



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

**Tutorial del lenguaje de
programación R orientado a
la inteligencia de mercado**

MATERIAL DIDÁCTICO

Que para obtener el título de

Ingeniero Industrial

P R E S E N T A

Diego Spamer Rivera

ASESOR DE MATERIAL DIDÁCTICO

Dr. Ricardo Torres Mendoza



Ciudad Universitaria, Cd. Mx., 2020

Tabla de contenido

Introducción	2
i. Antecedentes	2
ii. Objetivo	2
iii. Contenido	2
1. R y Rstudio, instalación e introducción.	3
1.1 Definición de programa R.	3
1.2 Rstudio	3
1.3 Instalación de paquetes	8
1.4 Aritmética con R	10
1.5 Asignación de Variables	11
1.6 Tipos de datos básicos en R	12
1.7 Identificar tipo de datos	13
1.8 Vectores	14
1.9 Cómo nombrar fácilmente una variable	16
1.10 Calcular totales	17
1.11 Selección de vectores	19
1.12 Seleccionar múltiples elementos de un vector	19
1.13 Seleccionar múltiples elementos de un vector (versión corta)	20
1.14 Seleccionar múltiples elementos de un vector (con nombres)	21
1.15 Operadores relacionales	22
1.16 Mayor y Menor que	24
1.17 Operadores Lógicos	25
1.18 El enunciado if	27
1.19 else if	29
1.20 Vectores (continuación)	31
1.21 Selección avanzada	34
1.22 Matrices	35
1.23 Nombrar una matriz	35
1.24 Cálculos por renglón	37
1.25 Agregar una columna a una matriz	38
1.26 Agregar una fila a una matriz	39
1.27 Suma de columna	39
1.28 Selección de elementos matriciales	40

1.29	Un poco de aritmética con matrices.	4
1.30	Comparación de matrices	43
2.	Conceptos clave en el análisis de datos y alcance del tutorial	45
2.1	Importación de datos	45
2.1.1	Leer, conectar, cargar fuentes de datos	45
2.2	Limpieza de datos	46
2.2.1	Eliminar duplicados	46
2.2.2	Outliers y valores faltantes	46
2.3	Transformación de datos	47
2.3.1	Tipos de Variables	47
2.3.2	Transponer sets de datos	47
2.3.3	Ordenar un set de datos	48
2.3.4	Muestras de un conjunto de datos en R	48
2.3.5	Combinar fuentes de datos	48
2.4	Visualización de datos	49
2.4.1	Crear histogramas	49
2.5	Modelado	50
2.5.1	Suma, cuenta y promedio de una variable en un set de datos	51
2.6	Exploración y Transformación de Datos	51
3.	Caso de Estudio 1: Análisis del flujo de autos en los estacionamientos de la Facultad.	52
3.1	Objetivo	52
3.2	Importación de datos	52
3.3	Limpieza de datos	53
3.4	Transformación de datos	58
3.5	Visualización de datos	60
4.	Caso de Estudio 2: Datos de Inmuebles	62
4.1	Objetivo	62
4.2	Importación de datos	62
4.3	Análisis exploratorio de datos	63
5.	Caso de Estudio 3: Horarios de Películas	76
5.1	Objetivo	76
5.2	Análisis exploratorio de datos	76
5.3	Preguntas de negocio	83
6.	Conclusión	84
7.	Referencias Bibliográficas	85

Introducción

i. Antecedente

Hoy en día existen diversas fuentes digitales¹ que ofrecen cursos de programación en lenguaje R y otros orientándose al análisis de datos, el objetivo de este tutorial será proporcionar un apoyo didáctico a la materia de Inteligencia de Negocios para brindar una introducción al entorno y lenguaje R, así como la interfaz gráfica RStudio, el uso de paquetes adicionales y análisis de datos.

ii. Objetivo

El objetivo de este tutorial será brindar una introducción al entorno y lenguaje R, se podrá familiarizar con el lenguaje R, así como la interfaz gráfica RStudio, el uso de paquetes adicionales y análisis de datos.

Se buscará sentar una base sólida para que una vez terminado el tutorial exista una clara idea de cómo plantear un problema de análisis de datos y empezar a resolverlo usando el lenguaje de programación R como herramienta, así mismo se incentiva la búsqueda individual de resultados y la solución de problemas, errores en el script con el uso de plataformas digitales como Stack Overflow.²

iii. Contenido

Los temas que se cubrirán con este tutorial son:

1. Notas del programador
2. Introducción al análisis de datos para la Inteligencia de Negocios
 - a. Conceptos clave en el análisis de datos
3. R y RStudio, instalación e introducción
 - a. * R y RStudio
 - b. * Tipos de Objetos
 - c. * Operadores lógicos y relacionales
 - d. * Bucles (Loops)

¹ [Udemy](#)
[Analytics Vidhya](#)
[Learn R | Codecademy](#)

² [Stack Overflow](#)

- e. * Lectura y limpieza de Datos
 - f. * Bases de Datos; acceso y generación
 - g. * Exploración de datos
 - h. * Introducción a la visualización de datos
4. Casos de Estudio

1. R y Rstudio, instalación e introducción.

1.1 Definición de programa R.

R³ es gratuito, es un ambiente de programación para cómputo estadístico y graficar datos. Como lenguaje de programación, R provee de una amplia variedad de operaciones estadísticas (modelos lineales y no lineales, pruebas estadísticas, análisis de series de tiempo, y redes neuronales por mencionar algunas) y técnicas gráficas, R es escalable, puede ser usado para pequeñas líneas de código hasta para elaborar gráficas y tableros basados en planteamientos particulares del problema de una manera sencilla para quienes están familiarizados con algún ambiente de programación.

Como medio ambiente de programación y siendo orientado al análisis estadístico, R permite manipular datos, realizar cálculos y graficar dinámicamente⁴.

1.2 Rstudio

Rstudio⁵ es un ambiente integrado de desarrollo, IDE por sus siglas en inglés. En este ambiente integrado, es posible visualizar los objetos con los que se trabaja en R que puede ser una tabla de datos, una conexión a una base de datos, o un modelo estadístico. Rstudio permite al programador interactuar y hacer interactuar a estos objetos a través de comandos de R.

Términos básicos en R y Rstudio

Espacio de Trabajo, es el actual ambiente de R e incluye cualquier objeto definido por el usuario (Vectores, matrices, data frames, listas, funciones). Al finalizar una sesión el usuario podrá guardar una imagen del actual *espacio de trabajo* y se cargará la siguiente vez que se inicie R.

IDE Rstudio El ambiente integrado de desarrollo (IDE), está dividido en 4 ventanas que se integran para manejar los objetos dentro del espacio de trabajo y desarrollar modelos,

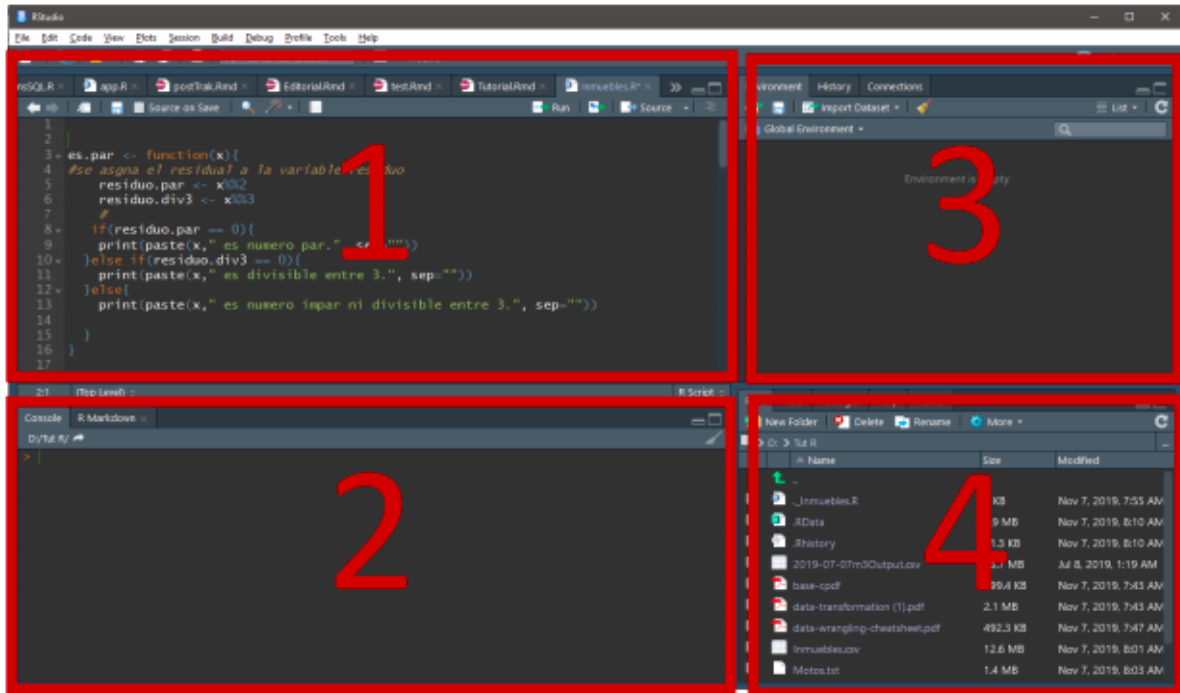
³ [R: What is R?](#)

⁴ <https://simplystatistics.org/2018/07/12/use-r-keynote-2018/>

⁵ [Download RStudio | The Popular Open-Source IDE from Posit](#)
[Download RStudio | The Popular Open-Source IDE from Posit](#)

gráficos y tableros interactivos con R: (Ver figura 1.1)

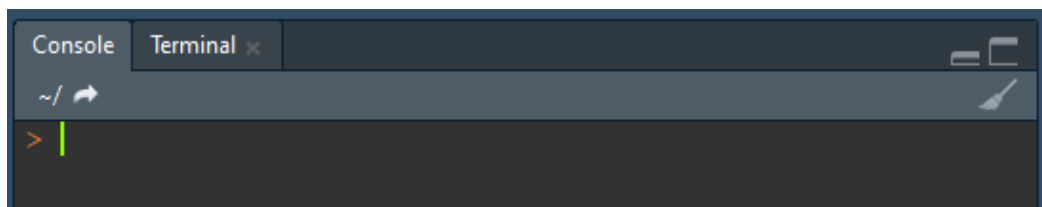
1. Source Editor - Ventana donde están escritos los comandos, se editan los scripts de R, Rstudio ofrece autocompletar cuando se presiona la tecla tabulador.
2. R Console.- Ventana donde se indica una instrucción y se recibe una respuesta de forma interactiva; escribir instrucción, pulsar enter y recibir el resultado.
3. Environment.- Ver variables, valores y objetos cargados en RStudio, permite conocer datos cargados y conexiones a bases de datos.
4. Explorador.- Ver archivos, gráficas, referencias de las librerías, la carpeta o directorio de trabajo,paquetería instalada y ayuda.



(Ver figura 1.1) Distribución de las 4 vistas descritas.

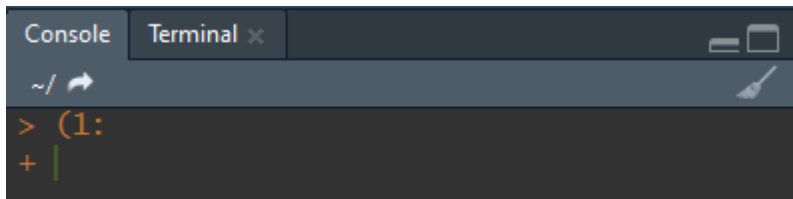
> Prompt

Con el símbolo ">", la consola indica que está lista para ejecutar un comando.(Ver figura 1.2)



(Figura 1.2) Se observa el cursor como una línea verde.

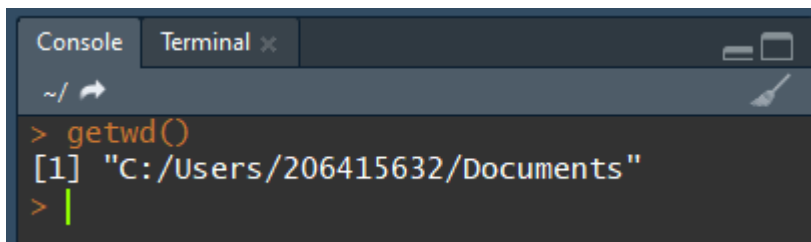
Con el símbolo "+", la consola indica que está esperando más argumentos o comandos.(Ver figura 1.3)



```
Console Terminal x
~/
> (1:
+ |
```

(Figura 1.3) Ahora al inicio de la línea se observa el símbolo “+” seguido del cursor. Directorio de trabajo o working directory es la carpeta en la cual “trabaja” R, esto quiere decir que es la carpeta en donde R va a leer o escribir archivos. Para conocer la ruta de nuestra computadora, se emplea el siguiente comando: (Ver figura 1.4)

`getwd()`

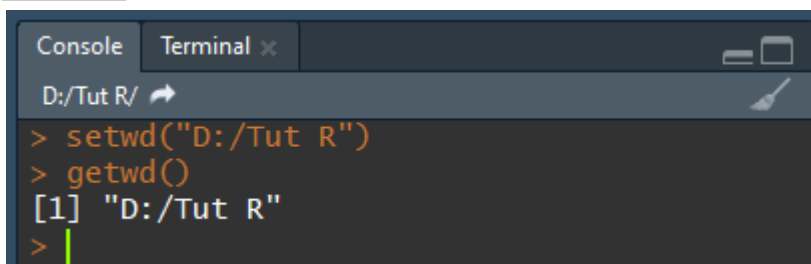


```
Console Terminal x
~/
> getwd()
[1] "C:/Users/206415632/Documents"
> |
```

(Figura 1.4) El resultado es la ruta del sistema operativo desde el disco duro.

Para definir una ruta diferente se usa el siguiente comando:(Ver figura 1.5)

`setwd()`



```
Console Terminal x
D:/Tut R/
> setwd("D:/Tut R")
> getwd()
[1] "D:/Tut R"
> |
```

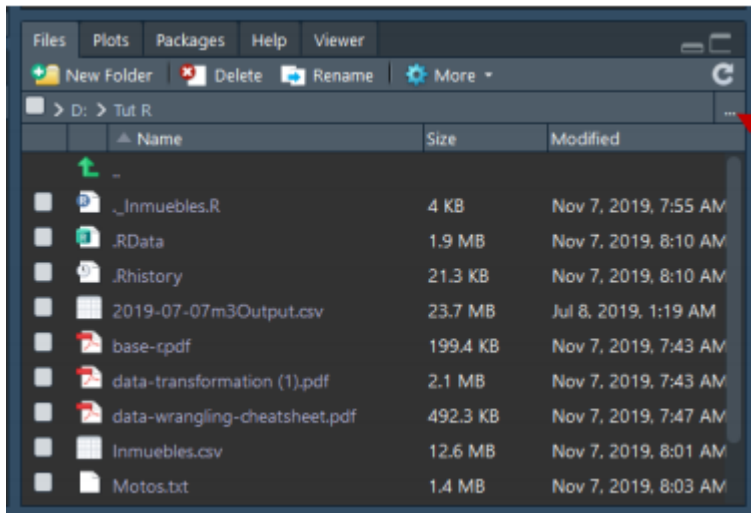
(Figura 1.5) Se vuelve a escribir el comando `getwd()` para comprobar la nueva ruta.

Escribiendo “`getwd()`” se obtiene la dirección del directorio de Windows en el que R está trabajando.

Cambiar directorio de trabajo

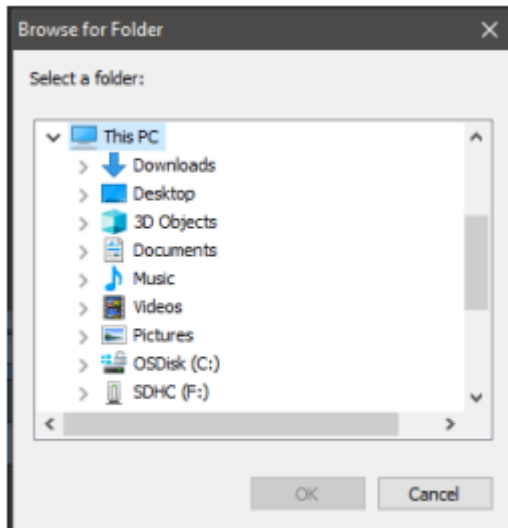
En la interfaz gráfica de RStudio se realiza lo siguiente:
Identificar en R estudio la ventana del explorador:

Dar clic en los 3 puntos mostrados arriba para mostrar el explorador de archivos:(Ver figura 1.6)



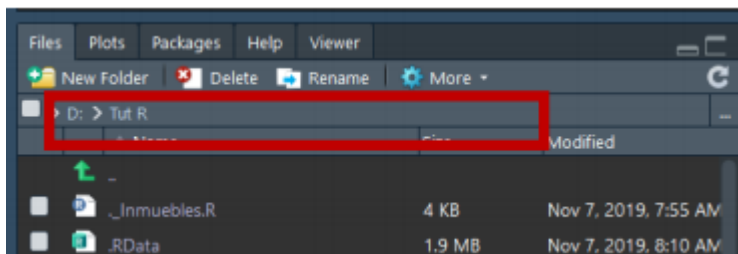
(Figura 1.6) Señalados con la flecha roja.

Seleccionar la carpeta donde están los archivos con los que se trabajará y dar clic en OK.
(Ver figura 1.7)



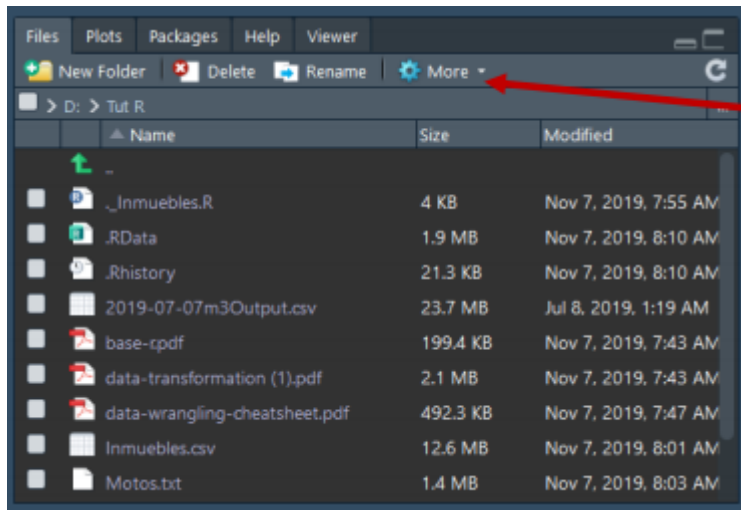
(Figura 1.7) Navegador en windows.

Se observará como la ruta y los archivos que aparecen en la ventana del explorador se actualizan según el contenido de la carpeta seleccionada:(Ver figura 1.8)



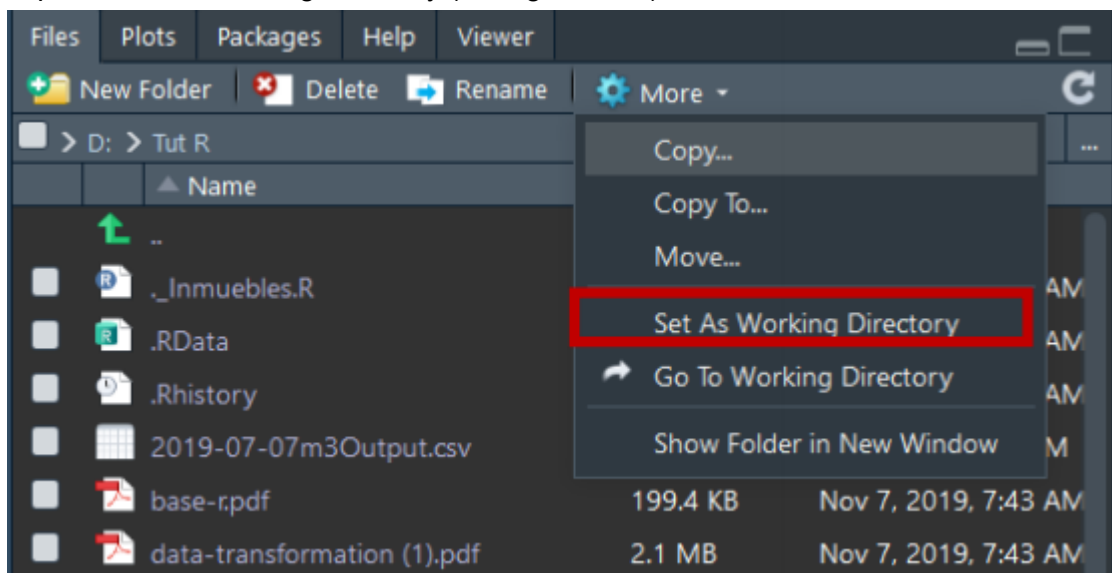
(Figura 1.8) Se marca en rojo la ruta desde el disco duro hasta el directorio de trabajo.

Dar clic en el engrane azul que dice *more*:(Ver figura 1.9)



(Figura 1.9) Este engrane está en la primer pestaña de la ventana del explorador.

Elegir la opción Set As Working Directory:(Ver figura 1.10)



(Figura 1.10)Menú una vez hecho clic en el engrane.

Verificar en la consola que efectivamente se haya cambiado el directorio de trabajo, se escribe el comando `getwd()` en la consola de R:(Ver figura 1.11)

```
> getwd()
[1] "D:/Tut R"
> |
```

(Figura 1.11) Se vuelve a comprobar el nuevo directorio de trabajo.

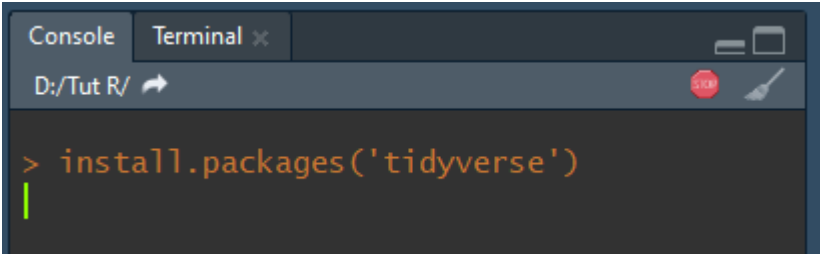
1.3 Instalación de paquetes

Un paquete de R es una colección de funciones, datos y documentación que extiende las capacidades de base R, todas las librerías y paquetes pueden consultarse en el CRAN⁶, también pueden hacerse instalaciones desde Github⁷, al ser software abierto, muchas personas en distintos lugares pueden colaborar, esto acelera el desarrollo de software.

Como ejemplo, instalaremos la librería Tidyverse⁸. Los paquetes de Tidyverse comparten filosofía con la programación en R, y están diseñados para trabajar juntos de manera natural.

Una vez teniendo instalado R y R Studio, se podrá instalar esta paquetería, será necesario escribir lo siguiente en la consola de R:(Ver figura 1.12)

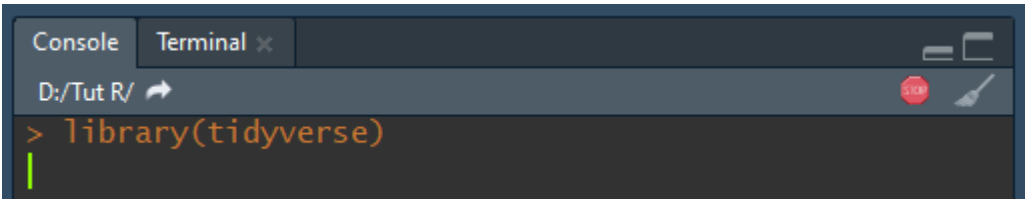
```
install.packages("tidyverse")
```

A screenshot of an R console window. The window has a title bar with 'Console' and 'Terminal x' tabs. Below the title bar, the current directory is shown as 'D:/Tut R/'. The main area of the console is dark with orange text. The command '> install.packages('tidyverse')' is entered, with a green cursor at the end of the line.

(Figura 1.12)Note que pueden ser usadas las comillas dobles “ o comilla simple ‘ .

Cada vez que vayamos a crear un script que ocupe comandos de Tidyverse tendremos que cargar la librería con la siguiente función:(Ver figura 1.13)

```
library(tidyverse)
```

A screenshot of an R console window, similar to the one in Figure 1.12. The title bar shows 'Console' and 'Terminal x'. The directory is 'D:/Tut R/'. The command '> library(tidyverse)' is entered in orange text on a dark background, with a green cursor at the end.

(Figura 1.13)Al cargar la librería podrán ser usadas las funciones contenidas en ella.

Comentarios

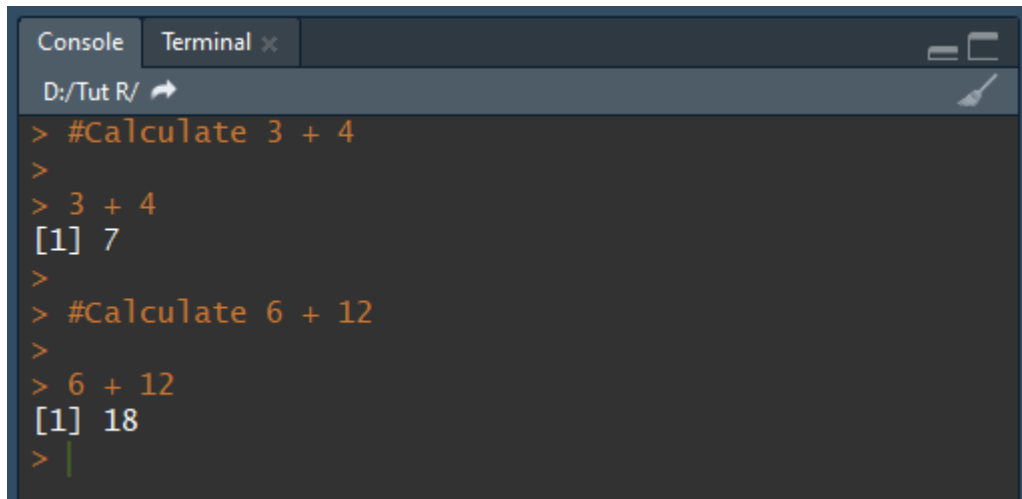
Para poder hacer notas entre líneas de código, basta con iniciar con #. En la sección de abajo hay un par de ejemplos. ¿cuáles líneas son código y cuáles comentarios? (Ver figura 1.14)

⁶ [The Comprehensive R Archive Network](#)

⁷ [Trending R repositories on GitHub today](#)

⁸ [Tidyverse](#)

```
#Calculate 3 + 4
3 + 4
#Calculate 6 + 12
6 + 12
```



The screenshot shows an R console window with the following text:

```
D:/Tut R/
> #Calculate 3 + 4
>
> 3 + 4
[1] 7
>
> #Calculate 6 + 12
>
> 6 + 12
[1] 18
> |
```

(Figura 1.14) Al escribir una línea y presionar enter, se observa que con el comentario no regresa nada la consola mientras que con la suma, la consola regresa el resultado.

