R pragmático

Julio Trecenti

2016-10-19

Contents

Chapter 1

Setup

O minicurso "R pragmático" é baseado no `tidyverse` (universo "arrumado"), um conjunto de pacotes do R que auxiliam o estatístico / cientista de dados na execução de diversas tarefas corriqueiras de forma eficiente e unificada. Pense em eficiência, mas não no sentido de velocidade de execução de algoritmos, mas sim na velocidade de solução de problemas.

Atualmente, o melhor lugar para aprender sobre o `tidyverse` é no livro R for data science. Nesse minicurso abordamos partes desse livro e adicionamos outros, como práticas de modelgem preditiva e estudos de caso.

Público-alvo

- Estudantes de graduação em estatística que desejam ganhar tempo nos trabalhos da faculdade e entrar no mercado de trabalho com bons diferenciais.
- Profissionais do mercado de trabalho que desejam inserir o R mo fluxo de atividades do setor/empresa.
- Acadêmicos com interesse em tornar suas análises e códigos mais legíveis, reprodutíveis, eficientes e organizados.

Workflow das aulas:

- Aulas no laboratório de computação (CEC). Não precisa (mas pode) levar notebook.
- Exercícios durante as aulas.
- Leituras complementares e opcionais fora da sala de aula.

Requisitos básicos:

- Lógica de programação.
- Veja essa apresentação (aprox. 10 min) (slides: 13 ao 43).
- Leia esse post de blog (aprox. 5 min).
- Se quiser ganhar tempo, crie uma conta no Github.

Conteúdo:

- Primeiro dia (04/10): introdução ao `tidyverse`, o operador `pipe`, trabalhando textos com `stringr`, trabalhando datas com `lubridate`.
- Segundo dia (05/10): transformação de dados com `dplyr` e `tidyr`, visualização de dados com `ggplot2`.
- Terceiro dia (06/10): elaboração de relatórios com `knitr` e `rmarkdown`, modelagem preditiva (parte 1).
- Quarto dia (07/10) modelagem preditiva (parte 2), case studies e feedback. R for Data Science

Não vamos falar de:

- Programação eficiente com R. Para isso, veja esse livro, que aborda temas importantíssimos como *profiling*, paralelização, `Rcpp`.
- Estudos envolvendo "big data". Para isso estude sobre sparklyr e tensorflow e mongodb.

1.1 Diferenças entre C/C++ e R

Na análise realizada na Seção 3 notei que boa parte dos inscritos têm background em C/C++. Em uma comparação simples, o foco do C é eficiência e transparência, enquanto o do R é análise de dados e interatividade. Isso faz com que as duas linguagens sejam bem diferentes!

Na prática, temos que

- C é compilável, R é uma linguagem script.
- R é uma linguagem funcional. Por exemplo, `(`, `&` e `+` são funções do R.
- R é vetorizado. Observe esse sacrilégio

```
a <- c(1, 2, 3)
b <- c(1, 2, 3, 4, 5, 6)
a + b
```

```
## [1] 2 4 6 5 7 9
```

Sim, isso funciona! O que acontece aqui é o fenômeno da *recliclagem* de vetores do R. Caso não esteja acostumado com essas idiossincrasias do R, veja essa aula.

- Você raramente usará loops (`for`, `while`) no R. Eles são ineficientes e não combinam com o estilo funcional da linguagem. Busque sempre realizar as operações com vetores, pois a maioria delas são implementadas em C e, portanto, mais eficientes.

1.2 Pacotes

Se você não está no CEC, precisará instalar alguns pacotes para acompanhar o curso. Para instalar todas as dependências, rode

```
install.packages('devtools')
devtools::install_github('curso-r/ragmatic')
```

Para visualizar todos os documentos que compõem esse livro, acesse essa página.

Chapter 2

Princípios

2.1 O tidyverse

O `tidyverse` é um pacote do R, cuja única função é carregar outros pacotes do R. O conjunto desses pacotes forma o `tidyverse`. É considerado um "universo" a parte do R pois todas suas ferramentas possuem formas de uso consistentes e funcionam muito bem em conjunto.

Os princípios do `tidyverse` seguem abaixo.

1. **Eficiência algorítmica vs eficiência de trabalho**. Suposição: o tempo que o estatísco gasta pensando em como realizar uma operação é mais importante do que o tempo que o computador gasta para realizar um cálculo.
2. **Tidy data**. Princípio para arrumação de base de dados que resolve 90% dos problemas reais. O objetivo em *arrumação de dados* é extrair e transformar uma base de dados até que ela esteja em formato *tidy*. Essa é uma boa prática de análise de dados que economiza muito tempo em qualquer trabalho. Uma base de dados é considerada "tidy" se
   - Cada observação é uma linha do bd.
   - Cada variável é uma coluna do bd.
   - Cada dado está numa célula do bd.
3. **Utilização do operador %>% (pipe)**.

   > "No matter how complex and polished the individual operations are, it is often the quality of the glue that most directly determines the power of the system."
   > – Hal Abelson
4. **Aparato mnemônico**. Pacotes baseados em teoria e API consistentes.
5. **Minimalidade e funções puras**. Funções sem *side-effects*. Interagem com o mundo através de inputs e outputs. Encaixa perfeitamente com o princípio do pipe
6. *workflow* para ciência de dados

2.2 RStudio

O RStudio é a melhor IDE para usar o R. O programa possui diversas vantagens e praticamente nenhuma desvantagem. Caso tenha interesse em se ambientar e entender as características do RStudio, veja essa página.

Uma importante funcionalidade do RStudio é a possibilidade de criar projetos. Uma estrutura recomendada para organização de pacotes segue abaixo.

```
project/
  - README.Rmd    # Descrição do pacote
  - set-up.R      # Pacotes etc
  - R/            # Código R, organizado com 0-load.R, 1-tidy.R, 2-vis.R, ...
  - data/         # Dados (estruturados ou não)
  - figures/      # gráficos (pode ficar dentro de output/)
  - output/       # Relatórios em .Rmd, .tex etc
  - project.Rproj
```

Outra possível forma de estruturar um projeto é organizando-o como um pacote do R:

```
project/
  - README.md     # Descrição do pacote
  - DESCRIPTION   # Metadados estruturados do pacote e dependências
  - NAMESPACE     # importações e exportações do pacote
  - vignettes/    # Relatórios em .Rmd
  - R/            # Funções do R
  - data/         # Dados estruturados (tidy data)
  - data-raw/     # Dados não estruturados e arqs 0-load.R, 1-tidy.R, 2-vis.R, ...
  - project.Rproj
```

Para detalhes de como criar pacotes no R de forma eficiente, leia o r-pkgs. Recomendo a adoção de um critério consistente para organização de projetos. O estatístico não pode perder tempo com a estruturação das pastas, então é melhor forçar uma estrutura pré-fixada do que planejar a melhor forma de organização para cada projeto.

2.3 RMarkdown

O RMarkdown é um tipo de documento especial que contém tanto textos (em markdown) quanto códigos em R (em chunks). O markdown nada mais é do que um documento de texto com alguns padrões básicos de formatação, como negrito, itálico, títulos, subtítulos, itemização e referências cruzadas. Já os chunks são pedaços de códigos em R encapsulados por três crases “‘‘‘. Os códigos são executados sempre que o documento é processado para algum formato específico.

A utilização do RMarkdown para produção de relatórios é essencial para o estatístico pragmático. O RMarkdown possui diversas vantagens:

1. **Simplicidade e foco**. Obriga o usuário a focar na análise e não na formatação do documento.
2. **Versátil**. Pode ser utilizado para gerar documentos em LaTeX, Word, HTML e apresentações em beamer, pptx e HTML (de vários tipos). Pode ainda gerar sites, livros, dissertações de mestrado e até mesmo dashboards interativos.
3. **Reprodutível**. O RMarkdown nada mais é que um arquivo de texto. Além disso, ele tenta te obrigar a fazer o documento mais autocontido possível. Assim, um documento .Rmd é fácil de compartilhar e de ser utilizado pelo receptor. Lembre-se, o receptor pode ser o futuro você! Vale enfatizar que a reprodutibilidade é considerada como um dos princípios fundamentais para a ciência. Então só de usar RMarkdown, você já está colaborando com a ciência :)
4. **Eficiente**. É possível configurar e criar templates de análises para quaisquer tipos de aplicações e clientes.

Para detalhes sobre como utilizar o RMarkdown, leia aqui e aqui.

2.4 GitHub

O GitHub é uma plataforma online para compartilhar códigos. Projetos do GitHub são baseados no `git`, uma ferramenta de versionamento de software.

Utilizar o GitHub é uma boa prática de organizar projetos pois é uma forma de manter os códigos organizados e atualizados na web, sem o perigo de perder tudo acidentalmente. Esse site também é essencial para projetos colaborativos, pois aumenta a produtividade e permite que pessoas de todo lugar ajudem nos projetos. O `tidyverse` só é o que é hoje por conta do *social coding*.

Para detalhes, faça o data science toolbox.

2.5 Pipe

O operador *pipe* foi uma das grandes revoluções recentes do R, tornando a leitura de códigos mais lógica, fácil e compreensível. Este operador foi introduzido por Stefan Milton Bache no pacote `magrittr` e já existem diversos pacotes construidos para facilitar a sua utilização.

Basicamente, o operador `%>%` usa o resultado do seu lado esquerdo como primeiro argumento da função do lado direito. Só isso!

Para usar o operador `%>%`, primeiramente instale o pacote `magrittr`.

```
install.packages("magrittr")
```

e carregá-lo com a função `library()`

```
library(magrittr)
```

Feito isso, vamos testar o operador calculando a raiz quadrada da soma de alguns números.

```
x <- c(1, 2, 3, 4)
x %>% sum %>% sqrt
```

```
## [1] 3.162278
```

O caminho que o código acima seguiu foi enviar o objeto `x` como argumento da função `sum()` e, em seguida, enviar a saida da expressão `sum(x)` como argumento da função `sqrt()`. Observe que não é necessário colocar os parênteses após o nome das funções.

Se escrevermos esse cálculo na forma usual, temos o seguinte código:

```
sqrt(sum(x))
```

```
## [1] 3.162278
```

A princípio, a utilização do `%>%` não parece trazer grandes vantagens, pois a expressão `sqrt(sum(x))` é facilmente compreendida. No entanto, se tivermos um grande número de funções aninhadas, a utilização do `pipe` transforma um código confuso e difícil de ser lido em algo simples e intuitivo. Como exemplo, imagine que você precise escrever uma receita de um bolo usando o R, e cada passo da receita é uma função:

```
esfrie(asse(coloque(bata(acrescente(recipiente(rep("farinha", 2), "água", "fermento", "leite", "óleo"),
```

Tente entender o que é preciso fazer. Nada fácil, correto? Agora escrevemos usando o operador `%>%`:

```
recipiente(rep("farinha", 2), "água", "fermento", "leite", "óleo") %>%
  acrescente("farinha", até = "macio") %>%
  bata(duração = "3min") %>%
  coloque(lugar = "forma", tipo = "grande", untada = T) %>%
  asse(duração = "50min") %>%
  esfrie("geladeira", "20min")
```

Agora o código realmente parece uma receita de bolo.

Para mais informações sobre o `pipe` e exemplos de utilização, visite a página Ceci n'est pas un pipe.

Chapter 3

Análise exploratória: inscritos no curso

```
library(magrittr)
library(tidyverse)
library(stringr)
library(lubridate)
library(forcats)
```

3.1 Objetivos

- Verificar a bagagem dos alunos.
- Verificar se há concentração de inscritos da graduação.
- Verificar se há diferenças entre a turma do CEC e do Jacy.

3.2 Base de dados

```
d_alunos <- read_csv('data/lista_anon.csv')
glimpse(d_alunos)
```

```
## Observations: 95
## Variables: 7
## $ Timestamp                                                                  <chr> ...
## $ Universidade:                                                              <chr> ...
## $ Estou no(a):                                                               <chr> ...
## $ Curso:                                                                     <chr> ...
## $ 1. Em que opção você situaria seu conhecimento em R?                       <chr> ...
## $ 2. Você considera ter conhecimento intermediário/avançado em quais linguagens abaixo? <chr> ...
## $ 3. Você atua no mercado de trabalho?                                       <chr> ...
```

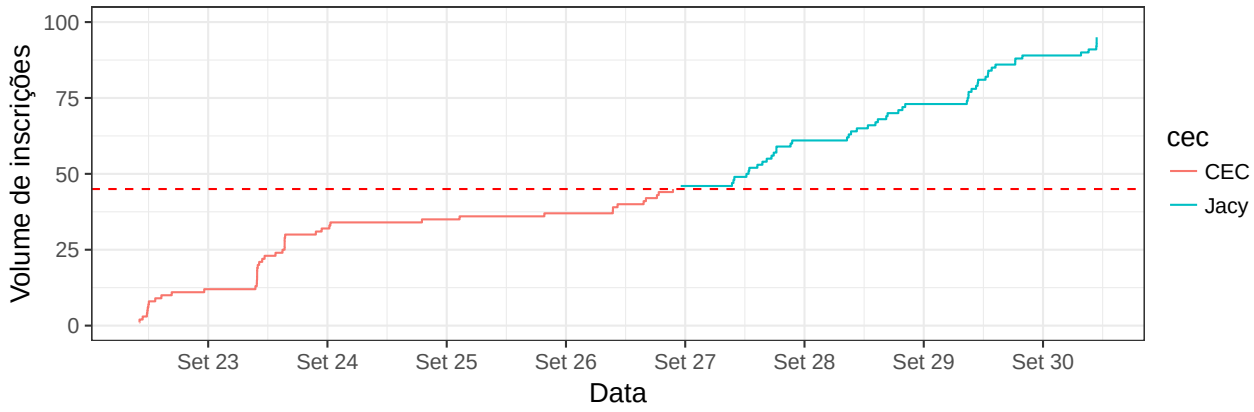
3.3 Data tidying

```
d_alunos %<>%
  mutate(Timestamp = mdy_hms(Timestamp)) %>%
  rownames_to_column('id_pessoa') %>%
  mutate(cec = id_pessoa %in% as.character(1:45)) %>%
  gather(pergunta, resposta, matches('^[0-9]')) %>%
  renomear() %>%
  spread(pergunta, resposta) %>%
  mutate(uni = with(., case_when(
    str_detect(universidade, re_usp) ~ 'USP',
    str_detect(universidade, re_ufscar) ~ 'UFSCar',
    str_detect(universidade, re_unip) ~ 'UNIP',
    TRUE ~ 'Outra'
  ))) %>%
  mutate(esc = with(., case_when(
    str_detect(estou_no_a, 'raduado|formado|Mestrado|Pós') ~ 'Formado / Pós',
    str_detect(estou_no_a, 'Graduação') ~ 'Graduação',
    TRUE ~ 'Outra'
  ))) %>%
  mutate(ime = str_detect(universidade, re_ime),
         cec = if_else(cec, 'CEC', 'Jacy'))
```

