



# R4Finance

R101

J. Gibrán Peniche

Versión 0.0.1

2020/06/05

 [jgpeniche](https://github.com/jgpeniche)

 [PenicheGibran](https://twitter.com/PenicheGibran)

[G.jgpeniche@gmail.com](mailto:G.jgpeniche@gmail.com)

# ¿Qué es R?

# ¿Qué es R?

R es un **language** de programación para computación estadística

- (No lo digo yo... está en su página) [r-project](https://www.r-project.org/)



# ¿Qué es RStudio?

# ¿Qué es RStudio?

- Es un **IDE**
- Esto es (versión corta) un editor de texto: R, C++, Python, Julia...
- Pero en realidad es (versión larga) Un ambiente de desarrollo:



En conclusión...





≠



nO son lo mismo

# R vs. Python: ¿Cuál es la diferencia?

R es un language de **dominio específico**  $\implies$  que ha tratado de migrar a un language de **dominio general**

- a. Shiny para aplicaciones y dashboards
- b. Rmarkdown para documentos word, pdf, powerpoint (o Xaringan como esta presentación), html, etc..

Python es un language de **dominio general**  $\implies$  que ha tratado de migrar hacia un language de **dominio específico**

- a. Pandas para manipulación de datos
- b. SciKitLearn para modelaje estadístico

# ¿Y en qué si son parecidos?

¡Son gratis!

Tanto como el language y el IDE son **open source**

Son bastante lentos

# ¿Por qué migrar hacia el open source?

## 3 razones

1



2





Microsoft

Microsoft 365

Productos ▾

Recursos ▾

Plantillas

Soporte técnico

Comprar ahora

Todo Microsoft ▾



# Encuentra la solución adecuada para ti

Para el hogar

Para empresas

Office 365 es ahora Microsoft 365. Nombre nuevo, más beneficios, mismo precio.

## Explorar Microsoft 365

¿Qué es Microsoft 365 para el hogar?

¿Buscas Office Hogar y Empresas 2019?

Microsoft 365 para estudiantes y educadores

Mejor rentabilidad: hasta 6 usuarios en PC o Mac

**Microsoft 365 Familia**  
(anteriormente Office 365 Hogar)

**MXN\$1,749.00**  
al año

**Microsoft 365 Personal**  
(anteriormente Office 365 Personal)

**MXN\$1,299.00**  
al año










**Office Hogar y Estudiantes 2019**

**MXN\$2,599.00**

3

View

PC > Documents > example

| Name ^   | Date modified         | Type               | Size     |
|--|-----------------------|--------------------|----------|
|  PDF Final Regional - Copy - Copy - Copy... | 02/04/2020 10:48 a.m. | Adobe Acrobat D... | 5,200 KB |
|  PDF Final Regional - Copy - Copy - Copy    | 02/04/2020 10:48 a.m. | Adobe Acrobat D... | 5,200 KB |
|  PDF Final Regional - Copy - Copy           | 02/04/2020 10:48 a.m. | Adobe Acrobat D... | 5,200 KB |
|  PDF Final Regional - Copy                  | 02/04/2020 10:48 a.m. | Adobe Acrobat D... | 5,200 KB |
|  PDF Final Regional - Final - Copy          | 02/04/2020 10:48 a.m. | Adobe Acrobat D... | 5,200 KB |
|  PDF Final Regional - Final-bueno           | 02/04/2020 10:48 a.m. | Adobe Acrobat D... | 5,200 KB |
|  PDF Final Regional - Final-Corregido       | 02/04/2020 10:48 a.m. | Adobe Acrobat D... | 5,200 KB |
|  PDF Final Regional - Final-FINAL           | 02/04/2020 10:48 a.m. | Adobe Acrobat D... | 5,200 KB |
|  PDF Final Regional                         | 02/04/2020 10:48 a.m. | Adobe Acrobat D... | 5,200 KB |

# Más razones...

4. No son reproducibles

5. Es difícil versionarlos

6. Son lentísimos

# Más razones

- 7. Los modelos complejos requieren de un esfuerzo titánico para programarse (¿alguna vez han programado una verosimilitud?)
- 8. Colaborar es muy difícil
- 9. Es difícil darles mantenimiento
- 10. Casi nunca están documentados

Razón  $\{n, n + 1, \dots\}$

¡ODIO EXCEL!

