta externa)   387.        370. 204.     3005
## 3 Deposito                 108.        100.  74.9   15887
```

Crear resúmenes para datos agrupados

Obtener medidas de tendencia central del tiempo de servicio para cada combinación de Transaccion y Nivel de Satisfaccion.

```
# Obtener medidas de tendencia central del
# tiempo de servicio para cada combinación de
# Transaccion y Nivel de Satisfaccion.
data_banco %>%
  group_by(Transaccion, Satisfaccion) %>%
  summarise(
    MEDIA= mean(Tiempo_Servicio_seg, na.rm=TRUE),
    MEDIA_ACOT= mean(Tiempo_Servicio_seg,
                     na.rm = TRUE, trim = 0.05),
    DESV= sd(Tiempo_Servicio_seg, na.rm=TRUE),
    CANTIDAD= n()
  )
```

`summarise()` has grouped output by 'Transaccion'. You can ov

```
## # A tibble: 15 x 6
## # Groups:   Transaccion [3]
##   Transaccion      Satisfaccion MEDIA MEDIA_ACOT
##   <chr>          <ord>      <dbl>    <dbl>
## 1 Cobrar cheque (Cta del Bco) Muy Malo      224.      214.
## 2 Cobrar cheque (Cta del Bco) Malo           218.      207.
## 3 Cobrar cheque (Cta del Bco) Regular          211.      199.
## 4 Cobrar cheque (Cta del Bco) Bueno            212.      200.
## 5 Cobrar cheque (Cta del Bco) Muy Bueno         206.      194.
## 6 Cobro/Pago (Cta externa) Muy Malo         429.      417.
## 7 Cobro/Pago (Cta externa) Malo              417.      400.
## 8 Cobro/Pago (Cta externa) Regular            384.      365.
## 9 Cobro/Pago (Cta externa) Bueno              387.      370.
## 10 Cobro/Pago (Cta externa) Muy Bueno          365.      351.
## 11 Deposito Muy Malo              116.      108.
## 12 Deposito Malo                  111.      104.
## 13 Deposito Regular              107.       98.8
## 14 Deposito Bueno                105.       97.4
## 15 Deposito Muy Bueno            104.       95.7
```

Crear resúmenes para datos agrupados y filtrados

Para la Sucursal 62, obtener medidas de tendencia central del tiempo de servicio para cada combinación de Transaccion y Nivel de Satisfaccion.

```
# Para la Sucursal 62
# Obtener medidas de tendencia central del
# tiempo de servicio para cada combinación de
# Transaccion y Nivel de Satisfaccion.
data_banco %>%
  filter( Sucursal== 62) %>%
  group_by(Transaccion, Satisfaccion) %>%
  summarise(
    MEDIA= mean(Tiempo_Servicio_seg, na.rm=TRUE),
    MEDIA_ACOT= mean(Tiempo_Servicio_seg,
                     na.rm = TRUE, trim = 0.05),
    DESV= sd(Tiempo_Servicio_seg, na.rm=TRUE),
    CANTIDAD= n()
  )
```

`summarise()` has grouped output by 'Transaccion'. You can ov

```
## # A tibble: 15 x 6
## # Groups:   Transaccion [3]
##   Transaccion      Satisfaccion MEDIA MEDIA_ACOT
##   <chr>          <ord>      <dbl>    <dbl>
## 1 Cobrar cheque (Cta del Bco) Muy Malo      405.      399.
## 2 Cobrar cheque (Cta del Bco) Malo          419.      417.
## 3 Cobrar cheque (Cta del Bco) Regular        413.      411.
## 4 Cobrar cheque (Cta del Bco) Bueno          400.      397.
## 5 Cobrar cheque (Cta del Bco) Muy Bueno      394.      392.
## 6 Cobro/Pago (Cta externa) Muy Malo      634.      628.
## 7 Cobro/Pago (Cta externa) Malo          617.      613.
## 8 Cobro/Pago (Cta externa) Regular        689.      683.
## 9 Cobro/Pago (Cta externa) Bueno          658.      653.
## 10 Cobro/Pago (Cta externa) Muy Bueno      603.      599.
## 11 Deposito      Muy Malo      238.      236.
## 12 Deposito      Malo          227.      225.
## 13 Deposito      Regular        227.      225.
## 14 Deposito      Bueno          229.      227.
## 15 Deposito      Muy Bueno      237.      236.
```

Crear resúmenes con funciones de más de una salida

Suponga que desea obtener medidas de posición para el tiempo de servicio según el Nivel de Satisfacción, ya vimos que la función `quantile` genera un vector y no sólo un número, ¿Cómo podríamos usar esta salida dentro de `summarise`?