1.4 Aritmética con R

En su forma más básica, R puede ser usado como una calculadora. Se consideran los siguientes Operadores Aritméticos.

- Suma: +
- Resta: -
- Multiplicación: *
- División: /
- Exponenciación: ^
- Residuo: %%


Residuo regresa el residuo de la división del número izquierdo entre el de la derecha; 5 Residuo 3 es 2.

Escribir 2^5 en el editor para calcular 2 elevado a 5.

- Escribir $28 \% 6$ para calcular 28 residuo 6.
- Nota el # para comentar líneas de código R. (Ver figura 1.15)

```
#Suma
5 + 5
#Resta
```

```
5 - 5
#Multiplicación
3 * 5
#División
(5 + 5) / 2
#Exponenciación
2^5
#Residuo
28 %% 6
```



The image shows a terminal window with a dark background and light-colored text. The window title is "Terminal" and the current directory is "D:/Tut R/". The terminal displays a series of commands and their outputs, each preceded by a prompt character ">". The commands and their outputs are as follows:

```
> #Suma
>
> 5 + 5
[1] 10
>
> #Resta
>
> 5 - 5
[1] 0
>
> #Multiplicación
>
> 3 * 5
[1] 15
>
> #Division
>
> (5 + 5) / 2
[1] 5
>
> #Exponenciación
>
> 2^5
[1] 32
>
> #Residuo
>
> 28 %% 6
[1] 4
>
```

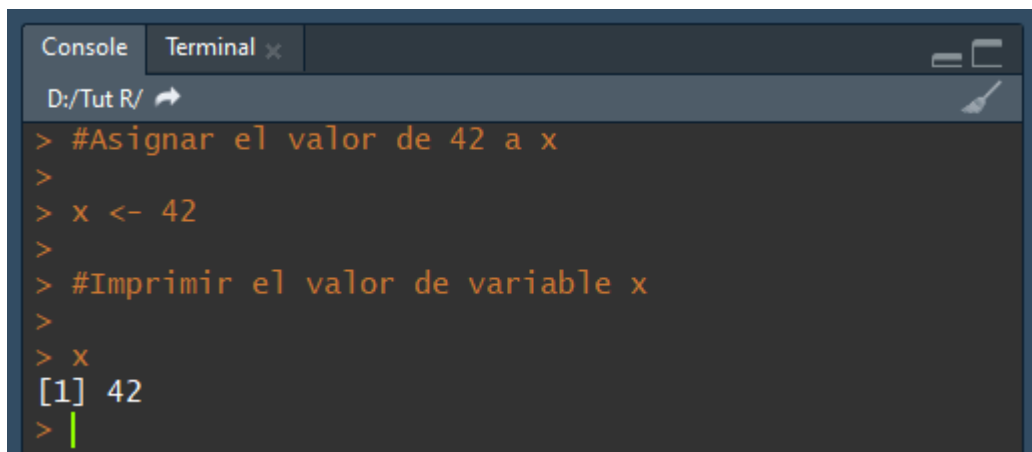
(Figura 1.15) Solución de operaciones aritméticas.

1.5 Asignación de Variables

Un concepto básico es llamado variable; una variable permite almacenar un valor dentro de un objeto en R.

Usando el nombre de la variable permite acceder a dicho valor u objeto. Con el siguiente comando se asignan variables:(Ver figura 1.16)

```
my_var <- 4
#Asignar el valor de 42 a x
x <- 42
#Imprimir el valor de variable x
x
```



```
Console Terminal x
D:/Tut R/ ↗
> #Asignar el valor de 42 a x
>
> x <- 42
>
> #Imprimir el valor de variable x
>
> x
[1] 42
> |
```

(Figura 1.16) cuando se escribe una variable, la consola imprime su valor.

1.6 Tipos de datos básicos en R

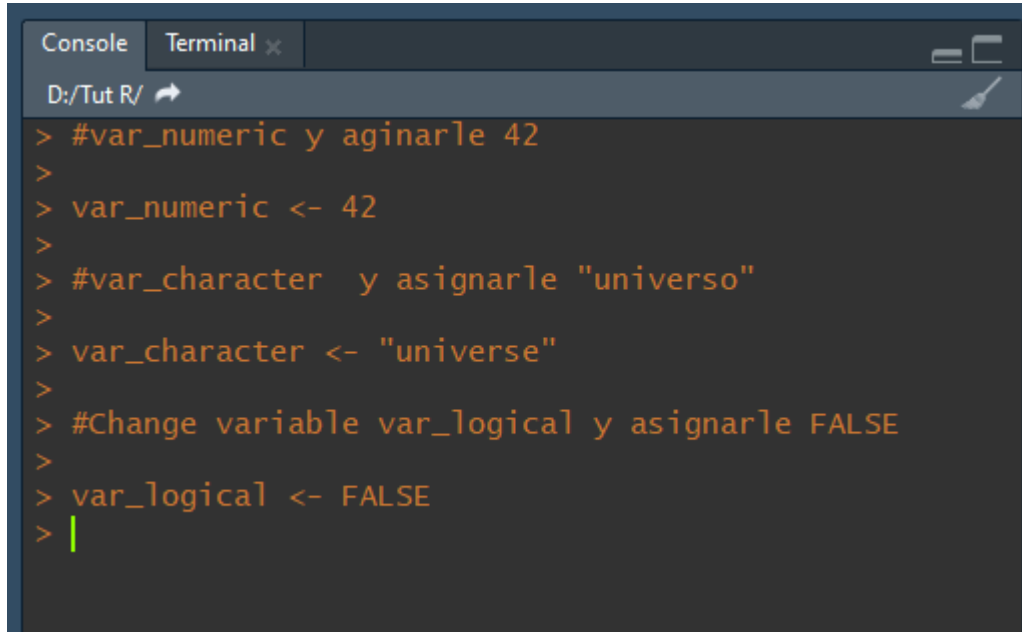
R trabaja con diversos tipos de datos, algunos de los básicos son:

- Números con decimales como 4.5 son llamados **numerics**.
 - Números naturales como 4 son llamados **integers**. Integer también es numérico
 - Valores booleanos como (TRUE or FALSE) son llamados **logical**.
 - Texto es llamado **characters**.
 - Generar una variable `var_numeric` y asignarle 42.
 - Generar una variable `var_character` y asignarle "universo".
- **Nota:** las comillas indican que "universo" es una cadena de texto. ******
- Generar una variable `var_logical` y asignarle FALSE.

(Ver figura 1.17)

```
#var_numeric y asignarle 42
```

```
var_numeric <- 42
#var_character y asignarle "universo"
var_character <- "universo"
#Change variable var_logical y asignarle FALSE
var_logical <- FALSE
```

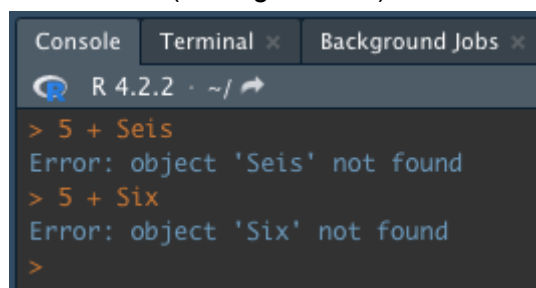


```
Console Terminal x
D:/Tut R/
> #var_numeric y aginarle 42
>
> var_numeric <- 42
>
> #var_character y asignarle "universo"
>
> var_character <- "universe"
>
> #Change variable var_logical y asignarle FALSE
>
> var_logical <- FALSE
> |
```

(Figura 1.17) Al asignar un valor a una variable la consola no regresa nada, solo almacena el objeto en la memoria.

1.7 Identificar tipo de datos

Si uno escribe `5 + "seis"` o `5 + "Six"`... (ver Figura 1.18)

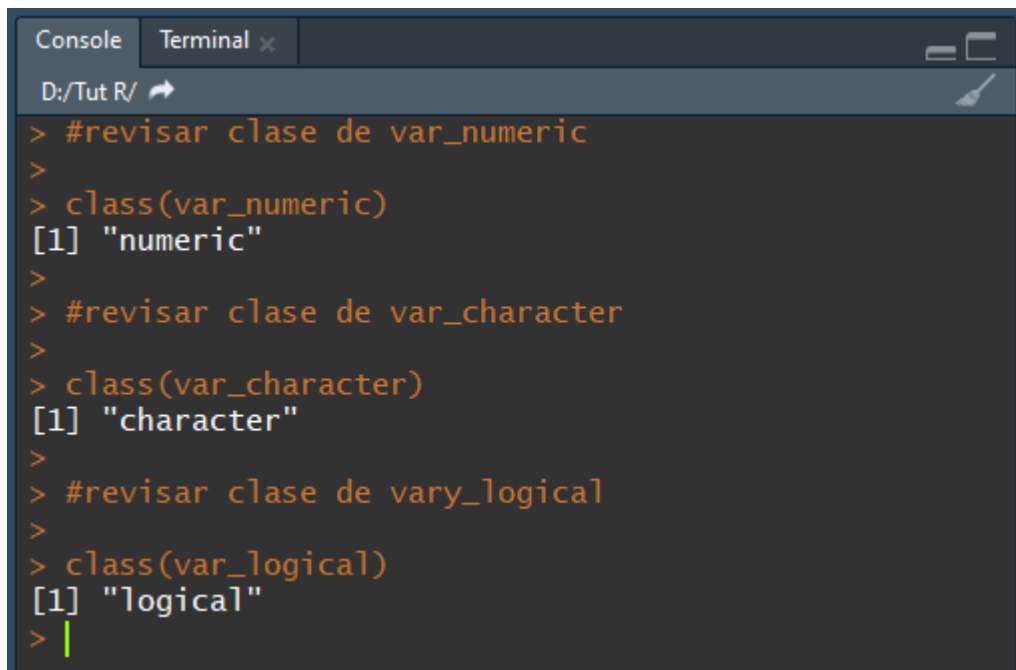


```
Console Terminal x Background Jobs x
R 4.2.2 · ~/
> 5 + Seis
Error: object 'Seis' not found
> 5 + Six
Error: object 'Six' not found
>
```

(Figura 1.18) Errores por escribir "seis" o "Six".

...tendremos un error por una diferencia de tipos de datos en una operación. Este error puede suceder en situaciones simples o complejas. Este error se puede evitar revisando el tipo de dato de una variable de antemano, para ello existe una función llamada `class()` (Ver figura 1.19)

```
#revisar clase de var_numeric
class(var_numeric)
#revisar clase de var_character
class(var_character)
#revisar clase de vary_logical
class(var_logical)
```



```
Console Terminal x
D:/Tut R/
> #revisar clase de var_numeric
>
> class(var_numeric)
[1] "numeric"
>
> #revisar clase de var_character
>
> class(var_character)
[1] "character"
>
> #revisar clase de vary_logical
>
> class(var_logical)
[1] "logical"
> |
```

(Figura 1.19) tal como se asignó a cada variable, la consola muestra el tipo de objeto que es cada una.

1.8 Vectores

Los vectores son matrices unidimensionales que pueden contener datos numéricos, datos de caracteres o datos lógicos. En otras palabras, un vector es una herramienta simple para almacenar datos. Por ejemplo, puede almacenar sus ganancias y pérdidas diarias en los casinos.

En R, crear un vector con la función combine `c()`. Colocar los elementos vectoriales separados por una coma entre paréntesis. Por ejemplo: (Ver figura 1.20)

```
vector_numeric <- c(1, 2, 3)
vector_character <- c("a", "b", "c")
```



```
Console Terminal x
D:/Tut R/ ↵
> vector_numeric <- c (1, 2, 3)
> vector_character <- c ("a", "b", "c")
> |
```

(Figura 1.20) Vector hecho con combinación de elementos.

Una vez que haya creado estos vectores en R, puede usarlos para hacer cálculos.

Generar código para un vector booleano con 3 elementos TRUE, FALSE y TRUE (en ese orden). (Ver figura 1.21)

```
boolean_vector <-c(TRUE,FALSE,TRUE)
```

```
Console Terminal x
D:/Tut R/ ↵
> boolean_vector <-c(TRUE,FALSE,TRUE)
```

(Figura 1.22) Otra combinación de elementos para crear un vector.

Nombrar a un vector

Como analista de datos, es importante tener una visión clara de los datos que se están utilizando. Comprender a qué se refiere cada elemento es, por lo tanto, esencial.

Se puede asignar un nombre a los elementos de un vector con la función `names ()`. Ejemplo:(Ver figura 1.23)

```
mi_vector <- c ("Juan Perez", "jugador de póker")
names (mi_vector) <- c ("Nombre", "Profesión")
```

```
Console Terminal x
D:/Tut R/ ↵
> mi_vector <- c ("Juan Perez", "jugador de póker")
> names (mi_vector) <- c ("Nombre", "Profesión")
> |
```

(Figura 1.23)De forma simple el nombre del vector indica que dato hay escrito.

Este código primero crea un vector `mi_vector` y luego le da un nombre a los dos elementos. Al primer elemento se le asigna el nombre Nombre, mientras que el segundo elemento se etiqueta Profesión. Imprimir los contenidos en la consola produce el siguiente resultado: (Ver figura 1.24)

```
mi_vector
```

```

> mi_vector
      Nombre      Profesión
"Juan Perez" "jugador de póker"
> |

```

(Figura 1.24) Con los vectores nombrados se entiende la información

Para los vectores poker_vector y roulette_vector, asignarles los nombres de los días de la semana (Ver figura 1.25)

```

# Ganancias por Poker Lunes a Viernes
poker_vector <- c(140, -50, 20, -120, 240)
# Ganancias Ruleta Lunes a Viernes
roulette_vector <- c(-24, -50, 100, -350, 10)

```

```

# Asignar nombres de los días a cada uno
names(poker_vector) <- c("Monday", "Tuesday", "Wednesday",
"Thursday", "Friday")
names(roulette_vector) <- c("Monday",
"Tuesday", "Wednesday", "Thursday", "Friday")

```

Podemos brevemente mirar la ventana de ambiente y notar como están ahí todos los objetos que hemos creado hasta este punto:

Object	Value
aa	num [1:4] 1 2 3 4
boolean_vector	logi [1:3] TRUE FALSE TRUE
days_vector	chr [1:5] "Monday" "Tuesday" "Wednesday" "Thursday" "Friday"
mi_vector	Named chr [1:2] "Juan Perez" ...
poker_vector	Named num [1:5] 140 -50 20 -120 240
roulette_vector	Named num [1:5] -24 -50 100 -350 10
total_vector	Named num [1:5] 116 -100 120 -470 250
var_character	"universe"
var_logiac1	FALSE
var_logical	FALSE
var_numeric	42
vector_character	chr [1:3] "a" "b" "c"
vector_numeric	num [1:3] 1 2 3
x	42

(Figura 1.25) Todos los elementos son Valores.

1.9 Cómo nombrar fácilmente una variable

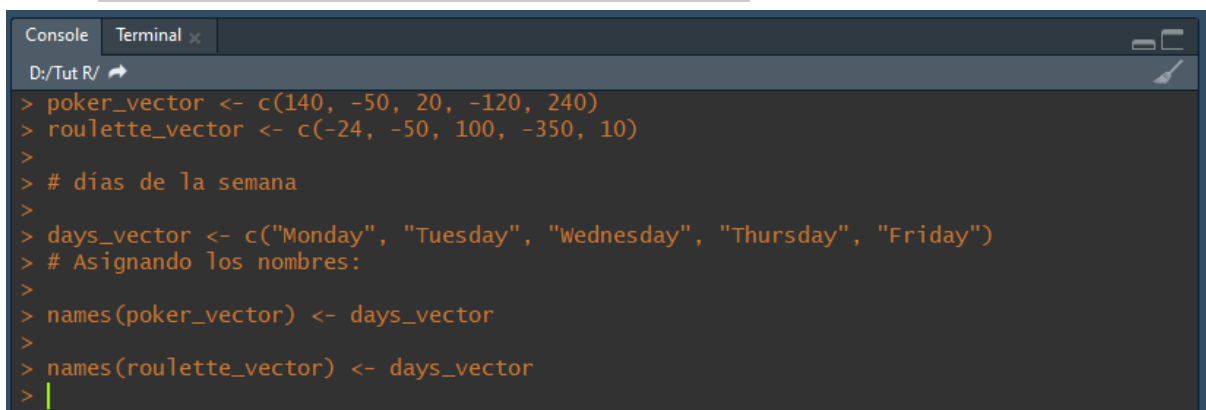
En los ejercicios anteriores probablemente experimentó que es aburrido y frustrante escribir y volver a escribir información, como los días de la semana. Sin embargo, cuando se mira desde una perspectiva más alta, hay una manera más eficiente de hacer esto, a saber, asignar los vectores del día de la semana a una variable

Al igual que lo hizo con las ganancias de póker y ruleta, también puede crear una variable que contenga los días de la semana. De esta manera puedes usarlo y reutilizarlo

Generar un vector que se llame días de la semana y asignarlos como nombres de los vectores de poker y roulette(Ver figura 1.27)

```
poker_vector <- c(140, -50, 20, -120, 240)
roulette_vector <- c(-24, -50, 100, -350, 10)

# días de la semana
days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday",
"Friday")
# Asignando los nombres:
names(poker_vector) <- days_vector
names(roulette_vector) <- days_vector
```

A screenshot of an R console terminal window. The window title is "Terminal" and the current directory is "D:/Tut R/". The terminal shows the following R code being executed line by line: > poker_vector <- c(140, -50, 20, -120, 240); > roulette_vector <- c(-24, -50, 100, -350, 10); > # días de la semana; > days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday"); > # Asignando los nombres:; > names(poker_vector) <- days_vector; > names(roulette_vector) <- days_vector; > The cursor is at the end of the last line. The terminal output is empty, indicating that the code was executed successfully without any printed output.

(Figura 1.27) al crear vectores la consola no imprime nada, solo los guarda en la memoria.

1.10 Calcular totales

Con las ganancias de póker y ruleta como vectores con nombre, puede comenzar a hacer algo de analítica de datos.

¿Cuánto ha sido su ganancia o pérdida general por día de la semana?

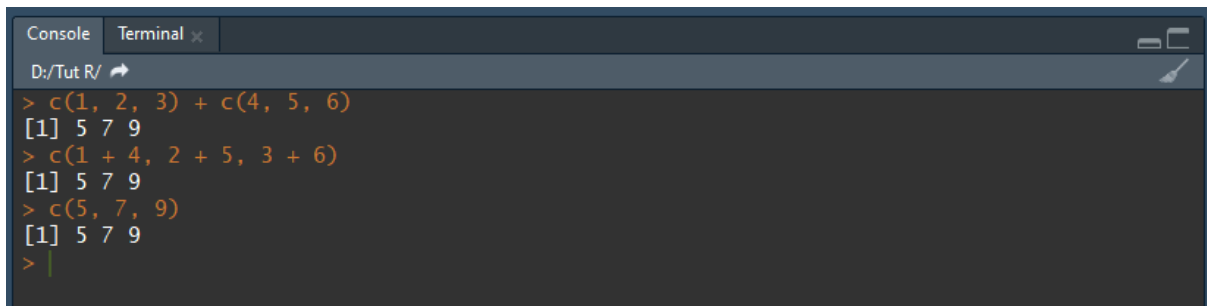
¿Ha perdido dinero durante la semana en total?

¿Está ganando / perdiendo dinero en el póker o en la ruleta?

Para obtener las respuestas, tienes que hacer cálculos aritméticos en vectores.

Es importante saber que si suma dos vectores en R, toma la suma de elementos. Por ejemplo, las siguientes tres declaraciones son completamente equivalentes (Ver figura 1.28)

```
c(1, 2, 3) + c(4, 5, 6)
c(1 + 4, 2 + 5, 3 + 6)
c(5, 7, 9)
```

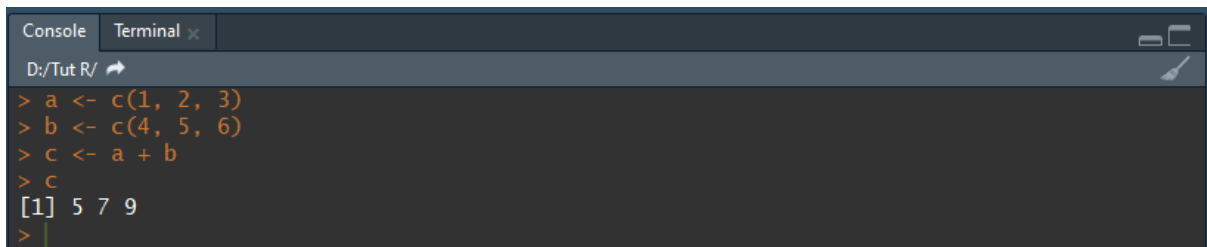


```
Console Terminal x
D:/Tut R/ ↗
> c(1, 2, 3) + c(4, 5, 6)
[1] 5 7 9
> c(1 + 4, 2 + 5, 3 + 6)
[1] 5 7 9
> c(5, 7, 9)
[1] 5 7 9
> |
```

(Ver figura 1.28)

Naturalmente, esto se puede hacer con variables que representan vectores:(Ver figura 1.29)

```
a <- c(1, 2, 3)
b <- c(4, 5, 6)
c <- a + b
c
```



```
Console Terminal x
D:/Tut R/ ↗
> a <- c(1, 2, 3)
> b <- c(4, 5, 6)
> c <- a + b
> c
[1] 5 7 9
> |
```

(Ver figura 1.29) Resultado de una suma de vectores.

¿Cuánto ha sido su ganancia o pérdida general por día de la semana?

```
total_vector <- poker_vector+roulette_vector
```

¿Ha perdido dinero durante la semana en total?

```
sum(total_vector)
```

¿Está ganando / perdiendo dinero en el póker o en la ruleta?

```
sum(poker_vector)
sum(roulette_vector)
```

(Ver figura 1.30)

```
Console Terminal x
D:/Tut R/ ↗
> #¿Cuánto ha sido su ganancia o pérdida general por día de la semana?
> total_vector <- poker_vector+roulette_vector
>
> #¿Ha perdido dinero durante la semana en total?
> sum(total_vector)
[1] -84
> #¿Está ganando / perdiendo dinero en el póker o en la ruleta?
>
> sum(poker_vector)
[1] 230
> sum(roulette_vector)
[1] -314
> |
```

(Ver figura 1.30) Es fácil ver que hay pérdidas en la ruleta.

1.11 Selección de vectores

Para seleccionar elementos de un vector (y matrices, *dataframes* y otros objetos), puede usar corchetes. Entre corchetes, se indica qué elementos seleccionar. Por ejemplo:

Para seleccionar el primer elemento del vector, escriba `poker_vector [1]`. Para seleccionar el segundo elemento del vector, escriba `poker_vector [2]`, etc.

Observe que el primer elemento en un vector tiene índice 1, no 0 como en muchos otros lenguajes de programación.

Asignar los resultados de Poker en Miércoles a la variable `poker_wednesday`. (Ver figura 1.31)

```
poker_vector
roulette_vector
# Definir una nueva variable
poker_wednesday <- poker_vector[3]
```

```
Console Terminal x
D:/Tut R/ ↗
> poker_vector
  Monday  Tuesday Wednesday  Thursday  Friday
    140     -50         20     -120     240
>
> roulette_vector
  Monday  Tuesday Wednesday  Thursday  Friday
   -24     -50         100     -350     10
>
> # Define a new variable based on a selection
>
> poker_wednesday <- poker_vector[3]
> |
```

(Figura 1.31) El tercer elemento del vector Poker, es el ingreso del miércoles.

1.12 Seleccionar múltiples elementos de un vector

Para seleccionar múltiples elementos de un vector, puede agregar corchetes al final del mismo. Puede indicar entre paréntesis qué elementos deben seleccionarse. Por ejemplo: suponga que desea seleccionar el primer y el quinto día de la semana: use el vector `c(1, 5)` entre corchetes. Por ejemplo, el siguiente código selecciona el primer y quinto elemento de `poker_vector`:

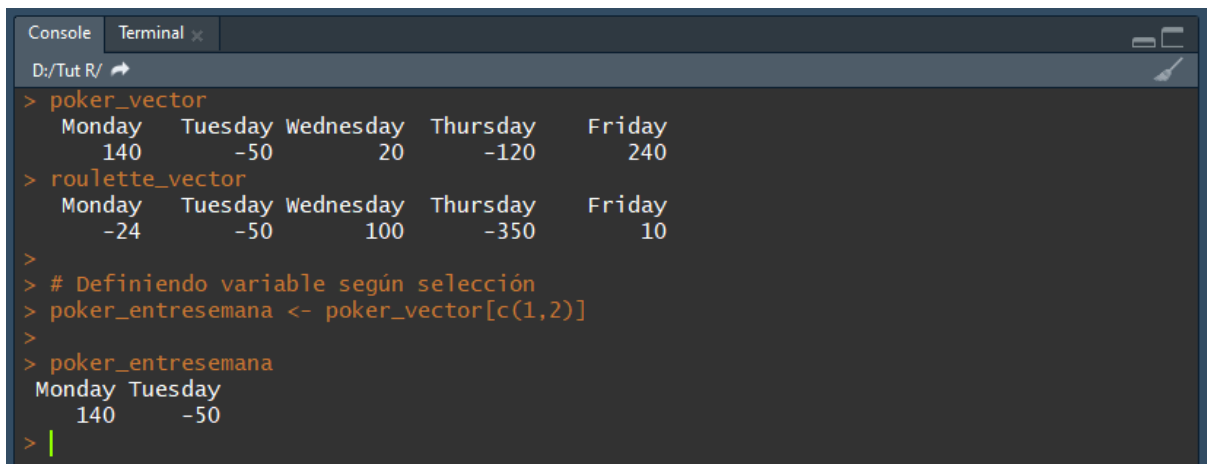
```
poker_vector[c(1, 5)]
```

Asignar los resultados de póker del martes, miércoles y jueves a la variable `poker_entresemana`. (Ver figura 1.33)

```
poker_vector
roulette_vector

# Definiendo variable según selección
poker_entresemana <- poker_vector[c(1,2)]

poker_entresemana
```



```
Console Terminal x
D:/Tut R/
> poker_vector
Monday Tuesday Wednesday Thursday Friday
140 -50 20 -120 240
> roulette_vector
Monday Tuesday Wednesday Thursday Friday
-24 -50 100 -350 10
>
> # Definiendo variable según selección
> poker_entresemana <- poker_vector[c(1,2)]
>
> poker_entresemana
Monday Tuesday
140 -50
> |
```

(Figura 1.33) Al haber escrito `c(1,2)` sólo están los datos de lunes y martes.

