Nivelamento em R

Bruno Tebaldi de Queriroz Barbosa

February 4, 2023

Table of Contents

Introdução

Objetivo

- O objetivo desta disciplina é ensinar o aplicativo R, a fim de preparar o aluno para análise de dados, e utilização de Métodos econométricos.
- Nesse sentido, pretendemos capacitar o estudante para realizar operações com o R de forma a executar tarefas de complexidade média utilizando o aplicativo R, bem como entender e adaptar códigos de outros usuários que utilizam o aplicativo R.
- Aprender R é igual aprender outra língua: você precisa treinar dezenas de vezes uma rotina até que ela se torne natural.

O que é o R?

História do R

- O pacote estatístico R nasceu como uma versão livre do pacote estatístico S. O software S foi criado nos laboratórios da Bell, em 1976, por John Chambers.
- Com a evolução do software S e a criação do S+, um grupo de professores neozelandeses da Universidade de Auckland resolveu criar uma versão aberta do S+, que veio a ser o R. A primeira versão estável do R foi lançada em 2000.
- Atualmente, o R é organizado em repositórios chamados CRAN (Comprehensive R Archive Networks), que são instituições nacionais e internacionais que hospedam o programa e as suas atualizações, bem como os mais de 12 mil pacotes estatísticos que acompanham o R.

Por que o R?

Vantagens do R

1. Aplicativo de análise estatística amplamente utilizada no meio profissional e acadêmico
2. Aplicativo para análises estatísticas + linguagem de programação
3. Criada e continuamente desenvolvida e atualizada por uma comunidade de estatísticos e programadores
4. A qualidade do software R é certificada por órgãos governamentais (ex: US FDA), corporações, instituições financeiras e academia.

Por que o R?

Vantagens do R

5. Open source e Gratuito: constantemente avaliado por estatísticos e cientistas computacionais.
6. R = Programa Base + Packages Extensível: Plataforma aberta. Qualquer um pode desenvolver uma biblioteca de funcionalidades (packages). Há milhares de bibliotecas desenvolvidas pela comunidade e disponibilizada
7. Lista de pacotes: http://cran.r-project.org/

Por que o R?

Vantagens do R

Argumentos mais técnicos....

- Alta performance, escalável e paralelizável
- Extensa biblioteca de gráficos
- Acesso a dados: arquivos texto, banco de dados, planilha excel, internet, RTD...
- Linguagem interpretada: escreve uma vez o programa (arquivo texto) e roda em todas as plataformas

O que é o R?

Desvantagens do R

No entanto, existem alguns inconvenientes em se aprender um software livre.

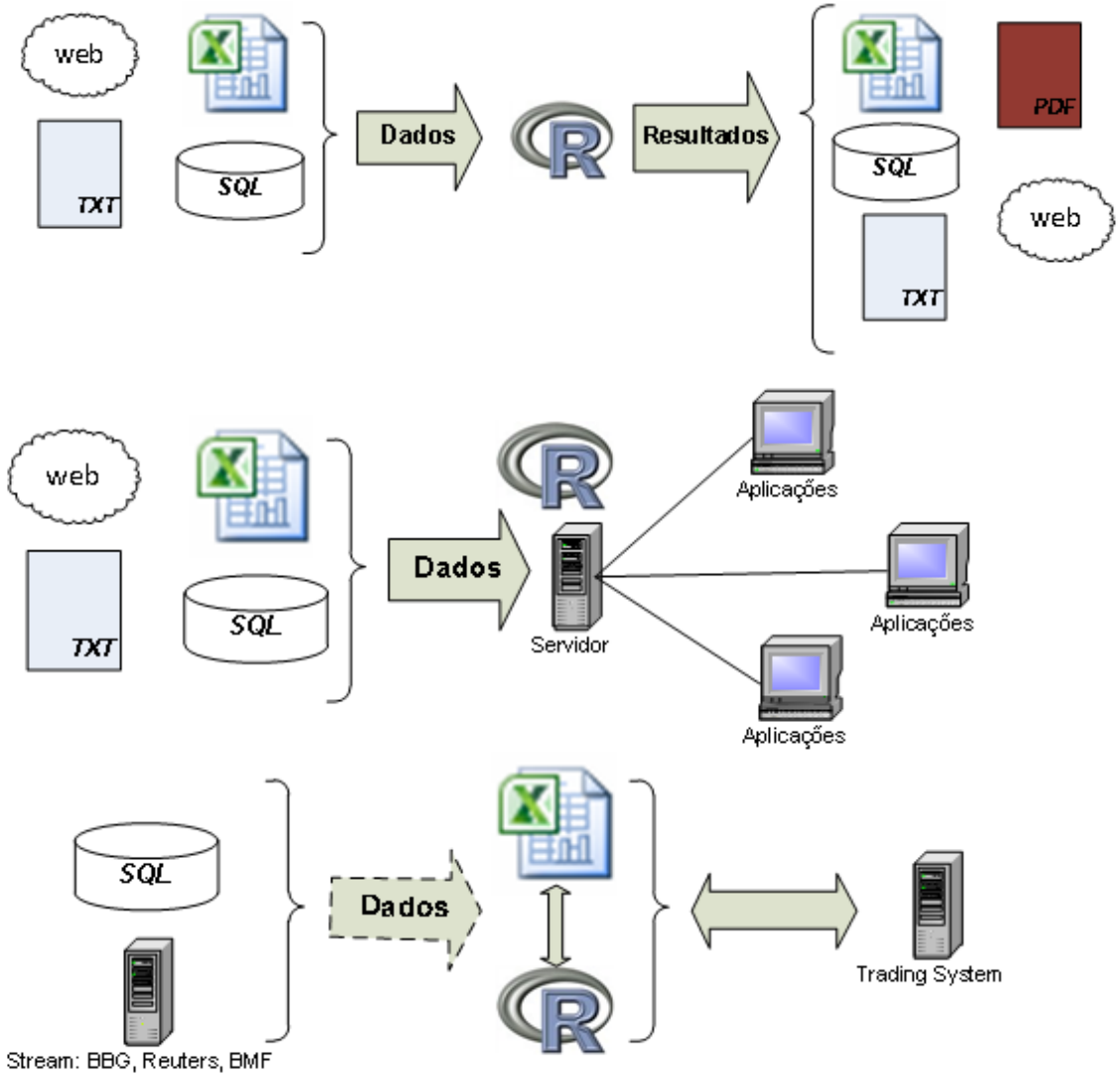
1. O primeiro é que o R é uma linguagem por script, ou seja, ele não segue a lógica *point-and-click* da maior parte dos programas.
2. O segundo é que não existe uma empresa que pode ser responsabilizada por erros no software.
3. Caso você nescessite de uma função especifica, que não existe no repositório, você mesmo terá de desenvolver a função.
4. Por ser um software livre, as atualizações são muito frequentes.

Arquitetura de 3 camadas

Você provavelmente já ouviu o termo "arquitetura de 3 camadas". Esse tipo de programação já é falado há algum tempo.
Separar seu programa em camadas é o ato de você separar sue código em "áreas" (*scripts*, funções, etc) especificas para algumas tarefas.
A abordagem típica de 3 camadas possui:

- uma interface com o usuário. No nosso caso será a saida de dados gerado pelo R. Podendo ser graficos ou arquivos texto, planilhas etc. Tudo que pode ser "Consumido pelo usuário"
- uma camada de lógica. No nosso caso será o conjunto de instruções que o R deve executar.
- uma leitura de dados. no nosso caso será representado por uma função ou rotina que faz a leitura do dado.

Sugestão de como trabalhar com R:

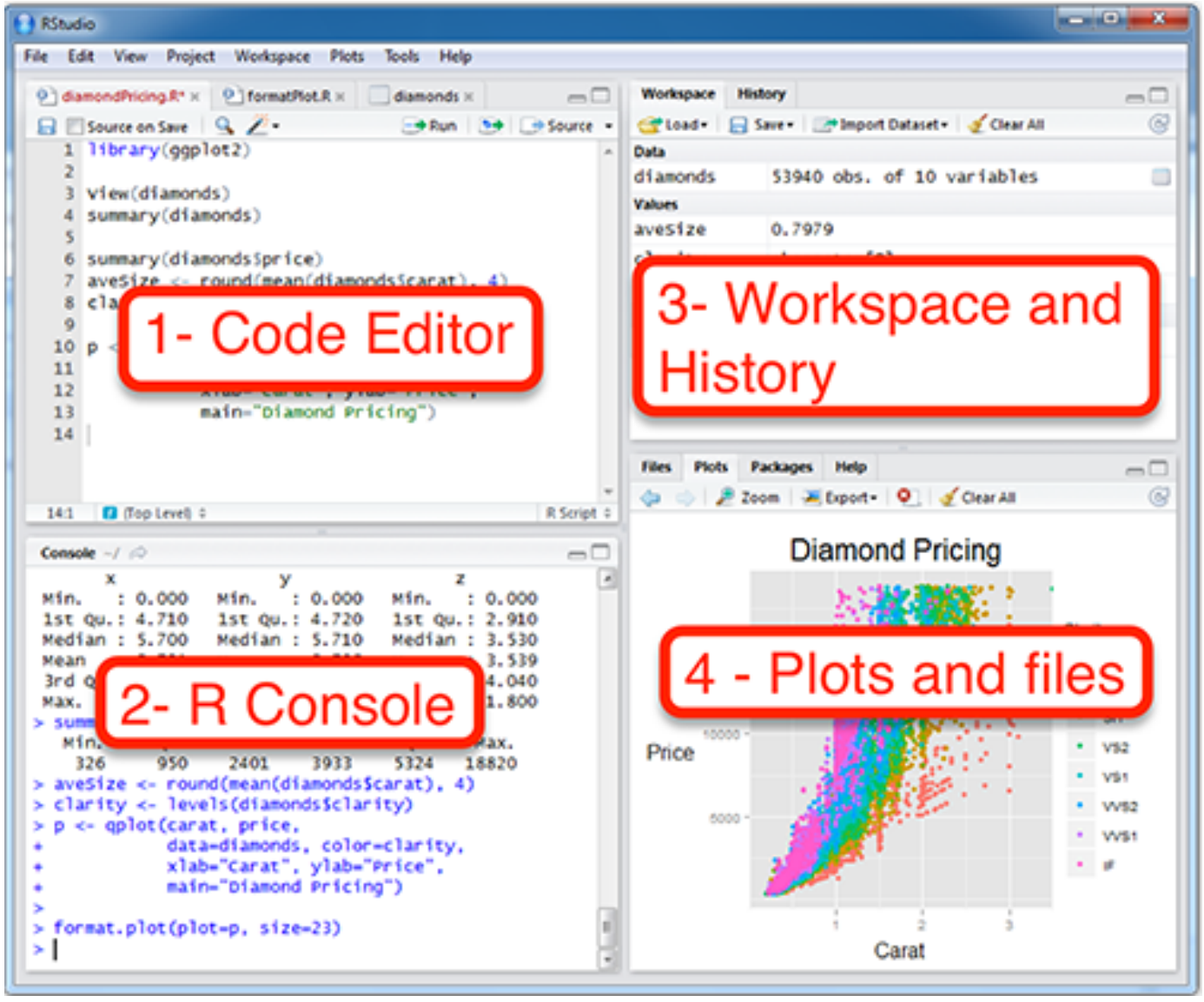


Instalando o R

Instalando o RStudio

RStudio é um ambiente de desenvolvimento integrado (IDE) para R. Inclui um console, editor de destaque de sintaxe que oferece suporte à execução direta de código, bem como ferramentas para plotagem, histórico, depuração e gerenciamento de espaço de trabalho.

Instalando o R



Sugestão de Estrutura de Diretórios

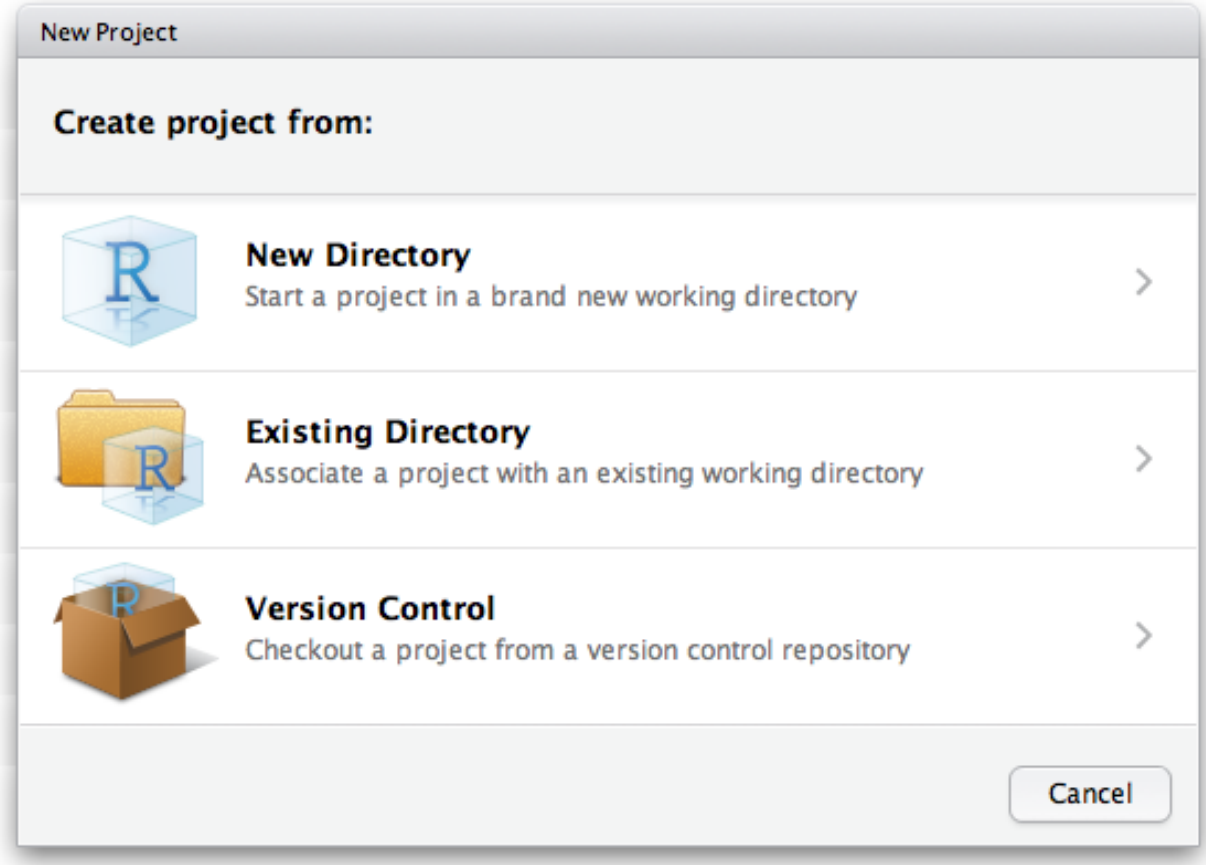
1. Um bom projeto O R permite trabalhas com qualquer estrutura de diretorios.

Projetos no RStudio

- Os projetos RStudio facilitam a divisão do seu trabalho em vários contextos, cada projeto tem o seu próprio diretório de trabalho, área de trabalho, histórico e documentos de origem.
- Os projetos RStudio são associados a diretórios de trabalho R. Você pode criar um projeto RStudio: Em um novo diretório; Em um diretório existente onde você já tem o código R e dados; Ao clonar um repositório de controle de versão (Git ou Subversion)

Projetos no RStudio

- Para criar um novo projeto use o comando Criar Projeto (disponível no menu Projetos e na barra de ferramentas global)



Projetos no RStudio

Quando um novo projeto é criado RStudio:

1. Cria um arquivo de projeto (com uma extensão .Rproj) dentro do diretório do projeto. Este arquivo contém várias opções de projeto (discutidas abaixo) e também pode ser usado como um atalho para abrir o projeto diretamente do sistema de arquivos.
2. Cria um diretório oculto (denominado .Rproj.user) onde os arquivos temporários específicos do projeto (por exemplo, documentos de origem salvos automaticamente, estado da janela, etc.) são armazenados. Este diretório também é adicionado automaticamente a .Rbuildignore, .gitignore, etc. se necessário.
3. Carrega o projeto no RStudio e exibe seu nome na barra de ferramentas Projetos (que está localizada no lado direito da barra de ferramentas principal)

Pacotes

Instalando o primeiro pacote

Você também precisará instalar alguns pacotes R.