```
# Obtener medidas de tendencia central
data_banco %>%
  group_by(Satisfaccion) %>%
  summarise(
    tibble(
      Quartil = c("Min", "Q1", "Q2", "Q3", "Max"),
      Valor= quantile(Tiempo_Servicio_seg,
                     c(0, 0.25, 0.5, 0.75, 1))
    )
  )
```

`summarise()` has grouped output by 'Satisfaccion'. You can c

```
## # A tibble: 25 x 3
## # Groups:   Satisfaccion [5]
##   Satisfaccion Quartil Valor
##   <ord>         <chr>   <dbl>
## 1 Muy Malo      Min        22
## 2 Muy Malo      Q1         71
## 3 Muy Malo      Q2        107
## 4 Muy Malo      Q3        207
## 5 Muy Malo      Max       1158
## 6 Malo          Min         18
## 7 Malo          Q1         70
## 8 Malo          Q2        105
## 9 Malo          Q3        207
## 10 Malo         Max       1616
## # ... with 15 more rows
```




Tablas de Frecuencia

- Agrupación de datos en clases mutuamente excluyentes, que muestra el número de observaciones que hay en cada clase.
 - Se agrupa en Intervalos si la variable es cuantitativa.
 - Se cuenta cada elemento si la variable es cualitativa.
 - Se lo muestra gráficamente con un histograma o gráfico de barras



Tablas de Frecuencia - V. Cualitativa

- En R base se usa el comando `table()`
- En Tidyverse se puede hacer esto:

```
# Obtener medidas de tendencia central
data_banco %>%
  group_by(Transaccion) %>%
  count(name = 'Frecuencia')
```

```
## # A tibble: 3 x 2
## # Groups:   Transaccion [3]
##   Transaccion      Frecuencia
##   <chr>          <int>
## 1 Cobrar cheque (Cta del Bco)      5407
## 2 Cobro/Pago (Cta externa)       3005
## 3 Deposito                    15887
```

Tablas de Frecuencia - V. Cualitativa

- En R base se usa el comando `table()`
- En Tidyverse se puede hacer esto:

```
# Obtener medidas de tendencia central
data_banco %>%
  group_by(Transaccion) %>%
  count(name = 'Frecuencia') %>%
  ungroup() %>%
  mutate(
    F_Relativa= round(
      Frecuencia/sum(Frecuencia) ,4) *100,
    F_Acum= cumsum(Frecuencia),
    F_Rel_Acum= round(
      cumsum(Frecuencia)/sum(Frecuencia) ,4) *100
  )
```

```
## # A tibble: 3 x 5
##   Transaccion      Frecuencia F_Relativa F_Acum F_
##   <chr>          <int>      <dbl>   <int>
## 1 Cobrar cheque (Cta del Bco)      5407      22.2    5407
## 2 Cobro/Pago (Cta externa)       3005      12.4    8412
## 3 Deposito                    15887      65.4   24299
```



Tablas de Frecuencia - V. Numerica

- Una opción es usar `library('fdth')`
- En Tidyverse se puede hacer esto:

```
# Límites
limites= seq(0, 7000, 1000)
# data_banco %>%
#   mutate(
#     Monto_Int= cut(Monto, limites, dig.lab= 5)) %>%
#   group_by(Monto_Int) %>%
#   count(name = 'Frecuencia')
```



Tablas de Frecuencia - V. Numerica

- Una opción es usar `library('fdth')`
- En Tidyverse se puede hacer esto:

```
# Límites
limites= seq(0, 7000, 1000)
# data_banco %>%
#   mutate(Monto_Int= cut(Monto, limites, dig.lab= 5 )
#   group_by(Monto_Int) %>%
#   count(name = 'Frecuencia') %>%
#   ungroup() %>%
#   mutate(
#     F_Relativa= round(
#       Frecuencia/sum(Frecuencia) ,4) *100,
#     F_Acum= cumsum(Frecuencia),
#     F_Rel_Acum= round(
#       cumsum(Frecuencia)/sum(Frecuencia) ,4) *100
#   )
```

Manipulación de datos - Unir Datos

Taller Tidyverse

Néstor Montaña P.

Unir Datos

Se va a crear unos data.frame simples para entender la idea detrás de los comandos a usar

```
# Crear un Data frame
df_1 <- data.frame(
  Nombre= c('Ana', 'Berni', 'Carlos', 'Daniel', 'Ericka'),
  Edad = c(20,19,20,19,18),
  Ciudad= factor(c('Gye', 'Uio', 'Cue', 'Gye', 'Cue')) )
df_1
```

```
##   Nombre Edad Ciudad
## 1   Ana   20   Gye
## 2  Berni   19   Uio
## 3 Carlos   20   Cue
## 4 Daniel   19   Gye
## 5 Ericka   18   Cue
```



Unir Datos

```
# Crear un Data frame
df_2 <- data.frame(
  Nombre= c('Fulton', 'Gilda'),
  Ciudad= factor(c('Mach', 'Gye')) ,
  Edad = c(21,18)
)
df_2
```

```
##   Nombre Ciudad Edad
## 1  Fulton   Mach   21
## 2   Gilda    Gye   18
```




Unir Datos

```
# Crear un Data frame
df_3 <- data.frame(
  Estado_Civil= c('S', 'D', "S", "C", "D")
)
df_3
```

```
##   Estado_Civil
## 1             S
## 2             D
## 3             S
## 4             C
## 5             D
```



Unir filas

Para unir filas se usa `bind_rows`

```
# Aumentar filas  
bind_rows(df_1, df_2)
```

```
##   Nombre Edad Ciudad  
## 1    Ana   20    Gye  
## 2   Berni  19    Uio  
## 3 Carlos  20    Cue  
## 4 Daniel  19    Gye  
## 5 Ericka  18    Cue  
## 6 Fulton 21    Mach  
## 7   Gilda  18    Gye
```



Unir filas

Para unir filas se usa `bind_rows`

```
# Aumentar filas  
bind_rows(df_1, df_3)
```

##	Nombre	Edad	Ciudad	Estado_Civil
## 1	Ana	20	Gye	<NA>
## 2	Berni	19	Uio	<NA>
## 3	Carlos	20	Cue	<NA>
## 4	Daniel	19	Gye	<NA>
## 5	Ericka	18	Cue	<NA>
## 6	<NA>	NA	<NA>	S
## 7	<NA>	NA	<NA>	D
## 8	<NA>	NA	<NA>	S
## 9	<NA>	NA	<NA>	C
## 10	<NA>	NA	<NA>	D

```
# rbind(df_1, df_3) #ERROR
```



Unir filas

Aumentar columna que indica origen de la fila

```
# Aumentar columna que indica origen de la fila  
bind_rows(list(df_1, df_2), .id = "id")
```

##	id	Nombre	Edad	Ciudad
## 1	1	Ana	20	Gye
## 2	1	Berni	19	Uio
## 3	1	Carlos	20	Cue
## 4	1	Daniel	19	Gye
## 5	1	Ericka	18	Cue
## 6	2	Fulton	21	Mach
## 7	2	Gilda	18	Gye



Unir filas

Aumentar columna que indica origen de la fila

