Aula 1: Programação em R para finanças e Análise exploratória de dados

Marcus L. Nascimento

14 de novembro de 2025

1. R Software

2. Tipos de objetos em R

3. Estruturas de dados em R

4. Análise exploratória

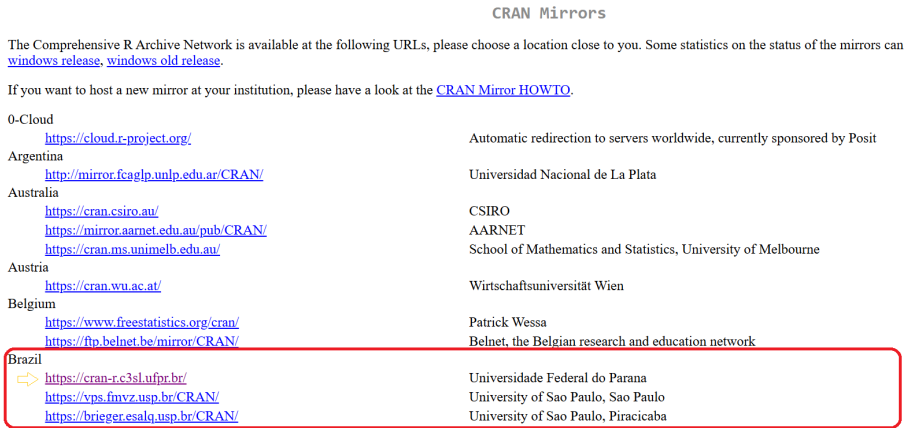
R Software

Introdução

- **R**: linguagem de programação para computação estatística e geração de gráficos.
  - Software livre;
  - Software de código aberto (Open-source software - OSS);
  - Compilado e executado em uma ampla variedade de plataformas UNIX e sistemas semelhantes (incluindo FreeBSD e Linux), além de Windows e macOS.
- Poderosa ferramenta para manipulação de dados, cálculos e exibição gráfica.
- Ambiente no qual diversas técnicas estatísticas são implementadas, tornando-o atrativo para modelagem financeira e econômica.

Instalação

- **Endereço**: `https://cran.r-project.org/mirrors.html`.



CRAN Mirrors

The Comprehensive R Archive Network is available at the following URLs, please choose a location close to you. Some statistics on the status of the mirrors can be found here: main page, windows release, windows old release.

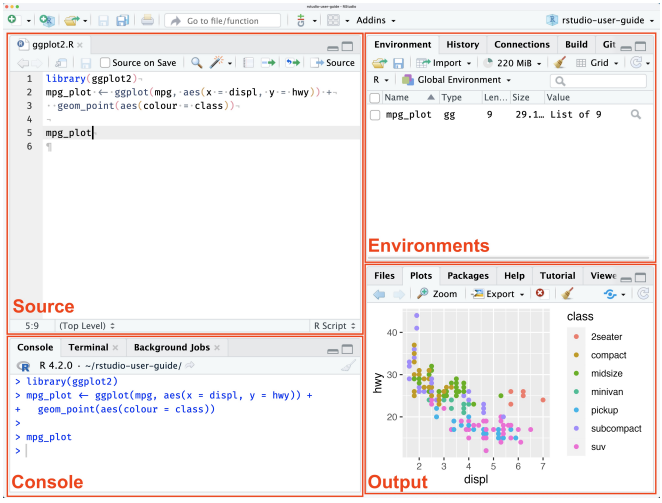
If you want to host a new mirror at your institution, please have a look at the CRAN Mirror HOWTO.