1.13 Seleccionar múltiples elementos de un vector (versión corta)

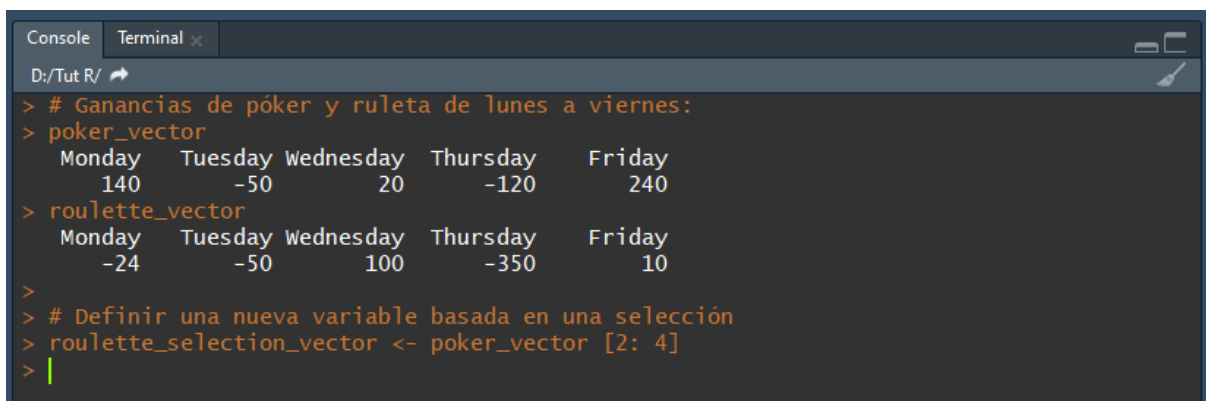
Seleccionar múltiples elementos de `poker_vector` con `c(2, 3, 4)` no es muy conveniente. Existe una forma más fácil de hacer esto: `c(2, 3, 4)` se puede abreviar a `2:4`, lo que genera un vector con todos los números naturales de 2 a 4.

Entonces, otra forma de encontrar los resultados de mitad de semana es `poker_vector[2:4]`. Observe cómo se coloca el vector `2:4` entre corchetes para seleccionar el elemento 2 hasta 4.

Asignar a `roulette_selection_vector` los resultados de la ruleta de martes a viernes; utiliza: si facilita las cosas.(Ver figura 1.34)

```
# Ganancias de póker y ruleta de lunes a viernes:
poker_vector
roulette_vector

# Definir una nueva variable basada en una selección
roulette_selection_vector <- poker_vector [2: 4]
```



```
Console Terminal x
D:/Tut R/
> # Ganancias de póker y ruleta de lunes a viernes:
> poker_vector
  Monday  Tuesday Wednesday  Thursday  Friday
    140    -50         20    -120     240
> roulette_vector
  Monday  Tuesday Wednesday  Thursday  Friday
    -24    -50         100    -350     10
>
> # Definir una nueva variable basada en una selección
> roulette_selection_vector <- poker_vector [2: 4]
> |
```

(Ver figura 1.34)

1.14 Seleccionar múltiples elementos de un vector (con nombres)

Otra forma de abordar el ejercicio anterior es mediante el uso de los nombres de los elementos del vector (lunes, martes, ...) en lugar de sus posiciones numéricas. Por ejemplo, `poker_vector ["Monday"]`.

Se seleccionó el primer elemento de `poker_vector` ya que "Monday" es el nombre de ese primer elemento.

Tal como lo hizo en el ejercicio anterior con números, también puede usar los nombres de elementos para seleccionar varios elementos, por ejemplo:(Ver figura 1.35)

```
poker_vector [c ("lunes", "martes")]
```

```
Console Terminal x
D:/Tut R/
> poker_vector [c ("Monday", "Tuesday")]
Monday Tuesday
140      -50
> |
```

(Figura 1.35) de manera equivalente a la notación numérica, es posible elegir elementos según el nombre del vector.

Seleccionar los 3 primeros elementos de cada uno usando los nombres de las variables "Monday", "Tuesday" and "Wednesday". Asignar el valor a un nuevo vector.
Calcular el promedio de los valores con la función mean() (Ver figura 1.36)

```
# Ganancias de póker y ruleta de lunes a viernes:
poker_vector
roulette_vector

# Seleccionar resultados de poker para Monday, Tuesday and
Wednesday
poker_start <- poker_vector[c("Monday", "Tuesday", "Wednesday")]

# Calcular el promedio
mean(poker_start)
```

```
Console Terminal x
D:/Tut R/
> # Ganancias de póker y ruleta de lunes a viernes:
> poker_vector
Monday Tuesday Wednesday Thursday Friday
140      -50          20      -120      240
> roulette_vector
Monday Tuesday Wednesday Thursday Friday
-24      -50          100      -350      10
>
> # Seleccionar resultados de poker para Monday, Tuesday and Wednesday
> poker_start <- poker_vector[c("Monday", "Tuesday", "Wednesday")]
>
> # Calcular el promedio
> mean(poker_start)
[1] 36.66667
> |
```

(Figura 1.36)

1.15 Operadores relacionales

Los operadores relacionales permiten observar la relación entre un objeto y otro.

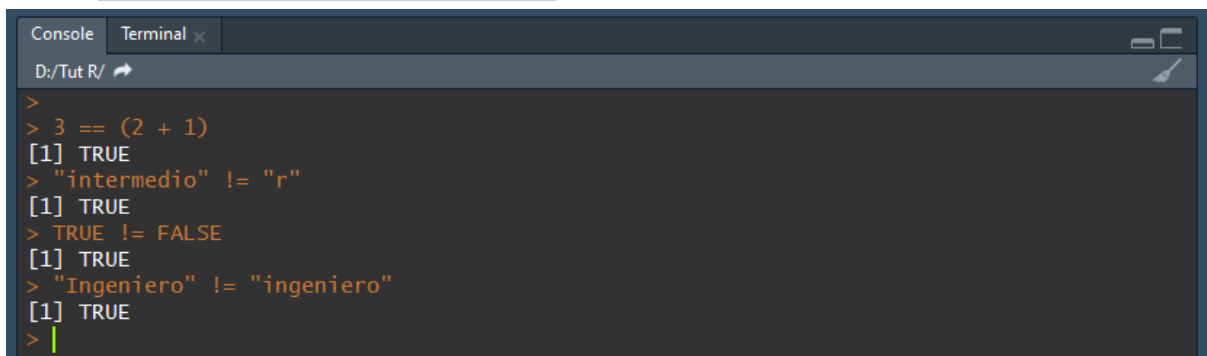
- Revisar si dos objetos son iguales (==) o diferentes (!=).

- Strings
 - Numbers
 - logicals
 - others
- Revisar si es mayor o menor con < >, >= <=.
 - En caracteres el default es orden alfabético.
 - En pruebas lógicas es True > False.
- Es posible comparar valores y vector VS vector.

Igualdad

Comparación más básica (Ver figura 1.37)

```
3 == (2 + 1)
"intermedio" != "r"
TRUE != FALSE
"Ingeniero" != "ingeniero"
```



```
Console Terminal x
D:/Tut R/
>
> 3 == (2 + 1)
[1] TRUE
> "intermedio" != "r"
[1] TRUE
> TRUE != FALSE
[1] TRUE
> "Ingeniero" != "ingeniero"
[1] TRUE
> |
```

(Figura 1.37) La consola de R imprime el resultado de la evaluación de un operador lógico.

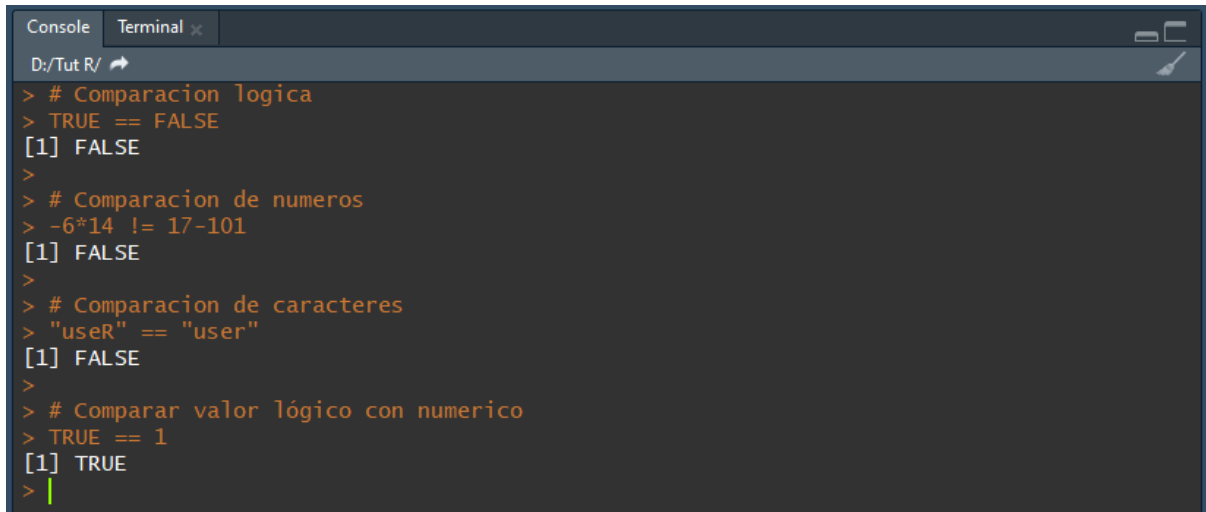
- Escribir en R si TRUE == FALSE.
- Revisar si -6 * 14 es diferente a: 17 - 101.
- Comparar cadenas de caracteres. Con R determine si "useR" y "user" son iguales.
- Descubrir que sucede si se comparan valores lógicos con números: TRUE y 1 son iguales? (Ver figura 1.38)

```
# Comparacion logica
TRUE == FALSE
```

```
# Comparacion de numeros
-6*14 != 17-101
```

```
# Comparacion de caracteres
"useR" == "user"
```

```
# Comparar valor lógico con numerico
TRUE == 1
```



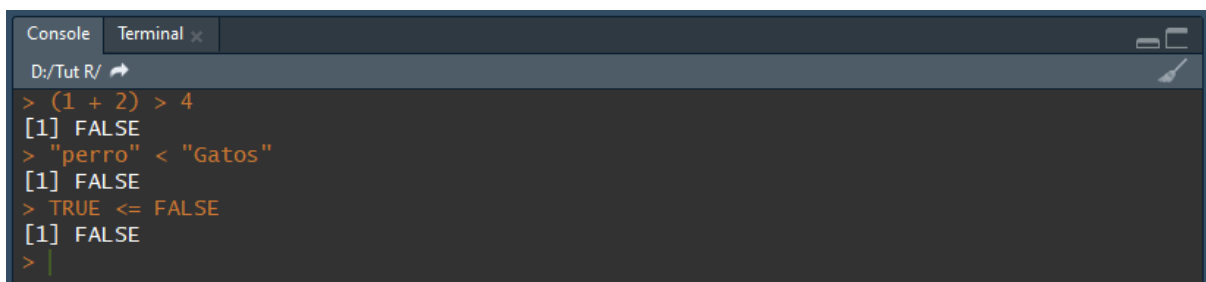
```
Console Terminal x
D:/Tut R/
> # Comparacion logica
> TRUE == FALSE
[1] FALSE
>
> # Comparacion de numeros
> -6*14 != 17-101
[1] FALSE
>
> # Comparacion de caracteres
> "user" == "user"
[1] FALSE
>
> # Comparar valor lógico con numerico
> TRUE == 1
[1] TRUE
> |
```

(Figura 1.38) Se puede apreciar también que R sigue la jerarquía de operaciones aritméticas.

1.16 Mayor y Menor que

Observar las siguientes expresiones en R, todas evalúan como FALSE: (Ver figura 1.39)

```
(1 + 2) > 4
"perro" < "Gatos"
TRUE <= FALSE
```



```
Console Terminal x
D:/Tut R/
> (1 + 2) > 4
[1] FALSE
> "perro" < "Gatos"
[1] FALSE
> TRUE <= FALSE
[1] FALSE
> |
```

(Figura 1.39) Es posible saber si perro es mayor que Gatos.

Al comparar caracteres, R determina la relación mayor que basado en orden alfabético. Además, TRUE es 1 y FALSE es 0. Por lo tanto, FALSE < TRUE es TRUE.

- ¿Cuál es el resultado de cada escenario?
- $-6 * 5 + 2$ es mayor que o igual que $-10 + 1$.
- "raining" es menor que o igual a "raining dogs".
- TRUE es mayor que FALSE.
- Con operadores relacionales, encontrar la respuesta lógica, i.e. TRUE or FALSE
- En qué días las visitas a los perfiles de LinkedIn excedieron 15?

- ¿Cuándo se vio 5 veces o menos?
 - ¿Cuándo se veía más LinkedIn que Facebook?
- (Ver figura 1.40)

```
# 1
```

```
# Comparacion de numeros
```

```
-6*5+2 >= -10 + 1
```

```
# Comparación de caracteres
```

```
"Ingeniero" <= "Ingeniero Industrial"
```

```
# Comparacion de valores logicos
```

```
TRUE > FALSE
```

```
# 2
```

```
# Se han creado los vectores de LinkedIn y Facebook
```

```
linkedin <- c(16, 9, 13, 5, 2, 17, 14)
```

```
facebook <- c(17, 7, 5, 16, 8, 13, 14)
```

```
# Días populares
```

```
linkedin > 15
```

```
# Días no populares
```

```
linkedin <= 5
```

```
# LinkedIn más popular que Facebook
```

```
linkedin > facebook
```

```
Console Terminal x
D:/Tut R/
> # 1
> # Comparacion de numeros
> -6*5+2 >= -10 + 1
[1] FALSE
>
> # Comparacion de caracteres
> "Ingeniero" <= "Ingeniero Industrial"
[1] TRUE
>
> # Comparacion de valores logicos
> TRUE > FALSE
[1] TRUE
>
> # 2
> # Se han creado los vectores de LinkedIn y Facebook
> linkedin <- c(16, 9, 13, 5, 2, 17, 14)
> facebook <- c(17, 7, 5, 16, 8, 13, 14)
>
> # Dias populares
> linkedin > 15
[1] TRUE FALSE FALSE FALSE FALSE TRUE FALSE
>
> # Dias no populares
> linkedin <= 5
[1] FALSE FALSE FALSE TRUE TRUE FALSE FALSE
>
> # LinkedIn más popular que Facebook
> linkedin > facebook
[1] FALSE TRUE TRUE FALSE FALSE TRUE FALSE
> |
```

(Figura 1.40) Es posible conocer en qué momentos de la semana se usa más una red social que otra.

1.17 Operadores Lógicos

Son usados para cambiar o combinar resultados de los operadores relacionales.

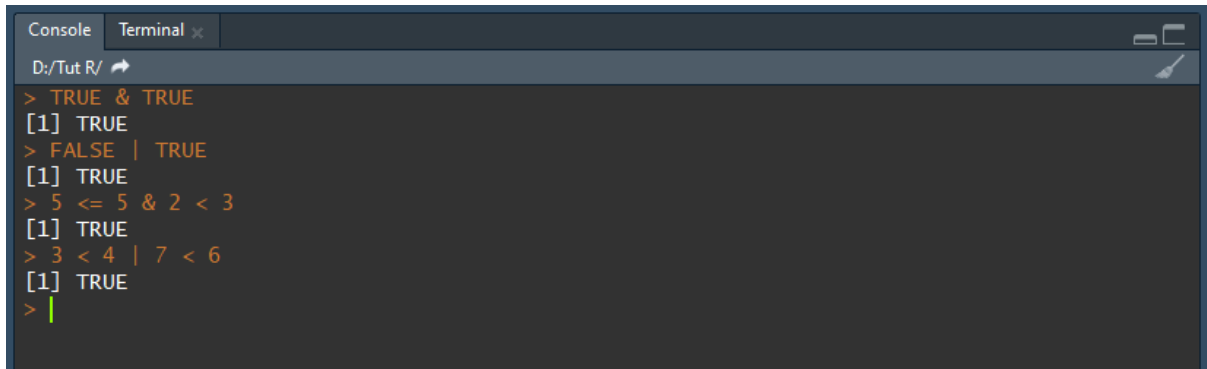
- AND operator &
 - Compara 2 valores lógicos y regresa TRUE si ambos valores lógicos son TRUE.
- OR operator |
 - Compara 2 valores lógicos y regresa TRUE si al menos uno de los valores lógicos es TRUE.
- NOT operator !
 - Valor negativo del valor lógico que antecede..

& and |

Observar cómo los operadores lógicos funcionan con las relaciones lógicas: (Ver figura 1.41)

```
TRUE & TRUE
```

```
FALSE | TRUE
5 <= 5 & 2 < 3
3 < 4 | 7 < 6
```



```
Console Terminal x
D:/Tut R/
> TRUE & TRUE
[1] TRUE
> FALSE | TRUE
[1] TRUE
> 5 <= 5 & 2 < 3
[1] TRUE
> 3 < 4 | 7 < 6
[1] TRUE
> |
```

(Figura 1.41) Con estos operadores lógicos es posible comparar objetos.

Importante: $3 < x < 7$

No funcionará, sin embargo $3 < x \& x < 7$ sí.

- Resolver lo siguiente con la variable last:
- last es menor a 5 o mayor a 10?
- last está entre 15 y 20, excluyendo 15 pero incluyendo 20? (Ver figura 1.42)

```
# 1
linkedin <- c(16, 9, 13, 5, 2, 17, 14)

# tail(vector, number of values)
last <- tail(linkedin, 1)
last

# Is last under 5 or above 10?
last < 5 | last > 10

# Is last between 15 (exclusive) and 20 (inclusive)?
last > 15 & last <= 20
```

```
Console Terminal x
D:/Tut R/
> # 1
> linkedin <- c(16, 9, 13, 5, 2, 17, 14)
>
> # tail(vector, number of values)
> last <- tail(linkedin, 1)
> last
[1] 14
>
> # Is last under 5 or above 10?
> last < 5 | last > 10
[1] TRUE
>
> # Is last between 15 (exclusive) and 20 (inclusive)?
> last > 15 & last <= 20
[1] FALSE
>
> |
```

(Figura 1.42) Es posible hacer simples enunciados algebraicos.

1.18 El enunciado if

(Ver figura 1.43)

Sintaxis:

```
if (condition) {
    expr
}
```

```
Console Terminal x
D:/Tut R/
> if (1 != 2+2){
+ print(TRUE)
+ }
[1] TRUE
>
> |
```

(Figura 1.43) Ejemplo de sintaxis del enunciado.

- Escribir una condición if que imprima "Mostrando información de LinkedIn" si la variable medio es "LinkedIn".
 - Escribir una condición if que diga "eres popular!" si la variable num_views es mayor a 15.
- (Ver figura 1.44)

```
# Variables
medium <- "LinkedIn"
num_views <- 14

# Examinar la condición if para medium
if (medium == "LinkedIn") {
    print("Mostrando información de LinkedIn ")
}
```

```
# Write the if statement for num_views
if (num_views > 15) {
  print("eres popular!")
}
```

```
Console Terminal x
D:/Tut R/
> # Variables
> medium <- "LinkedIn"
> num_views <- 14
>
> # Examinar la condicion if para medium
> if (medium == "LinkedIn") {
+   print("Mostrando informacion de LinkedIn ")
+ }
[1] "Mostrando informacion de LinkedIn "
>
> # Write the if statement for num_views
> if (num_views > 15) {
+   print("eres popular!")
+ }
>
```

(Figura 1.44) Se observa que en el segundo enunciado la consola no imprime nada dado que el resultado de la evaluación es FALSE.

Agregar else

Solo puede usarse la expresión else en combinación con una expresión if. La expresión else no requiere una condición, el código que contiene la expresión es ejecutado si todas las expresiones if anteriores son FALSE.(Ver figura 1.45)

```
if (condition) {
  expr1
} else {
  expr2
}
```

```
Console Terminal x
D:/Tut R/
> if (1 != 2+2){
+ print(TRUE)
+ } else {
+ print(FALSE)
+ }
[1] TRUE
>
```

(Figura 1.45) Es importante que *else* esté en el mismo renglón que el corchete de cierre de la expresión if.

Agregar una expresión else a las expresiones anteriores de modo qué:

- Se imprima "Medio Desconocido" cuando no sea "LinkedIn".
- Imprimir "No eres popular " cuando no se cumpla la condición num_views.

(Ver figura 1.46)

```

# Variables related to your last day of recordings
medium <- "LinkedIn"
num_views <- 14

# Estructura de Control para medium
if (medium == "LinkedIn") {
  print("Mostrando información de LinkedIn")
} else{
  print("Medio Desconocido")
}

# Estructura de Control para medium
if (num_views > 15) {
  print("eres popular!")
} else{
  print("No eres popular ")
}

```

```

Console Terminal x
D:/Tut R/
>
> # Variables related to your last day of recordings
> medium <- "LinkedIn"
> num_views <- 14
>
> # Estructura de Control para medium
> if (medium == "LinkedIn") {
+   print("Mostrando informacion de LinkedIn")
+ } else{
+   print("Medio Desconocido")
+ }
[1] "Mostrando informacion de LinkedIn"
>
> # Estructura de Control para medium
> if (num_views > 15) {
+   print("eres popular!")
+ } else{
+   print("No eres popular ")
+ }
[1] "No eres popular "
>

```

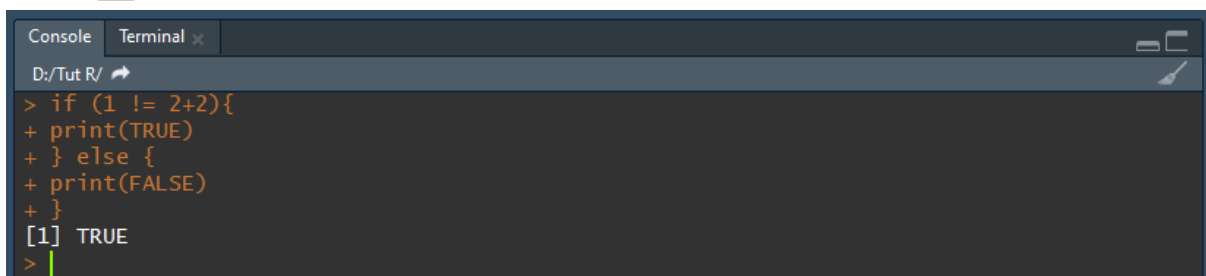
(Figura 1.46) El corchete de else se ejecuta solo si la expresión es evaluada como FALSE

1.19 else if

La expresión else if permite mayor estructura y control. Se pueden agregar tantos sean necesarios, hay que considerar que una vez que una expresión regrese TRUE y se ejecuten

sus expresiones, ya no se seguirán evaluando el resto de las expresiones siguientes. (Ver figura 1.47)

```
#if (condition1) {  
#  expr1  
#} else if (condition2) {  
#  expr2  
#} else if (condition3) {  
#  expr3  
#} else {  
#  expr4  
#}
```



```
Console Terminal x  
D:/Tut R/ ↗  
> if (1 != 2+2){  
+ print(TRUE)  
+ } else {  
+ print(FALSE)  
+ }  
[1] TRUE  
> |
```

(Figura 1.47) Simple enunciado else if.

Al igual que else, es importante que la expresión else if esté en la misma línea que el corchete de cierre.

Agregar una expresión else a las expresiones anteriores de modo que:

- Imprimir "Mostrando Información de Facebook" si medium es igual a "Facebook". Recuerda que R distingue Mayúsculas!
- Imprimir "Numero de vistas es promedio" si num_views está entre 15 (inclusivo) y 10 (exclusivo). (Ver figura 1.48)

```
# Variables related to your last day of recordings  
medium <- "LinkedIn"  
num_views <- 14  
  
# Estructura de control para medium  
if (medium == "LinkedIn") {  
  print("Mostrando información de LinkedIn")  
} else if (medium == "Facebook") {  
  # agregar código para true  
  print("Mostrando información de Facebook ")  
} else {  
  print("Medio Desconocido")  
}
```

```

# Estructura de control para num_views
if (num_views > 15) {
  print("eres popular!")
} else if (num_views <= 15 & num_views > 10) {
  # Add code to print correct string when condition is TRUE
  print("numero de vistas promedio")
} else {
  print("No eres popular")
}

```

```

Console Terminal x
D:/Tut R/
>
> # Variables related to your last day of recordings
> medium <- "LinkedIn"
> num_views <- 14
>
> # Estructura de Control para medium
> if (medium == "LinkedIn") {
+   print("Mostrando informacion de LinkedIn")
+ } else{
+   print("Medio Desconocido")
+ }
[1] "Mostrando informacion de LinkedIn"
>
> # Estructura de Control para medium
> if (num_views > 15) {
+   print("eres popular!")
+ } else{
+   print("No eres popular ")
+ }
[1] "No eres popular "
>

```

(Figura 1.48) Cuando se incluye el enunciado else, este se ejecuta cuando la evaluación es negativa.

1.20 Vectores (continuación)

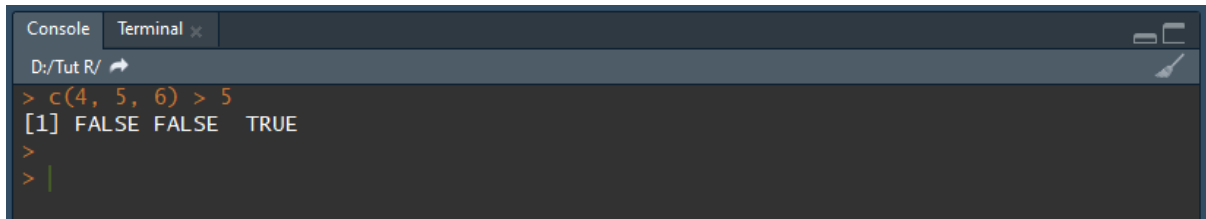
Selección por comparaciones. Con los operadores podemos responder diferentes preguntas de manera más ágil.

Repasando, los operadores que hay son:

- *'<' menor que
- *'>' mayor que
- *'<=' menor o igual
- *'>=' mayor o igual
- *'==' igualdad
- *'!=' diferencia

Como se vio en el capítulo anterior, declarar `6 > 5` devuelve VERDADERO. Lo bueno de R es que puede usar estos operadores de comparación también en vectores. Por ejemplo:(Ver figura 1.49)

```
c(4, 5, 6) > 5
```



```
Console Terminal x
D:/Tut R/
> c(4, 5, 6) > 5
[1] FALSE FALSE TRUE
>
>
```

(Figura 1.49) El comando probó el operador en cada uno de los elementos.

Comprobar qué elementos en `poker_vector` son positivos (es decir, `> 0`) y asígnele a `selection_vector`.

Imprimir `selection_vector` para que pueda inspeccionarse.

La impresión le indica si ganó (VERDADERO) o perdió (FALSO) dinero por cada día. (Ver figura 1.50)

```
# Ganancias de póker y ruleta de lunes a viernes:
poker_vector
roulette_vector

# Que días hubo ganancia en poker
selection_vector <- poker_vector > 0

selection_vector
```



```
Console Terminal x
D:/Tut R/
> # Ganancias de póker y ruleta de lunes a viernes:
> poker_vector
  Monday  Tuesday Wednesday  Thursday  Friday
    140    -50         20     -120     240
> roulette_vector
  Monday  Tuesday Wednesday  Thursday  Friday
   -24    -50         100     -350     10
>
> # Que días hubo ganancia en poker
> selection_vector <- poker_vector > 0
>
> selection_vector
  Monday  Tuesday Wednesday  Thursday  Friday
   TRUE    FALSE     TRUE     FALSE    TRUE
> |
```

(Figura 1.50) Al asignar el resultado de una operación a un objeto e imprimirlo, tenemos un resultado más claro por cada día de la semana.

