Introducción a las finanzas quantitativas

Gabriel Cabrera G.

2018-09-21

Contents

List of Tables

List of Figures

Prefacio

Este es un apunte en proceso pensado en el curso de Finanzas I de Ingeniería Comercial de la Universidad de Chile, las aplicaciones en R son pensados con un enfoque pedagógico y cualquier comentario o sugerencias son bienvenidas.
La información de la sesión de R cuando se compila este apunte es la siguiente:

```
sessionInfo()
```

```
## R version 3.4.4 (2018-03-15)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.1 LTS
##
## Matrix products: default
## BLAS: /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1
##
## locale:
##  [1] LC_CTYPE=es_CL.UTF-8
##  [2] LC_NUMERIC=C
##  [3] LC_TIME=es_CL.UTF-8
##  [4] LC_COLLATE=es_CL.UTF-8
##  [5] LC_MONETARY=es_CL.UTF-8
##  [6] LC_MESSAGES=es_CL.UTF-8
##  [7] LC_PAPER=es_CL.UTF-8
##  [8] LC_NAME=C
##  [9] LC_ADDRESS=C
## [10] LC_TELEPHONE=C
## [11] LC_MEASUREMENT=es_CL.UTF-8
## [12] LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets
## [6] methods   base
##
## other attached packages:
##  [1] derivmkts_0.2.3   readxl_1.1.0
##  [3] IntroCompFinR_1.0 pacman_0.4.6
##  [5] reshape2_1.4.3    quantmod_0.4-13
##  [7] TTR_0.23-3        xts_0.11-0
##  [9] zoo_1.8-3         bindrcpp_0.2.2
## [11] forcats_0.3.0     stringr_1.3.1
## [13] dplyr_0.7.6       purrr_0.2.5
## [15] readr_1.1.1       tidyr_0.8.1
## [17] tibble_1.4.2      ggplot2_3.0.0
## [19] tidyverse_1.2.1   gapminder_0.3.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.18     lubridate_1.7.4  lattice_0.20-35
##  [4] assertthat_0.2.0 rprojroot_1.3-2  digest_0.6.16
##  [7] mime_0.5         R6_2.2.2         cellranger_1.1.0
## [10] plyr_1.8.4       backports_1.1.2  evaluate_0.11
## [13] httr_1.3.1       highr_0.7        pillar_1.3.0
## [16] rlang_0.2.1      curl_3.2         lazyeval_0.2.1
## [19] rstudioapi_0.7   DT_0.4           rmarkdown_1.10
```

```
## [22] labeling_0.3     htmlwidgets_1.2  munsell_0.5.0
## [25] shiny_1.1.0      broom_0.5.0      compiler_3.4.4
## [28] httpuv_1.4.5     modelr_0.1.2     xfun_0.3
## [31] pkgconfig_2.0.1  mnormt_1.5-5     htmltools_0.3.6
## [34] tidyselect_0.2.4 bookdown_0.7     crayon_1.3.4
## [37] withr_2.1.2      later_0.7.3      grid_3.4.4
## [40] nlme_3.1-137     jsonlite_1.5     xtable_1.8-2
## [43] gtable_0.2.0     magrittr_1.5     scales_1.0.0
## [46] cli_1.0.0        stringi_1.2.4    promises_1.0.1
## [49] xml2_1.2.0       tools_3.4.4      glue_1.3.0
## [52] hms_0.4.2        crosstalk_1.0.0  yaml_2.2.0
## [55] colorspace_1.3-2 rvest_0.3.2      knitr_1.20
## [58] bindr_0.1.1      haven_1.1.2
```

Sobre el Autor

Aún no hay mucho que decir, pero te invito a visitar mi página web donde la mayoría del tiempo estoy publicando post sobre programación y/o data science con aplicaciones a las finanzas y al mundo real.
Me agradan las citas...

> "It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts."
> Sir Arthur Conan Doyle

Chapter 1

Introducción

Antes de comenzar tú viaje a través del lenguaje de programación R, necesitaras cuatro "herramientas" básicas para trabajar con este apunte, R (lenguaje), Rstudio(IDE[1]), una megalibrería que contiene *R packages* llamada *tidyverse* y librerías extras para trabajar en finanzas[2].

1.1 R

1.1.1 Un poco de historia

R es un lenguaje de programación creado por Ross Ihaka y Robert Gentleman del departamento de estadística de la universidad de Auckland en 1992 (Nueva Zelanda), teniendo su versión estable el 29 de Febrero del 2000.

1.1.2 Los primeros pasos

El primer paso es descargar R, para esto debes ir a CRAN[3], *comprehensive R archive network*. CRAN esta compuesto por un conjunto de *mirror servers* distribuidos alrededor del mundo y se usa para compartir los *R packages*. Como recomendación no elegir un *mirror* lejano a tú posición geográfica, por ende, usa https://cloud.r-project.org que los seleccionará automáticamente.
Como aún no instalas RStudio solo tendras el lenguaje, que puede ser ejecutado desde el *command Shell* o *prompt*, no obstante, esto es ineficiente desde el punto de vista que no tendrenmos todas las opciones que Rstudio nos entrega.
Si trabajas sin IDE veras algo como en la Figura 1.2.
Las versiones de R cambian una vez al año, y de 2 a 3 veces con cambios pequeños, por eso es una buena idea que mantengas actualizada tu versión.

[1] *Integrated Development Environment*

[2] Estas librerías las iremos cargando/utilizando a medida que avancemos en los capítulos

[3] Existe otra distribución de R por parte del area de *open source* de Microsoft , la ventaja de MRAN es que si bien funciona con CRAN, su objetivo va más orientado a computación en paralelo (paralleling computing)

Figure 1.1: logo de R

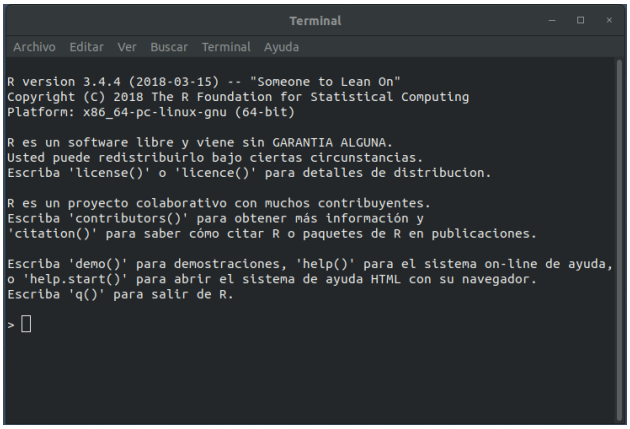


Figure 1.2: R desde la terminal

1.2 RStudio

¿Qué es una IDE? IDE es el acrónimo de *Integrated Development Environment (Entorno de Desarrollo Integrado)*. Esto quiere decir que RStudio es una aplicación que nos entrega herramientas para hacer más fácil el desarrollo de proyectos usando R y sobre todo cuando estemos trabajando con datos.
Para descargar e instalar Rstudio debes ir a http//www.rstudio.com/download y seleccionar *RStudio Desktop Open Source License* (gratuita) , cuando exista una actualización Rstudio te avisará.
Si quedó todo bien instalado, cuando abras Rstudio deberías ver algo así:

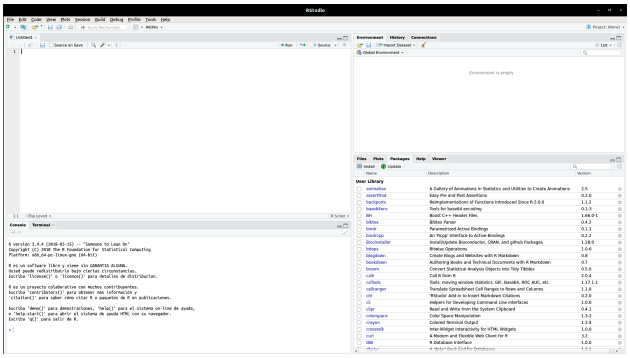


Figure 1.3: Rstudio

> IMPORTANTE: Si te aparece algún error durante este proceso, lo más probable es que sea por alguna configuración de tu sistema operativo. En ese caso, la mejor manera de buscar una solución es copiar el error que arroja R, pegarlo en tu motor de búsqueda favorito y ver cómo alguien que se enfrentó a eso antes lo resolvió.

1.2.1 Partes de RStudio

¿Para qué sirven estos paneles? Comentemos primero el panel de abajo a la derecha. Si te fijas, el panel tiene varias ventanas:

1. **Files** muestra el directorio (la carpeta) en la que te encuentras actualmente. En mi caso, no hay nada ahí porque por defecto RStudio me muestra el Escritorio (Desktop) y no tengo nada en él. Es posible que a ti te muestre otra carpeta (por ejemplo, Documentos).
2. **Plots** es el lugar donde aparecerán los gráficos que vayas creando. No hemos hecho ninguno por ahora, así que este panel también está vacío.
3. **Packages** muestra la lista de paquetes que tienes instalados en tu computador. Si recorres el panel verás que algunos tiene una marca al lado izquierdo. Eso quiere decir que el paquete está activo en ese momento (ya veremos cómo hacer eso). Solo los paquetes vinculados a R base se activan al abrir RStudio.
4. **Help**, como su nombre lo indica, es la pestaña en la que podemos encontrar ayuda. Si buscamos el nombre de un paquete o de una función, RStudio nos remitirá a la documentación asociada.