0-Cloud	
https://cloud.r-project.org/	Automatic redirection to servers worldwide, currently sponsored by Posit
Argentina	
http://mirror.fcaglp.unlp.edu.ar/CRAN/	Universidad Nacional de La Plata
Australia	
https://cran.csiro.au/	CSIRO
https://mirror.aarnet.edu.au/pub/CRAN/	AARNET
https://cran.ms.unimelb.edu.au/	School of Mathematics and Statistics, University of Melbourne
Austria	
https://cran.wu.ac.at/	Wirtschaftsuniversität Wien
Belgium	
https://www.freestatistics.org/cran/	Patrick Wessa
https://ftp.belnet.be/mirror/CRAN/	Belnet, the Belgian research and education network
Brazil	
https://cran-r.c3sl.ufpr.br/	Universidade Federal do Parana
https://vps.fmvz.usp.br/CRAN/	University of Sao Paulo, Sao Paulo
https://brieger.esalq.usp.br/CRAN/	University of Sao Paulo, Piracicaba

Ambientes de desenvolvimento (IDEs)

- Ambientes de desenvolvimento (IDEs) em R:
  - RStudio (Posit Workbench);
  - Visual Studio Code (VS Code) com extensões em R;
  - Jupyter Notebook/Lab.
- **Endereço**: `https://posit.co/download/rstudio-desktop/`

RStudio


Source
Environments
Console
Output
ggplot2.R
Source on Save
Source
library(ggplot2)
mpg_plot ← ggplot(mpg, aes(x = displ, y = hwy)) +
geom_point(aes(colour = class))
mpg_plot
Environment
History
Connections
Build
Import
220 MiB
Grid
Global Environment
Name
Type
Len...
Size
Value
mpg_plot
gg
9
29.1...
List of 9
Files
Plots
Packages
Help
Tutorial
Zoom
Export
class
2seater
compact
midsize
minivan
pickup
subcompact
suv
hwy
displ
Console
Terminal
Background Jobs
R 4.2.0 · ~/rstudio-user-guide/
> library(ggplot2)
> mpg_plot ← ggplot(mpg, aes(x = displ, y = hwy)) +
+ geom_point(aes(colour = class))
> mpg_plot

Pacotes

- R pode ser facilmente estendido por meio de pacotes.
  - Cerca de oito pacotes padrão fornecidos com a distribuição do R.
  - Muitos outros disponíveis cobrindo uma ampla variedade de técnicas estatísticas, econométricas e de aprendizado de máquina com aplicações em finanças.
  - Podemos verificar o número de pacotes através da função `available.packages()`.

```
R_packages <- available.packages(filters = "duplicates",
                                 repos = "https://cran.rstudio.com")

print(paste("There are", nrow(R_packages),
            "R packages available in CRAN as of",
            Sys.Date()))
```

  - Cada pacote possui documentação própria disponível online.

OBSERVAÇÃO: No código acima, estamos definindo uma variável `R_packages`. Variáveis são definidas através da aplicação do sinal de "=" ou "$<-$".

Pacotes

- Como os pacotes são instalados?.
  - Utilizando o console:

```
install.packages("packagename")
```

  - Utilizando a interface do RStudio: Tools > Install Packages.

- Como os pacotes são carregados?

```
library(packagename)
```

- Há ainda a possibilidade de pacotes serem instalados a partir do Github.

```
install.packages("devtools")
library(devtools)
install_github("username/repositoryname")
```

Pacotes

- Pacotes que possibilitam a extração de dados financeiros sem a necessidade de acesso em fontes externas:
  - **rbcb**: criado para permitir que o R interaja com a API do Banco Central do Brasil, o Sistema Gerenciador de Séries Temporais (SGS), permite a extração de dados de séries de preços de moedas;
  - **GetTDData**: facilita a obtenção de dados históricos do Tesouro Direto (LFT, LTN, NTN-C, NTN-B, NTN-B Principal e NTN-F);
  - **quantmod**: projetado para auxiliar *traders* quantitativos no desenvolvimento, teste e implantação de modelos de negociação baseados em estatísticas.
    - Permite a extração de dados de séries de preços de ações e índices;
    - Funciona bem para o mercado brasileiro, porém existem *bugs* para alguns ativos (por exemplo, ETF).
  - **tidyquant**: reúne os principais pacotes de análise quantitativa financeira do R em um formato *tidy*.