1. Um pacote R é uma coleção de funções, dados e documentação que estende os recursos da base R.
2. O uso de pacotes é a chave para o uso bem-sucedido de R. A maioria dos pacotes que abordaremos neste curso fazem parte do chamado *tidyverse*.
3. Os pacotes do tidyverse compartilham uma filosofia comum de programação R de dados e são projetados para trabalhar juntos naturalmente.
4. Você pode instalar o tidyverse completo com uma única linha de código no console: install.packages("tidyverse")

Console vs. Editor

Até o momento apenas apresentamos os ambiente de "Console" e "Editor". Porém existe uma diferença grande entre eles.

- No ambiente "console", as instruções são executadas logo após a tecla "enter" ser executada.
- No ambiente "editor" o código é um conjunto de instruções que só é executado quando requisitado. Logo, voce pode digitar todo o codigo e roda-lo de uma só vez.
- No ambiente "editor" também é póssivel executar o código linha a linha. Isso é uma funcionalidade do RStudio, e o seu atalho é "CTRL-ENTER"

Definição

Durante todo o curso iremos trabalhar no ambiente Editor. Criando um script e trabalhando no script.

Visualização de dados

- Este capítulo irá ensiná-lo a visualizar seus dados usando ggplot2.
- O R possui vários sistemas para fazer gráficos[1], mas ggplot2 é um dos mais elegantes e versáteis. O ggplot2 implementa a gramática dos gráficos, um sistema coerente para descrever e construir gráficos.

[1] A função nativa do R para fazer gráficos é a *plot()*

Visualização de dados

Carregando os pacotes

Este capítulo enfoca ggplot2, um dos principais membros do tidyverse. Para acessar os conjuntos de dados, páginas de ajuda e funções que usaremos neste capítulo, carregue o tidyverse executando este código:

```
> library(tidyverse)
#> Loading tidyverse: ggplot2
#> Loading tidyverse: tibble
#> Loading tidyverse: tidyr
#> Loading tidyverse: readr
#> Loading tidyverse: purrr
#> Loading tidyverse: dplyr
#> Conflicts with tidy packages ----------------
#> filter(): dplyr, stats
#> lag(): dplyr, stats
```

Visualização de dados

Carregando os pacotes

```
> library(tidyverse)
```

Essa linha de código carrega o tidyverse, contendo os pacotes que você usará em quase todas as análises de dados.

```
#> Conflicts with tidy packages ------------------
#> filter(): dplyr, stats
#> lag(): dplyr, stats
```

Note que o R também informa quais funções do tidyverse entram em conflito com funções na base R (ou de outros pacotes que você possa ter carregado).

Visualização de dados

Carregando os pacotes

- **Você só precisa instalar um pacote uma única vez!**
- **Mas precisa recarregá-lo sempre que iniciar uma nova sessão.**
- Isso vale para qualquer pacote!

Banco de dados mpg

- Vamos usar nosso primeiro gráfico para responder a uma pergunta: os carros com motores grandes usam mais combustível do que carros com motores pequenos?
- Como é a relação entre tamanho do motor e eficiência de combustível? É positivo? Negativo? Linear? Não linear?

Banco de dados mpg

- Você pode testar sua resposta com o banco de dados mpg encontrado em ggplot2 (também conhecido como ggplot2::mpg).
- Um banco de dados é uma coleção retangular de variáveis (nas colunas) e observações (nas linhas). mpg contém observações coletadas pela Agência de Proteção Ambiental dos EUA em 38 modelos de carros:

```
> mpg
#> # A tibble: 234 x 11
#>   manufacturer  model  displ  year   cyl    trans      drv
#>          <chr>  <chr>  <dbl>  <int>  <int>  <chr>      <chr>
#> 1         audi     a4    1.8   1999      4   auto(l5)  f
```

Entre as variáveis em mpg estão:

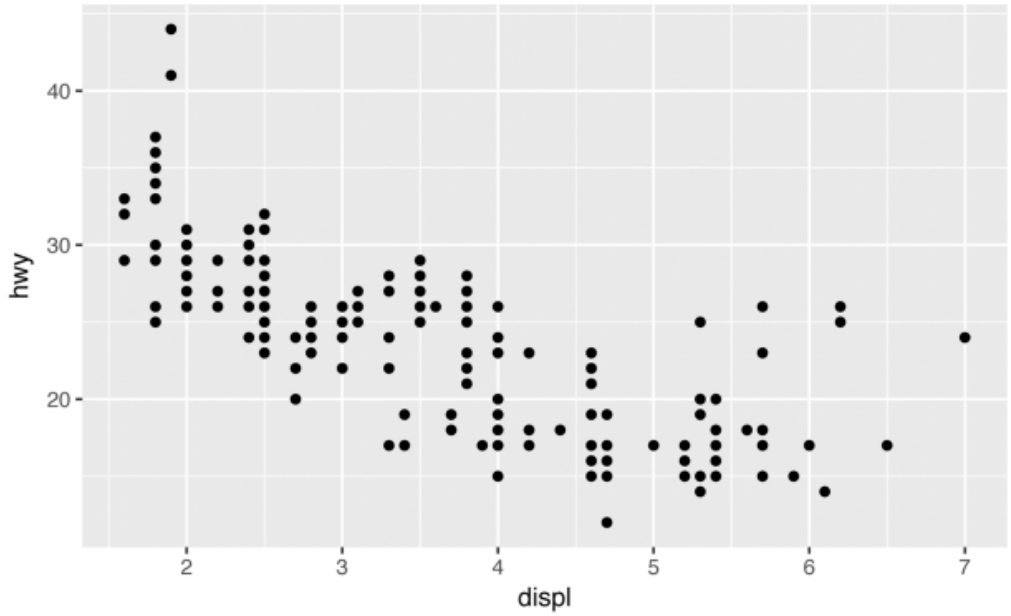
- displ, o tamanho do motor de um carro, em litros.
- hwy, a eficiência de combustível de um carro na rodovia, em milhas por galão (mpg).

ggplot2

Criando um gráfico de dispersão

Para plotar mpg, execute este código para colocar displ no eixo x e hwy no eixo y:

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



ggplot2

- Com ggplot2, você começa um gráfico com a função ggplot(). ggplot() cria um sistema de coordenadas ao qual você pode adicionar camadas.
- O primeiro argumento de ggplot() é o conjunto de dados a ser usado no gráfico.
- Você completa seu gráfico adicionando uma ou mais camadas ao ggplot(). A função geom_point() adiciona uma camada de pontos ao seu gráfico, que cria um gráfico de dispersão.
- O ggplot2 vem com muitas funções geom, cada uma delas adicionando um tipo diferente de camada a um gráfico.

Estética e mapeamento

ggplot2

- Toda função geom em ggplot2 recebe um argumento de mapeamento. O argumento de mapeamento é sempre pareado com aes(), e os argumentos x e y de aes() especificam quais variáveis são mapeadas para os eixos x e y.
- ggplot2 procura as variáveis utilizadas no mapeamento como o nome das colunas no banco de dados, neste caso, mpg.

Exercícios

- Execute o comando abaixo, O que você vê?

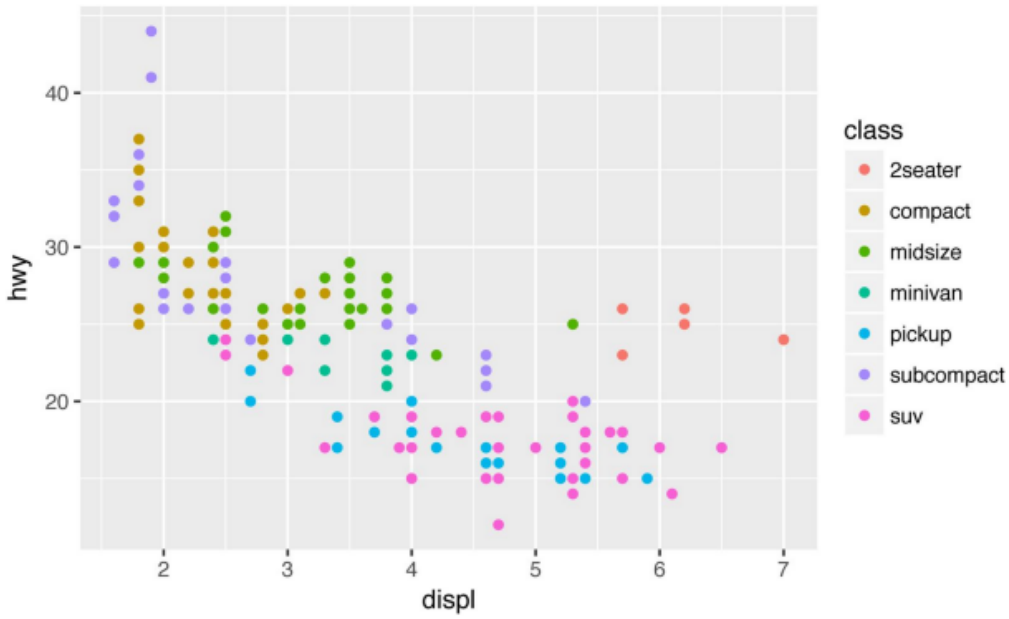
```
> ggplot(data = mpg).
```

- Quantas linhas existem em mtcars? Quantas colunas?
- Faça um gráfico de dispersão de hwy versus cyl.
- O que acontece se você fizer um gráfico de dispersão de classe versus drv? Por que esse gráfico não é útil?

Estética com mais informações

- Você pode transmitir informações sobre seus dados mapeando a estética em seu gráfico para as variáveis em seu banco de dados.
- Por exemplo, você pode mapear as cores de seus pontos para a variável de classe para revelar a classe de cada carro

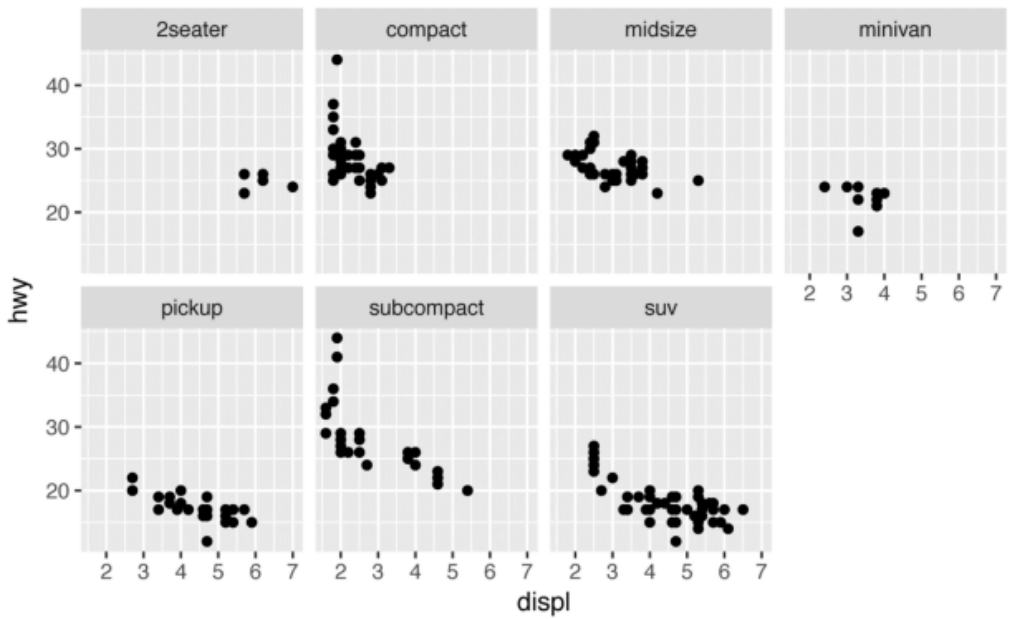
```
ggplot(data = mpg) + geom_point( mapping = aes( x = displ, y = hwy, color = class))
```



Facets

- Uma maneira de adicionar variáveis adicionais é com a estética. Outra maneira, particularmente útil para variáveis categóricas, é dividir seu gráfico em facetas (*Facets*), subparcelas em que cada uma exibe um subconjunto dos dados.

```
ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy)) + facet_wrap( ~ class, nrow = 2)
```



Facets

- Para facetar seu gráfico por uma única variável, use facet_wrap().
- O primeiro argumento de facet_wrap() deve ser uma fórmula, que você cria com ~ seguido por um nome de variável (aqui "fórmula" é o nome de uma estrutura de dados em R, não um sinônimo para "equação"). A variável que você passa para facet_wrap() deve ser discreta

Grafico de dispersão

- Para fazermos graficos de dispersão utilizamos o geom_point. Esse tipo de gráfico é recomendado para se explorar a relação entre duas variaveis.

```
ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy)) + facet_wrap( ~ class, nrow = 2)
```

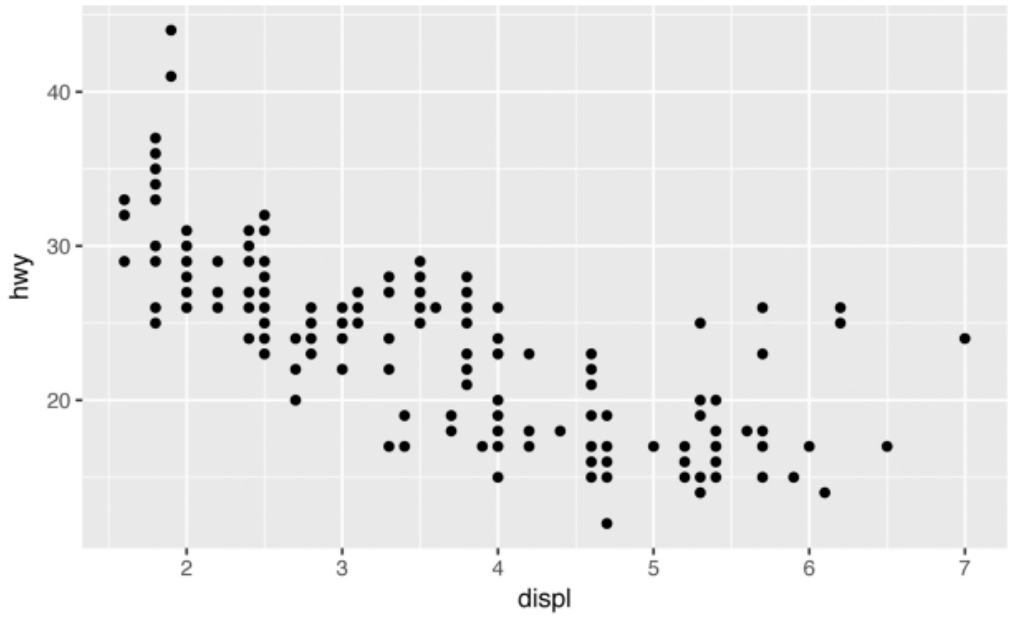


Gráfico de linha

- Para fazermos graficos de linha utilizamos o geom_line. Esse tipo de gráfico é normalmente utlizado em séries temporaris.

```
ggplot(data = economics) + geom_line(mapping = aes(x = date, y = uempmed))
```