5. **Viewer** es el panel para ver contenido web generado por algún paquete de R (gráficos para la web o aplicaciones interactivas). Por el momento no lo utilizaremos.

El panel de arriba a la derecha, por su parte, contiene el historial de funciones que hemos ejecutado (History), la opción para generar conexiones a bases de datos externas (Connections) y el Environment. Este último panel es muy importante y entender lo que nos muestra es fundamental para comprender cómo funciona R.

1.3 *Scripts*

El script podemos decir que es un cuarto panel[4], en donde escribiras tus códigos que queremos que ejecute R, para crear un script debes:

1. ir a `file > New File > R Script`
2. Otra forma es un atajo de teclado, `control + shift + n` (Linux/Windows) y `comando + shift + n` (Mac OS).
3. O bien ir a la barra superior de la ventana y seleccionar el tipo de archivo a trabajar.

1.4 Proyectos

Una de las ventajas de RStudio es que permite crear "proyectos". Un projecto es un espacio o contexto de trabajo asociado a una carpeta en particular, en la que se guardan nuestro(s) script(s), archivos de datos, etc. Cuando creamos un proyecto en RStudio, se crea un tipo especial de archivo (.Rproj) que lo que hace es vincular todo lo que se encuentra dentro de esa carpeta.

1.4.1 ¿Por qué esto es útil?

Si parte de nuestro script, por ejemplo, implica abrir un archivo que está en la carpeta de nuestro proyecto, no necesito indicar en mi código toda la ruta del archivo: lo que hará RStudio será buscarlo en el entorno/carpeta del proyecto. Si movemos la carpeta a otro lugar de nuestro computador o la compartimos con otra persona, nuestro código seguirá funcionando, ya que el archivo .Rproj mantendrá todo unido. Si no creara un proyecto, tendría que indicar al inicio de mi script cuál es la ruta de la carpeta que ocuparé como espacio de trabajo. El problema de esa opción es que si muevo la carpeta o le cambio el nombre, tendría que volver a escribir la ruta para que todo funcione. Al crear un proyecto eso deja de ser una preocupación.

1.4.2 ¿Cómo crear un projecto?

1. Puedes hacerlo desde el menú File > New Proyect.
2. Lo primero que nos pregunta es si queremos crearlo en una carpeta nueva o en una ya existente. Elegiremos esta vez una carpeta nueva, así que seleccionaremos New Directory.

[4] El tercer panel es la consola

3. La siguiente pregunta es qué tipo de proyecto queremos crear. En esta ocasión, elegiremos la primera: *New Project*.
4. Finalmente, le damos un nombre al proyecto y decidimos en qué parte de nuestro computador queremos que viva la carpeta que lo contiene.
5. Luego de apretar Create Project, RStudio se reinicia y se producen algunos cambios. El panel Files (abajo a la derecha) ahora nos muestra la carpeta de nuestro proyecto y el único archivo que hay en ella por ahora. Ese es el archivo mágico que mantiene unido todo lo que hay dentro de la carpeta. Cuando queramos volver a trabajar en nuestro proyecto, solo tenemos que abrir ese archivo.
   IMPORTANTE: RStudio ejecuta sesiones independientes de R para cada proyecto. Es decir, si tuvieras otro proyecto abierto te aparecería otro ícono, con su respectivo nombre. Esto nos permite trabajar en dos proyectos en paralelo sin que se nos mezclen los objetos del entorno, el código, los archivos, etc. Cada cosa en su lugar.

1.5 *Packages*

Cuando instalamos R por primera vez en nuestro computador, lo que estamos instalando es lo que se conoce como "R Base", es decir, los elementos centrales del lenguaje de programación. Una de las ventajas de R es que se trata de un lenguaje extensible: la propia comunidad de usuarios puede desarrollar nuevas posibilidades para utilizarlo. La manera de compartir estos nuevos desarrollos es a través de "paquetes", que incluyen, entre otras cosas, código y datos. Una analogía que se suele utilizar para explicar esto es que R Base es un teléfono celular tal como viene de fábrica y los paquetes las apps que descargamos para que tenga más funcionalidades.
Para usar las librerias ("packages") debemos usar el siguiente código:

```
# instala el package
install.packages("acá va el package")

# lo llama
library("acá va el package")
```

1.5.1 *tidyverse*

tidyverse es un "megapaquete" que incluye otros paquetes en su interior. Todos los paquetes que conforman "el Tidyverse" comparten la misma visión sobre el trabajo con datos y la escritura de código. Viene a formar parte de la nueva forma de programar en R, cuyo enfoque es netamente en realizar Data Science. Algunas librerias relevantes son:

1. ggplot2: Esta librería te permite realizar graficos avanzados.
2. dplyr: Su objetivo es la manipulación de datos (filtrar, seleccionar, generar, renombrar,etc).
3. magrittr: Contiene la denominada *pipe* (%>%), se explicará más adelante
4. purrr: Para realizar iteraciones.
5. readr: Para cargar datos en csv, lo importante que los transforma en tibble.

Para instalarlo basta escribir:

```
# instala el package
install.packages("tidyverse")

# lo llama
library("tidyverse")
```

Si te vas a la pestaña de *Packages* verás que estan seleccionadas aquellas librerias que se encuentran en el tidyverse.

1.6 Ejemplo

En este ejemplo vamos a trabajar con `gapminder`, un paquete que contiene una parte de los datos de Gapminder, una base de datos que incluye información mundial sobre población, expectativa de vida, PIB per cápita y otros. Su autor, Hans Rosling, ha hecho varias charlas TED que vale la pena mirar.
Instalamos la librería

```
# instala el package
install.packages("gapminder")
```

Cargamos tanto `gapminder` como `tidyverse`

```
# lo llama
library("gapminder")
library("tidyverse")
```

Calculamos el promedio de la expectativa de vida para los continentes en el 2007:

```
world.data <- gapminder

mean.lifeExp <- world.data %>%
                filter(year == 2007) %>%
                group_by(continent) %>%
                summarize(mean(lifeExp))
```

```
mean.lifeExp
```

Show 10 entries Search:

	continent	mean(lifeExp)
1	Africa	54.806038461538
2	Americas	73.6081
3	Asia	70.728484848484
4	Europe	77.648
5	Oceania	80.719

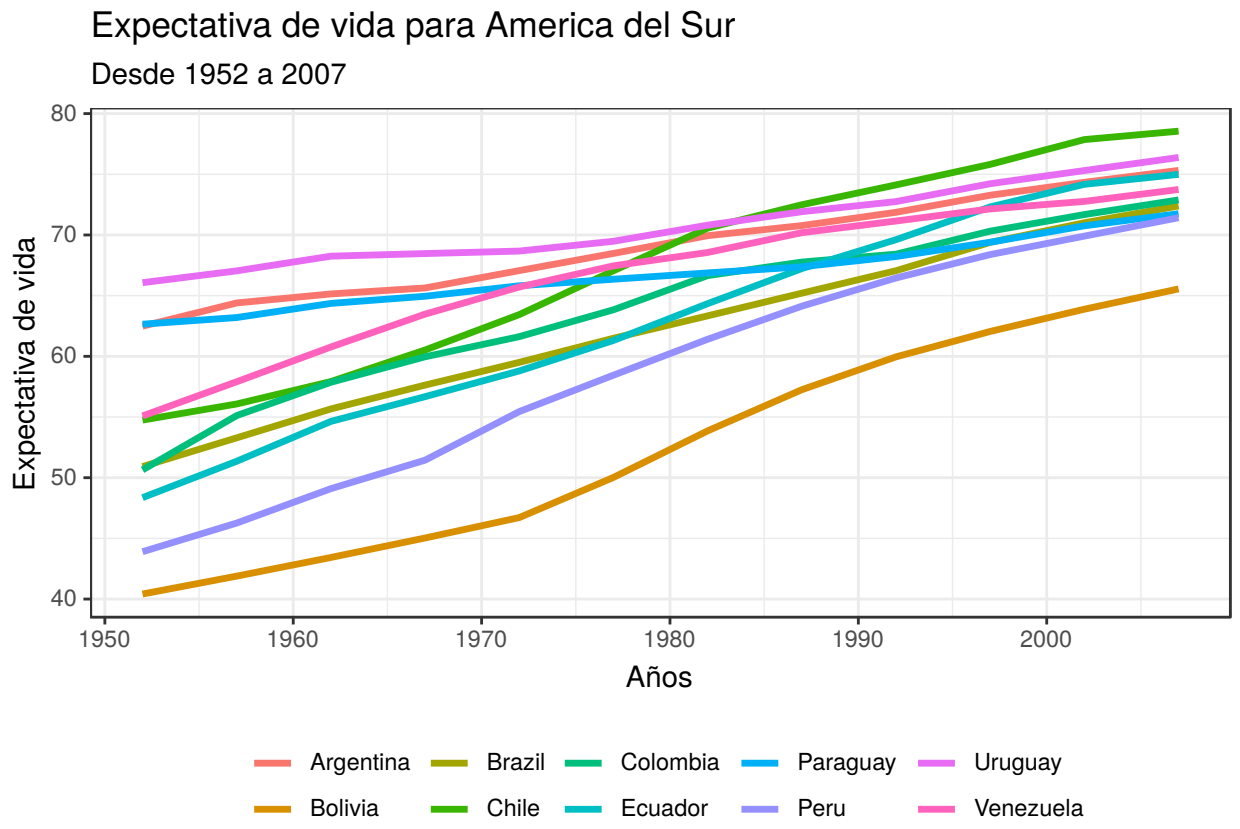
Showing 1 to 5 of 5 entries Previous 1 Nex