# En mi opinion...

Son populares porque se requiere ser mínimo funcional para utilizarlos

¿Qué ganamos si cambiamos a  
lenguajes de programación  
avanzados?



Separar la estructura (tabular) de los datos de la estructura (lineal) del ambiente del trabajo

# Más razones...

Potencia de computo

Control de Versiones

Modelos más complejos

Una apreciación en el orden de  $K \cdot 10^{-3}$  en su salario

# Entonces... ¿R o Python?

## 3 Razones para usar R y no Python

### 1 tidyverse

- a. Pipe operator `%>%` y sus amigos de `magrittr`
- b. Gramática de las visualizaciones con `ggplot2`
- c. sintáxis SQL para manipulación de datos tabulares `dplyr`

# Entonces... ¿R o Python?

## 3 Razones para usar R y no Python

2 `tidymodels`

- a. Resúmenes
- b. calibración y;
- c. ajuste de modelos todo en el esquema de **datos limpios** (Tidy Data)

# Entonces... ¿R o Python?

## 3 Razones para usar R y no Python

3 rstanarm

a. Integración entre R y la librería más estable para métodos bayesianos

¿Qué vamos a aprender en el curso?

# Temario

1. Sintáxis básica
2. Manipulación de datos con *dplyr*
3. Visualización de datos con *ggplot2*
4. Lectura de datos
5. Manipulación de datos financieros con *tidyquant*
6. Modelos estadísticos en el contexto de tidy data con *tidymodels*
7. Versionamiento semántico y programación orientada a proyectos
8. Reportes en RMarkdown
9. Apps y Dashboards con shiny

# 1. Sintaxis Básica



# Programación en R 101

```
# Declarar texto como comentario
```

```
# Asignar una variable
```

```
a <- 4
```

Utilizar '=' ó '<-' es básicamente lo mismo

Sin embargo, no puedes asignar argumentos de una función utilizando '<-'

Aunque son lo mismo es buena práctica asignar objetos con '<-'

+, -, /, \* todas funcionan de manera **vectorizada** sobre los objetos

# Más sintáxis básica

Creamos vectores con la función `c()` con cada elemento separado por una `,`

OJO también es posible crear vectores vacíos `x <- c()`

El tipo de los elementos puede ser:

1. Número ( `double` )
2. Texto ( `string` )
3. Boolean ( `TRUE` / `FALSE` )

# Más sintáxis básica

Ciclos *for*

```
for(i in algun_objeto){  
    # Hacer cosas  
}
```

# Más sintáxis básica

Declaracion *if*

```
if( alguna_condición ){  
# Hacer cosas  
}else{  
# Hacer más cosas  
}
```

# Más sintáxis básica

Declarar funciones

```
f <- function(x, y, z, ...){  
  # Hacer cosas  
  return(resultado)  
}
```

# *Primer* en funciones

Hay dos maneras de llamar argumentos de una función en R

Sea  $f(\alpha, \beta)$  una función

→ Podemos declarar los argumentos de una función

1. Por **nombre** `f(alpha = a, beta = b)`

2. Por **orden** `f(b, a)`

- Es buena práctica llamar los argumentos por nombre para que tu trabajo sea **legible y reproducible**

# Data Frames

Toda la infraestructura de R está diseñada para trabajar sobre una clase de objeto en particular: el **Data Frame**

Es alrededor de este objeto que se realiza todo el modelaje estadístico y la razón por la que R es ideal para hacer análisis de datos

```
# Requerimos una libreria
library(dplyr)

#Siempre recuerden sembrar su semilla!!!
set.seed(666)

#Creamos un Data Frame
df <- tibble(
  x_norm = rnorm( n = 100, mean = 0, sd = 1),
  x_exp = rexp(n = 100, rate = 1.5),
  x_seq = seq(from = 1, to = 7.897, length.out = 100)
)
```

Chela al que me diga que hacen las  
funciones

'rnorm', 'rexp', 'seq'



# Antes de continuar recuerden...

'?' es mi pastor

Todas las funciones y paquetes en R están documentadas

# Data Frames

Podemos acceder al contenido de un df de 2 maneras:

1 Con `[]` y llamando elementos por su posición dentro del df (la estructura básica es basicamente una matriz)

- ``abs(df[i, j])`` llama al valor en el renglón `*i*` y en la columna `*j*` y toma el valor absoluto
- ``mean(df[,column])`` llama a todos los renglones de la columna `*column*` y después tomamos la media
- ``sd(df[row,])`` llama a todas las columnas del renglón `*row*` y luego calcula su desviación estándar