Comparación de selección (solo vectores verdaderos)

Trabajar con comparaciones facilitará su vida analítica de datos. En lugar de seleccionar un subconjunto de días para investigarse (como antes), simplemente puede pedirle a R que regrese solo aquellos días en los que se dio cuenta de un retorno positivo para el póker.

En los ejercicios anteriores usó:

`selection_vector <- poker_vector > 0` para encontrar los días en los que tuvo un retorno de póker positivo. Ahora, le gustaría saber no solo los días en que ganó, sino también cuánto ganó en esos días.

Puede seleccionar los elementos deseados, colocando `selection_vector` entre los corchetes que siguen a `poker_vector`:

```
poker_vector [selection_vector]
```

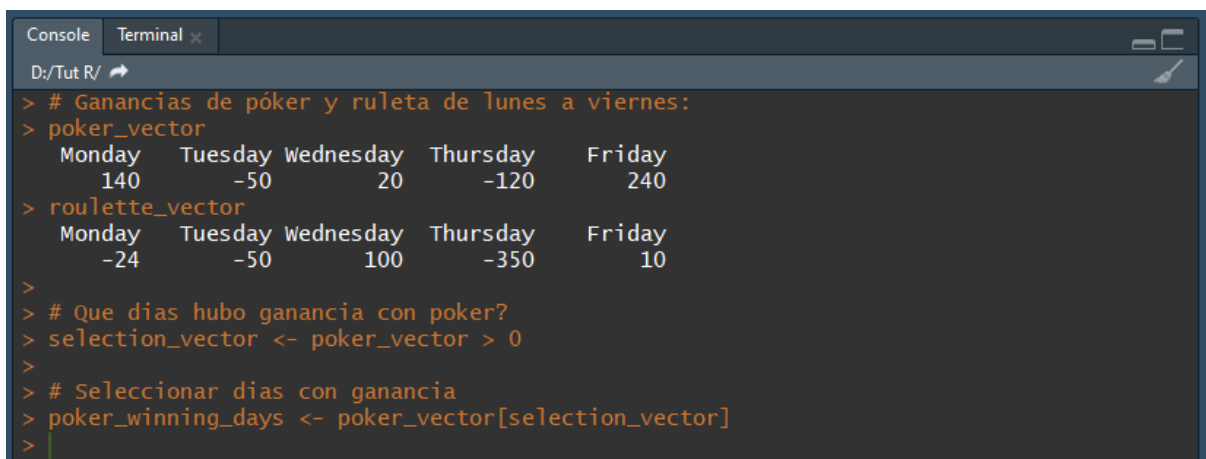
R sabe qué hacer cuando pasa un vector lógico entre corchetes: solo seleccionará los elementos que corresponden a VERDADERO en `selection_vector`.

Instrucciones: Use `selection_vector` entre corchetes para asignar las cantidades que ganó en los días rentables a la variable `poker_winning_days`. (Ver figura 1.52)

```
# Ganancias de póker y ruleta de lunes a viernes:
poker_vector
roulette_vector

# Que días hubo ganancia con poker?
selection_vector <- poker_vector > 0

# Seleccionar días con ganancia
poker_winning_days <- poker_vector[selection_vector]
```



```
Console Terminal x
D:/Tut R/
> # Ganancias de póker y ruleta de lunes a viernes:
> poker_vector
  Monday  Tuesday Wednesday  Thursday  Friday
    140    -50         20     -120     240
> roulette_vector
  Monday  Tuesday Wednesday  Thursday  Friday
   -24    -50         100     -350     10
>
> # Que días hubo ganancia con poker?
> selection_vector <- poker_vector > 0
>
> # Seleccionar días con ganancia
> poker_winning_days <- poker_vector[selection_vector]
>
```

(Figura 1.52) Los días con ganancia en Poker son asignados a la última variable.

1.21 Selección avanzada

Al igual que lo hizo para el póker, también desea saber aquellos días en los que se dio cuenta de un retorno positivo para la ruleta.

Crear la variable `selection_vector`, esta vez para ver si obtuvo ganancias con la ruleta durante diferentes días.

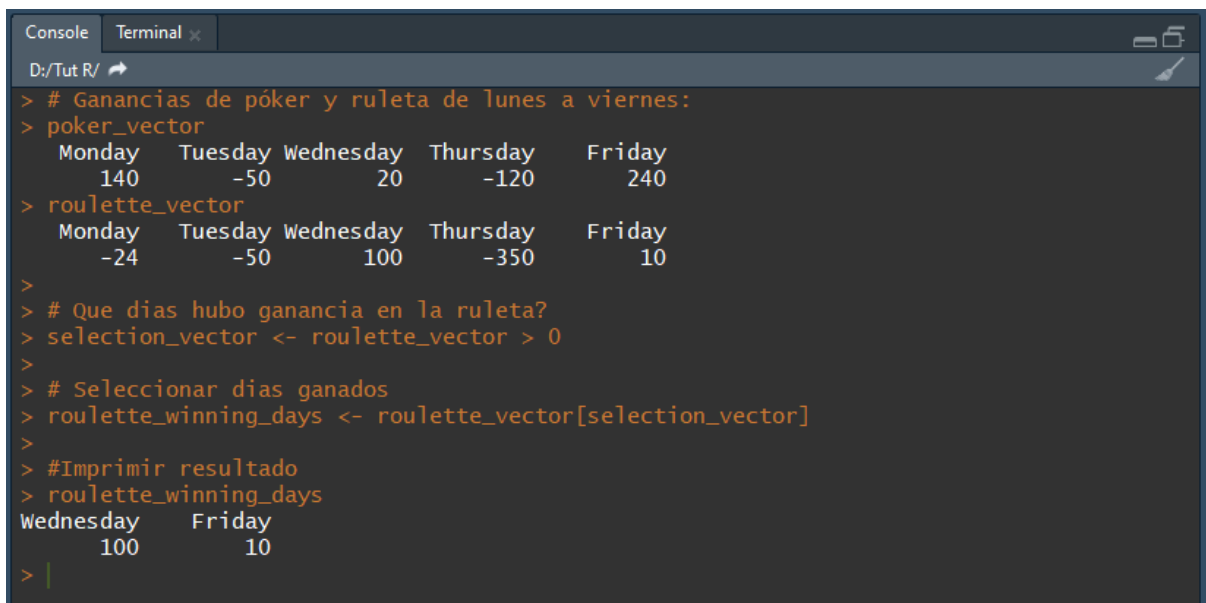
Asignar las cantidades que realizó en los días en que finalizó positivamente para la ruleta a la variable `roulette_winning_days`. Este vector contiene las ganancias positivas de `roulette_vector`.(Ver figura 1.53)

```
# Ganancias de póker y ruleta de lunes a viernes:
poker_vector
roulette_vector

# Que dias hubo ganancia en la ruleta?
selection_vector <- roulette_vector > 0

# Seleccionar dias ganados
roulette_winning_days <- roulette_vector[selection_vector]

#Imprimir resultado
roulette_winning_days
```



```
Console Terminal x
D:/Tut R/
> # Ganancias de póker y ruleta de lunes a viernes:
> poker_vector
  Monday  Tuesday Wednesday  Thursday  Friday
    140    -50         20    -120     240
> roulette_vector
  Monday  Tuesday Wednesday  Thursday  Friday
   -24    -50         100    -350     10
>
> # Que dias hubo ganancia en la ruleta?
> selection_vector <- roulette_vector > 0
>
> # Seleccionar dias ganados
> roulette_winning_days <- roulette_vector[selection_vector]
>
> #Imprimir resultado
> roulette_winning_days
Wednesday  Friday
    100         10
> |
```

(Figura 1.53) Los únicos días que la ruleta tiene ganancia son miércoles y viernes.

1.22 Matrices

Son las de álgebra.

En R, una matriz es una colección de elementos del mismo tipo (numeric, character, or logical) en un arreglo matricial con un número fijo de filas y columnas. Al trabajar con filas y columnas, se considera que la matriz es de 2 dimensiones.

```
matrix(1:9, byrow = TRUE, nrow = 3)
```

En la función `matrix()`:

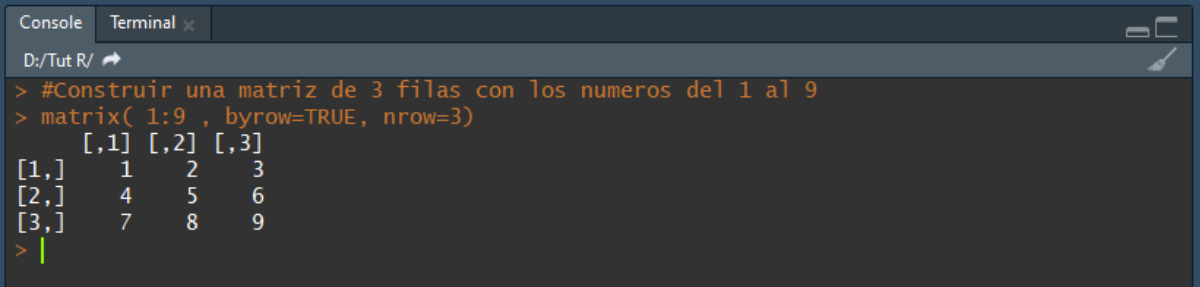
El primer argumento son los elementos que R acomodará matricialmente, Notar que `1:9` es una versión abreviada para un conjunto de números del 1 al 9 `c(1, 2, 3, 4, 5, 6, 7, 8, 9)`.

El argumento `byrow` indica que el arreglo matricial será por renglón. Si quisiéramos que fuera por columnas, se escribe `byrow = FALSE`.

El tercer argumento indica cuántos renglones habrá.

Construir una matriz de 3 filas con los números del 1 al 9 en un arreglo por renglón. (Ver figura 1.54)

```
#Construir una matriz de 3 filas con los números del 1 al 9  
matrix( 1:9 , byrow=TRUE, nrow=3)
```



```
Console Terminal x  
D:/Tut R/ ↗  
> #Construir una matriz de 3 filas con los numeros del 1 al 9  
> matrix( 1:9 , byrow=TRUE, nrow=3)  
      [,1] [,2] [,3]  
[1,]    1    2    3  
[2,]    4    5    6  
[3,]    7    8    9  
> |
```

(Figura 1.54) se observa la matriz generada.

1.23 Nombrar una matriz

Para recordar lo que está almacenado en alguna matriz, es deseable agregar los nombres para las filas o columnas. Esto no solo ayuda a leer los datos, sino que también es útil para seleccionar ciertos elementos de la matriz.

```
rownames (var_matrix) <- row_names_vector  
colnames (var_matrix) <- col_names_vector
```

Seguimos adelante y preparamos dos vectores para usted: `región` y `títulos`. Necesitará estos vectores para nombrar las columnas y filas de `star_wars_matrix`, respectivamente.

Usar `colnames()` para nombrar las columnas de `star_wars_matrix` con el vector de la region.
Usar `rownames()` para nombrar los renglones de `star_wars_matrix` con el vector de titulos.

Imprimir la matriz `star_wars_matrix` para ver el resultado. (Ver figura 1.55)

```
# Box office Star Wars
new_hope <- c(460.998, 314.4)
empire_strikes <- c(290.475, 247.900)
return_jedi <- c(309.306, 165.8)

# Construir matrix
star_wars_matrix <- matrix(c(new_hope, empire_strikes,
return_jedi), nrow = 3, byrow = TRUE)

# Vectores region y titles, usados para nombramiento
region <- c("US", "non-US")
titles <- c("A New Hope", "The Empire Strikes Back", "Return
of the Jedi")

# Nombrar columnas con region
colnames(star_wars_matrix) <- region

# Nombrar renglones con titulos
rownames(star_wars_matrix) <- titles

# Imprimir
star_wars_matrix
```

```

Console Terminal x
D:/Tut R/
> # Box office Star Wars
> new_hope <- c(460.998, 314.4)
> empire_strikes <- c(290.475, 247.900)
> return_jedi <- c(309.306, 165.8)
>
> # Construir matrix
> star_wars_matrix <- matrix(c(new_hope, empire_strikes, return_jedi), nrow = 3, byrow = TRUE)
>
> # Vectores region y titles, usados para nombramiento
> region <- c("US", "non-US")
> titles <- c("A New Hope", "The Empire Strikes Back", "Return of the Jedi")
>
> # Nombrar columnas con region
> colnames(star_wars_matrix) <- region
>
> # Nombrar renglones con titulos
> rownames(star_wars_matrix) <- titles
>
> # Imprimir
> star_wars_matrix
              US non-US
A New Hope    460.998  314.4
The Empire Strikes Back 290.475  247.9
Return of the Jedi    309.306  165.8
>

```

(Figura 1.55) El objeto `Star_wars_matrix` almacena la matriz y los nombres de renglones y columnas.

1.24 Cálculos por renglón

In R, la función `rowSums()` convenientemente calcula la suma de todos los renglones, esto crea un vector:(Ver figura 1.56)

```
rowSums(my_matrix)
```

Instrucciones, calcular la taquilla mundial

```

# Construir matrix
box_office <- c(460.998, 314.4, 290.475, 247.900, 309.306,
165.8)
star_wars_matrix <- matrix(box_office, nrow = 3, byrow = TRUE,
dimnames = list(c("A New Hope",
"The Empire Strikes Back", "Return of the Jedi"), c("US",
"non-US")))

# Calcular taquilla mundial
worldwide_vector <- rowSums(star_wars_matrix)

worldwide_vector

```



```
Console Terminal x
D:/Tut R/
> # Construir matriz
> box_office <- c(460.998, 314.4, 290.475, 247.900, 309.306, 165.8)
> star_wars_matrix <- matrix(box_office, nrow = 3, byrow = TRUE,
+   dimnames = list(c("A New Hope", "The Empire Strikes Back", "
Return of the Jedi"), c("US", "non-US")))
>
> # Calcular taquilla mundial
> worldwide_vector <- rowSums(star_wars_matrix)
>
> worldwide_vector
      A New Hope The Empire Strikes Back      Return of the Jedi
1       775.398                538.375                475.106
> |
```

(Figura 1.56) Resultado de la taquilla de cada película.

1.25 Agregar una columna a una matriz

Puede agregar una columna o varias columnas a una matriz con la función `cbind()`, que combina matrices y / o vectores por columna. Por ejemplo:

```
big_matrix <- cbind(matrix1, matrix2, vector1 ...)
```

Agregar `worldwide_vector` como una nueva columna a la matriz `star_wars_matrix` y asignar a `todo_starwars_matrix`. Usar la función `cbind()`(Ver figura 1.57)

```
# star_wars_matrix
box_office <- c(460.998, 314.4, 290.475, 247.900, 309.306,
165.8)
star_wars_matrix <- matrix(box_office, nrow = 3, byrow = TRUE,
dimnames = list(c("A New Hope",
"The Empire Strikes Back", "Return of the Jedi"),
c("US", "non-US")))
# taquilla mundial
worldwide_vector <- rowSums(star_wars_matrix)

#agregar la nueva variable a la matriz
all_wars_matrix <- cbind(star_wars_matrix,worldwide_vector)

all_wars_matrix
```

```

Console Terminal x
D:/Tut R/
> # star_wars_matrix
> box_office <- c(460.998, 314.4, 290.475, 247.900, 309.306, 165.8)
> star_wars_matrix <- matrix(box_office, nrow = 3, byrow = TRUE,
+                             dimnames = list(c("A New Hope", "The Empire Strikes Back", "
Return of the Jedi"),
+                                             c("US", "non-US")))
> # taquilla mundial
> worldwide_vector <- rowSums(star_wars_matrix)
>
> #agregar la nueva variable a la matriz
> all_wars_matrix <- cbind(star_wars_matrix,worldwide_vector)
>
> all_wars_matrix
              US non-US worldwide_vector
A New Hope    460.998  314.4             775.398
The Empire Strikes Back 290.475  247.9             538.375
Return of the Jedi    309.306  165.8             475.106
>

```

(Figura 1.57) Ahora se incluye la suma en la matriz.

1.26 Agregar una fila a una matriz

Puede agregar una fila o varias filas a una matriz con la función `rbind()`, que combina matrices y / o vectores juntos por fila. Por ejemplo:

```
big_matrix <- rbind (matriz1, matriz2, vector1 ...)
```

1.27 Suma de columna

Su espacio de trabajo de R ya contiene la matriz `all_wars_matrix` que construyó en el ejercicio anterior; escriba `all_wars_matrix` para verla. Ahora se calculan los ingresos totales de taquilla para toda la saga.

Calcule los ingresos totales para los EE. UU. Y la región no estadounidense y asigne `total_revenue_vector`. Puede usar la función `colSums()`.

Imprima `total_revenue_vector` para ver los resultados.

(Ver figura 1.59)

```
all_wars_matrix
```

```
# ganancias totales
```

```
total_revenue_vector <- colSums(all_wars_matrix)
```

```
total_revenue_vector
```

```
Console Terminal x
D:/Tut R/
> all_wars_matrix
              US non-US worldwide_vector
A New Hope   460.998 314.4       775.398
The Empire Strikes Back 290.475 247.9       538.375
Return of the Jedi   309.306 165.8       475.106
>
> # ganancias totales
> total_revenue_vector <- rowSums(all_wars_matrix)
>
> total_revenue_vector
              A New Hope The Empire Strikes Back       Return of the Jedi
              1550.796              1076.750              950.212
> |
```

(Figura 1.59)

1.28 Selección de elementos matriciales

Similar a los vectores, puede usar los corchetes [] para seleccionar uno o varios elementos de una matriz. Mientras que los vectores tienen una dimensión, las matrices tienen dos dimensiones. Por lo tanto, debe usar una coma para separar qué seleccionar de las filas de lo que desea seleccionar de las columnas. Por ejemplo:

my_matrix [1,2] selecciona el elemento en la primera fila y la segunda columna. my_matrix [1: 3,2: 4] da como resultado una matriz con los datos en las filas 1, 2, 3 y las columnas 2, 3, 4. Si desea seleccionar todos los elementos de una fila o columna, no hay número necesario antes o después de la coma, respectivamente:

my_matrix [, 1] selecciona todos los elementos de la primera columna. my_matrix [1,] selecciona todos los elementos de la primera fila.

Seleccione los ingresos no estadounidenses para todas las películas (la segunda columna completa de all_wars_matrix), almacene el resultado como non_us_all.

Use mean () en non_us_all para calcular el ingreso promedio no estadounidense para todas las películas. Simplemente imprima el resultado.

Esta vez, seleccione los ingresos no estadounidenses para las dos primeras películas en all_wars_matrix. Almacene el resultado como non_us_some.

Use mean () nuevamente para imprimir el promedio de los valores en non_us_some. (Ver figura 1.60)

```
all_wars_matrix
# Selecciona las non-US r
non_us_all <- all_wars_matrix[,2]
# promedio non-US revenue
mean(non_us_all)
```

```
# Seleccionar non-US revenue de las primeras 2 peliucias
non_us_some <- all_wars_matrix[1:2,2]
# promedio de las primeras 2 peliucias
mean(non_us_some)
```

```

Console Terminal x
D:/Tut R/
> all_wars_matrix
              US non-US worldwide_vector
A New Hope    460.998  314.4             775.398
The Empire Strikes Back 290.475  247.9             538.375
Return of the Jedi    309.306  165.8             475.106
>
> # Selecciona las non-US r
> non_us_all <- all_wars_matrix[,2]
>
> # promedio non-US revenue
> mean(non_us_all)
[1] 242.7
>
> # Seleccionar non-US revenue de las primeras 2 peliucias
>
> non_us_some <- all_wars_matrix[1:2,2]
>
> # promedio de las primeras 2 peliucias
> mean(non_us_some)
[1] 281.15
>

```

(Ver figura 1.60) Al igual que en álgebra se ejecutan operaciones en la matriz.

1.29 Un poco de aritmética con matrices.

Similar a lo que ha aprendido con los vectores, los operadores estándar como +, -, /, *, etc. funcionan de manera inteligente en matrices en R.

Por ejemplo, `2 * my_matrix` multiplica cada elemento de `my_matrix` por dos.

Al igual que `2 * my_matrix` multiplicó cada elemento de `my_matrix` por dos, `my_matrix1 * my_matrix2` crea una matriz donde cada elemento es el producto de los elementos correspondientes en `my_matrix1` y `my_matrix2`.

Después de ver el resultado del ejercicio anterior los precios de las entradas aumentaron con el tiempo. Se realiza el análisis en función de los precios que puede encontrar en `ticket_prices_matrix`

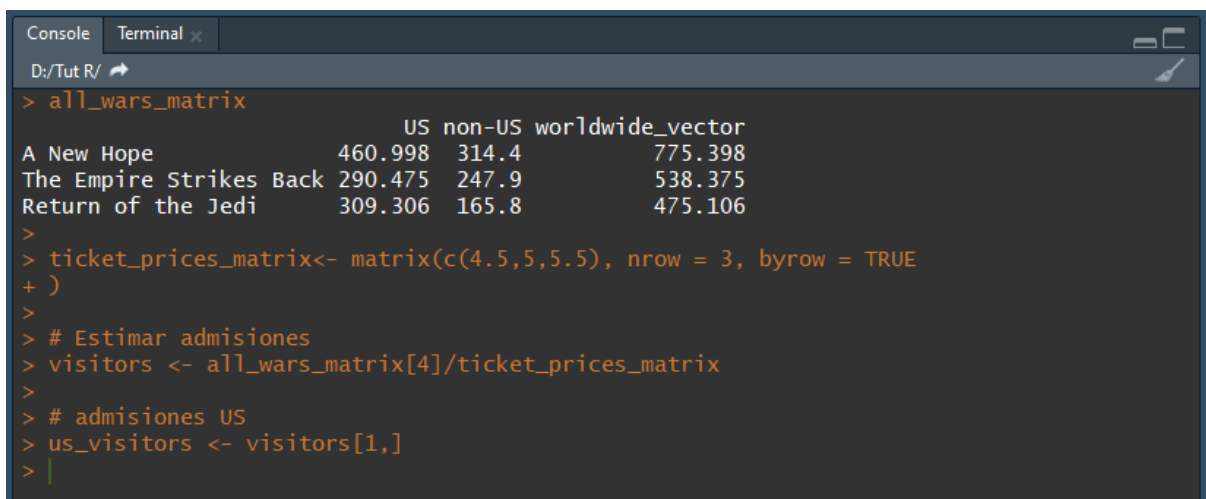
Aquellos que estén familiarizados con las matrices deben tener en cuenta que esta no es la multiplicación de matrices estándar para la que debe usar `%*%` en R.

Dividir `all_wars_matrix` por `ticket_prices_matrix` para obtener el número estimado de visitantes estadounidenses y no estadounidenses para las seis películas. Asignar el resultado a los visitantes.

En la matriz de visitantes, seleccione la primera columna completa, que representa el número de visitantes en los EE. UU. Almacene esta selección como `us_visitors`.

Calcule el número promedio de visitantes estadounidenses; imprime el resultado. (Ver figura 1.63)

```
all_wars_matrix
ticket_prices_matrix<- matrix(c(4.5,5,5.5), nrow = 3, byrow = TRUE)
# Estimar admisiones
visitors <- all_wars_matrix[4]/ticket_prices_matrix
# admisiones US
us_visitors <- visitors[1,]
```



```
Console Terminal x
D:/Tut R/ ↗
> all_wars_matrix
      US non-US worldwide_vector
A New Hope      460.998    314.4         775.398
The Empire Strikes Back 290.475    247.9         538.375
Return of the Jedi    309.306    165.8         475.106
>
> ticket_prices_matrix<- matrix(c(4.5,5,5.5), nrow = 3, byrow = TRUE
+ )
>
> # Estimar admisiones
> visitors <- all_wars_matrix[4]/ticket_prices_matrix
>
> # admisiones US
> us_visitors <- visitors[1,]
> |
```

(Figura 1.63) El valor de las admisiones estimadas está en el objeto `us_visitors`.

1.30 Comparación de matrices

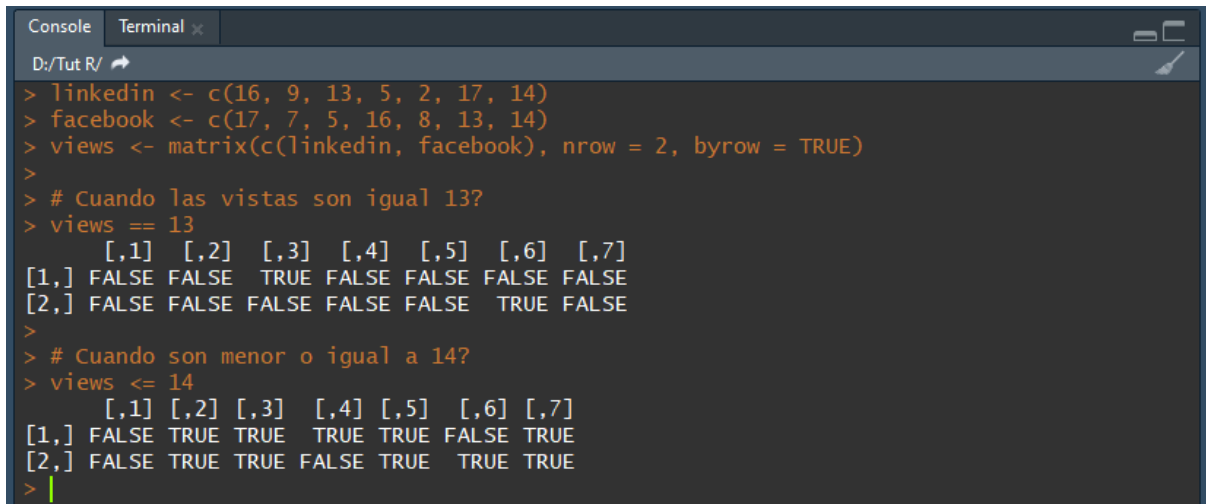
R posee una habilidad para trabajar con diferentes estructuras de datos; los operadores relacionales pueden ser usados en matrices.