Lo visualizamos

Construimos la base de datos y graficamos la evolución de la expectativa de vida para los países de America del Sur desde 1952 a 2007.

```
world.data <- world.data %>%
              filter(country %in% c("Argentina", "Bolivia", "Brazil", "Chile", "Colombia", "Ecuador", "Paraguay", "Peru", "Urugu

g1 <- ggplot(world.data) + geom_line(mapping = aes(year,lifeExp, colour= country), size = 1.2)
g1 <- g1 + theme_bw() + labs(title = "Expectativa de vida para America del Sur", subtitle = "Desde 1952 a 2007", colour = "")
g1 <- g1 + xlab("Años") + ylab("Expectativa de vida")
g1 <- g1 + theme(legend.position="bottom")
g1
```



Finalmente lo guardamos.

```
ggsave("Grafico.png")
```

Chapter 2

Quantmod

IMPORTATE: Aún no está del todo listo el formato en pdf, por lo que recomiendo verlo online.

El paquete quantmod para R esta diseñado para la asistencia quantitativa de los traders en el desarrollo de sus estrategias y modelos financieros.

2.1 ¿Que es quantmod?

Un entorno rápido, donde los operadores cuantitativos pueden explorar y construir modelos de negociación rápida y limpiamente. A través de la función getSymbols podemos extraer datos financieros desde varias fuentes: Google Finance, Yahoo Finance, Federal Reserve Bank of St. Louis FRED (más de 11,000 series !!!) y Oanda. Incluso desde fuentes propias: MySQL , R (Rdata) y Comma Separated Value files (csv).
No es el paquete definitivo dado que se complementa con otros, tales como: TTR, zoo y xts. En lo que respecta al análisis técnico son las más usadas en el mercado y usan todas las propiedades que hacen al lenguaje R útil para realizar análisis de datos[1].

2.2 Obtención de Datos

Para comenzar, como todo paquete en R se debe instalar

```
# Instalación package
install.packages("quantmod")
```

Una vez que esté instalado, creamos nuestro script usando `ctrl/cmd + shift + n` y lo "llamamos" con

```
# Cargamos "quantmod"
library("quantmod")
```

HINT: con `ctrl + R` en windows/Linux y `cmd + R` en MAC OS agregamos más rapido comentarios (sección) en Rstudio.

quantmod provee una función para descargar datos desde fuentes externas. Esta función se

[1] Proximamente incluire el tidyquant

llama `getSymbols`, para mayor información escribir en la linea de comandos `?getSymbols`[2]. Por defecto, se crea un objeto en el workspace (*Global Environment*) con el nombre del ticker/nemotécnico seleccionado. Imaginemos por un momento que necesitamos analizar el S&P 500 desde el 2010 hasta la fecha con periocidad diaria. Lo primero que debemos hacer es pensar desde que fuente vamos a descargar los datos, como es un índice accionario se recomienda usar *yahoo finance*, luego buscar el nemotécnico, en este caso es "^GSPC".

```
getSymbols("^GSPC", src = "yahoo", from = "2010-01-01", to = "2010-07-30", periodicity = "daily")
```

```
## [1] "GSPC"
```

2.2.1 ¿Qué hizo la función `getSymbols`?

La función `getSymbols` se construye basicamente de cinco opciones[3]:

1. El ticker/nemotécnico, eg. ^GSPC.
2. `src`, que es la abreviación de "source", eg. yahoo, FRED...
3. `from`, es el inicio de la fecha a descargar, tener presente que se incluye la fecha en nuestros datos.
4. `to`, es el final del periodo para los datos, este no se incluye.
5. `periodicity`, es la periodicidad de los datos, eg. daily, monthly o yearly, solo algunos datos se ajustan a las tres periodicidades.

En el ejemplo anterior se descargo desde yahoo los datos del S&P 500 desde Enero del 2010 hasta el viernes 27 de Julio del 2018 con periodicidad diaria, construyendo un objeto en formato **xts** cuyo nombre es GSPC.

Show 10 entries Search:

GSPC.Open	GSPC.High	GSPC.Low	GSPC.Close	GSPC.Volume	GSPC.Adjusted
1116.560059	1133.869995	1116.560059	1132.98999	3991400000	1132.98999
1132.660034	1136.630005	1129.660034	1136.52002	2491020000	1136.52002
1135.709961	1139.189941	1133.949951	1137.140015	4972660000	1137.140015
1136.27002	1142.459961	1131.319946	1141.689941	5270680000	1141.689941
1140.52002	1145.390015	1136.219971	1144.97998	4389590000	1144.97998
1145.959961	1149.73999	1142.02002	1146.97998	4255780000	1146.97998
1143.810059	1143.810059	1131.77002	1136.219971	4716160000	1136.219971
1137.310059	1148.400024	1133.180054	1145.680054	4170360000	1145.680054
1145.680054	1150.410034	1143.800049	1148.459961	3915200000	1148.459961
1147.719971	1147.77002	1131.390015	1136.030029	4758730000	1136.030029

Showing 1 to 10 of 144 entries Previous 1 2 3 4 5 ... 15 Next

[2]No solo funcióna con `getSymbols`, si no que con todas las funciones de distintas librerias, basta con ante poner `?` y luego el nombre la función

[3]Por el momento solo trabajaremos con estas opciones, exiten más.

2.3 Graficando con chartSeries

Aún no introducimos la librería ggplot2, sin embargo, quantmod también nos permite graficar.

2.3.1 chartSeries

Para graficar basta con escribir el nombre del objeto con clase (class) xts, en nuestro caso es GSPC que representa al Standard and Poor 500. Si escribimos TA = NULL, charSeries no mostrará el volume[4]

```
chartSeries(GSPC, TA=NULL)
```

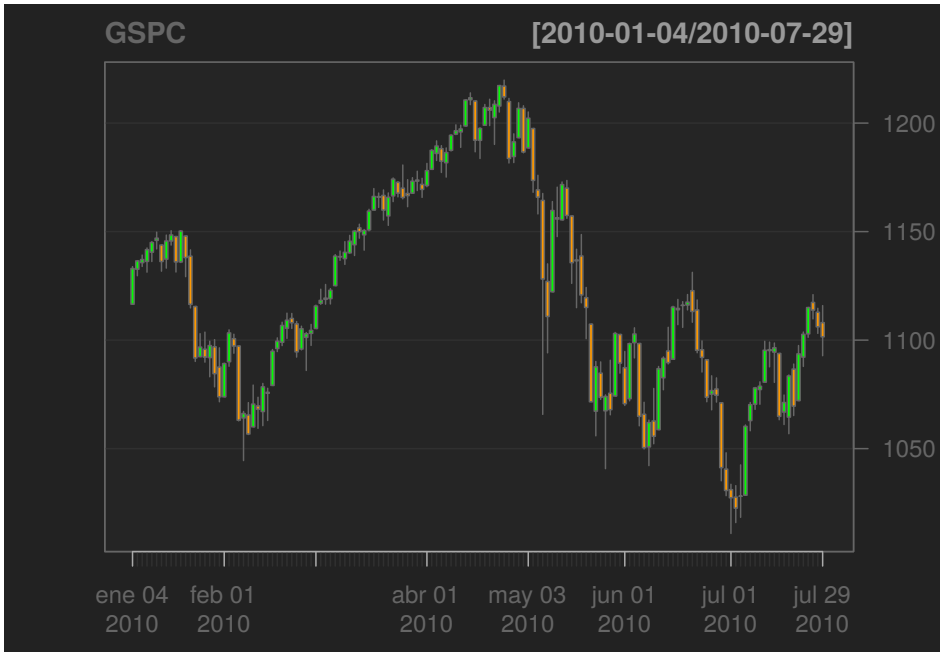


Figure 2.1: Gráfico con chartSeries con TA = NULL

```
chartSeries(GSPC, TA=NULL)
```

Pero cuando las series son muy largas, podemos ver tendencias pero dificulta ver cambios importantes a nivel de análisis técnico.

```
chartSeries(GSPC, subset = "last 3 months")
```

Con el código anterior nos enfocamos solo en los tres meses anteriores.

