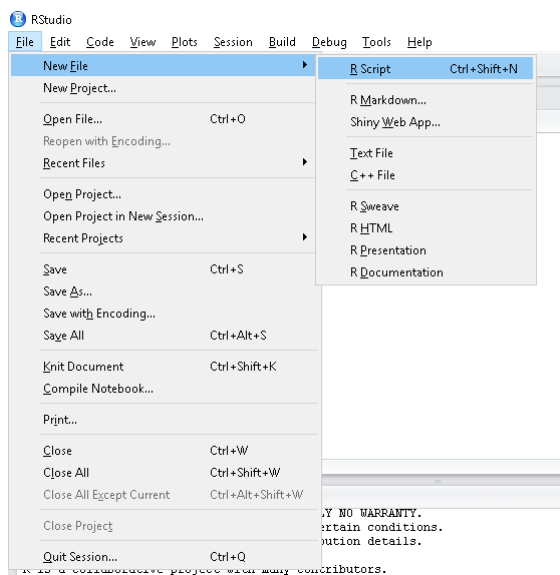


Conceitos Básicos – Como Executar um Script

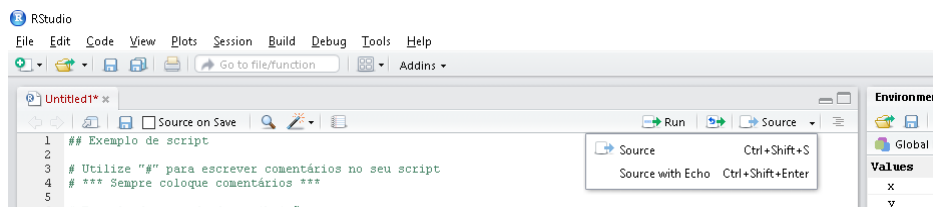
Scripts

- Arquivos textos com lista de comandos para serem executados
- Para criar um novo script: [File] [R Script] ou *Ctrl+Shift+N*



- Execução:

- Linha-a-Linha ou parcial: selecionar a linha + *Ctrl + Shift + Enter*
- Via menu:



- Podem ser gravados para serem executados posteriormente

Conceitos Básicos – Como Executar um Script

Exemplo de Script:

C:/Public/FGV-MPFE/Materias/MPFE-Metodos Quantitativos/Modulos/Introdução ao R/R - RStudio

File Edit Code View Plots Session Build Debug Tools Help

Go to file/function Addins

ExemploDeScript.r *

Source on Save Run Source

```

1  ## Exemplo de script
2
3  # Utilize "#" para escrever comentários no seu script
4  # *** Sempre coloque comentários ***
5
6  # Exemplo de comando de atribuição
7  x = 1;
8
9  # Finalizar os comandos com a mudança de linha ou com ';'
10 y = 2;
11
12 z = x + y;
13
14 print(z);
15
16 ## Tudo em uma linha
17 x = 1; y = 2; z = x + y; print(z);
18
19

```

1:1 [Top Level]

R Script

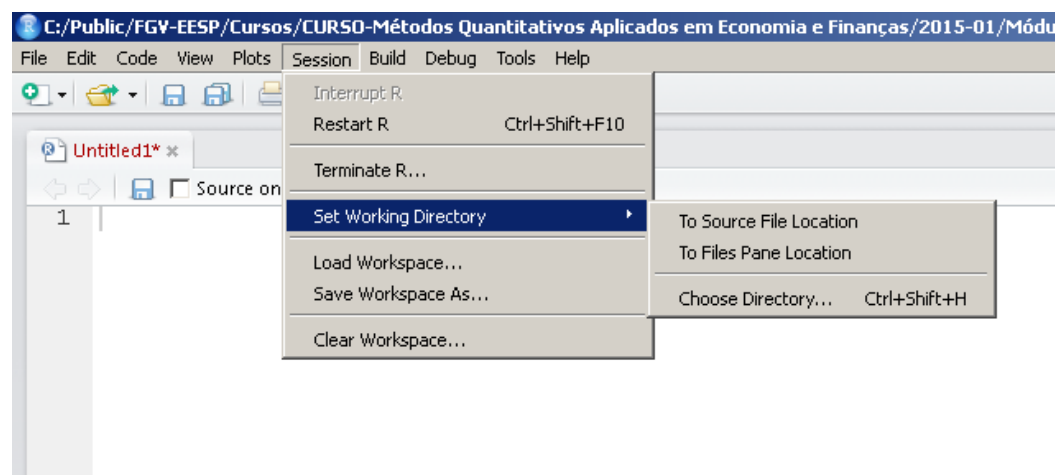
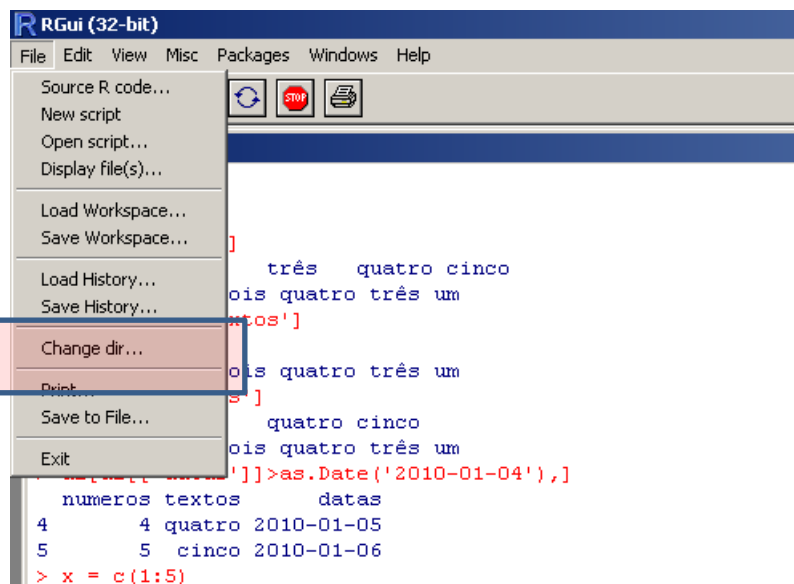
Conceitos Básicos – Como Executar um Script

Execução de script gravados:

```
source (NomeDoScript) ;
source (CaminhoAbsolutoDoScript) ;
source (CaminhoRelativoDoScript) ;
```

Atenção:

- Verificar o diretório corrente para se referenciar e executar o script.
- No R para indicar diretório, utilizar “/” e não “\”. Ex: C:/tmp/exemplo.r



Conceitos Básicos – Tipos de Dados

Tipos Básicos:

- **Numeric**

Números Reais

```
> x = 10
> x = 20/3
> print(x)
[1] 6.666667
```

- **String**

Sequência de caracteres ou texto

Indicado utilizando aspas simples (‘ ’) ou aspas duplas (“ ”)

```
> umastring = "uma string"
> uma_string = "uma string"
> outra_string = 'outra string'
> print (uma_string)
[1] "uma string"
> print(outra_string)
[1] "outra string"
```

Conceitos Básicos – Tipos de Dados

Tipos Básicos:

- **Boolean** : Verdadeiro/Falso; 1 = Verdadeiro /Não 1 = Falso

Atribuição direta:

```
> x = TRUE
> y = FALSE
> print(x)
[1] TRUE
> print(y)
[1] FALSE
```

```
> z = 0
> z == TRUE
[1] FALSE
> z = 1
> z == TRUE
[1] TRUE
> z = 2
> z == TRUE
[1] FALSE
> z = -10
> z == TRUE
[1] FALSE
```

Resultado de expressões lógicas:

```
> x = 10
> y = 20
> z = x == y
> z
[1] FALSE
> z = x == (y-10)
> z
[1] TRUE
```

```
> ontem = Sys.Date()-1
> hoje = Sys.Date()
> z = ontem == (hoje-1)
> z
[1] TRUE
```

Conceitos Básicos – Tipos de Dados

Tipos Básicos:

- **Date**

Padrão de representação: *yyyy-mm-dd*

Indicada como uma string entre aspas simples ou duplas

Conversão: utilizar a função `as.Date(<string>)`

```
> hoje = Sys.Date()
> print(hoje)
[1] "2015-01-07"
> ontem = as.Date("2015-01-06")
> print(ontem)
[1] "2015-01-06"
> strMyDate = "06/01/2015"
> myDate = as.Date(strMyDate)
> myDate
[1] "0006-01-20"
> myDate = as.Date(strMyDate, format = "%d/%m/%Y")
> myDate
[1] "2015-01-06"
```

Conceitos Básicos – Tipos de Dados

Tipos Básicos:

- NA

Not Available: A variável está definida mas o valor não está disponível

```
> x = NA
> is.na(x)
[1] TRUE
> y = 10
> is.na(y)
[1] FALSE
```

Tipo usado, por exemplo, para indicar que em uma amostra de dados uma ou mais características de um indivíduo, ou de uma das observações da amostra não está disponível.