Referências

- "Basic R for Finance" por Diethelm Würtz, Tobias Setz, Yohan Chalabi, Longhow Lam, Andrew Ellis.
- "An Introduction to Analysis of Financial Data with R" por Ruey S. Tsay.
- "Statistical Analysis of Financial Data With Examples In R" por James Gentle.
- "Statistical Analysis of Financial Data in R" por René Carmona.
  - Pacote **Rsafd**.

Tipos de objetos em R

Resumo

Figura: Tipos de objetos mais comuns em R.

Object Type:	
double	a vector containing real values
integer	a vector containing integer values
complex	a vector containing complex values
logical	a vector containing logical values
character	a vector containing character values
NULL	NULL

- Para identificar o tipo de um objeto em R, utiliza-se a função `typeof()`.

Factors

- *Factors* são utilizados para representar dados categóricos. Por exemplo:
  - Variável `exchange` que assume os valores "NASDAQ", "NYSE"e "AMEX";
  - Variável `FinCenter` que assume os valores "Europe/Zurich"e "London".
- Cada categoria é denominada *level*. Logo, a variável `exchange` é um fator com três níveis "NASDAQ", "NYSE"e "AMEX".
- Objetos do tipo *factor* podem ser criados a partir de objetos do tipo *character* e *numeric* através da função `factor()`.

```
> exchange <- c("NASDAQ", "NYSE", "NYSE", "AMEX", "NASDAQ")
> exchange <- factor(exchange)
> exchange
[1] NASDAQ NYSE NYSE AMEX NASDAQ
Levels: AMEX NASDAQ NYSE
>
> exchange <- c(1, 2, 2, 3, 1)
> exchange <- factor(exchange)
> exchange
[1] 1 2 2 3 1
Levels: 1 2 3
```

Data

- A função `as.Date()` é utilizada para criação de objetos com a classe *Date*.
- Datas podem ser geradas a partir de objetos do tipo *character*.

```
> timeStamp <- "1973-12-09"
> Date <- as.Date(timeStamp, "%Y-%m-%d")
> Date
[1] "1973-12-09"
```

- Datas são guardadas como um objeto do tipo *double*; números podem ser adicionados e são interpretados como número de dias.

```
> Date + 19
[1] "1973-12-28"
```

Dados faltantes

- Em R, dados faltantes são representados pelo símbolo `NA` (*Not Available*).
- A função `is.na()` é utilizada na detecção de `NA`.

```
> x <- as.double(c("1", "2", "qaz"))
> x
[1]  1  2 NA
> is.na(x)
[1] FALSE FALSE  TRUE
```

- Outro símbolo importante em R é `NaN` (Not a Number), que é detectado através da função `is.nan()`.

```
> z <- sqrt(c(1, -1))
> is.nan(z)
[1] FALSE TRUE
```

OBSERVAÇÃO: note que em ambos os códigos anteriores foram atribuídos à variável `x` vetores de dados. A seguir, falaremos sobre esta e outras estruturas de dados importantes em R.

Estruturas de dados em R

Introdução

- Estruturas de dados são formas de organizar dados de maneira coerente.
- As principais estruturas de dados em R são:
  - vetor (unidimensional);
  - Matriz (bidimensional);
  - *Array* (tridimensional);
  - *Data frame* (bidimensional);
  - Série de tempo (univariada e multivariada);
  - Lista.

Vetor

- Estrutura de dados mais simples em R.
- Objeto no qual todos os elementos são do mesmo tipo.
- EXEMPLO: vetor com nome "x" composto por quatro elementos do tipo "double" (10, 5, 3, 6) construído através da função `c()`.

```
> x <- c(10, 5, 3, 6)
> x
[1] 10 5 3 6
```

- A função `c()` combina um número arbitrário de vetores em um único vetor.

```
> y <- c(x, 0.55, x, x)
> y
[1] 10.00 5.00 3.00 6.00 0.55 10.00 5.00 3.00 6.00 10.00 5.00 3.00
[13] 6.00
```

Vetor

- Os símbolos para operações aritméticas elementares são '+', '-', '*', '/'. O símbolo de potência é '^'.

```
> x
[1] 10 5 3 6
> z <- x * x
> z
[1] 100 25 9 36
> y <- x^2
> y
[1] 100 25 9 36
```

- A maioria das funções matemáticas está disponível em R e também funciona elemento a elemento do vetor. Por exemplo, a função logarítmica.

```
> log(x)
[1] 2.3026 1.6094 1.0986 1.7918
```

Figura: Algumas funções matemáticas que podem ser aplicadas a vetores.

