

Ciência de Dados na Saúde com R

2020-10-31

Contents

1	Sobre esse livro	5
2	Introdução	7
2.1	Bases de dados	9
2.2	Sobre o software R	11
2.3	Instalação R e RStudio	12
2.4	Primeiros passos com R e RStudio.	30
2.5	Como obter ajuda no R	51
2.6	Pacotes	52
2.7	Materiais complementares	53
3	Conceitos iniciais	55
3.1	População	55
3.2	Amostra	56
3.3	Variáveis	57
3.4	Tabulação de dados	58
3.5	Importação de dados no R	58
3.6	Análise de consistência e tratamento dos dados	60
3.7	Transformação dos dados	69
3.8	Exercícios	72
4	Análise exploratória dos dados	75
4.1	Tabelas de frequências	76
4.2	Medidas resumo	77
4.3	Tabelas cruzadas - duas variáveis qualitativas	80
4.4	Gráficos	81
4.5	Materiais complementares para análise exploratória dos dados . .	92
4.6	Exercícios sobre análise exploratória dos dados	94
5	Aprendizado estatístico	97

Chapter 1

Sobre esse livro

Este livro tem como objetivo abordar todas as etapas de uma análise de dados, discutindo os tópicos do ponto de vista teórico e prático, utilizando o software R como ferramenta de análise de dados.

Na versão atual, a parte teórica do livro está bastante resumida. Para aquele leitor que tenha interesse em aprofundar mais nos assuntos aqui abordados, recomendamos os livros Morettin and Singer (2020), Bussab and MORETIN (2004) e Magalhaes and Lima (2002).

Este material está em construção e em revisão aberta. Fique à vontade para corrigir qualquer tipo de erro que encontrar no nosso material.

Chapter 2

Introdução

A Estatística é a ciência que engloba métodos para coleta, organização, descrição, análise e interpretação de dados, sendo estes estruturados (as estruturas usuais de bases de dados) ou não estruturados (como arquivos de textos, páginas da web, emails, mídias sociais etc). Assim, podemos dizer que por meio da Estatística transformamos dados em informações para o auxílio de tomadas de decisões em situações de incerteza.

Devido à alta capacidade de armazenamento das mídias e ao uso generalizado de computadores, muitos dados estão sendo coletados e o mundo depende cada vez mais de dados para criar conhecimento, obter informações relevantes e prever melhor o futuro. No seu livro *Homo Deus*, de 2016, Yuval Noah Harari argumenta que todas as estruturas políticas e sociais podem ser vistas como sistemas de processamento de dados e que daí surge a religião Dataísmo: “O Dataísmo declara que o universo consiste em fluxos de dados e que o valor de qualquer fenômeno ou entidade é determinada pela contribuição que dá para o processamento de dados” (<https://pt.wikipedia.org/wiki/Dataísmo>).

Na pesquisa médica, em especial, são realizados estudos experimentais ou observacionais, levando à coleção de dados e o objetivo da investigação é responder a uma questão científica. Para exemplificar esse ponto, vamos considerar um problema da área da medicina obstétrica que consiste no estudo da idade gestacional do parto em gestações gemelares (de gêmeos).

A importância de se estudar a idade gestacional do parto em gestação gemelares se deve pelo elevado risco de prematuridade (parto antes de 37 semanas) em gestações múltiplas. Entre as mulheres com gestação gemelar, o parto prematuro que ocorre antes das 37 semanas é observado em mais de 50% dos casos e quase 12% antes de 32 semanas completas de gestação (Silva, 1995). Devido a este fato, observa-se uma taxa de mortalidade neonatal nas gestações gemelares de 6,4 vezes maior que nas gestações únicas (único feto) e essa taxa se mantém inalterada desde o ano de 2000 (into Maternal, 2009).

O trabalho de parto é consequência de eventos fisiológicos, como por exemplo, o predomínio da ação estrogênica em relação à progesterônica. A progesterona é um hormônio fundamental para a manutenção da gravidez, e um declínio na ação da progesterona é fundamental para o início do parto na maioria das espécies de mamíferos, incluindo os primatas (Astle et al., 2003). A progesterona está presente na natureza, em humanos e animais (ovários, placenta, testículos e adrenal). Os seus precursores estão presentes nos vegetais, como a soja e o inhame, e constituem a principal fonte de produção da progesterona natural comercializada (de Oliveira et al., 2016).

Em gestações com colo curto, também com risco de prematuridade, o uso de progesterona comercializada é um tratamento conhecido na literatura para diminuir o risco de prematuridade. No projeto liderado pela obstetra e profa. Dra. Maria de Lourdes Brizot (<http://lattes.cnpq.br/6273300603065618>), a pergunta que se deseja responder é:

O uso de progesterona diminui o risco de prematuridade em gestações gemelares?

Para responder a essa pergunta, foi realizado um estudo prospectivo, randomizado, duplo ensaio cego controlado por placebo que envolveu 390 gestações gemelares sem histórico de parto prematuro. Mulheres com gestações gemelares entre 18 e 21 semanas e 6 dias de gestação foram designadas aleatoriamente em um de dois grupos:

- **Tratamento com progesterona** - progesterona vaginal diária (200 mg) até 34 semanas e 6 dias de gestação (ou até o parto se este ocorreu antes de 35 semanas).
- **Tratamento placebo** - óvulos de placebo até 34 semanas e 6 dias de gestação (ou até o parto se este ocorreu antes de 35 semanas).

Um comentário importante: placebo é toda e qualquer substância sem propriedades farmacológicas, administrada a pessoas ou grupo de pessoas como se tivesse propriedades terapêuticas. A palavra placebo vem do latim placere, que significa “agradar”. Não entraremos em detalhes nesse material sobre tipos de estudo. Para este assunto e maiores discussões sobre placebo, recomendamos ver os slides “Tipos de estudos” disponível em <https://daslab-ufes.github.io/materiais/>.

Voltando ao problema da progesterona, houve 6 perdas de segmento no grupo progesterona e 4 perdas de segmento no grupo placebo, resultando em $n = 189$ no grupo progesterona e $n = 191$ no grupo placebo.

A variação nos dados faz com que a resposta não seja óbvia. Precisamos de ferramentas estatísticas para determinar se a diferença é tão grande que devemos rejeitar a noção de que foi devido ao acaso. No caso do estudo da progesterona, a diferença na proporção de prematuridade do grupo progesterona e do grupo controle é devido à flutuações aleatórias ou é um indicio de que o uso de progesterona é um protocolo mais eficiente?

A seguir são apresentadas as bases de dados que consideramos no decorrer desse material.

2.1 Bases de dados

2.1.1 Gestações gemelares

A base de dados fictícia de gestações gemelares é baseada no estudo citado anteriormente sobre o efeito do uso de progesterona em gestações gemelares. Sabe-se que os históricos obstétrico e clínico da gestante e informações da gestação também podem influenciar a idade gestacional do parto e, por esse motivo, também foram avaliados. São as características observadas:

Código da variável	Descrição
ID	Identificação do paciente
Grupo	Grupo
1	Placebo
2	Progesterona
CORION	Corionicidade
Mono	Monocoriônica
Di	Dicoriônica
Data aval	Data da avaliação
Data nascimento	Data do nascimento materno
COR BRANCO	Cor branca
1	Branca
2	Não branca
Peso Pré	Peso (em Kg)
[espaço em branco]	Dados faltantes
ALT	Altura (em metros)
[espaço em branco]	Dados faltantes
Gesta	Número de gestações
Para	Número de partos
Aborto	Número de abortos
IND_AP	Indicador de antecedente pessoal
1	Sim
0	Não
Tabagismo	Se fuma
1	Sim
0	Não
Alcool	Se bebe bebidas alcólicas
1	Sim
0	Não
Drogas	Se usa drogas ilícitas
1	Sim
0	Não
IG_Aval	Idade gestacional da CTG (em semanas)
MedidaColo	Medida do colo observada no momento da CTG
Num_contra_CTG	Número de contrações
IGP semana	Semanas da IG (idade gestacional) do parto
IGP dia	Dias da IG do parto

Figure 2.1: Variáveis da base de dados gestações gemelares.

A seguir está o exemplo de como os dados de 5 indivíduos estão tabulados.

ID	Grupo	CORION	Data aval	Data nascimento	COR BRANCO	Peso Pré	ALT	Gesta
18	2	Di	2016-03-21	1987-03-29	1	80.0	1.59	
19	2	Di	2016-02-17	1980-02-26	1	NA	1.62	
20	1	Di	2017-12-14	1998-12-19	2	61.0	1.64	
21	1	Mono	2017-04-23	1988-04-30	1	44.0	1.64	
22	2	Di	2016-03-21	1995-03-27	2	100.0	1.59	

Essa base de dados está disponível em <https://daslab-ufes.github.io/materiais/>, chamado de “Dados gemelares”.

2.1.2 Gestações gemelares - depressão e amamentação

Esta base de dados fictícia apresenta informações sobre depressão e amamentação das mesmas observações consideradas na base de dados Gestações gemelares, apresentada no item anterior.

O questionário de depressão EDPS foi respondido pelas gestantes no primeiro trimestre gestacional e respondido novamente pelas mesmas gestantes no quarto mês pós-parto. Esse questionário retorna um escore (de 0 a 30 pontos), em que quanto maior seu valor, maior indicador de depressão.

Sobre a amamentação, foram divididos três grupos a depender das orientações sobre a amamentação recebidas durante o pré-natal. O grupo 1 é formado pelas gestantes que tiveram mentorias durante o pré-natal e acompanhamento nas amamentações das primeiras semanas; o grupo 2 é formado pelas gestações que receberam apenas orientações durante o pré-natal e no grupo 3 estão as gestantes que não receberam orientações sobre amamentação.

São as características observadas:

Codigo da variável	Descrição
ID	Identificação do paciente
EDPS_antes	Escore no questionário de depressão EPDS avaliado no 1º trimestre gestacional
EDPS_depois	Escore no questionário de depressão EPDS avaliado 4 meses pós-parto
EDPS_antes_SN	Indicador de depressão antes do parto (a depender do valor de EDPS_antes)
1	Sim (se EDPS_antes >= 12)
0	Não (se EDPS_antes < 12)
EDPS_depois_SN	Indicador de depressão pós-parto (a depender do valor de EDPS_depois)
1	Sim (se EDPS_depois >= 12)
0	Não (se EDPS_depois < 12)
Grupo_amamentacao	Identificação do grupo de amamentação
1	Acompanhamento (mentorias e acompanhamento das amamentações nas primeiras semanas)
2	Orientações sobre a amamentação durante o pré-natal
3	Nenhum acompanhamento ou orientação
Tempo_amamentacao_meses	Tempo de amamentação (em meses) de pelo menos um recém-nascido

Figure 2.2: Variáveis da base de dados gestações gemelares - depressão e amamentação.

A seguir está o exemplo de como os dados de 5 indivíduos estão tabulados.

ID	EPDS_antes	EPDS_depois	EDPS_antes_SN	EDPS_depois_SN	Grupo_amamentacao	Tempo_ar
18	2	1	0	0	1	
19	6	3	0	0	2	
20	10	8	0	0	2	
21	15	18	1	1	2	
22	3	4	0	0	2	

Essa base de dados está disponível em <https://daslab-ufes.github.io/materiais/>, chamado de “Dados gemelares - depressão e amamentação”.

2.2 Sobre o software R

R é um ambiente computacional e uma linguagem de programação para manipulação, análise e visualização de dados. É considerado um dos melhores ambiente computacional para essa finalidade. O R é mantido pela R Development Core Team e está disponível para diferentes sistemas operacionais: Linux, Mac e Windows.

O software é livre, ou seja, gratuito, com código aberto em uma linguagem acessível. Nele estão implementadas muitas metodologias estatísticas. Muitas destas fazem parte do ambiente base de R e outras acompanham o ambiente sob a forma de pacotes, o que torna o R altamente expansível. Os pacotes são bibliotecas com dados e funções para diferentes áreas do conhecimento relacionado a estatística e áreas afins, devidamente documentados.

O R possui uma comunidade extremamente ativa, engajada desde o aprimoramento de ferramentas e desenvolvimento de novas bibliotecas, até o suporte aos usuários. Sobre o desenvolvimento de novas bibliotecas, um pesquisador em Estatística que desenvolve um novo modelo estatístico pode disponibilizar o seu modelo em um pacote acessível a que se interessam pelo modelo.

Além disso, a disponibilidade e compartilhamento da pesquisa em um pacote no R é uma boa prática quando falamos de reprodutibilidade na Ciência. Ainda nesse ponto, realizar as análises de uma pesquisa aplicada em um programa livre e acessível a todos é um dos principais pontos para permitir reprodutibilidade.

Ao optar por programar em R também implica na escolha de uma IDE (Integrated Development Environment) que, na grande maioria dos casos, será o RStudio. O RStudio é um conjunto de ferramentas integradas projetadas para editar e executar os códigos em R. Assim, quando for o interesse utilizar o R, só precisa abrir o RStudio (R é automaticamente carregado).

Para instalação do R e do RStudio, veja a Seção que segue.

2.3 Instalação R e RStudio

2.3.1 Instalação R

Nessa Seção, vamos apresentar como instalar o R e o RStudio para os três sistemas operacionais: Windows, MAC e Linux, respectivamente.

2.3.1.1 Para Windows

Os passos para instalar o R quando o sistema operacional é Windows são os seguintes:

- 1) Entre neste link para acessar a página do R e clique em Download, como no link destacado em retângulo vermelho na Figura 2.3. Note que o 3.6.1 é o número da versão mais recente disponível no momento da construção desse material (5/7/19).



Figure 2.3: Download R para Windows

- 2) Salve o arquivo de instalação em algum caminho de interesse do seu computador. Por exemplo, na Figura 2.4 mostra que a pasta é “Downloads”.

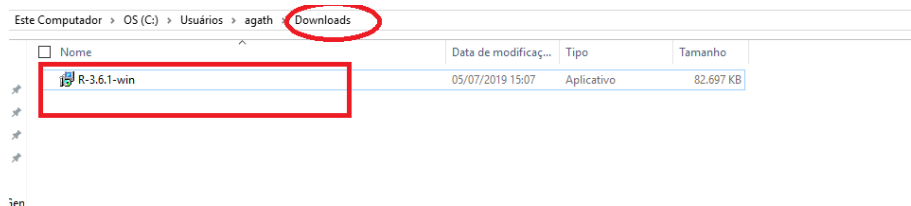


Figure 2.4: Instalador

- 3) Clique duas vezes com o botão esquerdo no instalador para iniciar a instalação. O próximo passo é escolher a língua para instalação. Na Figura 2.5 abaixo é português.
- 4) Clique em “Próximo” nas próximas janelas, como nas Figuras 2.6 a 2.11.

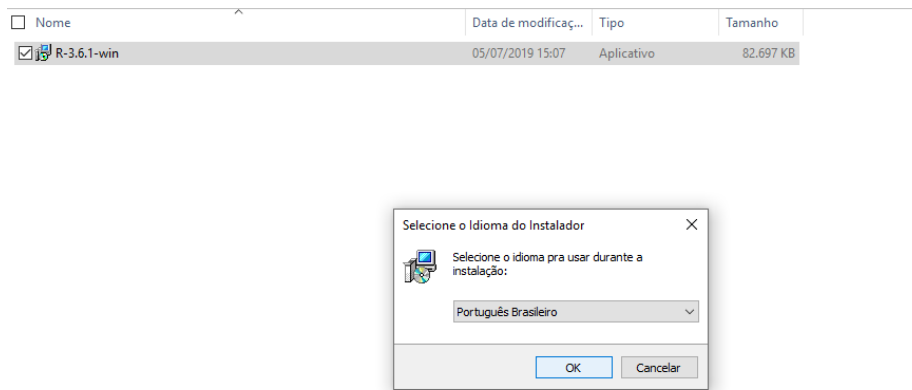


Figure 2.5: Escolha da lingua para instalação

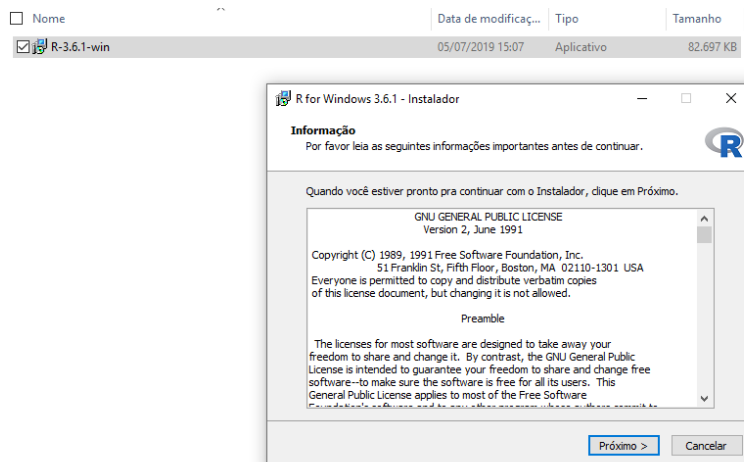


Figure 2.6: Próximo

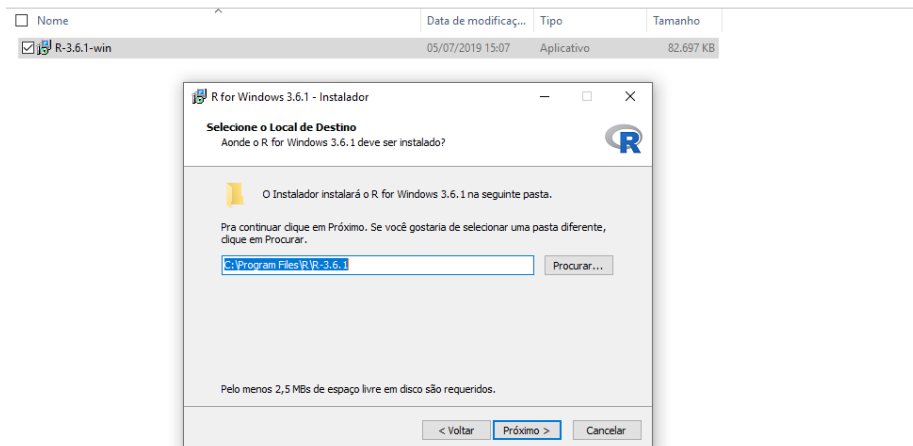


Figure 2.7: Próximo

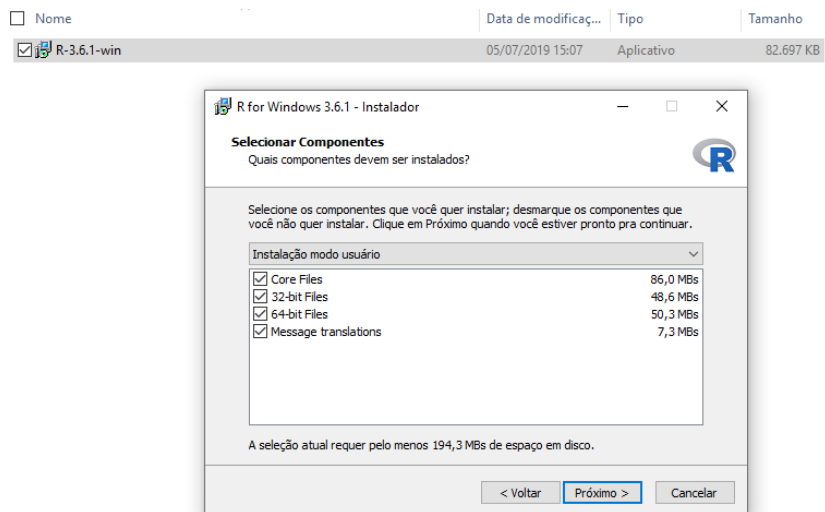


Figure 2.8: Próximo

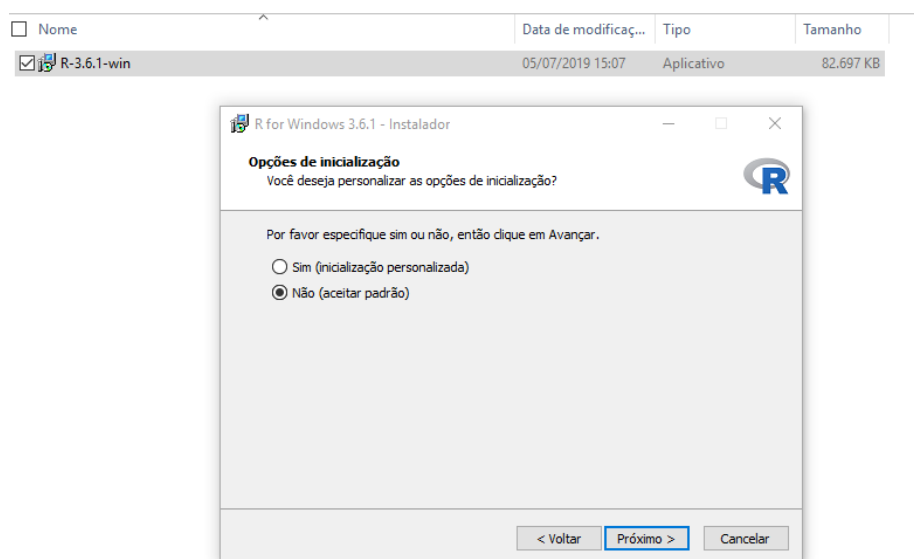


Figure 2.9: Próximo

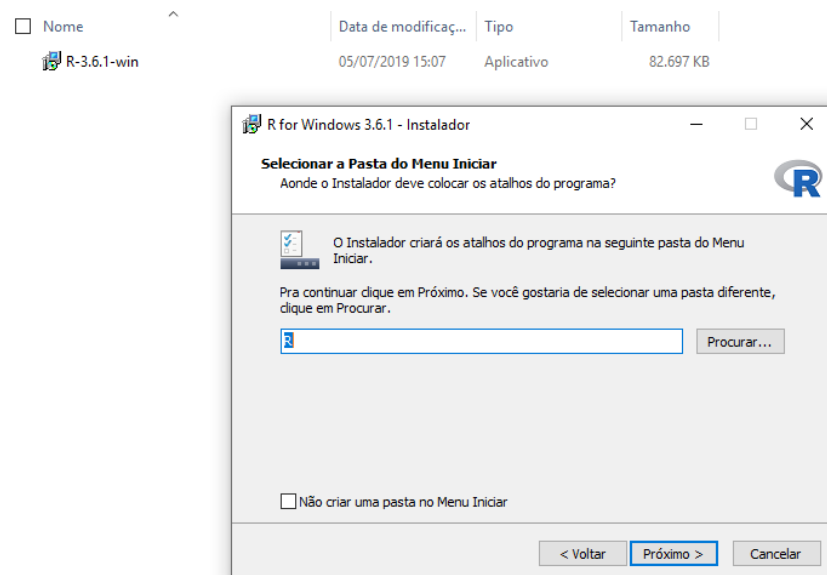


Figure 2.10: Próximo

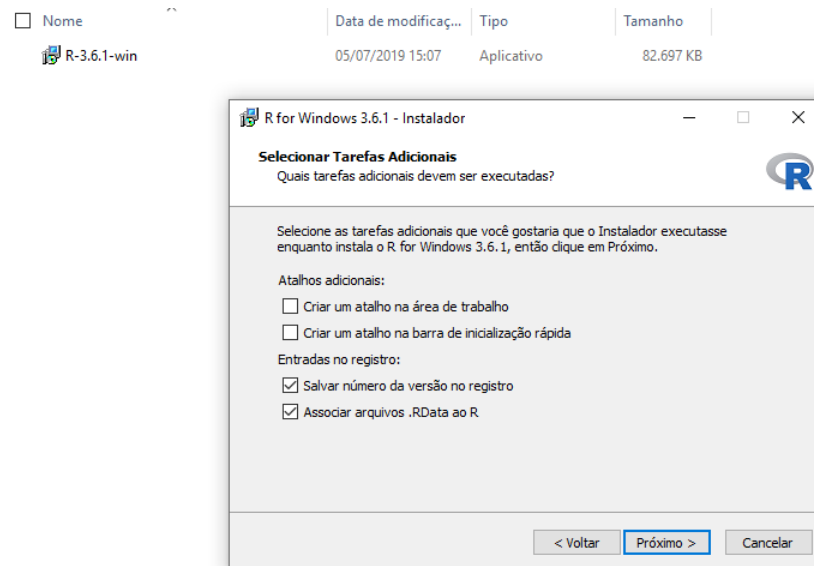


Figure 2.11: Próximo

- 5) Pronto, agora o software R será instalado, como na Figura 2.12, e quando terminar, aparecerá uma janela como apresentado na Figura 2.13.