Ahora, se ha generado una matriz con las mismas vistas a Facebook y LinkedIn, los vectores originales siguen disponibles. (Ver figura 1.64)

- ¿Cuándo ha habido 13 vistas? Usar la matriz `vistas`.
- ¿En qué días han sido igual o menores a 14?

```
linkedin <- c(16, 9, 13, 5, 2, 17, 14)
facebook <- c(17, 7, 5, 16, 8, 13, 14)
views <- matrix(c(linkedin, facebook), nrow = 2, byrow = TRUE)
# Cuando las vistas son igual 13?
```

```
views == 13
# Cuando son menor o igual a 14?
views <= 14
```



```
Console Terminal x
D:/Tut R/
> linkedin <- c(16, 9, 13, 5, 2, 17, 14)
> facebook <- c(17, 7, 5, 16, 8, 13, 14)
> views <- matrix(c(linkedin, facebook), nrow = 2, byrow = TRUE)
>
> # Cuando las vistas son igual 13?
> views == 13
  [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,] FALSE FALSE TRUE FALSE FALSE FALSE FALSE
[2,] FALSE FALSE FALSE FALSE FALSE TRUE FALSE
>
> # Cuando son menor o igual a 14?
> views <= 14
  [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,] FALSE TRUE TRUE TRUE TRUE FALSE TRUE
[2,] FALSE TRUE TRUE FALSE TRUE TRUE TRUE
> |
```

(Figura 1.64) El resultado es una matriz con cada elemento evaluado por separado.

2. Conceptos clave en el análisis de datos y alcance del tutorial

La ciencia de datos es una disciplina que permite convertir datos en bruto en percepciones acertadas y conocimiento a partir de dichos datos. A grosso modo, las herramientas usadas en un proyecto de ciencia de datos conviven de la siguiente manera:



9

2.1 Importación de datos

Es la lectura de datos brutos, que puede ser de las siguientes maneras:

- Leer un archivo
- Conectarse a una base de datos
- Conectarse a través de una API (Application Programming Interface)
- *Webscrapping* que consiste en un proceso sistemático en el cual se copia información que se muestra en una página web.

Existe información más detallada según el método de conexión respecto a las mejores prácticas.¹⁰

2.1.1 Leer, conectar, cargar fuentes de datos

Los datos que buscamos analizar pueden estar en diferentes formatos de archivos (.XLS, .TXT, .CSV, JSON) o bien en bases de datos, para los objetivos de este texto se propone un enfoque en archivos de texto csv.

Es muy sencillo cargar los datos en R por la simplicidad y disponibilidad de librerías.

```
# Leer CSV
Data <- read.csv(file="c:/TheDataIWantToReadIn.csv", header=TRUE,
sep=",")
#Leer TXT
Txt <- read.table("c:/TheDataIWantToReadIn.txt", sep="\t",
header=TRUE)
```

El resto de los comandos de la función Read tienen una estructura similar, al colocar el cursor en read.csv y presionar F1 se observa la viñeta de la documentación de R con las especificaciones de cada argumento.

⁹ Gráfico de: R for Data Science, Hardley Wickham

¹⁰ [Setting up ODBC Drivers](#)

2.2 Limpieza de datos

También conocido como curación de datos, suele ser una de las partes más laboriosas del proceso y fundamental ya que con la limpieza se busca tener los datos brutos almacenados de una forma consistente que haga sentido con la semántica de la fuente de datos, de esta manera tendremos que cada columna es una variable observada y cada columna es una observación.

2.2.1 Eliminar duplicados

De manera sencilla tenemos:

```
> set.seed(150)
> x <- round(rnorm(20, 10, 5))
> x
[1] 2 10 6 8 9 11 14 12 11 6 10 0 10 7 7 20 11 17 12 -1
> unique(x)
[1] 2 10 6 8 9 11 14 12 0 7 20 17 -1
```

2.2.2 Outliers y valores faltantes

Ubicar NA's es realizado de la siguiente manera:

```
> y <- c(4,5,6,NA)
> is.na(y)
[1] FALSE FALSE FALSE TRUE
```

y de esta manera se eliminan los vacíos:

```
y[is.na(y)] <- mean(y, na.rm=TRUE)
y
[1] 4 5 6 5
```

2.3 Transformación de datos

El primer paso común una vez que se tiene un dataset ordenado es transformarlo, y va desde hacer operaciones entre las variables como calcular la velocidad a partir de la aceleración y el tiempo, o bien filtrar alguna variable como podrían ser las personas de una ciudad o los datos en un periodo de tiempo

2.3.1 Tipos de Variables

Los tipos de variable funcionan como su nombre lo indica, cada uno le da diferentes atributos a un objeto, ya sea un número, cadena de texto, matriz o dataframe.

Usando **is.xyz** para probar si es del tipo **xyz**. Regresa TRUE o FALSE

Usando **as.xyz** para hacer una conversión explícita.

```
is.numeric(), is.character(), is.vector(), is.matrix(), is.data.frame()
```

```
as.numeric(), as.character(), as.vector(), as.matrix(), as.data.frame()
```

Sin embargo, convertir una estructura de datos es más crítico que transformar el formato:

	to one long vector	to matrix	to data frame
from vector	c(x,y)	cbind(x,y) rbind(x,y)	data.frame(x,y)
from matrix	as.vector(mymatrix)		as.data.frame(mymatrix)
from data frame		as.matrix(myframe)	

2.3.2 Transponer sets de datos

Cuando los datos son presentados de manera matricial, podemos hacer operaciones matriciales como la transposición:

id	Class	x1	x2
1	1	5	6
1	2	3	5
2	1	6	1
2	2	2	4



id	Class	variable	value
1	1	x1	5
1	2	x1	3
2	1	x1	6
2	2	x1	2
1	1	x2	6
1	2	x2	5
2	1	x2	1
2	2	x2	4

11

```
# Función melt
library(reshape)
mdata <- melt(mydata, id=c("id","time"))
```

¹¹ [An Introduction to reshape2 - Reshaping data easily with the reshape2 R package. - seananderson.ca](http://seananderson.ca)

2.3.3 Ordenar un set de datos

Con la función `order(nombre de la variable)` se pueden reacomodar las observaciones en un set de datos.

```
# ordenar por var1
newdata <- old[order(var1),]
# ordenar por var1 y var2 (descendiente)
newdata2 <- old[order(var1, -var2),]
```

2.3.4 Muestras de un conjunto de datos en R

Mostrar datos es común al momento de construir un modelo y probarlo. Para tomar una muestra de una fuente de datos se escribe lo siguiente:

```
mysample <- mydata[sample(1:nrow(mydata), 100,replace=FALSE),]
```

Este código contiene 100 observaciones aleatorias de 100 observaciones en la tabla `mydata`.

2.3.5 Combinar fuentes de datos

Otra operación bastante común:

```
# combinar 2 sets de datos por ID y por país:
total <- merge(data frameA,data frameB,by="ID")
# total <- merge(data frameA,data frameB,by=c("ID","Country"))
```

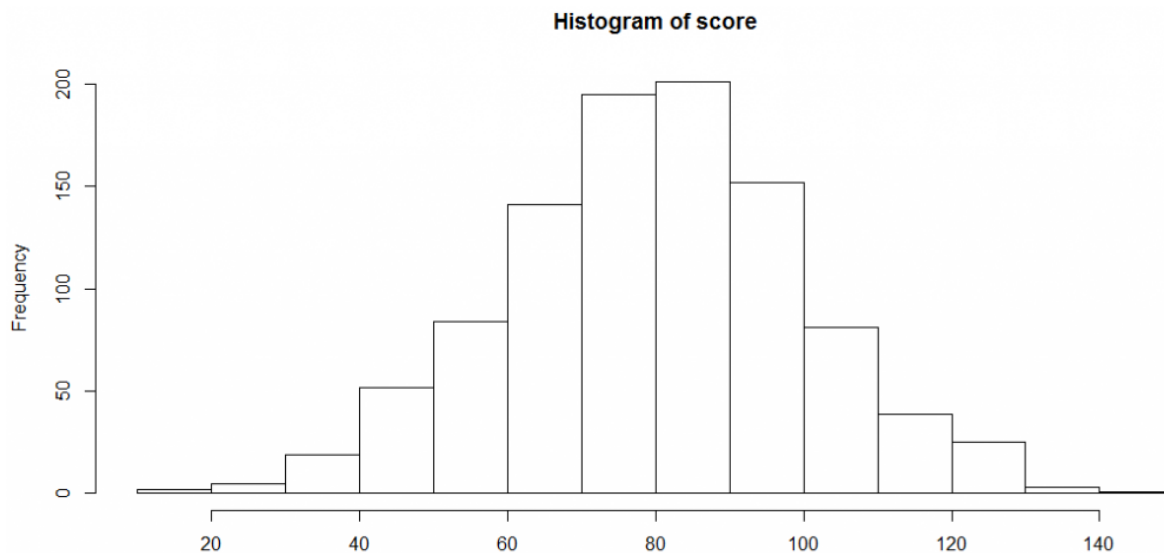
2.4 Visualización de datos

Visualizar es una actividad fundamental de los humanos, una buena visualización podrá mostrar patrones, outliers, y ayudará a saber si se está haciendo la pregunta correcta con los datos disponibles.

2.4.1 Crear histogramas

La sintaxis para la visualización de datos es muy sencilla y genera gráficos claros, a continuación se generará una distribución de puntos y se graficará su variación.

```
puntos <- rnorm(n=1000, m=80, sd=20)
hist(puntos)
```



Para conocer los atributos y características que R toma al general los datos escribimos lo siguiente:

```

histinfo<-hist(puntos)
histinfo
$breaks
[1] 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150
$count
[1] 2 5 19 52 84 141 195 201 152 81 39 25 3 1
$density
[1] 0.0002 0.0005 0.0019 0.0052 0.0084 0.0141 0.0195 0.0201 0.0152
[10] 0.0081 0.0039 0.0025 0.0003 0.0001
$mids
[1] 15 25 35 45 55 65 75 85 95 105 115 125 135 145
$xname
[1] "puntos"
$equidist
[1] TRUE
attr(,"class")
[1] "histogram"

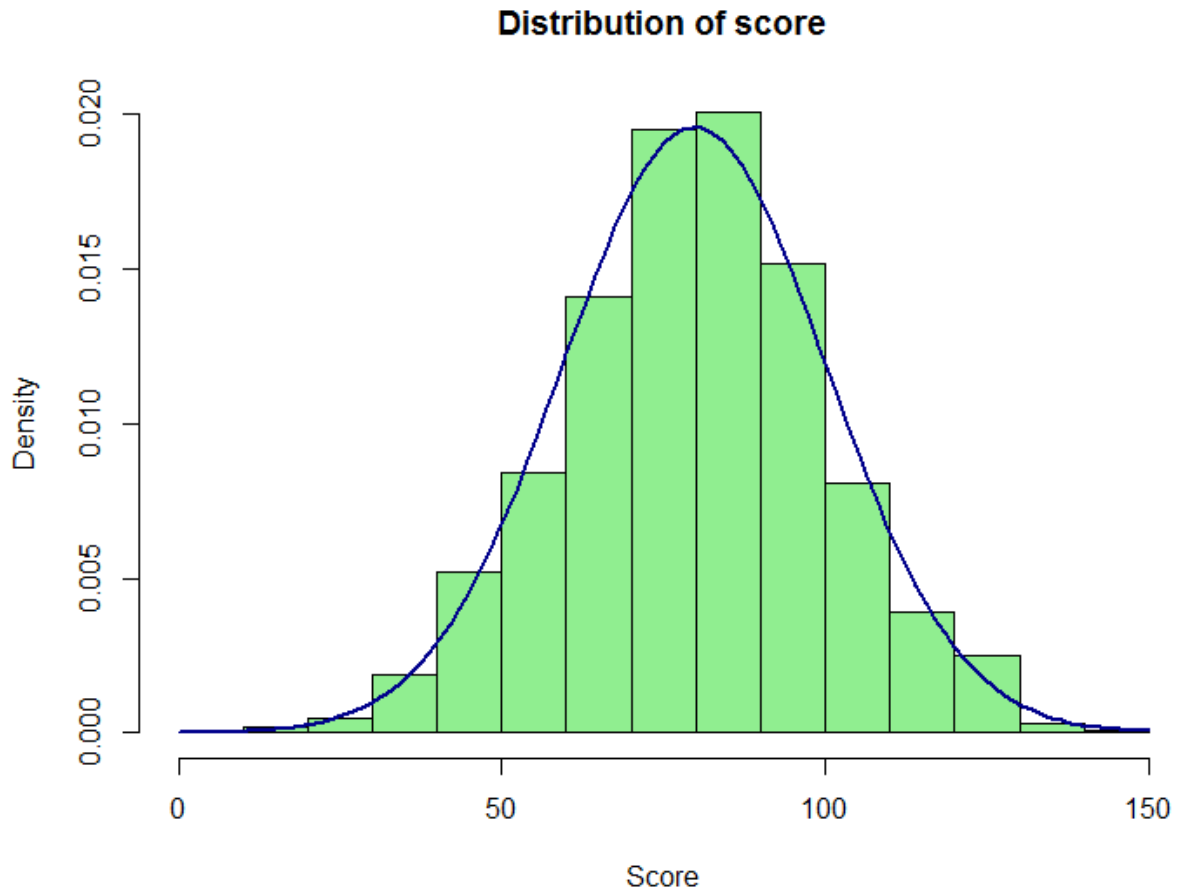
```

Como se puede observar, breaks indica en que puntos se hacen cortes, se pueden restringir los puntos de corte o variar la densidad. También se puede colorear la gráfica de barras y colocar encima una curva de distribución.

```

hist(puntos, freq=FALSE, xlab="Puntos", main="Distribucion de
puntos", col="lightgreen", xlim=c(0,150), ylim=c(0, 0.02))
curve(dnorm(x, mean=mean(score), sd=sd(score)), add=TRUE,
col="darkblue", lwd=2)

```



2.5 Modelado

Una forma de validar las preguntas de una forma precisa, se puede usar un modelo que explique los patrones de los datos con los que se cuenta y generar proyecciones y formular hipótesis.

2.5.1 Suma, cuenta y promedio de una variable en un set de datos

Las funciones Apply son útiles para estos cálculos:

```
> tapply(iris$Sepal.Length,iris$Species,sum)
setosa versicolor virginica
250.3 296.8 329.4
> tapply(iris$Sepal.Length,iris$Species,mean)
setosa versicolor virginica
5.006 5.936 6.588
```

2.6 Exploración y Transformación de Datos

La exploración, transformación y comunicación de datos son tareas fundamentales en proyectos de inteligencia de negocios. La exploración consiste en identificar patrones y tendencias en los datos, mientras que la transformación implica la limpieza y procesamiento de los datos para poder analizarlos adecuadamente. Y la comunicación consiste en transmitir las ideas generadas por los dos pasos anteriores a los miembros clave de la organización o equipo de trabajo.

En esta sección, se explicarán las técnicas y herramientas utilizadas para realizar estas tareas de manera efectiva, para abordar este tema he preparado los siguientes casos de estudio que ilustran este proceso de una manera prácticas con problemas reales.

Como lo he venido planteando desde la introducción, R es un lenguaje de programación muy utilizado en proyectos de inteligencia de negocios debido a su capacidad para importar y manipular datos de diferentes fuentes. Además, R ofrece una variedad de paquetes y librerías que permiten realizar análisis estadísticos y visualización de datos, así como también para la limpieza y preprocesamiento de datos que consiste en herramientas para tratar valores faltantes, valores atípicos y datos duplicados.

3. Caso de Estudio 1: Análisis del flujo de autos en los estacionamientos de la Facultad.

3.1 Objetivo

Analizar los datos para poder tener una matriz de datos limpia que sirva como entrada para un modelo de simulación.

Algunos datos que observar:

Conteo de tarjetas únicas por día

Histograma, identificar periodos intersemestrales y demanda baja

3.2 Importación de datos

Para leer archivos en Excel la librería *readxl()* será usada, al igual que en ejemplos anteriores usaremos *dplyr* de *Tidyverse*.

En la ventana del editor, se escriben las instrucciones para cargar las librerías necesarias para este análisis (Ver figura Caso 1.1):

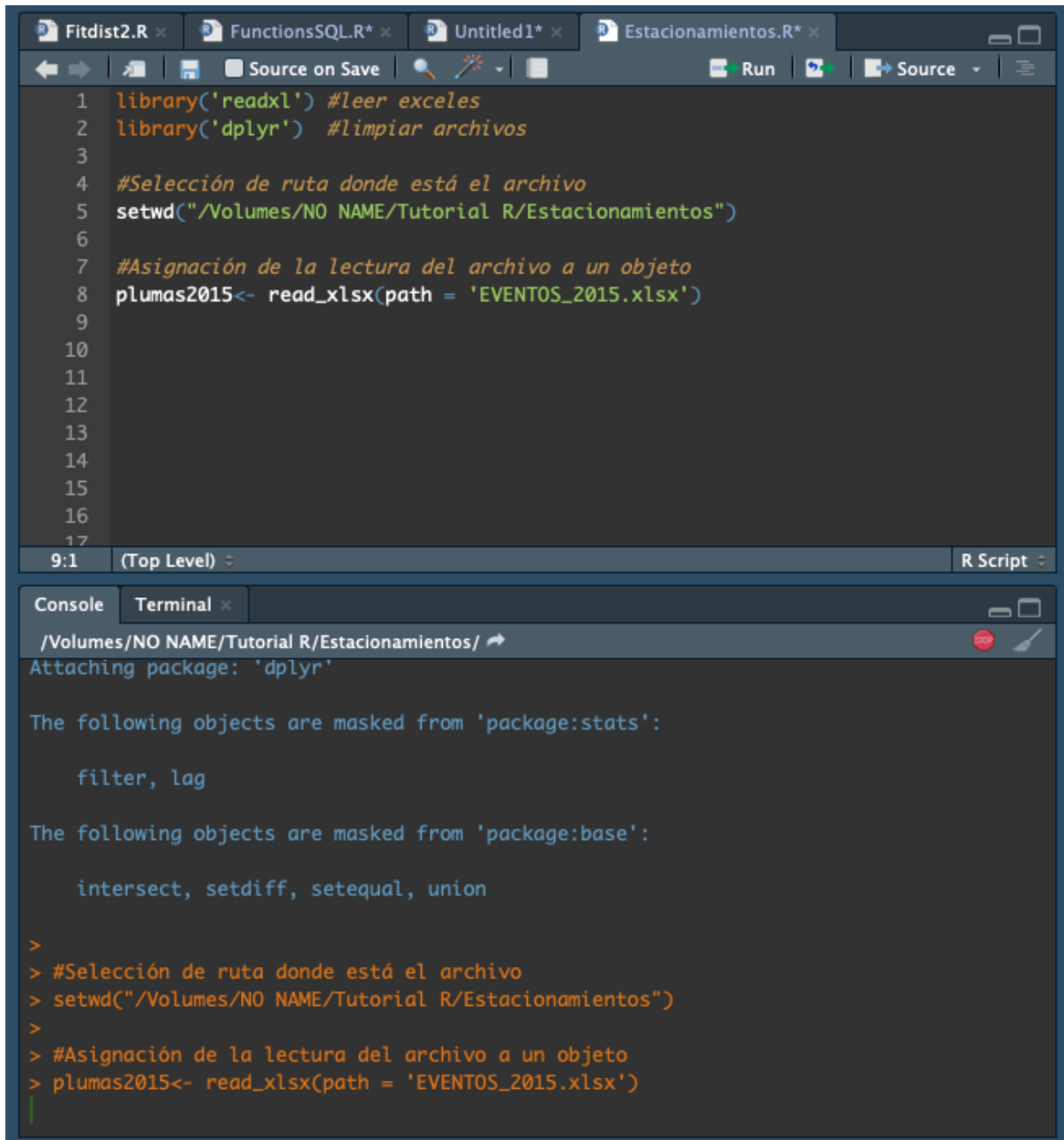
Se identificará el “Working Directory” para asegurar la lectura del archivo correcto.

Se indicará la ruta correcta con el comando *setwd()*

Se mostrará los archivos en el directorio de trabajo y se asignan al objeto *plumas2015* la información contenida en el archivo “EVENTOS_2015.xlsx”

```
library('readxl') #leer excel  
library('dplyr') #limpiar archivos
```

```
#Asígnanos a un objeto el archivo en Excel:  
plumas2015<- read_xlsx(path = 'EVENTOS_2015.xlsx')
```



The image shows a screenshot of the R Studio interface. The top pane is the script editor, displaying the following R code:

```
1 library('readxl') #leer exceles
2 library('dplyr') #limpiar archivos
3
4 #Selección de ruta donde está el archivo
5 setwd("/Volumes/NO NAME/Tutorial R/Estacionamientos")
6
7 #Asignación de la lectura del archivo a un objeto
8 plumas2015<- read_xlsx(path = 'EVENTOS_2015.xlsx')
9
10
11
12
13
14
15
16
17
```

The bottom pane is the console window, showing the execution of the code and the following output:

```
/Volumes/NO NAME/Tutorial R/Estacionamientos/ ↗
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

  filter, lag

The following objects are masked from 'package:base':

  intersect, setdiff, setequal, union

>
> #Selección de ruta donde está el archivo
> setwd("/Volumes/NO NAME/Tutorial R/Estacionamientos")
>
> #Asignación de la lectura del archivo a un objeto
> plumas2015<- read_xlsx(path = 'EVENTOS_2015.xlsx')
```

(Figura Caso 1.1) En la parte superior está en script actual y en la inferior las interacciones de la consola.

3.3 Limpieza de datos

Se inicia el análisis exploratorio con las funciones comunes para observar datos (Ver figura Caso 1.2):

```
head(plumas2015,10)
```

```

Console Terminal x
/Volumes/NO NAME/Tutorial R/Estacionamientos/ ↗
> head(plumas2015,10)
# A tibble: 10 x 8
  `UNIVERSIDAD NACIONAL ~ X__1      X__2  X__3    X__4  X__5  X__6  X__7
  <chr> <chr> <chr> <chr> <chr> <chr> <chr>
1 "FACULTAD DE INGENIER~ NA      NA      NA      NA      NA      NA      NA
2 "DEPARTAMENTO DE SISTE~ NA      NA      NA      NA      NA      NA      NA
3 "COORDINACI\u00d3N DE ~ NA      NA      NA      NA      NA      NA      NA
4 HISTORIO EVENTOS ESTAC~ NA      NA      NA      NA      NA      NA      NA
5 NA      NA      NA      NA      NA      NA      NA
6 NA      NA      NA      NA      NA      NA      NA
7 TARJETA      NOMBRE  AREA   GRUPO  PANEL  FECHA  HORA   TIPO ~
8 58031      VIGILANCI~ SA CSG MASTER  SUR 1~ 42005~ 42005~ ACCES~
9 11009      ENTRADA P~ SA CSG MASTER  NORTE~ 42005~ 42005~ ACCES~
10 20971      ALVAREZ C~ DIE    ACADEMI~ SUR 3~ 42005~ 42005~ ACCES~
>

```

(Figura Caso 1.2) Se aprecian valores vacíos como NA.

Con la notación matricial, se eliminan los renglones no deseados en el set de datos (Ver figura Caso 1.3):

```
plumas2015 <- plumas2015[-c(1:6),-c(2)]
```

```


Fitdist2.R x FunctionsSQL.R* x Untitled1* x Estacionamientos.R* x
Source on Save Run Source
1 library(readxl) #leer excel
2 library('dplyr') #limpiar archivos
3
4 #Selección de ruta donde está el archivo
5 setwd("/Volumes/NO NAME/Tutorial R/Estacionamientos")
6
7 #Asignación de la lectura del archivo a un objeto
8 plumas2015<- read_xlsx(path = 'EVENTOS_2015.xlsx')
9
10 #Notación matricial para eliminar renglones no deseados
11 plumas2015 <- plumas2015[-c(1:6),-c(2)]
12
13
14
15
16
17
12:1 (Top Level) R Script
Console Terminal x
/Volumes/NO NAME/Tutorial R/Estacionamientos/ ↗
> plumas2015 <- plumas2015[-c(1:6),-c(2)]
>

```

(Figura Caso 1.4) En el script se selecciona la línea de código y con ctrl + enter se ejecuta en la consola.

Al volver a correr el siguiente comando, se observa como los renglones no deseados ya no forman parte de nuestro objeto plumas2015, asimismo, se puede apreciar que en la línea 1 están los nombres de los encabezados del set de datos. (Ver figura Caso 1.4):

```
head(plumas2015)
```



```
> head(plumas2015)
# A tibble: 6 x 7
  `UNIVERSIDAD NACIONAL AUT~ X__2 X__3 X__4 X__5 X__6 X__7
  <chr> <chr> <chr> <chr> <chr> <chr> <chr>
1 TARJETA AREA GRUPO PANEL FECHA HORA TIPO EV~
2 58031 SA CSG MASTER SUR 1 E~ 42005.2~ 42005.2~ ACCESO ~
3 11009 SA CSG MASTER NORTE 1~ 42005.4~ 42005.4~ ACCESO ~
4 20971 DIE ACADEMIC~ SUR 3 E~ 42005.5~ 42005.5~ ACCESO ~
5 58031 SA CSG MASTER SUR 1 S~ 42005.5~ 42005.5~ ACCESO ~
6 24087 DICYG ACADEMIC~ SUR 3 E~ 42005.6~ 42005.6~ ACCESO ~
```

(Figura Caso 1.4) Por debajo de los encabezados se observa el tipo de variable <chr> carácter.

Con el siguiente comando se define el primer renglón como el nombre de las variables o encabezado del set de datos.

```
colnames(plumas2015)<-plumas2015[1, ]
```

Ahora que se observa que el primer renglón del set de datos corresponde al encabezado, puede ser eliminado el renglón. En cualquier momento podemos dar un vistazo al encabezado del set de datos con la función head() (Ver figura Caso 1.5)

```
plumas2015 <- plumas2015[-c(1), ]
```

```

7 #Asignación de la lectura del archivo a un objeto
8 plumas2015<- read_xlsx(path = 'EVENTOS_2015.xlsx')
9
10 #Notación matricial para eliminar renglones no deseados
11 plumas2015 <- plumas2015[-c(1:6),-c(2)]
12
13 #Asignación de nombres a las variables o columnas
14 colnames(plumas2015)<-plumas2015[1,]
15
16 #Se elimina el renglon con el encabezado
17 plumas2015 <- plumas2015[-c(1),]
18
19
20
21
22

```

```

> #Se elimina el renglon con el encabezado
> plumas2015 <- plumas2015[-c(1),]
> head(plumas2015)
# A tibble: 6 x 7
  TARJETA AREA   GRUPO      PANEL    FECHA      HORA    `TIPO EVENTO`
  <chr>   <chr> <chr>    <chr>    <chr>    <chr>    <chr>
1 58031   SA CSG MASTER SUR 1 ENTR~ 42005.2786~ 42005.2786~ ACCESO CONCED~
2 11009   SA CSG MASTER NORTE 1 EN~ 42005.4871~ 42005.4871~ ACCESO CONCED~
3 20971   DIE   ACADEMICOS_V~ SUR 3 ENTR~ 42005.5198~ 42005.5198~ ACCESO CONCED~
4 58031   SA CSG MASTER SUR 1 SALI~ 42005.5487~ 42005.5487~ ACCESO CONCED~
5 24087   DICYG ACADEMICOS SUR 3 ENTR~ 42005.6203~ 42005.6203~ ACCESO CONCED~
6 4511    IING  ACADEMICOS SUR 3 ENTR~ 42005.6994~ 42005.6994~ ACCESO CONCED~
>

```

(Figura Caso 1.5) la función head() muestra el estado actual del objeto plumas2015 después de las modificaciones ejecutadas sobre este.