3.4 Alguns gráficos

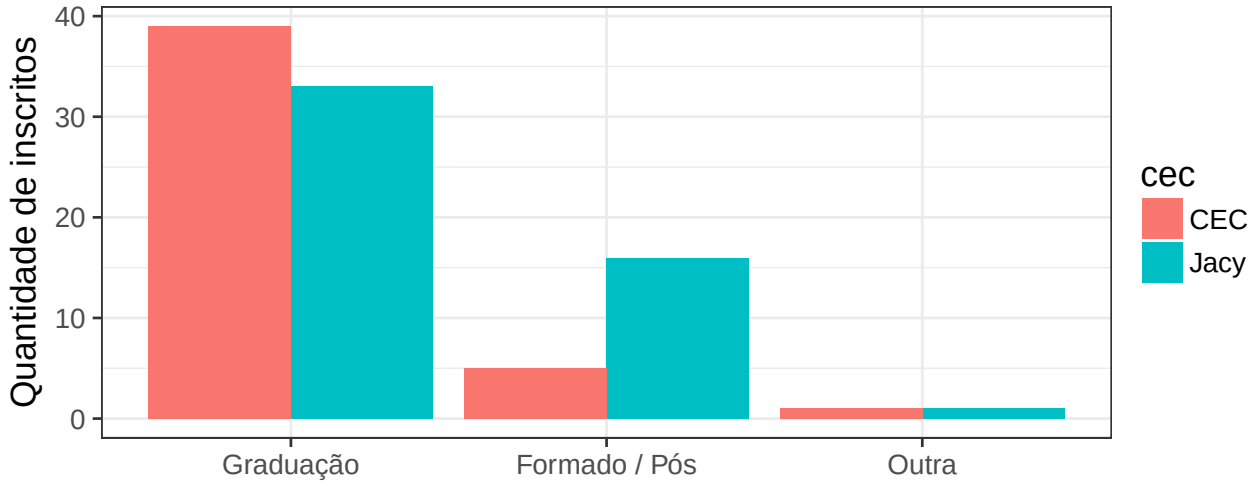
Volume de inscrições no tempo

```
d_alunos %>%
  arrange(timestamp) %>%
  mutate(um = 1, inscricoes = cumsum(um)) %>%
  ggplot(aes(x = timestamp, y = inscricoes, colour = cec)) +
  geom_step() +
  geom_hline(yintercept = 45, colour = 'red', linetype = 2) +
  scale_x_datetime(breaks = scales::date_breaks('1 day'),
                   labels = scales::date_format('%b %d')) +
  scale_y_continuous(breaks = 0:4 * 25, limits = c(0, 100)) +
  theme_bw(14) +
  xlab('Data') +
  ylab('Volume de inscrições')
```



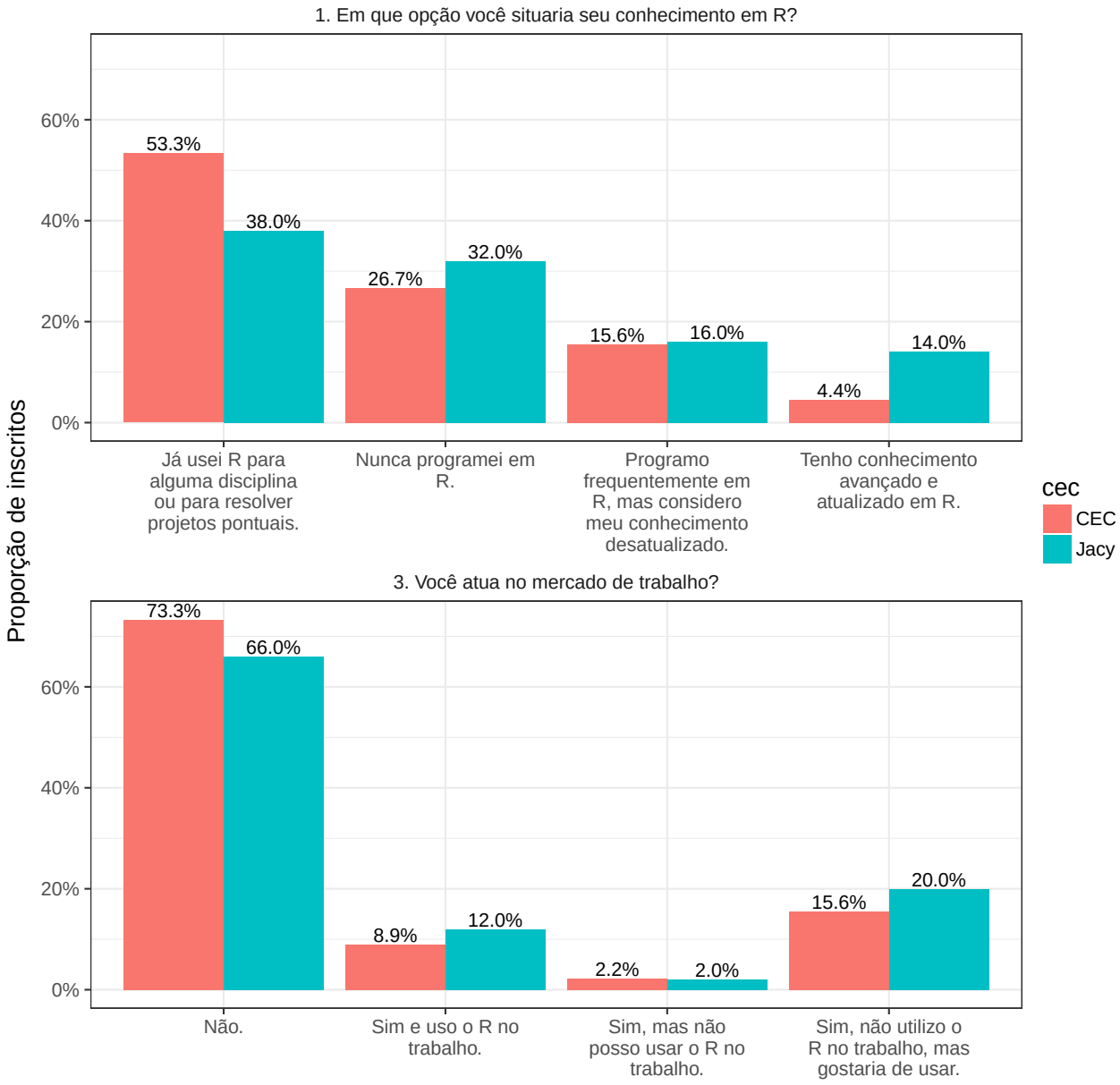
Escolaridade dos inscritos

```
d_alunos %>%
  replace_na(list(esc = 'Outra')) %>%
  mutate(esc = fct_infreq(esc)) %>%
  ggplot(aes(x = esc, fill = cec)) +
  geom_bar(position = 'dodge') +
  theme_bw(14) +
  xlab('') +
  ylab('Quantidade de inscritos')
```



Perguntas 1 e 3: sobre utilização do R.

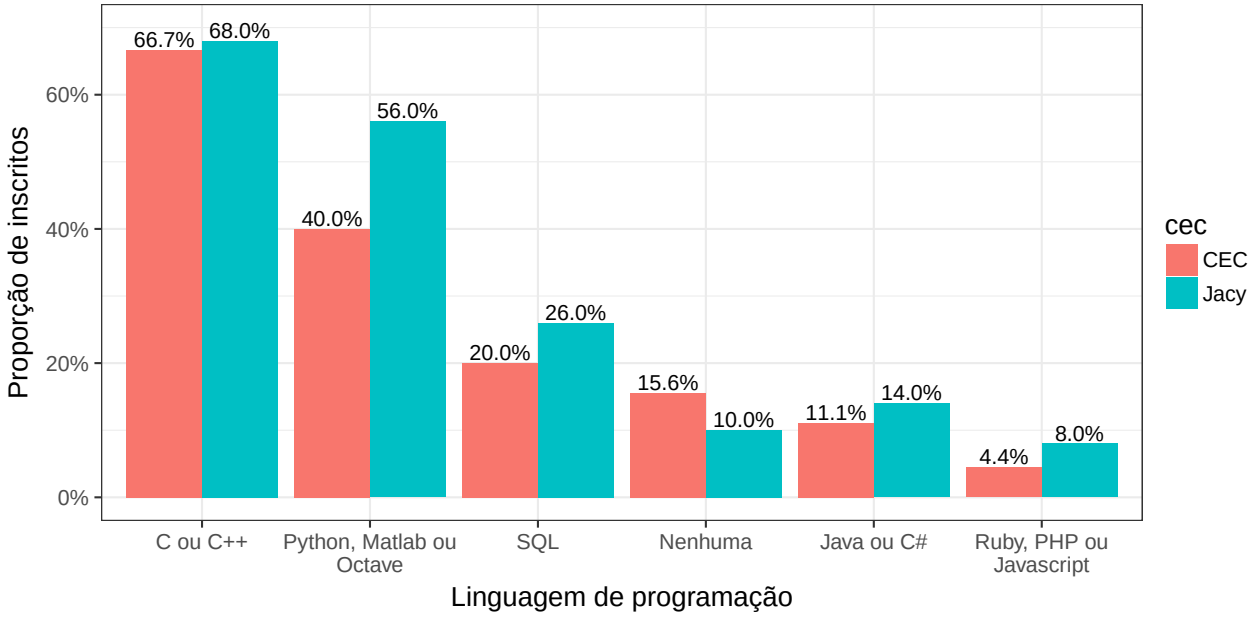
```
d_alunos %>%
  gather(questao, resposta, matches('^[13]')) %>%
  replace_na(list(resposta = 'Não.')) %>%
  count(cec, questao, resposta) %>%
  mutate(prop = n / sum(n)) %>%
  ggplot(aes(x = str_wrap(resposta, 20), fill = cec, y = prop)) +
  geom_bar(position = 'dodge', stat = 'identity') +
  facet_wrap(~questao, scales = 'free_x', ncol = 1) +
  scale_y_continuous(labels = scales::percent) +
  geom_text(aes(label = scales::percent(prop), group = cec),
            position = position_dodge(.9), vjust = -.2) +
  theme_bw(14) +
  theme(strip.background = element_blank()) +
  xlab('') +
  ylab('Proporção de inscritos')
```



Pergunta 2: sobre conhecimento em outras linguagens. Não soma 100%!

```
d_alunos %>%
  gather(questao, resposta, matches('^[2]')) %>%
  replace_na(list(resposta = 'Nenhuma')) %>%
  mutate(ling = str_split(resposta, '\\., ')) %>%
  unnest(ling) %>%
  mutate(ling = str_replace(ling, '\\.$', '')) %>%
  group_by(cec) %>%
  mutate(ntot = n_distinct(id_pessoa)) %>%
  group_by(cec, ling) %>%
  summarise(n = n_distinct(id_pessoa), ntot = first(ntot)) %>%
  mutate(prop = n / ntot) %>%
  mutate(ling = str_wrap(ling, 20) %>% fct_reorder(prop, .desc = TRUE)) %>%
  ggplot(aes(x = ling, fill = cec, y = prop)) +
  geom_bar(position = 'dodge', stat = 'identity') +
```

```
  scale_y_continuous(labels = scales::percent, limits = c(0, .7)) +
  geom_text(aes(label = scales::percent(prop), group = cec),
            position = position_dodge(.9), vjust = -.2) +
  theme_bw(14) +
  xlab('Linguagem de programação') +
  ylab('Proporção de inscritos')
```



Chapter 4

Trabalhando com vetores especiais

4.1 Pacote `lubridate` para trabalhar com datas

```
library(magrittr)
library(lubridate)
```

Originalmente, o `R` é bastante ruim para trabalhar com datas, o que causa frustração e perda de tempo nas análises. O pacote `lubridate` foi criado para simplificar ao máximo a leitura de datas e extração de informações dessas datas.

A função mais importante para leitura de dados no `lubridate` é a `ymd`. Essa função serve para ler qualquer data de uma `string` no formato `YYYY-MM-DD`. Essa função é útil pois funciona com qualquer separador entre os elementos da data e também porque temos uma função para cada formato (`mdy`, `dmy`, `dym`, `myd`, `ydm`).

Exemplo: dia-ano-mês

```
d1 <- '04/15/06'
dym(d1)
```

```
## [1] "2015-06-04"
```

Exemplo: ano-mês-dia

```
d2 <- '2015-01-02'
ymd(d2)
```

```
## [1] "2015-01-02"
```

Outras funções importantes

- `ymd_hms`: lê datas e horários, generalizando `ymd`. **Exemplo**:

```
d3 <- '07022016 10:11:47'
mdy_hms(d3)
```

```
## [1] "2016-07-02 10:11:47 UTC"
```

Observe que as classes são diferentes:

```
list(ymd(d2), mdy_hms(d3)) %>% lapply(class)
```

```
## [[1]]
## [1] "Date"
##
## [[2]]
```

```
## [1] "POSIXct" "POSIXt"
```

- `year`, `month`, `day`, `quarter`, `weekday`, `week`: extraem componentes da data.
- `years`, `months`, `days`: adicionam tempos a uma data, ajudando a criar vetores de datas. Por exemplo

```
ymd('2015-01-01') + months(0:11)
```

```
##  [1] "2015-01-01" "2015-02-01" "2015-03-01" "2015-04-01" "2015-05-01"
##  [6] "2015-06-01" "2015-07-01" "2015-08-01" "2015-09-01" "2015-10-01"
## [11] "2015-11-01" "2015-12-01"
```

- `floor_date` e `ceiling_date`: arredonda datas para uma unidade de interesse. Útil para agregar dados diários por semana, mês, trimestre etc.