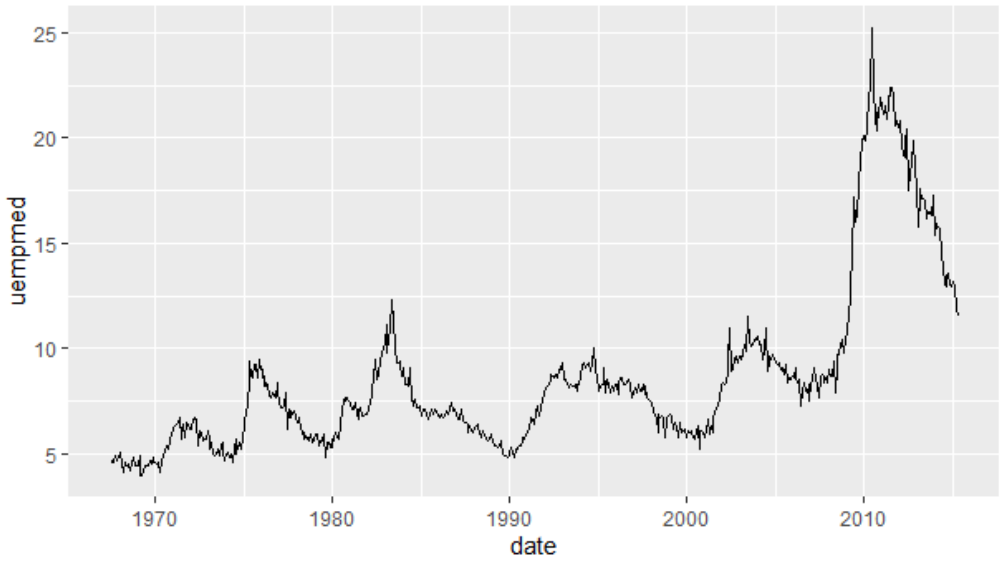


Gráfico de barras

- Para fazermos graficos de barras utilizamos o geom_bar. Esse tipo de gráfico só precisa da informação (estética) de um eixo.

```
ggplot(data = diamonds) + geom_bar(mapping = aes(x = cut))
```

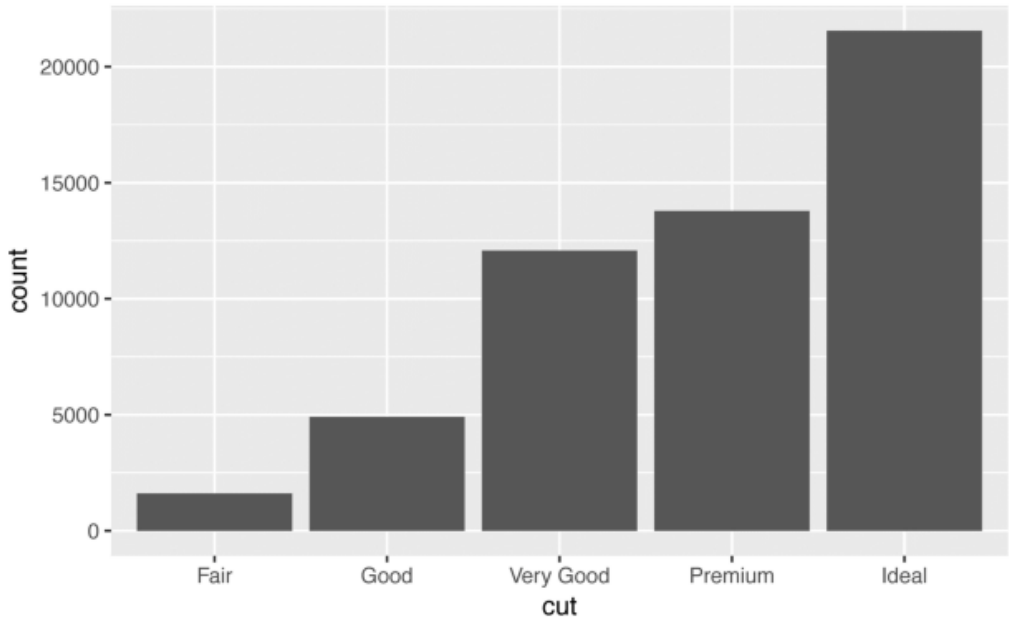
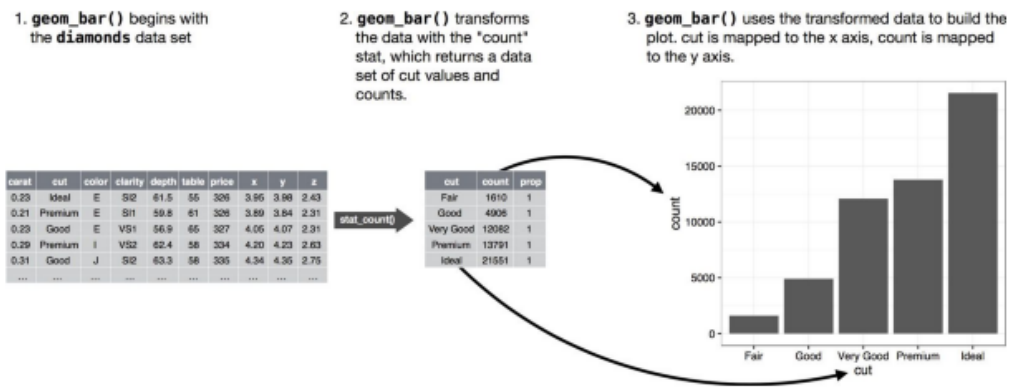


Gráfico de barras

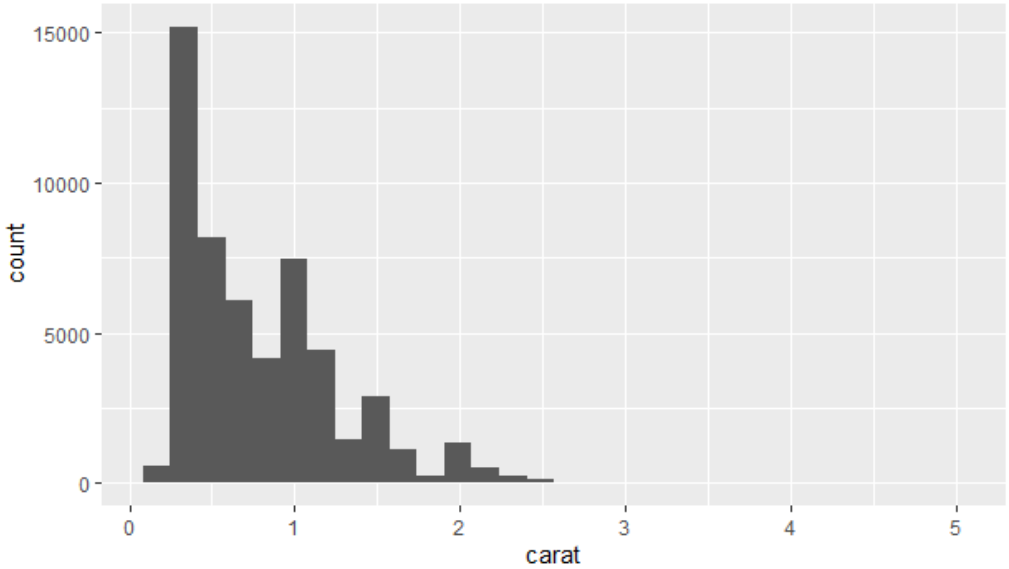
- Muitos gráficos, como diagramas de dispersão, representam os valores brutos de seu conjunto de dados. Outros gráficos, como gráficos de barras, calculam novos valores para plotar.
- O algoritmo usado para calcular novos valores para um gráfico é chamado de stat, abreviação de transformação estatística. A figura a seguir descreve como esse processo funciona com geom_bar().



histograma

- Para fazermos histogramas (que são bem semelhantes a gráficos de barras) utilizamos o geom_histogram. Esse tipo de gráfico só precisa da informação (estética) de um eixo e de uma variavel contínua.

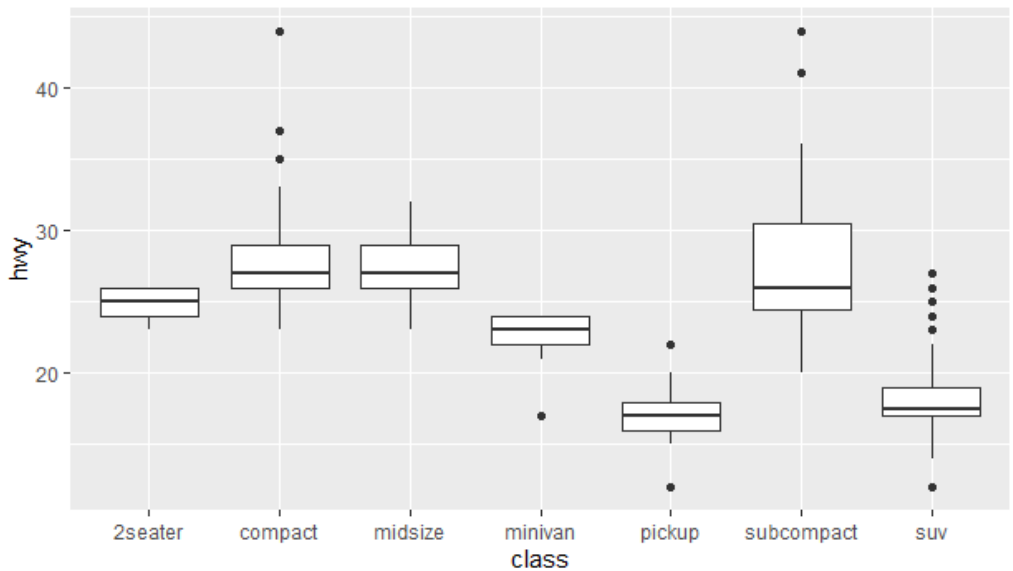
```
ggplot(data = diamonds) + geom_histogram(mapping = aes(x = carat))
```



Boxplot

- Para fazermos boxplot utilizamos o geom_boxplot. Esse tipo de gráfico usualmente tem como estetica uma variável continua e uma variável discreta.

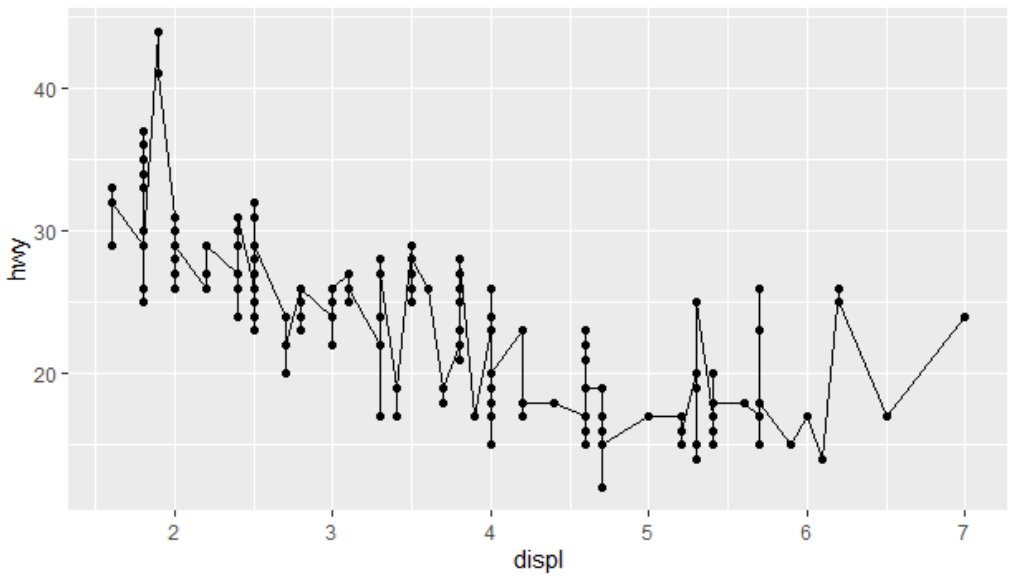
```
ggplot(data = mpg) + geom_boxplot(mapping = aes(x = class, y = hwy))
```



Dois gráficos juntos

- Para fazermos dois gráficos juntos, basta "adicionarmos" os gráficos. por exemplo vamos fazer os graficos de linha e de dispersão juntos.

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

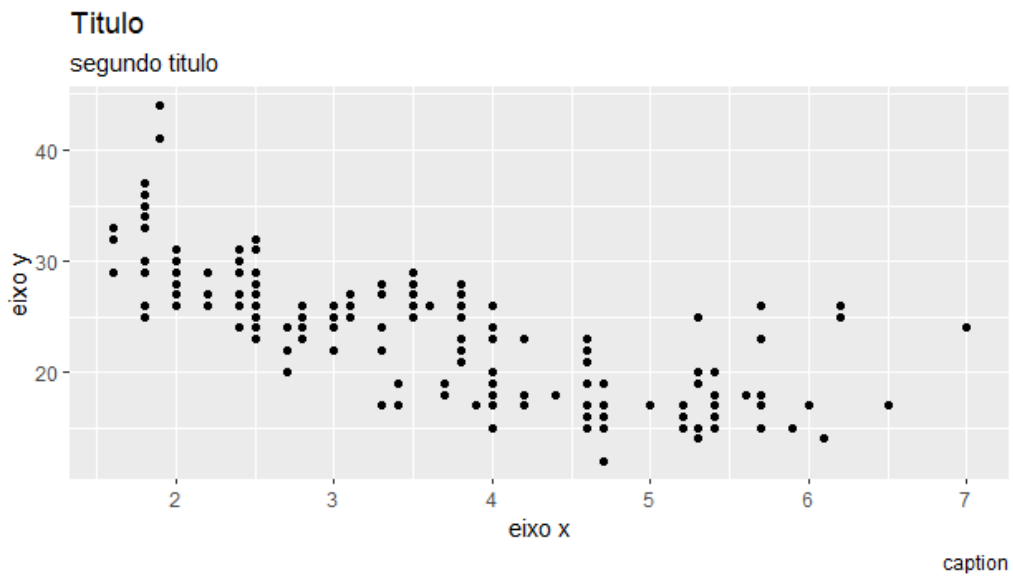


labs

Titulos e configurações

- Para adicionarmos elementos como tiltuo, nome dos eixos, etc, utilizamos a função labs().

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  labs(title="Titulo", subtitle = "segundo titulo",
       x= "eixo x", y="eixo y",caption = "caption")
```



Exercícios

- utilizando o banco de dados mpg, faça um grafico de barras da variavel fl.
- utilizando o banco de dados mpg, faça um histograma da variavel displ.
- utilizando o banco de dados mpg, faça um boxplot da variavel displ segregada por fl.

Exercícios

- utilizando o banco de dados diamonds, faça um histograma carat, separado em facetas da variavel color.
- utilizando o banco de dados diamonds, faça um grafico de barras da variavel color, e complemente a estetica indicando que o preenchimento das barras deve ser feito de acordom com a variavel color (“aes(x = color, fill=color)”).
- utilizando o banco de dados diamonds, faça um gráfico de dispersão, de “carat” por “price”, e colorido pela variável “color”.

Salvando o gráfico

- Para salvar os gráficos vamos utilizar o comando ggsave().
- Caso você não informe o gráfico a ser salvo ele salva o ultimo gráfico gerado.