Continuando con la fase de limpieza de datos, observe como la columna tarjeta tiene datos de tipo numérico, pero son tratados como caracteres. Además, la columna de fecha y hora está en formato numérico. El código mostrado a continuación, se asigna el formato correcto a cada variable:

```
plumas2015$TARJETA <- as.numeric(plumas2015$TARJETA)
```

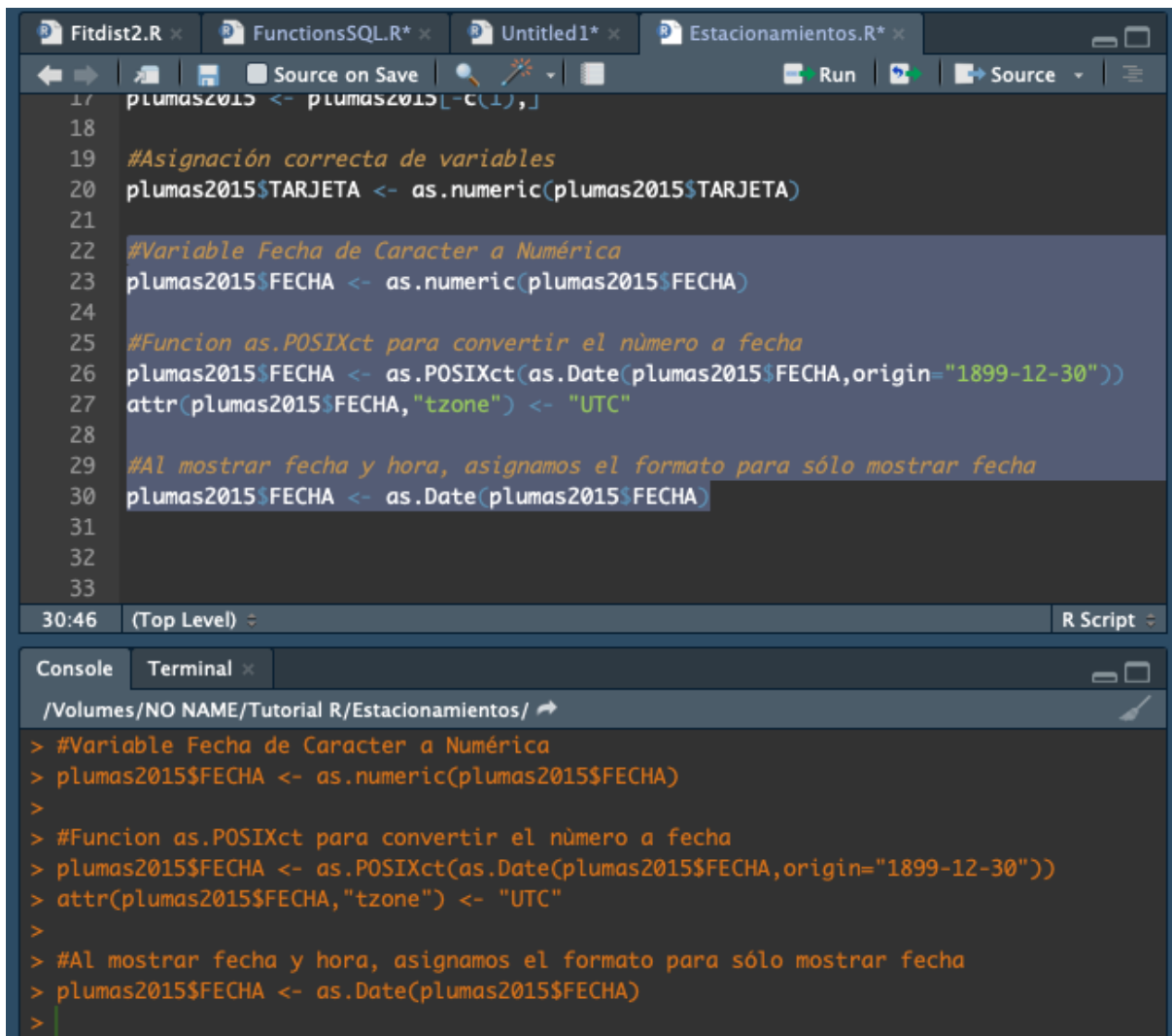
3.4 Transformación de datos

Para asignar el formato correcto a las columnas de fecha y hora existe la función `as.POSIXct` el cual es un estándar que puede manejar unidades de tiempo, con una definición de años a segundos. Para poder usar esta función, primero se convertirán las variables de tipo carácter a tipo numérico, posteriormente se asignará la zona horaria para evitar errores de conversión. (Ver figura Caso 1.7)

```
plumas2015$FECHA <- as.numeric(plumas2015$FECHA)
```

```
plumas2015$FECHA <-  
as.POSIXct(as.Date(plumas2015$FECHA,origin="1899-12-30"))  
attr(plumas2015$FECHA,"tzone") <- "UTC"
```

```
plumas2015$FECHA <- as.Date(plumas2015$FECHA)
```



The screenshot shows the R Studio interface with a script editor and a console. The script editor contains the following R code:

```
17 plumas2015 <- plumas2015[-c(1),]  
18  
19 #Asignación correcta de variables  
20 plumas2015$TARJETA <- as.numeric(plumas2015$TARJETA)  
21  
22 #Variable Fecha de Caracter a Numérica  
23 plumas2015$FECHA <- as.numeric(plumas2015$FECHA)  
24  
25 #Funcion as.POSIXct para convertir el número a fecha  
26 plumas2015$FECHA <- as.POSIXct(as.Date(plumas2015$FECHA,origin="1899-12-30"))  
27 attr(plumas2015$FECHA,"tzone") <- "UTC"  
28  
29 #Al mostrar fecha y hora, asignamos el formato para sólo mostrar fecha  
30 plumas2015$FECHA <- as.Date(plumas2015$FECHA)  
31  
32  
33
```

The console shows the execution of the code:

```
> #Variable Fecha de Caracter a Numérica  
> plumas2015$FECHA <- as.numeric(plumas2015$FECHA)  
>  
> #Funcion as.POSIXct para convertir el número a fecha  
> plumas2015$FECHA <- as.POSIXct(as.Date(plumas2015$FECHA,origin="1899-12-30"))  
> attr(plumas2015$FECHA,"tzone") <- "UTC"  
>  
> #Al mostrar fecha y hora, asignamos el formato para sólo mostrar fecha  
> plumas2015$FECHA <- as.Date(plumas2015$FECHA)  
>
```

(Figura Caso 1.7) Nuevamente, el código seleccionado se ejecuta si presionamos `ctrl + enter`.

Un problema común es que mientras en excel las fechas tienen un formato en la hoja de cálculo, al momento de ser escritas en un archivo son escritas como número. Este tipo de errores pueden o no llegar a ser frecuentes y es una buena forma de mostrar el poder de la colaboración en la programación, plataformas sociales como *Stackoverflow* sirven para preguntar este tipo de problemas particulares. (Ver figura Caso 1.8)

Converting excel DateTime serial number to R DateTime

Asked 6 years, 4 months ago Active 8 months ago Viewed 35k times

When excel tables are imported as xy points in ArcGIS I continue to lose my correct DateTime stamp for each point. Thus, I have formatted the DateTime serial number, created the .shp, and read the .shp into R using readOGR().

34

Once in R I can convert to the correct date using `as.Date()` and the `origin = "1899-12-30"` argument, but the time is left out. While I have seen examples with a sole Date, I have not seen worked examples with DateTime. I have been using `as.Date()` as well as `as.POSIXct()` but this seemingly simple task as been a bit frustrating, thus the post...

I have created a sample data set with 10 rows of the correct DateTime format as well as the excel serial number.

*Thanks Richard and thelatemail for their keen eye on an earlier hindrance. I have corrected the data and re-posted here.

Here is my sample data

```
helpData <- structure(list(ID = 1:10, DateTime = structure(c(9L, 1L, 2L, 3L, 4L, 5L, 6L, 7L, 8L, 8L), .Label = c("3/11/2011 7:55", "3/13/2011 7:55", "3/14/2011 0:00", "3/14/2011 10:04", "3/14/2011 7:55", "3/15/2011 19:55", "3/17/2011 7:55", "3/18/2011 4:04", "3/4/2011 6:00"), class = "factor"), ExcelNum = c(40606.25, 40613.32986, 40615.32986, 40616, 40616.41944, 40616.32986, 40617.82986, 40619.32986, 40620.16944, 40620.16944)), .Names = c("ID", "DateTime", "ExcelNum"), class = "data.frame", row.names = c(NA, -10L))
head(helpData)
```

(Figura Caso 1.8) Sitio web StackOverflow¹².

Con la variable horas se realiza algo similar, con la diferencia de que el formato usado será el de HH:MM:SS

Como siempre, es útil usar el comando head para ver el resultado de las instrucciones arriba mencionadas (Ver figura Caso 1.9)

```
> head(plumas2015)
# A tibble: 6 x 7
  TARJETA AREA GRUPO PANEL FECHA HORA `TIPO EVENTO`
  <chr> <chr> <chr> <chr> <date> <chr> <chr>
1 58031 SA CSG MASTER SUR 1 ENTRA~ 2015-01-01 06:41:~ ACCESO CONCEDI~
2 11009 SA CSG MASTER NORTE 1 ENT~ 2015-01-01 11:41:~ ACCESO CONCEDI~
3 20971 DIE ACADEMICOS_VESP~ SUR 3 ENTRA~ 2015-01-01 12:28:~ ACCESO CONCEDI~
4 58031 SA CSG MASTER SUR 1 SALIDA 2015-01-01 13:10:~ ACCESO CONCEDI~
5 24087 DICYG ACADEMICOS SUR 3 ENTRA~ 2015-01-01 14:53:~ ACCESO CONCEDI~
6 4511 IING ACADEMICOS SUR 3 ENTRA~ 2015-01-01 16:47:~ ACCESO CONCEDI~
```

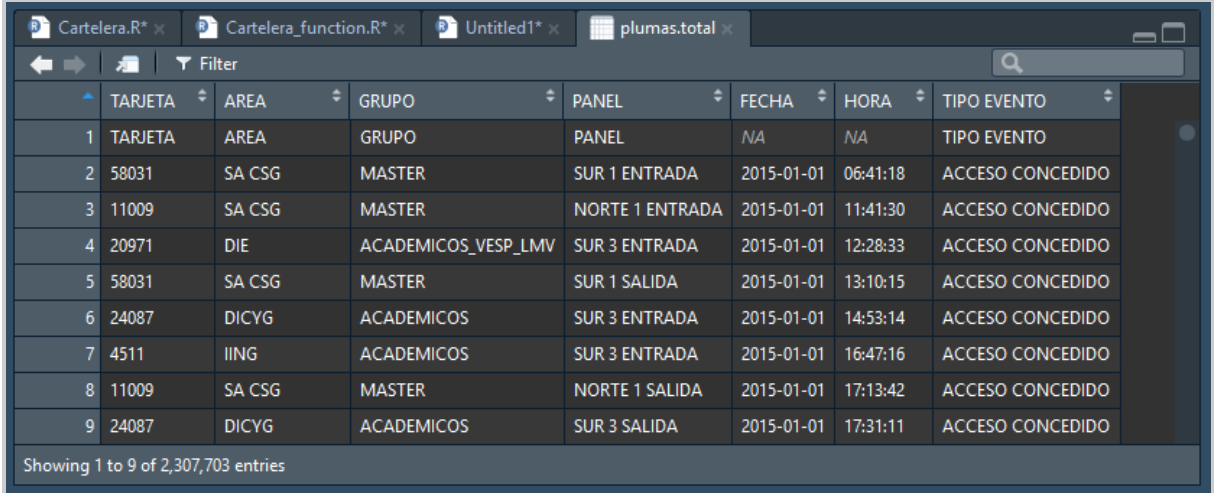
(Figura Caso 1.9) Observe como la columna fecha es un dato <date>.

¹² [Converting excel DateTime serial number to R DateTime - Stack Overflow](https://stackoverflow.com/questions/1011412/converting-excel-datetime-serial-number-to-r-datetime)

3.5 Visualización de datos

Para tener una vista completa, existe el comando `View()` que abre una nueva pestaña en la ventana Source Editor (Ver figura Caso 1.10)

```
View(plumas.total)
```



	TARJETA	AREA	GRUPO	PANEL	FECHA	HORA	TIPO EVENTO
1	TARJETA	AREA	GRUPO	PANEL	NA	NA	TIPO EVENTO
2	58031	SA CSG	MASTER	SUR 1 ENTRADA	2015-01-01	06:41:18	ACCESO CONCEDIDO
3	11009	SA CSG	MASTER	NORTE 1 ENTRADA	2015-01-01	11:41:30	ACCESO CONCEDIDO
4	20971	DIE	ACADEMICOS_VESP_LMV	SUR 3 ENTRADA	2015-01-01	12:28:33	ACCESO CONCEDIDO
5	58031	SA CSG	MASTER	SUR 1 SALIDA	2015-01-01	13:10:15	ACCESO CONCEDIDO
6	24087	DICYG	ACADEMICOS	SUR 3 ENTRADA	2015-01-01	14:53:14	ACCESO CONCEDIDO
7	4511	IING	ACADEMICOS	SUR 3 ENTRADA	2015-01-01	16:47:16	ACCESO CONCEDIDO
8	11009	SA CSG	MASTER	NORTE 1 SALIDA	2015-01-01	17:13:42	ACCESO CONCEDIDO
9	24087	DICYG	ACADEMICOS	SUR 3 SALIDA	2015-01-01	17:31:11	ACCESO CONCEDIDO

(Figura Caso 1.10)

Los siguientes bloques de código utilizan el método pipe notation¹³, en el cual se encadenan instrucciones con los caracteres “%>%” de esta manera dado un set de datos o dataframe se ejecutan diferentes acciones. Por ejemplo, para conocer los tipos de acceso, se seleccionan las columnas Tarjeta, Panel y Tipo Evento, después se agrupan por Panel y después se cuentan los Tipo Evento, esto en código en R luce de la siguiente manera:

```
#tipos de acceso
a<- (plumas.total %>%
  select(TARJETA, PANEL, `TIPO EVENTO`) %>%
  group_by(PANEL) %>%
  count(`TIPO EVENTO`)
)
```

¹³ [4 Pipes | The tidyverse style guide](#)

```

Console Terminal x
F:/
> #tipos de acceso
> a<-(plumas.total %>%
+   select(TARJETA,PANEL, TIPO.EVENTO) %>%
+   group_by(PANEL) %>%
+   count("TIPO.EVENTO")
+ )
> a

```

	TIPO.EVENTO	freq
1		33253 113
2	ACCESO CONCEDIDO	2277809
3	ACCESO DENEGADO ANTIPASSBACK	15460
4	ACCESO DENEGADO FECHA INVALIDA	11472
5	ACCESO DENEGADO TARJETA INACTIVA	135
6	ACCESO DENEGADO USUARIO DESCONOCIDO EN EL PANEL	2712
7	TIPO EVENTO	2

(Figura Caso 1.11)

```

#grupos de acceso
g<-(plumas.total %>%
  select(TARJETA,PANEL,GRUPO) %>%
  group_by(PANEL) %>%
  count(GRUPO)
)

```

```

Console Terminal x
F:/
> g<-(plumas.total %>%
+   select(TARJETA,PANEL,GRUPO) %>%
+   group_by(PANEL) %>%
+   count("GRUPO")
+ )
> g

```

	GRUPO	freq
1	ACADEMICOS	1369577
2	ACADEMICOS_MAT_LMV	58748
3	ACADEMICOS_MAT_MJ	31256
4	ACADEMICOS_MATUTINO	42388
5	ACADEMICOS_VESP_LMV	15801
6	ACADEMICOS_VESP_MJ	22319
7	ACADEMICOS_VESPERTINO	26168
8	ACADEMICOS_VIERNES	6
9	ADMINISTRATIVOS_NORTE	280316
10	ADMINISTRATIVOS_NORTE_SUR	54463
11	ADMINISTRATIVOS_SUR	201331
12	ADMINISTRATIVOS_SUR_SUR3	2109
13	ESPECIAL	213
14	ESPECIAL_SUR_1	9483
15	ESPECIAL_SUR_3	339
16	ESPECIAL_VIE_SAB	11842
17	ESTACIONAMIENTOS_SUR	137
18	ESTACIONAMIENTOS_NORTE	11875
19	GRUPO	2
20	Master	114077
21	MASTER	54769
22	SABADO	6
23	SUR4	478

(Figura Caso 1.12)

```

#fechas de acceso
f_entrada<-(plumas.total %>%
  select(TARJETA,PANEL,FECHA,HORA,TIPO.EVENTO) %>%
  filter(TIPO.EVENTO == "ACCESO CONCEDIDO") %>%
  filter(PANEL == "SUR 1 ENTRADA") %>%
  group_by(PANEL,TARJETA, FECHA, HORA) %>%
  arrange(FECHA)
)

```

The screenshot shows a terminal window with the following content:

```

> f_entrada<-(plumas.total %>%
+   select(TARJETA,PANEL,FECHA,HORA,TIPO.EVENTO) %>%
+   filter(TIPO.EVENTO == "ACCESO CONCEDIDO") %>%
+   filter(PANEL == "SUR 1 ENTRADA") %>%
+   group_by(PANEL,TARJETA, FECHA, HORA) %>%
+   arrange(FECHA)
+ )
> head(f_entrada)
# A tibble: 6 x 5
# Groups:   PANEL, TARJETA, FECHA, HORA [6]
  TARJETA PANEL      FECHA      HORA      TIPO.EVENTO
  <chr>   <chr>      <chr>      <chr>      <chr>
1 58031   SUR 1 ENTRADA 2015-01-01 06:41:18 ACCESO CONCEDIDO
2 58031   SUR 1 ENTRADA 2015-01-02 06:31:45 ACCESO CONCEDIDO
3 58031   SUR 1 ENTRADA 2015-01-02 07:00:49 ACCESO CONCEDIDO
4 58031   SUR 1 ENTRADA 2015-01-02 08:08:30 ACCESO CONCEDIDO
5 58031   SUR 1 ENTRADA 2015-01-02 11:36:24 ACCESO CONCEDIDO
6 58031   SUR 1 ENTRADA 2015-01-02 13:46:59 ACCESO CONCEDIDO
> |

```

(Figura Caso 1.13)

```

f_salida<-(plumas.total %>%
  select(TARJETA,PANEL,FECHA,HORA,TIPO.EVENTO) %>%
  filter(TIPO.EVENTO == "ACCESO CONCEDIDO") %>%
  filter(PANEL == "SUR 1 SALIDA") %>%
  filter(TARJETA == 11001) %>%
  group_by(PANEL,TARJETA, FECHA, HORA) %>%
  arrange(FECHA) %>%
  count(TIPO.EVENTO)
)

```

```
Console Terminal x
F:/ ➔
> f_salida<-(plumas.total %>%
+   select(TARJETA,PANEL,FECHA,HORA,TIPO.EVENTO) %>%
+   filter(TIPO.EVENTO == "ACCESO CONCEDIDO") %>%
+   filter(PANEL == "SUR 1 SALIDA") %>%
+   group_by(PANEL,TARJETA, FECHA, HORA) %>%
+   arrange(FECHA)
+ )
> head(f_salida)
# A tibble: 6 x 5
# Groups:   PANEL, TARJETA, FECHA, HORA [6]
  TARJETA PANEL      FECHA      HORA      TIPO.EVENTO
  <chr>   <chr>      <chr>      <chr>      <chr>
1 58031   SUR 1 SALIDA 2015-01-01 13:10:15 ACCESO CONCEDIDO
2 216     SUR 1 SALIDA 2015-01-03 06:34:42 ACCESO CONCEDIDO
3 216     SUR 1 SALIDA 2015-01-03 21:21:07 ACCESO CONCEDIDO
4 58031   SUR 1 SALIDA 2015-01-04 06:43:24 ACCESO CONCEDIDO
5 216     SUR 1 SALIDA 2015-01-04 20:55:04 ACCESO CONCEDIDO
6 216     SUR 1 SALIDA 2015-01-05 05:48:57 ACCESO CONCEDIDO
> |
```

(Figura Caso 1.14)

```
names(f_salida)[4] <- "HORA_SALIDA"
names(f_entrada)[4] <- "HORA_ENTRADA"
```

```
estacionamientos <-
left_join(f_entrada,f_salida,by=c("TARJETA","FECHA"))
```

```
write.csv(x = plumas.total,file =
"EstacionamientosTotal.csv",quote = FALSE,col.names = TRUE)
```


4. Caso de Estudio 2: Datos de Inmuebles

4.1 Objetivo

El objetivo en este caso de estudio es realizar un análisis exploratorio para conocer qué tan limpia luce la información, convertir sólo los valores en dólares a pesos y realizar un análisis que permita determinar qué operación de compra o renta conviene hacer en cada propiedad en una colonia en particular. Este análisis está basado en una publicación de FORBES¹⁴:

...Como indicador adicional, existe un principio económico llamado Price to rent ratio que puede servir como punto de referencia para determinar si, en cierta zona, es mejor alquilar o comprar. Este se calcula con la relación entre los precios de una vivienda y la renta anual de la misma.

*Calcular este ratio es bastante simple, básicamente hay que dividir el precio de venta entre la renta anual del **inmueble en cuestión**. La teoría del Price to rent ratio nos dice que, si el resultado es mayor a 20, es mucho más conveniente rentar que comprar, y si es menor a 15, debes aprovechar la oportunidad de comprar. Si la cifra se encuentra entre 16 y 19, habrá que evaluar otros puntos como el tiempo que planeas vivir en ella, los intereses de tu hipoteca, etc....*

4.2 Importación de Datos

En el conjunto de datos “inmuebles.csv”, hay caracteres especiales con errores de lectura, precios mezclados en dólares y pesos. Para empezar a analizar el set de datos será necesario empezar a escribir algunos de los comandos básicos de exploración en R en la ventana del editor (Ver figura Caso 2.1):

Se identificará el “Working Directory” para asegurar la lectura del archivo correcto.
Se indicará la ruta correcta con el comando `setwd()`
Se mostrarán los archivos en el directorio de trabajo y se asignan al objeto `Inmuebles` la información contenida en el archivo “inmuebles.csv”

En la ventana de la consola se observa lo que R regresa al correr los comandos de la ventana del editor. (Ver figura Caso 2.1)

¹⁴ [¿Rentar es tirar dinero a la basura?](#)

```
1 ###Inmuebles
2 #Conoceremos en que carpeta esta R
3 getwd()
4 #Definir escritorio de trabajo en carpeta con archivos
5 setwd("/Volumes/NO NAME/Tutorial R/Inmubeles")
6 #Listar archivos de la carpeta
7 list.files(getwd())
8 #Asignar a un objeto el archivo que leeremos
9 Inmuebles <- read.csv(file = "Inmuebles.csv", sep = ",",
10                       header = TRUE, encoding = 'UTF-8')
11
12
13
14
15
16
17
```

16:1 (Top Level) R Script

Console Terminal

```
/Volumes/NO NAME/Tutorial R/Inmubeles/
> ###Inmuebles
> #Conoceremos en que carpeta esta R
> getwd()
[1] "/Volumes/NO NAME"
> #Definir escritorio de trabajo en carpeta con archivos
> setwd("/Volumes/NO NAME/Tutorial R/Inmubeles")
> #Listar archivos de la carpeta
> list.files(getwd())
[1] "Inmuebles.R" "Inmuebles.csv"
> #Asignar a un objeto el archivo que leeremos
> Inmuebles <- read.csv(file = "Inmuebles.csv", sep = ",",
+                       header = TRUE, encoding = 'UTF-8')
>
```

(Figura Caso 2.1) Diálogos de la consola con los archivos del directorio de trabajo.

4.3 Análisis Exploratorio de Datos

Se inicia el análisis exploratorio mostrando el encabezado del archivo “inmuebles.csv” para ello, se usa el siguiente comando:(Ver figura Caso 2.2)

```
head(Inmuebles)
```

```

Console Terminal x
/Volumes/NO NAME/Tutorial R/Inmuebles/ ↗
1 Superficie de terreno 300 m<U+00B2> 19.3659096%2C-99.2449587
2 Superficie construida 25 m<U+00B2> 19.3659096%2C-99.2449587
3 Ba<U+00F1>os 1 19.3659096%2C-99.2449587
4 Superficie de terreno 20 m<U+00B2> Not available
5 Superficie construida 20 m<U+00B2> Not available
6 Ba<U+00F1>os 1 Not available

URL
1 https://inmueble.metroscubicos.com/MLM-688535411-rento-habitacion-con-bano-
junto-a-metro-x-polanco-_JM
2 https://inmueble.metroscubicos.com/MLM-688535411-rento-habitacion-con-bano-
junto-a-metro-x-polanco-_JM
3 https://inmueble.metroscubicos.com/MLM-688535411-rento-habitacion-con-bano-
junto-a-metro-x-polanco-_JM
4 https://inmueble.metroscubicos.com/MLM-690313303-suites-ejecutivas-tipo-contemporan
eo-en-tampico-tamps-mex-_JM
5 https://inmueble.metroscubicos.com/MLM-690313303-suites-ejecutivas-tipo-contemporan
eo-en-tampico-tamps-mex-_JM
6 https://inmueble.metroscubicos.com/MLM-690313303-suites-ejecutivas-tipo-contemporan
eo-en-tampico-tamps-mex-_JM
>

```

(Figura Caso 2.2) Se observan los datos brutos, es fácil apreciar que se necesitan limpiar.

Es posible conocer más información del archivo como el número de renglones con el comando `nrow()` o el número de columnas con el comando `ncol()`, o bien el tipo de variable del data frame con el comando `str()` o bien los nombres de los encabezados con `names()` (Ver figura Caso 2.3)

```

nrow(Inmuebles)
ncol(Inmuebles)
str(Inmuebles)
names(Inmuebles)

```

```
Console Terminal x
/Volumes/NO NAME/Tutorial R/Inmuebles/ ↗
> ncol(Inmuebles)
[1] 6
> nrow(Inmuebles)
[1] 65534
> str(Inmuebles)
'data.frame': 65534 obs. of 6 variables:
 $ Name      : Factor w/ 11411 levels "!!!! 0lv\u00eddate Del Stress Cita-dino Y Desfr
uta Del Aire Limpio Y Puro",..: 9817 9817 9817 10552 10552 10552 10552 9166 9166 9166
 ...
 $ Price     : Factor w/ 3612 levels "$1,000,000.00",..: 2012 2012 2012 1384 1384 1384
1384 2028 2028 2028 ...
 $ Attribute: Factor w/ 83 levels "\u00c1rea de juegos infantiles",..: 74 71 17 74 71
17 20 74 71 17 ...
 $ Value     : Factor w/ 4575 levels "-", "- m\u00b2",..: 2242 1901 27 1436 1436 27 27
1906 2041 27 ...
 $ Location  : Factor w/ 9573 levels "-12.0463731%2C-77.042754",..: 2029 2029 2029 957
3 9573 9573 9573 2744 2744 2744 ...
 $ URL       : Factor w/ 15401 levels "https://casa.metroscubicos.com/MLM-582716795-ca
sa-en-venta-fraccionamiento-bulevares-tuxtla-gutierrez-_JM",..: 12770 12770 12770 132
57 13257 13257 13257 15292 15292 15292 ...
> |
```

(Figura Caso 2.3) Información acerca del conjunto de datos.