2.4 Graficando con ggplot2

Si bien chartSeries es una alternativa a plot o plotly, este no nos permite realizar gráficos que se adapten a nuestras necesidades. ¿Qué pasa si queremos graficar retornos o retornos

[4] TA proviene de *Technical Analysis*

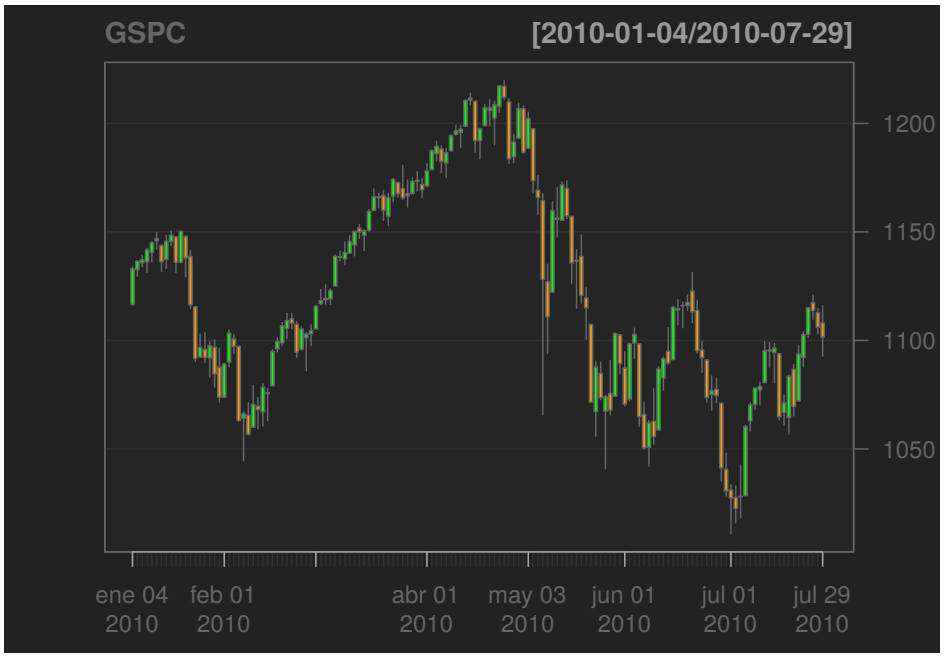


Figure 2.2: Gráfico con chartSeries sin TA = NULL

Figure 2.3: Los últimos 3 meses de GSPC

acumulados? la opción por exelencia es `ggplot2`.

2.4.1 Breve introducción a `ggplot2`

Todo ggplot2 plot tiene tres componentes:

1. Datos
2. Un conjunto de aesthetic mappings entre variables y propiedades de visualización.
3. Al menos una layer que describe la observación, son usualmente creadas con la función geom_*

```
library(ggplot2)
```

A continuación usaremos la base que viene pre cargada con R cuando lo instalamos, que es `cars`[5]

```
ggplot(mpg, aes(x = displ, y = hwy)) + geom_point()
```

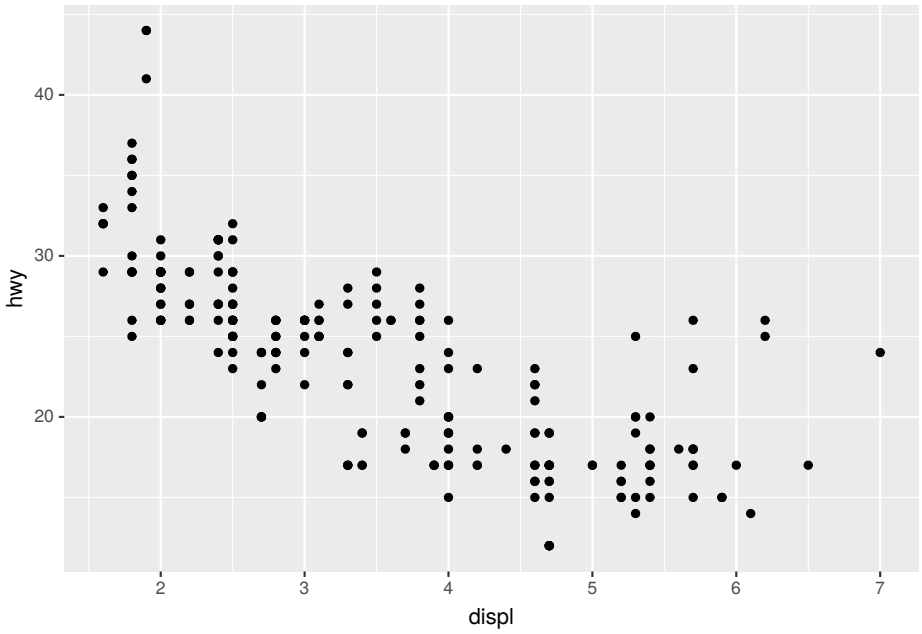


Figure 2.4: Ejemplo 1 usando ggplot2

Esto produce el scatterplot definido como:

1. Datos: mpg
2. Aesthetic mapping: tamaño del motor en la posición x, gasolina en la posición y.
3. Layer: puntos.

2.4.2 Color, tamaño, forma y otros atributos del *aesthetic*

Se debe usar otro aesthetics como `colour`, `shape` y `size` (ggplot acepta los nombres americanos como británicos)

[5]Esta base es muy común en los software estadísticos y presenta datos de autos.

```
ggplot(mpg, aes(displ, cty, colour = class)) +
  geom_point()
```

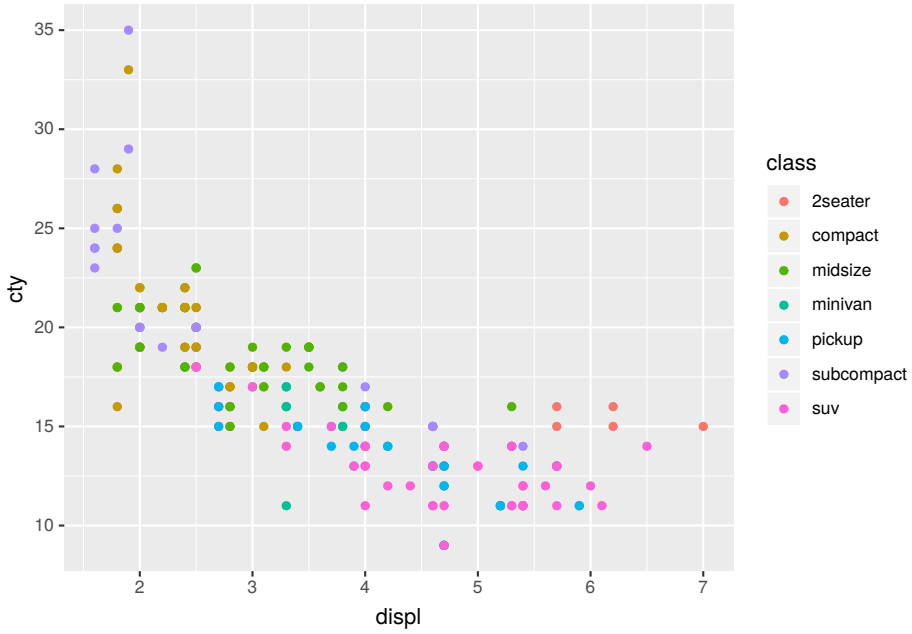


Figure 2.5: Ejemplo 2 usando ggplot2

- Como se puede ver, se creo una guía con los valores, leyenda, así podemos "leer" el gráfico.
- Si se quiere aesthetic para valores fijos, sin scaling:

```
ggplot(mpg, aes(displ, hwy)) + geom_point(aes(colour = "blue"))
```

```
ggplot(mpg, aes(displ, hwy)) + geom_point(colour = "blue")
```

Ejercicios:

```
aes(displ, hwy, colour = class)
aes(displ, hwy, shape = drv)
aes(displ, hwy, size = cyl)
```

- Se recomienda usar `colour` y `shape` con variables categóricas.
- Mientras que size funciona bien con variables continuas.

2.4.3 S&P 500 con `ggplot2`

```
gspc <- as.data.frame(GSPC)
```

```
g1 <- ggplot(gspc) + geom_line(mapping = aes(index(gspc),GSPC.Adjusted))
g1 <- g1 + labs(title = "S&P 500", subtitle = "Desde Enero 2010 a 2018") + xlab("Fecha") + ylab("")
g1 <- g1 + theme_bw()
g1
```