Mais informações: ver aqui e aqui.

4.1.1 Exercício

(...)

4.2 Pacote `stringr` para trabalhar com textos

O R básico não tem uma sintaxe consistente para trabalhar com textos. O pacote `stringr` ajuda a realizar todas as tarefas básicas de manipulação de texto, exigindo que o usuário estude apenas uma sintaxe. O `stringr` também é construído sobre a biblioteca ICU, implementada em `C` e `C++`, apresentando resultados rápidos e confiáveis.

As regras básicas do pacote são:

- As funções de manipulação de texto começam com `str_`. Caso esqueça o nome de uma função, basta digitar `stringr::str_` e apertar `TAB` para ver quais são as opções.
- O primeiro argumento da função é sempre uma `string`.

Antes de listar as funções, precisamos estudar o básico de expressões regulares.

4.2.1 Expressões regulares

Expressão regular ou *regex* é uma sequência concisa de caracteres que representa várias strings. Entender o básico de expressões regulares é indispensável para trabalhar com textos.

Vamos estudar expressões regulares através de exemplos e com a função `str_detect()`. Essa função retorna `TRUE` se uma string atende à uma expressão regular e `FALSE` em caso contrário.

A tabela abaixo mostra a aplicação de seis `regex` a seis strings distintas.

```
library(stringr)
testes <- c('ban', 'banana', 'abandonado', 'pranab anderson', 'BANANA', 'ele levou ban')

expressoes <- list(
  'ban', # reconhece tudo que tenha "ban", mas não ignora case
  'BAN', # reconhece tudo que tenha "BAN", mas não ignora case
  regex('ban', ignore_case = TRUE), # reconhece tudo que tenha "ban", ignorando case
  'ban$', # reconhece apenas o que termina exatamente em "ban"
  '^ban', # reconhece apenas o que começa exatamente com "ban"
  'b ?an' # reconhece tudo que tenha "ban", com ou sem espaço entre o "b" e o "a"
)
```

regex	ban	banana	abandonado	pranab anderson	BANANA	ele levou ban
ban	TRUE	TRUE	TRUE	FALSE	FALSE	TRUE
BAN	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
ban	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
ban$	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
^ban	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
b ?an	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

4.2.1.1 Quantificadores

Os caracteres `+`, `*` e `{x,y}` indicam quantas vezes um padrão se repete:

- `ey+` significa `e` e depois `y` "**uma vez** ou mais". Por exemplo, reconhece `hey`, `heyy`, `a eyyy`, mas não reconhece `e`, `y` nem `yy`.
- `ey*` significa "**zero vezes** ou mais". Por exemplo, reconhece `hey`, `heyy`, `a eyyy` e `e`, mas não reconhece `y` nem `yy`.
- `ey{3}` significa "exatamente três vezes". Por exemplo, reconhece `eyyy` e `eyyyy`, mas não reconhece `eyy`.
- `ey{1,3}` significa "entre uma e três vezes".

Para aplicar um quantificador a um conjunto de caracteres, use parênteses. Por exemplo, `(ey )+` reconhece `ey ey`.

4.2.1.2 Conjuntos

Colocando caracteres dentro de `[]`, reconhecemos quaisquer caracteres desse conjunto. Alguns exemplos práticos:

- `[Cc]asa` para reconhecer "casa" em maiúsculo ou minúsculo.
- `[0-9]` para reconhecer somente números. O mesmo vale para letras `[a-z]`, `[A-Z]`, `[a-zA-Z]` etc.
- O símbolo `^` dentro do colchete significa negação. Por exemplo, `[^0-9]` significa pegar tudo o que não é número.
- O símbolo `.` fora do colchete indica "qualquer caractere", mas dentro do colchete é apenas ponto.
- Use `[[:space:]]+` para reconhecer espaços e `[[:punct:]]+` para reconhecer pontuações.

4.2.1.3 Miscelânea

- Use `abjutils::rm_accent()` para retirar os acentos de um texto.
- Use `|` para opções, por exemplo `desfavor|desprov` reconhece tanto "desfavorável" quanto "desprovido"
- `\n` pula linha, `\f` é final da página, `\t` é tab. Use `\` para transformar caracteres especiais em literais.
- `tolower()` e `toupper()` para mudar o case de uma string.

A lista de possibilidades com expressões regulares é extensa. Um bom lugar para testar o funcionamento de expressões regulares é o regex101.

4.2.2 Funções do `stringr`

- `str_detect()` retorna `TRUE` se a regex é compatível com a string e `FALSE` caso contrário
- `str_lengh()` retorna o comprimento de uma string.

```
str_length('hye')
```

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

- `str_trim()` retira espaços e quebras de linha/tabs no início ou final de string.

```
string <- '\nessa      string é muito suja       \n'
str_trim(string)
```

```
## [1] "essa      string é muito suja"
```

- `str_replace()` e `str_replace_all()` substituem um padrão (ou todos) encontrado para um outro padrão

```
string <- 'heyyy ui yy'
str_replace(string, 'y', 'x')
```

```
## [1] "hexyy ui yy"
```

```
str_replace(string, 'y+', 'x')
```

```
## [1] "hex ui yy"
```

```
str_replace_all(string, 'y', 'x')
```

```
## [1] "hexxx ui xx"
```

```
str_replace_all('string     com    muitos espaços', ' +', ' ') # tirar espaços extras
```

```
## [1] "string com muitos espaços"
```

- `str_match()` e `str_match_all()` extrai pedaços da string identificados pela regex. Caso queira extrair somente a parte identificada, use parênteses.

```
frases <- c('a roupa do rei', 'de roma', 'o rato roeu')
str_match(frases, 'roe')
```

```
##      [,1]
## [1,] NA
## [2,] NA
## [3,] "roe"
```

```
str_match_all(frases, 'ro')
```

```
## [[1]]
##      [,1]
## [1,] "ro"
##
## [[2]]
##      [,1]
## [1,] "ro"
##
## [[3]]
##      [,1]
## [1,] "ro"
```

```
str_match(frases, 'o (ro)')
```

```
##      [,1]   [,2]
## [1,] NA     NA
## [2,] NA     NA
## [3,] "o ro" "ro"
```

- `str_split()` separa uma string em várias de acordo com um separador.

```
string <- 'eu sei, usar virgulas, de forma, perfeita'

str_split(string, ', ')
```

```
## [[1]]
## [1] "eu sei"        "usar virgulas" "de forma"      "perfeita"
```

```
str_split(string, ', ', simplify = TRUE)
```

```
##      [,1]     [,2]            [,3]       [,4]
## [1,] "eu sei" "usar virgulas" "de forma" "perfeita"
```

- `str_split_fixed()` faz o mesmo que `str_split()`, mas separa apenas `n` vezes

```
str_split_fixed(string, ', ', 3)
```

```
##      [,1]     [,2]            [,3]
## [1,] "eu sei" "usar virgulas" "de forma, perfeita"
```

```
str_split_fixed(string, ', ', 4) # igual a str_split(string, simplify = TRUE)
```

```
##      [,1]     [,2]            [,3]       [,4]
## [1,] "eu sei" "usar virgulas" "de forma" "perfeita"
```

- `str_sub()` extrai uma parte da string de acordo com os índices.

```
string <- 'quero pegar só uma parte disso'
str_sub(string, 13, 14)
```

```
## [1] "só"
```

```
str_sub(string, -5, -1) # usar números negativos para voltar do final da string
```

```
## [1] "disso"
```

```
indices <- str_locate(string, 'parte')
indices
```

```
##      start end
## [1,]    20  24
```

```
str_sub(string, indices) # pode ser útil usar com str_locate.
```

```
## [1] "parte"
```

- `str_subset()` retorna somente as strings compatíveis com a regex.

```
frases <- c('a roupa do rei', 'de roma', 'o rato roeu')
str_subset(frases, 'd[eo]')
```

```
## [1] "a roupa do rei" "de roma"
```

4.2.3 Exemplo

4.2.4 Exercícios

1. Considere o seguinte texto

```
txt <- "A função mais importante para leitura de dados no `lubridate` é a `ymd`. Essa função serve para
```

Extraia todas as combinações da função `ymd`, sem repetições.

2. Considere os textos abaixo

```
txts <- c(
  'o produto é muito bom',
  'o produto não é bom',
```

```
  'o produto não é muito bom',
  'o produto não é ruim',
  'o produto não é não bom'
)
```

Crie uma regra para identificar se o texto refere-se a um feedback positivo ou negativo sobre o produto (considera não bom = ruim e vice-versa). Retorne um vetor lógico que vale `TRUE` se o feedback é positivo e `FALSE` caso contrário.

4.3 Pacote `forcats` para trabalhar com factors

Factors sempre foram uma pedra no sapato para usuários de R. Esses objetos são estranhos pois parecem textos, mas na verdade são inteiros.

```
x <- factor(c('a', 'b', 'c'))
x
```

```
## [1] a b c
## Levels: a b c
```

```
typeof(x)
```

```
## [1] "integer"
```

Assim, eles podem levar a erros do tipo:

```
x <- factor(c('6', '5', '4'))
as.numeric(x)
```

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

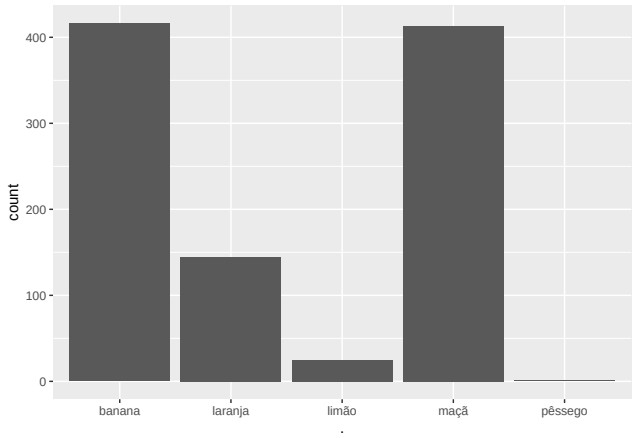
O problema é que, historicamente, esses objetos eram utilizados por diversos objetos do R. Em particular, o `data.frame` até hoje utiliza fatores como padrão. Felizmente, o `tidyverse` nos livra desse mal e permite que utilizemos fatores somente quando eles são realmente úteis.

Mas quando fatores são úteis? A resposta para essa pergunta vem da própria estatística: temos dois tipos de variáveis categóricas existentes, a nominal e a ordinal. Uma variável nominal pode ser completamente representada por um vetor de strings. Mas isso não vale para variáveis ordinais, pois um vetor de strings só pode ser ordenado alfabeticamente, o que em muitos casos não é suficiente. O `factor` permite que associemos um inteiro para cada valor de uma string, nos dando a possibilidade de ordená-las da forma que quisermos.

O pacote `forcats` (`for` - para, `cats` - categóricas, não gatos) serve justamente para reordenar fatores de diversas formas. Isso é especialmente útil para visualização, pois muitas vezes queremos ordenar coisas de acordo com alguma regra.

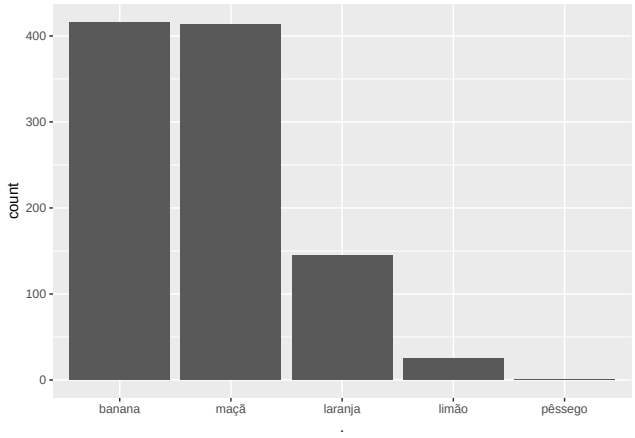
Por exemplo, considere o seguinte gráfico de barras (veremos sobre o ggplot na próxima vez).

```
set.seed(123)
labs <- c('banana', 'maçã', 'laranja', 'limão', 'pêssego')[rbinom(1000, 4, .2) + 1]
labs %>% ggplot2::qplot()
```



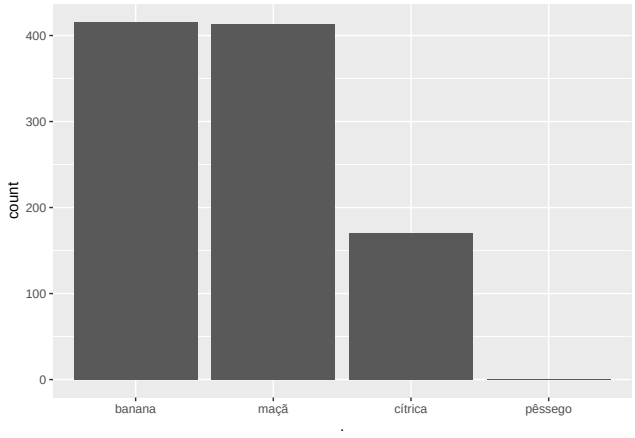
Note que o eixo `x` está ordenado alfabeticamente. Como a ordem das frutas não importa (pois é uma variável nominal), faz mais sentido ordenarmos as barras de acordo com a quantidade de frutas. Isso é feito com a função `fct_infreq`:

```
library(forcats)
labs %>% fct_infreq() %>% ggplot2::qplot()
```



Outra importante função do `forcats` possibilita agrupar fatores de forma eficiente:

```
labs %>%
  fct_collapse(cítrica = c('laranja', 'limão')) %>%
  fct_infreq() %>%
  ggplot2::qplot()
```

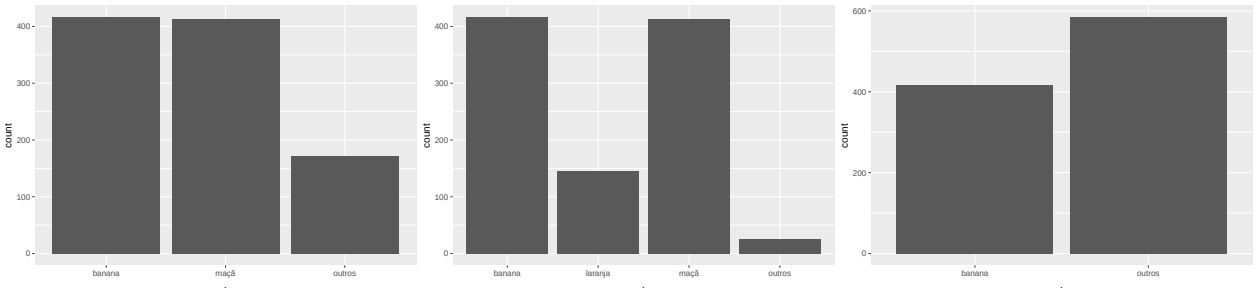


Outra forma comum de agrupar fatores é agrupar em "outros" as categorias que aparecem poucas vezes na base de dados. Para isso, utilizamos `fct_lump`:

```
labs %>% fct_count()
```

```
## # A tibble: 5 × 2
##         f     n
##    <fctr> <int>
## 1  banana   416
## 2 laranja   145
## 3   limão    25
## 4    maçã   413
## 5 pêssego     1
```

```
# agrupa todas desde que "outro" continue a menor categoria
labs %>% fct_lump(other_level = 'outros') %>% ggplot2::qplot()
# 10% menores
labs %>% fct_lump(prop = .10, other_level = 'outros') %>% ggplot2::qplot()
# mantém os n maiores
labs %>% fct_lump(n = 1, other_level = 'outros') %>% ggplot2::qplot()
```



Finalmente, uma função útil para produção de gráficos é a `fct_reorder`, que permite utilizar uma variável auxiliar (possivelmente fazendo sumarizações) para ordenar o fator.