# Data Frames

Podemos acceder al contenido de un df de 2 maneras:

2 Con el operador \$ que despliega una lista de los nombres de las variables contenidas en el df

- ``summary(df$x_norm)``

El hecho de que cada columna sea una variable es algo relevante que vale la pena discutir

# Datos limpios

En computación existe algo llamado la **tercera forma normal de Codd** que tiene que ver con bases de datos relacionadas, pero en términos estadísticos significa:

1. Cada variable es una columna
2. Cada renglón es una observación
3. Cada tabla es una unidad observacional

Todo el *framework* sobre el que vamos a trabajar parte del concepto de **tidy data**

La idea central de este tipo de forma de trabajar es que la **estructura** de los datos correspondan a la **semántica** de los mismos

Esto hace que el análisis y las funciones sean más eficientes

En la práctica, **80%** de su tiempo lo van a utilizar para llevar hojas de cálculo en excel a este formato

# Sintáxis Avanzada

# Pipe Operator

Una de las razones por la que R es tan poderoso para hacer análisis de datos es su **sintáxis**

El primer caso importante (de hecho en python se incorporó a pandas) es el **operador pipa** `%>%` de *magrittr*

En nuestros ejemplos anteriores se vería así:

```
mean(df$x_norm)
```

# Es equivalente a

```
library(magrittr)
```

```
df %>% mean(x_norm)
```

La *pipa* simplemente toma el argumento de la izquierda y lo usa como primer argumento de la función de la derecha

Esto hace el código mucho más *legible, mantenible y reproducible*

# Sintáxis Avanzada

Otro ejemplo de lo poderosa que es esta sintáxis es el *subsetting pipe*

```
cor(df$x_norm, df$x_exp)
```

Se puede escribir así:

```
df %>% cor(x_norm, x_exp)
```



# Exploremos algunos datos

1. Abre un nuevo script
2. Llama a las librerías *dplyr* y *magrittr* con la función `library()` (Si no tienes instalados los paquetes instálalos)
3. Llama a la memoria la base de datos 'EuStockMarkets' (Solo tienes que escribir el nombre y asignarlo porque la base de datos ya viene precargada en R) y asignala a una nueva variable como un Data Frame con la función `as_tibble()`
4. Revisa el contenido del objeto con la función `glimpse()` (recuerda usar la sintáxis avanzada)
5. Realiza un análisis exploratorio de datos con la función `str()`
6. Programa una función `retorno()` que reciba como argumento un vector precios y calcule los retornos
7. Prueba tu función con los precios del DAX

# Antes de empezar

## Pro Tip

Mucho a lo que te vas a enfrentar programando, es precisamente a no poder hacerlo

¡Usa las herramientas a tu disposición! La documentación de las funciones, buscar en StackOverflow, etc...

Corre tiempo...

# Exploremos algunos datos

1. Abre un nuevo script
2. Llama a las librerías *dplyr* y *magrittr* con la función `library()` (Si no tienes instalados los paquetes instálalos)
3. Llama a la memoria la base de datos 'EuStockMarkets' (Solo tienes que escribir el nombre y asignarlo porque la base de datos ya viene precargada en R) y asignala a una nueva variable como un Data Frame con la función `as_tibble()`
4. Revisa el contenido del objeto con la función `glimpse()` (recuerda usar la sintáxis avanzada)
5. Realiza un análisis exploratorio de datos con la función `summary()`
6. Programa una función `retorno()` que reciba como argumento un vector precios y calcule los retornos
7. Prueba tu función con los precios del DAX

10:00

# Solución

```
library(dplyr)
library(magrittr)

# Cargamos base de datos a la memoria

stocks <- EuStockMarkets %>%
  as_tibble() %>%
  janitor::clean_names()

stocks %>%
  glimpse()

stocks %>%
  summary()
```

# Solución

```
retorno <- function(precios){  
  # Creamos vector vacio de retornos  
  retornos <- c()  
  
  # Para un vector de n precios tenemos n - 1 retornos  
  for(i in 1:length(precios)){  
    if (i == 1){  
      retornos[i] <- 0  
    }else{  
      retornos[i] <- log(precios[i]) - log(precios[i-1])  
    }  
  }  
  return(retornos)  
}
```

# Solución

```
retornos_dax <- stocks$dax %>%  
  retorno()
```