- NULL

Variável não está definida. Não utiliza Espaço de memória do computador

```
> x = NULL
> is.null(x)
[1] TRUE
> is.na(x)
logical(0)
Warning message:
In is.na(x) : is.na() applied to non-(list or vector) of type 'NULL'
> y=NA
> is.null(y)
[1] FALSE
```

Conceitos Básicos – Operadores

Operador de Atribuição

| Operação | Símbolo | Exemplo | Resultado |
|------------|---------|-----------------|-----------|
| Atribuição | = <- | x = 5 x <- 5 | |
| Range | : | 3 : 5 | 3 4 5 |

Operadores Aritméticos

| Operação | Símbolo | Exemplo |
|---------------|---------|-------------|
| Adição | + | 5 + 3 |
| Subtração | - | 5 - 3 |
| Multiplicação | * | 5 * 3 |
| Divisão | / | 5 / 3 |
| Potenciação | ^ | 5 ^ (1 / 2) |

Conceitos Básicos – Operadores

Operadores Relacionais


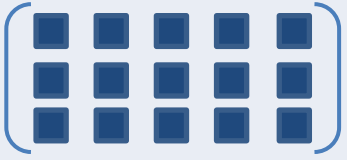
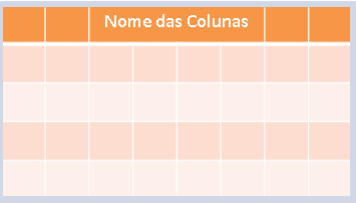
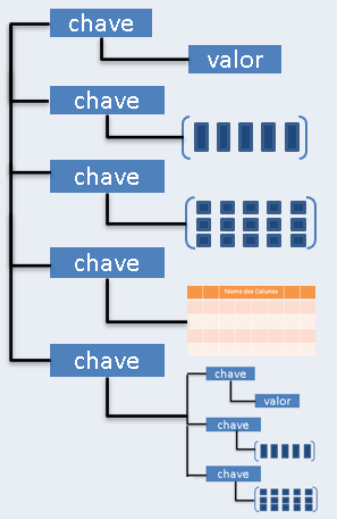
| Operador | Símbolo | Exemplo | Resultado |
|----------------|--------------|---------|-----------|
| Igualdade | == | 5 == 3 | FALSE |
| Diferença | != | 5 != 3 | TRUE |
| Maior que | > | 5 > 3 | TRUE |
| Menor que | < | 5 < 3 | FALSE |
| Maior ou Igual | >= | 5 >= 3 | TRUE |
| Menor ou Igual | <= | 5 <= 3 | FALSE |

Operadores Lógicos

| Operador | Símbolo | Exemplo | Resultado |
|-------------|-------------------|------------------------------|---------------|
| E vetorial | & | (TRUE, FALSE) & (TRUE, TRUE) | (TRUE, FALSE) |
| OU vetorial | | (TRUE, FALSE) (TRUE, TRUE) | (TRUE, TRUE) |
| E escalar | && | TRUE && FALSE | FALSE |
| OU escalar | | TRUE FALSE | TRUE |
| Negação | ! | !(5 < 3) | TRUE |

Conceitos Básicos – Estruturas de Dados

Principais Estruturas de dados

| Estrutura de Dados | Exemplo | Estruturas no R | |
|-----------------------|--|--|--|
| Vetores |  | <code>c(...)</code> | <pre>x = c(10,20,30,40,50) x = c("foo","bar","bass") [1] "foo" "bar" "bass"</pre> |
| Matrizes Numéricas |  | <code>array(...)</code> <code>matrix(...)</code> | <pre>x = array(1:6,c(2,3)) [,1] [,2] [,3] [1,] 1 3 5 [2,] 2 4 6</pre> |
| Tabelas / “Planilhas” |  | <code>data.frame(...)</code> <code>data.table(...)</code> | <pre>x = data.frame(a=c(10,20), b=c("foo","bar")) a b 1 10 foo 2 20 bar</pre> |
| Listas |  | <code>list(...)</code> | <pre>x = list(v=c(10,20,30), m=matrix(1:6,2,3), df=data.frame(a=c(10,20), b=c("foo","bar"))) \$v [1] 10 20 30 \$m [,1] [,2] [,3] [1,] 1 3 5 [2,] 2 4 6 \$df a b 1 10 foo 2 20 bar</pre> |

Conceitos Básicos – Estruturas de Dados

- **Vetores**

- **Atenção:** Todos os dados armazenados no R são vetores
- Estrutura unidimensional
- Para criar um vetor utilizar o comando “concatenar”: `c()`

```
> x = c(10,20,30)
> print(x)
[1] 10 20 30
> s = c('string 01',"string 02", "string 03")
> print(s)
[1] "string 01" "string 02" "string 03"
> b = c(TRUE,FALSE,TRUE)
> print(b)
[1] TRUE FALSE TRUE
> d = c(Sys.Date(),as.Date('2012-01-01'), as.Date('2013-10-01'))
[1] "2015-01-07" "2012-01-01" "2013-10-01"
> x = as.Date(c("2020-01-03","2020-02-10","2020-04-20"))
[1] "2020-01-03" "2020-02-10" "2020-04-20"
```

Conceitos Básicos – Estruturas de Dados

- **Vetores** (continuação)

- Para acessar um elemento do vetor:

`NomeVariavel[<lista index>]`

- Primeiro elemento é o 1
- Utilizar o operador Range ‘:’

```
> x = c(10,20,30)
```

```
> x[2]
```

```
[1] 20
```

```
> s = c('string 01',"string 02", "string 03")
```

```
> s[2]
```

```
[1] "string 02"
```

```
> s[2:3]
```

```
[1] "string 02" "string 03"
```

```
> y = 1:10
```

```
> y[5]
```

```
[1] 5
```

```
> y[6:9]
```

```
[1] 6 7 8 9
```

Conceitos Básicos – Estruturas de Dados

- **Vetores** (continuação)

```
> x = as.Date(c("2020-01-03", "2020-02-10", "2020-04-20"))
> x[2]
[1] "2020-02-10"
> x = c(Sys.Date()+1:5)
[1] "2020-04-30" "2020-05-01" "2020-05-02" "2020-05-03" "2020-05-04"
> x[3:4]
[1] "2020-05-02" "2020-05-03"
> x[c(1, 3, 5)]
[1] "2020-04-30" "2020-05-02" "2020-05-04"
```

Conceitos Básicos – Estruturas de Dados

- **Vetores** (continuação)
- Os elementos dos vetores podem ser nomeados e acessados pelo nome

```
> n = c('primeiro'=1, 'segundo'=2, terceiro=3)
```

```
> n
```

```
primeiro  segundo terceiro
          1          2          3
```

```
> n[c('primeiro','terceiro')]
```

```
primeiro terceiro
          1          3
```

```
> n[c(1,3)]
```

```
primeiro terceiro
          1          3
```

```
> names(n)
```

```
[1] "primeiro" "segundo"  "terceiro"
```

```
> names(n) = c("um","dois","tres")
```

```
> n
```

```
um dois tres
  1    2    3
```

```
> n[c("um","tres")]
```

```
um tres
  1    3
```