Function:	
abs	absolute value
asin acos atan	inverse geometric functions
asinh acosh atanh	inverse hyperbolic functions
exp log	exponent and natural logarithm
floor ceiling trunc	creates integers from floating point numbers
gamma lgamma	gamma and log gamma function
log10	logarithm with basis 10
round	rounding
sin cos tan	geometric functions
sinh cosh tanh	hyperbolic functions
sqrt	square root

Matriz

- Assim como nos vetores, todos os elementos de uma matriz devem ser do mesmo tipo.
- A principal maneira de se gerar uma matriz se dá através da função `matrix()`.

```
> x <- matrix(1:8, 2, 4)
> x
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

- Por padrão, a matriz é preenchida por coluna. Para que o preenchimento seja realizado por linha, o argumento `byrow = TRUE` deve ser especificado.

```
> x <- matrix(1:8, 2, 4, byrow = TRUE)
> x
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
```

Matriz

- O produto matricial no sentido matemático é aplicado através do uso do operador '%*%'. Vale lembrar que as dimensões das matrizes devem estar em conformidade.

```
> x <- matrix(1:8, ncol = 2)
> x %*% x
Error in x %*% x : non-conformable arguments
```

- A função `t()` é aplicada para se transpor matrizes.

```
> x %*% t(x)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]  774  820  866  912  958 1004 1050
[2,]  820  870  920  970 1020 1070 1120
[3,]  866  920  974 1028 1082 1136 1190
[4,]  912  970 1028 1086 1144 1202 1260
[5,]  958 1020 1082 1144 1206 1268 1330
[6,] 1004 1070 1136 1202 1268 1334 1400
[7,] 1050 1120 1190 1260 1330 1400 1470
```

Matriz

Figura: Algumas funções que podem ser aplicadas a matrizes.

Function:	
chol(x)	Choleski decomposition
col(x)	Matrix with column numbers of the elements
diag(x)	Create a diagonal matrix from a vector
ncol(x)	Returns the number of columns of a matrix
nrow(x)	Returns the number of rows of a matrix
qr(x)	QR matrix decomposition
row(x)	Matrix with row numbers of the elements
solve(A,b)	Solve the system Ax=b
solve(x)	Calculate the inverse
svd(x)	Singular value decomposition
var(x)	Covariance matrix of the columns

Array

- Assim como nos vetores e nas matrizes, todos os elementos de um *array* devem ser do mesmo tipo.
- A principal maneira de se gerar um *array* se dá através da função `array()`.

```
> newarray <- array(c(1:8, 11:18, 111:118), dim = c(2, 4, 3))
> newarray
, , 1
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
, , 2
     [,1] [,2] [,3] [,4]
[1,]   11   13   15   17
[2,]   12   14   16   18
, , 3
     [,1] [,2] [,3] [,4]
[1,]  111  113  115  117
[2,]  112  114  116  118
```

Data frame

- *Data frames* são estruturas que podem ter colunas com dados de diferentes tipos.
- Costumam ser a estrutura de dados mais conveniente para análise de dados em R.
- Muitas das rotinas de modelagem estatística em R requerem um *data frame* como *input*.
- A principal forma de se gerar um *data frame* se dá através da função `data.frame()`. Esta pode ser utilizada tanto para a criação de um novo *data frame* quanto para converter outras estruturas em *data frames*.