4.3.1 Exercício

A função `fct_reorder` foi utilizada na análise dos inscritos. Descubra como ela foi utilizada e qual seu efeito no gráfico.

Chapter 5

Visualização de dados

5.1 Com ggplot2

O ggplot2 é um pacote do R voltado para a criação de gráficos estatísticos. Ele é baseado na Gramática dos Gráficos (*grammar of graphics*, em inglês), criado por Leland Wilkinson, que é uma resposta para a pergunta: o que é um gráfico estatístico? Resumidamente, a gramática diz que um gráfico estatístico é um mapeamento dos dados a partir de atributos estéticos (cores, formas, tamanho) em formas geométricas (pontos, linhas, barras).

Para mais informações sobre a Gramática dos Gráficos, você pode consultar o livro *The Grammar of graphics*, escrito pelo Leland Wilkinson, ou o livro ggplot2: elegant graphics for data analysis, do Hadley Wickham. Um pdf do livro também está disponível.

Parei aqui. Daqui pra baixo está desatualizado!

5.1.1 Construindo gráficos

A seguir, vamos discutir os aspcetos básicos para a construção de gráficos com o pacote ggplot2. Para isso, utilizaremos o banco de dados contido no objeto mtcars. Para visualizar as primeiras linhas deste banco, utilize o comando:

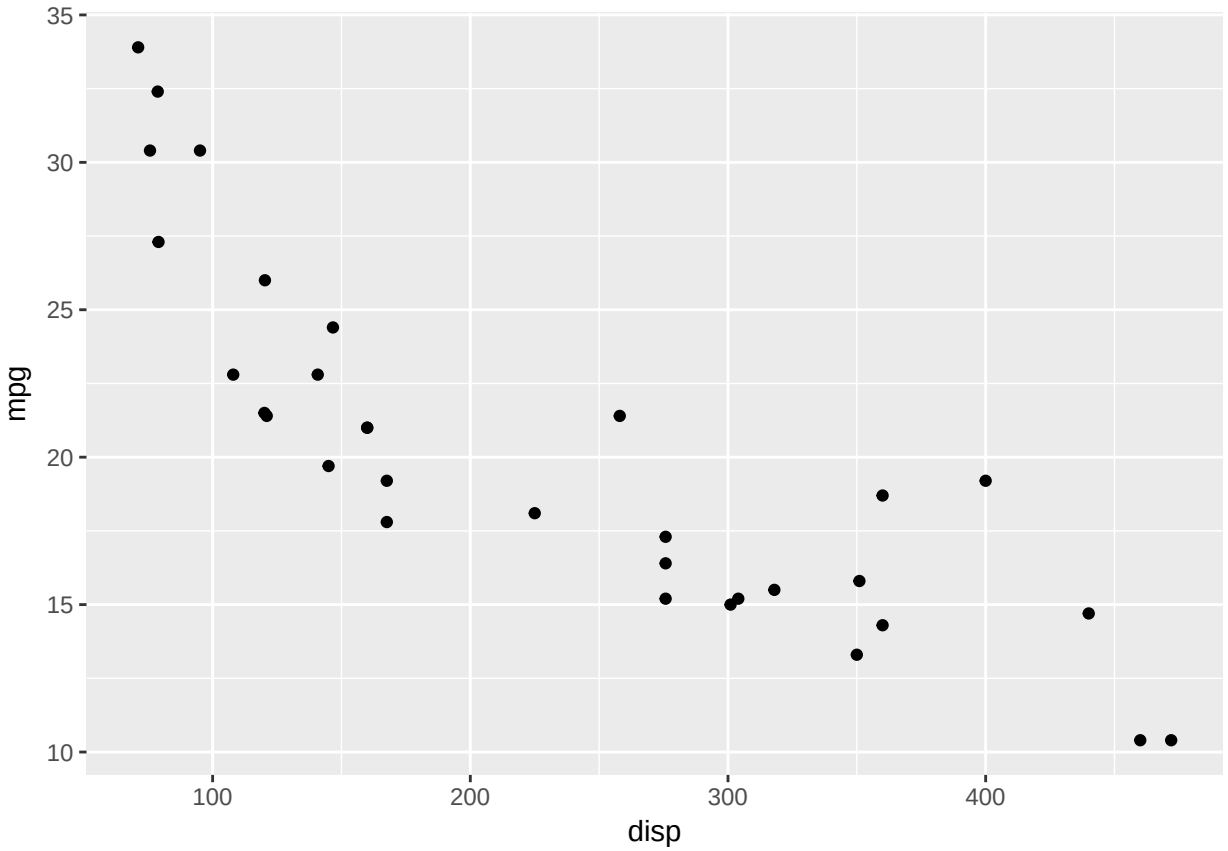
```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

5.1.2 As camadas de um gráfico

No ggplot2, os gráficos são construídos camada por camada (ou, *layers*, em inglês), sendo que a primeira delas é dada pela função ggplot (não tem o "2"). Cada camada representa um tipo de mapeamento ou personalização do gráfico. O código abaixo é um exemplo de um gráfico bem simples, construído a partir das duas principais camadas.

```
library(ggplot2)
ggplot(data = mtcars, aes(x = disp, y = mpg)) +
  geom_point()
```



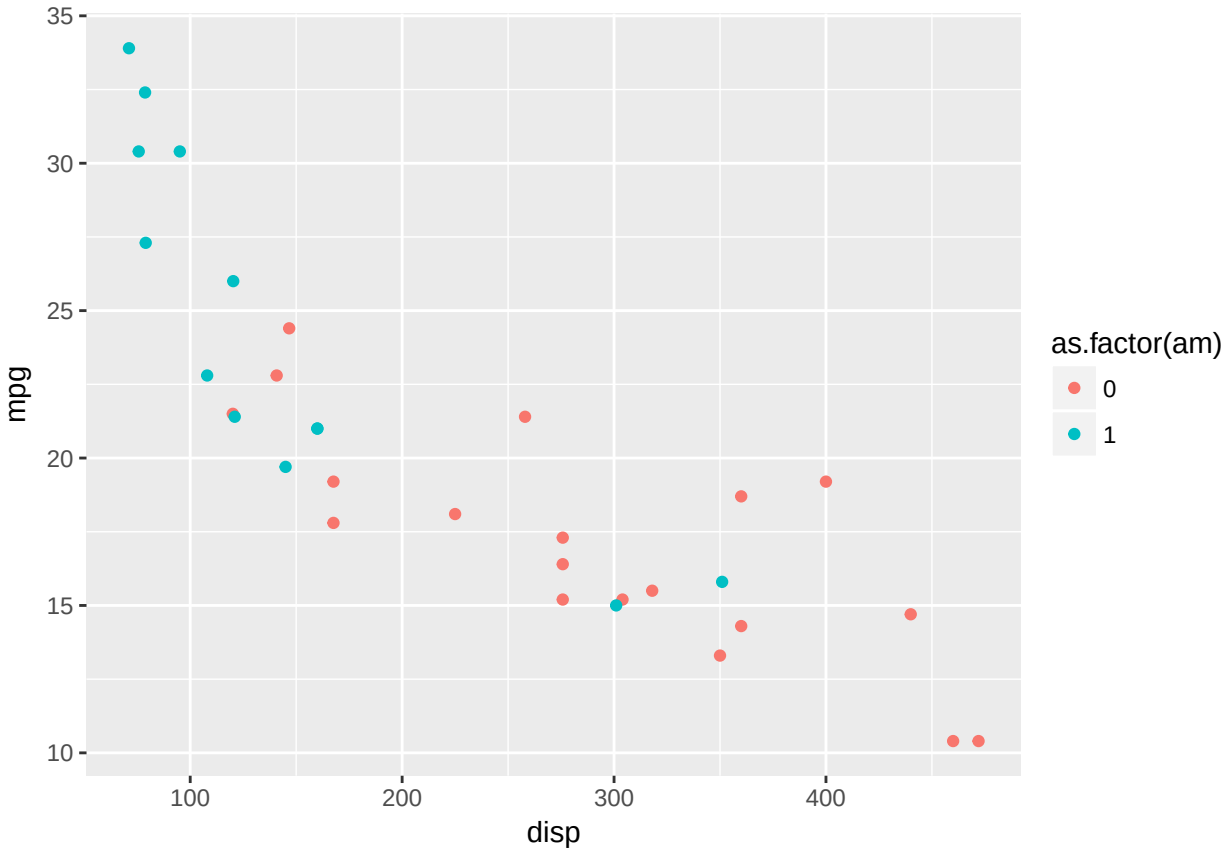
Observe que o primeiro argumento da função `ggplot` é um data frame. A função `aes()` descreve como as variáveis são mapeadas em aspectos visuais de formas geométricas definidas pelos *geoms*. Aqui, essas formas geométricas são pontos, selecionados pela função `geom_point()`, gerando, assim, um gráfico de dispersão. A combinação dessas duas camadas define o tipo de gráfico que você deseja construir.

5.1.2.1 Aesthetics

A primeira camada de um gráfico deve indicar a relação entre os dados e cada aspecto visual do gráfico, como qual variável será representada no eixo x, qual será representada no eixo y, a cor e o tamanho dos componentes geométricos etc. Os aspectos que podem ou devem ser mapeados depende do tipo de gráfico que você deseja fazer.

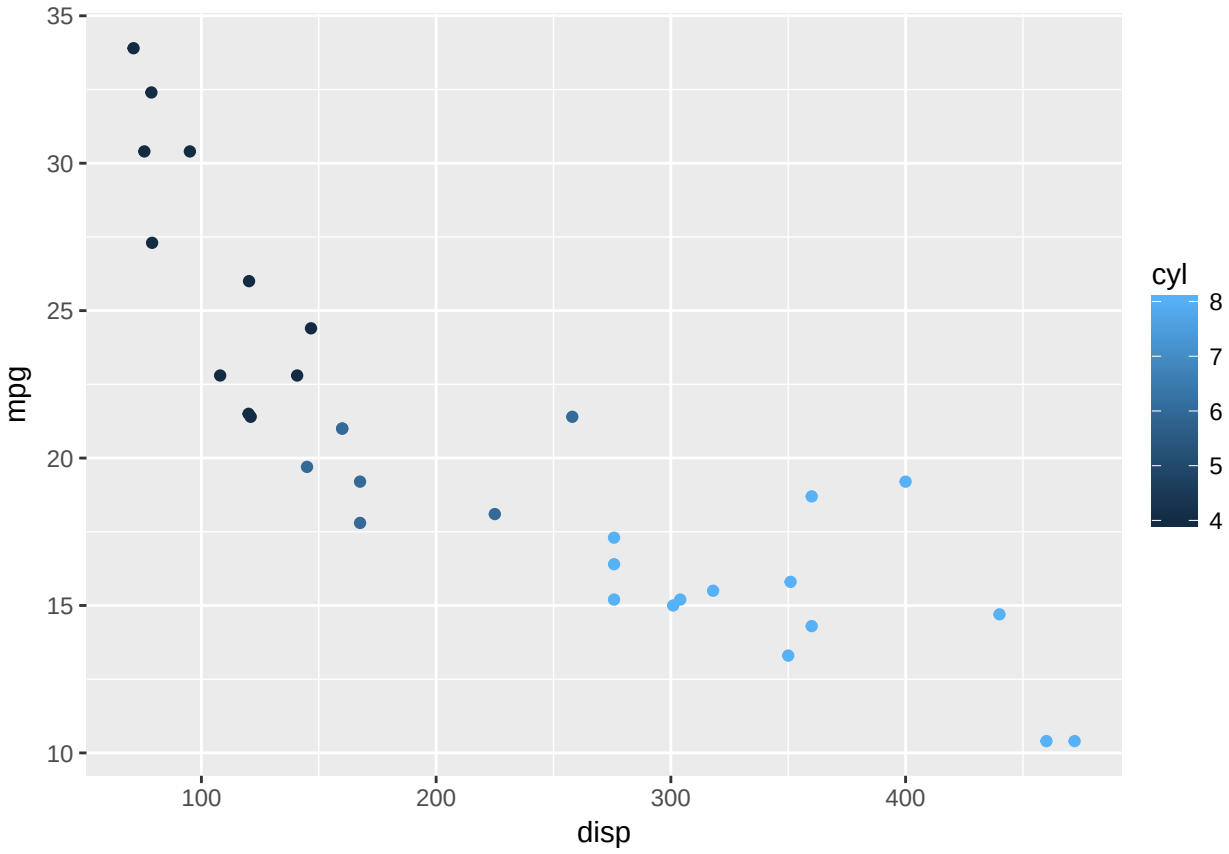
No exemplo acima, atribuímos aspectos de posição: ao eixo y mapeamos a variável `mpg` (milhas por galão) e ao eixo x a variável `disp` (cilindradas). Outro aspecto que pode ser mapeado nesse gráfico é a cor dos pontos