```
# Aumentar columna que indica origen de la fila  
bind_rows(list(data_1= df_1, data_2= df_2), .id = "DF_ORIGEN")
```

##	DF_ORIGEN	Nombre	Edad	Ciudad
## 1	data_1	Ana	20	Gye
## 2	data_1	Berni	19	Uio
## 3	data_1	Carlos	20	Cue
## 4	data_1	Daniel	19	Gye
## 5	data_1	Ericka	18	Cue
## 6	data_2	Fulton	21	Mach
## 7	data_2	Gilda	18	Gye

Unir filas

Bind más de dos dataframes

```
# Bind más de dos dataframes
bind_rows(list(df_1= df_1, df_2= df_2, df_3= df_3), .id = "DF_ORIGEN")
```

##	DF_ORIGEN	Nombre	Edad	Ciudad	Estado_Civil
## 1	df_1	Ana	20	Gye	<NA>
## 2	df_1	Berni	19	Uio	<NA>
## 3	df_1	Carlos	20	Cue	<NA>
## 4	df_1	Daniel	19	Gye	<NA>
## 5	df_1	Ericka	18	Cue	<NA>
## 6	df_2	Fulton	21	Mach	<NA>
## 7	df_2	Gilda	18	Gye	<NA>
## 8	df_3	<NA>	NA	<NA>	S
## 9	df_3	<NA>	NA	<NA>	D
## 10	df_3	<NA>	NA	<NA>	S
## 11	df_3	<NA>	NA	<NA>	C
## 12	df_3	<NA>	NA	<NA>	D



Unir columnas

Para unir columnas se usa `bind_cols`

```
# Bind más de dos dataframes  
bind_cols(df_1, df_3)
```

```
##   Nombre Edad Ciudad Estado_Civil  
## 1   Ana   20   Gye             S  
## 2  Berni  19   Uio             D  
## 3 Carlos  20   Cue             S  
## 4 Daniel  19   Gye             C  
## 5 Ericka  18   Cue             D
```



Unir columnas

Para unir columnas se usa `bind_cols`, se requiere que los datos tengan el mismo largo

```
# Bind más de dos dataframes  
bind_cols(df_1, df_2) #ERROR
```

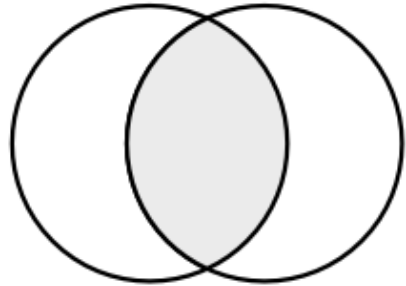



Unir datos - Merge|Join|Buscarv

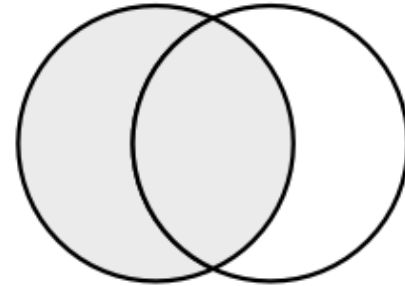
- Se tienen dos data.frames con columnas o variables que hacen las veces de “key” o “id” de los mismos
- Se desea agregar al primer conjunto el contenido del segundo conjunto de datos si y sólo si el “key” o “id” del segundo conjunto corresponde con el “key” o “id” del primer conjunto de datos.
- Parecido al Buscarv y Vlookup de excel
- Equivalente al Join de Bases de datos

Unir datos - Merge|Join|Buscar

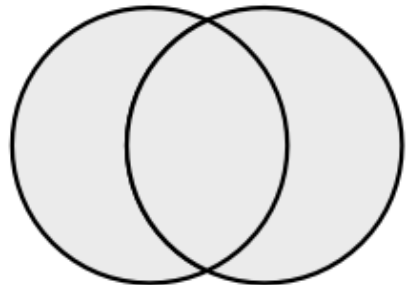
Entendiendo los tipos de Join



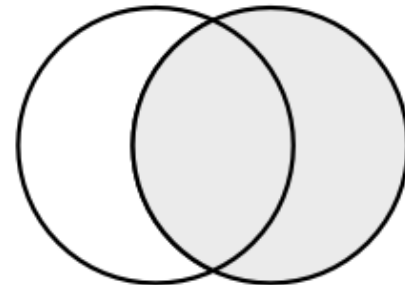
`inner_join(x, y)`



`left_join(x, y)`



`full_join(x, y)`



`right_join(x, y)`



Unir datos - Merge|Join|Buscar

Nuevo data.frame

