Curso de Jurimetria

Associação Brasileira de Jurimetria

2016-07-28

Contents

Chapter 1

Prerequisites

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports, e.g., a math equation $a^2 + b^2 = c^2$.

For now, you have to install the development versions of **bookdown** from Github:

```
devtools::install_github("rstudio/bookdown")
```

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

To compile this example to PDF, you need to install XeLaTeX.

Chapter 2

Introduction

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter 2. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter **??**.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 2.1. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table 2.1.

```
knitr::kable(
  head(iris, 20), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

You can write citations, too. For example, we are using the **bookdown** package (Xie, 2016) in this sample book, which was built on top of R Markdown and **knitr** (Xie, 2015). ddd aaa

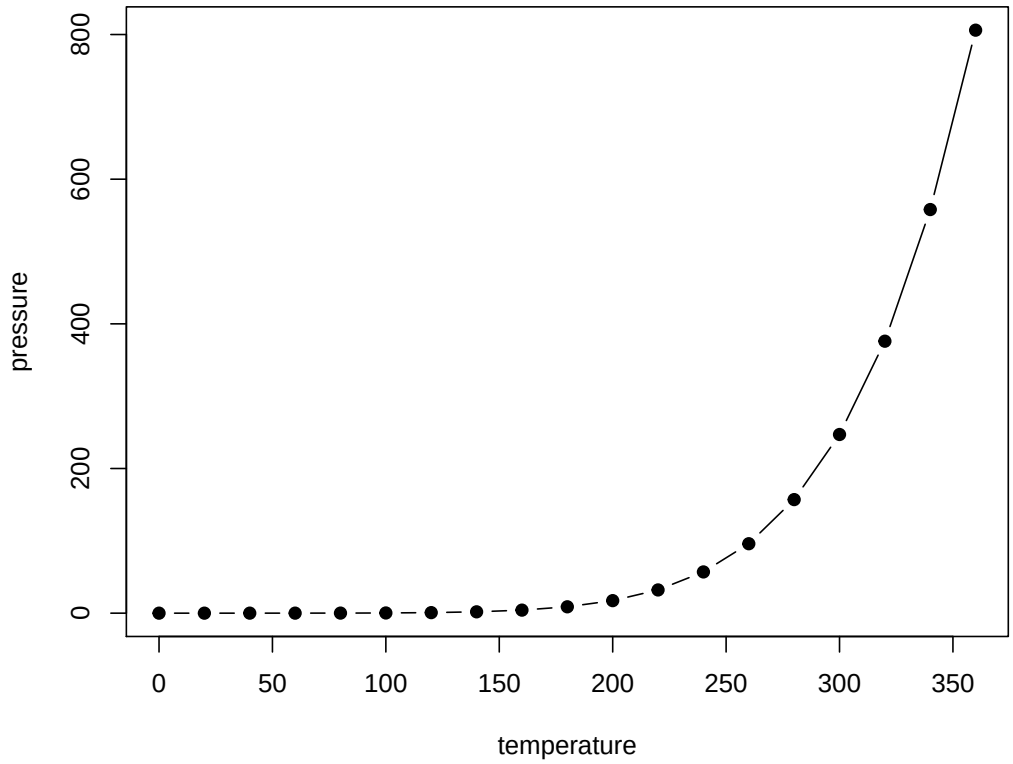


Figure 2.1: Here is a nice figure!

Table 2.1: Here is a nice table!

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

Chapter 3

Ferramental de trabalho da ABJ

As bases de dados utilizadas em estudos jurimétricos foram originalmente concebidas para fins gerenciais e não analíticos. Por isso, observamos muitos dados faltantes, mal formatados e com documentação inadequada. Uma boa porção dos dados só está disponível em páginas HTML e arquivos PDF e grande parte da informação útil está escondida em textos.

Chamamos esse fenômeno de "pré-sal sociológico". Temos hoje diversas bases de dados armazenadas em repositórios públicos ou controladas pelo poder público, mas que precisam ser lapidados para obtenção de informação útil.

Nessa parte do curso, vamos trabalhar com *data wrangling*, que consiste em trabalhar com ferramentas de extração, consolidação e transformação de dados. As ferramentas utilizadas fazem parte do chamado `tidyverse`, um universo de pacotes contemporâneos do `R` que são intuitivas, eficientes e úteis.

Os pacotes utilizados são `httr`, `xml2`, `rvest`, `dplyr`, `tidyr`, `purrr`, `lubridate`, `stringr`, `ggplot2`. Também utilizamos um pacote chamado `abjutils`, construído para atender algumas necessidades frequentes na ABJ. Recomenda-se a utilização do R 3.3.1 e o RStudio preview version. É possível instalar todos esses pacotes de uma vez rodando