```
ggplot(data = mtcars, aes(x = disp, y = mpg, colour = as.factor(am))) +
  geom_point()
```



Agora, a variável `am` (tipo de transmissão) foi mapeada à cor dos pontos, sendo que pontos vermelhos correspondem à transmissão automática (valor 0) e pontos azuis à transmissão manual (valor 1). Observe que inserimos a variável `am` como um fator, pois temos interesse apenas nos valores "0" e "1". No entanto, tambem podemos mapear uma variável contínua à cor dos pontos:

```
ggplot(mtcars, aes(x = disp, y = mpg, colour = cyl)) +
  geom_point()
```

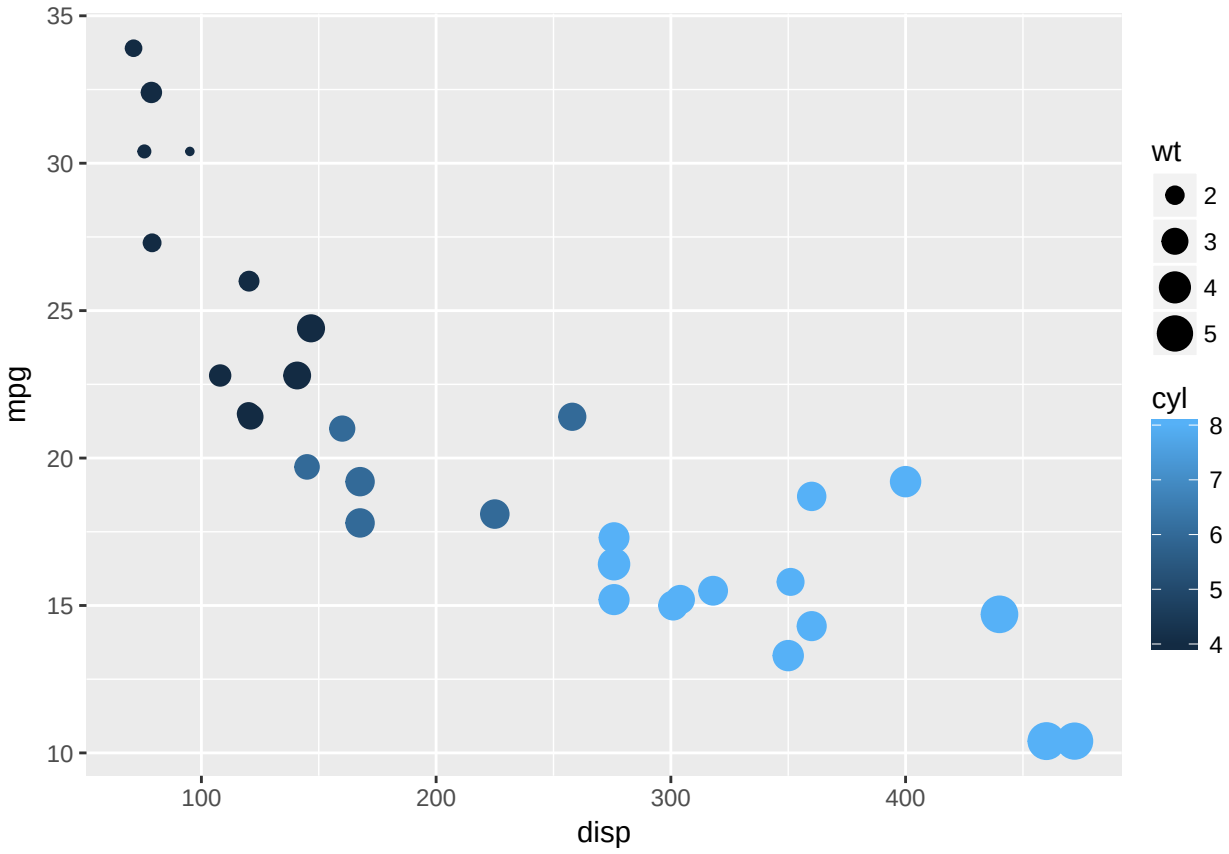


Aqui, o número de cilindros, `cyl`, é representado pela tonalidade da cor azul.

Nota: por *default*, a legenda é insirida no gráfico automaticamente.

Também podemos mapear o tamanho dos pontos à uma variável de interesse:

```
ggplot(mtcars, aes(x = disp, y = mpg, colour = cyl, size = wt)) +
  geom_point()
```



Exercício: pesquisar mais aspectos que podem ser alterados no gráfico de dispersão.

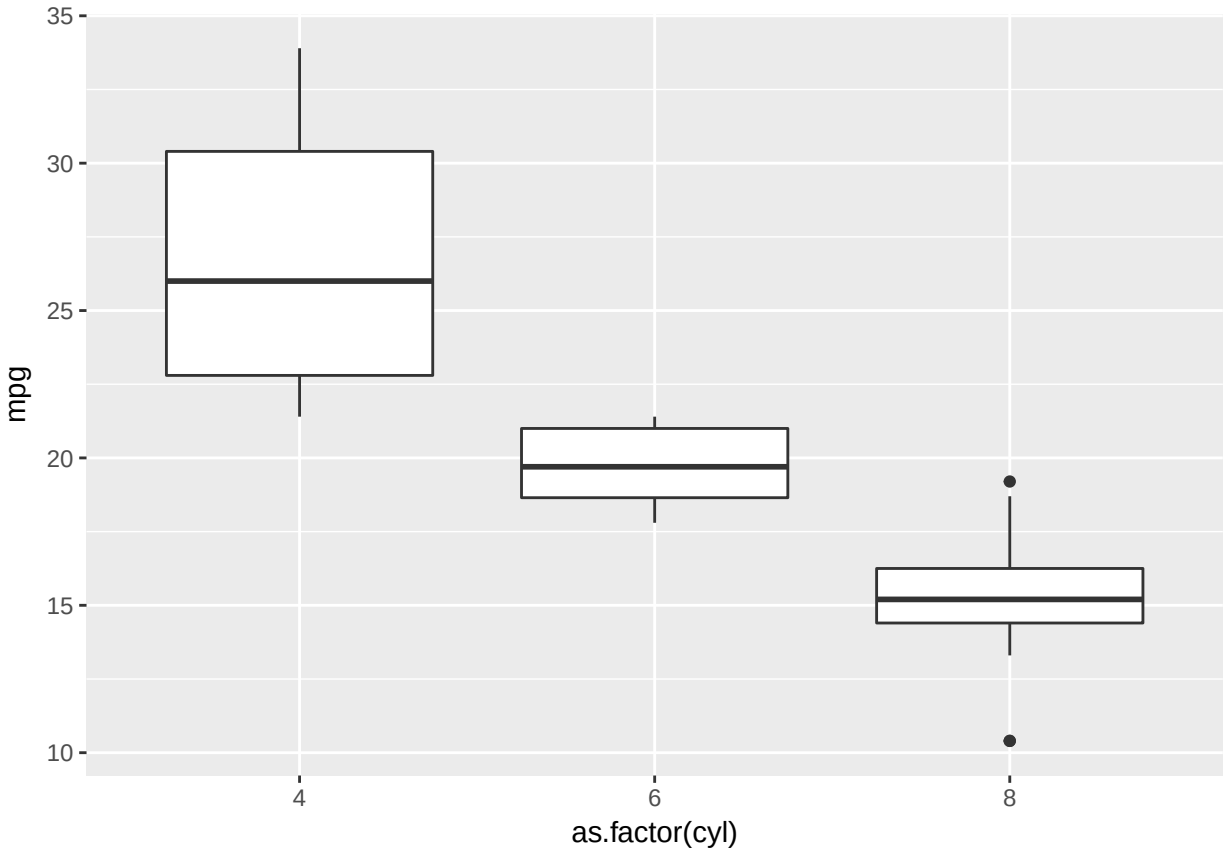
5.1.2.2 Geoms

Os *geoms* definem qual forma geométrica será utilizada para a visualização dos dados no gráfico. Como já vimos, 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_boxplot: para boxplots
- geom_histogram: para histogramas
- geom_density: para densidades
- geom_area: para áreas
- geom_bar: para barras

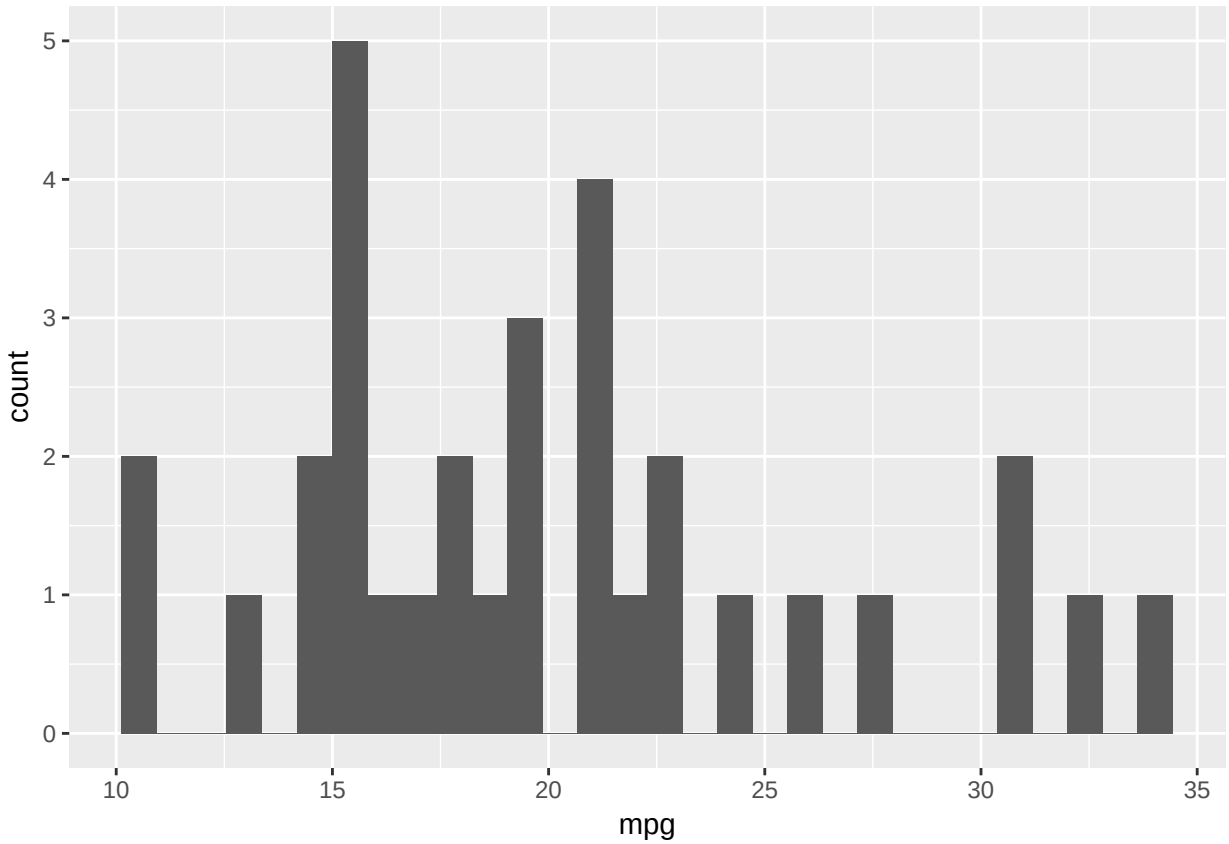
Veja a seguir como é fácil gerar diversos gráficos diferentes utilizando a mesma estrutura do gráfico de dispersão acima:

```
ggplot(mtcars, aes(x = as.factor(cyl), y = mpg)) +
  geom_boxplot()
```

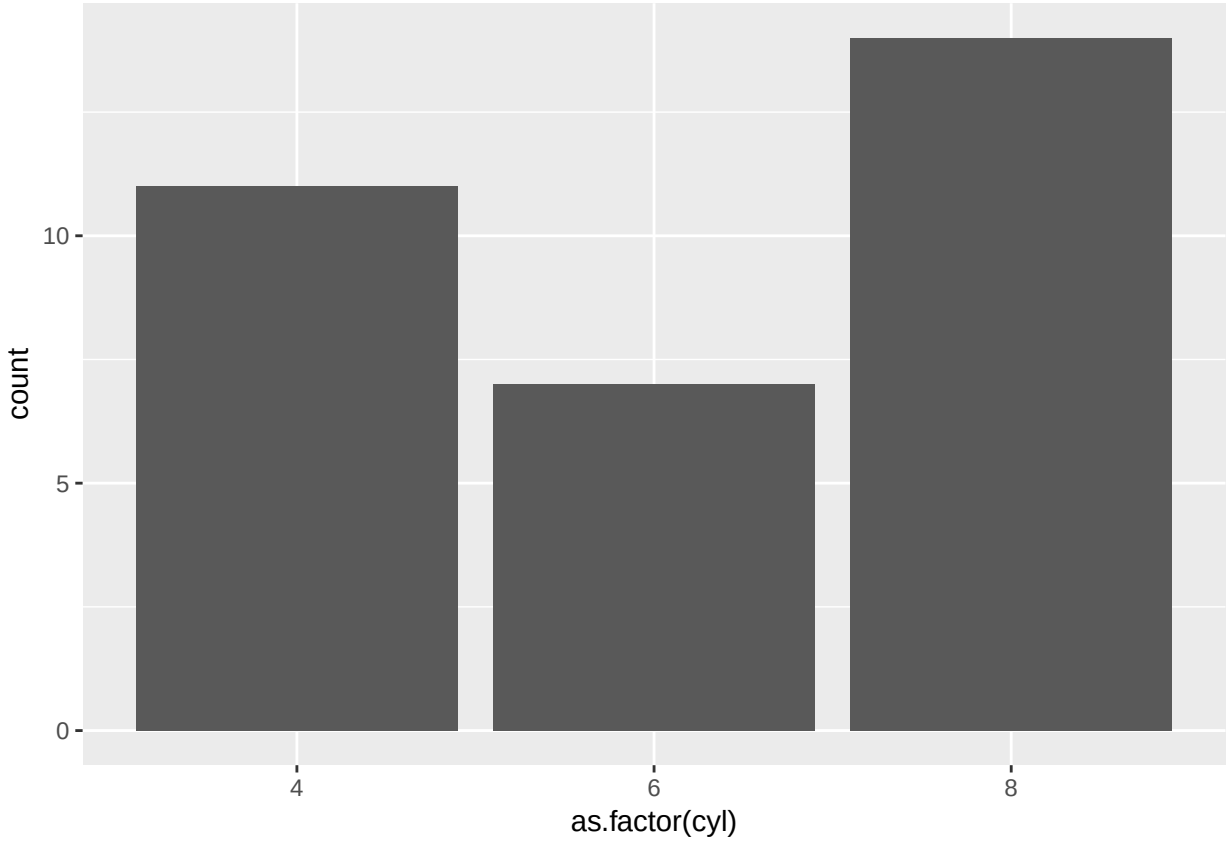