2.3.1.2 Para MAC

Os passos para instalar o R quando o sistema operacional é OS X (Mac) são os seguintes:

- 1) Entre no site e clique em Download R for (MAC) OS X, conforme destacado abaixo em retângulo vermelho na Figura 2.14.
- 2) Baixe o pacote R-3.6.1.pkg clicando no link indicado no retângulo vermelho na Figura 2.15. Note que o 3.6.1 é o número da versão mais recente disponível no momento da confecção desse material.
- 3) Caso você não tenha configurado a pasta de downloads, o pacote será baixado na pasta “Downloads”, como mostrado na seguinte Figura 2.16. Observe que dois arquivos são baixados, clique duas vezes no arquivo “R-3.6.1.pkg” para abrir o assistente de instalação que o guiará durante o processo.
- 4) Acompanhe os passos indicados pelo instalador (Figura 2.17).
- 5) Deve concordar com os termos da licença, clique em “Agree” (Figura 2.18).

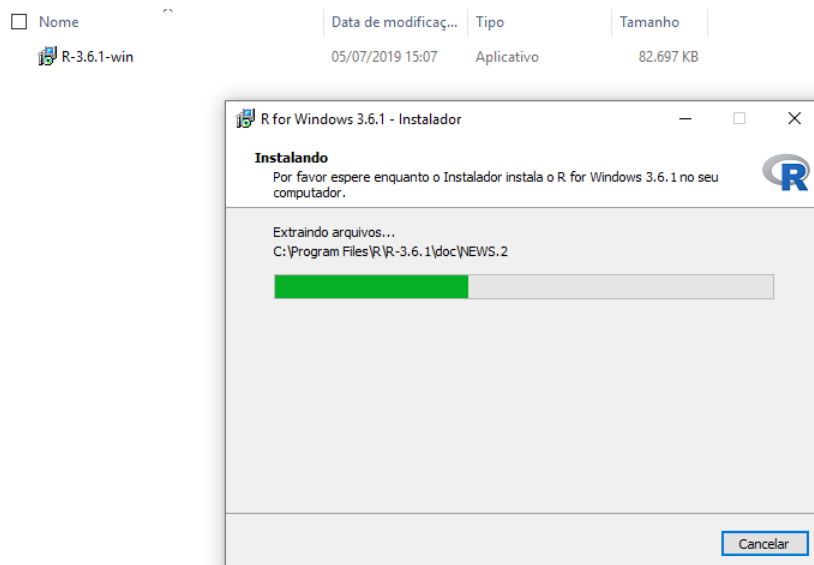


Figure 2.12: Instalação do R

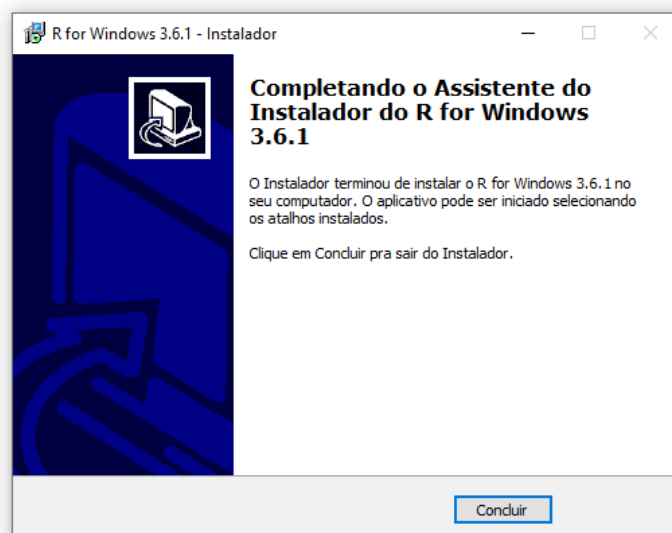


Figure 2.13: Pronto: R instalado



Figure 2.14: Download R para Mac



Figure 2.15: Download R para Mac

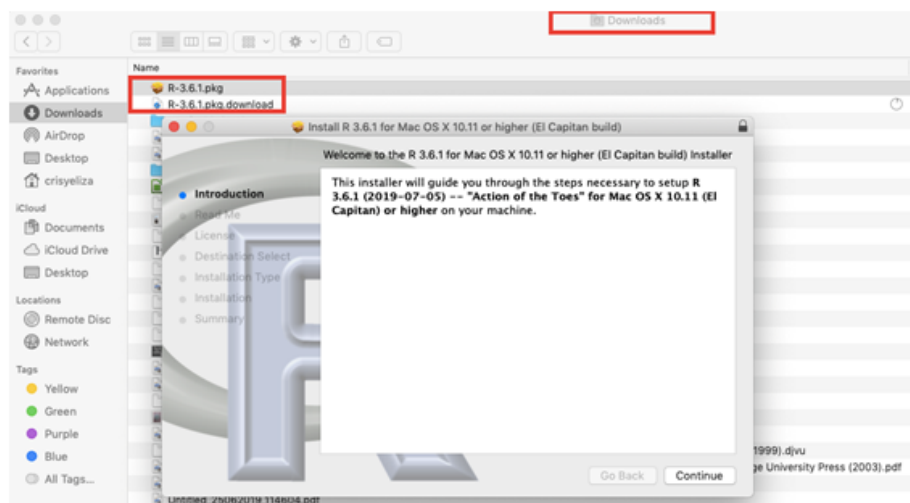


Figure 2.16: Pasta para instalação

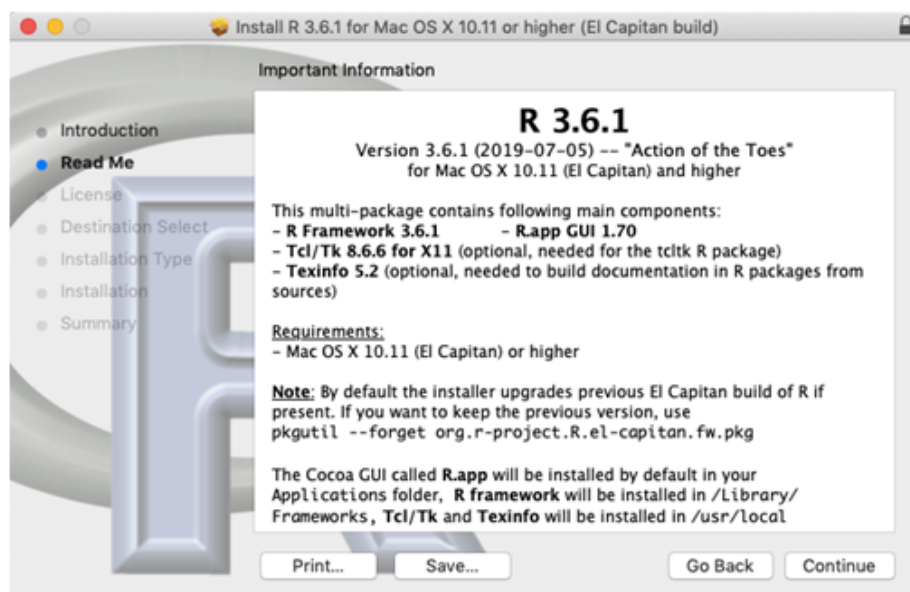


Figure 2.17: Instalação

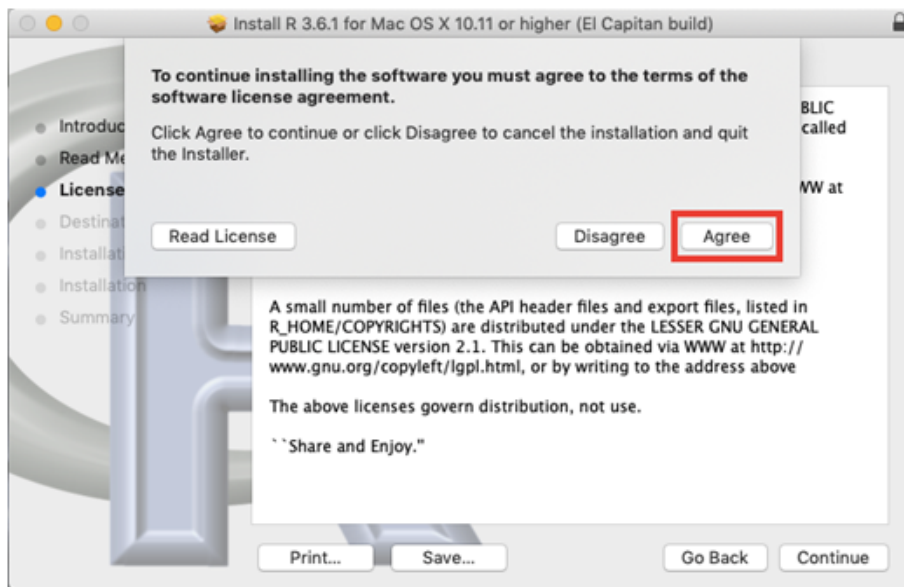


Figure 2.18: Instalação

- 6) Selecione o lugar onde instalará o programa, no caso de ter o disco particionado e assim desejar instalar em uma parte específica. Caso contrário, continue (Figura 2.19 e 2.20).
- 7) Para finalizar a instalação, o assistente lhe pedirá nome de usuário e senha do seu notebook, como apresentado na Figura 2.21.
- 8) Pronto, agora o software R será instalado, como na Figura 2.22, e quando terminar, aparecerá uma janela como apresentado na Figura 2.23.

2.3.1.3 Para Linux

A instalação do R no Linux depende da distribuição utilizada. Entre neste link para acessar a página do R e clique em Download R for Linux, como no link destacado em retângulo vermelho na Figura 2.24. Em seguida, clique no link referente à distribuição utilizada (Figura 2.25).

2.3.2 Instalação RStudio

O RStudio é um conjunto de ferramentas integradas projetadas (IDE - Integrated Development Environment) da linguagem R para auxiliar na produtividade ao utilizar o R.

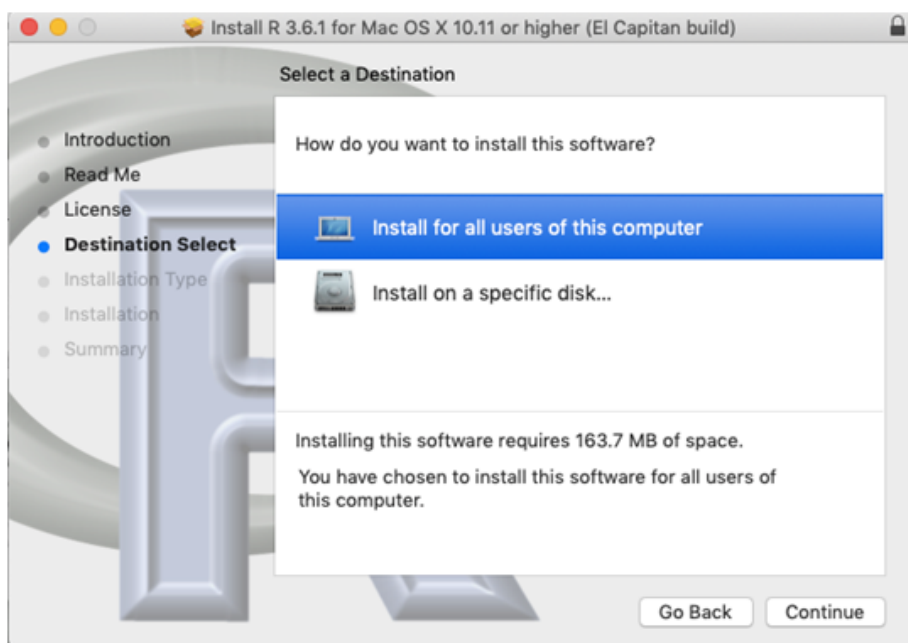


Figure 2.19: Instalação

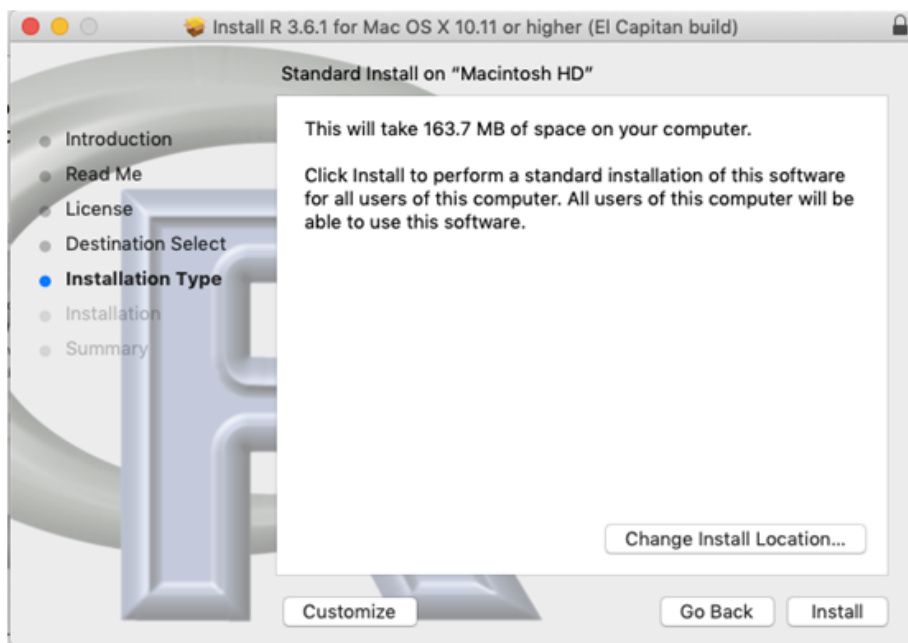


Figure 2.20: Instalação

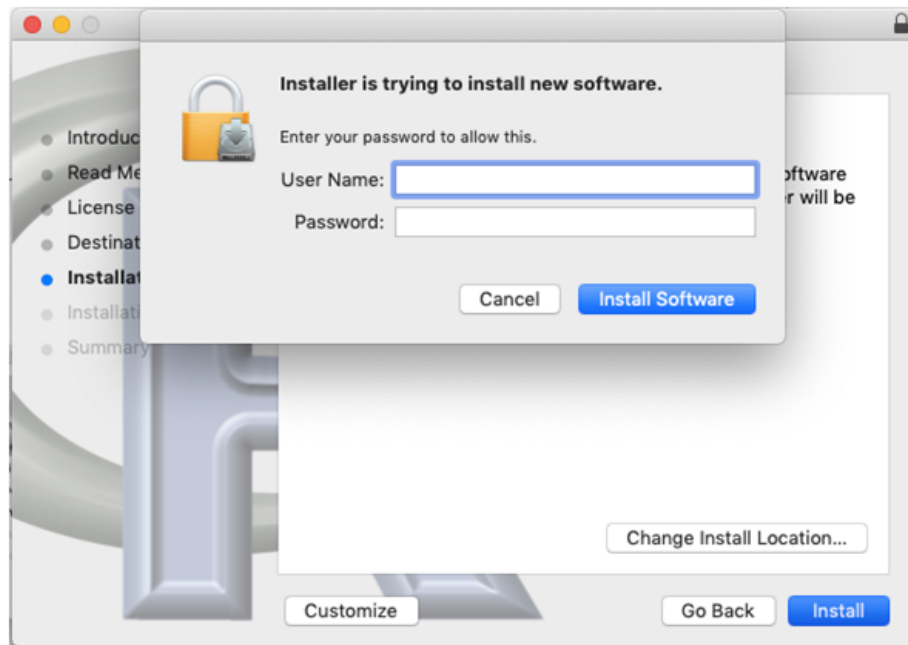


Figure 2.21: Instalação

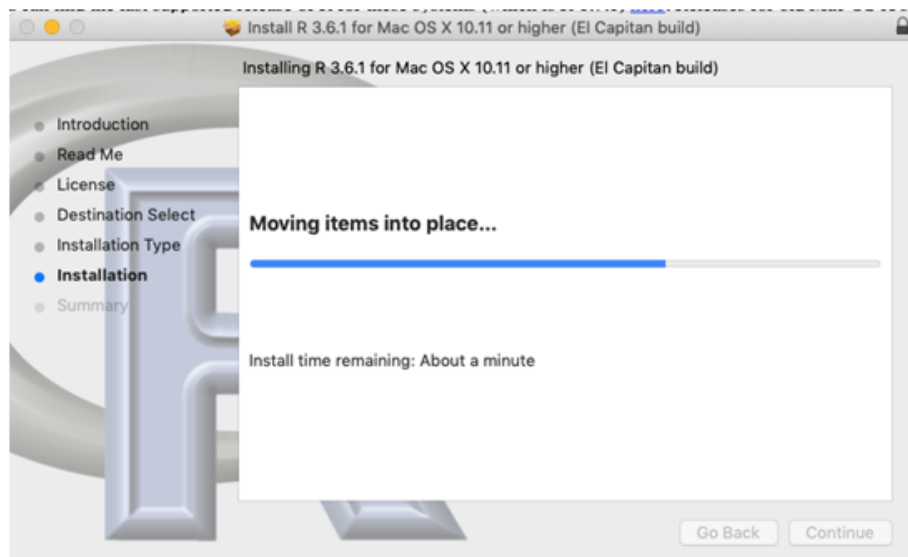


Figure 2.22: Instalação

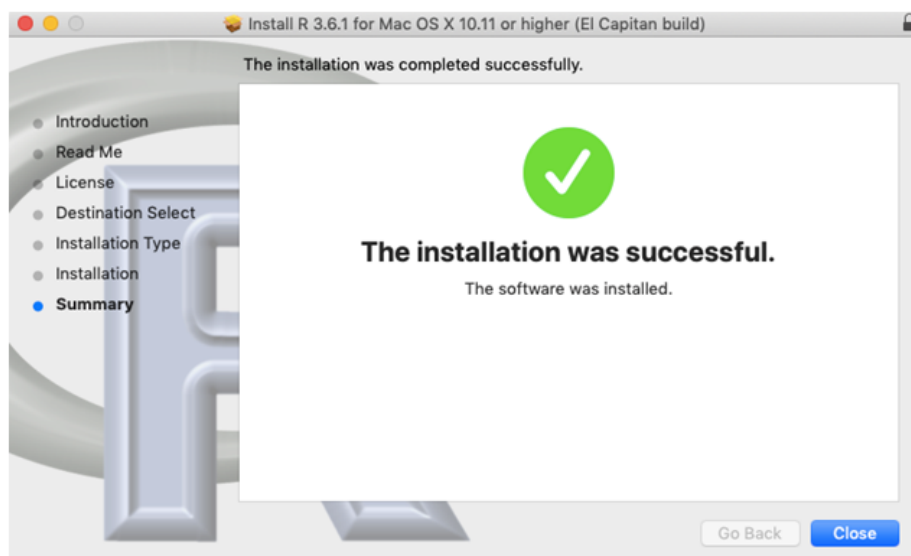


Figure 2.23: Instalação

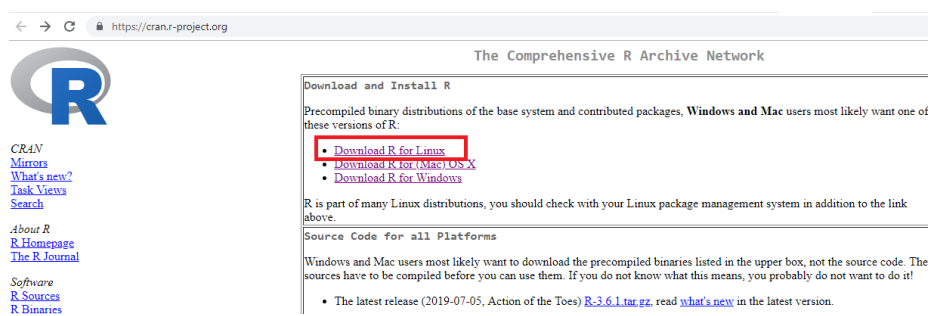


Figure 2.24: Download em Linux

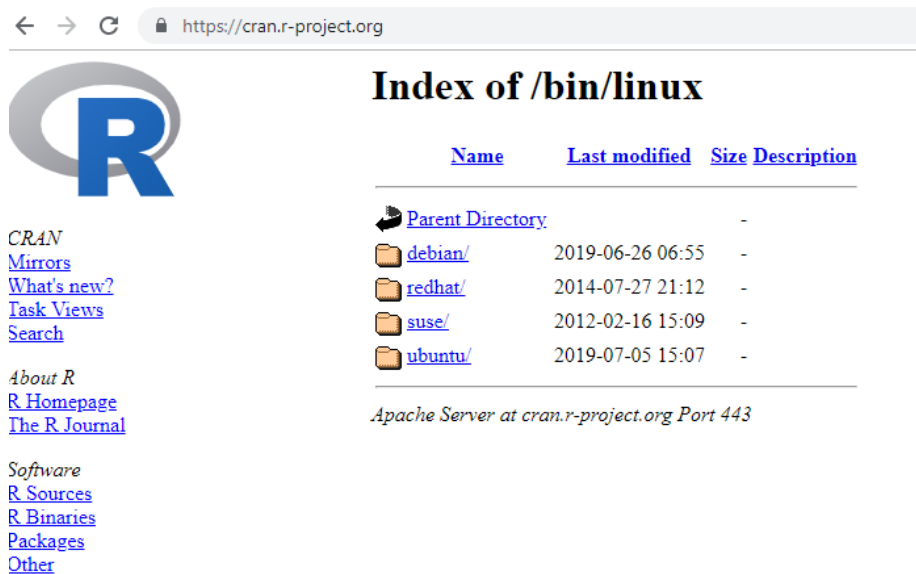


Figure 2.25: Download em Linux

2.3.2.1 Para Windows

- 1) Entre neste link e clique em Download como em destaque na Figura 2.26.
- 2) Clique no instalador em destaque na Figura 2.27.
- 3) Ao clicar no link, será feito o download do instalador e salvo na pasta de interesse. No caso da Figura 2.28, o instalador está na pasta Downloads. Dê dois cliques no botão esquerdo no arquivo para iniciar o download do arquivo.
- 4) Clique em “Próximo” nas próximas janelas e na última “Instalar”, como nas Figuras 2.29 a 2.31.
- 5) Pronto, a instalação será iniciada, como na Figura 2.32.