```
ggsave(filename = "meu arquivo.png")
```

- O comando ggsave() usa as configurações de tamanho da área de gráficos. O que muda constantemente. Para fixar o tamanho da imagem você deve informar os parametors de qualidade e tamanho.

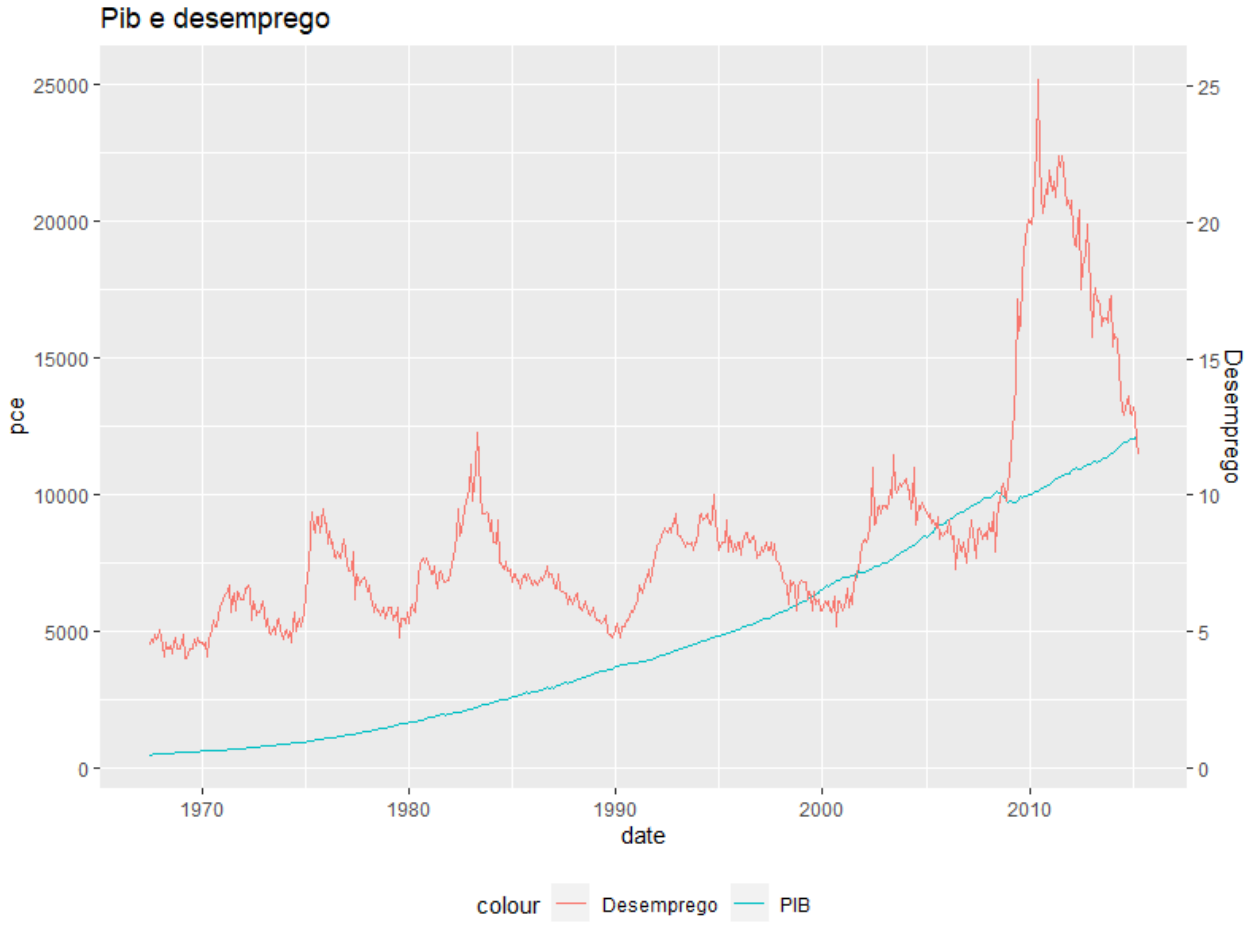
```
ggsave(filename = "meu arquivo.png",
                units = "in",
                width = 8, height = 6,
                dpi = 100)
```

Eixo secundário

Para fazer um eixo secundário, é necessário ajustar a escala do eixo e dos dados.

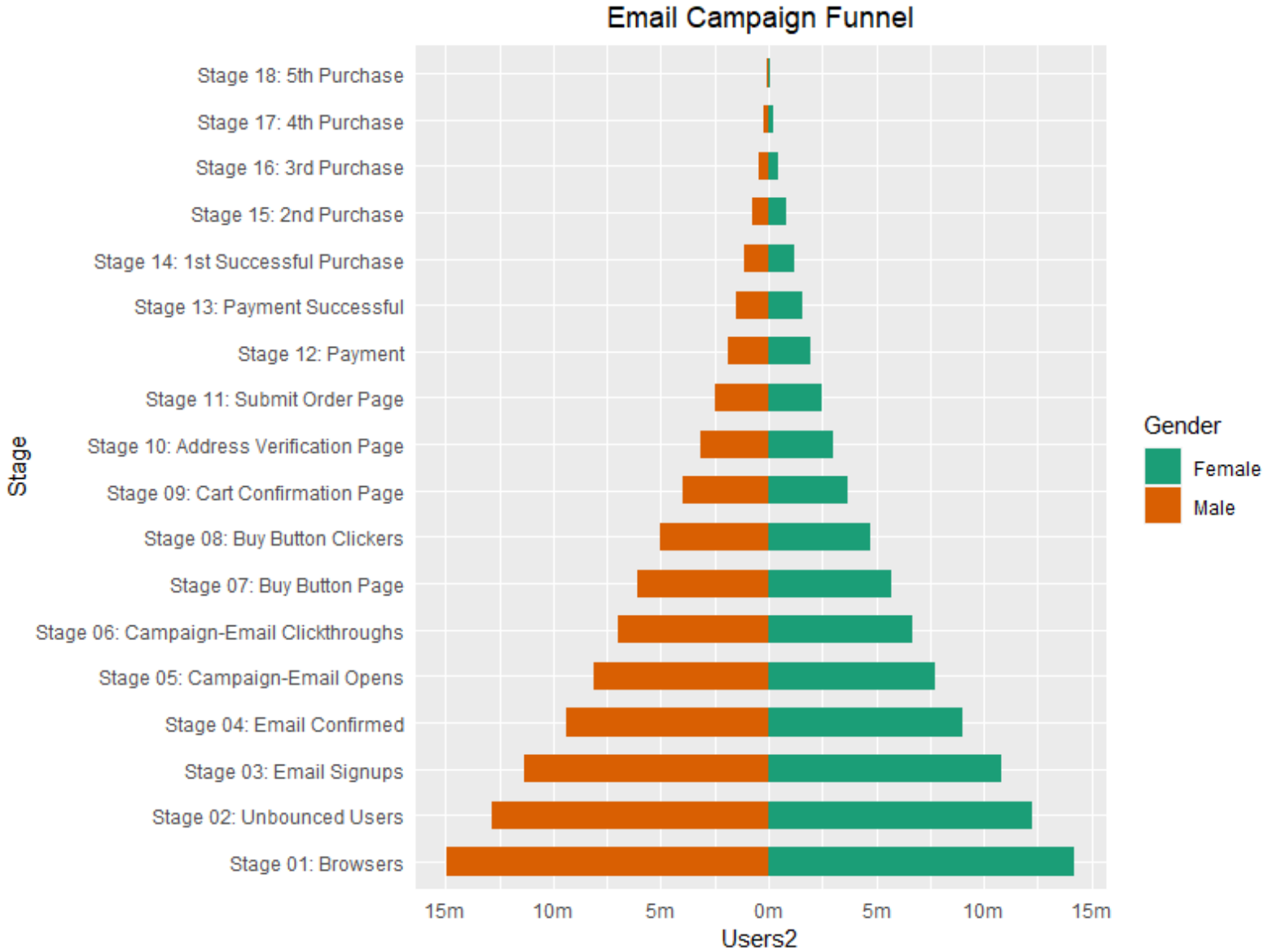
```
data(economics)

ggplot(economics) +
  geom_line(aes(x=date, y=pce, colour= "PIB")) +
  geom_line(aes(x=date, y=uempmed*1000, colour="Desemprego")) +
  scale_y_continuous(sec.axis = sec_axis(~ . / 1000, name = "Desemprego") ) +
  labs(title = "Pib e desemprego") + theme(legend.position="bottom")
```

Eixo secundário



Funil

- Uma variação do gráfico de barras é o gráfico em funil.

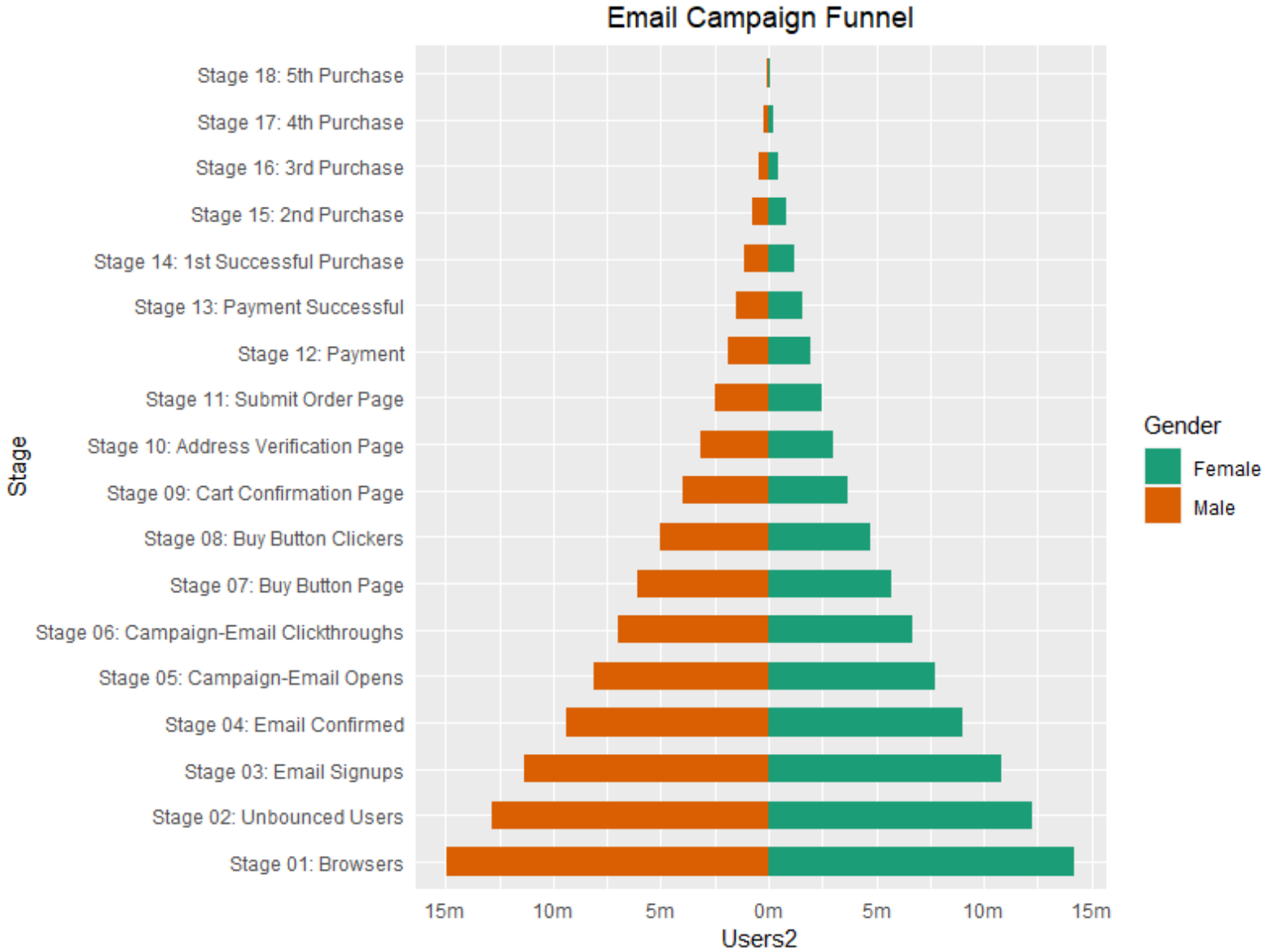


Funil

```
tbl = readRDS(file = "./databases/email_campaign_funnel.rds")

# troca o valor dos Males para negativo
tbl$Users[tbl$Gender == "Male"] = -tbl$Users[tbl$Gender == "Male"]

ggplot(data = tbl) +
  geom_bar(mapping = aes(x = Stage, y = Users, fill = Gender), stat = "identity") +
  coord_flip() +
  labs(title = "Email Campain Funnel") +
  scale_y_continuous(breaks = c(-15000000, -10000000, -5000000,
                                0,
                                15000000, 10000000, 5000000),
                     labels = c("15m", "10m", "5m", "0", "15m", "10m", "5m"))
```

Funil

- Uma variação do gráfico de barras é o gráfico em funil.



Funil

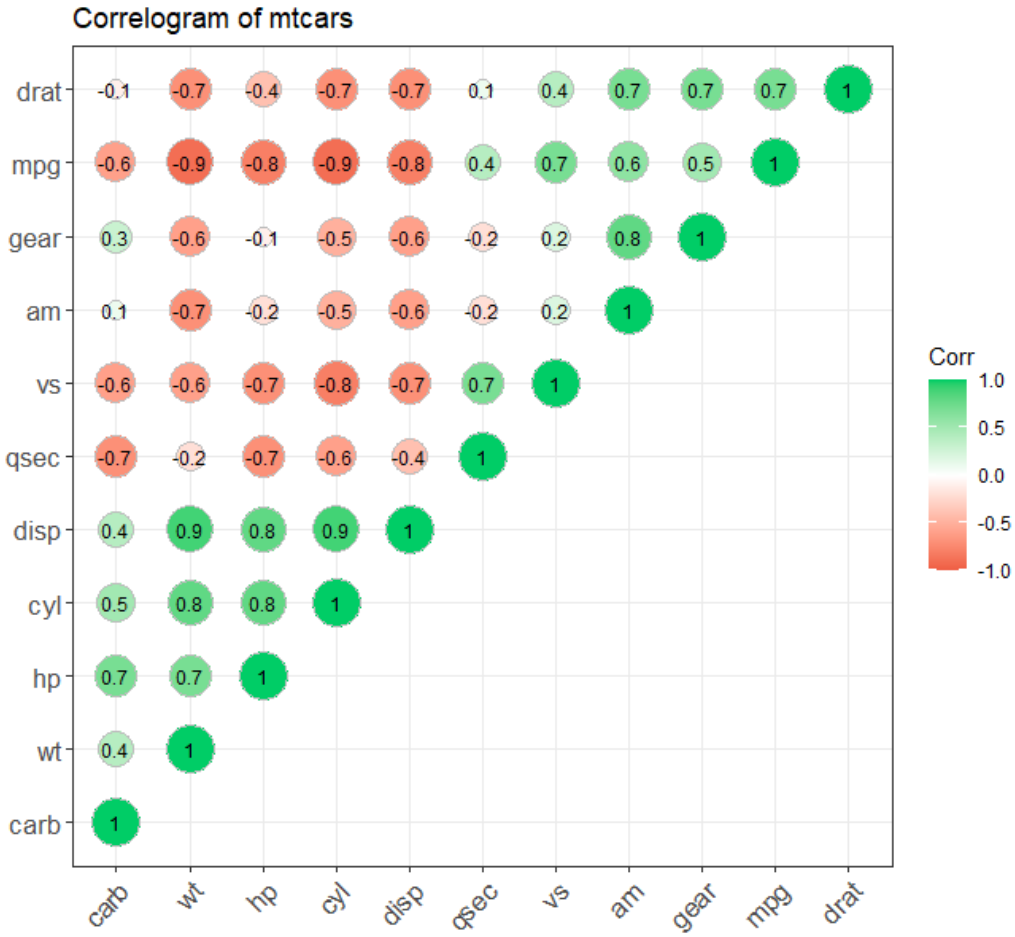
```
tbl = readRDS(file = "./databases/email_campaign_funnel.rds")

# troca o valor dos Males para negativo
tbl$Users[tbl$Gender == "Male"] = -tbl$Users[tbl$Gender == "Male"]

ggplot(data = tbl) +
  geom_col(mapping = aes(y = Stage, x = Users, fill = Gender)) +
  labs(title = "Email Campain Funnel") +
  scale_x_continuous(breaks = c(-15000000, -10000000, -5000000,
                                0,
                                15000000, 10000000, 5000000),
                     labels = c("15m", "10m", "5m", "0", "15m", "10m", "5m"))
```

Correlograma

- Você pode faezer um grafico de correlograma com o pacote **"ggcorrplot"**

```
library(ggplot2)
library(ggcorrplot)