```
df_6 <- data.frame(A= c('Ana', 'Daniel', 'Jose'), B= c(100,200,300))  
df_6
```

```
##           A     B  
## 1      Ana  100  
## 2 Daniel  200  
## 3    Jose  300
```



Unir datos - Merge|Join|Buscar

En base R se usaba el comando Merge

```
# Inner Join
merge(x= df_1, y= df_6, by.x= 'Nombre', by.y= 'A')
# Right Join
merge(x= df_1, y= df_6, by.x= 'Nombre', by.y= 'A', all.x= TRUE)
# Left Join
merge(x= df_1, y= df_6, by.x= 'Nombre', by.y= 'A', all.y= TRUE)
# Full Join
merge(x= df_1, y= df_6, by.x= 'Nombre', by.y= 'A', all= TRUE)
```



Unir datos - Inner Join

```
df_1 %>% inner_join(df_6, by = c("Nombre"="A"))
```

```
##   Nombre Edad Ciudad    B
## 1     Ana   20     Gye 100
## 2 Daniel   19     Gye 200
```



Unir datos - Left Join

```
df_1 %>% left_join(df_6, by = c("Nombre"="A"))
```

##	Nombre	Edad	Ciudad	B
## 1	Ana	20	Gye	100
## 2	Berni	19	Uio	NA
## 3	Carlos	20	Cue	NA
## 4	Daniel	19	Gye	200
## 5	Ericka	18	Cue	NA



Unir datos - Right Join

```
df_1 %>% right_join(df_6, by = c("Nombre"="A"))
```

```
##   Nombre Edad Ciudad    B
## 1    Ana   20    Gye 100
## 2 Daniel   19    Gye 200
## 3   Jose   NA   <NA> 300
```



Unir datos - Full Join

```
df_1 %>% full_join(df_6, by = c("Nombre"="A"))
```

```
##   Nombre Edad Ciudad    B
## 1    Ana   20    Gye 100
## 2  Berni   19    Uio  NA
## 3 Carlos   20    Cue  NA
## 4 Daniel   19    Gye 200
## 5 Ericka   18    Cue  NA
## 6   Jose   NA  <NA> 300
```




Unir datos - Merge|Join|Buscar

Vamos a duplicar un Valor en df_6 y a replicar los Joins para revisar qué sucede cuando se tiene "key" no únicos

```
df_6 <- data.frame(A= c('Ana', 'Daniel', 'Jose', 'Ana'), B= c(100, 200, 300, 110))  
df_6
```

```
##      A    B  
## 1   Ana 100  
## 2 Daniel 200  
## 3   Jose 300  
## 4   Ana 110
```



Unir datos - Inner Join

Nótese la repetición de "Ana"

```
df_1 %>% inner_join(df_6, by = c("Nombre"="A"))
```

```
##   Nombre Edad Ciudad    B
## 1    Ana   20    Gye  100
## 2    Ana   20    Gye  110
## 3 Daniel   19    Gye  200
```



Unir datos - Left Join

Nótese la repetición de "Ana"