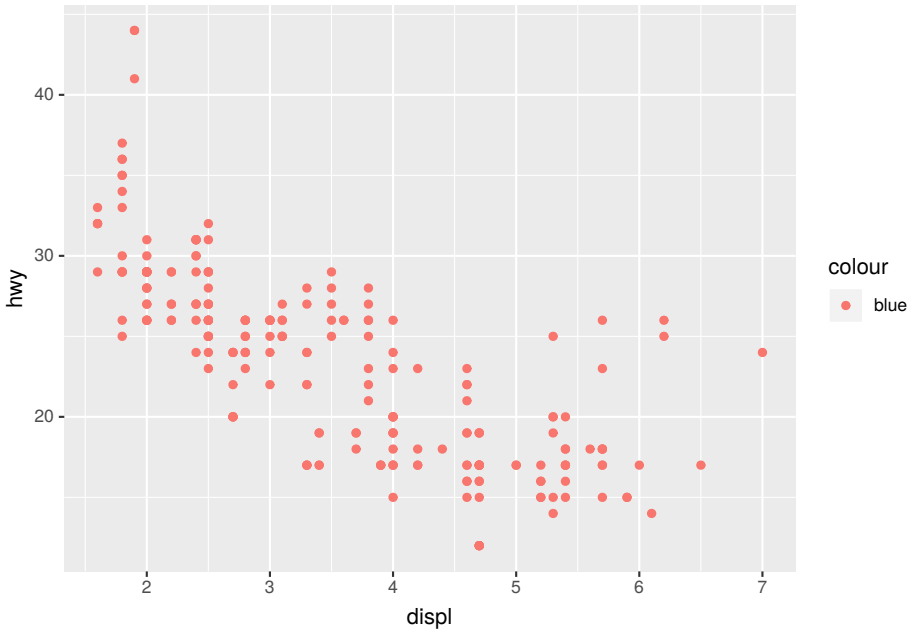


Figure 2.6: Ejemplo 3 usando ggplot2

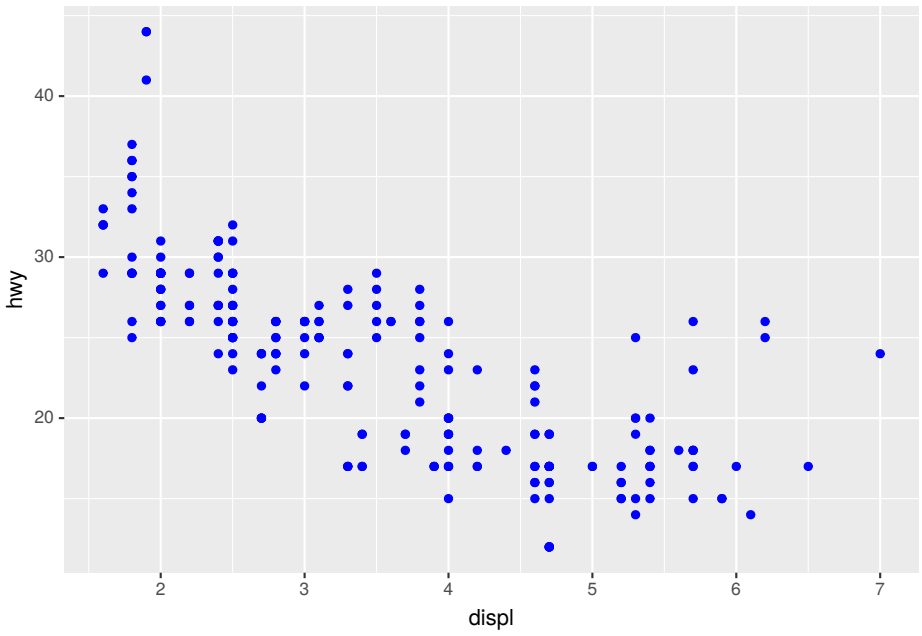


Figure 2.7: Ejemplo 3 usando ggplot2

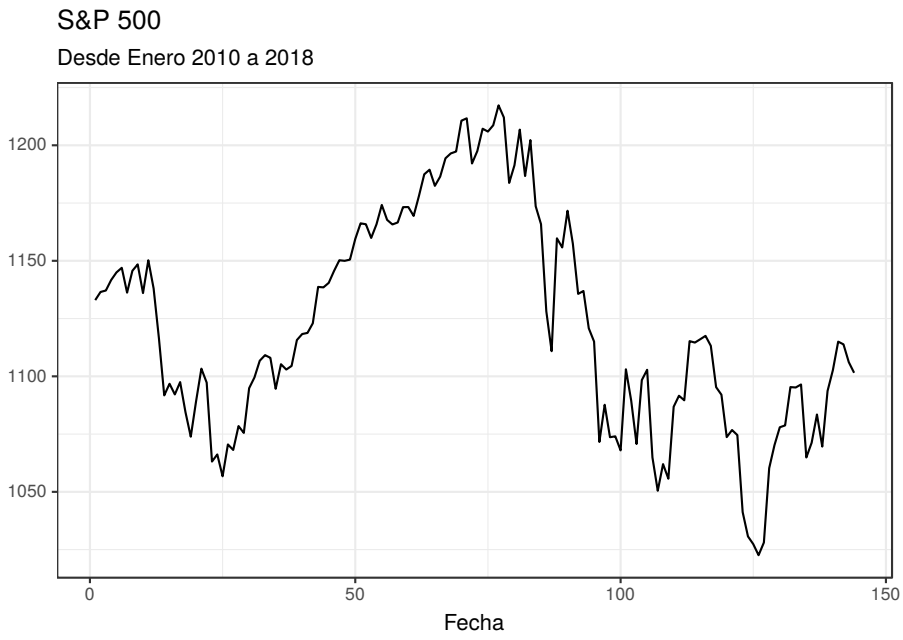


Figure 2.8: Standard and Poor 500 usando ggplot2

2.5 Multiples Datos

A continuación trabajaremos con las acciones de Oracle, Nvidia, IBM y AMD, comenzamos con crear un objeto con los nombres de los tickers

```
tickers <- c("ORCL","AMD","IBM","NVDA")
```

descargamos los datos con las características requeridas, que son las mismas que usamos anteriormente con S&P 500

```
getSymbols(tickers, src = "yahoo", from = "2010-01-01", to = "2018-07-30", periodicity = "daily")
```

```
## [1] "ORCL" "AMD"  "IBM"  "NVDA"
```

Acá deben tener mucha atención:

```
list <- lapply(tickers, function(x) Cl(get(x)))
precio.cierre <- do.call(merge,list)
```

2.5.1 Retornos

La formula para calcular (log) retornos es

$$r_t = log(1 + R_t) = log(\frac{P_t}{P_{t-1}}) = p_t - p_{t-1}$$

donde $p_t = log(P_t)$ es llamado "log price".
Ahora pasamos a construir el retorno.

```
retornos <- data.frame(apply(precio.cierre, 2, function(x) Delt(x, type = "log")),
                       fecha = index(precio.cierre)) %>%
            rename(orcl = ORCL.Close, amd = AMD.Close, ibm = IBM.Close, nvda = NVDA.Close) %>%
            na.omit()
```

2.5.2 Retornos Acumulados

Si graficamos los retornos no será muy descriptivo, una forma es trabajar con su acumulado. Con la misma lógica usamos la función `cumsum()`.

```
acumulados <- data.frame(apply(retornos[1:4], 2, function(x) cumsum(x)), fecha = index(precio.cierre[-1]))
```

2.5.2.1 Gráfico retornos acumulados

```
library("reshape2")
```

```
reshape <- melt(acumulados, id.vars = "fecha")
```

```
g2 <- ggplot(reshape) + geom_line(mapping = aes(fecha,value, color = variable))
g2 <- g2 + labs(title = "Retornos Acumulados", subtitle = "Oracle, AMD, IBM y Nvidia")
g2 <- g2 + theme_bw() + xlab("Fecha") + ylab("Retornos Acumulados")
g2 <- g2 + scale_color_manual("Tickers", values = c("red","blue","green","orange"))
g2 <- g2 + theme(legend.position = "bottom")
g2
```

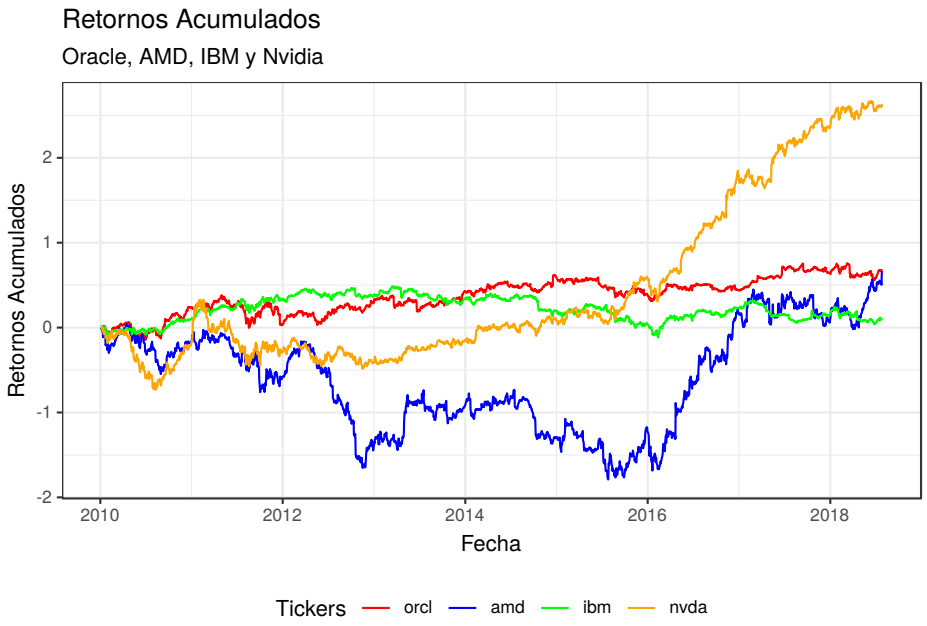


Figure 2.9: Retornos Acumulados de los tickers

2.6 Estadística Descriptiva

Existe muchas formas de obtener la estadística descriptiva en R, un librería es `fBasics`, la que a su vez contiene test de normalidad.

```
library("fBasics")
```

```
summary <- basicStats(retornos[1:4])[c("Mean", "Stdev", "Median", "Minimum", "Variance",
                                       "Maximum", "nobs","Skewness","Kurtosis"),]
```