```
ggplot(mtcars, aes(x = mpg)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ggplot(mtcars, aes(x = as.factor(cyl))) +
  geom_bar()
```

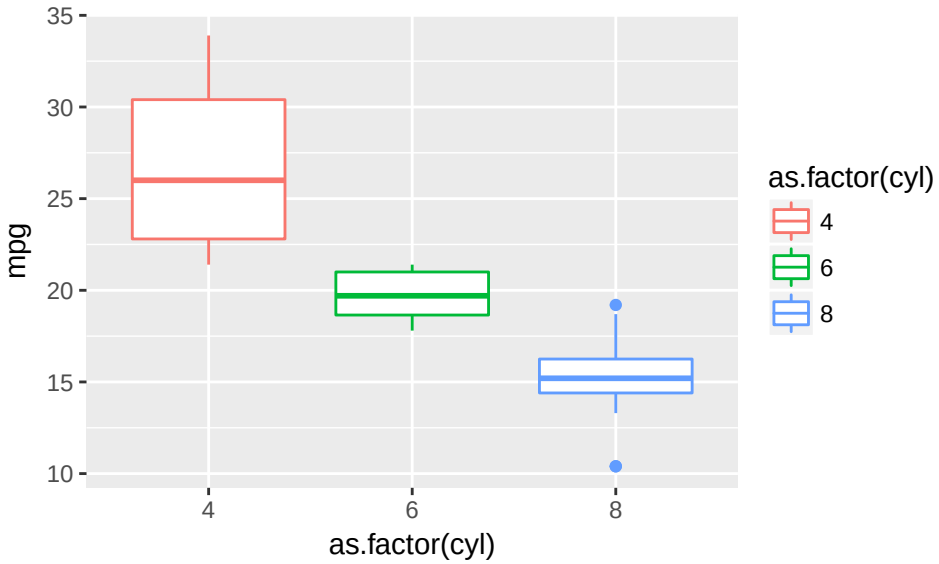


Para fazer um boxplot para cada grupo, precisamos passar para o aspecto x do gráfico uma variável do tipo fator. ### Personalizando os gráficos

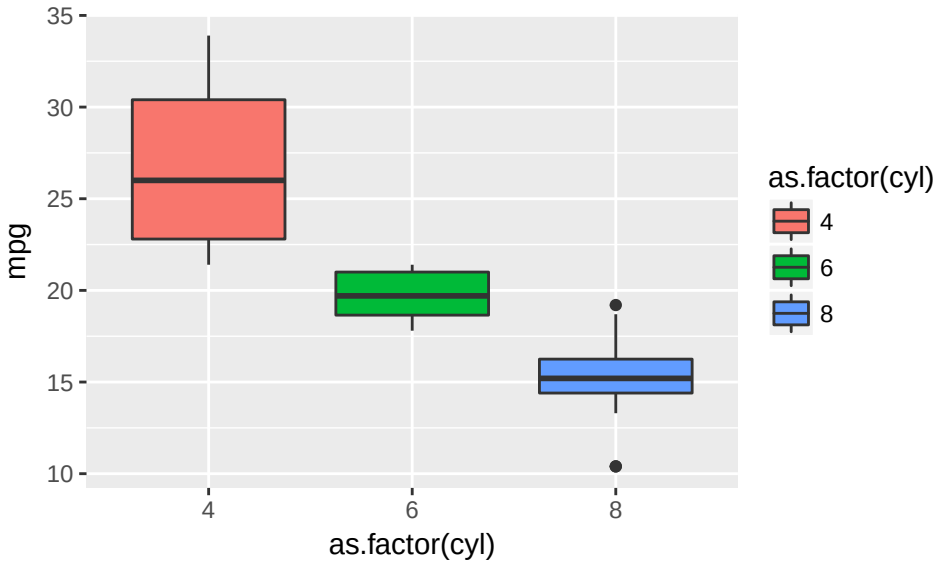
5.1.2.3 Cores

O aspecto colour do boxplot, muda a cor do contorno. Para mudar o preenchimento, basta usar o `fill`.

```
ggplot(mtcars, aes(x = as.factor(cyl), y = mpg, colour = as.factor(cyl))) +
  geom_boxplot()
```

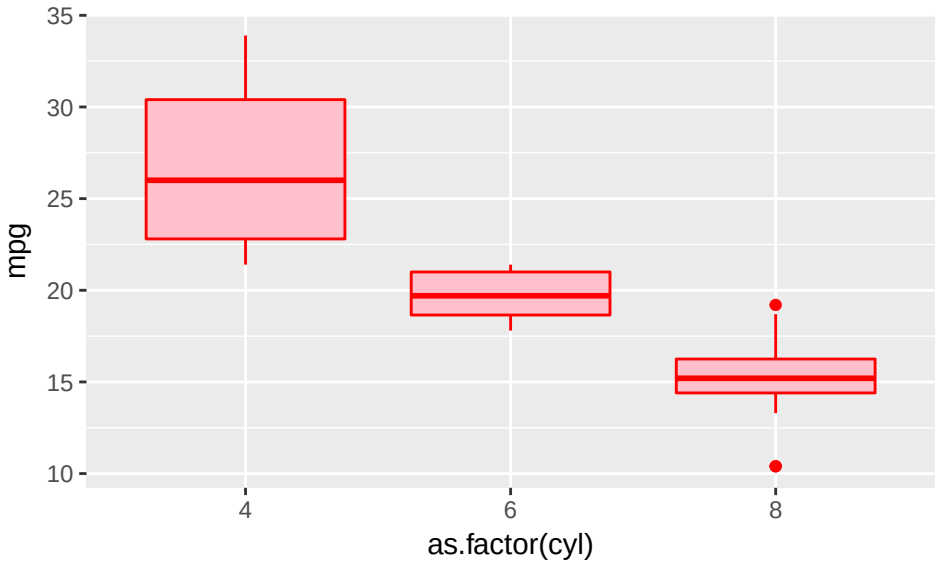


```
ggplot(mtcars, aes(x = as.factor(cyl), y = mpg, fill = as.factor(cyl))) + geom_boxplot()
```



Você pode também mudar a cor dos objetos sem mapeá-la a uma variável. Para isso, observe que os aspectos `colour` e `fill` são especificados fora do `aes()`.

```
ggplot(mtcars, aes(x = as.factor(cyl), y = mpg)) +
  geom_boxplot(color = "red", fill = "pink")
```

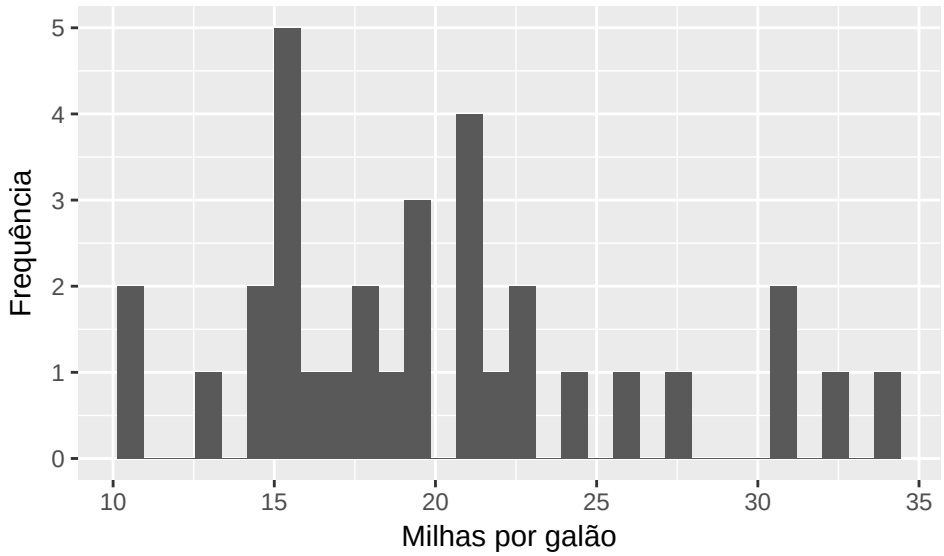


5.1.2.4 Eixos

Para alterar os labels dos eixos acrescentamos as funções `xlab()` ou `ylab()`.

```
ggplot(mtcars, aes(x = mpg)) +
  geom_histogram() +
  xlab("Milhas por galão") +
  ylab("Frequência")
```

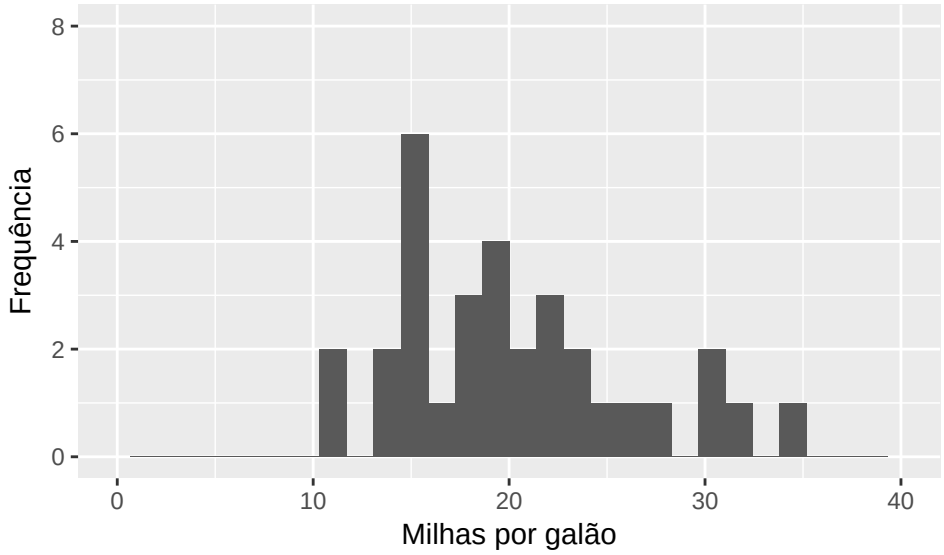
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Para alterar os limites dos gráficos usamos as funções `xlim()` e `ylim()`.

```
ggplot(mtcars, aes(x = mpg)) +
  geom_histogram() +
  xlab("Milhas por galão") +
  ylab("Frequência") +
  xlim(c(0, 40)) +
  ylim(c(0,8))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



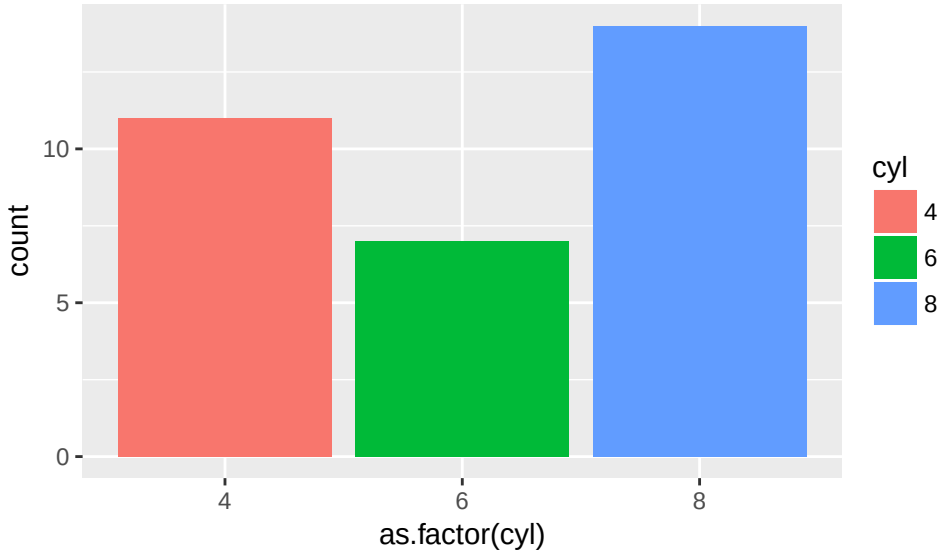
5.1.2.5 Legendas

A legenda de um gráfico pode ser facilmente personalizada.