Es posible que existan renglones duplicados, para comprobarlo se ejecuta el comando `unique()` anidado al comando `nrow()`: (Figura Caso 2.4)

```
Console Terminal x
/Volumes/NO NAME/Tutorial R/Inmuebles/ ↗
> nrow(Inmuebles)
[1] 65534
> nrow(unique(Inmuebles))
[1] 64247
> |
```

(Figura Caso 2.4)

Es evidente que hay líneas duplicadas por lo tanto se escribe en la ventana del editor el siguiente comando que asigna sólo las observaciones sin duplicados al objeto `Inmuebles` que se definió inicialmente, este comando deberá ser ejecutado en la consola de R: (Ver figura Caso 2.5)

```
Inmuebles <- unique(Inmuebles)
```

```

1  ###Inmuebles
2  #Conoceremos en que carpeta esta R
3  getwd()
4  #Definir escritorio de trabajo en carpeta con archivos
5  setwd("/Volumes/NO NAME/Tutorial R/Inmuebles")
6  #Listar archivos de la carpeta
7  list.files(getwd())
8  #Asignar a un objeto el archivo que leeremos
9  Inmuebles <- read.csv(file = "Inmuebles.csv", sep = ",",
10                       header = TRUE, encoding = 'UTF-8')
11
12 #Asignamos al objeto inmuebles las observaciones unicas
13 #o bien, se eliminan duplicados
14 Inmuebles <- unique(Inmuebles)
15
16
17
14:1 (Top Level) R Script

```

```

/Volumes/NO NAME/Tutorial R/Inmuebles/
> #Asignamos al objeto inmuebles las observaciones unicas
> #o bien, se eliminan duplicados
> Inmuebles <- unique(Inmuebles)
>

```

(Figura Caso 2.5) El código existe en el editor y se ejecuta en la ventana de abajo.

Según el resultado presentado en la figura Caso 2.3, todas las variables son factor, es decir, mezcla de números y caracteres por lo tanto no sería posible realizar cálculos como el promedio de precio (Ver figura Caso 2.6):

```
mean(Inmuebles$Price)
```

```

/Volumes/NO NAME/Tutorial R/Inmuebles/
> mean(Inmuebles$Price)
[1] NA
Warning message:
In mean.default(Inmuebles$Price) :
  argument is not numeric or logical: returning NA
>
>

```

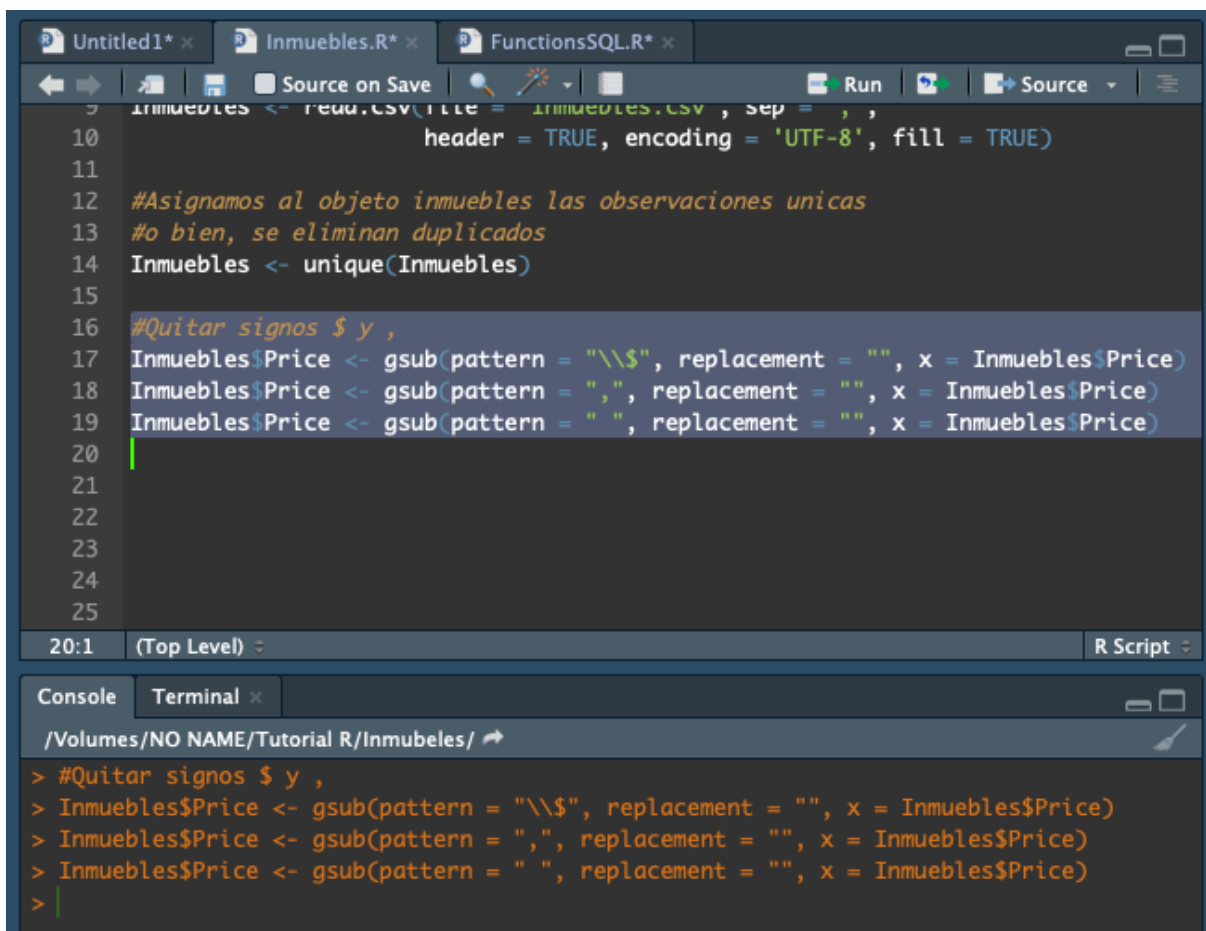
(Figura Caso 2.6) En esta configuración de Rstudio, los mensajes de error son de otro color.

La razón del error en el promedio es porque los valores de la variable no son numéricos, esto es porque hay caracteres con signos de "\$" y ",". Se recomienda usar la función `gsub()` Donde los argumentos son: (Ver figura Caso 2.7)

pattern Es la cadena de datos que se busca reemplazar
replacement Es el reemplazo de la cadena de caracteres
x Es el objeto al cual se le va a ejecutar la operación

Se escriben los siguientes comandos en la ventana del editor y se corren en la consola.

```
Inmuebles$Price <- gsub(pattern = "\\$", replacement = "", x =  
Inmuebles$Price)  
Inmuebles$Price <- gsub(pattern = ",", replacement = "", x =  
Inmuebles$Price)  
Inmuebles$Price <- gsub(pattern = " ", replacement = "", x =  
Inmuebles$Price)
```



```
Untitled1* x Inmuebles.R* x FunctionsSQL.R* x  
← → Source on Save Run Source  
9 Inmuebles <- read.csv(file = "Inmuebles.csv", sep = ",",  
10 header = TRUE, encoding = 'UTF-8', fill = TRUE)  
11  
12 #Asignamos al objeto inmuebles las observaciones unicas  
13 #o bien, se eliminan duplicados  
14 Inmuebles <- unique(Inmuebles)  
15  
16 #Quitar signos $ y ,  
17 Inmuebles$Price <- gsub(pattern = "\\$", replacement = "", x = Inmuebles$Price)  
18 Inmuebles$Price <- gsub(pattern = ",", replacement = "", x = Inmuebles$Price)  
19 Inmuebles$Price <- gsub(pattern = " ", replacement = "", x = Inmuebles$Price)  
20  
21  
22  
23  
24  
25  
20:1 (Top Level) R Script  
Console Terminal x  
/Volumes/NO NAME/Tutorial R/Inmuebles/  
> #Quitar signos $ y ,  
> Inmuebles$Price <- gsub(pattern = "\\$", replacement = "", x = Inmuebles$Price)  
> Inmuebles$Price <- gsub(pattern = ",", replacement = "", x = Inmuebles$Price)  
> Inmuebles$Price <- gsub(pattern = " ", replacement = "", x = Inmuebles$Price)  
>
```

(Figura Caso 2.7) Cada función gsub, sobrescribe al objeto inmuebles.

**Nota, al ser “\$” un carácter reservado para operaciones en R, se escribe la anti diagonal. Este problema pudo ser encontrado en stackoverflow¹⁵ de manera similar a la notación de horas del caso de estudio 1.

¹⁵ [How do I strip dollar signs \(\\$\) from data/ escape special characters in R? - Stack Overflow](https://stackoverflow.com/questions/10484000/how-do-i-strip-dollar-signs-from-data-escape-special-characters-in-r)

Si se intenta calcular el promedio de la variable Price, vuelve a haber un error ahora es porque la columna está en factores. Antes de convertirla a numérico es recomendable usar el comando `View()` para visualizar el set de datos completo y ordenar descendente y ascendente la variable para ver qué tipo de datos hay (Ver figura Caso 2.8)

	Name	Price	Attribute
52731	Casa En Venta, Los Cabos, Baja California Sur	US999500	Superficie de terreno
52732	Casa En Venta, Los Cabos, Baja California Sur	US999500	Superficie construida
52733	Casa En Venta, Los Cabos, Baja California Sur	US999500	Rec<U+00E1>maras
52734	Casa En Venta, Los Cabos, Baja California Sur	US999500	Ba<U+00F1>os
52735	Casa En Venta, Los Cabos, Baja California Sur	US999500	Estacionamientos
52736	Casa En Venta, Los Cabos, Baja California Sur	US999500	Cuota mensual de mantenimiento
52737	Casa En Venta, Los Cabos, Baja California Sur	US999500	Pisos
4577	Rancho La Esmeralda En Venta, San Miguel De Allende...	US995000	Superficie de terreno
4578	Rancho La Esmeralda En Venta, San Miguel De Allende...	US995000	Hect<U+00E1>reas totales
4579	Rancho La Esmeralda En Venta, San Miguel De Allende...	US995000	Rec<U+00E1>maras
4580	Rancho La Esmeralda En Venta, San Miguel De Allende...	US995000	Ba<U+00F1>os

Showing 1 to 11 of 65,534 entries

```

Console Terminal x
/Volumes/NO NAME/Tutorial R/Inmuebles/ ↗
> View(Inmuebles)
>
  
```

(Figura Caso 2.8) Los datos son ordenados por el valor de Price.

¿Qué hacer con las columnas que están en dólares?

¿Cómo identificar las columnas en dólares para hacer una operación si son dólares y otra si no?

Es posible darse cuenta que hay celdas con US, indicando que son precios en dólares. La función `grep()` evalúa en una lista la existencia de un "pattern" o patrón de manera similar a `gsub()` los argumentos son los siguientes:

- pattern* Es la cadena de datos que se busca reemplazar
- x* Es el objeto al cual se le va a ejecutar la operación

Corriendo el siguiente comando podemos ver que el resultado de la función `grep()` indica de los 64K renglones, cuales contienen "US" en modo booleano es decir en Falso y Verdadero: (Ver figura Caso 2.9)

```
grep(pattern = "US", x = Inmuebles$Price)
```

```

Console Terminal x
/Volumes/NO NAME/Tutorial R/Inmubeles/ ↗
> #Conocer en que observaciones del objeto inmueble
> #dicen "US" en la variable price
> grepl(pattern = "US",x = Inmuebles$Price)
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [49] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [85] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [97] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[109] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[121] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[145] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[157] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

```

(Figura Caso 2.9) Resultado de la evaluación de cada uno de los elementos del vector Inmuebles\$Price.

La función *which()* regresa el index, renglón o número de observación en donde el vector sea TRUE, se anida la función *grepl()* dentro de *which()* (Ver figura Caso 2.10)

```
which(grepl(pattern = "US",x = Inmuebles$Price))
```

```

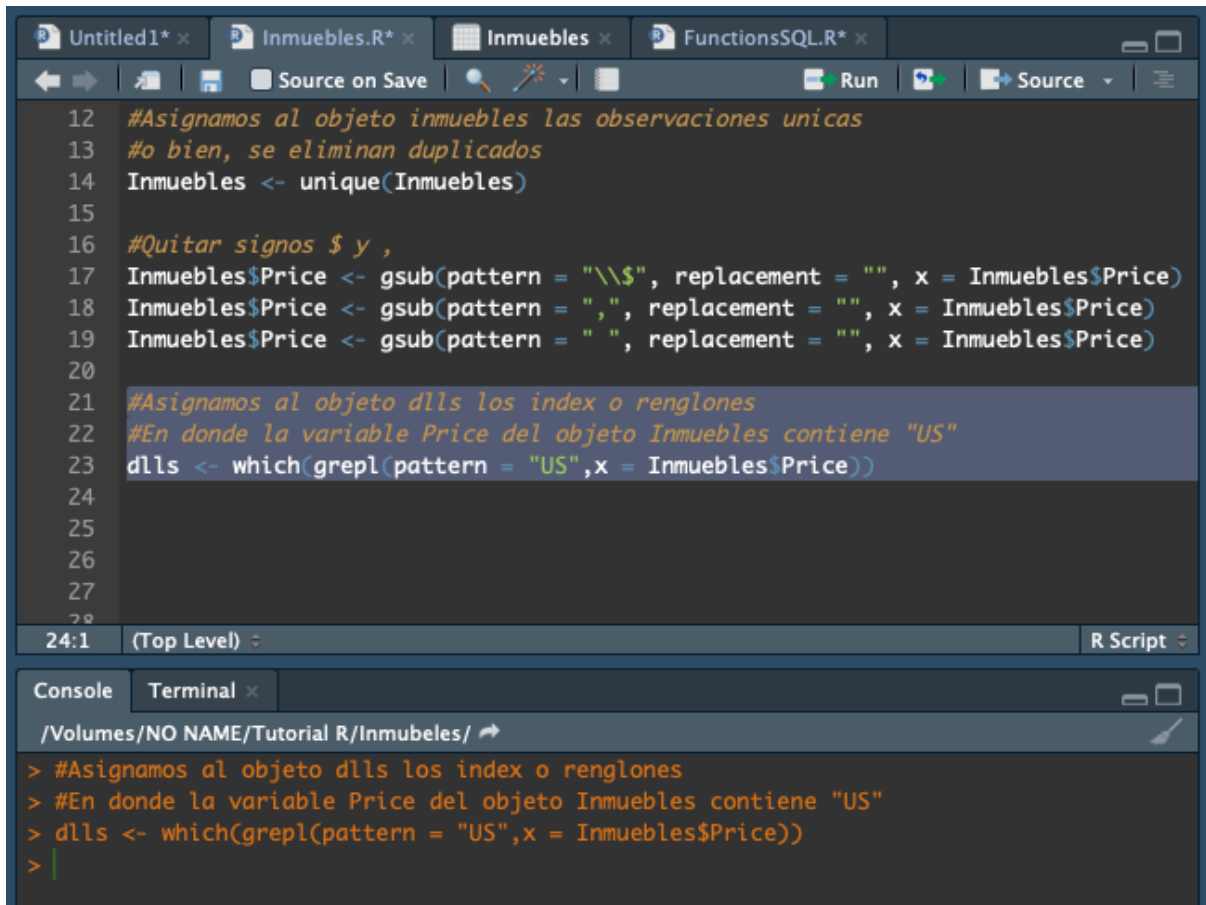
Console Terminal x
/Volumes/NO NAME/Tutorial R/Inmubeles/ ↗
> #Conocer el index de las observaciones en donde
> #grepl es verdadero
> which(grepl(pattern = "US",x = Inmuebles$Price))
 [1] 460 461 462 463 1613 1614 1615 1616 1617 1618 1619 1620
 [13] 1628 1629 1630 1631 1632 1654 1655 1656 1657 1658 1659 1674
 [25] 1675 1676 1677 1678 1679 1680 1681 1810 1811 1812 1813 1814
 [37] 1815 1816 1817 1818 1819 2071 2072 2073 2074 2075 2076 2077
 [49] 2078 2079 2216 2217 2218 2219 2220 2221 2222 2254 2255 2256
 [61] 2257 2258 2259 2260 2269 2270 2271 2272 2273 2274 2275 2276
 [73] 2277 2278 2279 2280 2306 2307 2308 2309 2310 2311 2312 2313
 [85] 2329 2330 2331 2332 2333 2334 2392 2393 2394 2449 2450 2451
 [97] 2452 2453 2454 2455 2468 2469 2470 2587 2588 2589 2608 2609
[109] 2610 2611 2612 2639 2640 2641 2642 2643 2644 2645 2717 2718
[121] 2878 2879 2880 2881 2882 2883 2884 2885 2908 2909 2910 2911
[133] 2912 2913 2914 2915 2916 2917 2918 2919 2920 2921 2922 2923
[145] 2931 2932 2933 2934 2935 2936 2937 2960 2961 2962 2963 3045
[157] 3046 3047 3048 3387 3388 3389 3390 3437 3438 3463 3464 3465

```

(Ver figura Caso 2.10) Se aprecia en el index las observaciones que fueron evaluadas como TRUE.

Ahora que se conoce cuál es el index, renglón u observación que contiene US, es posible asignar esa lista a un objeto al cuál se le llamará *dlls*. Se escribe el comando en la ventana del editor y se corre en la consola de R (Ver figura Caso 2.11)

```
dlls <- which(grepl(pattern = "US",x = Inmuebles$Price))
```



The screenshot shows the R Studio interface. The top pane is the source editor with the following code:

```
12 #Asignamos al objeto inmuebles las observaciones unicas
13 #o bien, se eliminan duplicados
14 Inmuebles <- unique(Inmuebles)
15
16 #Quitar signos $ y ,
17 Inmuebles$Price <- gsub(pattern = "\\$", replacement = "", x = Inmuebles$Price)
18 Inmuebles$Price <- gsub(pattern = ",", replacement = "", x = Inmuebles$Price)
19 Inmuebles$Price <- gsub(pattern = " ", replacement = "", x = Inmuebles$Price)
20
21 #Asignamos al objeto dlls los index o renglones
22 #En donde la variable Price del objeto Inmuebles contiene "US"
23 dlls <- which(grepl(pattern = "US",x = Inmuebles$Price))
24
25
26
27
28
```

The bottom pane is the console, showing the execution of the code:

```
/Volumes/NO NAME/Tutorial R/Inmuebles/ ↗
> #Asignamos al objeto dlls los index o renglones
> #En donde la variable Price del objeto Inmuebles contiene "US"
> dlls <- which(grepl(pattern = "US",x = Inmuebles$Price))
> |
```

(Figura Caso 2.11) A que el resultado de la función *which* se asigna a una variable, el resultado no se imprime en pantalla.

Teniendo escrito el index en la variable *dlls* se elimina el carácter "US" como se realizó con el signo "\$" y "," se escribe el comando en el editor y se corre en la consola de R: (Ver figura Caso 2.12)

```
Inmuebles$Price <- gsub(pattern = "US", replacement = "", x =
Inmuebles$Price)
```

```

15
16 #Quitar signos $ y ,
17 Inmuebles$Price <- gsub(pattern = "\\$", replacement = "", x = Inmuebles$Price)
18 Inmuebles$Price <- gsub(pattern = ",", replacement = "", x = Inmuebles$Price)
19 Inmuebles$Price <- gsub(pattern = " ", replacement = "", x = Inmuebles$Price)
20
21 #Asignamos al objeto d1ls los index o renglones
22 #En donde la variable Price del objeto Inmuebles contiene "US"
23 d1ls <- which(grepl(pattern = "US",x = Inmuebles$Price))
24
25 #Sabido que tenemos escrito el index en d1ls podemos
26 #Eliminar el caracter "US"
27 Inmuebles$Price <- gsub(pattern = "US", replacement = "", x = Inmuebles$Price)
28
29
30
31

```

28:1 (Top Level) R Script

```

Console Terminal
/Volumes/NO NAME/Tutorial R/Inmuebles/
> #Sabido que tenemos escrito el index en d1ls podemos
> #Eliminar el caracter "US"
> Inmuebles$Price <- gsub(pattern = "US", replacement = "", x = Inmuebles$Price)
>

```

(Figura Caso 2.12) Ahora ya no hay observaciones con el carácter US en los precios.

Sin el carácter "US", es posible multiplicar por el tipo de cambio los precios de las observaciones en dólares (d1ls)

En un data frame usamos \$ para llamar una columna y entre [] ponemos el index o renglón de los valores que sabemos que están en dólares; por lo tanto la siguiente expresión podrá ser leída como se indica:

```
Inmuebles$Price[d1ls]
```

Objeto: Inmuebles
 Columna: Price
 Renglones: elementos en la variable d1ls

De forma equivalente la notación matricial genera el mismo resultado:

```
Inmuebles[d1ls,2]
```

Para este ejemplo, se sabe que la columna price es la segunda del Data frame por lo tanto usando la nomenclatura arriba mencionada, se escribe el siguiente comando en la ventana del editor y se corre en la consola de R: (Ver figura Caso 2.13)

```
19 Inmuebles$Price <- gsub(pattern = " ", replacement = "", x = Inmuebles$Price)
20
21 #Asignamos al objeto d1ls los index o renglones
22 #En donde la variable Price del objeto Inmuebles contiene "US"
23 d1ls <- which(grepl(pattern = "US",x = Inmuebles$Price))
24
25 #Sabido que tenemos escrito el index en d1ls podemos
26 #Eliminar el caracter "US"
27 Inmuebles$Price <- gsub(pattern = "US", replacement = "", x = Inmuebles$Price)
28
29
30 #Sin el caracter "US", podemos multiplicar por el tipo de cambio
31 #los precios de las observaciones en dolares (d1ls)
32 Inmuebles$Price[d1ls] <- as.numeric(Inmuebles$Price[d1ls])*19
33
34
35
```

30:1 (Top Level) R Script

Console Terminal

```
/Volumes/NO NAME/Tutorial R/Inmuebles/
> #Sin el caracter "US", podemos multiplicar por el tipo de cambio
> #los precios de las observaciones en dolares (d1ls)
> Inmuebles$Price[d1ls] <- as.numeric(Inmuebles$Price[d1ls])*19
>
```

(Figura Caso 2.13) Operación de multiplicar por el tipo de cambio únicamente las columnas etiquetadas con dólares.

```
Inmuebles$Price[d1ls] <- as.numeric(Inmuebles$Price[d1ls])*19
```

Ahora es posible asignar el tipo de variable numérico a toda la columna usando el comando `as.numeric()`, se escribe el siguiente comando en la ventana del editor y se corre en la consola de R: (Ver figura Caso 2.14)

```
Inmuebles$Price <- as.numeric(Inmuebles$Price)
```

```

24
25 #Sabido que tenemos escrito el index en dlls podemos
26 #Eliminar el caracter "US"
27 Inmuebles$Price <- gsub(pattern = "US", replacement = "", x = Inmuebles$Price)
28
29
30 #Sin el caracter "US", podemos multiplicar por el tipo de cambio
31 #los precios de las observaciones en dolares (dlls)
32 Inmuebles$Price[dlls] <- as.numeric(Inmuebles$Price[dlls])*19
33
34 #Asignamos el tipo de variable numerico a toda la columna
35 Inmuebles$Price <- as.numeric(Inmuebles$Price)
36
37
38
39
40
34:1 (Top Level) R Script
Console Terminal
/Volumes/NO NAME/Tutorial R/Inmubeles/
> #Asignamos el tipo de variable numerico a toda la columna
> Inmuebles$Price <- as.numeric(Inmuebles$Price)
>

```

(Figura Caso 2.14) Al haber limpiado la columna Price por completo, ya no hay mensaje de error por cambiar a formato numérico.

Continuando el análisis exploratorio, es posible notar que la latitud y longitud son separadas por la cadena de caracteres "%2C" : (Ver figura Caso 2.15)

```
head(Inmuebles$Location)
```

```

/Volumes/NO NAME/Tutorial R/Inmubeles/
> head(Inmuebles$Location)
[1] 19.3659096%2C-99.2449587 19.3659096%2C-99.2449587 19.3659096%2C-99.2449587
[4] Not available Not available Not available
9573 Levels: -12.0463731%2C-77.042754 ... Not available
>

```

(Figura Caso 2.15)

Para separar una cadena de texto en dos partes, se usa la función `strsplit()`, sin embargo sólo funciona con variables de tipo `character`, es posible anidar la función `as.character()` dentro de `strsplit()` se puede ver el encabezado de esta operación con el siguiente comando: (Ver figura Caso 2.16)

```
head(strsplit(as.character(Inmuebles$Location), "%2C"))
```

```
Console Terminal x
/Volumes/NO NAME/Tutorial R/Inmuebles/ ↗
> head(strsplit(as.character(Inmuebles$Location),"%2C"))
[[1]]
[1] "19.3659096" "-99.2449587"

[[2]]
[1] "19.3659096" "-99.2449587"

[[3]]
[1] "19.3659096" "-99.2449587"

[[4]]
[1] "Not available"

[[5]]
[1] "Not available"

[[6]]
[1] "Not available"
```

(Figura Caso 2.16) Valores de locación separados por los caracteres %2C.

Es posible observar que la separación de la cadena de caracteres fué exitosa, sin embargo el resultado es una lista de vectores *list*, mientras que el objeto *inmuebles* es un *data frame* para resolver eso se usa la función *do.call()* con el argumento *rbind* para tener un *data frame*: (Ver figura Caso 2.17)

```
LatLong <-
do.call(rbind,strsplit(as.character(Inmuebles$Location),"%2C")
)
Inmuebles$Latitude <- LatLong[,1]
Inmuebles$Longitude <- LatLong[,2]
```

```

29 #Eliminar el caracter "US"
30 Inmuebles$Price <- gsub(pattern = "US", replacement = "", x = Inmuebles$Price)
31
32 #Sin el caracter "US", podemos multiplicar por el tipo de cambio
33 #los precios de las observaciones en dolares (dolls)
34 Inmuebles$Price[dolls] <- as.numeric(Inmuebles$Price[dolls])*19
35
36 #Asignamos el tipo de variable numerico a toda la columna
37 Inmuebles$Price <- as.numeric(Inmuebles$Price)
38
39 #Se anida dentro de la función do.call la combinacion de objetos
40 #resultantes de separar Location en latitud y longitud
41 a <- do.call(rbind, strsplit(as.character(Inmuebles$Location), "%2C"))
42 Inmuebles$Latitude <- a[,1]
43 Inmuebles$Longitude <- a[,2]
44
45 #####

```

44:1 (Top Level) R Script

```

/Volumes/NO NAME/Tutorial R/Inmubeles/
> #Se anida dentro de la función do.call la combinacion de objetos
> #resultantes de separar Location en latitud y longitud
> a <- do.call(rbind, strsplit(as.character(Inmuebles$Location), "%2C"))
> Inmuebles$Latitude <- a[,1]
> Inmuebles$Longitude <- a[,2]
>

```

(Figura Caso 2.17) Se generan 2 nuevas variables asignando el resultado de separación de caracteres.