2.7 Ratio de Sharpe

EL ratio de Sharpe es una medida de desempeño para portafolios, el que se define como

$$SR = \frac{E(R_i) - r_f}{\sigma_i}$$

Donde $E(R_i) = \mu_i$ es el retorno del portafolio i, r_f la tasa libre de riesgo y σ_i la desviaciòn estandar del portafolio i. Si asumimos como ejercicio que no diversificamos y "construimos" cuatro portafolios con el 100% es invertido en cada uno de los activos.
Para realizar el cálculo del ratio de Sharpe (SR) para Oracle

```
SR_orcl <- (mean(retornos$orcl) - 0.0000 )/sd(retornos$orcl)
```

con `mean(retornos$orcl)` vamos a obtener el promedio de la primera columna del objeto `retornos` que es Oracle (en el `data.frame` la columna se llama orcl, si tuviese otro nombre como por ejemplo `perrito`, entonces hubiese sido `mean(retornos$perrito)`), si quisieramos AMD debería ser `amd` y asì sucesivamente. El 0.0000 sería la tasa libre de riesgo que la asumimos mensual y `sd(retornos[1])` nos calcula la desviación estandar. Dado que poseemos 4 activos con sus respectivos retornos, deberiamos construir cuatro objetos que partan con `SR`.

2.8 Test de JarqueBera

El test de jarque-bera usa los coeficientes de la *skewness* y *kurtosis* de la muestra y se usa para testar normalidad. Otros test comunes son el de Anderson–Darling, Cramér–von Mises, y Kolmogorov–Smirnov.
En resumen compara que la *skewness* sea 0 y la *kurtosis* sea 3 bajo normalidad.

$$JB = n\widehat{Sk}^2/6 + (\widehat{KUr} - 3)^2/24$$

El test se encuentra en varias librerías, una de ellas es `fBasics` que deberíamos tener instalada y cargada. Para obtener los resultados del test para Oracle

```
jarqueberaTest(retornos$orcl)
```

Para los demàs activos hay que solo cambiar por el nombre de la columna correspondiente.

2.9 Recursos del capítulo

Chapter 3

Introducción a teoría de portafolio

IMPORTANTE: Aún no está del todo listo el formato en pdf, por lo que recomiendo verlo online.

3.1 Librería IntroCompFinR

Para la teoría de portfolio vamos a utilzar la librería IntroCompFinR (**Intro** to **Comp**utational **Fin**ance in **R**) creado por el profesor Eric Zivot.

1. Debemos instalar primero las librerías que utiliza IntroCompFinR:

```
if(!require("pacman")) install.packages("pacman")
p_load("PerformanceAnalytics","quadprog","xts")
```

2. Ya instaladas las dependencias, descargamos IntroCompFinR :

```
install.packages("IntroCompFinR", repos="http://R-Forge.R-project.org")
```

3.1.1 Funciones útiles de IntroCompFinR

Funciones	Descripciones
getPortfolio	Crea un portafolio (objeto)
globalMin.portfolio	Computa el portafolio de mímina varianza
efficient.portfolio	Computa el portafolio de mímina varianza sujeto a un retorno
tangency.portfolio	Computa el portafolio tangente
efficient.frontier	Computa la frontera eficiente

3.2 Cargando la librería y la base de datos

Una vez la instalada la librería, procedemos a cargarla en conjunto con aquellas que utilizaremos en esta ayudantía:

```
if(!require("pacman")) install.packages("pacman")
p_load("IntroCompFinR","readxl","tidyverse")
```

Como ya está cargado `readxl` cargamos el archivo **stocks.xlsx**, que ya posee los retornos[1].

```
# acá están los retornos ya calculados, para replicarlos vean el apunte
stocks <- read_xlsx("datasets/stocks.xlsx")
```

Considerando tres activos riesgosos (Starbucks, Nordstrom y Microsoft), definimos un vector columna $3x1$ el que tendrá los retornos y los pesos:

$$\mathbf{R} = \begin{pmatrix} R_a \\ R_b \\ R_c \end{pmatrix}, \mathbf{x} = \begin{pmatrix} x_a \\ x_b \\ x_c \end{pmatrix}$$

El vector de retornos esperados es:

$$E[\mathbf{R}] = E\left[\begin{pmatrix} R_a \\ R_b \\ R_c \end{pmatrix}\right] = \begin{pmatrix} E[R_a] \\ E[R_b] \\ E[R_c] \end{pmatrix} = \begin{pmatrix} \mu_a \\ \mu_b \\ \mu_c \end{pmatrix} = \mu$$

La matriz $3x3$ de varianza y covarianza de los retornos es:

$$var[\mathbf{R}] = \begin{pmatrix} \sigma_a^2 & \sigma_{ab} & \sigma_{ac} \\ \sigma_{ab} & \sigma_b^2 & \sigma_{bc} \\ \sigma_{ac} & \sigma_{bc} & \sigma_c^2 \end{pmatrix} = \Sigma$$

Notar que la matriz de covarianza es simétrica ($\Sigma = \Sigma'$).
Para construir las matrices anteriores en R:

```
# Promedio
mean <- apply(stocks[2:4], 2 , function(x) mean(x))
# Desviación Estandar
sd   <- apply(stocks[2:4], 2 , function(x) sd(x))
# Covarianza
cov  <- cov(stocks[2:4])
```

[1] si quieren replicarlo vean los videos

Chapter 4

Renta Fija

IMPORTANTE: Aún no está del todo listo el formato en pdf, por lo que recomiendo verlo online.

```
if(!require("pacman")) install.packages("pacman")
p_load("tidyverse", "quantmod")
```

4.1 Precio de un Bono

El precio de un bono se calcula como:

$$P_B = \sum_{t=1}^{T} \frac{C}{(1+r)^t} + \frac{\text{ValorNominal}}{(1+r)^t}$$

Donde:

- P_B: Precio del Bono
- C_t: Pago interéses o cupones
- T : Números de períodos o madurez
- r : Tasa de descuento o yield-to-maturity semi-anual

Consideremos el siguiente ejercicio:
Calcular el precio de un bono con pago de cupón semestral, Madurez 25 años, Tasa cupón 6.5%, Yield semi-anual de 6.9% y Valor nominal de 100.
Para desarrollar el ejercicio, debemos construir por "parte"" los componentes de nuestro bono.

1. Construimos tanto la tasa cupón como la *Yield semi-anual*.

```
tc <- 0.065
y  <- 0.069
```

2. Construimos un vector con los valores de los cupones más el principal

```
pago <- c(rep(tc*100/2,49),(100 + tc*100/2))
```

tanto `tc`, `r` y `pago` se encontraran en *Values* del *global environment*, en tipo `numeric`. `c` es una función generica para crear vectores, `rep` hace una repetición de `tc*100/2` 49 veces y el numero 50, es igual al principal más `tc*100/2`. Recordar que como es semi-anual se divide por 2 el valor nominal de 100.

3. Para poder trabajar con nuestra base de datos, transformamos nuestro vector `pago` que está en forma `numeric` a `data frame`.

```
pago <- as.data.frame(pago)
```

Ahora existe un objeto con una estructura de datos `data frame` en nuestro global environment. Nuestro nuevo objeto `pago` se podría haber llamado de cualquier forma.

4.2 Dos formas de hacer lo mismo

Ya construido nuestro objeto `pago` veremos que en **R** existen muchas formas de hacer lo mismo:

1. Al principio del capítulo cargamos la libreria `tidyverse`, está nos permitirá trabajar con un "megapaquete" que incluye otros paquetes en su interior (ggplot2, dplyr, magittr, entre otros). Todos los paquetes que conforman "el Tidyverse" comparten la misma visión sobre el trabajo con datos y la escritura de código. Si vamos a la pestaña packages y escribimos `dplyr` veremos que está activo, pero nunca lo "llamamos", esto se debe a `tidyverse` lo hizo por nosotros.

```
pago1 <- pago %>%
         mutate(t1 = as.numeric(index(pago)),factor_desc = 1/(1+y/2)^(t1),
                val_present = pago*factor_desc) %>%
         summarise(sum(val_present)) %>%
         rename(`Precio Bono` = `sum(val_present)`)

pago1
```

```
##   Precio Bono
## 1       95.27
```

El precio del bono es 95.2663.

2. La otra forma es:

```
# replicamos el objeto
pago2 <- pago

pago2$t2 <- as.numeric(rownames(pago2))

# Calculamos el factor de descuento
pago2$factor_desc <- 1 / (1 + y/2)^(pago2$t2)
# Calculamos el valor presente
pago2$val_present <-  pago2$factor_desc*pago2$pago
# Calculamos el precio
sum(pago2$val_present)
```

```
## [1] 95.27
```

Como es de esperarse obtenemos el mismo precio del bono, 95.2663.