```
> myLogical <- sample(c(TRUE, FALSE), size = 6, replace = TRUE)
> myNumeric <- rnorm(6)
> myCharacter <- sample(c("AA", "A", "B", "BB"), size = 6, replace = TRUE)
> myDataFrame <- data.frame(myLogical, myNumeric, myCharacter)
> myDataFrame
  myLogical myNumeric myCharacter
1     FALSE   1.13404           B
2      TRUE   0.83017           B
3      TRUE   1.87290           B
4      TRUE   0.92148           A
5     FALSE  -0.61139          AA
6      TRUE  -1.15180          AA
```

Série de tempo

- Em R, um objeto de série de tempo pode ser criado através da função `ts()`.

Figura: Argumentos da função `ts()`.

```
Arguments:
data        numeric vector or matrix of the observed values
start       time of the first observation
end         time of the last observation
frequency   number of observations per unit of time.
deltat      fraction of sampling period between observations
ts.eps      time series comparison tolerance
class       class to be given to the result
names       character vector of names for multiple time series
```

Série de tempo

- A função `ts()` combina duas componentes:
  - (i) Vetor ou matriz de dados de valores numéricos;
  - (ii) Marcação temporal. Importante notar que a marcação é equidistante no tempo, ou seja, temos que a função `ts()` gera séries de tempo regulares.

- EXEMPLO: série temporal univariada mensal com início em janeiro de 1987 e 100 intervalos de tempo.

```
> ts(data = round(rnorm(100), 2), start = c(1987), freq = 12)
       Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
1987  0.31  0.24 -0.11 -0.22  0.01  0.99 -0.43  0.40  0.12  0.17  0.22  1.26
1988 -2.04 -1.45 -1.95  0.58 -0.11  0.00 -0.28 -0.44  0.72  0.43 -1.17  0.48
1989  1.82 -0.88 -1.20  0.62  1.46 -0.92  0.23 -0.68  0.36  1.56  0.63  0.51
1990  1.41 -2.82 -0.74  0.89  0.15 -1.07  0.09  0.40  0.09  0.44 -0.29 -0.96
1991  1.27 -0.42 -0.08 -0.32  0.80 -0.06 -1.04  0.28  0.84  0.77 -1.25  0.24
1992  0.62  0.80  0.87  0.82  0.96  0.73 -0.89  0.26 -2.09  1.19 -0.07  0.63
1993 -1.01 -0.36 -0.26  1.74  2.10  0.37 -0.58 -0.07 -0.56  0.50 -0.02 -0.13
1994  0.74 -1.02 -0.87  0.65 -0.59  0.58  0.54  1.23 -0.02  0.16  0.31 -0.12
1995 -0.02  1.68 -0.44  0.54
```

Série de tempo

- EXEMPLO: série temporal bivariada mensal com início em abril de 1987 e 12 intervalos de tempo.

```
> ts(data = matrix(rnorm(24), ncol = 2), start = c(1987, 4), freq = 12)
          Series 1  Series 2
Apr 1987 -0.11846 -0.775010
May 1987  2.04562  0.770052
Jun 1987  0.73857  0.478091
Jul 1987  1.45352  0.256210
Aug 1987  0.35742 -0.120296
Sep 1987  1.68397 -0.983123
Oct 1987 -0.98398  0.627804
Nov 1987  0.62864  1.660877
Dec 1987  0.14493 -0.079505
Jan 1988  1.24564  0.128253
Feb 1988 -1.70675  0.743185
Mar 1988 -1.36858  0.754068
```

Lista

- Lista é um vetor no qual as entradas podem ser objetos de qualquer tipo e estrutura.
- Uma lista pode conter outra lista e pode ser aplicada na construção de estruturas de dados arbitrárias.
- A principal forma de se gerar um *data frame* se dá através da função `list()`.

```
> x1 <- 1:5
> x2 <- c(TRUE, TRUE, FALSE, FALSE, TRUE)
> myList <- list(numbers = x1, wrong = x2)
> myList
$numbers
[1] 1 2 3 4 5
$wrong
[1] TRUE TRUE FALSE FALSE TRUE
```

Lista

- Os itens de uma lista podem ser acessados de diferentes formas:
  - Número do elemento: `myList[1]` indica o primeiro elemento da lista `myList`. Retorna, portanto, um objeto do tipo lista.