Conceitos Básicos – Estruturas de Dados

Operações lógicas com vetores:

NomeVariavel[<Expressão Lógica>]

Função: length(<VetorBase>)

Operador %in%: <VetorBase> %in% <VetorReferência>

```
> x = 1:20
> x
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

> length(x)
[1] 20

> x[x>5 & x <10]
[1] 6 7 8 9

> x[x%%2==0]
[1] 2 4 6 8 10 12 14 16 18 20

> y = seq(0,20,by=2)
> y
[1] 0 2 4 6 8 10 12 14 16 18 20

> y %in% x
[1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

Conceitos Básicos – Estruturas de Dados

Comandos para criar sequências e vetores:

- Sequências:

```
seq(...)
```

```
seq(from = ..., to = ..., by = ...)
```

Sequência crescente: 'from' sempre presente na sequência

```
> x = seq(from=1,to=10,by=2)
[1] 1 3 5 7 9
```

Sequência decrescente: 'from' sempre presente na sequência

```
> x = seq(from=10,1,by = -2)
[1] 10 8 6 4 2
```

Sequência crescente: 'from' sempre presente na sequência

```
> x = seq(from=as.Date('2010-01-05'),to=as.Date('2010-08-05'), by = "2 month")
[1] "2010-01-05" "2010-03-05" "2010-05-05" "2010-07-05"
```

Sequência decrescente: 'from' sempre presente na sequência

```
> x = seq(from=as.Date('2010-08-01'),to=as.Date('2010-01-20'), by = "-2 month")
[1] "2010-08-01" "2010-06-01" "2010-04-01" "2010-02-01"
```


Conceitos Básicos – Estruturas de Dados

Comandos para criar sequências:

Reversão da ordem de elementos de sequências (*Reverse Elements*):

```
rev(...)
```

Garantir que o último elemento esteja presente na sequência: Sequência decrescente + Reversão

```
> x = seq(from=10, 1, by = -2)
```

```
[1] 10 8 6 4 2
```

```
> rev(x)
```

```
[1] 2 4 6 8 10
```

```
> x = seq(from=as.Date('2010-01-20'), to=as.Date('2010-08-01'), by = "2 month")
```

```
[1] "2010-01-20" "2010-03-20" "2010-05-20" "2010-07-20"
```

```
> x = seq(from=as.Date('2010-08-01'), to=as.Date('2010-01-20'), by = "-2 month")
```

```
[1] "2010-08-01" "2010-06-01" "2010-04-01" "2010-02-01"
```

```
> x = rev(x)
```

```
[1] "2010-02-01" "2010-04-01" "2010-06-01" "2010-08-01"
```

Único Comando: Sequência decrescente + Reversão

```
> x = rev(seq(from=as.Date('2010-08-01'), to=as.Date('2010-01-20'), by = "-2 month"))
```

```
[1] "2010-02-01" "2010-04-01" "2010-06-01" "2010-08-01"
```

Conceitos Básicos – Estruturas de Dados

Aplicação: Dias Úteis

Gerar a sequência de pagamento do fluxo de uma NTN-F

https://www.anbima.com.br/pt_br/informar/taxas-de-titulos-publicos.htm

| Títulos Públicos Federais | | | | | | | | | | 28/Abr/2020 |
|---------------------------|-------------------|---------------------------|------------|-----------|-----------------|--------------|----------------------|-------------|--------------|--------------|
| Papel PREFIXADO | | NTN-F - Taxa (% a.a.)/252 | | | | | | | | |
| Código SELIC | Data Base/Emissão | Data de Vencimento | Tx. Compra | Tx. Venda | Tx. Indicativas | PU | Intervalo Indicativo | | | |
| | | | | | | | Mínimo (D0) | Máximo (D0) | Mínimo (D+1) | Máximo (D+1) |
| 950199 | 05/02/2010 | 01/01/2021 | 2,8675 | 2,8518 | 2,8600 | 1.077,384884 | 2,4017 | 3,6617 | 2,4019 | 3,2789 |
| 950199 | 09/03/2012 | 01/01/2023 | 4,9117 | 4,8972 | 4,9050 | 1.153,549862 | 4,0749 | 7,0511 | 4,0714 | 6,1972 |
| 950199 | 10/01/2014 | 01/01/2025 | 6,4763 | 6,4599 | 6,4684 | 1.167,279587 | 5,5489 | 8,6705 | 5,5470 | 7,7797 |
| 950199 | 15/01/2016 | 01/01/2027 | 7,3452 | 7,3302 | 7,3385 | 1.165,529524 | 6,4324 | 9,5153 | 6,4311 | 8,6423 |
| 950199 | 05/01/2018 | 01/01/2029 | 7,7259 | 7,6985 | 7,7156 | 1.170,249165 | 6,8381 | 9,7054 | 6,8373 | 8,9471 |
| 950199 | 10/01/2020 | 01/01/2031 | 8,0020 | 7,9704 | 7,9882 | 1.170,763258 | 7,1355 | 9,8829 | 7,1350 | 9,1615 |

Package: bizdays

Conceitos Básicos – Estruturas de Dados

Aplicação: Dias Úteis

Gerar a sequência de pagamento do fluxo de uma NTN-F

Package: bizdays

```
## Package bizdays
library(bizdays);

## Cria o calendário ANBIMA
calendar = create.calendar("ANBIMA", holidaysANBIMA, weekdays=c("saturday", "sunday"))

## Verifica se é um dia útil
is.bizday("2013-01-01",calendar); ## ou is.bizday("2013-01-01","ANBIMA");

## Ajusta para o próximo dia útil
adjust.next("2013-01-01", calendar)

## Exemplo: NTN-F
dtVencTo = as.Date("2025-01-01");

## Data de referência
dtRef = Sys.Date();

## Sequência
diasPgto = rev(seq(from=dtVencTo, to=dtRef, by="-6 months"));
is.bizday(diasPgto,calendar)

## Ajusta para o próximo dia útil
diasPgto = adjust.next(diasPgto, calendar)

## Dias úteis até as datas de pagamento
wdays = bizdays(from=dtRef,to=diasPgto,calendar)
```

Conceitos Básicos – Estruturas de Dados

Comandos para criar sequencias e vetores:

- Repetição:

```
rep(x, ...)
```

```
rep.int(x, times)
```

```
> x = rep(c(1:3),10)
```

```
> x
[1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
```

```
> x = rep(c(1:3),5)
```

```
> x
[1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
```

```
> d = rep(c("um","dois"),3)
```

```
> d
[1] "um"    "dois"  "um"    "dois"  "um"    "dois"
```

Conceitos Básicos – Estruturas de Dados

- **Arrays**

- Estruturas de dados com uma, duas ou mais dimensão
- Criada com o comando: `array()` ou `matrix()`

```
array (data = NA, dim = c(dim01,dim02,...), dimnames = NULL)
```

```
matrix (data = NA, nrow=1, ncol=1, byrow=FALSE, dimnames=NULL)
```

- Acessar os elementos utilizando: [`<index01>`,`<index02>`,...]
- Operadores *range* para cada dimensão individual
- Linhas e colunas podem ser nomeadas
- `matrix()` são exclusivas para números
- Números: diferença significativa de performance para procedimentos numéricos
- Não matrizes de *Date* ou *Strings*.

Conceitos Básicos – Estruturas de Dados

1ª dimensão

NomeVariavel [$\underbrace{\langle \text{Índices/ExprLógica} \rangle}_{1^{\text{a}} \text{ dimensão}}, \underbrace{\langle \text{Índices/ExprLógica} \rangle}_{2^{\text{a}} \text{ dimensão}}, \dots]$

Inicialização com um único valor

```
> m2d = array(1,c(2,3))
```

```
> m2d
```

| | [,1] | [,2] | [,3] |
|------|------|------|------|
| [1,] | 1 | 1 | 1 |
| [2,] | 1 | 1 | 1 |

Inicialização com todos os valores