2.3.2.2 Para MAC

- 1) Entre neste link e clique em Download como em destaque na Figura 2.33.
- 2) Clique no instalador como destacado na Figura 2.34.
- 3) Ao clicar no link, será feito o download do instalador e salvo na pasta de interesse. Caso você não tenha configurado a pasta de downloads, o instalador ficará na pasta “Downloads”, como na Figura 2.35.
- 4) Clicando duas vezes no arquivo “RStudio-1.2.1335.dmg” (versões mais atual do RStudio), será feita a descarga do mesmo abrindo a janela conforme

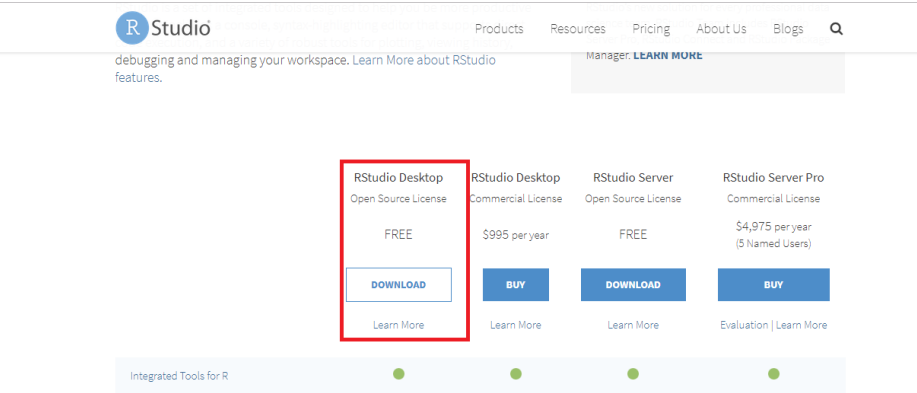


Figure 2.26: Site para download do RStudio

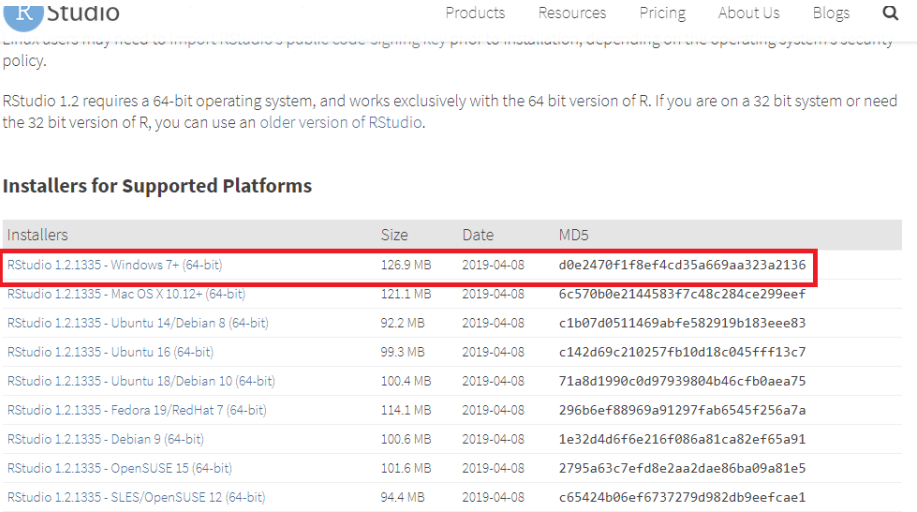


Figure 2.27: Link para download do RStudio

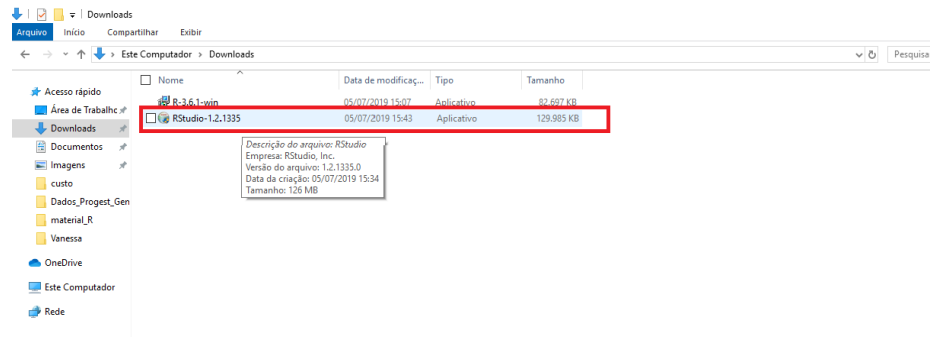


Figure 2.28: Instalador

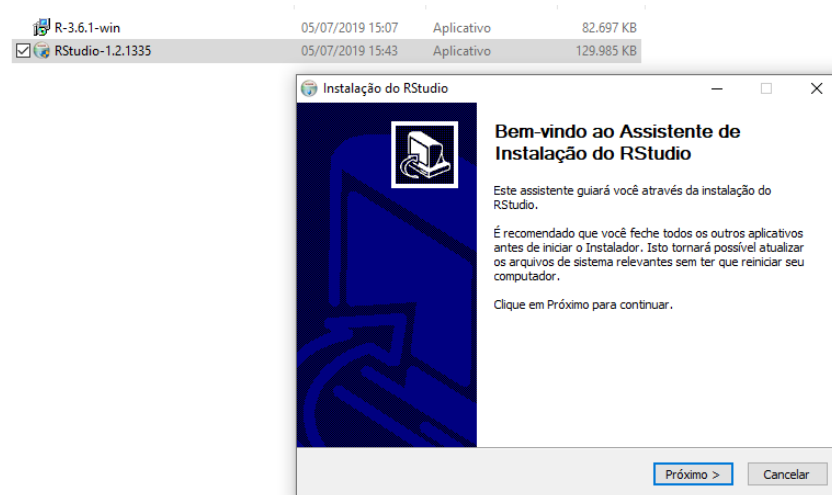


Figure 2.29: Instalação

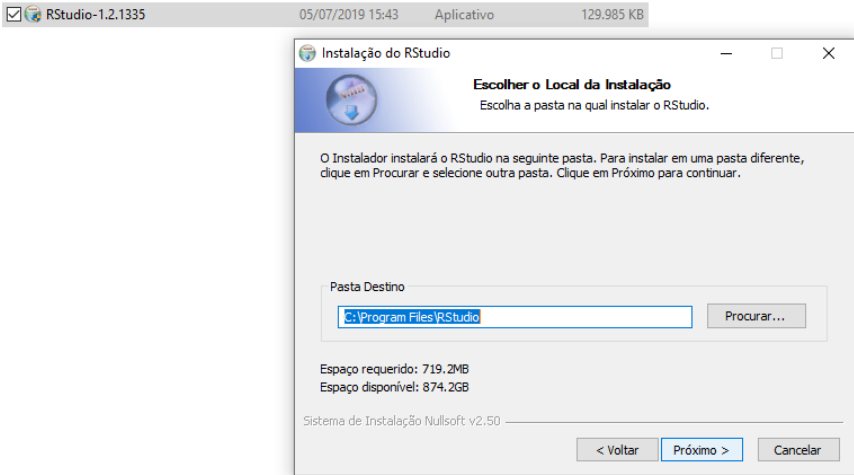


Figure 2.30: Instalação

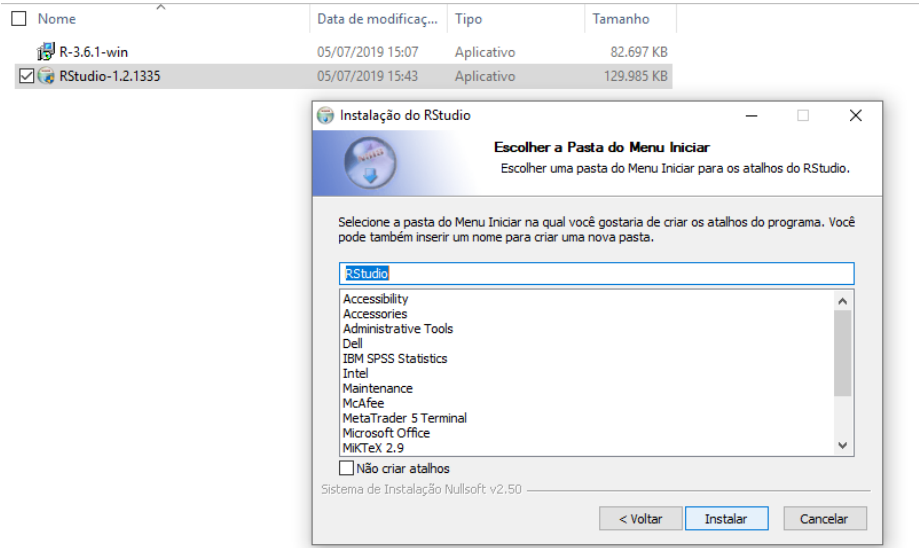


Figure 2.31: Instalação

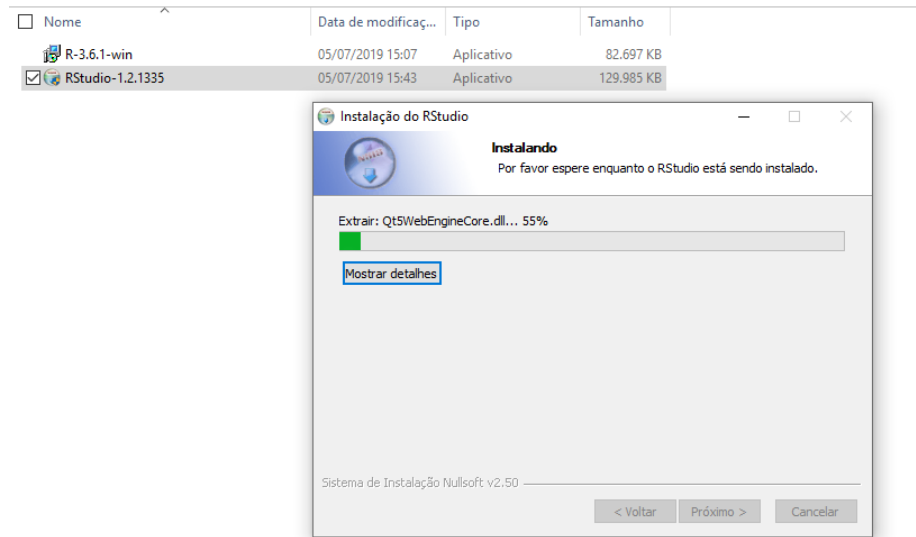


Figure 2.32: Instalação

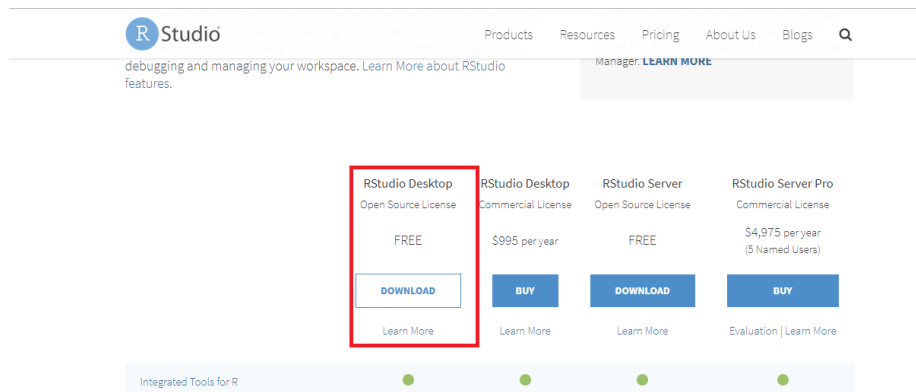
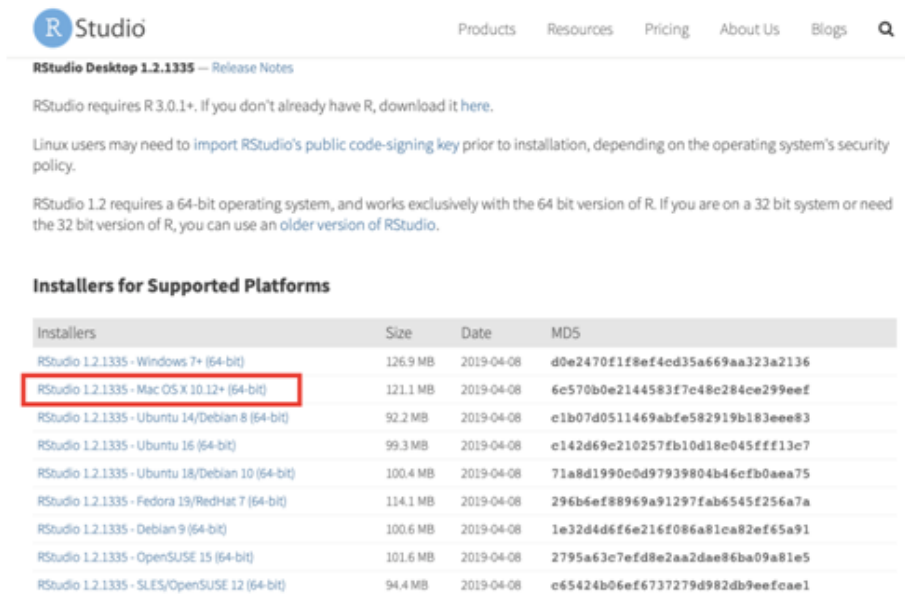


Figure 2.33: Site para download do RStudio



RStudio Desktop 1.2.1335 — Release Notes

RStudio requires R 3.0.1+. If you don't already have R, download it [here](#).

Linux users may need to [import RStudio's public code-signing key](#) prior to installation, depending on the operating system's security policy.

RStudio 1.2 requires a 64-bit operating system, and works exclusively with the 64 bit version of R. If you are on a 32 bit system or need the 32 bit version of R, you can use an [older version of RStudio](#).

Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.2.1335 - Windows 7+ (64-bit)	126.9 MB	2019-04-08	d0e2470f1f8ef4cd35a669aa323a2136
RStudio 1.2.1335 - Mac OS X 10.12+ (64-bit)	121.1 MB	2019-04-08	6e570b0e2144583f7c48c284ce299eef
RStudio 1.2.1335 - Ubuntu 14/Debian 8 (64-bit)	92.2 MB	2019-04-08	e1b07d0511469abfe582919b183eee83
RStudio 1.2.1335 - Ubuntu 16 (64-bit)	99.3 MB	2019-04-08	e142d69e210257fb10d18e045fff13e7
RStudio 1.2.1335 - Ubuntu 18/Debian 10 (64-bit)	100.4 MB	2019-04-08	71a8d1990c0d97939804b46cfeb0aa75
RStudio 1.2.1335 - Fedora 19/RedHat 7 (64-bit)	114.1 MB	2019-04-08	296b6ef88969a91297fab6545f256a7a
RStudio 1.2.1335 - Debian 9 (64-bit)	100.6 MB	2019-04-08	1e32d4d6f6e216f086a81ca82ef65a91
RStudio 1.2.1335 - OpenSUSE 15 (64-bit)	101.6 MB	2019-04-08	2795a63c7efd8e2aa2dae86ba09a81e5
RStudio 1.2.1335 - SLES/OpenSUSE 12 (64-bit)	94.4 MB	2019-04-08	c65424b06ef6737279d982db9eefcae1

Figure 2.34: Site para download do RStudio para Mac

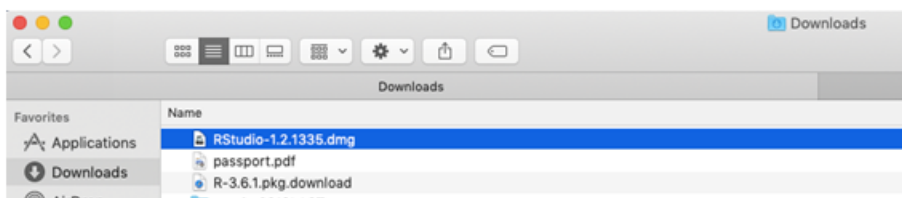


Figure 2.35: Instalador salvo em pasta

na Figura 2.36. Clique no aplicativo de RStudio destacado em vermelho também na Figura 2.36.



Figure 2.36: Instalação

- 5) O instalador pode perguntar se está seguro que o aplicativo será baixado da internet e clique em “Open” (Figura 2.37).

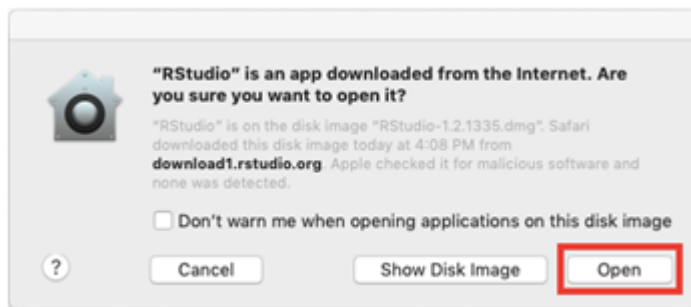


Figure 2.37: Instalação

- 6) Pronto! Imediatamente abre o RStudio, como na Figura 2.38, e você já pode utilizar.

2.3.2.3 Para Linux

- 1) Entre neste link e clique em Download como em destaque na Figura 2.39.
- 2) Clique no link referente à distribuição utilizada (Figura 2.40).

2.4 Primeiros passos com R e RStudio.

2.4.1 Primeiros contatos com RStudio

O RStudio é um conjunto de ferramentas integradas projetadas (IDE - Integrated Development Environment) da linguagem R para editar e executar os códigos em R.

Tem quatro áreas, conforme a Figura 2.41.

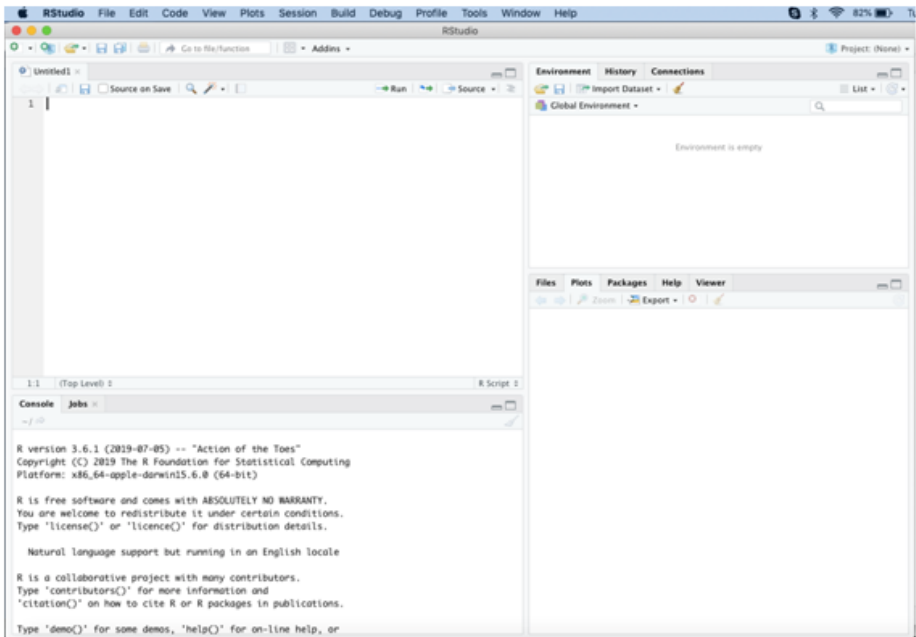


Figure 2.38: Instalação

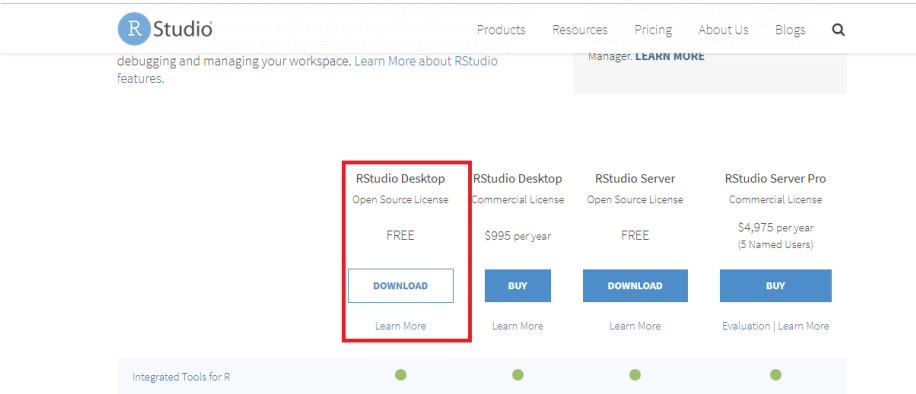


Figure 2.39: Site para download do RStudio

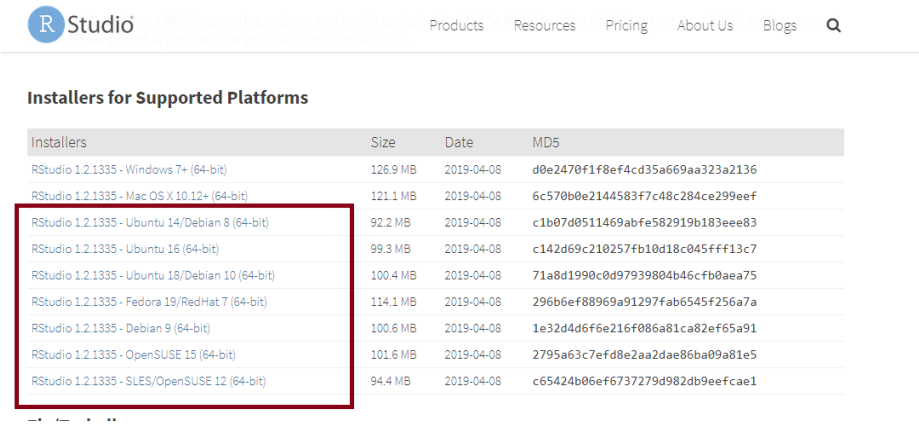


Figure 2.40: Download do RStudio

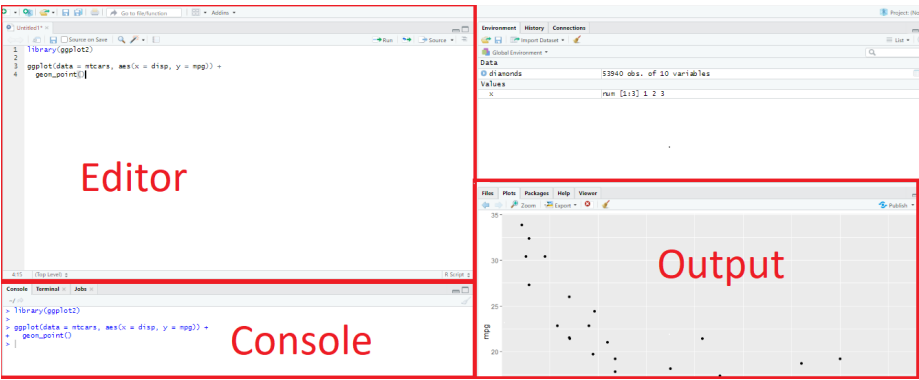


Figure 2.41: Visualização do RStudio

A seguir descrevemos melhor os painéis e abas do RStudio:

- Editor/Scripts: É onde escrever os códigos. Arquivos do tipo .R.
- Console: Executar os comandos e ver os resultados.
- Enviroment: Painel com todos os objetos criados.
- History: História dos comandos executados.
- Files: Navegar em pastas e arquivos.
- Plots: Onde os gráficos serão apresentados.
- Packages: Pacotes instalados (sem ticar) e habilitados (ticados).
- Help: Retorna o tutorial de ajuda do comando solicitado com `help()` ou `?comando`. Ver melhor como pedir ajuda no R no final desse capítulo.

O usuário pode alterar a aparência do RStudio, como fonte e cor. Como exemplo, as Figuras 2.42 e 2.43 apresentam os passos para mudar o tema do script, no exemplo, deixar com fundo preto.