# Correlation matrix
data(mtcars)
corr <- round(cor(mtcars), 1)

# Plot
ggcorrplot(corr,
           hc.order = TRUE,
           type = "upper",
           lab = TRUE,
           lab_size = 3,
           show.diag = TRUE,
           method="circle",
           colors = c("tomato2", "white", "springgreen3"),
           title="Correlogram of mtcars",
           ggtheme=theme_bw)
```

Correlograma



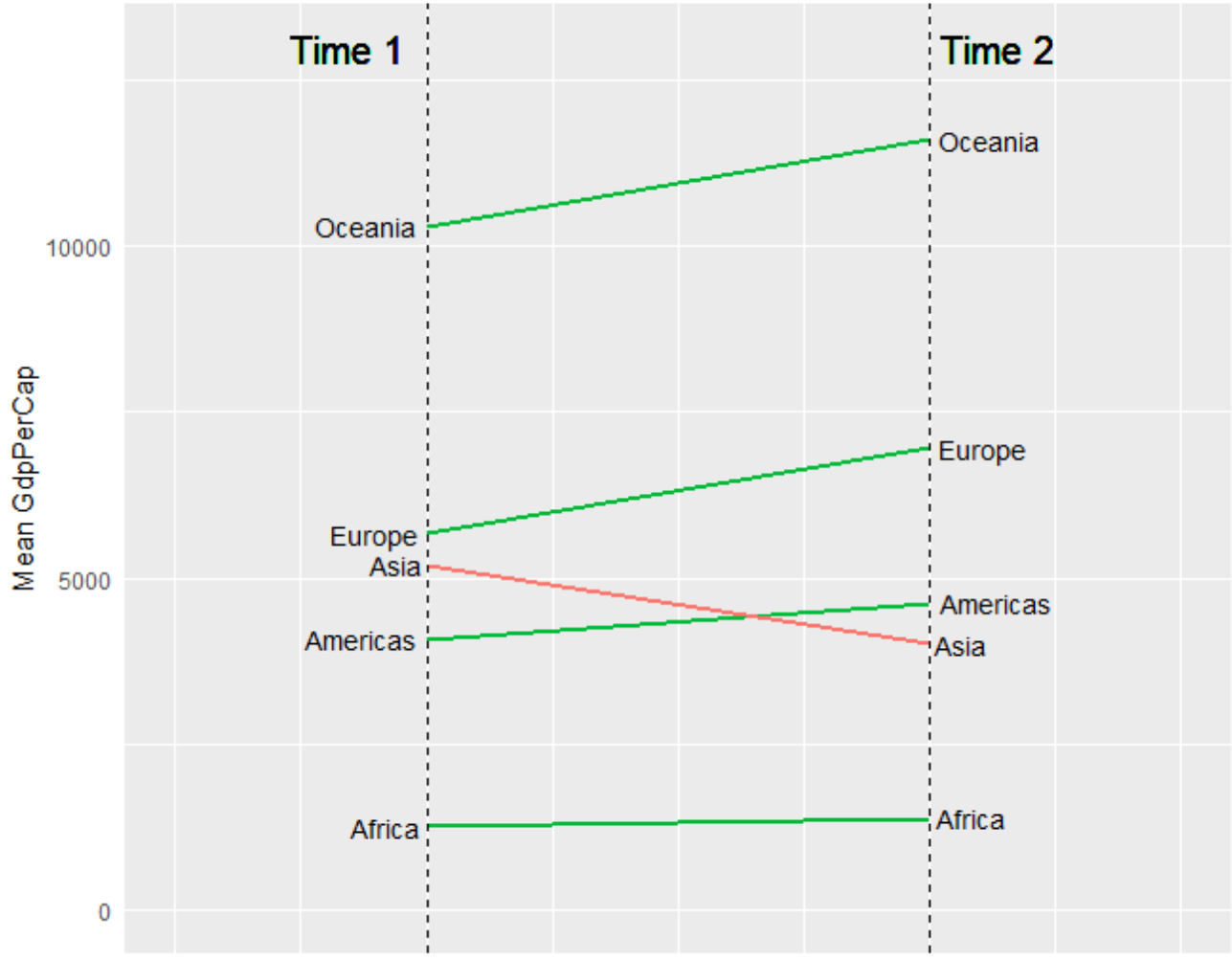
Slope Chart

- Vamos Criar um grafico de variação temporal. Para isso vamos utilizar o banco de dados “SlopePlot.rds”

```
> df <- readRDS(file = "./databases/SlopePlot.rds")
> df
  continent     Y1952     Y1957 class
1    Africa  1252.572  1385.236    Up
2  Americas  4079.063  4616.044    Up
3      Asia  5195.484  4003.133  Down
4    Europe  5661.057  6963.013    Up
5   Oceania 10298.086 11598.522    Up
```

Slope Chart

- Nosso objetivo é criar o gráfico a seguir



Slope Chart

```
ggplot(df) +
  geom_segment(aes(x=1, xend=2, y=Y1952, yend=Y1957, col=class), size=.75, show.legend=F) +
  geom_vline(xintercept=1, linetype="dashed", size=.1) +
  geom_vline(xintercept=2, linetype="dashed", size=.1) +
  scale_color_manual(labels = c("Subiu", "Desceu"),
                     values = c("Up"="#00ba38", "Down"="#f8766d")) +  # color of lines
  labs(x="", y="Mean GdpPerCap") +  # Axis labels
  xlim(.5, 2.5) + ylim(0,(13000)) +  # X and Y axis limits
  geom_text(mapping = aes(label=continent, y=Y1952, x=rep(1, nrow(df))), hjust=1.1, size=3.5) +
  geom_text(mapping = aes(label=continent, y=Y1957, x=rep(2, nrow(df))), hjust=-0.1, size=3.5) +
  geom_text(mapping = aes(label="Time 1", x=1, y=13000), hjust=1.2, size=5) + # title
  geom_text(mapping = aes(label="Time 2", x=2, y=13000), hjust=-0.1, size=5) +
  theme(axis.ticks = element_blank(),
        axis.text.x = element_blank(),
        plot.margin = unit(c(1,2,1,2), "cm"))
```

Table of Contents

Noções básicas de codificação

Tipos de dados básicos e estruturas de dados

Para tirar o melhor proveito da linguagem R, você precisará de um forte conhecimento dos tipos e estruturas de dados básicos e de como operar neles.

É muito importante entender as estruturas de dados porque esses são os objetos que você manipulará diariamente em R. Lidar com conversões de objetos é uma das fontes mais comuns de frustração para iniciantes.

Noções básicas de codificação

R tem 6 tipos de **dados** básicos.

1. character (letras)
2. numeric (Número real)
3. integer (Número inteiro)
4. logical (Número binário)
5. complex (Número Complexo)
6. raw (bytes)

Noções básicas de codificação

Os elementos desses tipos de dados podem ser combinados para formar estruturas de dados, como vetores atômicos. Quando chamamos um vetor de atômico, queremos dizer que o vetor contém apenas dados de um único tipo de dados. Abaixo estão exemplos de vetores de caracteres atômicos, vetores numéricos, vetores de inteiros, etc.

1. character: “a”, “swc”
2. numeric: 2, 15.5
3. integer: 2L (O “L” diz ao R para guardar o número como inteiro)
4. logical: TRUE, FALSE
5. complex: 1+4i

Noções básicas de codificação

O R fornece muitas funções para examinar características de vetores e outros objetos, por exemplo

- class() - que tipo de objeto é (alto nível)?
- typeof() - qual é o tipo de dados do objeto (baixo nível)?
- length() - qual o tamanho? E quanto a objetos bidimensionais?
- attributes() - tem algum metadado?

Noções básicas de codificação

Example

```
# Example
x <- "dataset"
typeof(x)
# [1] "character"
y <- 1:10
y
# [1]  1  2  3  4  5  6  7  8  9 10
typeof(y)
[1] "integer"
length(y)
# [1] 10
z <- as.numeric(y)
z
# [1]  1  2  3  4  5  6  7  8  9 10
typeof(z)
# [1] "double"
```

Noções básicas de codificação

Os **dados** são agrupados em um conjunto de **estruturas de dados**. O R tem diversas estruturas de dados. Vamos abordar as estruturas mais comuns:

- atomic vector
- list
- matrix
- data frame
- factors

Vetores

Um vetor é a estrutura de dados mais comum e básica em R e é praticamente a estrutura padrão do R. Um vetor é uma coleção de elementos que são mais comumente de caractere de modo lógico, inteiro ou numérico.

A função c() (para combinar) pode ser usada para criar ou adicionar elementos a um vetor.

```
> x <- c(1, 2, 3)
> x
[1] 1 2 3
> y <- c(x, 10, 11, 12)
> y
[1]  1  2  3 10 11 12
> class(x)
[1] "numeric"
> class(y)
[1] "numeric"
> length(y)
[1] 6
```

Dados especiais

- **Dados ausentes:** R suporta dados ausentes em vetores. Eles são representados como NA (não disponível) e podem ser usados para todos os tipos de vetores.
- **Outros Valores Especiais:** O Inf é infinito. Você pode ter infinito positivo ou negativo

```
> x <- c(1, 2 , NA, 4, 5, Inf)
> x
[1]   1   2  NA   4   5 Inf
> class(x)
[1] "numeric"
> length(x)
[1] 6
```

Matrizes

Em R, as matrizes são uma extensão dos vetores numéricos ou de caracteres. Eles não são um tipo separado de objeto, mas simplesmente um vetor atômico com dimensões; o número de linhas e colunas. Tal como acontece com os vetores atômicos, **os elementos de uma matriz devem ser do mesmo tipo de dados**.

```
> m <- matrix(nrow = 2, ncol = 2)
> m
     [,1] [,2]
[1,]   NA   NA
[2,]   NA   NA
> class(m)
[1] "matrix" "array"
> m <- matrix(c(1:3))
> m
     [,1]
[1,]    1
[2,]    2
[3,]    3
> class(m)
[1] "matrix" "array"
```

Lista

- No R, as listas agem como contêineres. Ao contrário dos vetores atômicos, **o conteúdo de uma lista não se restringe a um único modo e pode abranger qualquer mistura de tipos de dados**.
- As listas às vezes são chamadas de vetores genéricos, porque os elementos de uma lista podem ser de qualquer tipo de objeto R, até mesmo listas contendo outras listas.
- Crie listas usando list().

```
> x <- list(1, "a", TRUE, 1+4i)
> x
[[1]]
[1] 1

[[2]]
[1] "a"

[[3]]
[1] TRUE

[[4]]
[1] 1+4i
```

Data Frame

- Um data.frame é uma estrutura de dados muito importante em R. É basicamente uma tabela. Cada coluna contem um tipo de dado especifico, sendo que cada linha guarda o valor do dado em si.
- Um data.frame é um tipo especial de lista em que cada elemento da lista tem o mesmo comprimento (ou seja, o quadro de dados é uma lista "retangular").

```
> dat <- data.frame(id = letters[1:10], x = 1:10, y = 11:20)
> dat
   id  x  y
1   a  1 11
2   b  2 12
3   c  3 13
4   d  4 14
5   e  5 15
6   f  6 16
7   g  7 17
8   h  8 18
9   i  9 19
10  j 10 20
```

Data Frame

Algumas informações adicionais sobre frames de dados:

- São usualmente criados pelos métodos de importação de dados.
- Assumindo que todas as colunas em um data.frame são do mesmo tipo, o data.frame pode ser convertido em uma matriz com data.matrix() (preferencial) ou as.matrix().
- O número de linhas e colunas pode ser obtido com nrow(dat) e ncol(dat), respectivamente.

Data Frame

Navegando no data.frame

Como os quadros de dados são retangulares, os elementos do quadro de dados podem ser referenciados especificando-se a linha e o índice da coluna em colchetes simples (semelhante à matriz).

```
> dat <- data.frame(id = letters[1:10], x = 1:10, y = 11:20)
> dat
   id  x  y
1   a  1 11
2   b  2 12
3   c  3 13
4   d  4 14
5   e  5 15
6   f  6 16
7   g  7 17
8   h  8 18
9   i  9 19
10  j 10 20
> dat[1, 3]  # linha 1; coluna 3 do data frame
[1] 11
> dat[4, "id"]  # linha 1; coluna "id" do data frame
```

Data Frame

Tibble

Existem alguns data.frames mais "poderosos", entre eles vamos apresentar a tibble.

Tibble: Tibbles são data.frames que são mais cuidadosos ao tratar os dados, forçando você a analisar os dados, levando a um código mais limpo e expressivo. Tibbles também tem um método print() aprimorado que os torna mais fáceis de usar com grandes conjuntos de dados contendo objetos complexos.

```
> library(tibble)
> dat <- tibble(id = letters[1:5], x = 1:5, y = 11:15)
> dat
# A tibble: 10 x 3
   id        x     y
   <chr> <int> <int>
 1 a         1    11
 2 b         2    12
 3 c         3    13
 4 d         4    14
 5 e         5    15
```

Data Frame

Tibble

Uma das grandes vantagens do data.frame é pode “navegar” os seus dados de forma vetorizada. Isso é feito por meio de referencia à coluna desejada e indicando a linha desejada.

Essa sintaxe de navegação consiste em ”DataFrame$Coluna[linha]”. Abaixomostramos um exemplo.

```
> dat = data.frame(id = 1:10,  letra=letters[1:10])
> dat$letra[4]
[1] "d"
```

Noções básicas de codificação

Vamos revisar alguns princípios básicos que omitimos até agora. Você pode usar R como uma calculadora:

```
1 / 200 * 30
#> [1] 0.15

(59 + 73 + 2) / 3
#> [1] 44.7

sin(pi / 2)
#> [1]
```

Você pode criar novos variáveis (objetos) com "<-"

```
x <- 3 * 4
```

Todas as instruções R onde você cria objetos, instruções de atribuição, têm a mesma forma: "objeto_nome <- valor" Ao ler o código, diga "o nome do objeto obtém valor" em sua cabeça.

Noções básicas de codificação

- Você fará várias atribuições e <- é muito trabalhoso de digitar.
- Você pode usar " = " e irá funcionar.
- Você também pode usar o atalho de teclado do RStudio: Alt-– (o sinal de menos).

Operadores R

- O R tem muitos operadores para realizar diferentes operações matemáticas e lógicas.
- Os operadores em R podem ser classificados principalmente nas seguintes categorias.
  - Operadores aritméticos
  - Operadores relacionais
  - Operadores lógicos
  - Operadores de atribuição

Operadores aritméticos em R

Operações básicas

Esses operadores são usados para realizar operações matemáticas como adição e multiplicação. Aqui está uma lista de operadores aritméticos disponíveis em R.

Operador	**Descrição do operador**
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
^	Expoente
%%	Modulus (Restante da divisão)
%/%	Divisão Inteira

Operadores aritméticos em R

Operações básicas

Um exemplo de execução

```
> x <- 5
> y <- 16
> x+y
[1] 21
> x-y
[1] -11
> x*y
[1] 80
> y/x
[1] 3.2
> y%/%x
[1] 1
> y%%x
[1] 1
> y^x
[1] 1048576
```

Operadores relacionais

Operações básicas

Operadores relacionais são usados para comparar valores. Abaixo está uma lista de operadores relacionais disponíveis no R.

Operador	**Descrição do operador**
$<$	Menor que
$>$	Maior que
$<=$	Menor ou igual a
$>=$	Maior ou igual a
$==$	Igual a
$!=$	Diferente de

Uma operação relacional retorna uma variavel lógica (boleana) do dizendo se a comparação é verdadeira ou falsa.

Operadores relacionais

Operações básicas

Um exemplo de execução

```
> x <- 5
> y <- 16
> x>y
[1] FALSE
> x<=y
[1] TRUE
> x == y
[1] FALSE
> y>y
[1] FALSE
> y>=y
[1] TRUE
> x!=y
[1] TRUE
```

Operadores lógicos

Operações básicas

Operadores lógicos são usados para realizar operações booleanas como "AND", "OR" etc. Abaixo está uma lista de operadores lógicos disponíveis no R.

Operador	**Descrição do operador**
!	NÃO Lógico
&	E lógico para cada elemento
&&	E lógico
|	OU lógico para cada elemento
||	OU lógico

- Operadores & e | realizam operação elemento a elemento produzindo resultado com comprimento do operando mais longo utilizado na comparação.
- Mas && e || examina apenas o primeiro elemento dos operandos resultando em um vetor lógico de comprimento únitario.
- Zero é considerado FALSO e números diferentes de zero são considerados VERDADEIROS.

Operadores lógicos

Operações básicas

Um exemplo de execução.

```
> x <- c(TRUE,FALSE,0,6)
> y <- c(FALSE,TRUE,FALSE,TRUE)
> !x
[1] FALSE  TRUE  TRUE FALSE
> x&y
[1] FALSE FALSE FALSE  TRUE
> x&&y
[1] FALSE
> x|y
[1]  TRUE  TRUE FALSE  TRUE
> x||y
[1] TRUE
```

Operação em Vetores

Operações básicas

Os operadores acima mencionados também funcionam em vetores. Todas as operações são realizadas de maneira elementar.