```
df_1 %>% left_join(df_6, by = c("Nombre"="A"))
```

##	Nombre	Edad	Ciudad	B
## 1	Ana	20	Gye	100
## 2	Ana	20	Gye	110
## 3	Berni	19	Uio	NA
## 4	Carlos	20	Cue	NA
## 5	Daniel	19	Gye	200
## 6	Ericka	18	Cue	NA



Unir datos - Right Join

Nótese la repetición de "Ana"

```
df_1 %>% right_join(df_6, by = c("Nombre"="A"))
```

##	Nombre	Edad	Ciudad	B
## 1	Ana	20	Gye	100
## 2	Ana	20	Gye	110
## 3	Daniel	19	Gye	200
## 4	Jose	NA	<NA>	300



Unir datos - Full Join

Nótese la repetición de "Ana"

```
df_1 %>% full_join(df_6, by = c("Nombre"="A"))
```

##	Nombre	Edad	Ciudad	B
## 1	Ana	20	Gye	100
## 2	Ana	20	Gye	110
## 3	Berni	19	Uio	NA
## 4	Carlos	20	Cue	NA
## 5	Daniel	19	Gye	200
## 6	Ericka	18	Cue	NA
## 7	Jose	NA	<NA>	300



Unir datos

Y ¿si repetimos "Ana" en el df_1?



Unir datos

Y si repetimos "Ana" en el df_1?

```
df_1 %>%  
  bind_rows( data.frame(Nombre="Ana", Edad= 42, Ciudad= 'Cue')) %>%  
  full_join(df_6, by = c("Nombre"="A"))
```

##	Nombre	Edad	Ciudad	B
## 1	Ana	20	Gye	100
## 2	Ana	20	Gye	110
## 3	Berni	19	Uio	NA
## 4	Carlos	20	Cue	NA
## 5	Daniel	19	Gye	200
## 6	Ericka	18	Cue	NA
## 7	Ana	42	Cue	100
## 8	Ana	42	Cue	110
## 9	Jose	NA	<NA>	300



Filterin Joins

`semi_join(x, y)` da todas las observaciones en x que tienen al menos una coincidencia en y.

```
df_1 %>% semi_join(df_6, by = c("Nombre"="A"))
```

```
##   Nombre Edad Ciudad
## 1    Ana   20    Gye
## 2 Daniel   19    Gye
```




Filterin Joins

`anti_join(x, y)` da todas las observaciones en x que NO tienen coincidencia en y.

```
df_1 %>% anti_join(df_6, by = c("Nombre"="A"))
```

```
##   Nombre Edad Ciudad
## 1  Berni   19    Uio
## 2 Carlos   20    Cue
## 3 Ericka   18    Cue
```



Join - Ejemplo transacciones bancarias

¿Qué join necesitamos realizar con nuestros datos?

```
View(data_banco)  
View(data_sucursal)
```



Join - Ejemplo transacciones bancarias

Esto nos dará error, ¿Qué falta?

```
data_banco %>%  
  left_join(data_sucursal, by= c("Sucursal"= "ID_Sucursal"))
```



Join - Ejemplo transacciones bancarias

Esto nos dará error, ¿Qué falta?

R: Corregir tipo de datos

```
data_sucursal <- data_sucursal %>%  
  mutate(ID_Sucursal= as.character(ID_Sucursal))  
data_sucursal
```

##	ID_Sucursal	Sucursal	Nuevo_Sistema
## 1	62	Riocentro Sur	No
## 2	85	Centro	Si
## 3	267	Alborada	Si
## 4	443	Mall del Sol	Si
## 5	586	Via Daule	No

Join - Ejemplo transacciones bancarias

Nótese el problema con el nombre "Sucursal"

```
data_banco %>%
  left_join(data_sucursal, by= c("Sucursal"= "ID_Sucursal"))
```