4.3 Funciones

Dominar por completo las funciones en R lleva practica y dedicación, no obstante, la dificultad que utilizaremos en este capítulo es baja y es un buen ejemplo practico para comenzar. Toda función en R, tiene tres partes.

1. El `body()`, el código dentro de la función.
2. El `formals()`, la lista de argumentos que controlan como puedes llamar la función.
3. El `environment()`, el "mapa" de la locación de las variables de la función.

A continuación vamos a crear un función que permita calcular el cuadrado de cualquier número:

```
f <- function(x){
    x^2
}
```

Para obtener el cuadrado de 2 y de 4:

```
f(2)
```

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

```
f(4)
```

```
## [1] 16
```

En el ejemplo anterior el `formals()` de `f`:

```
formals(f)
```

```
## $x
```

el `body()` de `f`:

```
body(f)
```

```
## {
##     x^2
## }
```

y el `environment()` de `f`:

```
environment(f)
```

```
## <environment: R_GlobalEnv>
```

Ya visto una breve introducción a funciones en R, procedemos a construir una función que nos permitirá valorizar cualquier bono que **pague cupones iguales**:

```
# p: valor nominal; tc: tasa cupón; t: madurez; y: yield to maturity
precio.bono <- function(p,tc,t,y){
  pago   <- c(rep(tc*p, t - 1),p*(1 + tc))
  pago   <- as.data.frame(pago)
  pago$t <- as.numeric(rownames(pago))
  pago$factor_desc <- 1 / (1 + y)^(pago$t)
  pago$valor_prese <- pago$factor_desc*pago$pago
  sum(pago$valor_prese)
}
```

```
precio.bono(100,0.065/2,50,0.069/2)
```

```
## [1] 95.27
```

Usando el mismo ejemplo de la sección anterior, obtenermos un precio igual a 95.2663.

4.4 Relación precio del Bono y Yield

4.4.1 Valorización

Ahora utilizando la función `precio.bono` valorizaremos un bono con las siguientes características:

- Principal : 100
- Tasa Cupón: 5%
- Madurez: 10 años
- Yield: 4.29%

```
# Valoramos el siguiente Bono
precio.bono(p = 100, tc = 0.05, t = 10, y = 0.0429)
```

```
## [1] 105.7
```

4.4.2 Construcción yields

Se contruirá una secuencia de yields:

```
# Cosntruimos yields
yields <- seq(0.02, 0.4, 0.01)
```

La función `seq` generá una secuencia. En este caso parte del 0.02 hasta el 0.4 pero con intervalos de 0.01.

```
# Convertimos yields a data frame como antes
yields <- as.data.frame(yields)
```

4.4.3 Loops

```
# Calaculamos el precio del bono para distintas yields
for (i in 1:nrow(yields)) {
  yields$precio[i] <- precio.bono(100, 0.10, 20, yields$yields[i])
}
```

4.4.4 Graficando

Una manera de visualizar datos es usar ggplot2, se recomienda que añadan por parte lo que desean en su gráfico.

```
# Graficamos con ggplot2
g1 <- ggplot(data = yields,aes(x = yields*100, y = precio)) + geom_line(size = 1.5, color = "red")
g1 <- g1 + geom_point(size = 3, color = "red")
g1 <- g1 + ggtitle("Relación inversa:", subtitle = "Precio del Bono vs Yield")
g1 <- g1 + xlab("Yield (%)") + ylab("Precio del bono")
g1 <- g1 + geom_ribbon(aes(ymin = 0, ymax = pmax(precio,0)), fill="pink", col="red", alpha=0.5)
g1 <- g1 + theme_bw()
g1 <- g1 + theme(panel.border = element_rect(colour = "black", fill = NA, size = .5),
                 panel.grid.major = element_line(colour = "#d3d3d3"))
g1
```

```
# Guarmados gráfico
ggsave("retorno-yield.png",width = 8.5, height = 4.5, dpi = 300)
```

4.5 Trabajando con yields reales

quantmod es uno de las librerías más ocupadas en R para extraer datos financieros, te permite graficar, realizar análisis técnico, calcular retornos (`Delt(x)`), etc. Aunque las series son descargadas con estructura xts, la podemos transformar a data frame. A continuación descargaremos la yield de los bonos del tesoro de Estados Unidos a 10 años:

```
t10yr <- getSymbols(Symbols = "DGS10", src = "FRED", auto.assign = FALSE)

t10yr <- subset(t10yr["2000-01-01/2018-04-17"])
```

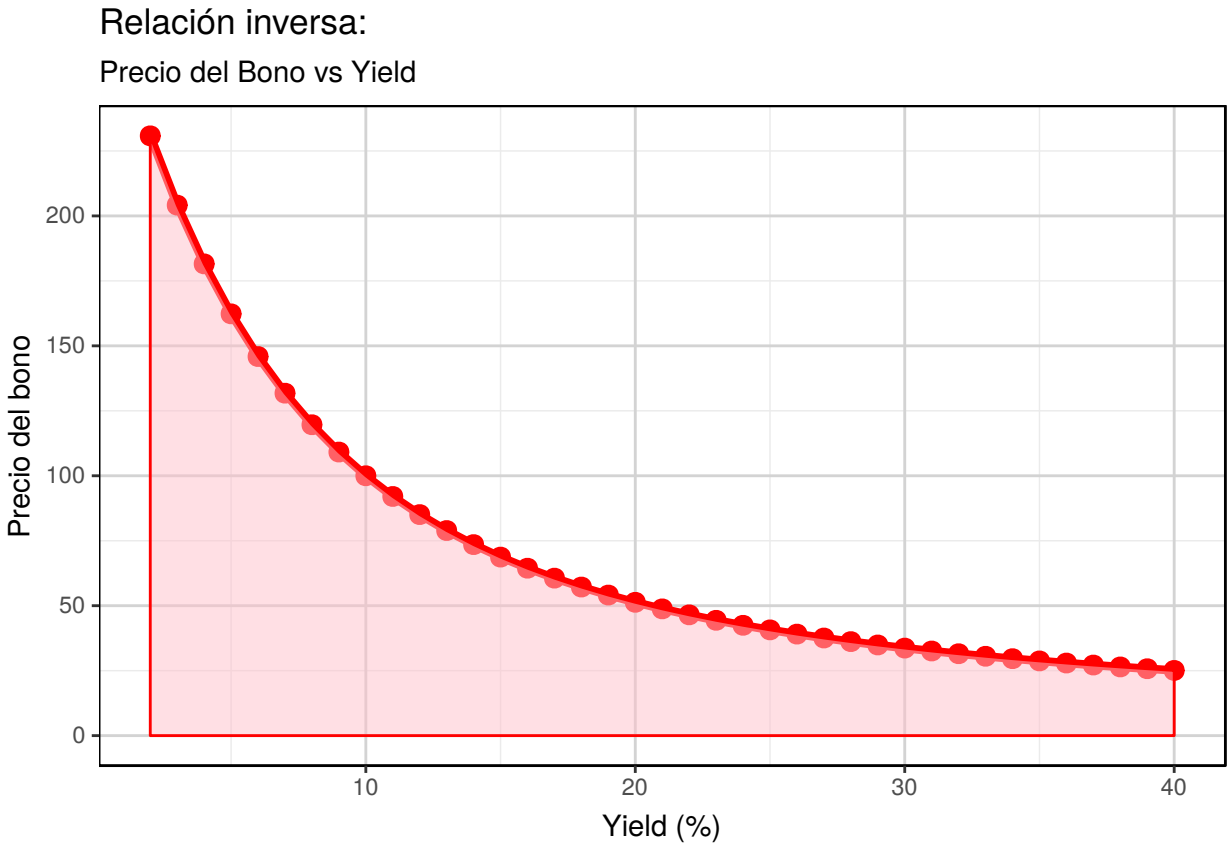


Figure 4.1: Relación Bono vs Yield

Con la función `subset` extramos una parte de los datos, especificamente desde 2000-01-01 hasta 2018-04-17. Luego graficamos usando la función `chartSeries` de `quantmod`. Tener cuidado con la función, dado que solo funciona con extensión xts.

```
# Grafico con chartSeries de quantmod solo funciona con xts
chartSeries(t10yr,  theme = "white")
```