Para trocar o *label* da leganda:

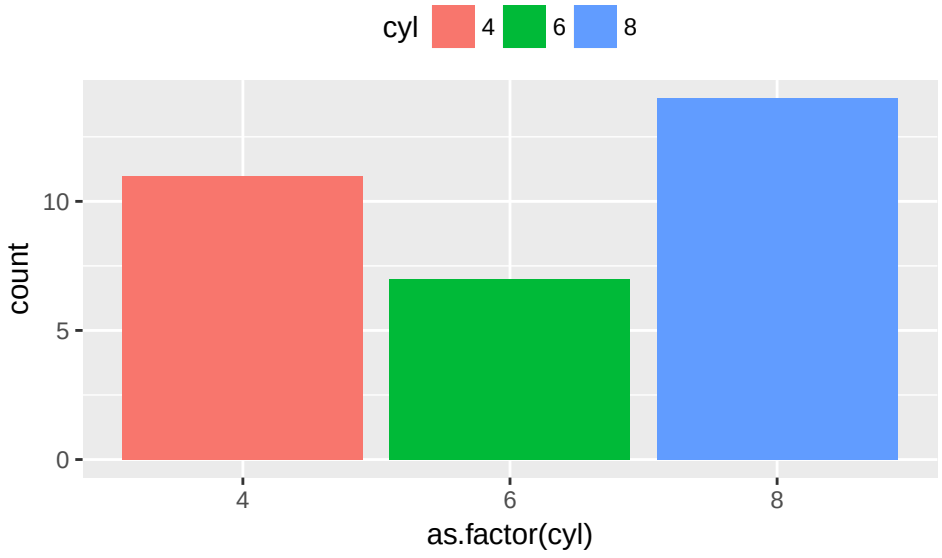
```
ggplot(mtcars, aes(x = as.factor(cyl), fill = as.factor(cyl))) +
  geom_bar() +
```

```
  labs(fill = "cyl")
```



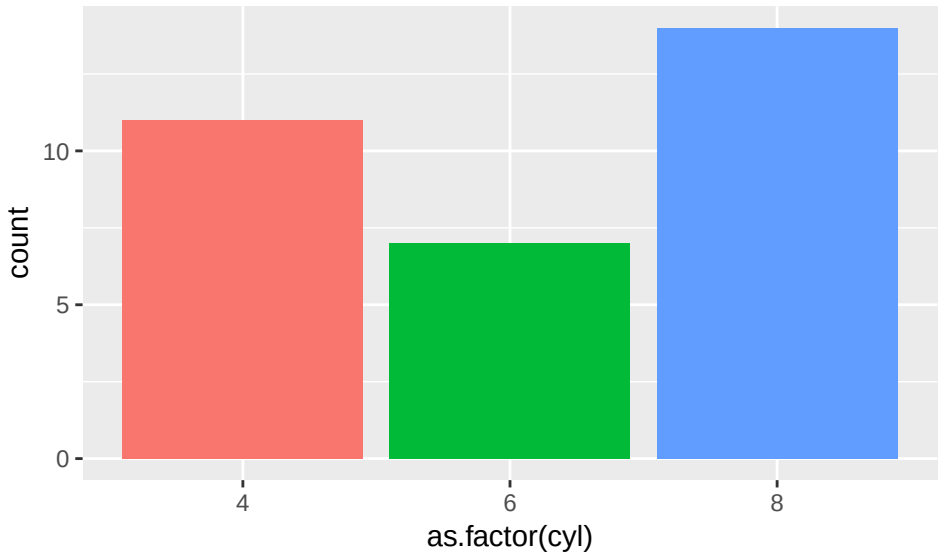
Para trocar a posição da legenda:

```
ggplot(mtcars, aes(x = as.factor(cyl), fill = as.factor(cyl))) +
  geom_bar() +
  labs(fill = "cyl") +
  theme(legend.position="top")
```



Para retirar a legenda:

```
ggplot(mtcars, aes(x = as.factor(cyl), fill = as.factor(cyl))) +
  geom_bar() +
  guides(fill=FALSE)
```

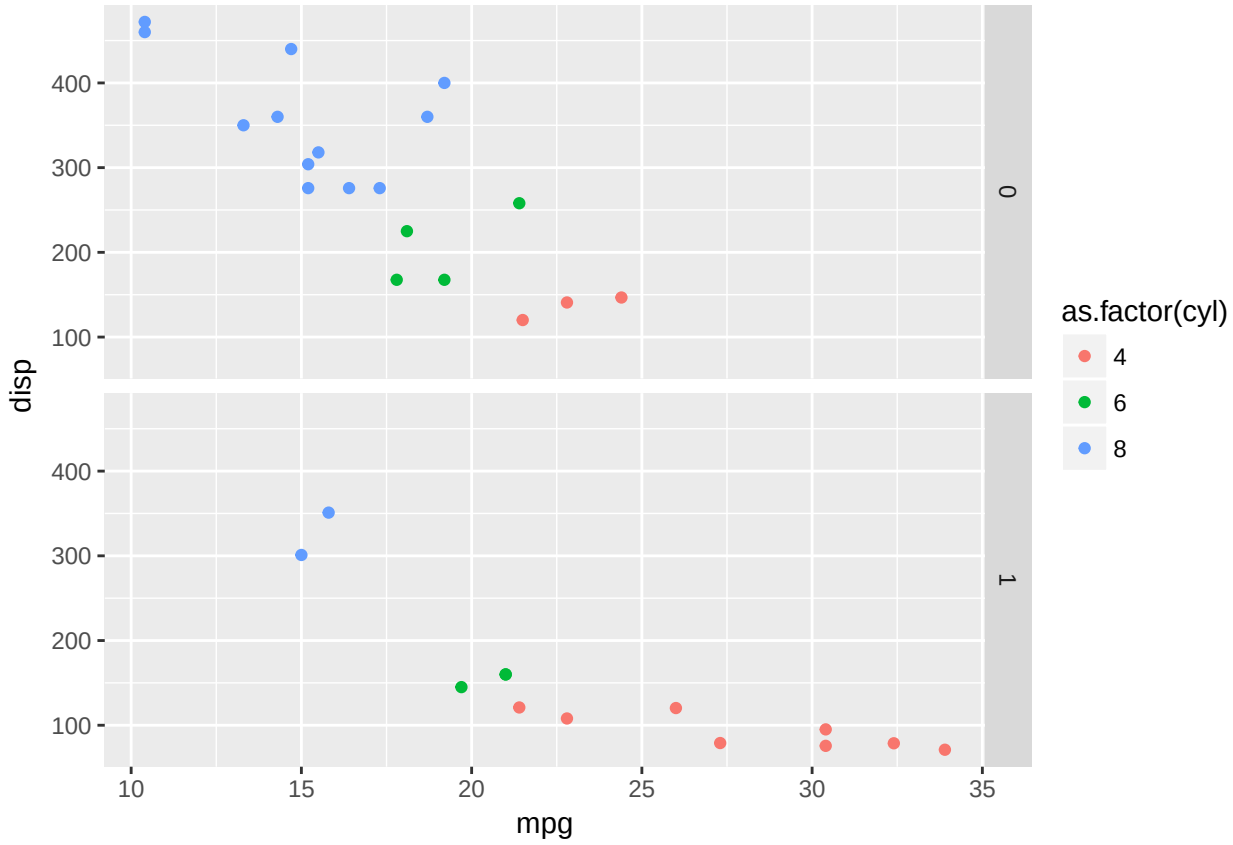


Veja mais opções de personalização aqui!

5.1.2.6 Facets

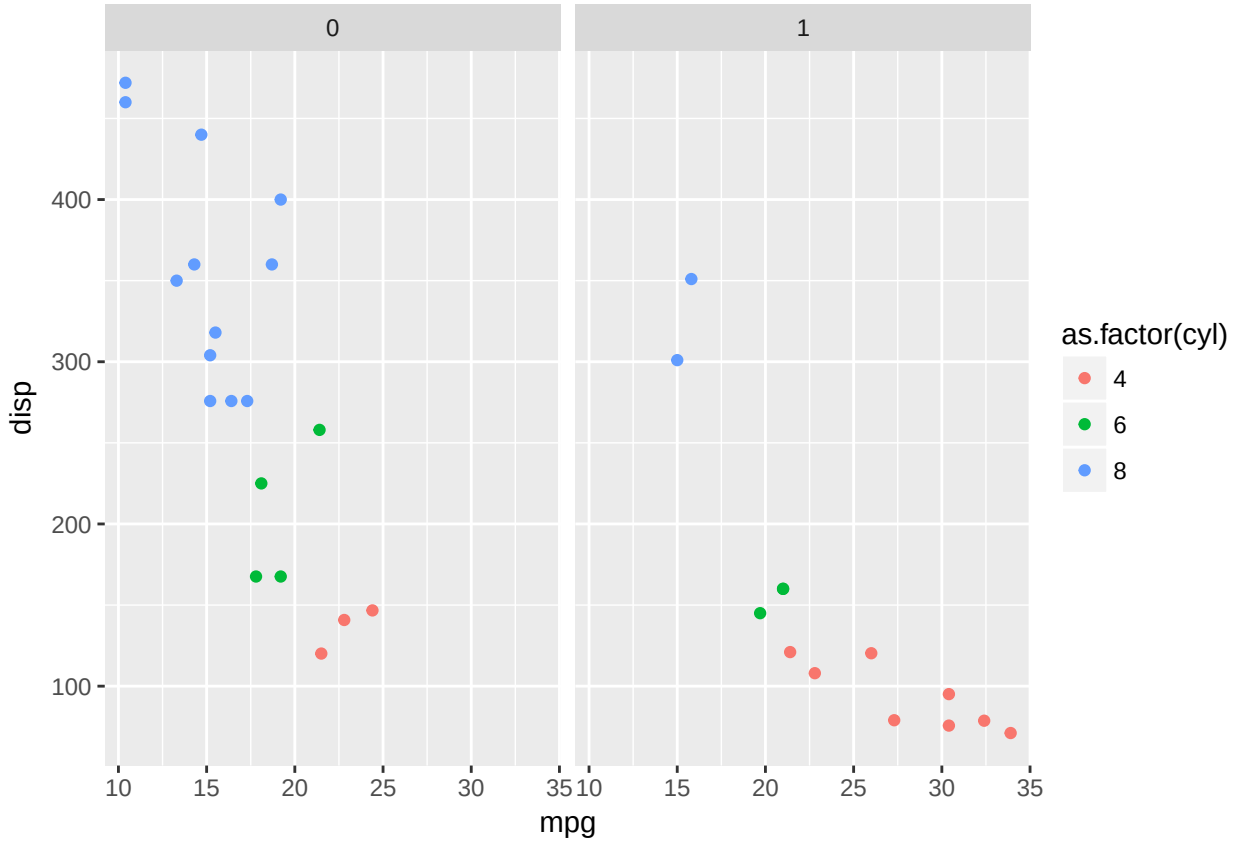
Outra funcionalidade muito importante do ggplot é o uso de *facets*.

```
ggplot(mtcars, aes(x = mpg, y = disp, colour = as.factor(cyl))) +
  geom_point() +
  facet_grid(am~.)
```



Podemos colocar os graficos lado a lado também:

```
ggplot(mtcars, aes(x = mpg, y = disp, colour = as.factor(cyl))) +
  geom_point() +
  facet_grid(.~am)
```



Chapter 6

Transformação de dados

A transformação de dados é uma tarefa usualmente dolorosa e demorada, podendo tomar a maior parte do tempo da análise. No entanto, como nosso interesse geralmente é na modelagem dos dados, essa tarefa é muitas vezes negligenciada.

> "(...) The fact that data science exists as a field is a colossal failure of statistics. To me, [what I do] is what statistics is all about. It is gaining insight from data using modelling and visualization. Data munging and manipulation is hard and statistics has just said that's not our domain."
>
> Hadley Wickham

6.1 Pacotes `dplyr` e `tidyr`

O `dplyr` é um dos pacotes mais úteis para realizar manipulação de dados, e procura aliar simplicidade e eficiência de uma forma bastante elegante. Os scripts em `R` que fazem uso inteligente dos verbos `dplyr` e as facilidades do operador *pipe* tendem a ficar mais legíveis e organizados, sem perder velocidade de execução.

Por ser um pacote que se propõe a realizar um dos trabalhos mais árduos da análise estatística, e por atingir esse objetivo de forma elegante, eficaz e eficiente, o `dplyr` pode ser considerado como uma revolução no `R`.

6.1.1 Trabalhando com `tibbles`

A `tibble` nada mais é do que um `data.frame`, mas com um método de impressão mais adequado. Outras diferenças podem ser estudadas neste link.

Vamos assumir que temos a seguinte base de dados:

```
d_cjsg
```

```
## # A tibble: 184,250 × 14
##                                                    arq    id cd_acordao
##                                                  <chr> <chr>      <chr>
## 1  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     1    9381267
## 2  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     2    8671548
## 3  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     3    8634338
## 4  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     4    8536605
## 5  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     5    8509346
## 6  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     6    8490681
## 7  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     7    8466583
## 8  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     8    8449087
```

```
## 9  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     9    8429536
## 10 /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html    10    8331899
## # ... with 184,240 more rows, and 11 more variables: n_processo <chr>,
## #   comarca <chr>, data_julgamento <chr>, data_registro <chr>,
## #   ementa <chr>, orgao_julgador <chr>, outros_numeros <chr>,
## #   relatora <chr>, classe_assunto <chr>, txt_ementa <chr>, result <chr>
```

6.1.2 As cinco funções principais do `dplyr`

- `filter`
- `mutate`
- `select`
- `arrange`
- `summarise`

6.1.3 Características

- O *input* é sempre uma `tibble`, e o *output* é sempre um `tibble`.
- No primeiro argumento colocamos o `tibble`, e nos outros argumentos colocamo o que queremos fazer.
- A utilização é facilitada com o emprego do operador `%>%`

6.1.4 Vantagens

- Utiliza `C` e `C++` por trás da maioria das funções, o que geralmente torna o código mais eficiente.
- Pode trabalhar com diferentes fontes de dados, como bases relacionais (SQL) e `data.table`.