```
> myList[1]
$numbers
[1] 1 2 3 4 5
```

  - Número do componente: `myList[[1]]` indica o conteúdo do primeiro elemento da lista `myList`.

```
> myList[[1]]
[1] 1 2 3 4 5
```

  - Nome do componente: `myList$nome` indica o elemento da lista `myList` com nome *name*.

```
> myList$wrong
[1]  TRUE  TRUE FALSE FALSE  TRUE
```

Análise exploratória

Extração de dados

- Para extração de dados, exploraremos um pouco do pacote **tidyquant**.
- O pacote **tidyquant** integra uma coleção de pacotes chamada **tidyverse**.
  - A coleção **tidyverse** é especialmente desenhada para ciência de dados.
  - Todos seus pacotes compartilham a mesma filosofia, gramática e estruturas de dados.
- **tidyquant** é, provavelmente, a melhor ferramenta para coleta e análise de dados financeiros em R.
- Como exemplo, analisaremos os preços das ações da Apple

```
# Get stock prices for Apple stock from Yahoo! finance site.
aapl_stock_prices <- tq_get("AAPL")
```

Extração de dados

- Para visualizar o objeto que há na variável `aapl_stock_prices` aplicamos a função `str()`.

```
> # str function is a way to display the structure of an R object.
> str(aapl_stock_prices)
tibble [2,696 x 8] (S3: tbl_df/tbl/data.frame)
 $ symbol  : chr [1:2696] "AAPL" "AAPL" "AAPL" "AAPL" ...
 $ date    : Date[1:2696], format: "2015-01-02" "2015-01-05" ...
 $ open    : num [1:2696] 27.8 27.1 26.6 26.8 27.3 ...
 $ high    : num [1:2696] 27.9 27.2 26.9 27 28 ...
 $ low     : num [1:2696] 26.8 26.4 26.2 26.7 27.2 ...
 $ close   : num [1:2696] 27.3 26.6 26.6 26.9 28 ...
 $ volume  : num [1:2696] 2.13e+08 2.57e+08 2.63e+08 1.60e+08 2.37e+08 ...
 $ adjusted: num [1:2696] 24.3 23.6 23.6 23.9 24.8 ...
```

- Temos 2696 observações diárias para 7 variáveis: data, preço na abertura, máxima, mínima, preço no fechamento, volume e preço ajustado.

Tibble

- Dentre as estruturas de dados vistas anteriormente, não havia nenhuma denominada *tibble*.
- *Tibbles* são estruturas de dados que são parte do pacote **tibble**, que por sua vez integra a coleção **tidyverse**.
- *Tibbles* são *data frames* ajustados com intuito de deixá-los mais amigáveis para cientistas de dados e podem ser gerados de duas formas:
  - Convertendo um *data frame* em *tibble* através da função `as_tibble()`;
  - A partir de vetores individuais, utilizando a função `tibble()`.

```
tibble(
    coluna1 = c("a", "b", "c", "d"),
    coluna2 = 1:4,
    coluna3 = coluna2 ^ 2,
    coluna4 = 0
)
```

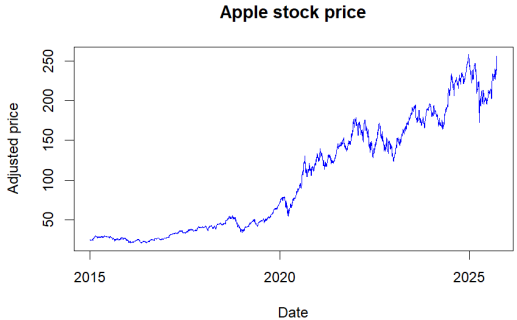
Tibble

- As funções `head()` e `tail()` permitem que as primeiras e últimas observações da base sejam visualizadas.