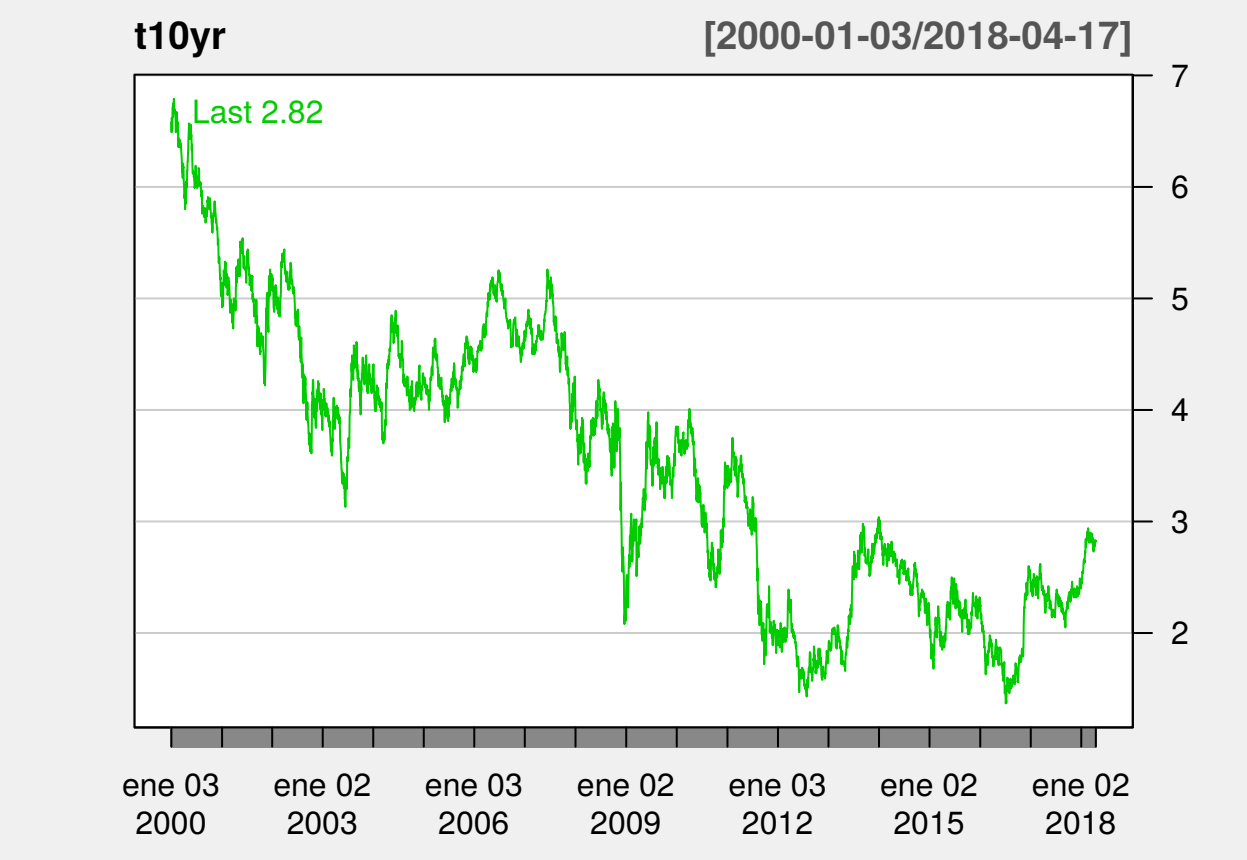


Figure 4.2: yield del tesoro de los Estados Unidos con chartSeries

En caso que se quiera graficar usando `ggplot2`:

```
t10yr.df <- as.data.frame(t10yr)

t10yr.df <- t10yr.df %>%
            mutate(fecha = as.Date(rownames(t10yr.df))) %>%
            na.omit()

g3 <- ggplot(data = t10yr.df,aes(x = fecha , y = DGS10)) + geom_line(size = 1, color = "green")
g3 <- g3 + ggtitle("10-Year US Treasury Yields", subtitle = "Desde 2000-01-01 hasta 2018-04-17")
g3 <- g3 + ylab("Fecha") +xlab("Yield(%)")
g3 <- g3 + theme_bw() + theme(panel.border = element_rect(colour = "black", fill = NA, size = .5),
                              panel.grid.major = element_line(colour = "#d3d3d3"))
g3
```

```
# Guarmados gráfico
ggsave("treasury-yields.png",width = 8.5, height = 4.5, dpi = 300)
```

Figure 4.3: yield del tesoro de los Estados Unidos con ggplot2

4.6 Duración y Convexidad de un Bono

4.6.1 Extración de la yield

La función `subset` permite extraer una parte de tu base según un criterio como vimos con anterioridad. En el código presentado a continuación, extraemos un valor de la yield para una fecha en especifico, 2017-03-03 y luego la dividimos por 100.

```
# Extraemos un valor en específico
t10yr_yield <- t10yr.df %>%
               subset(fecha == "2017-03-03")

t10yr_yield <- as.numeric(t10yr_yield$DGS10)*0.01
```

4.6.2 Duración

Existen dos Duraciones, la de Macaulay y modificada (o de Hicks), las que miden sensibilidad del precio ante cambios de la yield. Dos bonos con la misma duración tendrá el mismo cambio en precio estimado.
Macaulay:

$$\text{Duracion de Macaulay} = \left[\frac{1+y}{y} - \frac{1+y+[n\cdot(c-y)]}{[c\cdot((1+y)^n-1)]+y}\right]$$

Modificada:

$$\text{Duracion Modificada} = \text{Duracion de Macaulay}/(1+y)$$

Aproximación Duración Modificada:

$$Aprox.Dur.Mod. = \frac{MV_- - MV_+}{2\cdot\Delta y\cdot MV_0}$$

4.6.2.1 Duración Macaulay

Como ya presentamos la formula de la duración de Macaulay, la construimos como una función.

```
# duracion de Macaulay
macaulay <- function(y,n,c,t,T){
         mac <- (1 + y)/y - (1+y+(n*(c-y)))/(c*((1+y)^n -1) + y)
         print(mac)
}
```

Usandola con la yield extraida:

```
macaulay <- macaulay(t10yr_yield,10,0.03)
```

```
## [1] 8.817
```

4.6.2.2 Duración Modificada

La aplicación de la duración modificada es directa.

```
# duración modificada
modificada <- macaulay/(1+t10yr_yield)
modificada
```

```
## [1] 8.603
```

4.6.2.3 Aproximación Duración Modificada

Una aproximación a la duración Modificada se puede obtener como:

```
# Para la aproximación de la duración modificada
precio.arriba <- precio.bono(p = 100, tc = 0.03, t = 10, y = t10yr_yield + 0.01)
precio        <- precio.bono(p = 100, tc = 0.03, t = 10, y = t10yr_yield)
precio.abajo  <- precio.bono(p = 100, tc = 0.03, t = 10, y = t10yr_yield - 0.01)
```

Uniendo los objetos creados.

```
# Calculo de aproximación duración modificada
aprox.dur.mod <- (precio.abajo - precio.arriba)/(2 * precio * 0.01)
aprox.dur.mod
```

```
## [1] 8.62
```

Así obtenemos una diferencia de 0.0169.

4.6.3 Duración con librería

Una librería útil para calcular las duraciones como la convexidad (en la siguiente secciòn) es `derivmkts`. Si no estamos usando la librería `pacman` recordar:

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

```
# Con librerias
p_load("derivmkts")

# Duración moficada
duration(precio, 3, 10, 100, 1, modified = TRUE)
```

```
## [1] 8.603
```

```
# Duración Macaulay
duration(precio, 3, 10, 100, 1, modified = FALSE)
```

```
## [1] 8.817
```

Si escribimos `modified = TRUE`, la función `duration` computa la duración modificada, por otro lado, si `modified = FALSE` obtenemos la duración de Macaulay. Losa valores son los mismo obtenidos sin librería.

4.6.4 Convexidad

La convexidad es la segunda derivada de la curva de rendimiento y es màs precisa que la duración cuando el cambio de la *yield* es màs "grande". Esto es porque la duración es la línea tangente en el punto calculado de la curva de rendimientos, el problema es que a medida que nos alejamos por la curva de rendimientos la distancia entre la curva y esa línea calculada se vuelve cada vez más grande.

$$\text{Convexidad} = \frac{1}{P \times (1+y)^2} \sum_{t=1}^{T} \left[\frac{CF_t}{(1+y)^t} (t^2 + t) \right]$$

Donde:

- P : Precio Bono.
- y : *yield to maturity*.
- T : Madurez en años.
- CF_t : *Cash flow* en el tiempo t.

```
# Calculamos medida de convexidad
convexidad <- (precio.arriba + precio.abajo  - 2 * precio)/(precio * (0.01)^2)
convexidad
```

```
## [1] 88.45
```

4.6.4.1 Aproximación Convexidad

$$\text{Aproximacion Convexidad} = \frac{MV_- + MV_+ - 2 * MV_0}{MV_0 * \Delta y^2}$$

donde:

- MV_0 : Precio del Bono.
- MV_- : Precio del Bono cuando la tasa de interes aumenta.
- MV_+ : Precio del Bono cuando la tasa de interes disminuye.
- Δy : Cambio en la tasa de interes.

4.6.5 Convexidad con librería:

```
convexity(precio, 3, 10, 100, 1)
```

```
## [1] 88.34
```

4.7 Efecto Dolar

$$\text{Efecto Dolar} = \Delta P_{duration} + \Delta P_{convexity}$$

Donde:

$$\text{Duration Dollar Change} = -D \times \Delta y \times P$$

Donde:

- D : Duración.
- Δy : Cambio en la *yield*.
- P : Precio Bono.

$$\text{Convexity Dollar Change} = 0.5 \times C \times (\Delta y)^2 \times P$$

Donde:

- C : Convexidad.
- $(\Delta y)^2$: Cambio en la *yield* al cuadrado.
- P : Precio Bono.

4.8 Ejercicio

4.9 Recursos del capítulo

Chapter 5

Opciones, Derivados y Futuros

IMPORTANTE: Aún no está del todo listo el formato en pdf, por lo que recomiendo verlo online.