```
> x <- c (2,8,3)
> y <- c (6,4,1)
> x + y
[1] 8 12 4
> x > y
[1] FALSE TRUE TRUE
```

Operação em Vetores

Operações básicas

- Quando há uma incompatibilidade de comprimento (número de elementos) de vetores de operando, os elementos em um mais curto são "reciclados" de maneira cíclica para coincidir com o comprimento do mais longo.
- O R emitirá um aviso se o comprimento do vetor mais longo não for um múltiplo do vetor mais curto.

```
> x <- c(2,1,8,3)
> y <- c(9,4)
> x+y # Element of y is recycled to 9,4,9,4
[1] 11  5 17  7
> x-1 # Scalar 1 is recycled to 1,1,1,1
[1] 1 0 7 2
> x+c(1,2,3)
[1]  3  3 11  4
Warning message:
In x + c(1, 2, 3) :
longer object length is not a multiple of shorter object length
```

Operadores de atribuição

Operações básicas

Operadores de atribuição no R são usados para atribuir valores a variáveis.

Operador	**Descrição do operador**
$<-$, $<<-$, $=$	Atribuição à esquerda
$->$, $->>$	Atribuição à direita

Os operadores $<-$ e $=$ podem ser usados, quase que indistintamente, para atribuir a variável no mesmo ambiente. O operador $<<-$ é usado para atribuir variáveis nos ambientes "pai" (mais como atribuições globais). As atribuições à direita, embora disponíveis, raramente são usadas.

Operadores de atribuição

Operações aritméticas básicas

Um exemplo de execução.

```
> x <- 5
> x
[1] 5
> x = 9
> x
[1] 9
> 10 -> x
> x
[1] 10
```

Exercício

Suponha que você esta matriculado na disciplina de estatística. E você quer calcular a sua média de aprovação. O critério de aprovação é

$$Media = 0.1LT + 0.15TR + 0.25P1 + 0.5P2 \tag{1}$$

Aonde LT, TR, P1 e P2 significam Lista, Trabalho, Prova 1 e Prova 2 respectivamente. Sabendo que suas notas são $LT = 5.0$, $TR = 8.0$, $P1 = 6.0$, $P2 = 5.0$

1) Crie um vetor com o nome das avaliações.
2) Crie um vetor com suas notas.
3) Crie um vetor com os pesos.
4) Crie um vetor com o resultado de pesos * notas.
5) Crie um data.frame com 4 colunas, a primeira chamada de "Avaliacao" contendo o nome das avaliações, a segunda chamada "Peso" contendo os pesos das notas, a terceira chamada "Nota" contendo o valor da nota, e por ultimo uma coluna chamada "SubTotal" contendo o resultado de pesos * notas. Salve esse data.frame em uma variavel chamada tabela.

Exercício

6) Calcule a sua media final somando cada item da coluna SubTotal do data.frame

7) O professor notou que cometeu um erro na correção da ultima avaliação alterando sua nota da prova 2 para 6.0. Faça as devidas alterações e reporte a sua nova nota.

Funções aritméticas básicas

O R já contém varias funções aritimeticas pre-programadas.

Descricao	**Função**
Logaritmos e exponenciais	log(x), log2(x), log10(x), exp(x)
Funções trigonométricas	cos(x), sin(x), tan(x), acos(x), asin(x), atan(x)
Valor absoluto	abs (x)
Raiz quadrada	sqrt(x)

```
> log2(4)
[1] 2
> exp(1)
[1] 2.718282
> log(exp(1))
[1] 1
> abs(-4)
[1] 4
> sqrt(4)
[1] 2
```

Funções vetoriais básicas

As funções abaixo são aplicadas a uma estrutura vetorial de dados.

Descricao	**Função**
Máximo e mínimo	max(x), min(x)
Intervalo	range(x)
Tamanho	length(x)
Soma	sum(x)
Média	mean(x)
Produtório	prod(x)
Desvio Padrão	sd(x)
Variancia	var(x)
Reordenação	sort(x)
resumo estatistico	summary(x)

```
> idades <- c(27, 25, 29, 26)
> mean(idades)
[1] 26.75
> max(idades)
[1] 29
> sum(idades)
[1] 107
> sort(idades)
[1] 25 26 27 29
```

Funções matriciais básicas

As funções abaixo são aplicadas a uma estrutura matricial de dados. As funções de vetores também podem ser utilizada nas matrizes.

Descricao	**Função**
Número de colunas ou linhas	ncol(mat), nrow(mat)
Dimensão da matrix	dim(mat)
funcões em linhas ou colunas	rowSums(mat), colSums(mat), rowMeans(mat), colMeans(mat) apply(mat, 2, sum)
Transposição	t(mat)

```
> mat <- matrix(1:6, ncol = 2)
> mat
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> nrow(mat)
[1] 3
> dim(mat)
[1] 3 2
> t(mat)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Funções de data.frame básicas

As funções abaixo são aplicadas a uma estrutura matricial de data.frame. As funções de matrizes também funcionanm nos data.frames.

Descricao	**Função**
Criar data.frame	data.frame()
Extender data.frame com colunas	cbind()
Extender data.frame com linhas	rbind()

```
> dat = data.frame(id = 1:4, nome=c("paulo", "maria", "joao", "bruno"), aprovado = c(T,F, F, T))
> dat
  id  nome aprovado
1  1 paulo     TRUE
2  2 maria    FALSE
3  3  joao    FALSE
4  4 bruno     TRUE
> dat[2, 2]
[1] "maria"
> dat[4, c(2, 3)]
   nome aprovado
4 bruno     TRUE
```

if/else

Às vezes, precisamos realizar ações diferentes com base em condições diferentes. Para fazer isso, podemos usar a instrução *"if"*, em conjunto com a instrução *"else"* (caso necessário).

A sintaxe geral do bloco if-else é:

```
# Syntax of for loop
if (condicao) {
# instrucoes a serem executadas caso condicao seja TRUE
} else {
# instrucoes a serem executadas caso condicao seja FALSE
}
```

if/else

A declaração "if"

A instrução *if (condicao)* avalia uma condição entre parênteses e, se o resultado for verdadeiro, executa um bloco de código.

```
ano = 2020
if (ano == 2020) {
  print("1o ano da pandemia")
}
# [1] "1o ano da pandemia"
```

if/else

A cláusula "else"

A instrução "*if*" pode conter um bloco "*else*" **opcional**. Ele é executado quando a condição é falsa.

```
ano = 2019
if (ano == 2020) {
  print("2020 é o 1o ano da pandemia")
} else {
  print("Nao é o ano de 2020")
}
# [1] "Nao é o ano de 2020"
```

A cláusula "else if"

Às vezes, gostaríamos de testar várias variantes de uma condição. A cláusula "*else if*" nos permite fazer isso.

```
ano = 2029
if (ano == 2020) {
  print("2020 é o 1o ano da pandemia")
} else if (ano > 2020) {
  print("Ainda nao sabemos se havera pandemia do COVID-19")
} else {
  print("Ano sem COVID-19")
}
# [1] "Ainda nao sabemos se havera pandemia do COVID-19"
```

Looping

O que são loops? um *loop* é um processo iterativo no qual um conjunto de instruções é uma replicado para cada interação. Não é nada mais do que automatizar um processo de várias etapas organizando sequências de ações e agrupando as partes que precisam ser repetidas.
as construções de loop são:

- for
- while
- repeat (não será abordado)

com as cláusulas adicionais

- break
- next

for loop

Um *for loop* é usado para iterar sobre um vetor na programação R. sua sintaxe é:

```
# Syntax of for loop
for (val in sequence)
{
# instrucoes a serem executadas
}
```

Aqui, "*sequence*" é um vetor e "val" assume cada um de seus valores durante o *loop*. Em cada iteração, a instrução é avaliada.

```
# Syntax of for loop
for (val in 1:10) {
    print(val)
}
# [1] 1
# [1] 2
# [1] 3
# [1] 4
# [1] 5
# [1] 6
# [1] 7
# [1] 8
# [1] 9
# [1] 10
```

for loop

Example

1) Crie um *for loop* que faz a soma dos numeros de 1 a 42.
2) Crie um vetor com 5 nomes. Faça um *for loop* que imprima (utilize o comando print()) todos os nomes um a um.
2) Crie um vetor com os números de 1 a 10. Faça um *for loop* que imprima todos o exponencial de cada número.

while loop

O *while loop* é um bloco de instruções que será executado enquanto uma condição lógica for atendida.
sua sintaxe é:

```
# Syntax of while loop
while (condicao)
{
# instrucoes a serem executadas
}
```

```
number = 0
while (number < 3)
{
  print(number)
  number = number +1
}
# [1] 0
# [1] 1
# [1] 2
```

while loop

Example

1) Crie um *while looping* que faz a soma dos numeros de 1 a 42.
2) crie um *while looping* que dado um número (escolhido previamente), inteiro, ele faz a adição ou subtração de uma unidade (+/- 1) a cada interação, até atingir o número 42.

Interrompendo o Looping

Uma pergunta que você deve se fazer é: como você sai de um loop? Existem duas maneiras de "parar" um *looping*, os comandos são:

- **next:** O comando next para a interação atual, mas o looping como um todo ainda é executado até o final.
- **break:** o comando break para a interação atual e para também o looping como um todo.

```
for (number in 1:100)
{
  if(number ==2) {
    print("pulando o 2")
    next
  }
  if(number == 4)
  {
 print("Parando no 4")
 break
  }
  print(number)
}
# [1] 1
# [1] "pulando o 2"
# [1] 3
# [1] "Parando no 4"
```

Exercicio modulo 3

Neste exercício iremos tentar abordar todos os assuntos:

- Dados
- Estrutura de dados
- If-Else
- looping

Exercicio modulo 3

Example

Você esta ajudando o professor de uma matéria a calcular a nota dos alunos. Para isso você irá utilizar o software R e automatizar boa parte do processo.

1) Inicie um novo script e salve o mesmo como "ExercicioModulo3.R"
2) Carregue no baco de dados os dados das notas. Para isso utilize o comando "Load('./databases/AlunosModulo3.Rdata')"
3) Verifique que a variavel AlunosNotas esta carregada.
4) Verifique que a variavel AlunosNotas é um data.frame.
5) imprima um "summary" dos dados no data.frame AlunosNotas.
6) Declare um vetor que irá conter os pesos das avaliações. Ao todo serão 4 avaliações "TR = 10", "LT=15", "P1=25", "P2=50".

Exercicio modulo 3

Example

7) Faça uma verificação de que a soma total dos pessos seja igual a 1. Se a soma não for igual a 1, faça a "Normalização" dividindo todos os pesos pela soma total.

8) Adicione uma coluna ao data.frame chamada MF, a qual ira conter a informacao da media final do aluno. Para isso utilize o comando

```
AlunosNotas$MF = as.numeric(NA)
```

9) O que o comando anterior fez? Como estão as médias finais dos alunos? Qual o tipo de variável (inteiro, logica, número real)? (dica de funções para auxiliar a investigação "head(AlunosNotas)" "summary(AlunosNotas)".

Exercicio modulo 3

Example

10) Construa um looping que percorre todas as linhas do data.frame AlunosNotas, e calcula a nota para cada aluno. Guarde a média calculada na coluna MF.

11) Determine qual a quantidade de alunos que foram aprovados.

12) Suponha que agora o professor altere os pesos para "TR = 10", "LT=15", "P1=50", "P2=50". Qual a quantidade de alunos aprovados com esses pesos?

Table of Contents

Data Import

- Introdução Trabalhar com as ferramentas fornecidas por pacotes R é uma ótima maneira de aprender as ferramentas da ciência de dados, mas em algum ponto você deseja parar de aprender e começar a trabalhar com seus próprios dados.
- Vamos abordar dois métodos de importação de dados que vão cobrir a maioria das suas necessidades. Para isso vamos utilizar os Pacotes *readr* e *readxl*. Vamos começar instalando estes pacotes.

```
install.packages("readr")
install.packages("readxl")
```

Data Import with readr

A maioria das funções do *readr* estão voltadas em transformar arquivos simples em data.frame.

Método	**Descrição**
read_csv()	lê arquivos delimitados por vírgulas
read_csv2()	lê arquivos separados por ponto e vírgula
read_tsv()	lê a arquivos separados por tabulação
read_delim()	lê em arquivos com qualquer delimitador

Os dados podem não ter nomes de coluna. Você pode usar "col_names = FALSE" para dizer ao método *read_csv()* para não tratar a primeira linha como títulos, com isso as colunas serão rotuladas sequencialmente de X1 a Xn:

```
library(readr)
AlunosNotas <- read_csv("./databases/Modulo4_AlunosNotas.csv",
    col_names = TRUE)
```

Data Import with readxl

Para trabalhar com os dados em excel vamos utilizar o pacote *readxl*. Abaixo listamos os métodos mais utilizados.

Método	Descrição
read_excel()	lê ambos os arquivos "xls" e "xlsx".
excel_sheets()	lista quais as planilhas presentes em um arquivo excel.

Um ponto importante é que uma planilha pode ser especificada pelo "nome" ou pela sua numeração.

```
library(readxl)
# Lista as planilhas
excel_sheets("Database/Modulo4_AlunosInfo.xlsx")
# [1] "AlunosNotas"
Modulo4_AlunosInfo <- read_excel("Database/Modulo4_AlunosInfo.xlsx",  sheet = "AlunosNotas")
head(Modulo4_AlunosInfo)
# # A tibble: 6 x 4
#   Codigo  Nome    Email                          Telefone
#   <chr>   <chr>   <chr>                          <chr>
# 1 C210987 Aluno 1 aluno.aleatorio1@dominio.com 9555-0001
# 2 C476201 Aluno 2 aluno.aleatorio2@dominio.com 9555-0002
# 3 C026318 Aluno 3 aluno.aleatorio3@dominio.com 9555-0003
# 4 C342650 Aluno 4 aluno.aleatorio4@dominio.com 9555-0004
# 5 C479805 Aluno 5 aluno.aleatorio5@dominio.com 9555-0005
```

Exercício

exercício

Faça a importação dos arquivos abaixo:

- Modulo4_AlunosNotas.csv
- Modulo4_AlunosInfo.xlsx

Apos importados os arquivos, mostre um "summary()" de cada base de dados, bem como as primeiras linhas de cada banco. Quais as variáveis em cada banco? Quais os tipos de cada variavel?

Data Transformation

- A visualização é uma ferramenta importante para a geração de insights, mas é raro que você obtenha os dados exatamente da forma certa. Freqüentemente, você precisará criar algumas novas variáveis ou resumos, ou talvez queira apenas renomear as variáveis ou reordenar as observações para tornar os dados um pouco mais fáceis de trabalhar.
- Para isso vamos utilizar dois pacotes o *dplyr* e o *tidyr*. Vamos começar instalando estes pacotes.

```
install.packages("dplyr")
install.packages("tidyr")
```

Tidy data

Tidy data é uma maneira específica de organizar dados em um formato consistente que se conecta ao conjunto tidyverse de pacotes para R.

Não é a única maneira de armazenar dados e há razões pelas quais você pode não armazenar dados neste formato, mas muito provavelmente você precisará converter seus dados em um formato organizado para analisá-los com eficiência.

Existem três regras que tornam um conjunto de dados organizado (*Tidy data*):

- Cada variável deve ter sua própria coluna.
- Cada observação deve ter sua própria linha.
- Cada valor deve ter sua própria célula.