```
> head(aapl_stock_prices)
# A tibble: 6 x 8
  symbol date         open  high   low close    volume adjusted
  <chr>  <date>      <dbl> <dbl> <dbl> <dbl>     <dbl>    <dbl>
1 AAPL   2015-01-02  27.8  27.9  26.8  27.3 212818400     24.3
2 AAPL   2015-01-05  27.1  27.2  26.4  26.6 257142000     23.6
3 AAPL   2015-01-06  26.6  26.9  26.2  26.6 263188400     23.6
4 AAPL   2015-01-07  26.8  27.0  26.7  26.9 160423600     23.9
5 AAPL   2015-01-08  27.3  28.0  27.2  28.0 237458000     24.8
6 AAPL   2015-01-09  28.2  28.3  27.6  28.0 214798000     24.9
>
> tail(aapl_stock_prices)
# A tibble: 6 x 8
  symbol date         open  high   low close    volume adjusted
  <chr>  <date>      <dbl> <dbl> <dbl> <dbl>     <dbl>    <dbl>
1 AAPL   2025-09-15  237   238.  235.  237.  42699500     237.
2 AAPL   2025-09-16  237.  241.  236.  238.  63421100     238.
3 AAPL   2025-09-17  239.  240.  238.  239.  46508000     239.
4 AAPL   2025-09-18  240.  241.  237.  238.  44249600     238.
5 AAPL   2025-09-19  241.  246.  240.  246. 163741300     246.
6 AAPL   2025-09-22  248.  257.  248.  256. 105413200     256.
```

Análise Exploratória

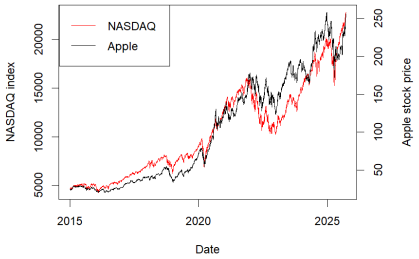
```
> plot(aapl_stock_prices$date, aapl_stock_prices$adjusted,
>      type = "l", xlab = "Date", ylab = "Adjusted price",
>      main = "Apple stock price", col = "blue")
```



- Um comando mais atual para geração de figuras em R se dá através da função ggplot() pertencente ao pacote **ggplot2**.

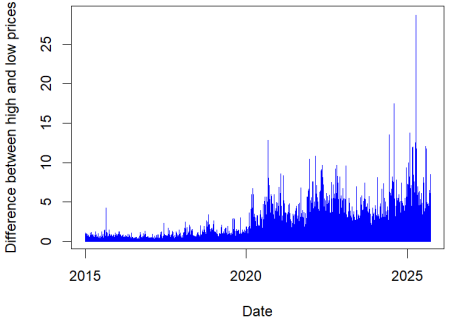
Análise Exploratória

```
> ndq = tq_get(c("^IXIC"), get = "stock.prices")
> par(mar = c(5, 5, 2, 5))
> plot(ndq$date, ndq$adjusted, type = "l", col = "red",
>      ylab = "NASDAQ index", xlab = "Date")
> par(new = T)
> plot(aapl_stock_prices$date, aapl_stock_prices$adjusted,
>      type = "l", axes = F, xlab = NA, ylab = NA, cex = 1.2)
> axis(side = 4)
> mtext(side = 4, line = 3, "Apple stock price")
> legend("topleft",
>        legend=c("NASDAQ", "Apple"),
>        lty = 1, col = c("red", "black"))
```



Análise Exploratória

```
> aapl_diff = aapl_stock_prices$high - aapl_stock_prices$low
> plot(aapl_stock_prices$date, aapl_diff, type = "h",
>      xlab = "Date",
>      ylab = "Difference between high and low prices",
>      col = "blue")
```



Análise Exploratória

```
> highest_change = which.max(aapl_diff)
> when = aapl_stock_prices$date[highest_change]
> top = aapl_diff[highest_change]
> plot(aapl_stock_prices$date, aapl_diff, type = "h",
>      xlab = "Date",
>      ylab = "Difference between high and low prices",
>      col = "blue")
> # Here, we add the red point.
> points(when, top, pch = 19, col = "red", cex = 1.5)
```