4.4 Código del Caso

Código en lenguaje R del caso 2

```

###Inmuebles
#Conoceremos en que carpeta esta R
getwd()

#Definir escritorio de trabajo en carpeta con archivos
setwd("/Volumes/NO NAME/Tutorial R/Inmubeles")

#Listar archivos de la carpeta
list.files(getwd())

#Asignar a un objeto el archivo que leeremos
Inmuebles <- read.csv(file = "Inmuebles.csv", sep = ",",
                      header = TRUE, encoding = 'UTF-8', fill = TRUE)

#Asignamos al objeto inmuebles las observaciones unicas
#o bien, se eliminan duplicados

```

```

Inmuebles <- unique(Inmuebles)

#Quitar signos $ y ,
Inmuebles$Price <- gsub(pattern = "\\$", replacement = "", x =
Inmuebles$Price)
Inmuebles$Price <- gsub(pattern = ",", replacement = "", x =
Inmuebles$Price)
Inmuebles$Price <- gsub(pattern = " ", replacement = "", x =
Inmuebles$Price)

#Asignamos al objeto dlls los index o renglones
#En donde la variable Price del objeto Inmuebles contiene "US"
dlls <- which(grepl(pattern = "US",x = Inmuebles$Price))

#Sabido que tenemos escrito el index en dlls podemos
#Eliminar el caracter "US"
Inmuebles$Price <- gsub(pattern = "US", replacement = "", x =
Inmuebles$Price)

#Sin el caracter "US", podemos multiplicar por el tipo de cambio
#los precios de las observaciones en dolares (dlls)
Inmuebles$Price[dlls] <- as.numeric(Inmuebles$Price[dlls])*19

#Asignamos el tipo de variable numerico a toda la columna
Inmuebles$Price <- as.numeric(Inmuebles$Price)

#Se anida dentro de la función do.call la combinacion de objetos
#resultantes de separar Location en latitud y longitud
a <- do.call(rbind, strsplit(as.character(Inmuebles$Location), "%2C"))
Inmuebles$Latitude <- a[,1]
Inmuebles$Longitude <- a[,2]

```

5. Caso de Estudio 3: Horarios de Películas

5.1 Objetivo

El objetivo en este caso de estudio es trabajar con la librería *Tidyverse* para poder conocer información de valor. Se limpiará un archivo y crear una función que limpie archivos similares y escriba en una base de datos.

Una vez creada la base de datos, se deberán de resolver ciertas preguntas de negocio

5.2 Análisis Exploratorio de Datos

Se realizará un análisis exploratorio con uno de los archivos proporcionados (Ver figura Caso 3.1)

```
#Cargar primer archivo
```

```

setwd("D:/")
list.files(getwd())
cines1 <- read.csv(file = list.files(getwd())[1],sep =
",",header = FALSE)
#Breve vistazo a los datos
head(cines1,10)
tail(cines1,10)
View(cines1)
names(cines1)

```

```

D:/New folder/
> list.files(getwd())
[1] "CNPL081119.csv"
> cines1 <- read.csv(file = list.files(getwd())[1],sep = ",",header = FALSE, encoding="UTF-8", stringsAsFactors=FALSE)
> #Breve vistazo a los datos
> head(cines1,10)

      V1
1      1+CIN<c9>POLIS PLAZA AC<c1>MBARONUEVA APERTURA+D<ed>a de Muertos+ESP+16:35
2      1+CIN<c9>POLIS PLAZA AC<c1>MBARONUEVA APERTURA+Doctor Sue<f1>o+ESP+15:30
3      1+CIN<c9>POLIS PLAZA AC<c1>MBARONUEVA APERTURA+Doctor Sue<f1>o+ESP+15:30
4      1+CIN<c9>POLIS PLAZA AC<c1>MBARONUEVA APERTURA+Jugando con Fuego+ESP+15:00 17:00 19:05 21:10
5      1+CIN<c9>POLIS PLAZA AC<c1>MBARONUEVA APERTURA+Los Locos Addams+ESP+15:15 17:10 19:05 21:00
6      1+CIN<c9>POLIS PLAZA AC<c1>MBARONUEVA APERTURA+Los Bedrosqueros y el Mago de All...

```

(Figura Caso 3.1) En la exploración de datos, se observa un error de encoding.

Es fácil apreciar que el archivo no es legible, incluso usando el comando View observamos lo siguiente: (Ver figura Caso 3.2)

	V1
1	1+CIN<c9>POLIS PLAZA AC<c1>MBARONUEVA APERTURA..
2	1+CIN<c9>POLIS PLAZA AC<c1>MBARONUEVA APERTURA..
3	1+CIN<c9>POLIS PLAZA AC<c1>MBARONUEVA APERTURA..
4	1+CIN<c9>POLIS PLAZA AC<c1>MBARONUEVA APERTURA..
5	1+CIN<c9>POLIS PLAZA AC<c1>MBARONUEVA APERTURA..
6	1+CIN<c9>POLIS PLAZA AC<c1>MBARONUEVA APERTURA..
7	1+CIN<c9>POLIS PLAZA AC<c1>MBARONUEVA APERTURA..
8	1+CIN<c9>POLIS PLAZA AC<c1>MBARONUEVA APERTURA..
9	2+CIN<c9>POLIS VIP GALER<cd>AS DIANA ACAPULCO+D..
10	2+CIN<c9>POLIS VIP GALER<cd>AS DIANA ACAPULCO+Es..
11	2+CIN<c9>POLIS VIP GALER<cd>AS DIANA ACAPULCO+Es..

Showing 1 to 11 of 5,379 entries

(Ver figura Caso 3.2) Aparentemente todos los datos están en una columna

Se puede observar que las columnas están separadas por el signo + y que hay caracteres especiales como Ñ o acentos; por lo tanto ajustamos la instrucción para leer el archivo:
(Ver figura Caso 3.3)

```
#Podemos Ahora volver a cargar el archivo y corregir lo que
observamos
cines1 <- read.table(file = "CNPL081119.csv", sep = "+", header
= FALSE, stringsAsFactors = FALSE, fill = TRUE)
unique(cines1[,4])
levels(cines1[,4])
head(cines1)
```

```
> #Podemos Ahora volver a cargar el archivo y corregir lo que observamos
> cines1 <- read.table(file = "CNPL081119.csv", sep = "+", header = FALSE, stringsAsFactor
s = FALSE, fill = TRUE)
> head(cines1,10)
```

	V1	V2	V3	V4
1	1 CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA		Día de Muertos	ESP
2	1 CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA		Doctor Sueño	ESP
3	1 CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA		Doctor Sueño	SUB
4	1 CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA		Jugando con Fuego	ESP
5	1 CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA		Los Locos Addams	ESP
6	1 CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA	Los Rodríguez y el Más Allá		ESP
7	1 CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA	Maléfica: Dueña Del Mal		ESP
8	1 CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA	Terminator: Destino Oscuro		ESP
9	2 CINÉPOLIS VIP GALERÍAS DIANA ACAPULCO		Doctor Sueño	SUB
10	2 CINÉPOLIS VIP GALERÍAS DIANA ACAPULCO	Estafadoras de Wall Street		SUB

(Figura Caso 3.3) Se aprecia de forma legible el contenido del archivo

```
#Podemos conocer los valores únicos de cada columna
unique(cines1[,4])
```

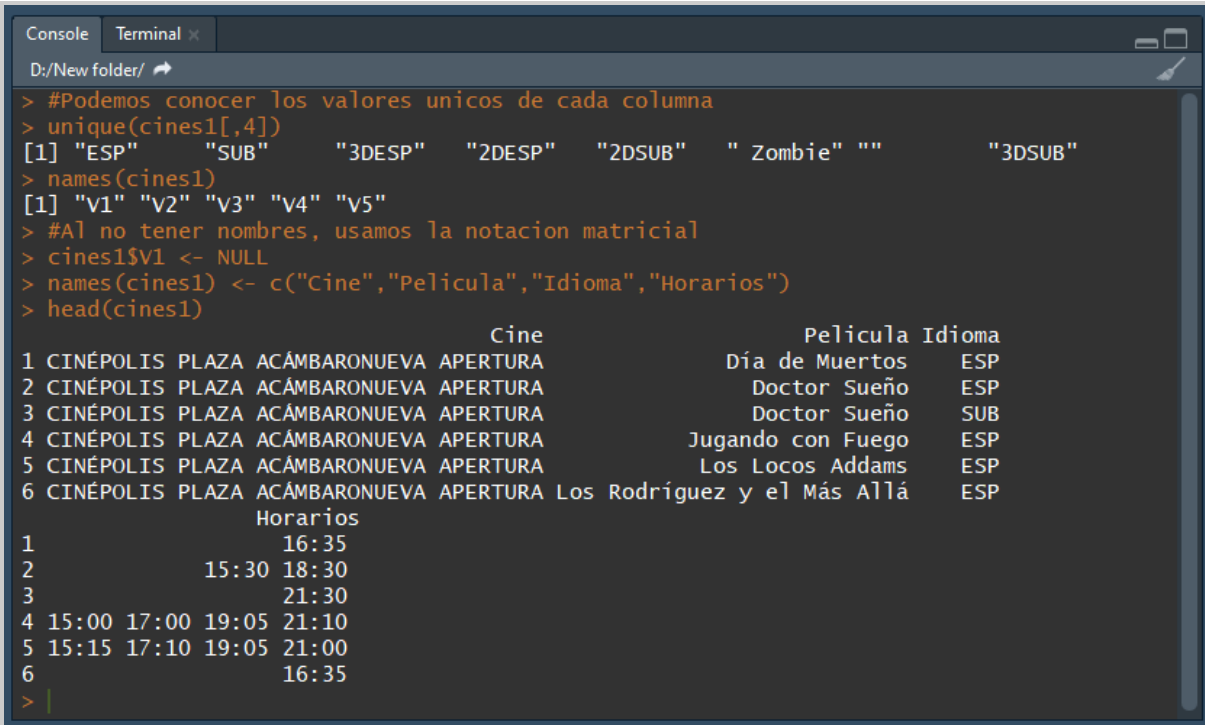
```
names(cines1)
```

```
#Al no tener nombres, usamos la notación matricial
```

Así mismo, sabemos que la columna V1 no tiene información útil para el análisis, le podemos asignar NULL para eliminar la columna, y después nombrar a las columnas con nombres que le corresponden según el dato: (Ver figura Caso 3.4)

```
cines1$V1 <- NULL
```

```
names(cines1) <- c("Cine", "Película", "Idioma", "Horarios")
```



```
> #Podemos conocer los valores únicos de cada columna
> unique(cines1[,4])
[1] "ESP" "SUB" "3DESP" "2DESP" "2DSUB" " Zombie" "" "3DSUB"
> names(cines1)
[1] "v1" "v2" "v3" "v4" "v5"
> #Al no tener nombres, usamos la notación matricial
> cines1$V1 <- NULL
> names(cines1) <- c("Cine", "Película", "Idioma", "Horarios")
> head(cines1)
```

	Cine	Película	Idioma
1	CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA	Día de Muertos	ESP
2	CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA	Doctor Sueño	ESP
3	CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA	Doctor Sueño	SUB
4	CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA	Jugando con Fuego	ESP
5	CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA	Los Locos Addams	ESP
6	CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA	Los Rodríguez y el Más Allá	ESP

	Horarios
1	16:35
2	15:30 18:30
3	21:30
4	15:00 17:00 19:05 21:10
5	15:15 17:10 19:05 21:00
6	16:35

(Figura Caso 3.4) Observando el resultado que devuelve la función unique, es posible darse cuenta de que hay entradas con errores.

Particularmente hay espacios vacíos y la palabra zombies: (Ver figura Caso 3.5)

```
#Eliminar vacíos y zombies
```

```
#Usando which, grepl y operadores condicionales
```

```
#grep para saber si hay "zombie" en la columna
```

```
unique(grepl(pattern = unique(cines1[,3])[6], x =
cines1$Idioma))
```

```

Console Terminal x
D:/New folder/
> #Eliminar vacios y zombies
> #Usando which, grepl y operadores condicionales
> #grep para saber si hay "zombie" en la colimna
> unique(grepl(pattern = unique(cines1[,3])[6],x = cines1$Idioma))
[1] FALSE TRUE
>

```

(Figura Caso 3.5) la evaluación de unique(cines1[,3])[6] es el carácter Zombie.

```

#Wich para conocer en que renglones hay TRUE
which(grepl(pattern = "Zombie", x = cines1$Idioma))

```

```

Console Terminal x
D:/New folder/
> #Wich para conocer en que renglones hay TRUE
> which(grepl(pattern = "Zombie", x = cines1$Idioma))
[1] 843 4161
>

```

(Figura Caso 3.6) Hay 2 observaciones con la palabra zombie.

```

#comprobando que si digan zombie
cines1[c(843,4161),]

```

```

Console Terminal x
D:/New folder/
> #comprobando que si digan zombie
> cines1[c(843,4161),]
      Cine      Pelicula Idioma Horarios
843      CINÉPOLIS SATÉLITE VR Engineerium  Zombie
4161 CINÉPOLIS ESFERA QUERÉTARO VR Engineerium  Zombie
>

```

(Figura Caso 3.7) Efectivamente son observaciones erróneas.

```

#Función anidada que elimina función
cines1 <- cines1[-c(which(grepl(pattern =
unique(cines1$Idioma)[6], x = cines1$Idioma))),]

```

```

Console Terminal x
D:/New folder/
> #Funcion anidada que elimina funcion
> cines1 <- cines1[-c(which(grepl(pattern = unique(cines1$Idioma)[6], x = cines1$Idioma)))
,]
> #Verificar espacios en blanco
> unique(cines1$Idioma)
[1] "ESP" "SUB" "3DESP" "2DESP" "2DSUB" "" "3DSUB"
> length(unique(cines1$Idioma))
[1] 7
> cines1 <- cines1[-c(which(cines1$Idioma=="")),]
>

```

(Figura Caso 3.8) Se eliminaron las observaciones con error.

```
#Verificar espacios en blanco
unique(cines1$Idioma)
length(unique(cines1$Idioma))
cines1 <- cines1[-c(which(cines1$Idioma=="")),]
```

```
Console Terminal x
D:/New folder/
> #Verificar espacios en blanco
> unique(cines1$Idioma)
[1] "ESP" "SUB" "3DESP" "2DESP" "2DSUB" "3DSUB"
> length(unique(cines1$Idioma))
[1] 6
> |
```

(Figura Caso 3.9) Se eliminaron las observaciones con espacios vacíos.

```
#separar horarios por columna con espacio
horarios <- strsplit(as.character(cines1$Horarios)," ")
horarios <- do.call(rbind, horarios)
cines1 <- data.frame(cines1, horarios)
```

```
Console Terminal x
D:/New folder/
> #separar horarios por columna con espacio
> horarios <- strsplit(as.character(cines1$Horarios)," ")
> horarios <- do.call(rbind, horarios)
Warning message:
In (function (... , deparse.level = 1) :
number of columns of result is not a multiple of vector length (arg 4)
> cines1 <- data.frame(cines1, horarios)
> head(cines1)
      Cine Película Idioma
1 CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA Día de Muertos ESP
2 CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA Doctor Sueño ESP
3 CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA Doctor Sueño SUB
4 CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA Jugando con Fuego ESP
5 CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA Los Locos Addams ESP
6 CINÉPOLIS PLAZA ACÁMBARONUEVA APERTURA Los Rodríguez y el Más Allá ESP
      Horarios X1 X2 X3 X4 X5 X6 X7 X8 X9
1 16:35 16:35 16:35 16:35 16:35 16:35 16:35 16:35 16:35 16:35 16:35
2 15:30 18:30 15:30 18:30 15:30 18:30 15:30 18:30 15:30 18:30 15:30
3 21:30 21:30 21:30 21:30 21:30 21:30 21:30 21:30 21:30 21:30 21:30
4 15:00 17:00 19:05 21:10 15:00 17:00 19:05 21:10 15:00 17:00 19:05 21:10 15:00
5 15:15 17:10 19:05 21:00 15:15 17:10 19:05 21:00 15:15 17:10 19:05 21:00 15:15
6 16:35 16:35 16:35 16:35 16:35 16:35 16:35 16:35 16:35 16:35 16:35
      X10 X11 X12 X13 X14
1 16:35 16:35 16:35 16:35 16:35
2 18:30 15:30 18:30 15:30 18:30
3 21:30 21:30 21:30 21:30 21:30
4 17:00 19:05 21:10 15:00 17:00
5 17:10 19:05 21:00 15:15 17:10
6 16:35 16:35 16:35 16:35 16:35
> |
```

(Figura Caso 3.10) Se asigna un nombre a cada nueva variable.

Sin embargo podemos ver como se rellenan los espacios con el mismo horario, lo cuál estaría generando errores en la información, para corregir el error se tendría que volver a cargar el archivo, eliminar entradas con errores y separar los horarios por columna. Si se tienen escritas las instrucciones en la ventana del editor de forma ordenada, será muy fácil llegar a este punto. Con el siguiente comando, no se duplica información (Ver figura Caso 3.11)

```
library('plyr')
horarios <- strsplit(as.character(cines1$Horarios)," ")
horarios <- ldply(horarios, rbind)
cines1 <- data.frame(cines1, horarios)
```

```

> horarios <- strsplit(as.character(cines1$Horarios)," ")
> horarios <- ldply(horarios, rbind)
> cines1 <- data.frame(cines1, horarios)
> head(cines1)

```

	Cine				Horarios												Pelicula	Idioma		
					X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	X14		
1	CINÉPOLIS	PLAZA	ACÁMBARONUEVA	APERTURA															Día de Muertos	ESP
2	CINÉPOLIS	PLAZA	ACÁMBARONUEVA	APERTURA															Doctor Sueño	ESP
3	CINÉPOLIS	PLAZA	ACÁMBARONUEVA	APERTURA															Doctor Sueño	SUB
4	CINÉPOLIS	PLAZA	ACÁMBARONUEVA	APERTURA															Jugando con Fuego	ESP
5	CINÉPOLIS	PLAZA	ACÁMBARONUEVA	APERTURA															Los Locos Addams	ESP
6	CINÉPOLIS	PLAZA	ACÁMBARONUEVA	APERTURA															Los Rodríguez y el Más Allá	ESP

```

> head(cines1)

```

(Figura Caso 3.11) hay espacios vacios para películas con pocos horarios.

```
#Formato de horas
cines1[,5:18] <- lapply(cines1[,5:18], strptime,format =
"%H:%M")
head(cines1)
```

5.3 Preguntas de Negocio

De la misma manera que en el caso 1, se podrán correr los siguientes bloques de código para resolver preguntas de negocio alrededor de los horarios de las películas en cines, se deja este ejercicio al lector

```
#total de horarios por formato
#cines con mas formatos de un tipo
#ordenar tablas con horario ascendente o descendente
```

```
library(dplyr)
```

```
horarios.pelicula <- (cines1 %>%
  select(Pelicula,Horarios)%>%
  group_by(Pelicula)%>%
  summarise(cuenta =
n_distinct(Horarios))
)
```

```
head(horarios.pelicula)
```

```
horarios.Cine <- (cines1 %>%
  select(Cine,Horarios)%>%
  group_by(Cine)%>%
  summarise(cuenta =
n_distinct(Horarios))
)
head(horarios.Cine)
```

6. Conclusiones

Como se pudo apreciar a lo largo del tutorial y durante las sesiones que fueron impartidas durante los semestres 2019-1 y 2019-2 de estas notas, la mejor manera de aprender un lenguaje de programación (en este caso R) y orientarlo hacia la inteligencia de negocios, es tomar un problema real y resolverlo. Distintos problemas van a encontrarse con grados de madurez de datos diferentes, es decir, habrá situaciones en donde los datos se encuentren “limpios” en una base de datos en donde la tarea será conectarse a dicha base para empezar a trabajar con la información; en otros casos, los datos se podrán encontrar dentro de carpetas con varios archivos históricos en formato excel en donde la tarea será hacer una curaduría de los datos mediante la automatización del proceso de leer de los archivos, extraer los datos necesarios y validarlos; o bien, los datos tendrán que buscarse o empezar a generarse desde cero.

Independientemente de la madurez o accesibilidad en la que se encuentren los datos, a la persona que busque implementar una inteligencia de negocio a partir del análisis de datos, le va a corresponder realizar una correcta abstracción del problema de negocio, interpretar los datos, plantear un modelo, visualizar la información, y generar hallazgos de valor para el negocio a partir del análisis que se realice y así poder comunicar de manera efectiva el diagnóstico de los siguientes pasos a tomar al resto del equipo y miembros de la organización.

Es posible que este tutorial tenga ciertas carencias en cuanto a profundidad de los temas, debido a que se enfoca en la resolución de problemas específicos y no abarca todas las funcionalidades del lenguaje de programación. Sin embargo, es importante tener en cuenta que este tutorial es solo una introducción al tema y puede ser utilizado como un punto de partida para profundizar en temas específicos de acuerdo a las necesidades del usuario. Además, el tutorial proporciona una buena comprensión de los conceptos clave necesarios para comenzar a aplicar el lenguaje de programación R en la inteligencia de negocios, como la limpieza y manipulación de datos, la visualización de información y la interpretación de resultados. Estos conceptos son fundamentales para cualquier proyecto de inteligencia de negocios y proporcionan una base sólida para profundizar en temas específicos.

En resumen, el tutorial del Lenguaje de Programación R orientado a la inteligencia de negocios proporciona herramientas valiosas para aquellos interesados en aplicar análisis de datos a problemas de negocio. A través de la resolución de problemas reales, el tutorial enfatiza la importancia de la abstracción, interpretación y visualización de los datos para obtener hallazgos valiosos para la organización. A pesar de que los datos pueden encontrarse en diferentes estados de madurez, es crucial que el analista sea capaz de manipularlos y utilizarlos para comunicar de manera efectiva el diagnóstico de los siguientes pasos a tomar al resto del equipo y miembros de la organización. En definitiva, el tutorial ofrece una guía práctica para aquellos que buscan aplicar el Lenguaje de Programación R a la inteligencia de negocios.

7. Referencias de Fuentes Consultadas

- 1.- *Online Courses - Learn Anything, On Your Schedule | Udemy*. (n.d.). Udemy. <https://www.udemy.com/>
Analytics Vidhya. (n.d.). Analytics Vidhya. <https://courses.analyticsvidhya.com/>
Learn R | Codecademy. (n.d.). Codecademy. <https://www.codecademy.com/learn/learn-r>
- 2.- *Stack Overflow - Where Developers Learn, Share, & Build Careers*. (n.d.). Stack Overflow. <https://stackoverflow.com/>
- 3.- *R: What is R?* (n.d.). R: What Is R? <https://www.r-project.org/about.html>
- 4.- *Simply Statistics: Teaching R to New Users - From tapply to the Tidyverse*. (2018, July 12). Simply Statistics. <https://simplystatistics.org/posts/2018-07-12-use-r-keynote-2018/>
- 5.- *Posit*. (n.d.). Posit. <https://www.posit.co/>
- 6.- *The Comprehensive R Archive Network*. (n.d.). The Comprehensive R Archive Network. <https://cran.r-project.org/>
- 7.- *G*. (n.d.). *Build software better, together*. GitHub. <https://github.com>
- 8.- *Tidyverse*. (n.d.). Tidyverse. <https://www.tidyverse.org/>
- 9.- Golemund, G., & Wickham, H. (2017). *R for Data Science*. O'Reilly Media.
- 10.- *Solutions - Setting up ODBC Drivers*. (n.d.). Solutions - Setting up ODBC Drivers. <https://db.rstudio.com/best-practices/drivers/>
- 11.- *seananderson.ca*. (n.d.). An Introduction to Reshape2 - Reshaping Data Easily With the Reshape2 R Package. - seananderson.ca. <https://seananderson.ca/2013/10/19/reshape/>
- 12.- *Converting excel DateTime serial number to R DateTime*. (2013, October 4). Stack Overflow. <https://stackoverflow.com/questions/19172632/converting-excel-datetime-serial-number-to-r-datetime>
- 13.- Wickham, H. (n.d.). *4 Pipes | The tidyverse style guide*. 4 Pipes | the Tidyverse Style Guide. <https://style.tidyverse.org/pipes.html>
- 14.- Zorrilla, J. P. (2017, December 26). *¿Rentar es tirar dinero a la basura?* Forbes. Retrieved December 5, 2019, from <https://www.forbes.com.mx/rentar-es-tirar-dinero-a-la-basura/>
- 15.- *How do I strip dollar signs (\$) from data/ escape special characters in R?* (2011, July 10). Stack Overflow. <https://stackoverflow.com/questions/6639713/how-do-i-strip-dollar-signs-from-data-escape-special-characters-in-r>

8. Bibliografía

- Grolemund, G., & Wickham, H. (2017). *R for Data Science*. O'Reilly Media.
- The R Foundation (n.d.). *The R Project for Statistical Computing*. R-Project. Retrieved December 5, 2019, from <https://www.r-project.org/>
- Posit Software (n.d.). *The open source data science company*. 2023 Posit Software, PBC Formerly RStudio, PBC. Retrieved December 5, 2019, from <https://www.rstudio.com/>
- Srivastava, T. (2015, April 26). *Comprehensive guide for Data Exploration in R*. Analytics Vidhya. Retrieved December 5, 2019, from <https://www.analyticsvidhya.com/blog/2015/04/comprehensive-guide-data-exploration-r/>
- Chou, L. (2019, November 7). *How Can Beginners Create a Great Dashboard? Towards Data Science*. Retrieved December 5, 2019, from <https://towardsdatascience.com/how-can-beginners-create-a-great-dashboard-cf48c0f68cd5>
- Kristiansen, S. L. (2018, March 16). *Create PDF reports using R, R Markdown, LaTeX and knitr (on macOS High Sierra)*. Medium. Retrieved December 5, 2019, from <https://medium.com/@sorenlind/create-pdf-reports-using-r-r-markdown-latex-and-knitr-on-macos-high-sierra-e7b5705c9fd>
- Posit Software (n.d.). *Posit Cheatsheets*. 2023 Posit Software, PBC Formerly RStudio, PBC. Retrieved December 5, 2019, from <https://www.rstudio.com/resources/cheatsheets/>
- Karn, U. (n.d.). *R Data Science Tutorials*. Github. Retrieved December 5, 2019, from <https://github.com/ujjwalkarn/DataScienceR>