Ordene linhas com *arrange()*

pacote: dplyr

arrange() muda a ordem dos dados. É necessário um data.frame e uma ordem baseada nos nomes das coluna (ou expressões) para ordenar

```
> arrange(AlunosNotas, P2, desc(P1))
# A tibble: 50 x 6
      X1 id         TR    LT    P1    P2
   <dbl> <chr>   <dbl> <dbl> <dbl> <dbl>
 1     1 C210987   2.4   5.2   6.6   0.2
 2    47 C687291   8.7   0.6   5.3   1.2
 3    26 C290175   7.2   1.1   5.9   1.4
 4    13 C913587   3.3   0.3   4.6   1.5
 5     9 C036179   8.1   1.2   6     1.8
 6     3 C026318   9.1   4     6     2.1
 7    16 C643592   3.1   7.7   6.6   2.6
 8     6 C378625   4.8   7.9   5.6   2.6
 9    35 C276583   7.8   7.2   6.6   2.7
10    19 C273465   7.4   5.4   5.7   2.8
# ... with 40 more rows
```

Adicione novas variáveis com *mutate()*
pacote: dplyr

Muitas vezes é útil adicionar novas colunas que são funções de colunas existentes. Esse é o trabalho de mutate(). Esta função sempre adiciona novas colunas no final do seu conjunto de dados.

```
> mutate(AlunosNotas, meia_nota = P2/2)
# A tibble: 50 x 7
      X1 id         TR    LT    P1    P2 meia_nota
   <dbl> <chr>   <dbl> <dbl> <dbl> <dbl>     <dbl>
 1     1 C210987   2.4   5.2   6.6   0.2       0.1
 2     2 C476201   3.5   1.4   7.2   6.7       3.35
 3     3 C026318   9.1   4     6     2.1       1.05
 4     4 C342650   8.7   7.2   6.1   7.3       3.65
 5     5 C479805   2.7   5.9   5.8   3.3       1.65
 6     6 C378625   4.8   7.9   5.6   2.6       1.3
 7     7 C954367   4.8   0.4   5.6   4.4       2.2
 8     8 C480723   5.6   7.1   6.4   3.3       1.65
 9     9 C036179   8.1   1.2   6     1.8       0.9
10    10 C951437   8     7.9   7.1   7         3.5
# ... with 40 more rows
```

Selecione colunas com *select()*
pacote: dplyr

select() permite que você selecione rapidamente um subconjunto útil usando operações baseadas nos nomes das variáveis.

```
> select(AlunosNotas, P1, P2)
# A tibble: 50 x 2
      P1    P2
   <dbl> <dbl>
 1   6.6   0.2
 2   7.2   6.7
 3   6     2.1
 4   6.1   7.3
 5   5.8   3.3
 6   5.6   2.6
 7   5.6   4.4
 8   6.4   3.3
 9   6     1.8
10   7.1   7
# ... with 40 more rows
```

Filtre linhas com *filter()*
pacote: dplyr

filter() permite que você crie um subconjunto de observações com base em seus valores. O primeiro argumento é o data.frame. O segundo argumento e os subsequentes são as expressões que filtram o quadro de dados.

```
> filter(AlunosNotas, P2 >= 6)
# A tibble: 15 x 6
      X1 id         TR    LT    P1    P2
   <dbl> <chr>   <dbl> <dbl> <dbl> <dbl>
 1     2 C476201   3.5   1.4   7.2   6.7
 2     4 C342650   8.7   7.2   6.1   7.3
 3    10 C951437   8     7.9   7.1   7
 4    12 C180467   3.4   7.5   6.5   6.6
 5    15 C154379   4.3   7.4   7     6.8
 6    20 C824013   6.7   1.6   4.9   6.6
```

Vários argumentos podem ser utilizados no método *filter()*, porém eles são combinados com "e". Todas as expressões devem ser verdadeiras para que uma linha seja incluída na saída. Para outros tipos de combinações, você mesmo precisará usar operadores lógicos.

Exercício

Utilizando os dados presente em "Modulo4_AlunosNotas.csv" determine:

1. A soma total de todas as provas para cada aluno.
2. Calcule a média de cada aluno, sabendo que os pesos são: $TR = 0.1$, $LT = 0.2$, $P1 = 0.3$ e $P2 = 0.4$
3. Selecione apenas as colunas Id e as Média.
4. A partir do resultado anterior, ordene os dados do melhor aluno para o pior aluno.

Summirise() e group_by()

Agrupe os dados com *summarise()*
pacote: dplyr

O método summarise() reduz um data.frame em uma única linha: *summarise()* só é útil quando usado em conjunto com *group_by()*. O comando *group_by()* muda a unidade de análise do banco de dados completo para grupos. Sendo assim utilizando os dois comandos em conjuntos temos que o *group_by()* indica a unidade de agrupamento, e o *summarise()* indica qual a informação a ser extraida de cada grupo.

```
> # adiciono uma coluna que informa se a P2 fpi acima de seis
> AlunosNotas2 <- mutate(AlunosNotas, AcimaDeSeis = P2 > 6)
> # Agrupo pela coluna "AcimaDeSeis"
> AlunosNotas.Grouped <- group_by(AlunosNotas2, AcimaDeSeis)
> # Resume os dados
> summarise(AlunosNotas.Grouped,
+           MediaP1 = mean(P1, na.rm = TRUE),
+           MediaP2 = mean(P2, na.rm = TRUE),
+           TotalAlunos = n(),
+           )
# A tibble: 2 x 4
  AcimaDeSeis MediaP1 MediaP2 TotalAlunos
  <lgl>         <dbl>   <dbl>       <int>
1 FALSE          5.87    3.79          35
2 TRUE           6.17    7.57          15
```

Relational Data - Joins

junte os dados com *Joins()*
pacote: dplyr

Uma junção permite combinar variáveis de duas tabelas. Para isso é necessario que as observações sejam identificadas por uma "'chave de identificação', ele combina as observações por suas chaves e, em seguida, copia as variáveis de uma tabela para a outra.
Vamos abordar aqui tres tipos de junção:

1. Inner Join
2. Left Join
3. Right Join

Existem outros tipos de junção, porém os três tipos listados são os mais utilizados

inner_join()

inner_join()
pacote: dplyr

O tipo mais simples de associação é a associação do tipo *inner_join()*. Esse tipo de junção corresponde a pares de observações sempre que suas chaves são iguais: A saida de uma junção *inner_join()* é um novo data.frame que contém a chave, os valores da primeira tabela e os valores da segunda tabela.

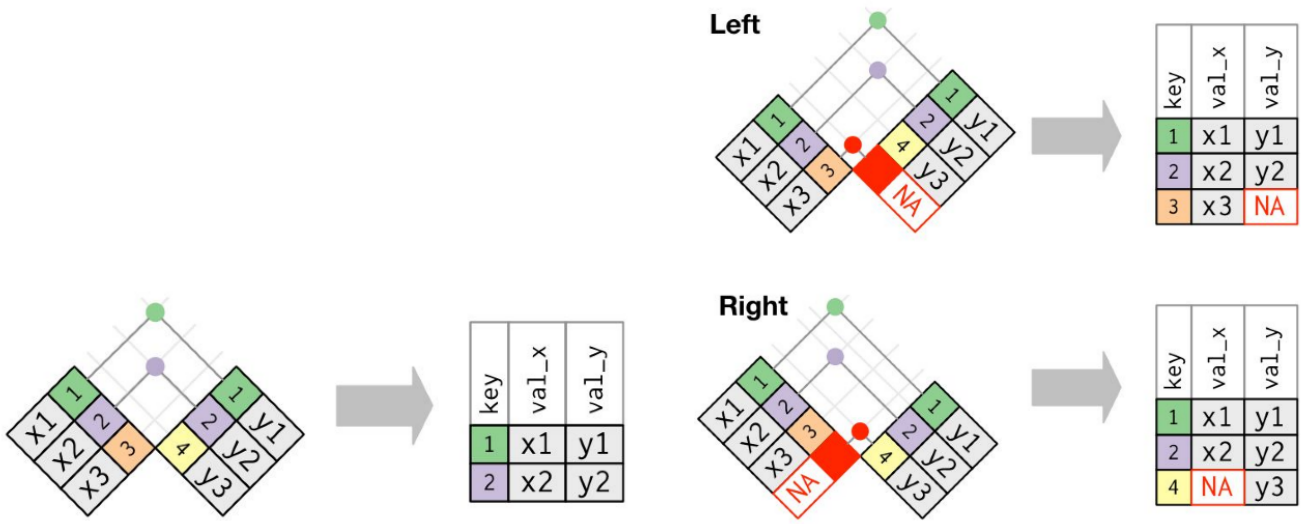
```
> inner_join(AlunosNotas, AlunosInfo, by=c("id"="Codigo"))
# A tibble: 50 x 9
      X1 id          TR    LT    P1    P2 Nome     Email                          Telefone
   <dbl> <chr>    <dbl> <dbl> <dbl> <dbl> <chr>    <chr>                          <chr>
 1     1 C210987   2.4   5.2   6.6   0.2 Aluno 1  aluno.aleatorio1@dominio.com  9555-0001
 2     2 C476201   3.5   1.4   7.2   6.7 Aluno 2  aluno.aleatorio2@dominio.com  9555-0002
 3     3 C026318   9.1   4     6     2.1 Aluno 3  aluno.aleatorio3@dominio.com  9555-0003
 4     4 C342650   8.7   7.2   6.1   7.3 Aluno 4  aluno.aleatorio4@dominio.com  9555-0004
 5     5 C479805   2.7   5.9   5.8   3.3 Aluno 5  aluno.aleatorio5@dominio.com  9555-0005
 6     6 C378625   4.8   7.9   5.6   2.6 Aluno 6  aluno.aleatorio6@dominio.com  9555-0006
 7     7 C954367   4.8   0.4   5.6   4.4 Aluno 7  aluno.aleatorio7@dominio.com  9555-0007
 8     8 C480723   5.6   7.1   6.4   3.3 Aluno 8  aluno.aleatorio8@dominio.com  9555-0008
 9     9 C036179   8.1   1.2   6     1.8 Aluno 9  aluno.aleatorio9@dominio.com  9555-0009
10    10 C951437   8     7.9   7.1   7   Aluno 10 aluno.aleatorio10@dominio.com NA
# ... with 40 more rows
```

left_join() ou *right_join()*

pacote: dplyr

Uma junção do tipo *inner_join()* mantém as observações que aparecem em ambas as tabelas. Uma junção externa (*left_join()* ou *right_join()*) mantém as observações que aparecem em pelo menos uma das tabelas.

Existe ainda a junção do tipo *full_join()* a qual mantém todos os registros.



pivot_wider()
pacote: tidyr

pivot_wider() "amplia" os dados, aumentando o número de colunas e diminuindo o número de linhas. A transformação inversa é *pivot_longer()*.

```
> AlunosNotas.wider <- pivot_longer(AlunosNotas, cols = c("TR", "LT", "P1", "P2"),
names_to = "Avalicao", values_to = "Nota")
AlunosNotas.wider
# A tibble: 200 x 4
      X1 id      Avalicao  Nota
   <dbl> <chr>   <chr>    <dbl>
 1     1 C210987 TR         2.4
 2     1 C210987 LT         5.2
 3     1 C210987 P1         6.6
 4     1 C210987 P2         0.2
 5     2 C476201 TR         3.5
 6     2 C476201 LT         1.4
 7     2 C476201 P1         7.2
 8     2 C476201 P2         6.7
 9     3 C026318 TR         9.1
10     3 C026318 LT         4
# ... with 190 more rows
```

Exercício

Utilizando os dados presente em "Modulo4_AlunosNotas.csv" e em "Modulo4_AlunosInfo.xlsx" determine:

1. Construa um banco de dados com as informações dos dois arquivos.
2. Para o banco de dados completo, calcule a média de cada aluno, sabendo que os pesos são: $TR = 0.1$, $LT = 0.2$, $P1 = 0.3$ e $P2 = 0.4$
3. determine quem são os alunos com media final calculada mas que não tem telefone de contato.

pivot_longer()
pacote: tidyr

pivot_longer() "alonga" os dados, aumentando o número de linhas e diminuindo o número de colunas. A transformação inversa é *pivot_wider()*

```
> pivot_wider(AlunosNotas.wider, id_cols = c("X1", "id"), names_from = "Avalicao", values_from="Nota")
# A tibble: 50 x 6
      X1 id         TR    LT    P1    P2
   <dbl> <chr>   <dbl> <dbl> <dbl> <dbl>
 1     1 C210987   2.4   5.2   6.6   0.2
 2     2 C476201   3.5   1.4   7.2   6.7
 3     3 C026318   9.1   4     6     2.1
 4     4 C342650   8.7   7.2   6.1   7.3
 5     5 C479805   2.7   5.9   5.8   3.3
 6     6 C378625   4.8   7.9   5.6   2.6
 7     7 C954367   4.8   0.4   5.6   4.4
 8     8 C480723   5.6   7.1   6.4   3.3
 9     9 C036179   8.1   1.2   6     1.8
10    10 C951437   8     7.9   7.1   7
# ... with 40 more rows
```

Gravando em um arquivo

readr também vem com duas funções úteis para gravar dados de volta no disco: *write_csv()* e *write_tsv()*. Ambas as funções aumentam as chances de o arquivo de saída ser lido de volta corretamente.
Se você quiser exportar um arquivo CSV para o Excel, use *write_excel_csv()* - isso grava um caractere especial (uma "marca de ordem de bytes") no início do arquivo, que informa ao Excel que você está usando a codificação UTF-8.

```
write_csv(x=AlunosNotas, file = "AlunosNotas.csv")

write_excel_csv(x=AlunosNotas, file = "AlunosNotas.csv")
```

Gravando em um arquivo

write_csv() vs write.csv()

A write_csv() é uma melhoria para a função write.csv() porque é aproximadamente duas vezes mais rápida.

Uma das maiores diferencças é que ao contrário de write.csv(), a função write_csv() não inclui nomes de linha como uma coluna no arquivo escrito.

Gravando em um arquivo

Também é possivel escrever um arquivo Excel (com extensão .xlsx) utilizando a função write_xlsx() do pacote **writexl**. Para criar um "xlsx" com (várias) planilhas (abas), basta definir uma **lista** nomeada de **data.frames**.

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

write_xlsx(x = mtcars,
           path = "mtcars.xlsx")

my_list <- list("Aba_1" = mtcars, "Aba_2" = mtcars)
write_xlsx(x = my_list,
           path = "VariasMtcars.xlsx")
```

Table of Contents

Trabalhando com Scripts

source()

O que é um "script"?
Um script é um programa ou sequência de instruções que é interpretado ou executado.

Logo, um "'script" é um arquivos (no R são gravados na forma de texto) com lista de comandos para serem executados. Os "scripts" no R tem a extensão "*.R"

Trabalhando com Scripts

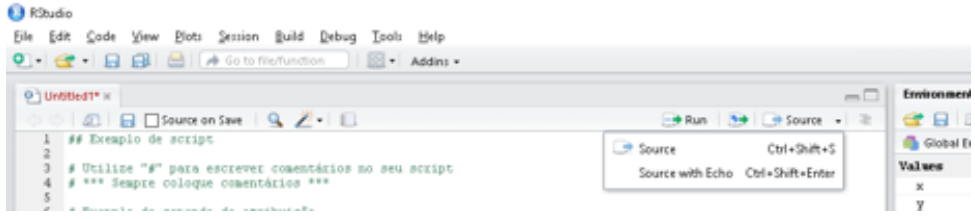
source()

Criando um novo script

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:



Trabalhando com Scripts

source()

Os scripts podem ser gravados para serem executados posteriormente. Para executar scripts previamente gravados utiliza-se a função *source()* a qual lê e interpreta um script pre-definido.

A função *source()* faz com que R analise e interprete o arquivo no ambiente atual.