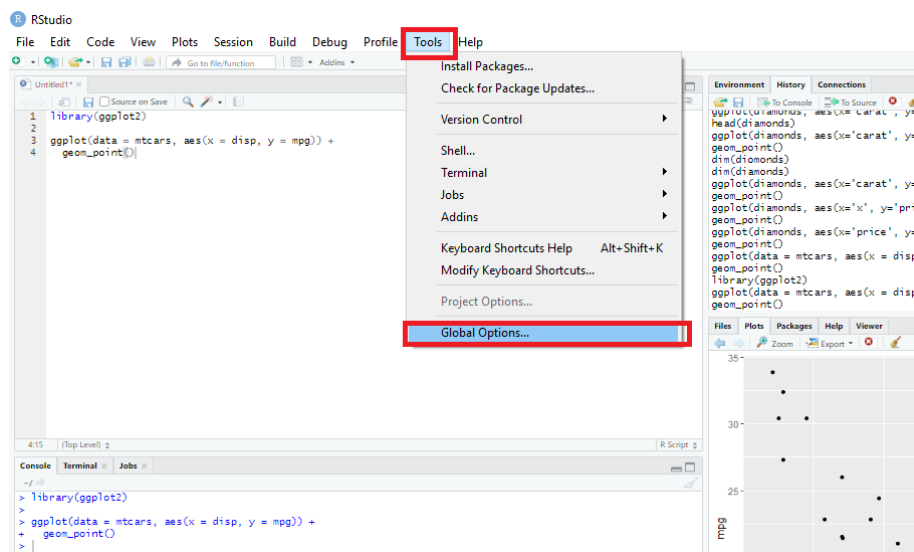
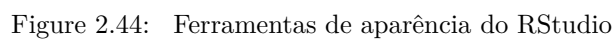
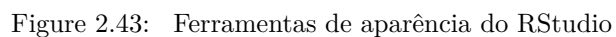


Figure 2.42: Ferramentas de aparência do RStudio

Ainda no menu `Tools` → `Global Options` → `Pane Layout`, o usuário pode organizar a ordem dos quadrantes do RStudio, como apresentado nas Figuras 2.44, 2.45 e 2.46. No exemplo, o painel console foi transferido para o lado do painel Script, o que facilita a visualização dos comandos rodados.



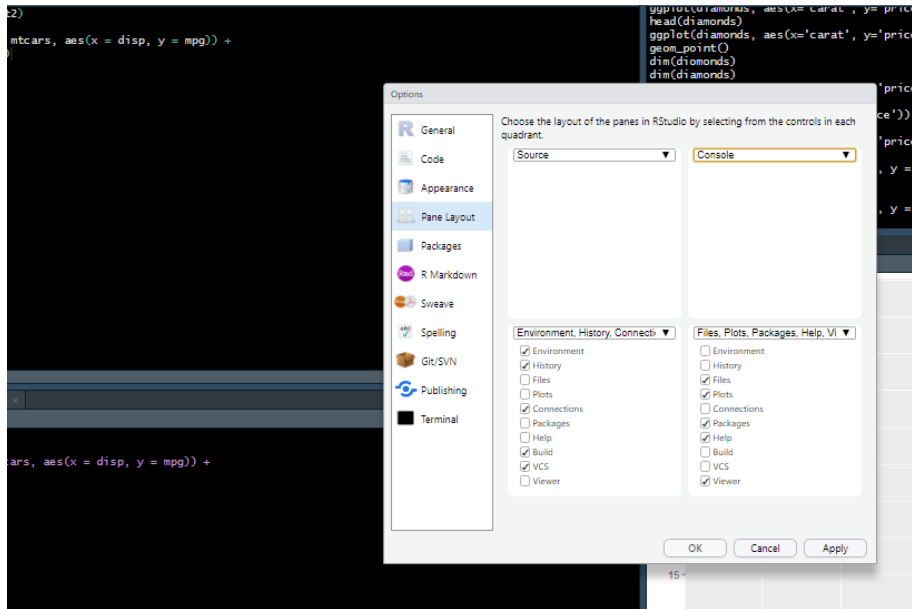


Figure 2.45: Ferramentas de aparência do RStudio

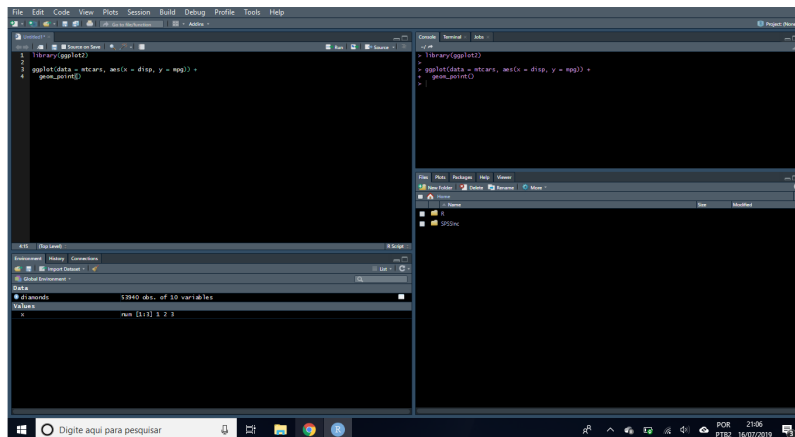


Figure 2.46: Ferramentas de aparência do RStudio

2.4.1.1 Projetos

Uma funcionalidade importante é a criação de projetos, permitindo dividir o trabalho em múltiplos ambientes, cada um com o seu diretório, documentos e workspace.

Para criar um projeto, os seguintes passos podem ser seguidos:

- 1) Clique na opção “File” do menu, e então em “New Project”.
- 2) Clique em “New Directory”.
- 3) Clique em “New Project”.
- 4) Escreva o nome do diretório (pasta) onde deseja manter seu projeto, ex “my_project”.
- 5) Clique no botão “Create Project”.

Para criar um novo script para escrever os códigos, vá em File -> New File -> R Script

2.4.1.2 Boas práticas

Comente bem o seu código: É possível fazer comentários usando o símbolo ‘#’. É sempre bom explicar o que uma variável armazena, o que uma função faz, porque alguns parâmetros são passados para uma determinada função, qual é o objetivo de um trecho de código, etc.

Evite linhas de código muito longas: Usar linhas de código mais curtas ajuda na leitura do código.

Escreva um código organizado. Por exemplo, adote um padrão no uso de minúsculas e maiúsculas, uma lógica única na organização de pastas e arquivos, pode ser adotada uma breve descrição (como comentário) indicando o que um determinado script faz.

Carregue todos os pacotes que irá usar sempre no início do arquivo: Quando alguém abrir o seu código será fácil identificar quais são os pacotes que devem ser instalados e quais dependências podem existir.

2.4.2 Primeiros passos no R

Posso escrever o código no Script e submeter ao apertar o botão “Run” ou com o atalho no teclado Cmd/Ctrl+Enter.

2.4.2.1 R como calculadora

- 1) Operadores

```
#adição  
10+15
```

```
## [1] 25
```

```
#subtração  
10-2
```

```
## [1] 8
```

```
#multiplicação  
2*10
```

```
## [1] 20
```

```
#divisão  
30/2
```

```
## [1] 15
```

```
#raiz quadrada  
sqrt(4)
```

```
## [1] 2
```

```
#potência  
2^2
```

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

Se você digitar um comando incompleto, como `10 *`, o R mostrará um `+`. Isso não tem a ver com a soma e apenas que o R está esperando você completar seu comando. Termine seu comando ou aperte `Esc` para recomençar. Vale também ressaltar que se você digitar um comando que o R não reconhece, ele retornará uma mensagem de erro e você pode digitar outro comando normalmente em seguida.

2.4.2.2 Atribuição

Podemos salvar valores dentro de um objeto, que é simplesmente um nome que guarda um valor, vetor, matriz, lista ou base de dados.

Para atribuir a um objeto, o sinal de atribuição é `=` ou `<-` (preferível).

Exemplos:

```
x <- 10/2  
x
```

```
## [1] 5
```

```
X
```

```
## Error in eval(expr, envir, enclos): objeto 'X' não encontrado
```

Por que tivemos um erro acima?

O R é case sensitive, isto é, faz a diferenciação entre as letras minúsculas e maiúsculas. Portanto, x é diferente de X.

2.4.2.3 Objetos em R

Existem cinco classes básicas no R:

- character: “UAH!”
- numeric: 0.95 (números reais)
- integer: 100515 (inteiros)
- complex: $2 + 5i$ (números complexos, $a + bi$)
- logical: TRUE (booleanos, TRUE/FALSE)

Vamos atribuir a x a string banana.

```
x <- banana

## Error in eval(expr, envir, enclos): objeto 'banana' não encontrado
x <- "banana"
x

## [1] "banana"
```

O primeiro caso (`x <- banana`) não deu certo, pois ele entendeu que estamos atribuindo a x outro objeto banana, que não foi declarado. Para atribuir o string banana à x, precisamos colocar entre aspas ou aspas simples. Uma string sem aspas é entendido como um objeto, veja abaixo:

```
banana <- 30
x <- banana
x
```

```
## [1] 30
```

Para saber a classe de um objetivo, use a função `class()`.

```
y <- "ola"
class(y)
```

```
## [1] "character"
```

```
x <- 2.5
class(x)
```

```
## [1] "numeric"
```

2.4.2.4 Apagar objetos

E se eu quiser apagar um objeto?

```
x <- 20
x
```

```
## [1] 20
```

```
remove(x)
x
```

```
## Error in eval(expr, envir, enclos): objeto 'x' não encontrado
```

E se eu quiser limpar o console - apaga todos os objetos atribuídos até aqui:

```
rm(list=ls())
```

2.4.2.5 Vetor

Como atribuir vários valores a um objeto? Para entrar com vários números (ou nomes, ou qualquer outro grupo de coisas), precisamos usar uma função para dizer ao programa que os valores serão combinados em um único vetor.

```
x <- c(2,3,4)
x
```

```
## [1] 2 3 4
```

```
y <- seq(1,10)
y
```

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

```
z <- rep(1,10)
z
```

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

```
a <- 1:10
a
```

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

```
bicho <-c("macaco","pato","galinha","porco")
bicho
```

```
## [1] "macaco" "pato" "galinha" "porco"
```

E se quisermos visualizar o conteúdo da posição 2 no vetor bicho?

```
bicho[2]
```

```
## [1] "pato"
```

As operações vetoriais podem ser realizadas de maneira bastante intuitiva. Como exemplos:

```
x <- c(2,3,4)
x
```

```
## [1] 2 3 4
```

```
ops <- x-1
ops
```

```
## [1] 1 2 3
```

```
k <- x*2
k
```

```
## [1] 4 6 8
```

Vamos agora considerar um vetor de pesos em kg e altura em metros de 6 pessoas.

```
peso <- c(62, 70, 52, 98, 90, 70)
peso
```

```
## [1] 62 70 52 98 90 70
```

```
altura <- c(1.70, 1.82, 1.75, 1.94, 1.84, 1.61)
altura
```

```
## [1] 1.70 1.82 1.75 1.94 1.84 1.61
```

Vale mencionar que o separador de decimais no R é . (ponto)!

Como calcularia o IMC? Lembrando que o IMC é dado pelo peso (em kg) dividido pela altura (em metros) ao quadrado.

```
imc <- peso/(altura^2)
imc
```

```
## [1] 21.45329 21.13271 16.97959 26.03890 26.58318 27.00513
```

Para saber o tamanho do vetor, use a função `length()`.

```
length(imc)
```

```
## [1] 6
```

2.4.2.6 Matrizes

Matrizes são vetores numéricos com duas dimensões, que são simplesmente a linha e a coluna às quais o elemento pertence.

```
x <- matrix(seq(1,16), nrow=4,ncol=4)
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
```



```
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
```

Note que os números de 1 a 16 foram dispostos na matriz coluna por coluna ou seja, preenchendo de cima para baixo e depois da esquerda para a direita.

Como sei qual elemento está na segunda linha e terceira coluna da matriz x?

```
x[2,3]
```

```
## [1] 10
```

```
x[3, ] # seleciona a 3ª linha
```

```
## [1]  3  7 11 15
```

```
x[, 2] # seleciona a 2ª coluna
```

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

```
x[1, 2] # seleciona o elemento da primeira linha e segunda coluna
```

```
## [1] 5
```

E se eu quiser substituir a primeira linha por (13,15,19,30)?

```
x[1,] <- c(13,15,19,30)
```

```
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   13   15   19   30
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
```

Seja o vetor d

```
d <- c(128,124,213,234)
```

E se quisermos substituir a terceira coluna por d?

```
x[,3] <- d
```

Qual a dimensao da matriz x?

Vimos que para vetor usamos o comando length(). Serve para matriz? Vamos testar:

```
length(x)
```

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

Note que retorna o número de colunas vezes o número de linhas ($4*4=16$). Mas o que quero saber é o número de linhas e de colunas. Para isso, o comando é `dim()`.

```
dim(x)
```

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

Para concatenar linhas em uma matriz, podemos usar o comando `rbind()`:

```
vet <- c(2,20,12,34)
x2 <- rbind(x,vet)
x2
```

```
##      [,1] [,2] [,3] [,4]
##      13   15  128   30
##       2    6  124   14
##       3    7  213   15
##       4    8  234   16
## vet    2   20   12   34
```

Para concatenar colunas em uma matriz, podemos usar o comando `cbind()`:

```
v2 <- c(25,10,15,4)
x3 <- cbind(x,v2)
x3
```

```
##              v2
## [1,] 13 15 128 30 25
## [2,]  2  6 124 14 10
## [3,]  3  7 213 15 15
## [4,]  4  8 234 16  4
```

2.4.2.7 Fator

Fatores podem ser vistos como vetores de inteiros que possuem rótulos (labels). Eles são úteis para representar uma variável categórica (nominal e ordinal).

```
sexo <- c("M", "H", "H", "H", "M", "M", "H")
sex <- as.factor(sexo)
sex
```

```
## [1] M H H H M M H
## Levels: H M
```

```
levels(sex)
```

```
## [1] "H" "M"
```

2.4.2.8 Data frame

Trata-se de uma “tabela de dados” onde as colunas são as variáveis e as linhas são os registros. Essas colunas podem ser de classes diferentes. Essa é a grande diferença entre `data.frame`'s e matrizes (matriz é só numérica).

Posso criar um data frame no R com os vetores, por exemplo:

```
ID <- seq(1,6)
pes <- c(62, 70, 52, 98, 90, 70)
alt <- c(1.70, 1.82, 1.75, 1.94, 1.84, 1.61)
imc <- pes/(alt^2)
dados <- data.frame(ID=ID,peso=pes,altura=alt, imc=imc)
dados
```

```
##   ID peso altura    imc
## 1  1   62   1.70 21.45329
## 2  2   70   1.82 21.13271
## 3  3   52   1.75 16.97959
## 4  4   98   1.94 26.03890
## 5  5   90   1.84 26.58318
## 6  6   70   1.61 27.00513
```

Posso pensar que o data frame tem a mesma ideia de matriz. Quero olhar os dados de altura. Sei que altura está na coluna 3.

```
dados[,3]
```

```
## [1] 1.70 1.82 1.75 1.94 1.84 1.61
```

Mas existe uma maneira mais fácil de selecionar a variável de interesse sem ter que saber em qual coluna ela está. Por ser um data frame, posso usar `$` da seguinte maneira:

```
dados$altura
```

```
## [1] 1.70 1.82 1.75 1.94 1.84 1.61
```

Putz, esqueci de colocar a variável de grupo no data frame. Tenho que criar tudo de novo? Não:

```
gr <- c(rep(1,3),rep(2,3))
dados$grupo <- gr
```

```
dados
```

```
##   ID peso altura    imc grupo
## 1  1   62   1.70 21.45329     1
## 2  2   70   1.82 21.13271     1
## 3  3   52   1.75 16.97959     1
## 4  4   98   1.94 26.03890     2
```

```
## 5 5 90 1.84 26.58318 2
## 6 6 70 1.61 27.00513 2
```

Veja que no “dados\$grupo” foi inserido o objeto “gr”. Se “gr” não tivesse o mesmo número de linhas do data frame retornaria um erro.

Funções úteis para data.frame:

Ainda não falamos com muito detalhes sobre funções no R, faremos isso mais adiante. Mas por enquanto, considere que sejam nomes já salvos no R e que, ao colocar o objeto da base de dados (no nosso exemplo é **dados**) dentro dos parênteses, retorna algumas informações úteis sobre a base de dados. São algumas delas:

- `head()` - Mostra as primeiras 6 linhas.
- `tail()` - Mostra as últimas 6 linhas.
- `dim()` - Número de linhas e de colunas.
- `names()` - Os nomes das colunas (variáveis).
- `str()` - Estrutura do data.frame. Mostra, entre outras coisas, as classes de cada coluna.

```
head(dados)
```

```
## ID peso altura imc grupo
## 1 1 62 1.70 21.45329 1
## 2 2 70 1.82 21.13271 1
## 3 3 52 1.75 16.97959 1
## 4 4 98 1.94 26.03890 2
## 5 5 90 1.84 26.58318 2
## 6 6 70 1.61 27.00513 2
```

```
dim(dados)
```

```
## [1] 6 5
```

```
names(dados)
```

```
## [1] "ID" "peso" "altura" "imc" "grupo"
```

```
str(dados)
```

```
## 'data.frame': 6 obs. of 5 variables:
## $ ID : int 1 2 3 4 5 6
## $ peso : num 62 70 52 98 90 70
## $ altura: num 1.7 1.82 1.75 1.94 1.84 1.61
## $ imc : num 21.5 21.1 17 26 26.6 ...
## $ grupo : num 1 1 1 2 2 2
```

2.4.2.9 Operadores lógicos

A operação lógica nada mais é do que um teste que retorna verdadeiro (**TRUE**) ou falso (**FALSE**). Esses valores dois valores recebem uma classe especial: **logical**.

- Igual a: `==`

Vamos ao testar se um valor é igual ao outro.

Exemplo:

```
10==11
```

```
## [1] FALSE
```

```
11==11
```

```
## [1] TRUE
```

No primeiro retornou **FALSE**, pois realmente 10 não é igual a 11 e no segundo caso acima retornou **TRUE**, pois realmente 11 é igual a 11. De maneira análoga funciona para os operadores abaixo:

- Diferente de: `!=`

Exemplo:

```
10!=11
```

```
## [1] TRUE
```

- Maior que: `>`
- Maior ou igual: `>=`
- Menor que: `<`
- Menor ou igual: `<=`

Exemplos:

```
10>5
```

```
## [1] TRUE
```

```
10>=10
```

```
## [1] TRUE
```

```
4<4
```

```
## [1] FALSE
```

```
4<=4
```

```
## [1] TRUE
```

- Um outro operador muito útil é o `%in%`. Com ele, podemos verificar se um valor está dentro de um vetor.

```
ex <- 1:15
3 %in% ex
```

```
## [1] TRUE
```

- E: & - será verdadeiro se os dois forem TRUE

```
x <- 15
x > 10 & x < 30
```

```
## [1] TRUE
```

```
x < 10 & x < 30
```

```
## [1] FALSE
```

- OU: | - será verdadeiro se um dos dois forem TRUE

```
x <- 15
x > 10 | x < 30
```

```
## [1] TRUE
```

```
x < 10 | x < 30
```

```
## [1] TRUE
```

- Negação: !

```
x <- 15
!x < 30
```

```
## [1] FALSE
```

2.4.2.10 Dados faltantes, infinitos e indefinições matemáticas

- NA (Not Available): dado faltante/indisponível. Exemplo:

```
x <- c(1,6,9)
x[4]
```

```
## [1] NA
```

retornou NA porque não há elemento na posição 4 do vetor x.

- NaN (Not a Number): indefinições matemáticas. Como 0/0 e log(-1). Exemplo:

```
log(-10)
```

```
## [1] NaN
```

- Inf (Infinito): número muito grande ou o limite matemático. Aceita sinal negativo -Inf. Exemplo:

```
10^14321
```

```
## [1] Inf
```

2.4.2.11 Condicionamento : If e else

As estruturas if e else servem para executar um código apenas se uma condição (teste lógico) for satisfeita.

```
a <- 224
b <- 225
if (a==b) {
  v <- 10
} else {
  v <- 15
}
v
```

```
## [1] 15
```

Veja que o R só executa o conteúdo das chaves {} se a expressão dentro dos parênteses () retornar TRUE.

Note que a condição de igualdade é representada por dois iguais ==. Como dito anteriormente, apenas um igual = é símbolo de atribuição (preferível <-).

Veja outro exemplo:

```
a <- 224
b <- 225
if (a==b) {
  v <- 10
} else if (a > b) {
  v <- 15
} else {
  v <- 25
}
v
```

```
## [1] 25
```

Veja que nesse exemplo gostaria de usar mais de duas condições e por isso usamos a estrutura intermediária else if.

2.4.2.12 Iterador for

O for serve para repetir uma mesma tarefa para um conjunto de valores diferentes. Cada repetição é chamada de iteração.

Como exemplo, considere o vetor atribuído ao objeto m como segue:

```
m <- c(1,20,50,60,100)
```

Quero criar um novo vetor, p digamos, que seja formado por cada elemento de m dividido por sua posição.

```
p <- NULL
for (i in 1:length(m)){
  p[i] <- m[i]/i
}
p
```

```
## [1] 1.00000 10.00000 16.66667 15.00000 20.00000
```

Note que primeiro definimos o objeto p, recebendo NULL. O NULL representa a ausência de um objeto e serve para já declarar algum objeto que receberá valor na sequência. No caso, ao rodar o for, o p é um vetor de tamanho 5 (tamanho do vetor m).

No exemplo, temos 5 iterações e para cada valor de i, correndo de 1 até 5 (tamanho de m), pegamos o valor de m na posição i e dividimos por sua posição. Assim formamos o vetor p.

2.4.2.13 Funções

Funções no R são nomes que guardam um código de R. A ideia é que sempre que rodar a função com os seus argumentos, o código que ela guarda será executado e o resultado será retornado.

Já usamos anteriormente algumas funções que estão na base do R. Por exemplo, quando usamos `class()` para entender a classe do objeto que o R está entendendo. Colocamos um argumento dentro do parênteses e o R retornou qual a classe do objeto em questão. Relembre o que falamos ao perguntar ao R qual a classe do vetor oi criado:

```
oi <- c(10,20,2,1,0.5)
class(oi)
```

```
## [1] "numeric"
```

Agora vamos conversar sobre outra função já criada e disponibilizada na base do R: `mean`. Essa função retorna a média do vetor que está em seu argumento. Vamos calcular a média dos valores do vetor oi:

```
mean(oi)
```

```
## [1] 6.7
```

Considere que, por algum motivo, tenha no vetor oi uma observação faltante. No R, dado faltante é caracterizado por NA.


```
oi <- c(10,20,2,1,0.5,NA)
```

Perceba que, apesar de NA ser um texto, não coloquei entre aspas porque quero falar para o R que naquela posição não tem valor e o R entende isso ao ler NA (sem aspas). Se colocar entre aspas, ele entenderá como sendo um texto e não mais como valor faltante.

```
mean(oi)
```

```
## [1] NA
```

Como não sabemos o valor do elemento na posição 6 do vetor oi, o R não teria como calcular a média de todos os 6 valores e por isso devolve NA. No entanto, queremos calcular a média dos elementos de oi ao retirar os valores faltantes, ou seja, queremos fazer: $(10+20+2+1+0.5)/5$. Então devemos falar para o R o que queremos e, para isso, podemos utilizar o argumento `na.rm = TRUE`:

```
mean(oi,na.rm = TRUE)
```

```
## [1] 6.7
```

Importantes:

- 1) se a função tiver mais de um argumento, eles são sempre separados por vírgulas;
- 2) cada função tem os seus próprios argumentos. Para saber quais são e como usar os argumentos de uma função, basta acessar a sua documentação. Uma forma de fazer isso é pela função `help`, cujo argumento é o nome da função que precisa de ajuda:

```
help(mean)
```

Veja que abrirá a documentação sobre a função `mean` no menu “Help” do RStudio e lá é possível ver os argumentos e exemplos de uso da função em questão.

Ainda sobre funções já presentes no R, vamos considerar agora a função `sample`. Veja a documentação dessa função para ver o que ela faz:

```
help(sample)
```

A função `sample` retorna uma amostra de um vetor com tamanho especificado em um de seus argumentos com ou sem reposição. Ela apresenta quatro argumentos: `sample(x, size, replace = FALSE, prob = NULL)`, em que: `x` é o vetor do qual será amostrado o número de elementos especificado no argumento `size`, seja com ou sem reposição (argumento `replace`) e com dadas probabilidades de seleção, especificadas em `prob`.