```
> m2d = array(1:6,c(2,3))
```

```
> m2d
```

| | [,1] | [,2] | [,3] |
|------|------|------|------|
| [1,] | 1 | 3 | 5 |
| [2,] | 2 | 4 | 6 |

Inicialização com rotação

```
> m2d = array(1:2,c(2,3))
```

```
> m2d
```

| | [,1] | [,2] | [,3] |
|------|------|------|------|
| [1,] | 1 | 1 | 1 |
| [2,] | 2 | 2 | 2 |

Inicialização com 'NA'

```
> m2d = array(NA,c(2,3))
```

```
> m2d
```

| | [,1] | [,2] | [,3] |
|------|------|------|------|
| [1,] | NA | NA | NA |
| [2,] | NA | NA | NA |

Conceitos Básicos – Estruturas de Dados

- Arrays - Indexação

Todas as colunas



NomeVariavel[<ListaÍndice ou ExprLógica> ,]

NomeVariavel[, <ListaÍndice ou ExprLógica>]



Todas as linhas

```
> m2d = array(1:12,c(4,3))
```

```
> m2d
```

| | [,1] | [,2] | [,3] |
|------|------|------|------|
| [1,] | 1 | 5 | 9 |
| [2,] | 2 | 6 | 10 |
| [3,] | 3 | 7 | 11 |
| [4,] | 4 | 8 | 12 |

```
## Apenas as linhas 1 e 2
```

```
> m2d[c(1,2),]
```

| | [,1] | [,2] | [,3] |
|------|------|------|------|
| [1,] | 1 | 5 | 9 |
| [2,] | 2 | 6 | 10 |

```
## Linhas com elementos da coluna 2 maiores que 6
```

```
> m2d[m2d[,2]>6,]
```

| | [,1] | [,2] | [,3] |
|------|------|------|------|
| [1,] | 3 | 7 | 11 |
| [2,] | 4 | 8 | 12 |

```
## linhas 2 e 3, colunas 2 e 3
```

```
> m2d[c(2,3),c(2,3)]
```

| | [,1] | [,2] |
|------|------|------|
| [1,] | 6 | 10 |
| [2,] | 7 | 11 |

```
## Apenas a linha 2
```

```
> m2d[2,]
```

```
[1] 2 6 10
```

```
## Apenas a coluna 2
```

```
> m2d[,2]
```

```
[1] 5 6 7 8
```

Conceitos Básicos – Estruturas de Dados

• Arrays – Indexação e Dimensões

```
> m2d = array(1:12,c(4,3))
```

```
> m2d
```

```
      [,1] [,2] [,3]
[1,]     1     5     9
[2,]     2     6    10
[3,]     3     7    11
[4,]     4     8    12
```

```
## Apenas as linhas 1 e 2
```

```
> m2d[c(1,2),]
```

```
      [,1] [,2] [,3]
[1,]     1     5     9
[2,]     2     6    10
```

```
## linhas 2 e 3, colunas 2 e 3
```

```
> m2d[c(2,3),c(2,3)]
```

```
      [,1] [,2]
[1,]     6    10
[2,]     7    11
```

```
## Apenas a linha 2
```

```
> m2d[2,]
```

```
[1]  2  6 10
```

```
## Apenas a coluna 2
```

```
> m2d[,2]
```

```
[1]  5  6  7  8
```

```
## Linhas com elementos da coluna 2 maiores que 6
```

```
> m2d[m2d[,2]>6,]
```

```
      [,1] [,2] [,3]
[1,]     3     7    11
[2,]     4     8    12
```

```
## Dimensões do Array
```

```
> dim(m2d)
```

```
[1]  4  3
```


Conceitos Básicos – Estruturas de Dados

- Arrays – Nome das Dimensões

```
> x = array(1:6,c(2,3))
> rownames(x) = c("Linha01","Linha02");
> colnames(x) = c("Col01","Col02","Col03");
```

```
> x
```

| | Col01 | Col02 | Col03 |
|---------|-------|-------|-------|
| Linha01 | 1 | 3 | 5 |
| Linha02 | 2 | 4 | 6 |

```
> x["Linha01",c("Col02","Col03")]
```

| Col02 | Col03 |
|-------|-------|
| 3 | 5 |

```
> x["Linha02",]
```

| Col01 | Col02 | Col03 |
|-------|-------|-------|
| 2 | 4 | 6 |

Conceitos Básicos – Operações com Vetores

- Operações aritméticas com vetores e matrizes

```
# Potência, soma e produto de vetores
```

```
> x = c(1:5)
```

```
> x^2
```

```
[1] 1 4 9 16 25
```

```
> y = c(11:15)
```

```
> x+y
```

```
[1] 12 14 16 18 20
```

```
> x^2+y
```

```
[1] 12 16 22 30 40
```

```
> x * y
```

```
[1] 11 24 39 56 75
```

```
> x + 5
```

```
[1] 6 7 8 9 10
```

```
## Produto matricial. Operador %*%
```

```
> m = array(1:10,c(2,5))
```

```
> m
```

```
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

```
> m %*% x
```

```
      [,1]
[1,]    95
[2,]   110
```

Conceitos Básicos – Exercício

Utilizar a base de dados:

BBG-BMFBOVESPA-Equities-PX_LAST.xlsx

e o script:

VALE5-vol.r

Para calcular :

- a) A série histórica de retornos logaritmo diário da ação VALE5
- b) Calcular a média da série histórica de retornos calculadas no item (a)
- c) Calcular a volatilidade (desvio padrão) da séries histórica de retornos

Conceitos Básicos – Estruturas de Dados

Funções:

`sum (. . .)` : calcula a soma dos elementos de um vetor

`mean (. . .)` : calcula a média dos elementos de um vetor

`sd (. . .)` : calcula o desvio padrão dos elementos de um vetor

`var (. . .)` : calcula a variância dos elementos de um vetor

`cov (. . .)` : calcula a covariância dos elementos de uma matriz

```
> help("var")
```

Correlation, Variance and Covariance (Matrices)

Description

`var`, `cov` and `cor` compute the variance of `x` and the covariance or correlation of `x` and `y` if these are vectors. If `x` and `y` are matrices then the covariances (or correlations) between the columns of `x` and the columns of `y` are computed.

`cov2cor` scales a covariance matrix into the corresponding correlation matrix *efficiently*.

Usage

```
var(x, y = NULL, na.rm = FALSE, use)
```

```
cov(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))
```

Conceitos Básicos – Exercício

Utilizar a base de dados:

BBG-BMFBOVESPA-Equities-PX_LAST.xlsx

e o script:

Equities-covariancia.r

Para as ações CSNA3, ELET6, PETR3, PETR4, VALE5, IBOV

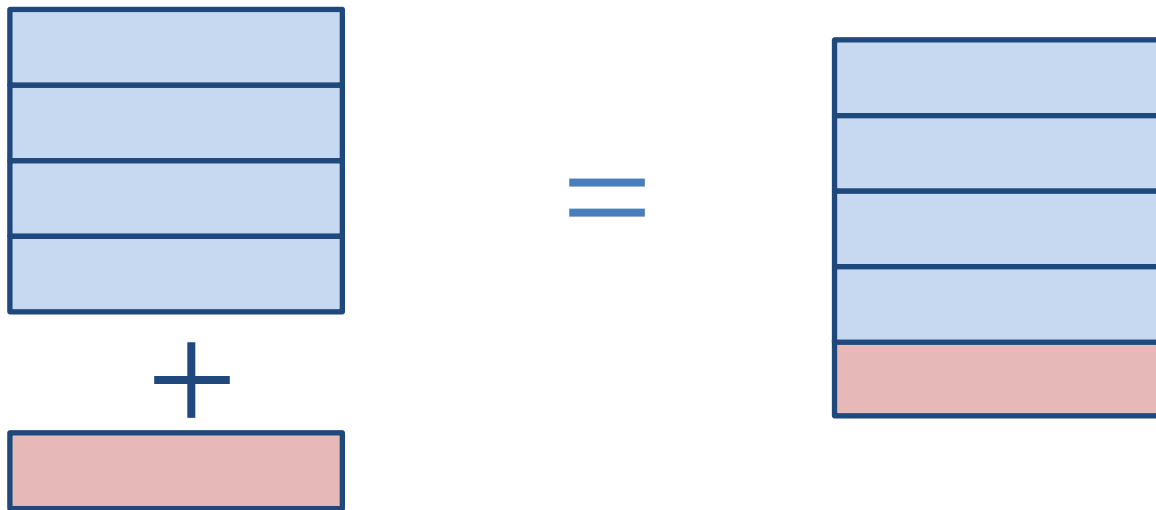
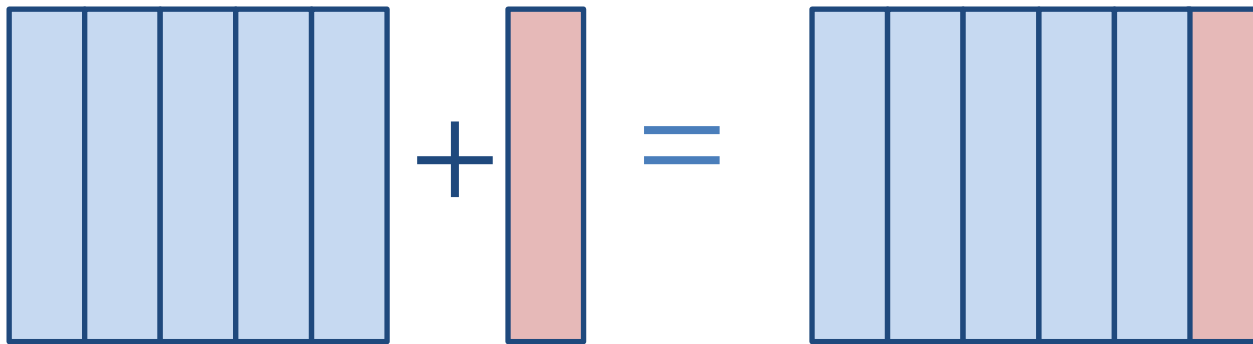
- a) A série histórica de retornos logaritmo diário das ações
- b) Calcular a matriz de covariância das ações

Conceitos Básicos – Estruturas de Dados

Funções:

`cbind(...)`: “cola” dados em colunas (*columns*)

`rbind(...)`: “cola” dados em linhas (*rows*)



Conceitos Básicos – Estrutura de Dados

- Data Frames
- Estrutura de dados que armazena dados de tipo diferentes mas com a **mesma dimensões**

```
NomeDaDataFrame = data.frame(Nome=Valor,...)
```

- Acesso aos dados:

```
NomeDaDataFrame$NomeColuna
```

```
NomeDaDataFrame[[ 'NomeColuna' ]]
```

```
NomeDaDataFrame[Linha, 'Nome' /NúmeroColuna]
```

etc...

- Indexação e operações lógicas análogas a vetores e matrizes

Conceitos Básicos – Estrutura de Dados

- Data Frames - Exemplo

```
> df = data.frame(numeros=1:5,
                  textos=c("um", "dois", "três", "quatro", "cinco"),
                  datas=as.Date('2010-01-01')+1:5)
```

```
> df
  numeros textos      datas
1       1     um 2010-01-02
2       2    dois 2010-01-03
3       3   três 2010-01-04
4       4 quatro 2010-01-05
5       5   cinco 2010-01-06
```

```
> df[[1]]
[1] 1 2 3 4 5
> df[['textos']]
[1] um     dois   três   quatro cinco
Levels: cinco dois quatro três um
```

```
> df[c(2,5), 'textos']
[1] dois   cinco
Levels: cinco dois quatro três um
```

```
> df[2:5, 'textos']
[1] dois   três   quatro cinco
Levels: cinco dois quatro três um
```

```
> df[df[['datas']]>as.Date('2010-01-04'),]
  numeros textos      datas
4       4 quatro 2010-01-05
5       5   cinco 2010-01-06
```


Conceitos Básicos – Estrutura de Dados

- Data Frames - Exemplo

```
### Data Frame
data("mtcars")

# A data frame with 32 observations on 11 variables.
#
# [, 1] mpg   Miles/(US) gallon
# [, 2] cyl   Number of cylinders
# [, 3] disp  Displacement (cu.in.)
# [, 4] hp    Gross horsepower
# [, 5] drat  Rear axle ratio
# [, 6] wt    Weight (1000 lbs)
# [, 7] qsec  1/4 mile time
# [, 8] vs    V/S
# [, 9] am    Transmission (0 = automatic, 1 = manual)
# [,10] gear  Number of forward gears
# [,11] carb  Number of carburetors

## Tipo de estrutura de dados
class(mtcars)

## número linhas
nrow(mtcars);

## número colunas
ncol(mtcars);

## Calcular a média do 'mpg' por 'cyl'
mpgByCylMean = aggregate(mpg ~ cyl, mtcars, mean);

## Calcular o desvio padrão do 'mpg' por 'cyl'
mpgByCylStd = aggregate(mpg ~ cyl, mtcars, sd);
```

Conceitos Básicos – Estrutura de Dados

- Data Tables
- Estrutura de dados que armazena dados de tipo diferentes mas com a **mesma dimensões**

```
NomeDaDataTable = data.table(Nome=Valor,...)
```

- Eficiência na manipulação de dados e agrupamentos

`DataTable[i: linhas/seleção, j: colunas ou agrupamento, by][order(...)]`

- Acesso aos dados:

Semelhante ao *data.frame*

Coluna da coluna: `NomeDaDataTable[<i>, "Nome"]`

Conteúdo da coluna: `NomeDaDataTable[<i>, Nome]`

- Seleções e Agrupamentos:

```
Datatable[condições, .(agrupamentos), keyby=.(NomesCampos)]
```

- Operações de totalização, médias, agrupamentos

Conceitos Básicos – Estrutura de Dados

- Data Tables:

<https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html>

```
## Packages
library(data.table);

## Carrega a base de operações de Renda Fixa
dt = as.data.table(read.table("../database/27-04-2020_NEGOCIOSBALCAO.csv", sep = ";",
                             dec=",", header=TRUE, skip = 1, stringsAsFactors = FALSE));

## Colunas Originais
names(dt)

## Definição dos nomes
names(dt) = c("Instr", "Emissor", "CodIF", "QtdNeg", "PrecoNeg", "Volume", "TxNeg", "OrigNeg",
              "Horario", "Data", "Cod"); ## Formas de acesso

## Colunas do Data Table
x = dt[Instr=="CRA", c("Emissor", "QtdNeg", "PrecoNeg")]

### - Modos de seleção

## Conteúdo da coluna
x = dt[, Emissor]

## Conteúdo da coluna
x = dt[["Emissor"]]

## Coluna do Data Table
x = dt[, "Emissor"]

## Conteúdo da coluna
x = dt[Instr=="DEB", Emissor]

## Coluna do Data Table
x = dt[Instr=="DEB" & Emissor=="LIGHT", .(SomaVolume=sum(Volume), PrecoMedio=mean(PrecoNeg))]

## Coluna do Data Table
x = dt[Instr=="DEB" & Emissor=="LIGHT",
      .(SomaVolume=sum(Volume), PrecoMedio=mean(PrecoNeg)),
      keyby=.(Instr, Emissor, CodIF)][order(CodIF)]
```

Conceitos Básicos – Exercício

Utilizar a base de dados:

BBG-BMFBOVESPA-Equities-PX_LAST.xlsx

e o script:

Equities-covariância-data.frame.r

Para calcular a matriz de covariância das ações:

CSNA3, ELET6, PETR3, PETR4, VALE5, IBOV

Conceitos Básicos – Estrutura de Dados

- List

- Estrutura de dados que armazena dados de tipo e dimensões diferentes

```
NomeDaLista = list(Nome=Valor,...)
```

- Acesso aos dados:

```
NomeDaLista$Nome
```

```
NomeDaLista[[ 'Nome' ]]
```

```
> list01 = list(a=1:10, d=Sys.Date(), m=array(1:6,c(2,3)))
```

```
> list01
```

```
$a
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
$d
```

```
[1] "2015-01-19"
```

```
$m
```

```
      [,1] [,2] [,3]
```

```
[1,]     1     3     5
```

```
[2,]     2     4     6
```

```
> list01$m[2,c(2,3)]
```

```
[1] 4 6
```

```
> list01[['m']][2,c(2,3)]
```

```
[1] 4 6
```

Conceitos Básicos – Estrutura de Dados

- List: Exemplo de Aplicação
 - Armazenar dados de configurações:
 - Parâmetros de Modelos
 - Nomes de arquivos e colunas de dados
 - Endereço/Servidor e autenticação em bases de dados

Exemplo:

- Configuração de acesso a um banco de dados

```
database = list(url='sqlserver01',
               schemna='MyDatabase',
               username="ReadOnly",
               passwd='ReadOnly');
```

- Configuração para leitura de um arquivo de dados

```
datafile01 = list(path='c:/tmp/file01.dat',
                  type='csv',
                  dec='.',
                  colnames = c('DATE', 'VALE5', 'PETR4', , 'GGBR4'));
```

Conceitos Básicos – Funções

- Chamada de funções:

```
NomeDaFuncao (Arg01, Arg02, ...);
```

ou

```
NomeDaFuncao (Nome01 = Arg01, ..., NomeX=ArgX, ...);
```

- Declaração de função:

```
NomeDaFuncao = function(<Lista de Argumentos>) {
    <Comandos>
    return (<ListaDeValorDeRetorno>);
}
```

Conceitos Básicos – Funções

Exemplo da chamada da função *log*:

```
log(x, base = exp(1));
```

Onde: 2 argumentos:

x obrigatório

base opcional, se o valor da não for indicado a base default é $e=2,7182\dots$

```
## Exemplo: log(x, base = exp(1))
## Chama a função log passando um
x = log(100);
print(x); # [1] 4.60517
```

```
## Não nomeia os argumentos
x = log(100,10);
print(x); # [1] 2
```

```
## Nomeia os argumentos
x = log(100,base=10);
print(x); # [1] 2
```


Conceitos Básicos – Funções

Exercício:

Construir a função `raiz(.)`:

```
raiz(x, radix = 2);
```

Onde: 2 argumentos:

`x` obrigatório

`radix` opcional, se o valor da não for indicado a base default é 2

Conceitos Básicos – Funções

Solução: Função `raiz(x, radix = 2)`;

Onde: 2 argumentos:

`x` obrigatório

`radix` opcional, se o valor da não for indicado a base default é 2

```
## Definição de função
```

```
raiz = function(x, radix=2) {
```

```
  y = x^(1/radix);
```

```
  return(y)
```

```
}
```

```
## Exemplo execução
```

```
x = raiz(64);
```

```
print(x); # [1] 8
```

```
x = raiz(64,radix=6);
```

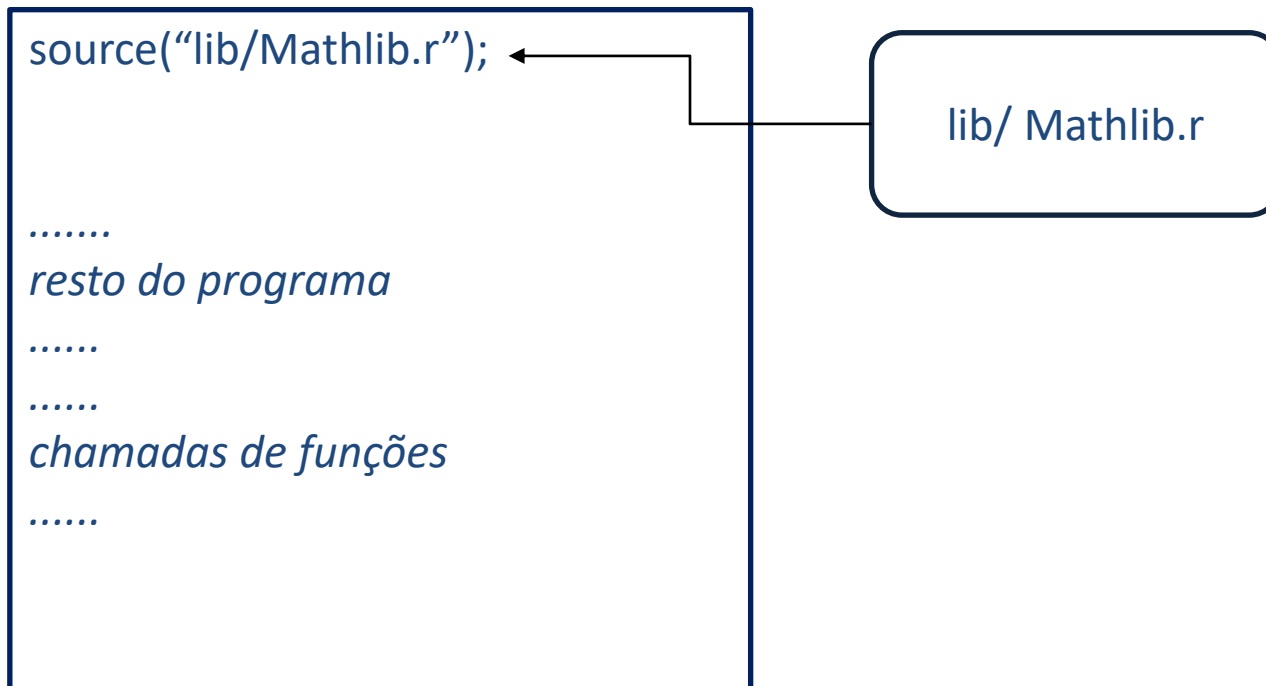
```
print(x); # [1] 2
```

Conceitos Básicos – Funções

Criar bibliotecas de funções e reutilizar o código da função em diferentes scripts ou sistemas utilizando o comando: *source(.)*

Exemplo:

- Gravar a função *radix* no arquivo *lib/Mathlib.r* e utilizar em um outro script



Conceitos Básicos – *Package*

Arquivo padrão do R para agrupar funções relacionadas entre si para e relativas a um determinado domínio do conhecimento, manipulação de tipos de dados ou acesso a dados.

Para utilizar as funções é necessário instalar o pacotes:

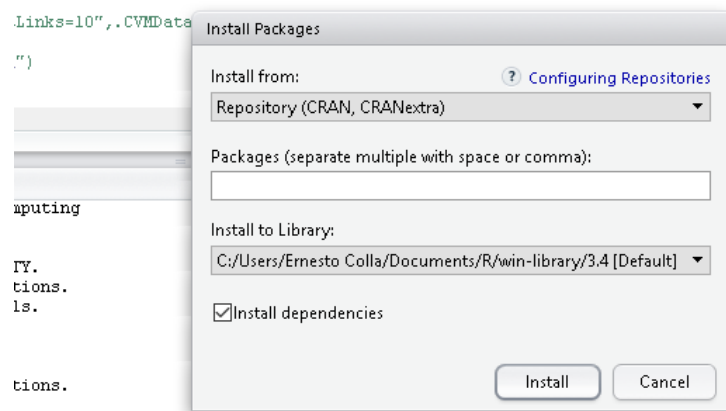
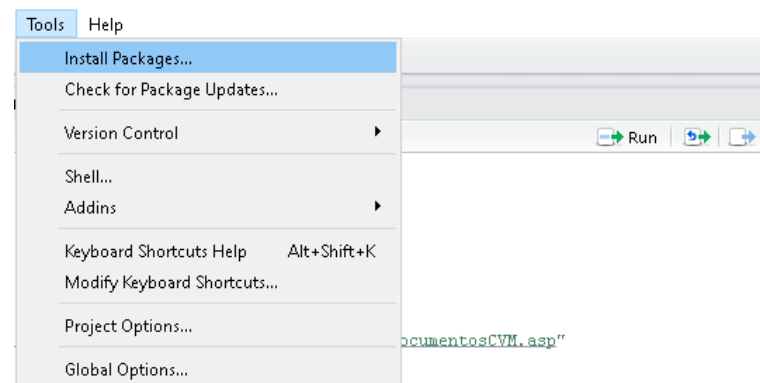
Exemplo:

XLConnect e *readxl*: funções para ler e gravar dados em planilhas Excel

lubridate: manipular séries temporais

urca e *var*: modelos VAR, VEC e testes de raiz unitária

Sintaxe: *library(<nome do pacote>)*



Conceitos Básicos – Funções

Problemas na modelagem da função:

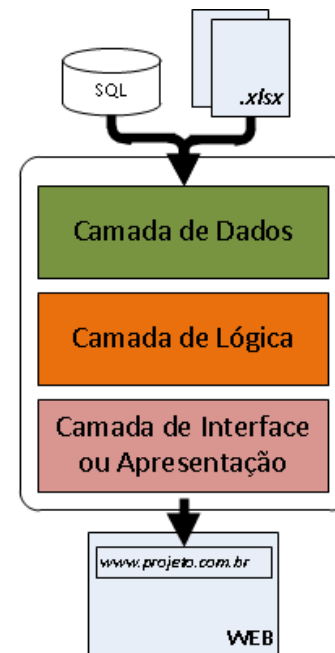
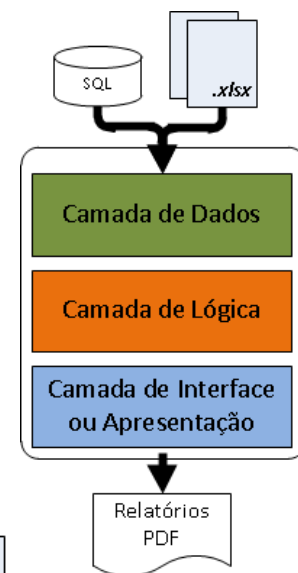
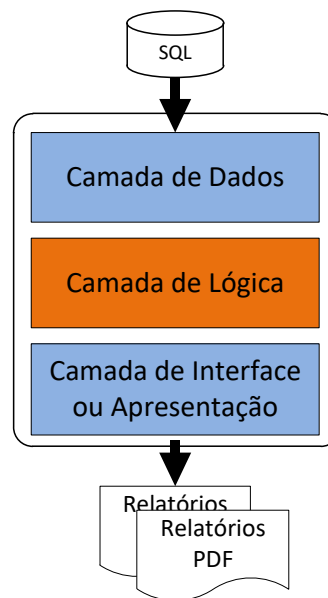
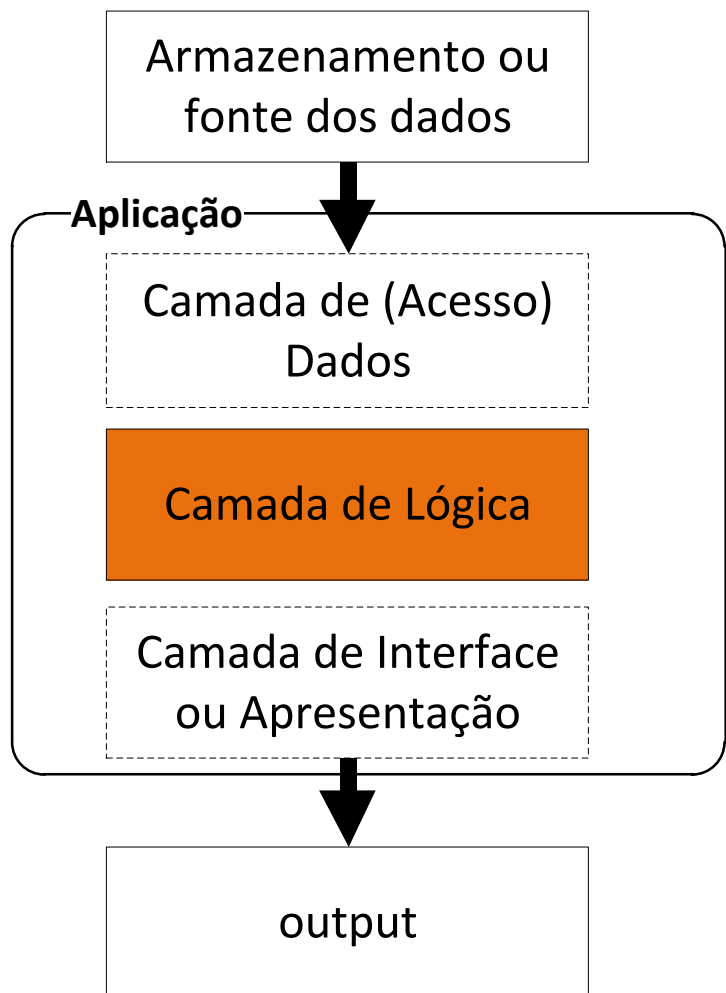
- Mudar o nome da planilha (ou a posição da planilha)
- Alterar a posição dos dados na planilha
- Alterar a forma de armazenamento dos dados
- ...entre outros.

Melhor solução:

- “Cada função deve ter uma função específica” e um programa (ou script chama as funções.

Modelagem Quantitativa - Arquitetura

Modelo de 3 Camadas



Conceitos Básicos – Funções

Exercício:

Utilizar a base de dados:

BBG-BMFBOVESPA-Equities-PX_LAST.xlsx

Construir a função `EquitiesCov (...)`:

Argumentos:

`prices`: data frame com Preços de ações

`codes` : lista de códigos das ações

Retorno:

Matriz de covariância para as ações selecionadas

Instrução:

- Utilizar como base o script: *Equities-covariancia-function.r*

Conceitos Básicos – Funções

Exercício: Com base no arquivo *Equities-covariancia-libs.r*

Utilizar a base de dados:

BBG-BMFBOVESPA-Equities-PX_LAST.xlsx

Arquivo: lib/DataLoader.r

```
XLDataLoader (pathWorkbook)
```

Argumentos:

`pathWorkbook`: path da planilha de dados

Retorno: `prices` : Matriz de histórico de preços

Arquivo: No arquivo lib/DataAnalysis.r

```
EquitiesCov (prices, codes);
```

`prices`: Matriz de histórico de preços

`codes` : lista de códigos das ações

Retorno: Matriz de covariância

Conceitos Básicos – Funções

Exercício: Construir as funções nos arquivo indicados

Retorno: `prices` : Matriz de histórico de preços

Arquivo: `lib/DataAnalysis.r`

```
EquitiesCov(prices, codes,
             from=1, interval=NULL);
```