```
# Carregando o script.r
source("Script.r")
```

Trabalhando com Scripts

Exemplo de Script:

```
# Script OiMundo.r
rm(list = ls())
frase <- sprintf("%s\nOi mundo\n%s\n",
                 paste(rep("*", 20), collapse = ""),
                 paste(rep("*", 20), collapse = ""))
cat(frase)
frase <- sprintf("Imprimindo base de dados: Harman Example 2.3\n")
cat(frase)
print(Harman23.cor)
```

Example

- Baixe o arquivo “OiMundo.r”
- Execute o comando source(“OiMundo.r”)

O que é uma função?

O que é uma função?
Uma função é um conjunto de instruções (que podem estar organizadas em um *script*) para realizar uma tarefa específica. O R tem um grande número de funções embutidas e o usuário pode criar suas próprias funções.

No R, uma função é um objeto (variável) para o qual o R é capaz de executar, passando junto os argumentos que podem ser necessários para a função realizar as ações.

Definição de Função

Funções Nativas
O R tem muitas funções embutidas que podem ser chamadas diretamente no programa sem defini-las primeiro. Também podemos criar e usar nossas próprias funções, chamadas de funções definidas pelo usuário.

Algumas funções **nativas** que vamos ver:

- Repetição (rep)
- Sequencia (seq)
- Reverter (rev)

Criando uma Função

Uma função no R é criada usando a designação *function*. A sintaxe básica de uma definição de função é a seguinte:

```
function_name <- function(arg_1, arg_2, ...) {
   # Function body
   return(retun_value)
}
```

A chamada de uma função:

```
        function_name(Arg01, Arg02, ...);
# ou
        function_name(arg_1 = Arg01,.., arg_2=ArgX,...);
```

Componentes de Função

Uma função tem diferentes partes, elas podem ser classificadas como:

- **Nome da função:** É o nome real da função. Ele é armazenado no ambiente R como um objeto com este nome.
- **Argumentos:** Um argumento é uma variável que a função utiliza para seus cálculos. Quando uma função é chamada, você passa um valor para o argumento. Os argumentos são opcionais; ou seja, uma função pode não ter argumentos. Além disso, os argumentos podem ter valores pre-definidos.
- **Corpo da função:** O corpo da função contém uma coleção de instruções que definem o que a função faz.
- **Valor de retorno:** O valor de retorno de uma função é a última expressão no corpo da função a ser avaliada.

Exercício

Considere o problema de calcular as raízes de uma equação de segundo grau:

$$ax^2 + bx + c = 0 \quad a \neq 0$$

Escrever a função: RaizesQuadratica(a, b, c)

Argumentos:
a: vetor numérico com n elementos
b: vetor numérico com n elementos
c: vetor numérico com n elementos
Retorno:

- x: Matriz $n \times 2$ com a solução da equação quadrática.
- Duas raízes reais distintas $x[i,1] = x1$ e $x[i,2] = x2$ com $x1 < x2$
- Uma única raiz real $x[i,1] = x1$ e $x[i,2] = \text{NA}$
- Não existe raízes reais: $x[i,1] = x[i,2] = \text{NA}$

Quando você deve escrever uma função?

Regra de bolso

Você deve considerar escrever uma função sempre que copiou e colou um bloco de código mais de duas vezes (ou seja, agora você tem três cópias do mesmo código).

Noções básicas de codificação

Example

Analise o código abaixo.

- Você é capaz de "encapsular" esse codigo em uma função?
- Você é capaz de identificar o erro no código?

```
df <- tibble:: tibble(a = rnorm(10), b = rnorm(10), c = rnorm(10), d = rnorm(10))
df$a <- (df$a - min(df$a, na.rm = TRUE))/(max(df$a, na.rm = TRUE) - min(df$a, na.rm = TRUE))
df$b <- (df$b - min(df$b, na.rm = TRUE))/(max(df$b, na.rm = TRUE) - min(df$a, na.rm = TRUE))
df$c <- (df$c - min(df$c, na.rm = TRUE))/(max(df$c, na.rm = TRUE) - min(df$c, na.rm = TRUE))
df$d <- (df$d - min(df$d, na.rm = TRUE))/(max(df$d, na.rm = TRUE) - min(df$d, na.rm = TRUE))
```

Técnicas para "debugar" o código

Técnicas para "*Debug*"[2] de código

- *breakpoints* com inspeção de variáveis (ex: print("checkpoint"))
- *breakpoints* com RStudio
- *breakpoints* com "browser()"

[2] Grace Brewster Murray Hopper (1947)

Obtendo ajuda

Como obter ajuda sobre uma função?
Para obter ajuda de uma função utilize o comando *help()*

```
# help([nome da funcao entre aspas])
help("cov")
```

Para obter as funções que tratam de um assunto especifico podemos utilizar o "??"

```
# lista funcoes com a palavra cov
??"cov"
??cov
```

O que é um pacote?
Os pacotes R são coleções de funções e conjuntos de dados desenvolvidos pela comunidade. Eles aumentam a potência do R melhorando as funcionalidades de base do R existentes ou adicionando novas. Dois dos pacotes R mais populares que já vimos são o dplyr e ggplot2.

Logo, um pacote é uma forma adequada de organizar seu próprio trabalho e, se desejar, compartilhá-lo com outras pessoas. Normalmente, um pacote inclui código (não apenas código do R!), Documentação para o pacote e as funções internas, alguns testes para verificar se tudo funciona como deveria e conjuntos de dados.

Pacotes

O que é um pacote?

As informações básicas sobre um pacote são fornecidas no arquivo **DESCRIPTION**, onde você pode descobrir o que o pacote faz, quem é o autor, a qual versão a documentação pertence, a data, o tipo de licença de uso e as dependências do pacote.

```
> packageDescription("stats")
Package: stats
Version: 4.0.2
Priority: base
Title: The R Stats Package
Author: R Core Team and contributors worldwide
Maintainer: R Core Team <R-core@r-project.org>
Description: R statistical functions.
License: Part of R 4.0.2
Imports: utils, grDevices, graphics
Suggests: MASS, Matrix, SuppDists, methods, stats4
NeedsCompilation: yes
Built: R 4.0.2; x86_64-w64-mingw32; 2020-06-22 08:44:20 UTC; windows

-- File: C:/Program Files/R/R-4.0.2/library/stats/Meta/package.rds
> help(package = "stats")
```

O que é um repositório?

Um repositório é um local onde os pacotes estão disponíveis para que você possa instalá-los. Embora você ou sua organização possam ter um repositório local, normalmente eles estão online e acessíveis a todos. Três dos repositórios mais populares para pacotes R são:

- CRAN: o repositório oficial, é uma rede de servidores ftp e web mantida pela comunidade R em todo o mundo.
- Biocondutor: trata-se de um repositório de tópicos específicos, destinado a softwares de código aberto para bioinformática.
- Github: embora não seja específico do R, o Github é provavelmente o repositório mais popular para projetos de código aberto.

Como instalar um pacote do R?

Instalando Pacotes do CRAN

Como você pode instalar um pacote dependerá de onde ele está localizado. Portanto, para pacotes disponíveis publicamente, isso significa a qual repositório ele pertence. A forma mais comum é usar o repositório CRAN, então você só precisa do nome do pacote e usar o comando install.packages("pacote").

```
> install.packages("MASS")
WARNING: Rtools is required to build R packages but is not currently installed.
Please download and install the appropriate version of Rtools before proceeding:

https://cran.rstudio.com/bin/windows/Rtools/
Installing package into ‘C:/Users/bteba/Documents/R/win-library/4.0’
(as ‘lib’ is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.0/MASS_7.3-53.zip'
Content type 'application/zip' length 1187627 bytes (1.1 MB)
downloaded 1.1 MB

package ‘MASS’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
...
```

Como Carregar Pacotes?

Depois que um pacote é instalado, você está pronto para usar suas funcionalidades.

library() é o comando usado para carregar um pacote e se refere ao local onde o pacote está contido, geralmente uma pasta em seu computador.

```
> library(MASS)
Warning message:
package 'MASS' was built under R version 4.0.3
```

Como descarregar um Pacote?

Para descarregar um determinado pacote, você pode usar a função *detach()*. O uso será:

```
> detach("package:MASS", unload = TRUE)
```

Note que é necessário que a sintaxe seja:
"package:[NOME DO PACOTE]"

Fontes alternativas de documentação

Conforme mencionado o arquivo **DESCRIPTION** contém informações básicas sobre um pacote e, embora essas informações sejam muito benéficas, elas nem sempre irão ajudá-lo a usar este pacote para sua análise.

Arquivos de ajuda Como no R básico, os comandos "?()" E "help()" são a primeira fonte de documentação quando você está iniciando com um pacote. Cada função pode ser explorada individualmente por help("nome da função") ou help(função, pacote = "pacote").

```
> help(area, package = "MASS")
```

Projeto

Vamos por em pratica tudo o que aprendemos até o momento.

Descrição do problema: Suponha que você é um professor e precisa automatizar o lançamento de notas dos seus alunos. Os seguintes arquivos são disponibilizados:

- Notas_P1.csv : Contém a informação sobre as notas na primeira prova
- Notas_P2.csv : Contém a informação sobre as notas na segunda prova
- Notas_TrabalhoEListas.xlsx : Contém a informação sobre as notas no trabalho e nas listas
- Info_Alunos.csv : Contém a informação sobre os alunos

Para a média final utiliza-se a seguinte formula:

$$MF = 0.1 \times Trabalho + 0.2 \times Lista + 0.3 \times P1 + 0.4 \times P2$$

Projeto

O objetivo do programa é juntar todas as informações, gerar um histograma das notas de cada prova bem como para a média final (O programa deve considerar uma mudança futura dos pesos)

1. Crie um Projeto no RStudio chamado CalculoDeNotas
2. Crie um arquivo chamado DataLoader.R. Este arquivo deverá conter toda a logica de carregamento dos dados. Nele deverão constar os seguintes procedimentos:
   - Leitura dos arquivos de notas.
   - Junção de todos os dados em um único data.frame chamado Dados_Alunos
3. Crie um arquivo chamado functionMedia.R. Este arquivo deverá conter uma função que será responsável pela logica de calculo da média.

Projeto

4. Crie um arquivo chamado CalculadorNotas.R. Este arquivo deverá conter toda a logica de calculo das notas. Nele deverão constar os seguintes procedimentos:
   - listagem das bibliotecas utilizadas.
   - Leitura do arquivo DataLoader.R.
   - Leitura do arquivo functionMedia.R.
   - Um looping de calculo da média para cada aluno.
   - O data frame deve conter uma coluna que indica se o aluno foi aprovado ou nao. ($media \geq 6$)
   - Um processo que gera os gráficos solicitados
   - Um conjunto de instruções que salva a planilha Dados_Alunos em um arquivo csv os gráficos solicitados.

Table of Contents

- Pipes são uma ferramenta poderosa para expressar claramente uma sequência de várias operações.
- Até agora, você tem usado as funções sem saber da existência do *Pipe*, ou de como funciona.
- Agora é hora de apresentarmos o *Pipe* com mais detalhes. Você aprenderá as alternativas ao tubo, quando não deve usar o tubo e algumas ferramentas úteis relacionadas.

Pipe

- O pipe, %>%, vem do pacote **magrittr** de Stefan Milton Bache. Pacotes no arrumado carregam %>% para você automaticamente, então você normalmente não precisa carregar o pacote **magrittr** explicitamente. Aqui, no entanto, estamos focando na tubulação e não estamos carregando nenhum outro pacote, então vamos carregá-lo explicitamente

```
# Carregando o pacote magrittr
library(magrittr)
```

- O objetivo do pipe é ajudá-lo a escrever código de uma maneira que seja mais fácil de ler e entender.
- Para ver por que o pipe é tão útil, vamos explorar várias maneiras de escrever o mesmo código. Utilizando o banco de dados de *ChickWeight*: (1) Vamos criar uma coluna a mais; (2) Vamos informar como agrupar os dados; (3) Fazemos o agrupamento; (4) A partir dos dados agrupados imprimimos um grafico.

Pipe

- A maneira mais fácil de se executar os passos anteriormente descritos é fazendo uma serie de passos intermediários.

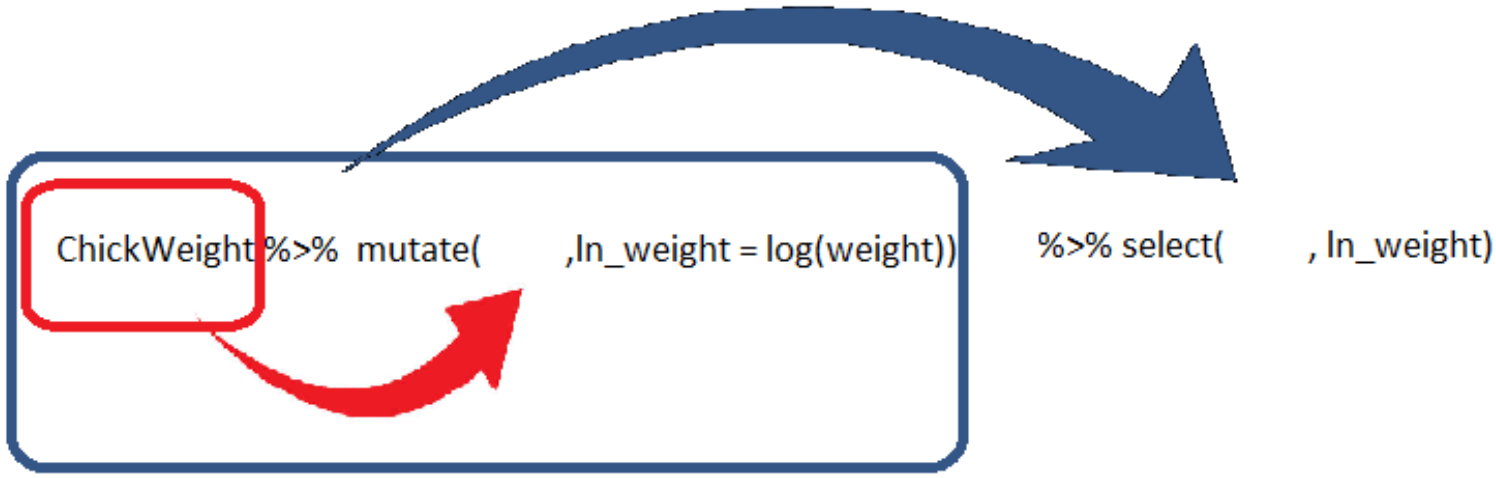
```
# Crio uma nova coluna
tbl_1 <- mutate(ChickWeight, ln_weight = log(weight))

# Informa a maneira de se agrupar os dados
tbl_2 <- group_by(tbl_1, Diet)

# Faco um agrupamento
tbl_3 <- summarise(tbl_2, mu.wight = mean(ln_weight),
                   sd.wight = mean(ln_weight),
                   qtd = n())

#  A partir dos dados agurpados faço um grafico
ggplot(tbl_3) + geom_col(aes(x=Diet, y=mu.wight))
```

Pipe

- A principal desvantagem dessa abordagem é que ela força você a nomear cada elemento intermediário. Isso leva a dois problemas:
  - O código está cheio de nomes sem importância;
  - Você tem que incrementar cuidadosamente o sufixo em cada linha.

Pipe

- Se você nunca viu %>% antes, não tem ideia do que esse código faz. O pipe funciona realizando uma "transformação léxica" passando o objeto anterior como o primeiro argumento da função seguinte.



Pipe

```
# Apenas para leitura dos dados
ChickWeight %>%
  mutate(weight = log(weight)) %>%
  group_by(Diet) %>%
  summarise(mu.wight = mean(weight),
            sd.wight = mean(weight),
            qtd = n()) %>%
  ggplot() +
  geom_col(aes(x=Diet, y=mu.wight))
```

Pipe

- Caso você queira utilizar o *Pipe* porem o destino não é o primeiro argumento, você poder designar o local de destino com um “.”

```
# Os comandos abaixo sao equivalentes
# Opcao 1
ChickWeight %>% lm(weight ~ Time + Diet, data = .) %>% summary()

# Opcao 2
mdl <- lm(weight ~ Time + Diet, data = ChickWeight)
summary(mdl)
```

The new pipe (R 4.1)

- A partir da versao 4.1, o R passou a ter um pipe nativo, representado por |>.
- O pipe nativo (|>) é construído de uma maneira diferente e apresenta ganhos de velocidade e processamento quando comparado com seu predecessor.
- Para 90% das vezes o |> e %>% são provavelmente intercambiáveis. O que você precisa prestar atenção são as diferenças sutis (ou não tão sutis) entre os dois *Pipes*.
- A diferença mais importante seja que o **magrittr** tem um elemento *placeholder* para quando não se quer que o resultado do lado esquerdo vá para o primeiro argumento da expressão do lado direito.

Pipe shortcut

- O RStudio tem um atalho para o Pipe. O qual permite que você insira o memso programaticamente com agilidade.
- a escolha do tipo de Pipe (|> OU %>%) pode ser escolhido nas opcoes do R. (por padrão o RStudio utiliza o %>%)

R-Studio - Pipe shortcut

CTRL-SHIFT-M