6.1.5 `select`

- Utilizar `starts_with(x)`, `contains(x)`, `matches(x)`, `one_of(x)`, etc.
- Possível colocar nomes, índices, e intervalos de variáveis com `:`.

```
d_cjsg %>%
  select(id, cd_acordao, comarca, relator = relatora)
```

```
## # A tibble: 184,250 × 4
##       id cd_acordao             comarca                  relator
##    <chr>      <chr>               <chr>                    <chr>
## 1      1    9381267           São Paulo           Encinas Manfré
## 2      2    8671548           Guarulhos           Edison Brandão
## 3      3    8634338              Osasco             Ivan Sartori
## 4      4    8536605          Adamantina          Walter da Silva
## 5      5    8509346             Limeira Guilherme de Souza Nucci
## 6      6    8490681           São Paulo         Roberto Solimene
## 7      7    8466583              Suzano             Poças Leitão
## 8      8    8449087 Presidente Prudente             Ivan Sartori
## 9      9    8429536           São Paulo           Willian Campos
## 10    10    8331899           São Paulo     Luis Soares de Mello
## # ... with 184,240 more rows
```

```
d_cjsg %>%
  select(cd_acordao:comarca, classe_assunto)
```

```
## # A tibble: 184,250 × 4
##    cd_acordao                n_processo             comarca
##         <chr>                     <chr>               <chr>
## 1     9381267 0081568-68.2012.8.26.0050           São Paulo
## 2     8671548 2132739-15.2014.8.26.0000           Guarulhos
## 3     8634338 2204759-04.2014.8.26.0000              Osasco
## 4     8536605 2179832-71.2014.8.26.0000          Adamantina
## 5     8509346 2055376-49.2014.8.26.0000             Limeira
## 6     8490681 0014476-05.2014.8.26.0050           São Paulo
## 7     8466583 0067859-48.2014.8.26.0000              Suzano
## 8     8449087 0054802-60.2014.8.26.0000 Presidente Prudente
## 9     8429536 2128810-71.2014.8.26.0000           São Paulo
## 10    8331899 2002343-13.2015.8.26.0000           São Paulo
## # ... with 184,240 more rows, and 1 more variables: classe_assunto <chr>
```

```
d_cjsg %>%
  select(n_processo, starts_with('data_'))
```

```
## # A tibble: 184,250 × 3
##                   n_processo data_julgamento data_registro
##                        <chr>           <chr>         <chr>
## 1  0081568-68.2012.8.26.0050      29/01/2015    27/04/2016
## 2  2132739-15.2014.8.26.0000      27/01/2015    04/08/2015
## 3  2204759-04.2014.8.26.0000      27/01/2015    22/07/2015
## 4  2179832-71.2014.8.26.0000      22/01/2015    16/06/2015
## 5  2055376-49.2014.8.26.0000      27/01/2015    02/06/2015
## 6  0014476-05.2014.8.26.0050      29/01/2015    27/05/2015
## 7  0067859-48.2014.8.26.0000      29/01/2015    19/05/2015
## 8  0054802-60.2014.8.26.0000      27/01/2015    13/05/2015
## 9  2128810-71.2014.8.26.0000      29/01/2015    06/05/2015
## 10 2002343-13.2015.8.26.0000      27/01/2015    29/03/2015
## # ... with 184,240 more rows
```

6.1.6 `filter`

- Parecido com `subset`.
- Condições separadas por vírgulas é o mesmo que separar por `&`.

```
d_cjsg %>%
  select(id, cd_acordao, comarca, relator = relatora) %>%
  filter(comarca == 'São Paulo')
```

```
## # A tibble: 41,488 × 4
##       id cd_acordao   comarca               relator
##    <chr>      <chr>     <chr>                 <chr>
## 1      1    9381267 São Paulo        Encinas Manfré
## 2      6    8490681 São Paulo      Roberto Solimene
## 3      9    8429536 São Paulo        Willian Campos
## 4     10    8331899 São Paulo  Luis Soares de Mello
## 5     12    8317638 São Paulo Sérgio Mazina Martins
## 6     13    8314983 São Paulo         Péricles Piza
## 7     14    8314866 São Paulo         Péricles Piza
## 8     17    8293299 São Paulo        Ivo de Almeida
## 9     19    8284691 São Paulo        Edison Brandão
## 10    21    8274010 São Paulo  Cesar Mecchi Morales
```

```
## # ... with 41,478 more rows
```

```
library(lubridate)
d_cjsg %>%
  select(id, cd_acordao, comarca, data_julgamento, relator = relatora) %>%
  filter(comarca %in% c('Campinas', 'Sorocaba') &
         (day(dmy(data_julgamento)) >= 29 | day(dmy(data_julgamento)) < 25))
```

```
## # A tibble: 6,262 × 5
##       id cd_acordao  comarca data_julgamento               relator
##    <chr>      <chr>    <chr>           <chr>                 <chr>
## 1     33    8221266 Sorocaba      29/01/2015 Alcides Malossi Junior
## 2    136    8211543 Campinas      29/01/2015        Walter da Silva
## 3    174    8211235 Sorocaba      22/01/2015        Walter da Silva
## 4    188    8211081 Sorocaba      29/01/2015        Walter da Silva
## 5    190    8211053 Campinas      29/01/2015        Walter da Silva
## 6    229    8206214 Sorocaba      29/01/2015      Ricardo Tucunduva
## 7    312    8194316 Sorocaba      29/01/2015           Poças Leitão
## 8    354    8193920 Campinas      29/01/2015           Sérgio Ribas
## 9    417    8193329 Campinas      29/01/2015           Poças Leitão
## 10   427    8193274 Sorocaba      29/01/2015           Poças Leitão
## # ... with 6,252 more rows
```

```
d_cjsg %>%
  select(comarca) %>%
  filter(str_detect(comarca, '^[gG]'))
```

```
## # A tibble: 6,463 × 1
##      comarca
##        <chr>
## 1  Guarulhos
## 2  Guarulhos
## 3  Guarulhos
## 4  Guarulhos
## 5      Guará
## 6  Guarulhos
## 7  Guarulhos
## 8  Guarulhos
## 9  Guarulhos
## 10 Guarulhos
## # ... with 6,453 more rows
```

6.1.7 `mutate`

- Parecido com `transform`, mas aceita várias novas colunas iterativamente.
- Novas variáveis devem ter o mesmo `length` que o `nrow` do bd oridinal ou `1`.

```
library(stringr)
d_cjsg %>%
  select(id, n_processo, data_julgamento) %>%
  mutate(ano_julgamento = year(dmy(data_julgamento)),
         ano_proc = str_sub(n_processo, 12, 15),
         ano_proc = as.numeric(ano_proc),
         tempo_anos = ano_julgamento - ano_proc)
```

```
## # A tibble: 184,250 × 6
```

```
##       id                n_processo data_julgamento ano_julgamento ano_proc
##    <chr>                     <chr>           <chr>          <dbl>    <dbl>
## 1      1 0081568-68.2012.8.26.0050      29/01/2015           2015     2012
## 2      2 2132739-15.2014.8.26.0000      27/01/2015           2015     2014
## 3      3 2204759-04.2014.8.26.0000      27/01/2015           2015     2014
## 4      4 2179832-71.2014.8.26.0000      22/01/2015           2015     2014
## 5      5 2055376-49.2014.8.26.0000      27/01/2015           2015     2014
## 6      6 0014476-05.2014.8.26.0050      29/01/2015           2015     2014
## 7      7 0067859-48.2014.8.26.0000      29/01/2015           2015     2014
## 8      8 0054802-60.2014.8.26.0000      27/01/2015           2015     2014
## 9      9 2128810-71.2014.8.26.0000      29/01/2015           2015     2014
## 10    10 2002343-13.2015.8.26.0000      27/01/2015           2015     2015
## # ... with 184,240 more rows, and 1 more variables: tempo_anos <dbl>
```

6.1.8 arrange

- Simplesmente ordena de acordo com as opções.
- Utilizar `desc` para ordem decrescente.

```
library(stringr)
d_cjsg %>%
  select(id, n_processo, data_julgamento) %>%
  mutate(ano_julgamento = year(dmy(data_julgamento)),
         ano_proc = str_sub(n_processo, 12, 15),
         ano_proc = as.numeric(ano_proc)) %>%
  mutate(tempo_anos = ano_julgamento - ano_proc) %>%
  arrange(desc(tempo_anos))
```

```
## # A tibble: 184,250 × 6
##       id                n_processo data_julgamento ano_julgamento ano_proc
##    <chr>                     <chr>           <chr>          <dbl>    <dbl>
## 1   1797 0901097-67.1957.8.26.0050      30/04/2015           2015     1957
## 2  13650 0815784-50.1971.8.26.0050      12/11/2015           2015     1971
## 3   7222 9000001-18.1976.8.26.0309      16/07/2015           2015     1976
## 4   8228 9000001-74.1979.8.26.0224      23/02/2015           2015     1979
## 5   6512 0909882-46.1979.8.26.0050      26/05/2015           2015     1979
## 6   3439 9000001-81.1981.8.26.0005      26/02/2015           2015     1981
## 7   1469 0000014-17.1982.8.26.0292      29/01/2015           2015     1982
## 8  12850 0000784-59.1982.8.26.0114      02/07/2015           2015     1982
## 9   7040 0000019-02.1983.8.26.0584      27/01/2015           2015     1983
## 10  3920 2050003-20.1984.8.26.0281      26/02/2015           2015     1984
## # ... with 184,240 more rows, and 1 more variables: tempo_anos <dbl>
```

6.1.9 summarise

- Retorna um vetor de tamanho 1 a partir de uma conta com as variáveis.
- Geralmente é utilizado em conjunto com `group_by`.
- Algumas funções importantes: `n()`, `n_distinct()`.

```
d_cjsg %>%
  select(id, n_processo, comarca, data_julgamento, orgao_julgador) %>%
  mutate(ano_julgamento = year(dmy(data_julgamento)),
         ano_proc = str_sub(n_processo, 12, 15),
         ano_proc = as.numeric(ano_proc)) %>%
```

```
  mutate(tempo_anos = ano_julgamento - ano_proc) %>%
  arrange(desc(tempo_anos)) %>%
  group_by(comarca, orgao_julgador) %>%
  summarise(n = n(),
            media_anos = mean(tempo_anos),
            min_anos = min(tempo_anos),
            max_anos = max(tempo_anos)) %>%
  filter(n > 5) %>%
  arrange(desc(media_anos))
```

```
## Source: local data frame [4,351 x 6]
## Groups: comarca [271]
##
##                  comarca                    orgao_julgador     n
##                    <chr>                             <chr> <int>
## 1                 Piraju 2ª Câmara Criminal Extraordinária     8
## 2         José Bonifácio 6ª Câmara Criminal Extraordinária     6
## 3           Miguelópolis 2ª Câmara Criminal Extraordinária     6
## 4                Piedade 3ª Câmara Criminal Extraordinária     6
## 5               Valinhos 1ª Câmara Criminal Extraordinária     6
## 6       Lençóis Paulista     2ª Câmara de Direito Criminal     8
## 7        Pindamonhangaba 5ª Câmara Criminal Extraordinária     8
## 8           Praia Grande 6ª Câmara Criminal Extraordinária     8
## 9  Santa Bárbara D Oeste 4ª Câmara Criminal Extraordinária     7
## 10             Ituverava 3ª Câmara Criminal Extraordinária     6
## # ... with 4,341 more rows, and 3 more variables: media_anos <dbl>,
## #   min_anos <dbl>, max_anos <dbl>
```

```
d_cjsg %>%
  count(relatora, sort = TRUE) %>%
  mutate(prop = n / sum(n), prop = scales::percent(prop))
```

```
## # A tibble: 111 × 3
##                     relatora     n  prop
##                        <chr> <int> <chr>
## 1              Sérgio Coelho  3424 1.86%
## 2    Miguel Marques e Silva  3352 1.82%
## 3               Poças Leitão  2859 1.55%
## 4           Walter da Silva  2834 1.54%
## 5           Francisco Bruno  2767 1.50%
## 6            Edison Brandão  2748 1.49%
## 7                Souza Nery  2739 1.49%
## 8              Carlos Bueno  2730 1.48%
## 9             Ivo de Almeida  2661 1.44%
## 10 Guilherme de Souza Nucci  2639 1.43%
## # ... with 101 more rows
```

6.1.10 gather

- "Empilha" o banco de dados

```
library(tidyr)
d_cjsg %>%
  select(cd_acordao:data_registro) %>%
```

```
  gather(key, value, -cd_acordao) %>%
  arrange(cd_acordao)
```

```
## # A tibble: 737,000 × 3
##    cd_acordao            key                     value
##         <chr>          <chr>                     <chr>
## 1     8137574      n_processo 0209050-18.2013.8.26.0000
## 2     8137574         comarca                 São Paulo
## 3     8137574 data_julgamento                22/01/2015
## 4     8137574   data_registro                22/01/2015
## 5     8137600      n_processo 0016319-58.2014.8.26.0000
## 6     8137600         comarca                 São Paulo
## 7     8137600 data_julgamento                22/01/2015
## 8     8137600   data_registro                22/01/2015
## 9     8137628      n_processo 0104838-68.2005.8.26.0050
## 10    8137628         comarca                 São Paulo
## # ... with 736,990 more rows
```

6.1.11 spread

- "Joga" uma variável nas colunas
- É essencialmente a função inversa de `gather`

```
d_cjsg %>%
  distinct(cd_acordao, .keep_all = TRUE) %>%
  select(cd_acordao:data_registro) %>%
  gather(key, value, -cd_acordao) %>%
  spread(key, value)
```

```
## # A tibble: 184,249 × 5
##    cd_acordao                  comarca data_julgamento data_registro
## *       <chr>                    <chr>           <chr>         <chr>
## 1     8137574                São Paulo      22/01/2015    22/01/2015
## 2     8137600                São Paulo      22/01/2015    22/01/2015
## 3     8137628                São Paulo      22/01/2015    22/01/2015
## 4     8137638                São Paulo      22/01/2015    22/01/2015
## 5     8137663 Espírito Santo do Pinhal      22/01/2015    22/01/2015
## 6     8137671     Presidente Venceslau      22/01/2015    22/01/2015
## 7     8137677       Paraguaçu Paulista      22/01/2015    22/01/2015
## 8     8137690                Guarulhos      22/01/2015    22/01/2015
## 9     8137695                   Osasco      22/01/2015    22/01/2015
## 10    8137696                Americana      22/01/2015    22/01/2015
## # ... with 184,239 more rows, and 1 more variables: n_processo <chr>
```

6.1.12 Funções auxiliares

- `unite` junta duas ou mais colunas usando algum separador (_, por exemplo).
- `separate` faz o inverso de `unite`, e uma coluna em várias usando um separador.

```
d_cjsg %>%
  select(n_processo, classe_assunto) %>%
  separate(classe_assunto, c('classe', 'assunto'), sep = ' / ',
           extra = 'merge', fill = 'right') %>%
  count(assunto, sort = TRUE)
```

```
## # A tibble: 287 × 2
##                                                 assunto     n
##                                                   <chr> <int>
## 1                  Tráfico de Drogas e Condutas Afins 30993
## 2                                               Roubo 17556
## 3                                     Roubo Majorado 17461
## 4  Crimes de Tráfico Ilícito e Uso Indevido de Drogas 13635
## 5                                   Furto Qualificado 13275
## 6                          Pena Privativa de Liberdade 10801
## 7                                               Furto  9870
## 8                                Progressão de Regime  7068
## 9                                          Receptação  5978
## 10                Crimes do Sistema Nacional de Armas  5675
## # ... with 277 more rows
```

6.1.13 Um pouco mais de transformação de dados

- Para juntar tabelas, usar `inner_join`, `left_join`, `anti_join`, etc.
- Para realizar operações mais gerais, usar `do`.
- Para retirar duplicatas, utilizar `distinct`.

Bibliography