Quero usar essa função para amostrar do objeto oi (`x=oi`) dois elementos (`size=2`) em uma seleção com reposição (`replace = TRUE`) e que a probabilidade de seleção seja a mesma para todos os elementos do vetor oi. No caso da

probabilidade, como podemos ver na documentação da função `sample`, o *default* (padrão se o usuário não mudar o argumento) é ser a mesma probabilidade de seleção para todos os elementos. Assim, se o usuário nada especificar para esse argumento, o R entenderá o seu *default*. O mesmo vale para o argumento `replace`: caso fosse o interesse fazer a seleção sem reposição, não precisaríamos colocar esse argumento porque seu *default* é `FALSE`.

```
sample(x=oi,size=2,replace=TRUE) #não colocamos argumento prob porque vamos usar o seu
```

```
## [1] 10 1
```

Também poderíamos usar a mesma função sem colocar o nome dos argumentos:

```
sample(oi,2,TRUE)
```

```
## [1] 10.0 0.5
```

Mas, nesse caso, é importante que se respeite a ordem dos argumentos: o vetor tem que ser o primeiro, o segundo argumento é `size` e assim por diante.

Vale ressaltar que as duas últimas saídas não necessariamente serão as mesmas, porque é feito um sorteio aleatório de dois elementos de `oi` em cada uma delas.

Além de usar funções já prontas, podemos criar novas funções. Suponha que queremos criar uma função de dois argumentos que retorna o primeiro mais três vezes o segundo argumento. Criamos a função no que segue:

```
f_conta <- function(x,y) {
  out <- x+3*y
  return(out)
}
```

A função acima tem:

- o nome: `f_conta`;
- os argumentos: `x` e `y`;
- o corpo `out <- x+3*y` e
- o que retorna `return(out)`.

Suponha que eu queira fazer a conta: $10+3*20$. Podemos fazer isso ao chamar a função criada `f_conta`.

```
f_conta(x=10,y=20)
```

```
## [1] 70
```

Veja que o cálculo acima retorna exatamente o mesmo que o seguinte:

```
f_conta(y=20,x=10)
```

```
## [1] 70
```

Pois mudei a ordem dos argumentos, mas acompanhado com os nomes dos argumentos. Se eu não quiser colocar os nomes dos argumentos, precisa tomar cuidado para não errar a ordem deles. Pois:

```
f_conta(10,20)
```

```
## [1] 70
```

é diferente de

```
f_conta(20,10)
```

```
## [1] 50
```

2.5 Como obter ajuda no R

Listamos aqui 3 maneiras para buscar ajuda no R:

- Help/documentação do R (comandos `help(nome_da_funcao)` ou `?nome_da_funcao`). Como exemplo,

```
help(mean) #ou
?mean
```

- Google. Na Figura 2.47 está o exemplo de busca de ajuda no Google. Repare no ‘r’ no início da busca, isso pode ajudar.

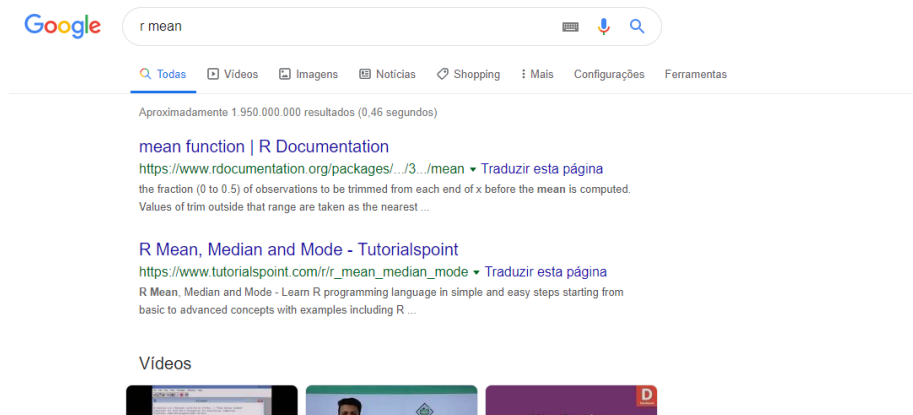


Figure 2.47: Pesquisa no Google

- Comunidade.

O Stack Overflow e o Stack Overflow em Português são sites de Pergunta e Resposta amplamente utilizados por todas as linguagens de programação, e o R é uma delas.

2.6 Pacotes

Como dito quando falamos “Sobre o R”, o R apresenta funções na sua base e também em forma de pacotes (conjunto de funções bem documentado), que precisam ser instalados (uma vez no seu computador) e carregados na sessão de utilização do R (carregado em toda sessão aberta).

Difícilmente você vai fazer uma análise apenas com as funções básicas do R e dificilmente não vai existir um pacote com as funções que você precisa. Por esse motivo, falamos a seguir em como instalar e carregar pacotes.

2.6.1 Instalação

- **Via CRAN:**

```
install.packages("nome-do-pacote")
```

Exemplo: Instalação do pacote `dplyr`.

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

Note que o nome do pacote está entre aspas.

- **Via Github:** Para instalar via Github, precisa primeiramente instalar o pacote `devtools`.

```
devtools::install_github("nome-do-repo/nome-do-pacote")
```

Exemplo:

```
devtools::install_github("tidyverse/dplyr")
```

2.6.2 Carregar pacotes:

Uma vez um pacote de interesse instalado em sua máquina, para carregar um pacote na sessão atual do R é só rodar a seguinte linha de comando:

```
library(nome-do-pacote)
```

Veja que para carregar o pacote, não usa aspas.

Como exemplo, o carregamento do pacote `dplyr`:

```
library(dplyr)
```

Só é necessário instalar o pacote uma vez, mas precisa carregá-lo toda vez que começar uma nova sessão.

Uma vez que o pacote carregado ao rodar a função `library()`, todas as funções desse pacote podem ser usadas sem problemas.

Caso você não queira carregar o pacote e apenas usar uma função específica do pacote, você pode usar `nome-do-pacote::nome-da-funcao`. Por exemplo:

```
dplyr::distinct(...)
```

Se você tivesse carregado o pacote `dplyr` anteriormente (pela função `library()`), não seria necessário colocar `dplyr::` antes da função `distinct` do pacote.

2.7 Materiais complementares

- Critical Thinking in Clinical Research. Felipe Fregni & Ben M. W. Illigens. 2018.
- Sites:
 - <https://www.bmj.com/about-bmj/resources-readers/publications/statistics-square-one/1-data-display-and-summary>
 - <http://www.sthda.com/english/wiki/statistical-tests-and-assumptions>
- CHAPTER 3: Selecting the Study Population. In: Critical Thinking in Clinical Research by Felipe Fregni and Ben Illigens. Oxford University Press 2018.
- Fandino W. Formulating a good research question: Pearls and pitfalls. Indian J Anaesth. 2019;63(8):611–616. doi:10.4103/ija.IJA_198_19
- Riva JJ, Malik KM, Burnie SJ, Endicott AR, Busse JW. What is your research question? An introduction to the PICOT format for clinicians. J Can Chiropr Assoc. 2012;56(3):167–171.
- External validity, generalizability, and knowledge utilization. Ferguson L1. J Nurs Scholarsh. 2004;36(1):16-22.
- Peter M Rothwell; Commentary: External validity of results of randomized trials: disentangling a complex concept, International Journal of Epidemiology, Volume 39, Issue 1, 1 February 2010, Pages 94–96, <https://doi.org/10.1093/ije/dyp305>

Chapter 3

Conceitos iniciais

3.1 População

A população é o conjunto de todos os elementos sob investigação, é sobre quem é a pergunta. São alguns exemplos:

1. Deseja-se avaliar se a discordância de peso de fetos gemelares aumenta o risco de óbito fetal.

População: Fetos gemelares.

2. Deseja-se saber se um novo tratamento para hipertensão grave é ou não efetivo.

População: Todas as pessoas que sofrem de hipertensão grave.

No estudo sobre o uso de progesterona em gestações gemelares, uma pessoa bem intencionada, mas sem conhecimento de Estatística, poderia pensar: “como gestação gemelar é de alto risco para prematuridade, assim como gestação com colo curto, em que neste último já é protocolo o uso de progesterona, podemos também usar progesterona em gestações gemelares sem a necessidade de estudo para essa população”. No entanto, esse raciocínio não está correto, pois nos estudos de colo curto só consideram gestações únicas, ou seja, não é estudada a população de gestações gemelares.

Essa discussão é importante, pois a extrapolação de resultado de uma população para outra não tem embasamento Estatístico e qualquer tipo de extrapolação dessa natureza deveria ser evitada.

3.2 Amostra

Muitas vezes não é possível acessar toda a população para estudarmos as características de interesse. Essa impossibilidade pode ser devido à razões econômicas, geográficas e/ou éticas.

Devido às dificuldades para se observar todos os elementos da população, alguns deles são considerados para o estudo. Este subconjunto representativo da população é denominado de **amostra**.

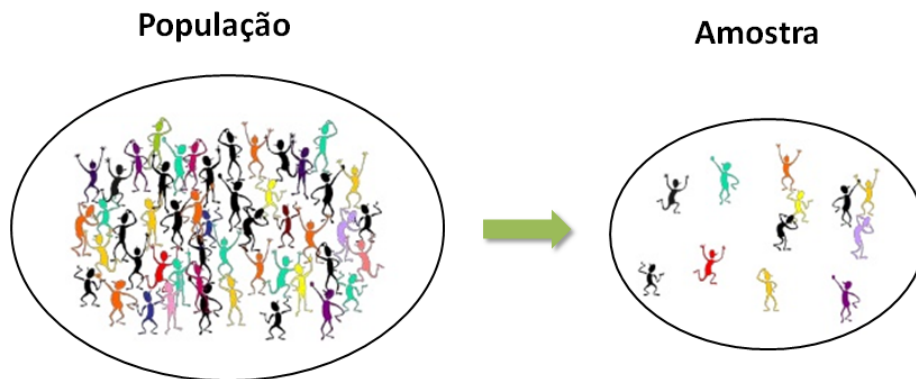


Figure 3.1: Esquema de população e amostra

Assim, a motivação de obter uma amostra se deve pela inviabilidade ou até mesmo impossibilidade de se observar toda a população. Como vantagens da amostra, podemos citar: redução de custos, redução de tempo de coleta e reproduz satisfatoriamente a realidade, se a amostra for representativa. A desvantagem se deve pelo motivo desta poder carregar vícios quando feita sem os devidos cuidados.

A técnica considerada para selecionar amostra representativa da população de interesse é a **amostragem**. A escolha da técnica de amostragem depende de muitas questões e decisões específicas de cada projeto.

Pela experiência em pesquisas na área médica, a definição da técnica de amostragem não é suficiente quando é planejada a coleta da amostra. O que precisa ser definido é o plano amostral.

Plano amostral: Estratégias e especificações metodológicas e estatísticas com o intuito de avaliar a questão científica de maneira adequada.

No plano amostral, algumas questões precisam ser pensadas:

- Está claramente definida a população de interesse?
- O desenho do estudo é adequado?

- Como será a amostragem?
- O tamanho da amostra é suficiente?
- Estão bem definidas as limitações do estudo?
- O estudo é viável frente a custo, tempo e disponibilidade das unidades?
- Existem variáveis confundidoras?
- Devo pensar em uma análise entre observadores?

3.3 Variáveis

Definimos aqui uma variável como sendo uma característica de interesse associada a uma população. Uma variável pode ser classificada em dois tipos:

- **Qualitativas:** apresentam como possíveis realizações uma qualidade (ou atributo) do indivíduo pesquisado. Exemplos: sexo, grau de instrução, estado civil, classe social, presença de diabetes (sim ou não), etc.
- **Quantitativas:** apresentam como possíveis realizações números resultantes de uma contagem ou de uma mensuração. Exemplos: Número de filhos, salário, temperatura, pressão arterial, concentração de alguma substância, etc.

Ainda, uma **variável qualitativa** pode ser dividida em **ordinal** ou **nominal**, em que:

- **ordinal:** Existe uma ordem nos seus resultados. Exemplo: grau de instrução, estadiamento de uma doença, etc.
- **nominal:** Não existe nenhuma ordenação nas possíveis realizações. Exemplo: Sexo, estado civil, presença de diabetes, etc.

Já uma **variável quantitativa** pode ser dividida em **discreta** ou **contínua**, em que:

- **discreta:** Possíveis valores formam um conjunto finito ou enumerável de pontos e que resultam, frequentemente, de uma contagem. Exemplo: número de filhos (0,1,2,...), quantidade de acidentes em um mês (0, 1, 2, ...), número de carros que passam no pedágio de Vitória para Vila Velha em 1 hora, etc.
- **contínua:** Características mensuráveis que assumem valores em um intervalo de números reais. Exemplo: salário, temperatura, pressão arterial, etc.

Podemos resumir os tipos de variáveis como o esquema na Figura 3.2.

A identificação do tipo de variável é importante para a tabulação de dados de maneira correta e também para realizar a análise dos dados de maneira adequada. A seguir falamos sobre a tabulação de dados.

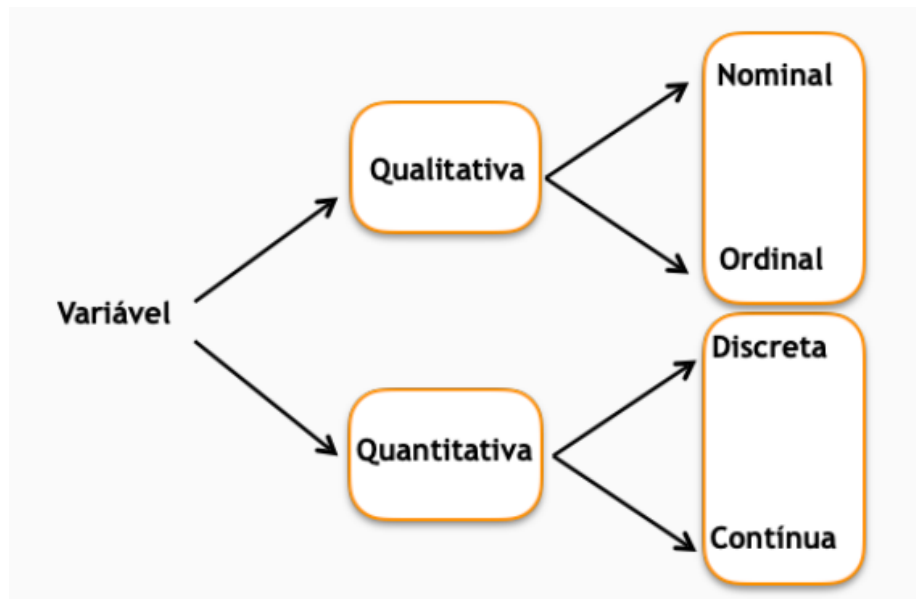


Figure 3.2: Esquema dos tipos de variáveis

3.4 Tabulação de dados

Bases de dados são estruturas organizadas com o objetivo de permitir sua análise estatística. Em geral, cada linha da planilha corresponde a uma unidade de investigação e cada coluna corresponde a uma variável.

O início dos trabalhos da tabulação dos dados deve ser feito antes da coleta de dados. O planejamento da planilha contribui tanto para o entendimento do processo de coleta de dados quanto para a especificação das variáveis a serem avaliadas. Uma das primeiras medidas é a elaboração de um dicionário com a especificação das variáveis.

Para mais detalhes sobre tabulação de dados, acesse <https://daslab-ufes.github.io/materiais/> e clique em “Tabulacao de Dados”.

3.5 Importação de dados no R

3.5.1 Extensão .txt ou .csv

Para importar dados para o R, com extensão .txt ou .csv, uma sugestão é utilizar o pacote `readr`. Como exemplo, consideramos um arquivo chamado `dados1` que queremos importar para o R.

```
library(readr)
dados_csv <- read_csv(file = "caminho-para-o-arquivo/dados1.csv")
dados_txt <- read_delim(file = "caminho-para-o-arquivo/dados1.txt", delim = " ")
```

O argumento `file=` representa o caminho onde o arquivo está alocado. Se o arquivo estiver no diretório de trabalho (quando criamos o projeto e colocamos os arquivos de dados na pasta criada pelo projeto), não precisa especificar o caminho até o arquivo, como segue:

```
dados_csv <- read_csv(file = "dados1.csv")
dados_txt <- read_delim(file = "dados1.txt", delim = " ")
```

O argumento `delim=` indica qual o separador das colunas no arquivo de texto.

Outra opção para leitura de arquivo `.txt` é usar a função `read.table` que já está salva na base, ou seja, não é necessário instalar pacotes.

```
dados_txt2 <- read.table(file="dados1.txt",header=T)
```

O argumento `header` indica se a primeira linha do arquivo consta o nome das variáveis. Se for `T` (TRUE), a primeira linha é indicada como nome das variáveis. O *default* (padrão se o usuário não mudar o argumento) é `header=F`.

Vale ressaltar que para cada função `read_`, existe uma respectiva função `write_` para salvar o arquivo no formato de interesse. Como exemplo, queremos salvar a base de dados `mtcars` na pasta do meu computador com o nome `cars`:

```
write_csv(x = mtcars, path = "cars.csv")
write_delim(x = mtcars, delim = " ", path = "cars.txt"))
```

3.5.2 Arquivos em Excel

O pacote `readxl` pode ser utilizado para leitura de arquivos do Excel, como `.xls` e `.xlsx`.

```
library(readxl)
dados_excel <- read_xls(path = "dados1.xls") #Leitura do arquivo .xls
dados_excelx <- read_xlsx(path = "dados1.xlsx") #Leitura do arquivo .xlsx
```

Dentre os argumentos dessas funções (veja o `help()` de cada uma delas), o argumento `na` indica quais strings (textos) devem ser interpretados como `NA`. O *default* é considerar espaço em branco no excel.

Uma maneira mais simples é a utilização da função `read_excel()`, pois ela auto detecta a extensão do arquivo.

```
library(readxl)
dados_excel1 <- read_excel(path = "dados1.xls")
dados_excelx1 <- read_excel(path = "dados1.xlsx")
```

Podemos também exportar um arquivo em excel (.xls e .xlsx) ao considerar a função `write_xlsx` do pacote `writexl`. Suponha que temos o interesse em salvar a base de dados `dados` em excel na pasta do computador (exportar) com o nome de `dados_correto`:

```
library(writexl)
write_xlsx(dados, "dados_correto.xlsx")
```

3.5.3 Arquivos de outros softwares

Para ler dados salvos em extensão de outros softwares: SPSS, STATA e SAS, podem ser utilizadas as funções do pacote `haven`.

```
library(haven)
dados_stata <- read_stata("dados1.dta")
dados_spss <- read_spss("dados1.sav")
dados_sas <- read_sas("dados1.sas7bdat")
```

Outra opção de pacote para importação de dados de outros softwares é o `foreign`. Além do SAS, STAT e SPSS, ele também lê dados do Octave, Minitab e Epi Info.

3.6 Análise de consistência e tratamento dos dados

O tratamento dos dados toma muitas vezes a maior parte do tempo de uma análise estatística.

A análise de consistência consiste em realizar uma primeira análise dos dados com o intuito de encontrar inconsistências. São exemplos de inconsistências:

- boas práticas para nome das variáveis.
- como erros de digitação;
- indivíduos imputados mais de uma vez na planilha de dados de maneira errada;
- identificar casos missings e avaliar se a observação está ausente de maneira correta ou não;
- identificar as categorias de variáveis qualitativas.

Para exemplificar o que foi discutido, consideramos os dados fictícios de $n = 104$ gestações gemelares, apresentado no Capítulo 2. O dicionário contém a explicação das variáveis, a identificação dos valores ausentes de cada variável e o rótulo das categorias de variáveis qualitativas. Uma boa prática na tabulação dos dados é manter o dicionário junto com a base de dados. Se a tabulação for realizada no Excel, por exemplo, manter o dicionário em uma outra aba.

3.6.1 Importação dos dados

```
library(readxl)
dados <- read_excel(path = "dataset/dados_gemelares.xlsx")
dados
```

```
## # A tibble: 108 x 21
##       ID Grupo CORION `Data aval`      `Data nascimento`  `COR BRANCO`
##   <dbl> <dbl> <chr>   <dtm>                <dtm>                <dbl>
## 1     1     2 Di    2017-04-23 00:00:00  1988-04-30 00:00:00         1
## 2     2     1 Mono  2016-03-21 00:00:00  1982-03-30 00:00:00         1
## 3     2     1 Mono  2016-03-21 00:00:00  1982-03-30 00:00:00         1
## 4     3     1 Di    2016-02-17 00:00:00  1991-02-23 00:00:00         2
## 5     5     2 Di    2017-04-23 00:00:00  1988-04-30 00:00:00         2
## 6     6     1 Di    2016-03-21 00:00:00  1989-03-28 00:00:00         2
## 7     7     2 Di    2016-02-17 00:00:00  1985-02-24 00:00:00         2
## 8     8     1 Di    2017-12-14 00:00:00  1988-12-21 00:00:00         1
## 9     9     1 Di    2017-04-23 00:00:00  1980-05-02 00:00:00         3
## 10    10     2 Di    2016-03-21 00:00:00  1984-03-29 00:00:00         1
## # ... with 98 more rows, and 15 more variables: `Peso Pré` <dbl>, ALT <dbl>,
## #   Gesta <dbl>, Para <dbl>, Aborto <dbl>, IND_AP <dbl>, Tabagismo <dbl>,
## #   Alcool <dbl>, Drogas <dbl>, IG_Aval <dbl>, MedidaColo <dbl>,
## #   Num_contra_CTG <dbl>, `IGP semana` <dbl>, `IGP dia` <dbl>, oi <lgl>
```

Exercício: Na base em excel, substitua o espaço em branco (dados faltantes) por NA e rode o seguinte comando:

```
dados <- read_excel(path = "dataset/dados_gemelares.xlsx", na="NA")
```

O default do missing é o espaço em branco. Acesse o help em `?read_excel` e veja `na = ""`.

3.6.2 Arrumação da base de dados

Inicialmente, vamos verificar os nomes das variáveis na base de dados por meio da função `names`. Note que os nomes tem letras maiúsculas, espaços e acento. Utilizar os dados com essas características não impossibilita as futuras análises, mas pode atrapalhar quando precisamos selecionar algumas dessas variáveis.

```
names(dados)
```

```
## [1] "ID"          "Grupo"       "CORION"      "Data aval"
## [5] "Data nascimento" "COR BRANCO"  "Peso Pré"    "ALT"
## [9] "Gesta"       "Para"       "Aborto"      "IND_AP"
## [13] "Tabagismo"   "Alcool"     "Drogas"      "IG_Aval"
## [17] "MedidaColo"  "Num_contra_CTG" "IGP semana"  "IGP dia"
## [21] "oi"
```

uma boa prática consiste em padronizar os nomes das variáveis, até para facilitar

a lembrança deles. Para isso, utilizaremos o pacote `janitor` para a arrumação da base de dados. Usamos a função `clean_names()` para primeiro ajuste dos nomes das variáveis.

```
library(janitor)
```

```
dados <- janitor::clean_names(dados)
```

Agora vamos ver como ficou o nome das variáveis:

```
names(dados)
```

```
## [1] "id"          "grupo"       "corion"      "data_aval"
## [5] "data_nascimento" "cor_branco"  "peso_pre"    "alt"
## [9] "gesta"       "para"       "aborto"     "ind_ap"
## [13] "tabagismo"   "alcool"     "drogas"     "ig_aval"
## [17] "medida_colo" "num_contra_ctg" "igp_semana" "igp_dia"
## [21] "oi"
```

Veja que ele deixou todos os nomes minúsculos, substituindo o espaço por “_” e tirando acentos. Isso ajuda a evitar problemas futuros em algumas análises que não lidam muito bem com acentos e espaços nos nomes das variáveis.