```
## # A tibble: 24,299 x 10
##   Sucursal Cajero ID_Transaccion Transaccion      Tiempo_Servicio~ Satisfaccion
##   <chr>    <chr>   <chr>          <chr>          <dbl> <ord>
## 1 62      4820     2      Cobro/Pago (Cta~      622 Muy Bueno
## 2 62      4820     2      Cobro/Pago (Cta~      329 Malo
## 3 62      4820     2      Cobro/Pago (Cta~      643 Regular
## 4 62      4820     2      Cobro/Pago (Cta~      422 Regular
## 5 62      4820     2      Cobro/Pago (Cta~      378 Muy Bueno
## 6 62      4820     2      Cobro/Pago (Cta~      487 Bueno
## 7 62      4820     2      Cobro/Pago (Cta~      384 Regular
## 8 62      4820     2      Cobro/Pago (Cta~      910 Bueno
## 9 62      4820     2      Cobro/Pago (Cta~      694 Muy Bueno
## 10 62     4820     2      Cobro/Pago (Cta~      650 Muy Bueno
## # ... with 24,289 more rows, and 4 more variables: Monto <dbl>,
## #   Tiempo_Servicio_Min <dbl>, Sucursal.y <chr>, Nuevo_Sistema <chr>
```

Join - Ejemplo transacciones bancarias

Cambiamos el nombre y ya reemplazamos el data_banco con este resultado

```
data_banco <- data_banco %>%
  rename("ID_Sucursal"="Sucursal") %>%
  left_join(data_sucursal, by= c("ID_Sucursal"))
data_banco
```

```
## # A tibble: 24,299 x 10
##   ID_Sucursal Cajero ID_Transaccion Transaccion Tiempo_Servicio~ Satisfaccion
##   <chr>      <chr>   <chr>          <chr>          <dbl> <ord>
## 1 62        4820     2          Cobro/Pago (~      622 Muy Bueno
## 2 62        4820     2          Cobro/Pago (~      329 Malo
## 3 62        4820     2          Cobro/Pago (~      643 Regular
## 4 62        4820     2          Cobro/Pago (~      422 Regular
## 5 62        4820     2          Cobro/Pago (~      378 Muy Bueno
## 6 62        4820     2          Cobro/Pago (~      487 Bueno
## 7 62        4820     2          Cobro/Pago (~      384 Regular
## 8 62        4820     2          Cobro/Pago (~      910 Bueno
## 9 62        4820     2          Cobro/Pago (~      694 Muy Bueno
## 10 62       4820     2          Cobro/Pago (~      650 Muy Bueno
## # ... with 24,289 more rows, and 4 more variables: Monto <dbl>,
## #   Tiempo_Servicio_Min <dbl>, Sucursal <chr>, Nuevo_Sistema <chr>
```

Ejemplo transacciones bancarias

El data_banco ahora es

```
glimpse(data_banco)
```

```
## Rows: 24,299
## Columns: 10
## $ ID_Sucursal      <chr> "62", "62", "62", "62", "62", "62", "62", "62", "6~
## $ Cajero           <chr> "4820", "4820", "4820", "4820", "4820", "4820", "4~
## $ ID_Transaccion   <chr> "2", "2", "2", "2", "2", "2", "2", "2", "2", "2", ~
## $ Transaccion       <chr> "Cobro/Pago (Cta externa)", "Cobro/Pago (Cta exter~
## $ Tiempo_Servicio_seg <dbl> 622, 329, 643, 422, 378, 487, 384, 910, 694, 650, ~
## $ Satisfaccion      <ord> Muy Bueno, Malo, Regular, Regular, Muy Bueno, Buen~
## $ Monto             <dbl> 2889.30, 1670.69, 3172.49, 1764.92, 1835.69, 2165.~
## $ Tiempo_Servicio_Min <dbl> 10.366667, 5.483333, 10.716667, 7.033333, 6.300000~
## $ Sucursal          <chr> "Riocentro Sur", "Riocentro Sur", "Riocentro Sur",~
## $ Nuevo_Sistema     <chr> "No", "No", "No", "No", "No", "No", "No", "No", "N~
```

FIN

Taller Tidyverse

Néstor Montaña P.