`codes` : lista de códigos das ações

`prices` : Matriz de histórico de preços

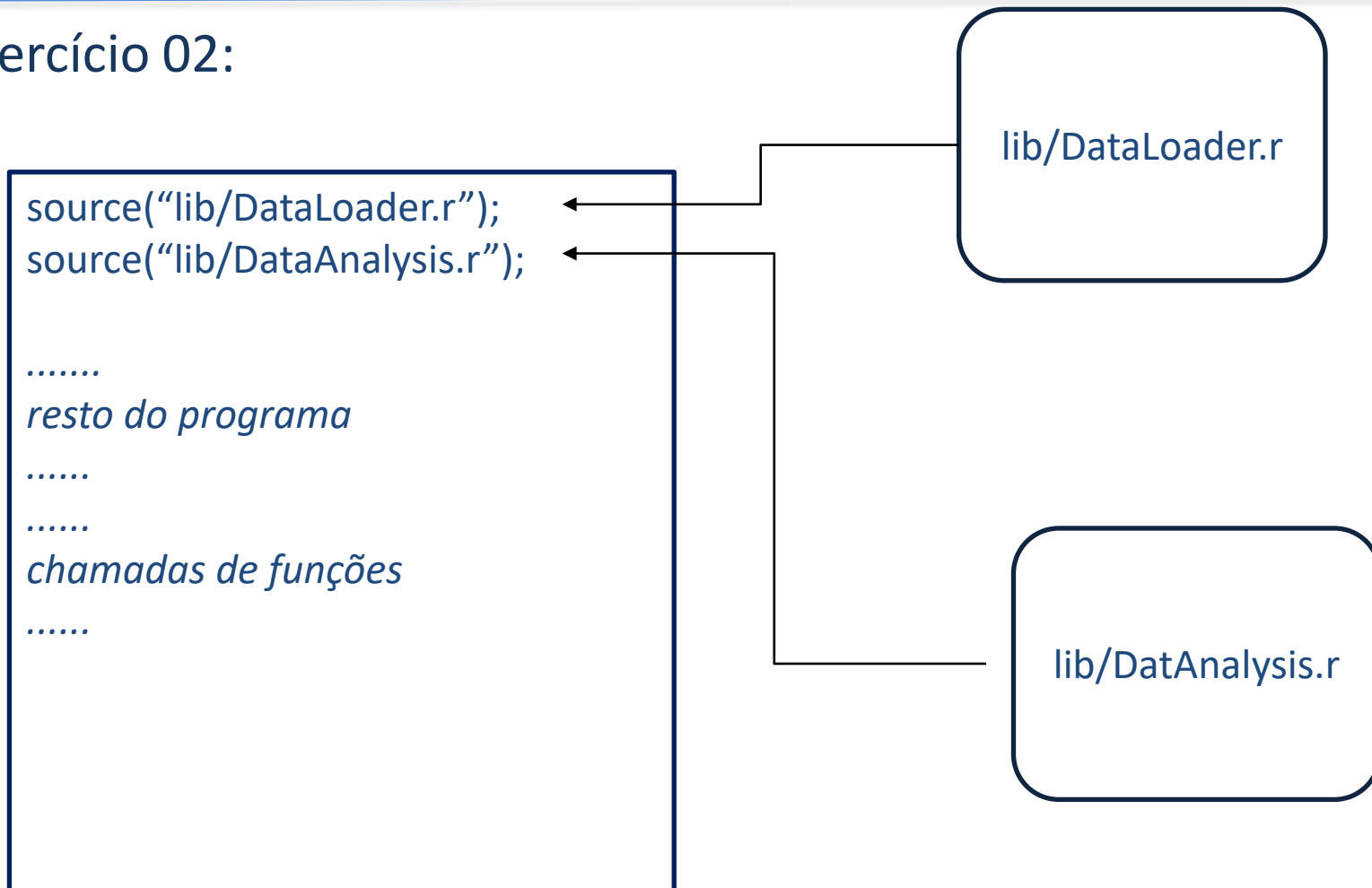
`from` : (opcional) Índice de início do intervalo de tempo

`interval` : (opcional) número de observações, se NULL utilizar o restante

Retorno: Matriz de covariância

Conceitos Básicos – Funções

Exercício 02:



Exercício: Dias Úteis

Escrever no arquivo `MetQuantLib.r` função `WorkDays(.)` que calcula o número de dias úteis entre duas datas

Inputs: *dateBegin*: data inicial

dateEnd: data final

holidays: vetor de feriados

Valor de retorno:

Número de dias úteis entre duas datas.

Instruções:

Utilizar a função: `LoadBRHolidays()` do arquivo `MetQuantLib.r` para carregar o vetor de feriados do Brasil

Planilha de suporte: `NetPresentValue-dev.xlsx`

Utilizar a função `wday(.)` do package `lubridate`.

Exercício: Valor presente líquido de fluxo de caixa

Escrever no arquivo `MetQuantLib.r` a função `NetPresentValue(.)` para calcular o valor presente líquido de um fluxo de caixa.

Inputs: *cashflow*: fluxo de caixa

periods: período como fração do período da taxa de juros

yield: taxa de juros

Valor de retorno:

Valor presente do fluxo de caixa em BRL (*cashflowBRL*) descontado pela taxa de juros (*yieldBRL %aa252*) considerando capitalização composta por dias úteis (*workdays*).

Instruções:

- Script Base: *NetPresentValue-main.r*
- Utilizar a função: *LoadCashFlow()* do arquivo `MetQuantLib.r` para carregar o fluxo de caixa
- Planilha de suporte: *NetPresentValue-dev.xlsx*

Exercício: Valor presente líquido de fluxo de caixa

Date Ref **2018-04-24**

| Date | workdays | calendardays | USDBRL | yieldBRL | yieldUSD | cashflowBRL |
|------------|----------|--------------|--------|----------|----------|---------------|
| 2018-04-24 | 0 | 0 | 3,4724 | 6,39% | 1,70% | -1.000.000,00 |
| 2018-07-27 | 66 | 94 | 3,4947 | 6,21% | 1,81% | -129.115,97 |
| 2018-09-10 | 96 | 139 | 3,5063 | 6,20% | 1,86% | 138.142,41 |
| 2018-09-26 | 108 | 155 | 3,51 | 6,20% | 1,87% | -217.502,89 |
| 2018-12-26 | 169 | 246 | 3,5296 | 6,22% | 1,98% | 263.662,15 |
| 2019-04-23 | 249 | 364 | 3,5579 | 6,35% | 2,10% | 478.415,73 |
| 2019-05-13 | 262 | 384 | 3,5627 | 6,38% | 2,11% | -453.167,34 |
| 2019-07-17 | 308 | 449 | 3,5824 | 6,51% | 2,16% | 374.111,29 |
| 2019-09-06 | 345 | 500 | 3,6019 | 6,63% | 2,20% | 251.608,41 |
| 2019-09-20 | 355 | 514 | 3,6072 | 6,67% | 2,21% | 202.923,77 |
| 2019-10-28 | 381 | 552 | 3,6208 | 6,76% | 2,23% | 379.382,50 |
| 2020-03-19 | 479 | 695 | 3,6812 | 7,16% | 2,32% | -35.321,00 |
| 2020-05-20 | 520 | 757 | 3,7102 | 7,34% | 2,34% | 474.732,86 |
| 2020-07-03 | 551 | 801 | 3,734 | 7,47% | 2,36% | 398.801,61 |
| 2020-08-06 | 575 | 835 | 3,7542 | 7,57% | 2,37% | -361.594,46 |
| 2020-12-17 | 667 | 968 | 3,8306 | 7,91% | 2,42% | 102.660,28 |
| 2020-12-30 | 675 | 981 | 3,8376 | 7,93% | 2,42% | 475.571,94 |
| 2021-05-04 | 759 | 1106 | 3,9177 | 8,22% | 2,45% | 87.086,07 |
| 2021-07-02 | 801 | 1165 | 3,9606 | 8,36% | 2,46% | -207.340,23 |
| 2021-07-19 | 812 | 1182 | 3,9719 | 8,40% | 2,47% | 499.782,46 |
| 2021-09-02 | 845 | 1227 | 4,006 | 8,50% | 2,47% | 440.190,40 |
| 2021-09-13 | 851 | 1238 | 4,0122 | 8,52% | 2,48% | 57.807,82 |
| 2022-01-25 | 944 | 1372 | 4,1099 | 8,77% | 2,50% | 128.383,22 |
| 2022-05-20 | 1023 | 1487 | 4,1947 | 8,94% | 2,51% | 152.193,22 |
| 2022-06-20 | 1043 | 1518 | 4,2173 | 8,98% | 2,51% | 39.641,87 |
| 2022-10-25 | 1132 | 1645 | 4,3184 | 9,15% | 2,52% | -464.973,85 |
| 2023-02-03 | 1203 | 1746 | 4,3919 | 9,24% | 2,53% | -8.063,49 |