Outro problema comum é a presença de linhas e colunas vazias. Na base de dados em questão, não há linhas em branco, como pode ser visto na saída abaixo.

```
janitor::remove_empty(dados, "rows")
```

```
## # A tibble: 108 x 21
##       id grupo corion data_aval      data_nascimento cor_branco
##   <dbl> <dbl> <chr>   <dtm>          <dtm>              <dbl>
## 1     1     2 Di    2017-04-23 00:00:00 1988-04-30 00:00:00     1
## 2     2     1 Mono  2016-03-21 00:00:00 1982-03-30 00:00:00     1
## 3     2     1 Mono  2016-03-21 00:00:00 1982-03-30 00:00:00     1
## 4     3     1 Di    2016-02-17 00:00:00 1991-02-23 00:00:00     2
## 5     5     2 Di    2017-04-23 00:00:00 1988-04-30 00:00:00     2
## 6     6     1 Di    2016-03-21 00:00:00 1989-03-28 00:00:00     2
## 7     7     2 Di    2016-02-17 00:00:00 1985-02-24 00:00:00     2
## 8     8     1 Di    2017-12-14 00:00:00 1988-12-21 00:00:00     1
## 9     9     1 Di    2017-04-23 00:00:00 1980-05-02 00:00:00     3
## 10    10     2 Di    2016-03-21 00:00:00 1984-03-29 00:00:00     1
## # ... with 98 more rows, and 15 more variables: peso_pre <dbl>, alt <dbl>,
## #   gesta <dbl>, para <dbl>, aborto <dbl>, ind_ap <dbl>, tabagismo <dbl>,
## #   alcool <dbl>, drogas <dbl>, ig_aval <dbl>, medida_colo <dbl>,
## #   num_contra_ctg <dbl>, igp_semana <dbl>, igp_dia <dbl>, oi <lgl>
```

Propositalmente, inclui a coluna “oi” vazia para podermos eliminá-la com o comando abaixo:

```
dados <- janitor::remove_empty(dados,"cols") #limpando colunas vazias
dados
```

```
## # A tibble: 108 x 20
##       id grupo corion data_aval      data_nascimento cor_branco
##   <dbl> <dbl> <chr>   <dtm>          <dtm>              <dbl>
## 1     1     2 Di      2017-04-23 00:00:00 1988-04-30 00:00:00     1
## 2     2     1 Mono    2016-03-21 00:00:00 1982-03-30 00:00:00     1
## 3     3     1 Mono    2016-03-21 00:00:00 1982-03-30 00:00:00     1
## 4     4     3 Di      2016-02-17 00:00:00 1991-02-23 00:00:00     2
## 5     5     2 Di      2017-04-23 00:00:00 1988-04-30 00:00:00     2
## 6     6     1 Di      2016-03-21 00:00:00 1989-03-28 00:00:00     2
## 7     7     2 Di      2016-02-17 00:00:00 1985-02-24 00:00:00     2
## 8     8     1 Di      2017-12-14 00:00:00 1988-12-21 00:00:00     1
## 9     9     1 Di      2017-04-23 00:00:00 1980-05-02 00:00:00     3
## 10    10     2 Di      2016-03-21 00:00:00 1984-03-29 00:00:00     1
## # ... with 98 more rows, and 14 more variables: peso_pre <dbl>, alt <dbl>,
## #   gesta <dbl>, para <dbl>, aborto <dbl>, ind_ap <dbl>, tabagismo <dbl>,
## #   alcool <dbl>, drogas <dbl>, ig_aval <dbl>, medida_colo <dbl>,
## #   num_contra_ctg <dbl>, igp_semana <dbl>, igp_dia <dbl>
```

3.6.3 Identificação de casos duplicados

Uma boa prática consiste em identificar casos duplicados, isto é, identificar se há casos erroneamente repetidos. No exemplo, a variável chave é “id”, em que cada indivíduo distinto apresenta um id distinto. Para identificar casos duplicados pela variável chave “id”, usamos a função `get_dupes` do pacote `janitor`.

```
janitor::get_dupes(dados, id)
```

```
## # A tibble: 8 x 21
##       id dupe_count grupo corion data_aval      data_nascimento
##   <dbl>      <int> <dbl> <chr>   <dtm>          <dtm>
## 1     2          2     1 Mono    2016-03-21 00:00:00 1982-03-30 00:00:00
## 2     2          2     1 Mono    2016-03-21 00:00:00 1982-03-30 00:00:00
## 3    11          2     2 Di      2016-02-17 00:00:00 1981-02-25 00:00:00
## 4    11          2     2 Di      2016-02-17 00:00:00 1981-02-25 00:00:00
## 5    17          2     1 Di      2017-04-23 00:00:00 1993-04-29 00:00:00
## 6    17          2     1 Di      2017-04-23 00:00:00 1993-04-29 00:00:00
## 7    23          2     2 Di      2016-02-17 00:00:00 1997-02-21 00:00:00
## 8    23          2     2 Di      2016-02-17 00:00:00 1997-02-21 00:00:00
## # ... with 15 more variables: cor_branco <dbl>, peso_pre <dbl>, alt <dbl>,
## #   gesta <dbl>, para <dbl>, aborto <dbl>, ind_ap <dbl>, tabagismo <dbl>,
## #   alcool <dbl>, drogas <dbl>, ig_aval <dbl>, medida_colo <dbl>,
## #   num_contra_ctg <dbl>, igp_semana <dbl>, igp_dia <dbl>
```

No exemplo, note que os ID's = 2, 11, 17 e 23 aparecem duas vezes cada,

o que não está correto para essa aplicação. Para eliminar linhas duplicadas identificadas, usamos a função `distinct` do pacote `dplyr`.

```
library(dplyr)
```

```
dados <- dplyr::distinct(dados,id, .keep_all = TRUE)
dados
```

```
## # A tibble: 104 x 20
##       id grupo corion data_aval      data_nascimento cor_branco
##   <dbl> <dbl> <chr>   <dtm>          <dtm>          <dbl>
## 1     1     2 Di    2017-04-23 00:00:00 1988-04-30 00:00:00     1
## 2     2     1 Mono  2016-03-21 00:00:00 1982-03-30 00:00:00     1
## 3     3     1 Di    2016-02-17 00:00:00 1991-02-23 00:00:00     2
## 4     5     2 Di    2017-04-23 00:00:00 1988-04-30 00:00:00     2
## 5     6     1 Di    2016-03-21 00:00:00 1989-03-28 00:00:00     2
## 6     7     2 Di    2016-02-17 00:00:00 1985-02-24 00:00:00     2
## 7     8     1 Di    2017-12-14 00:00:00 1988-12-21 00:00:00     1
## 8     9     1 Di    2017-04-23 00:00:00 1980-05-02 00:00:00     3
## 9    10     2 Di    2016-03-21 00:00:00 1984-03-29 00:00:00     1
## 10   11     2 Di    2016-02-17 00:00:00 1981-02-25 00:00:00     1
## # ... with 94 more rows, and 14 more variables: peso_pre <dbl>, alt <dbl>,
## #   gesta <dbl>, para <dbl>, aborto <dbl>, ind_ap <dbl>, tabagismo <dbl>,
## #   alcool <dbl>, drogas <dbl>, ig_aval <dbl>, medida_colo <dbl>,
## #   num_contra_ctg <dbl>, igp_semana <dbl>, igp_dia <dbl>
```

Ao chamar os dados, apenas as dez primeiras linhas são impressas na tela e as colunas que não couberam na largura do console serão omitidas. Vale ressaltar que, também são apresentadas a dimensão da tabela (no exemplo, 104 X 20) e as classes de cada coluna.

3.6.4 Identificar problemas nas variáveis da base de dados

Outra etapa importante na análise de consistência é identificar o tipo de variável e ver se o R está interpretando corretamente o tipo de cada variável.

Temos na nossa base de dados colunas de data, além de variáveis qualitativas e quantitativas (veja o dicionário das variáveis na Figura 2.1). São as variáveis de data: “data_aval” e “data_nascimento”. As variáveis qualitativas são: “grupo” (progesterona ou placebo), “corion” (mono ou dicorônica), “cor_branco” (branca ou não branca), “ind_ap” (sim ou não), “tabagismo” (sim ou não), “alcool” (sim ou não) e “drogas” (sim ou não). As demais variáveis são quantitativa.

Mas precisamos entender se o R realmente entendeu todas as variáveis da maneira correta. Uma maneira de identificar isso e também de ver algumas descritivas das variáveis que nos auxiliam a ver possíveis inconsistências na base de dados é a a função `skim` do pacote `skimr`.


```
skimr::skim(dados)
```

```
## Error in kable_latex(x = structure(c("Name", "Number of rows", "Number of columns", : unused a
```

No R, as variáveis qualitativas são nomeadas “factor”, as variáveis quantitativas são nomeadas “numeric” e as variáveis de data são “date”. Note que na importação dos dados o R não entendeu corretamente os tipos de variáveis. Mas vamos corrigir isso no que segue.

Começando pela data, vamos rodar o seguinte código:

```
dados$data_aval <- as.Date(dados$data_aval)
dados$data_nascimento <- as.Date(dados$data_nascimento)
```

A função `as.Date` informa para o R que a variável do seu argumento é de data. Vamos ver como ficou:

```
skimr::skim(dados)
```

```
## Error in kable_latex(x = structure(c("Name", "Number of rows", "Number of columns", : unused a
```

Agora vamos lidar com as variáveis qualitativas. Veja que “para”corion” foi identificada como *character*. Isso acontece porque ela foi tabulada como texto. Já as demais variáveis qualitativas estão identificadas como *numeric*, pois na tabulação suas categorias estão codificadas com números. Para então dizer ao R o verdadeiro tipo dessas variáveis, vamos utilizar os seguintes comandos:

```
dados$grupo <- as.factor(dados$grupo)
dados$corion <- as.factor(dados$corion)
dados$cor_branco <- as.factor(dados$cor_branco)
dados$ind_ap <- as.factor(dados$ind_ap)
dados$tabagismo <- as.factor(dados$tabagismo)
dados$alcool <- as.factor(dados$alcool)
dados$drogas <- as.factor(dados$drogas)
```

A função `as.factor` é análoga à função `as.Date`, mas agora informando que as variáveis no argumento da função são qualitativas (factor). Vamos ver como ficou:

```
skimr::skim(dados)
```

```
## Error in kable_latex(x = structure(c("Name", "Number of rows", "Number of columns", : unused a
```

Veja que agora as variáveis qualitativas da base estão identificadas no bloco *factor*.

Mas ainda sobre as variáveis qualitativas, podemos identificar algumas inconsistências. Começando pela variável “cor_branco”, tem uma categoria 3, o que está errado (1 - branca e 2- não branca). Para poder corrigir essa observação, vamos primeiro identificar quem é esse caso. Para isso, usamos o seguinte comando:

```
dados[dados$cor_branco==3,]
```

```
## # A tibble: 1 x 20
##       id grupo corion data_aval data_nascimento cor_branco peso_pre alt gesta
##   <dbl> <fct> <fct>   <date>      <date>          <fct>      <dbl> <dbl> <dbl>
## 1     9 1     Di     2017-04-23 1980-05-02      3         72  1.65    8
## # ... with 11 more variables: para <dbl>, aborto <dbl>, ind_ap <fct>,
## #   tabagismo <fct>, alcool <fct>, drogas <fct>, ig_aval <dbl>,
## #   medida_colo <dbl>, num_contra_ctg <dbl>, igp_semana <dbl>, igp_dia <dbl>
```

Identificamos que o caso é id=9. O pesquisador viu em seus registros que foi um erro de digitação é o correto é 1 (cor branca). Vamos então corrigir esse ponto na base de dados:

```
dados$cor_branco <- ifelse(dados$id==9,1,dados$cor_branco)
```

```
#precisamos informar novamente ao R que cor_branco é fator:
dados$cor_branco <- as.factor(dados$cor_branco)
```

A função `ifelse` tem três argumentos. No primeiro, a condição é colocada (no exemplo, se a variável `id` for igual a 9), o segundo argumento é o que retorna se a condição for verdadeira (no exemplo, se `id` for realmente igual a 9, a variável “`cor_branco`” recebe 1) e, por fim, o último argumento é o que retorna se a condição for falsa (se `id` não for igual a 9 - todas as outras observações da base de dados, a variável “`cor_branco`” continua com o valor que está).

Outra variável qualitativa com algum problema aparente é “`corion`”. Essa variável tem duas categorias: `mono` e `dicorionica`, mas perceba que o R está entendendo que ela tem 4 categorias. Isso acontece porque ela é uma variável de texto (*character*) e o R é caso-sensível a letras maiúsculas e minúsculas (o R entende as 4 categorias: `Mono`, `MONO`, `Di` e `DI`). Por isso, devemos corrigir e deixar todas as respostas “`mono`” escritas da mesma maneira e fazer o mesmo com a categoria “`di`”.

Para lidar com variáveis de texto (como é o caso de “`corion`”), um pacote no R bastante útil é o `stringr`. No nosso caso, conseguimos resolver o problema ao colocar todas as palavras em letra minúscula. Para isso, usamos a função `str_to_lower` do pacote `stringr`:

```
dados$corion <- stringr::str_to_lower(dados$corion)
```

Como usei uma função para variáveis de texto, preciso avisar novamente ao R que “`corion`” é fator:

```
dados$corion <- as.factor(dados$corion)
```

Ótimo! Corrigimos as inconsistências das variáveis qualitativas. Mas outra questão surge: como faço para usar um rótulo nos números codificados nas categorias das variáveis qualitativas? Para o grupo, por exemplo, ao invés de apare-

cer 1 quero que apareça placebo. Para isso, vamos utilizar o pacote `forcats` que lida com variáveis qualitativas (categóricas). Para renomear as categorias das variáveis, vamos usar a função `fct_recode` desse pacote:

```
dados$grupo <- forcats::fct_recode(dados$grupo,
                                   placebo = "1",
                                   progest = "2")

# Agora vamos para a variável cor_branco (1- branca e 2- nbranca)
dados$cor_branco <- forcats::fct_recode(dados$cor_branco,
                                         branca = "1",
                                         nbranca = "2")

# Agora vamos para a variável ind_ap (1- sim e 0 -não)
dados$ind_ap <- forcats::fct_recode(dados$ind_ap,
                                     sim = "1",
                                     nao = "0")

# Agora vamos para a variável tabagismo (1- sim e 0 -não)
dados$tabagismo <- forcats::fct_recode(dados$tabagismo,
                                       sim = "1",
                                       nao = "0")

# Agora vamos para a variável alcool (1- sim e 0 -não)
dados$alcool <- forcats::fct_recode(dados$alcool,
                                    sim = "1",
                                    nao = "0")

# Agora vamos para a variável drogas (1- sim e 0 -não)
dados$drogas <- forcats::fct_recode(dados$drogas,
                                    sim = "1",
                                    nao = "0")
```

Aqui vale mais uma dica: se você rodar a função `View(dados)` vai abrir uma janela com a planilha dos dados para visualização. Aparecerá como segue:

Finalmente chegamos nas variáveis quantitativas. Uma forma de identificar problemas em variáveis quantitativas é avaliar os valores mínimo e máximo de cada variável e ver se tem algum valor impossível para a mesma.

Veja que tem uma altura de 163. A unidade de medida é em metros e a altura em questão foi um erro de digitação. O certo era 1.63. Então vamos primeiro identificar o caso e depois vamos corrigir.

```
dados[dados$alt==163,]
```

```
## # A tibble: 3 x 20
```

```
##       id grupo corion data_aval data_nascimento cor_branco peso_pre alt gesta
```

	id	grupo	corion	data_aval	data_nascimento	cor_branco	peso_pre	alt	gesta
1	1	progest	di	2017-04-23	1988-04-30	branca	93.0	1.63	
2	2	placebo	mono	2016-03-21	1982-03-30	branca	59.0	1.45	
3	3	placebo	di	2016-02-17	1991-02-23	nbranca	87.0	1.69	
4	5	progest	di	2017-04-23	1988-04-30	nbranca	78.0	1.59	
5	6	placebo	di	2016-03-21	1989-03-28	nbranca	62.0	1.64	
6	7	progest	di	2016-02-17	1985-02-24	nbranca	54.0	NA	
7	8	placebo	di	2017-12-14	1988-12-21	branca	72.0	1.64	
8	9	placebo	di	2017-04-23	1980-05-02	branca	72.0	1.65	
9	10	progest	di	2016-03-21	1984-03-29	branca	102.0	1.68	
10	11	progest	di	2016-02-17	1981-02-25	branca	55.0	1.45	
11	12	progest	di	2017-12-14	1986-12-22	branca	66.0	1.60	
12	13	progest	di	2017-04-23	1991-04-30	branca	65.0	1.73	
13	14	placebo	mono	2016-03-21	1998-03-26	branca	48.0	1.56	
14	15	progest	di	2016-02-17	1981-02-25	branca	75.0	1.61	

Figure 3.3: Tela dos dados após comando View(dados)

```
##      <dbl> <fct> <fct>   <date>      <date>          <fct>          <dbl> <dbl> <dbl>
## 1      NA <NA>  <NA>    NA          NA          <NA>          NA   NA   NA
## 2     27 prog~ di      2016-02-17 1997-02-21    nbranca          58  163   1
## 3      NA <NA>  <NA>    NA          NA          <NA>          NA   NA   NA
## # ... with 11 more variables: para <dbl>, aborto <dbl>, ind_ap <fct>,
## #   tabagismo <fct>, alcool <fct>, drogas <fct>, ig_aval <dbl>,
## #   medida_colo <dbl>, num_contra_ctg <dbl>, igp_semana <dbl>, igp_dia <dbl>
```

O caso em questão é o id=27 e vamos novamente usar a função ifelse para substituir o valor:

```
dados$alt <- ifelse(dados$id==27,1.63,dados$alt)
```

Há também um valor absurdo para “ig_aval” (idade gestacional da avaliação) de 83.86. Vamos identificar o id para depois corrigir:

```
dados[dados$ig_aval==83.86,]
```

```
## # A tibble: 1 x 20
##       id grupo corion data_aval  data_nascimento cor_branco peso_pre  alt gesta
##   <dbl> <fct> <fct>   <date>      <date>          <fct>          <dbl> <dbl> <dbl>
## 1    21 plac~ mono   2017-04-23 1988-04-30    branca          44  1.64   1
## # ... with 11 more variables: para <dbl>, aborto <dbl>, ind_ap <fct>,
## #   tabagismo <fct>, alcool <fct>, drogas <fct>, ig_aval <dbl>,
## #   medida_colo <dbl>, num_contra_ctg <dbl>, igp_semana <dbl>, igp_dia <dbl>
```

Como podemos observar, o `id=21` é o caso inconsistente para “`ig_aval`” e o pesquisador checkou que o correto é 33.86. Vamos então corrigir novamente com a função `ifelse`.

```
dados$ig_aval <- ifelse(dados$id==21,33.86,dados$ig_aval)
```

Agora está tudo certo?

```
skimr::skim(dados)
```

```
## Error in kable_latex(x = structure(c("Name", "Number of rows", "Number of columns", : unused a
```

3.7 Transformação dos dados

Nessa parte do material discutiremos como fazer algumas transformações úteis nas variáveis. Para isso, utilizaremos as funções do pacote `dplyr`, um pacote bastante útil para a manipulação de dados.

3.7.1 Transformação de variáveis quantitativas

Primeiramente, temos o interesse em criar a variável IMC, dada pelo peso (em kg) dividido pela altura (em metros) ao quadrado. Para isso, usamos a função `mutate` do `dplyr`:

```
dados <- dplyr::mutate(dados,imc = peso_pre/(alt^2))
```

Vamos ver então como ficou:

```
skimr::skim(dados$imc) #coloquei só imc para só fazer descritivas para imc.
```

```
## Error in kable_latex(x = structure(c("Name", "Number of rows", "Number of columns", : unused a
```

Note que há três valores ausentes para “`imc`”, uma vez que há dois valores ausentes para “`alt`” e outro para “`peso_pre`”.

Observe que na planilha tem duas colunas sobre idade gestacional: `igp_semanas` (idade gestacional em semanas fechadas) e `igp_dia` (dias ainda não de 1 semana completa). Vamos criar uma coluna que combine essas duas informações e que transforme os dias em semanas de maneira fracionada.

```
dados <- dplyr::mutate(dados,igp = igp_semana+igp_dia/7)
```

Vamos ver então como ficou:

```
skimr::skim(dados$igp) #coloquei só igp para só fazer descritivas para igp.
```

```
## Error in kable_latex(x = structure(c("Name", "Number of rows", "Number of columns", : unused a
```

3.7.2 Transformação de variáveis qualitativas

A variável “gesta” indica o número de gestações, contando com a atual da gestante em gestão. Logo, uma gestante com gesta=1 está em sua primeira gestação, ou seja, é primigesta. Queremos criar uma nova variável indicadora de gestação primigesta. Há diferentes forma de fazer isso. Vamos usar o comando `ifelse` já utilizado anteriormente:

```
dados$primigesta <- ifelse(dados$gesta==1,1,0)
```

Agora vamos recodificar `primigesta` com o nome de cada categoria:

```
dados$primigesta <- as.factor(dados$primigesta)
dados$primigesta <- forcats::fct_recode(dados$primigesta,
                                       nao = "0",
                                       sim = "1")
```

Outra variável qualitativa que é função de uma variável quantitativa da planilha é a prematuridade. Definimos aqui prematuridade se idade gestacional do parto for menor que 37 semanas.

```
dados$prematuridade <- ifelse(dados$igp<37,1,0)
```

Agora vamos recodificar “prematuridade” com o nome de cada categoria:

```
dados$prematuridade <- as.factor(dados$prematuridade)
dados$prematuridade <- forcats::fct_recode(dados$prematuridade,
                                       nao = "0",
                                       sim = "1")
```

Vamos ver então como ficou:

```
skimr::skim(dados,primigesta,prematuridade)
```

```
## Error in kable_latex(x = structure(c("Name", "Number of rows", "Number of columns",
```

Veja que coloquei “primigesta” e “prematuridade” no segundo e terceiro argumento do `skim` para apenas retornar as descritivas dessas duas variáveis.

Exercício:

- 1) Crie a variável `indicador_aborto` (sim e nao) - sim se `aborto >=1` e nao se `aborto=0`.
- 2) Crie a variável `primipara` (sim e nao) - sim se `para >=1` e nao se `para=0`.

3.7.3 Manipulação de datas

Na base de dados, há duas variáveis de data: data da avaliação e data do nascimento. A diferença entre as duas datas, em anos, é a idade da gestante no momento da avaliação. Para realizar operações com data, usaremos o pacote `lubridate`.

A data está salva no formato ano-mês-dia e por isso usamos a função `ymd(.)` para as variáveis de data. Para calcular a diferença entre as data, usamos a função `intervalo`, atribuindo ao objeto `intervalo`. Por fim, obtemos a idade ao dividir o `intervalo` por ano.

```
intervalo <- lubridate::ymd(dados$data_nascimento) %--% lubridate::ymd(dados$data_aval)
dados$idade <- intervalo / lubridate::dyears(1) #número de anos
dados$idade <- trunc(dados$idade) #usamos a idade completada
```

A última linha do código acima utiliza a função `trunc` para truncar a idade para não usar a fração dos meses da idade ainda não completa e sim usar a idade completada no último aniversário.

Vale ressaltar que há várias funções importantes para lidar com variáveis de data no pacote `lubridate`. Para mais detalhes, ver a documentação do `lubridate`.

3.7.4 Filtragem de observações

Uma função bastante importante quando estamos analisando dados é filtrar de acordo com uma condição de interesse. Suponha que temos interesse em realizar uma determinada análise apenas com gestantes com menos de 23 anos. Para selecionar apenas as gestantes mais novas, podemos utilizar a função `filter` do pacote `dplyr`.

```
dados_jovens <- filter(dados, idade < 23)
```

Veja que agora o objeto `dados_jovens` é a base de dados apenas com aquelas cuja idade é menor que 23 (use o comando `View(dados_jovens)` para verificar).

Utilizando os operadores lógicos “&” (e) e “|” (ou), conseguimos realizar algumas condições, como mostrado no exemplo abaixo.

Agora, queremos selecionar apenas as gestantes mais novas e do grupo `progest`. Para isso:

```
dados_jovens_progest <- filter(dados, idade < 23 & grupo == "progest")
```

Acho que aqui vale uma nota sobre o pacote `dplyr` e por isso dediquei uma subseção para ele, no que segue.

3.7.4.1 Pacote `dplyr`

O `{dplyr}` é um pacote muito útil para realizar transformação de dados.

Suas principais funções são:

- `filter()` - filtra linhas;
- `select()` - seleciona colunas;
- `arrange()` - ordena a base;

- `mutate()` - cria/modifica colunas.

Já falamos das funções `filter` e `mutate` anteriormente, então falamos brevemente das outras funções no que segue.

A função `select()` seleciona colunas (variáveis). Vamos supor que eu tenha interesse em criar um objeto com só as variáveis “id”, “grupo”, “prematuridade”.

```
dados_s <- dplyr::select(dados, id, grupo, prematuridade)
```

Veja em `View(dados_s)` que essa base de dados só contém as colunas selecionadas.

Sempre quando queremos retirar algo do banco de dados, colocamos um “-” antes da variável. No exemplo abaixo, queremos retirar “igp_semana” e “igp_dia”.

```
dados_s1 <- dplyr::select(dados, -igp_semana, -igp_dia)
```

Veja em `View(dados_s1)` que essa base de dados só contém as colunas selecionadas.

Com a função `arrange()`, conseguimos ordenar a base de acordo com uma ou mais colunas. Para gerar uma ordem decrescente de alguma variável, utilizamos a função `desc` como segue:

```
dados_cres <- dplyr::arrange(dados, alt)
```

```
dados_decres <- dplyr::arrange(dados, desc(alt))
```

No primeiro código acima, ordenamos a planilha em ordem crescente pela altura. Já no segundo código, ordenamos a planilha em ordem decrescente pela altura (a primeira observação na planilha `dados_decres` é aquela cuja gestante é a mais alta).

3.8 Exercícios

3.8.1 Teóricos

1. Por que é importante ter a população bem definida?
2. Por que é importante pensar em todo planejamento amostral ao invés de pensar apenas no cálculo do tamanho amostral?
3. O que é uma variável confundidora?
4. Classifique as seguintes variáveis:
 - a. Sexo
 - b. Altura
 - c. Peso

- d. Fuma (sim ou não)
- e. Tolerância ao cigarro (indiferente, incomoda pouco, incomoda muito)
- f. Consumo de café (nunca, 1 a 2 vezes por semana, 3 a 6 vezes por semana, uma vez por dia, > 1 vez por dia)
- g. Horas de atividade física por semana

3.8.2 Práticos no R

5. Considere a base de dados “dados_gemelares”:
 - a. Crie a variável `igp` (idade gestacional do parto) obtida ao somar `igp_semana` e `igp_dia/7`.
 - b. Crie a variável `indicador_aborto` (sim e não) - sim se `aborto >=1` e não se `aborto=0`.
 - c. Crie a variável `primipara` (sim e não) - não se `para >=1` e sim se `para=0`.
 - d. Obtenha uma nova base de dados ao considerar só os casos `primigestas` (note que queremos filtrar quem é `primigesta=sim`).
 - e. Obtenha uma nova base de dados ao considerar só os casos que não fumam e não fazem uso de álcool.
6. Considere agora a base de dados “dados_gemelares_2”. Realize uma análise de consistência similar a que realizamos para a base de dados “dados_gemelares” na Seção 3.6.

Chapter 4

Análise exploratória dos dados

Neste capítulo consideramos a análise descritiva das variáveis da base de dados de interesse.

A análise descritiva pode ser vista como o conjunto de técnicas numéricas e gráficas utilizadas para detectar padrões, resumir informação e apresentar visivelmente características de um conjunto de dados. Assim, podemos identificar (Morettin and Singer, 2020):

- i) qual a frequência com que cada valor aparece no conjunto de dados ou seja, qual a distribuição de frequências dos dados?
- ii) quais são alguns valores típicos do conjunto de dados, como mínimo e máximo?
- iii) qual seria um valor para representar a posição (ou localização) central do conjunto de dados?
- iv) qual seria uma medida da variabilidade ou dispersão dos dados?
- v) existem valores atípicos ou discrepantes (*outliers*) no conjunto de dados?
- vi) os dados podem ser considerados simétricos?

Para isso, utilizamos:

- tabelas de frequências;
- medidas para resumir os dados;
- gráficos.

As técnicas empregadas na análise descritiva dependem do tipo de variáveis que compõem o conjunto de dados em questão. Por isso, recomendo que reveja a Seção 3.3 sobre os tipos de variáveis.

4.1 Tabelas de frequências

Uma tabela contendo as frequências absolutas (número de casos) e/ou relativas (número de casos relativo ao total) para cada categoria da variável qualitativa é chamada de distribuição de frequências dessa variável.

Ao considerar a base de dados de gestações gemelares, vamos contruir uma tabela de frequências para a variável “prematuridade”. No R, há diversas maneiras de obter uma tabela de frequências. Vamos utilizar aqui a função `freq` do pacote `summarytools`.

```
library(summarytools)
```

```
summarytools::freq(dados$prematuridade, cumul=FALSE)
```

```
## Frequencies
##
##           Freq  % Valid  % Total
## -----
##      nao      44    42.31    42.31
##      sim      60    57.69    57.69
##      <NA>      0      0.00      0.00
##      Total    104   100.00   100.00
```

Pela tabela acima, observamos que quase 58% dos nascimentos gemelares foram prematuros e que não há observações faltantes para essa variável (linha `< NA >` está vazia). Caso queira que não apareça na tabela a linha com dados faltantes, só é necessário acrescentar o argumento `report.nas = FALSE` na função, como segue:

```
summarytools::freq(dados$prematuridade, cumul=FALSE,
                   report.nas=FALSE)
```

```
## Frequencies
##
##           Freq  %
## -----
##      nao      44  42.31
##      sim      60  57.69
##      Total    104 100.00
```

Observe também que colocamos o argumento `cumul=FALSE`, indicando que não temos interesse em acrescentar na tabela a coluna com as frequências relativas acumuladas. Essa coluna é útil quando a variável é qualitativa ordinal, o que não é o caso da prematuridade.

Outra dica importante é utilizar a função `view` (dessa vez tudo minúsculo). Veja que, ao utilizar `view(freq(dados$prematuridade, cumul=FALSE, report.nas=FALSE))` a tabela aparecerá mais bonita no menu “Viewer” do RStudio. Dali você pode

copiar para o destino de interesse.

4.2 Medidas resumo

Se utilizarmos uma tabela de frequências para uma variável quantitativas (especialmente no caso de variáveis contínuas), obteríamos frequências muito pequenas (em geral 1) para os diversos valores da variável, deixando de atingir o objetivo de resumir os dados.

Nesse sentido, apresentamos aqui medidas resumo que podem ser utilizadas para variáveis quantitativas. Essas serão divididas entre medidas de posição e medidas de dispersão.

4.2.1 Medidas de posição

As medidas de posição, como o nome diz, indicam posições de interesse da variável. Consideramos aqui as seguintes medidas: valor mínimo, valor máximo, moda, média, mediana e percentis.

O **mínimo** é o menor valor observado e **máximo** é o maior valor observado.

A moda, a média e a mediana são todas **medidas de posição central**: medidas que buscam descrever um valor típico que a variável tende a apresentar.

A **moda** é o(s) valor(es) mais frequente(s). Vale citar que a moda também pode ser usada para variáveis qualitativas.

A **média** é a medida obtida ao somar todos os valores da variável dividida pelo número de observações. Podemos interpretar a média como sendo um ponto de equilíbrio: os valores da variáveis são representadas como pesos de mesma massa posicionados sobre uma reta de massa desprezível nas posições referentes aos valores da variável em questão.

Já a **mediana** divide os dados de forma que 50% deles são menores e 50% deles são maiores que a mediana.

Para considerar dados de posição não centrais, podemos citar os quantis. Um quantil de ordem p é um valor da variável que deixa $100p\%$ ($0 < p < 1$) das observações a sua esquerda, ou seja, são menores que ele.

São alguns quantis conhecidos:

- **Percentis** - valores inteiros de $100p\%$. O percentil 20, por exemplo, é o valor da variável que 20% das observações apresentam menor que ele.
- **Decis** - dados divididos em 10 partes iguais. O primeiro decil, por exemplo, é o percentil 10 e o sexto decil é o percentil 60.
- **Quartis** - dados divididos em 4 partes iguais. O primeiro quartil é o percentil 25, o segundo quartil é o percentil 50 e o terceiro quartil é o percentil 75. Vale notar que o segundo quartil é a mediana.

4.2.2 Medidas de dispersão

As medidas de dispersão são valores que quantificam quão dispersos os dados são. Consideramos aqui as seguintes medidas: amplitude, intervalo interquartil, variância e desvio padrão.

A **amplitude** é a diferença entre o valor máximo e o valor mínimo. O **intervalo interquartil** é a diferença entre o terceiro e o primeiro quartil, ou seja, é a amplitude entre os 50% dos dados centrais.

Queremos uma medida de dispersão que não considere apenas dois valores da amostra (mínimo e máximo ou primeiro e terceiro quartis) e sim todos os dados. Uma medida bastante intuitiva seria considerar a soma dos desvios das observações em torno da média. Mas aí temos um problema: a soma dos desvios da média é sempre zero! Isso acontece porque sempre há desvios positivos e negativos que se anulam. Uma solução para essa questão é considerar alguma função que considere apenas o valor do desvio e não o seu sinal. Uma função candidata é a função quadrática (lembre que, por exemplo, $(-2)^2 = 4$). Nessa construção surge a **variância**: soma dos desvios quadrados dividida pelo total de observações, ou seja, a média dos desvios quadrados. Assim, a variância quantifica o quanto os dados estão dispersos da média, em média.

Como a unidade de medida da variância é o quadrado da unidade de medida da variável correspondente, convém definir outra medida de dispersão que mantenha a unidade de medida original. Uma medida com essa propriedade é a raiz quadrada da variância, conhecida por **desvio padrão**.

No R, para obter essas medidas resumo vamos utilizar a função `descr` também do pacote `summarytools`. No comando abaixo pedimos ao R as medidas descritivas da variável quantitativa “medida_colo”.

```
descr(dados$medida_colo)
```

```
## Descriptive Statistics
## value
## N: 104
##
##                               value
## -----
##           Mean      24.67
##        Std.Dev      9.93
##           Min       2.70
##           Q1      17.90
##          Median     24.75
##           Q3      33.05
##           Max      46.30
##           MAD      11.27
##           IQR      15.07
##           CV       0.40
```

```
##           Skewness    -0.03
##          SE.Skewness    0.24
##           Kurtosis    -0.74
##           N.Valid    104.00
##           Pct.Valid    100.00
```

Veja que a função retorna outras medidas resumo além daquelas citadas anteriormente. Se quiser uma tabela com algumas medidas resumo, podemos informar ao R por meio do argumento `stats`. Ainda, se quisermos que na tabela as medidas resumo fiquem na coluna, usamos o argumento `transpose = TRUE`, como segue:

```
descr(dados$medida_colo, stats = c("min", "mean", "med", "sd", "max"),
      transpose = TRUE) #sd é o desvio padrão
```

```
## Descriptive Statistics
## value
## N: 104
##
##           Min      Mean   Median   Std.Dev      Max
## -----
##          value  2.70   24.67   24.75     9.93   46.30
```

Outro pacote bastante interessante para medidas descritivas é o `modelsummary`. Destacamos algumas funções desse pacote:

- `datasummary_skim`: retorna as medidas descritivas das variáveis do banco de dados a depender do tipo identificado no argumento `type=` (categorical ou numeric);
- `datasummary`: retorna as medidas descritivas das variáveis a depender de como monta os argumentos da função, permitindo retornar as medidas descritivas das variáveis quantitativas de interesse por categorias de outra(s) variável(is).

```
library(modelsummary)
```

Ao usar a função `datasummary_skim`, vamos obter as medidas descritivas das variáveis quantitativas (argumento `type = "numeric"`) e das variáveis qualitativas (argumento `type = "categorical"`), respectivamente:

```
datasummary_skim(
  dados,
  type = "numeric",
  histogram = FALSE
)
```

```
datasummary_skim(
  dados,
  type = "categorical"
```

```
)
```

Agora vamos comentar sobre a função mais interessante desse pacote, a função `datasummary`. Suponha que eu tenho interesse em obter as medidas descritivas das variáveis “igp”, “idade”, “imc”, “medida_colo” e “num_contra_ctg” por grupo (progesterona ou placebo), apresentando as seguintes medidas descritivas: média, mediana, desvio padrão, mínimo, máximo e tamanho da amostra válido (sem considerar observações faltantes para a variável em questão).

```
### Essas funcoes abaixo são auxiliares para calcular as descritivas
#em cenário de presença de dados faltantes
media <- function(x) mean(x, na.rm = TRUE)
medi <- function(x) median(x, na.rm = TRUE)
dp <- function(x) sd(x, na.rm = TRUE)
mini <- function(x) min(x, na.rm = TRUE)
maxi <- function(x) max(x, na.rm = TRUE)
n <- function(x) sum(!is.na(x))
```

```
datasummary((igp+idade+imc+medida_colo+num_contra_ctg)~
             grupo*(n+media+dp+mini+medi+maxi), data = dados)
```

Agora veja como fica se eu considerar as medidas descritivas das variáveis quantitativas por duas variáveis qualitativas:

```
datasummary(primigesta*grupo~(igp+idade+imc+medida_colo+
                             num_contra_ctg)*(n+media+dp+mini+medi+maxi), data = dados)
```

Para mais detalhes sobre as medidas descritivas veja o Capítulo 3 de Morettin e Singer (Morettin and Singer, 2020).

4.3 Tabelas cruzadas - duas variáveis qualitativas

Tabelas cruzadas ou tabelas de contingência são tabelas que apresentam frequências de duas variáveis qualitativas conjuntamente.

No R, para obter tabelas cruzadas vamos utilizar a função `ctable` também do pacote `summarytools`. No comando abaixo pedimos ao R uma tabela cruzada entre as variáveis qualitativas “grupo” e “prematuridade”.

```
ctable(dados$grupo,y=dados$prematuridade,prop="t")
```

```
## Cross-Tabulation, Total Proportions
## dados$grupo * dados$prematuridade
##
```

```
## -----
##              dados$prematuridade      nao      sim      Total
```



```
## dados$grupo
## placebo          19 (18.3%)   33 (31.7%)   52 ( 50.0%)
## progest          25 (24.0%)   27 (26.0%)   52 ( 50.0%)
## Total            44 (42.3%)   60 (57.7%)  104 (100.0%)
## -----
```

No argumento `prop=` indica como será o cálculo das porcentagens apresentadas entre parênteses. Se o interesse for a porcentagem com relação ao total, o argumento é `prop="t"`. Se você desejar que a porcentagem seja calculada com relação às categorias da variável que está na linha é `prop="r"`:

```
ctable(dados$grupo,y=dados$prematividade,prop="r")
```

```
## Cross-Tabulation, Row Proportions
## dados$grupo * dados$prematividade
##
## -----
##          dados$prematividade      nao      sim      Total
## dados$grupo
## placebo          19 (36.5%)   33 (63.5%)   52 (100.0%)
## progest          25 (48.1%)   27 (51.9%)   52 (100.0%)
## Total            44 (42.3%)   60 (57.7%)  104 (100.0%)
## -----
```

E, por fim, se for de interesse que a porcentagem seja calculada com relação às categorias da variável que está na coluna é `prop="c"`:

```
ctable(dados$grupo,y=dados$prematividade,prop="c")
```

```
## Cross-Tabulation, Column Proportions
## dados$grupo * dados$prematividade
##
## -----
##          dados$prematividade      nao      sim      Total
## dados$grupo
## placebo          19 ( 43.2%)   33 ( 55.0%)   52 ( 50.0%)
## progest          25 ( 56.8%)   27 ( 45.0%)   52 ( 50.0%)
## Total            44 (100.0%)   60 (100.0%)  104 (100.0%)
## -----
```

4.4 Gráficos

Um gráfico pode ser a maneira mais adequada para resumir e apresentar um conjunto de dados. Tem a vantagem de facilitar a compreensão de uma determinada situação que queira ser descrita, permitindo uma interpretação rápida e visual das suas principais características.

A visualização dos dados é uma etapa importantíssima da análise estatística,

pois é também a partir dela que criamos a intuição necessária para escolher o teste ou modelo mais adequado para o nosso problema.

No Capítulo 3 do livro de Morettin e Singer também tem um conteúdo bastante interessante sobre gráficos e recomendamos sua leitura para a apresentação dos gráficos mais tradicionais em Estatística do ponto de vista mais teórico e, no que segue, discutimos como obter alguns gráficos no R.

4.4.1 Pacote ggplot2

Um pacote maravilhoso para gráficos no R é o `ggplot2`. A ideia por trás desse pacote é que o fato que um gráfico é um mapeamento dos dados a partir de atributos estéticos (cores, formas, tamanho) de formas geométricas (pontos, linhas, barras).

```
library(ggplot2)
```

Atributos estéticos

A função `aes()` descreve como as variáveis são mapeadas em aspectos visuais (qual variável será representada no eixo x, qual será representada no eixo y, a cor e o tamanho dos componentes geométricos e etc) de formas geométricas definidas pelos geoms. Os aspectos que podem ou devem ser mapeados vão depender do tipo de gráfico que estamos querendo construir.

Aspectos visuais mais utilizados:

- `color`: altera a cor de formas que não têm área (pontos e retas);
- `fill`: altera a cor de formas com área (barras, caixas, densidades, áreas);
- `size`: altera o tamanho de formas;
- `type`: altera o tipo da forma, geralmente usada para pontos;
- `linetype`: altera o tipo da linha.

Formas geométricas

Os geoms definem qual forma geométrica será utilizada para a visualização das observações. A função `geom_point()` gera gráficos de dispersão transformando pares (x,y) em pontos. Veja a seguir outros geoms bastante utilizados:

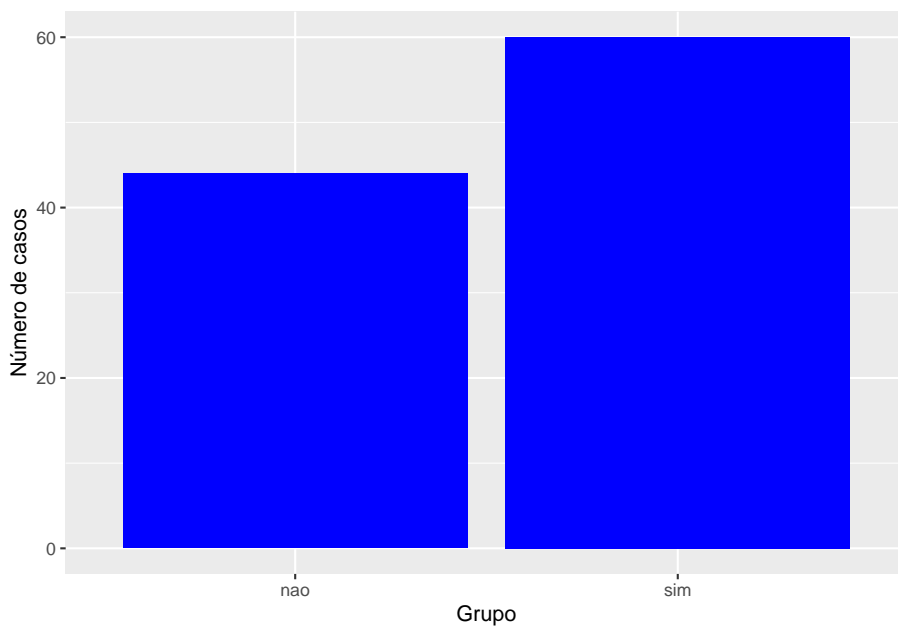
- `geom_line`: para retas definidas por pares (x,y);
- `geom_abline`: para retas definidas por um intercepto e uma inclinação;
- `geom_hline`: para retas horizontais;
- `geom_bar`: para barras;
- `geom_histogram`: para histogramas;
- `geom_boxplot`: para boxplots;

- `geom_density`: para densidades;
- `geom_area`: para áreas.

Vamos fazer alguns gráficos para as variáveis da base de dados `gestacoes gemelares`.

Gráfico de barras - analisando uma variável qualitativa:

```
ggplot(dados, aes(x = prematuridade)) +  
  geom_bar(fill = "blue") +  
  labs(x = "Grupo", y = "Número de casos")
```



```
#No eixo y é percentagem  
ggplot(dados, aes(x = prematuridade)) +  
  geom_bar(aes(y = (..count..)/sum(..count..)), fill="purple") +  
  scale_y_continuous(labels=scales::percent) +  
  ylab("Porcentagem")
```

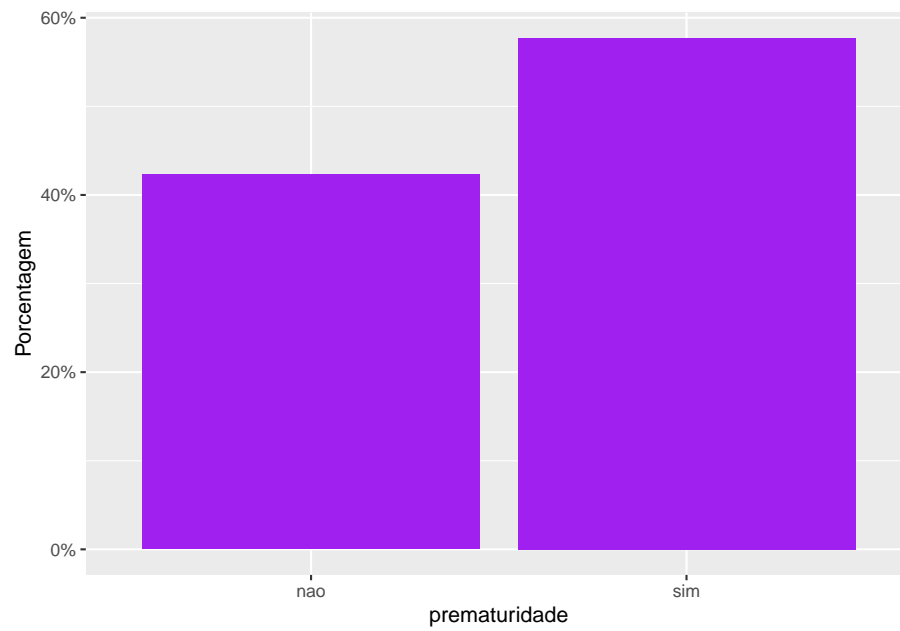
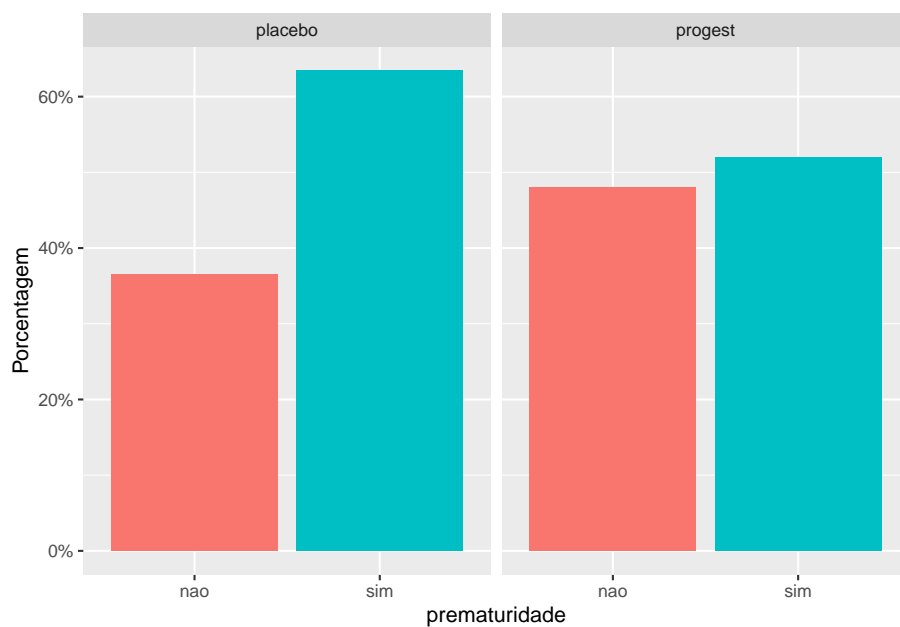


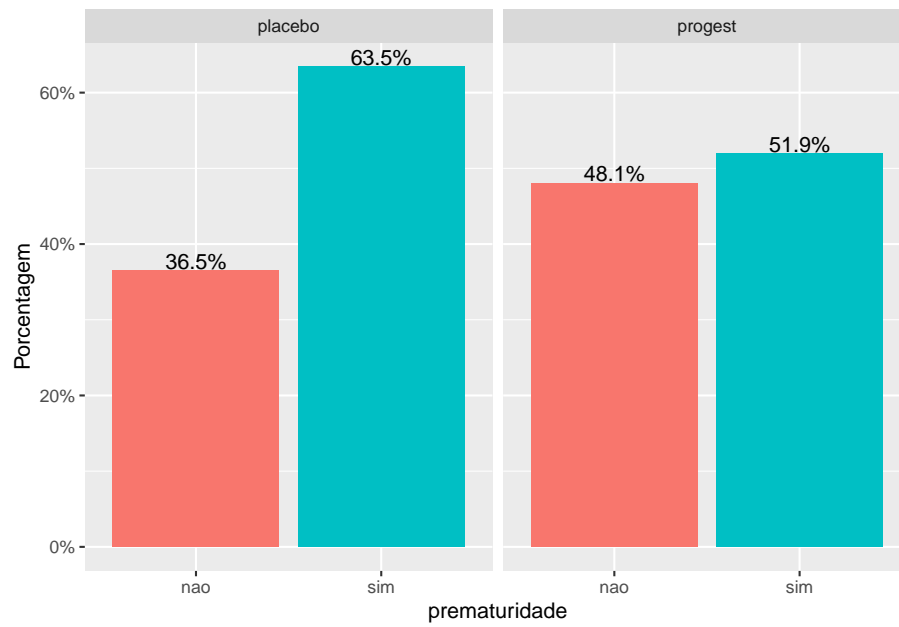
Gráfico de barras - analisando duas variáveis qualitativas:

```
ggplot(dados, aes(prematuridade, group = grupo)) +  
  geom_bar(aes(y = ..prop.., fill = factor(..x..)), stat="count") +  
  scale_y_continuous(labels=scales::percent) +  
  theme(legend.position = "none") +  
  ylab("Porcentagem") +  
  facet_grid(~grupo)
```



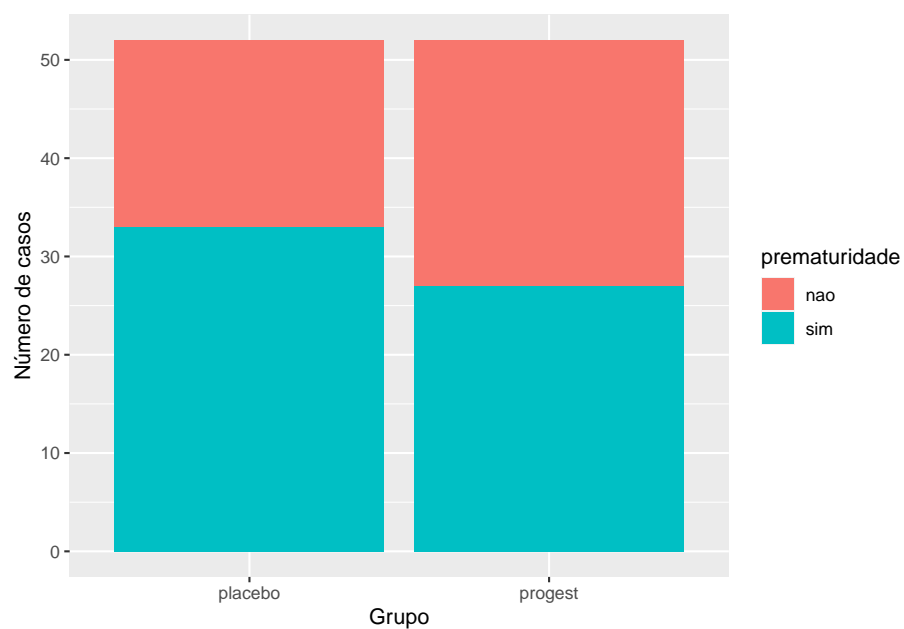
Mesmo gráfico acima, mas agora com a porcentagem escrito no gráfico:

```
ggplot(dados, aes(x=prematuidade, group = grupo)) +
  geom_bar(aes(y = ..prop.., fill = factor(..x..)), stat="count") +
  geom_text(aes( label = scales::percent(..prop..),
                y= ..prop.. ), stat= "count", vjust = -.1) +
  labs(y = "Porcentagem", fill=" ") +
  scale_y_continuous(labels = scales::percent) +
  theme(legend.position = "none")+
  facet_grid(~grupo)
```



Outra forma de fazer um gráfico de barras para duas variáveis qualitativas:

```
ggplot(dados, aes(x = grupo, fill = prematividade)) +  
  geom_bar() +  
  scale_fill_hue() +  
  labs(x = "Grupo", y = "Número de casos")
```

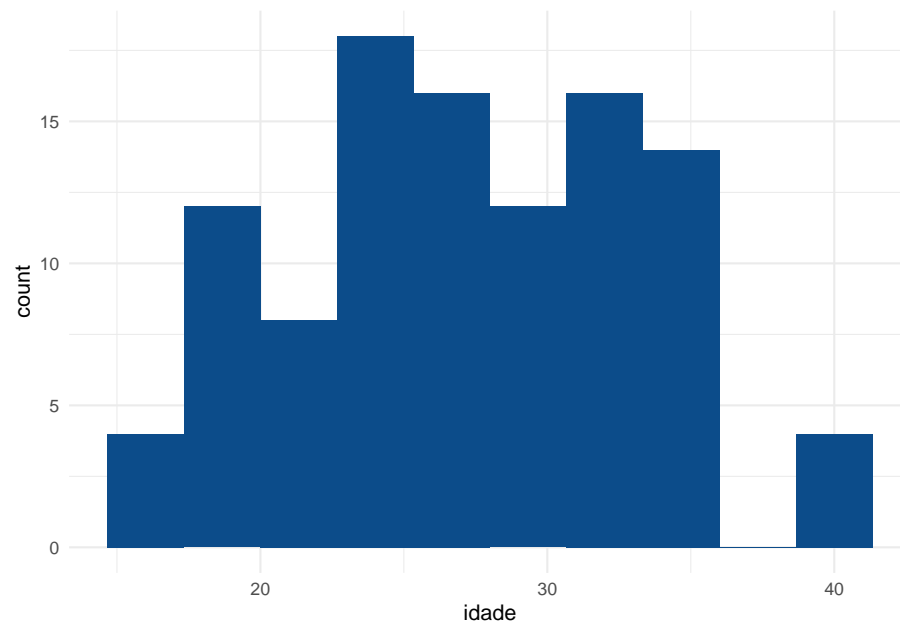


Veja o site para mais detalhes sobre gráfico de barras: <http://www.sthda.com/english/wiki/ggplot2-barplots-quick-start-guide-r-software-and-data-visualization#barplot-of-counts>.

Gráficos para variáveis quantitativas - só uma variável quantitativa

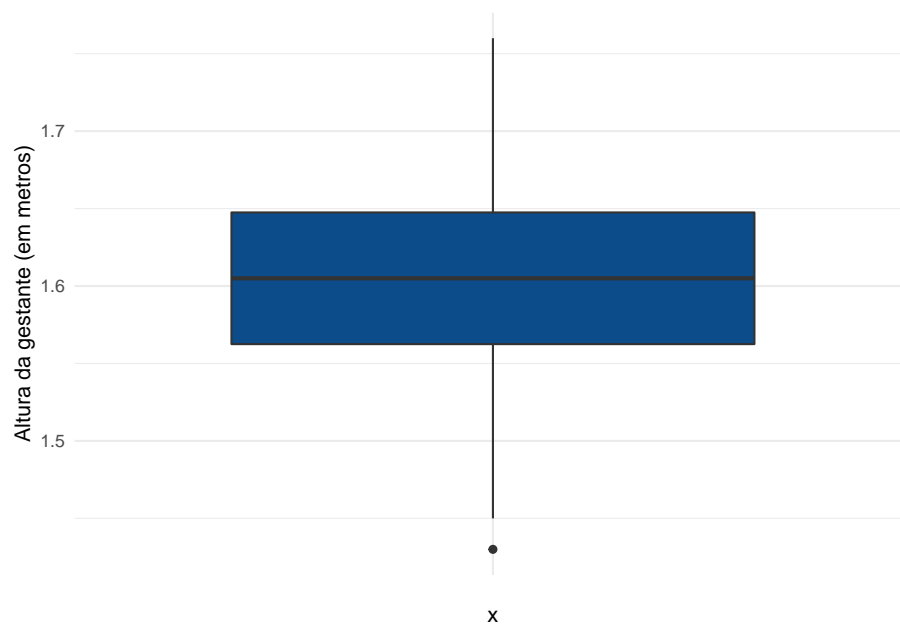
Histograma:

```
ggplot(dados) +  
  aes(x = idade) +  
  geom_histogram(bins = 10L, fill = "#0c4c8a") +  
  theme_minimal()
```



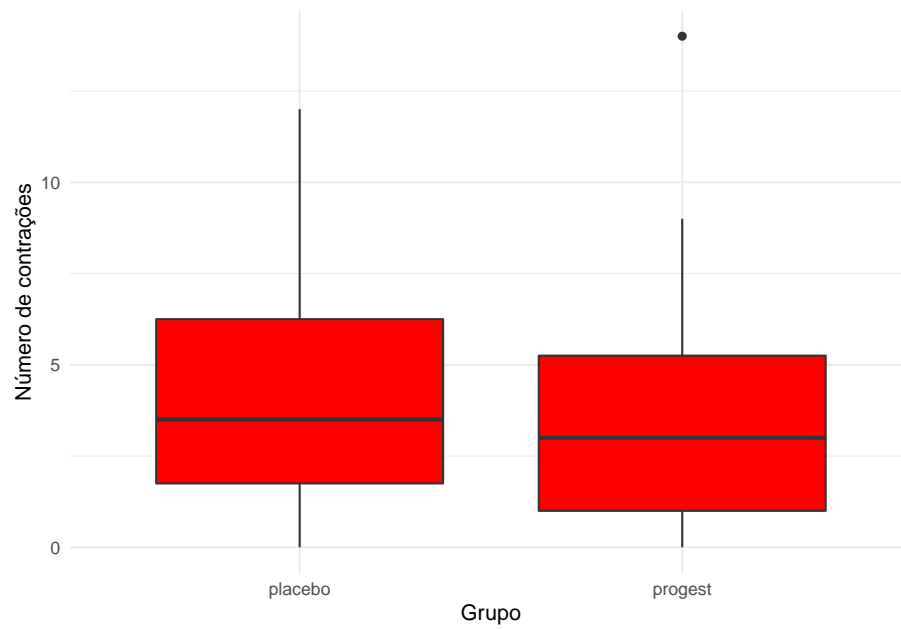
Boxplot:

```
ggplot(dados) +  
  aes( y = alt, x = "") +  
  geom_boxplot(fill = "#0c4c8a") +  
  labs(y = "Altura da gestante (em metros)") +  
  theme_minimal()
```

Gráficos para variáveis quantitativas - variável quantitativa por uma variável de grupo:

```
ggplot(dados) +  
  aes(x = grupo, y = num_contra_ctg) +  
  geom_boxplot(fill = "red") +  
  labs(x = "Grupo", y = "Número de contrações") +  
  theme_minimal()
```



Gráficos para variáveis quantitativas - duas variáveis quantitativas:

Gráfico de dispersão:

```
ggplot(dados) +  
  aes(x = imc, y = medida_colo) +  
  geom_point(size = 1.84, colour = "#4292c6") +  
  labs(x = "IMC", y = "Medida do colo") +  
  theme_minimal()
```

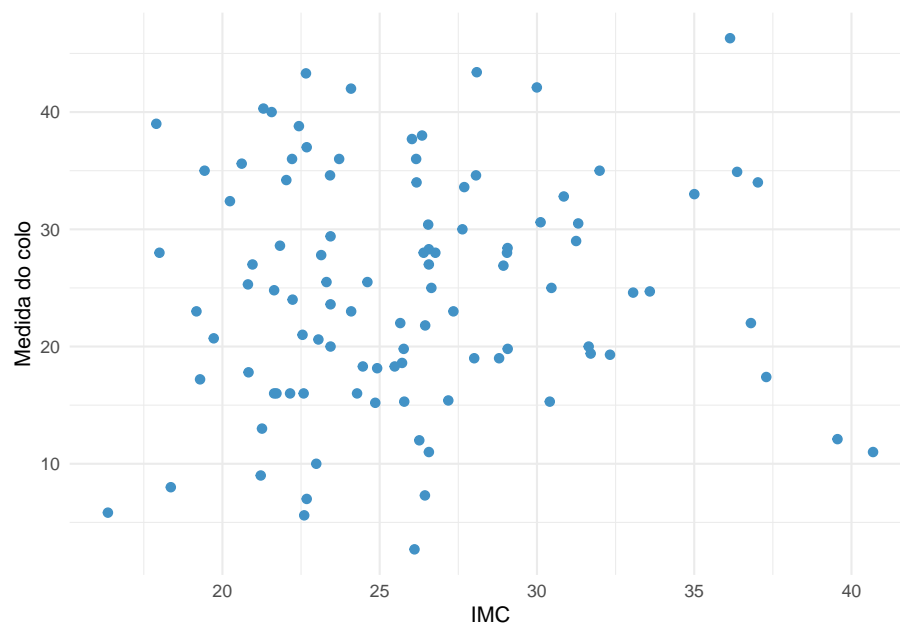
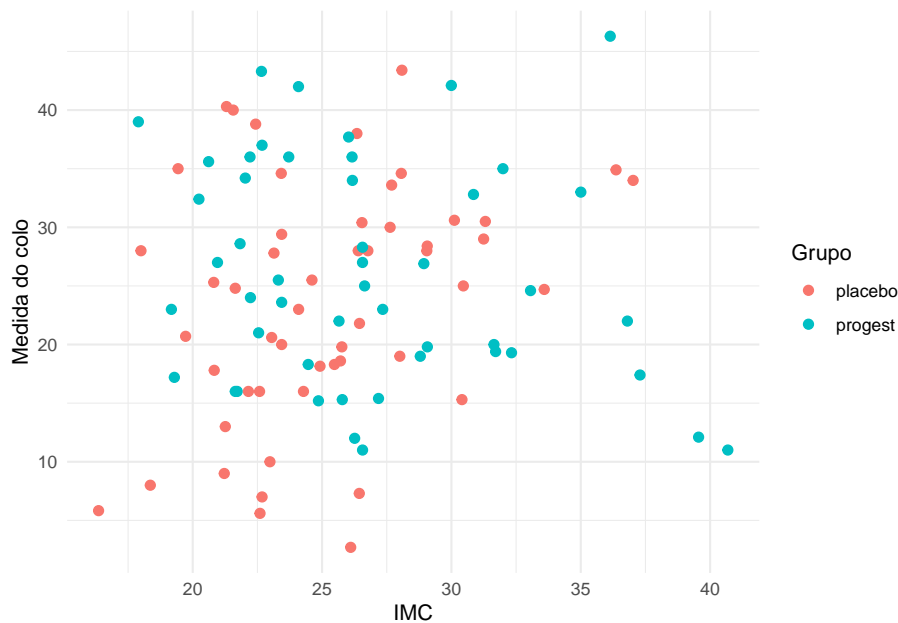


Gráfico de dispersão identificando o grupo (progesterona ou placebo):

```
ggplot(dados) +  
  aes(x = imc, y = medida_colo, colour = grupo) +  
  geom_point(size = 2.02) +  
  scale_color_hue() +  
  labs(x = "IMC", y = "Medida do colo", color = "Grupo") +  
  theme_minimal()
```



4.4.2 Pacote esquisse

O pacote **esquisse** disponibiliza um *dashboard* interativo para criação de gráficos por meio do **ggplot2**.

```
library(esquisse)
```

Ao rodar o código **esquisser()**, abrirá a janela da Figura 4.1, em que usuário escolhe a base de dados que trabalhará e ao finalizar nessa janela, abrirá a janela da Figura 4.2 para fazer os gráficos. Um bom tutorial sobre a utilização do pacote **esquisse**: <https://www.youtube.com/watch?v=VbzxNQAUBw>.

```
esquisse::esquisser()
```

4.5 Materiais complementares para análise exploratória dos dados

- Mercier F, Consalvo N, Frey N, Phipps A, Ribba B. From waterfall plots to spaghetti plots in early oncology clinical development. *Pharm Stat.* 2019 Oct;18(5):526-532. doi: 10.1002/pst.1944. Epub 2019 Apr 3. PMID: 30942559.
- Understanding waterfall plots. *J Adv Pract Oncol.* 2012 Mar;3(2):106-11. Gillespie TW1.

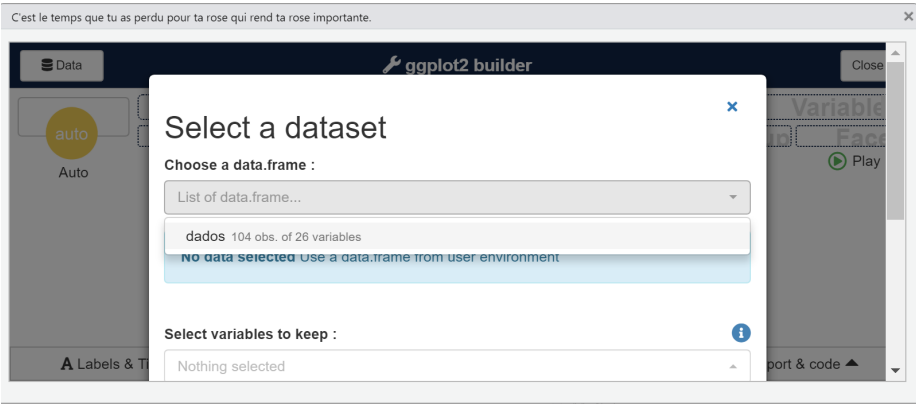


Figure 4.1: Primeira tela do esquisser

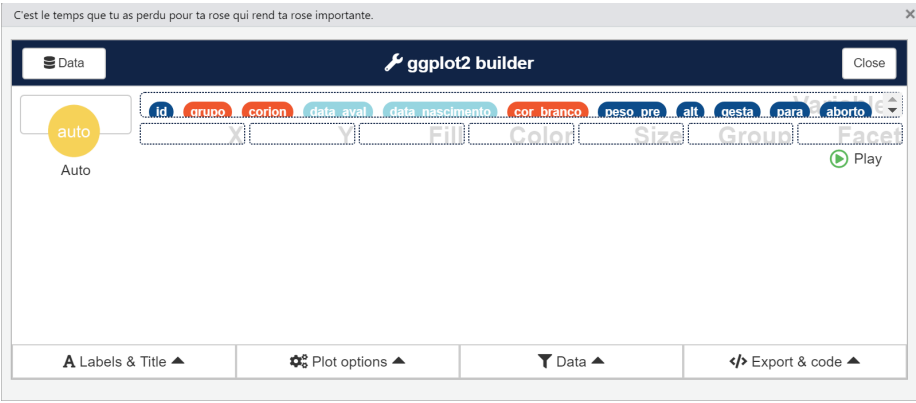


Figure 4.2: Segunda tela do esquisser

- Describing data: statistical and graphical methods. Radiology. 2002 Dec;225(3):622-8.Sonnad SS.3. In J, Lee S. Statistical data presentation. Korean Journal of Anesthesiology. 2017;70(3):267-276. doi:10.4097/kjae.2017.70.3.267.

4.6 Exercícios sobre análise exploratória dos dados

1. Realize uma análise exploratória dos dados “dados_gemelares”. Faça as análises que achar pertinente e que obrigatoriamente contenha os seguintes itens:
 - Apresente tabelas de frequências de todas as variáveis qualitativas;
 - Apresente tabelas com as medidas descritivas de todas as variáveis quantitativas;
 - Construa tabelas cruzadas de grupo (progesterona e placebo) com todas as outras variáveis quantitativas, apresentando a porcentagem por grupo (se a variável de grupo estiver na linha, pedir a porcentagem por linha).
 - Selecione só as gestações dicorônicas. Desses casos selecionados, apresente tabela com as medidas descritivas das variáveis quantitativas por grupo (progesterona e placebo);
 - Apresente os seguintes gráficos: - Gráfico de barras conjuntamente de grupo e primigesta (gráfico de barras para duas variáveis). - Boxplot da medida do colo por grupo (progesterona e placebo). - Gráfico de dispersão entre número de contrações (eixo y) e medida do colo (eixo x). - Refaça o gráfico de dispersão anterior, mas agora identificando o grupo (progesterona e placebo) pela cor do ponto.
2. Realize uma análise exploratória dos dados “dados_gemelares_2”. Faça as análises que achar pertinente e que obrigatoriamente contenha os seguintes itens:
 - Apresente tabelas de frequências de todas as variáveis qualitativas;
 - Apresente tabelas com as medidas descritivas de todas as variáveis quantitativas;
 - Construa uma tabela cruzada entre EDPS_antes_SN e EDPS_depois_SN;
 - Apresente uma tabela com as medidas descritivas da variável tempo_amamentacao_meses por grupo de amamentação. Refaça essa tabela, mas agora só com os casos sem depressão antes do parto (filtrando os casos EDPS_antes_SN=não).
 - Faça boxplot do tempo_amamentacao_meses por grupo de amamentação.

4.6. EXERCÍCIOS SOBRE ANÁLISE EXPLORATÓRIA DOS DADOS 95

- Faça um gráfico de dispersão EDPS antes versus EDPS depois.

Observação importante: Interprete todas as análises realizadas.

Chapter 5

Aprendizado estatístico

EM CONSTRUÇÃO

DISCUSSÃO SOBRE APRENDIZADO SUPERVISIONADO E NÃO SUPERVISIONADO

PROBLEMAS DE CLASSIFICAÇÃO VERSUS REGRESSÃO

Bibliography

- Astle, S., Slater, D. M., and Thornton, S. (2003). The involvement of progesterone in the onset of human labour. *European Journal of Obstetrics & Gynecology and Reproductive Biology*, 108(2):177–181.
- Bussab, W. and MORETIN, P. (2004). Estatística básica. 5^a edição, editora sariva, s.
- de Oliveira, L. A., Brizot, M. L., Liao, A. W., Bittar, R. E., Francisco, R. P., and Zugaib, M. (2016). Prenatal administration of vaginal progesterone and frequency of uterine contractions in asymptomatic twin pregnancies. *Acta Obstetrica et Gynecologica Scandinavica*, 95(4):436–443.
- into Maternal, C. E. (2009). Child health (cemach) perinatal mortality 2007. *United Kingdom. London: CEMACH*.
- Magalhaes, M. and Lima, A. (2002). Noções de estatística e probabilidade. *São Paulo: EdUSP*.
- Morettin, P. A. and Singer, J. M. (2020). Introdução ciência de dados. *Texto Preliminar, IME-USP*.
- Silva, J. (1995). Prematuridade: aspectos obstétricos. *Neme B. Obtetricia básica*, 2:372–379.