```
if (!require(devtools)) install.packages('devtools')
devtools::install_github('abjur/abjutils')
```

Recomenda-se também a utilização de `linux`.

3.1 Exemplos trabalhados

3.1.1 Câmaras de gás

Uma das principais questões que surgem quando o tema é impunidade e que gerou as ideias iniciais desse artigo é: quando um réu condenado deve começar a cumprir pena? A justiça deve esperar o encerramento definitivo do processo, com o chamado trânsito em julgado, ou pode iniciar o cumprimento já a partir de uma decisão terminativa, como a sentença ou o acórdão de segundo grau?

Uma forma de solucionar esse debate é calcular as taxas de reforma de decisões em matéria criminal. Uma condição necessária para a viabilidade da antecipação do cumprimento de pena é uma baixa taxa de reforma das decisões, pois uma taxa alta implicaria que muitas pessoas seriam presas injustamente.

Com o objetivo de obter essas taxas, a presente pesquisa utiliza como base de dados um le- vantamento de 157.379 decisões em segunda instância, das quais pouco menos de 60.000 envolvem apelações contra o Ministério Público, todas proferidas entre 01/01/2014 e 31/12/2014 nas dezes- seis Câmaras de Direito Criminal do Estado de São Paulo, e nas Câmaras Extraordinárias. Todas as informações foram obtidas

através de ferramentas computacionais a partir de bases de dados disponí- veis publicamente, o que permite a reprodutibilidade da pesquisa. Os dados semi-estruturados foram organizados a partir da utilização de técnicas de mineração de texto. Também foi necessário utilizar procedimentos estatísticos adequados para lidar com problemas de dados faltantes.

Os resultados revelam taxas de reforma próximas a 50%. As taxas obtidas são relevantes e justificam a não antecipação do cumprimento de pena para a decisão em primeira instância. Com o intuito de complementar e aprofundar a pesquisa, realizamos análises para tipos específicos de crime, como roubo e tráfico de drogas, comparando as taxas de reforma em cada subpopulação. Realizamos também a comparação dos resultados relativamente às câmaras de julgamento e relatores.

A partir dessa análise, observamos uma alta variabilidade na taxa de reforma entre as vinte câmaras de julgamento. Encontramos câmaras com mais de 75% de recursos negados (quarta e sexta) e câmaras com menos de 30% de recursos negados (primeira, segunda e décima segunda). O resultado é contraintuitivo pois teoricamente a alocação de novos recursos nas câmaras é aleatória.

3.1.2 Waze do Judiciário

A produtividade de varas e juízes é um tema corrente na administração do judiciário. É importante mensurar a produtividade para fins de promoção e identificação de boas ou más práticas de atuação. No entanto, a complexidade processual, que não é observável, torna o problema de mensuração mais complicado do que simplesmente contar estatísticas de sentenças por mês.

Em 2014 a ABJ trabalhou com o TJSP na realização de um projeto para auxiliar na administração do judiciário. Um dos objetivos desse projeto foi realizar análise de agrupamento de varas do Tribunal, com o intuito de detectar varas problemáticas e também varas-modelo, que poderiam auxiliar outras varas na gestão dos processos.

A análise de agrupamento pode ser utilizada para separar varas em grupos e investigar o perfil de produtividade das varas de cada um dos grupos formados. Por exemplo, ao separar um conjunto de 10 varas em 3 grupos, poderíamos detectar um grupo que está produzindo pouco em relação à quantidade de funcionários, ou então um conjunto formado por uma vara só, que possui processos de execução de títulos extrajudiciais em excesso. Dessa forma, a identificação e investigação das características desses grupos poderiam ajudar em ações estratégicas do tribunal, como critérios para alocação de recursos (investimento em varas problemáticas) e criação de treinamentos (a partir da identificação de varas-modelo).

Ao comparar varas é usual considerar informações de orçamento, recursos humanos (número de funcionários e magistrados), e informações de movimentação processual (número de processos distribuídos, tamanho do acervo, quantidade de julgamentos, etc). Não se pode comparar diretamente maçã com banana: por exemplo, varas especializadas em execução fiscal são difíceis de comparar varas de família.

A base de dados do projeto foi obtida automaticamente através de ferramentas de web scraping e mineração de documentos PDF. Os dados contêm informações de quantidade de funcionários e diversas contagens mensais da vara como acervo, número de sentenças e número de distribuições.

O produto final do projeto foi chamado de “waze do judiciário”. Trata-se de um aplicativo online de visualização interativa, em que o usuário pode selecionar as entrâncias, alguns tipos de varas e a quantidade de grupos a serem formados.

3.2 Data tidying

Uma base de dados é considerada “tidy” se

- Cada observação é uma linha do bd.
- Cada variável é uma coluna do bd.
- Para cada unidade observacional temos um `data.frame` separado (possivelmente com chaves de associação).

O objetivo em *data wrangling* é extrair e transformar uma base de dados até que ela esteja em formato *tidy*. Em seguida, mostraremos como fizemos isso no exemplo das câmaras. Adicionalmente, vamos apresentar como foram trabalhados os arquivos PDF no caso do Waze do judiciário.

3.3 Web scraping

Este documento contém algumas melhores práticas na contrução de ferramentas no R que baixam e processam informações de sites disponíveis na web. O objetivo é ajudar o desenvolvedor a produzir um pacote que seja fácil de adaptar no tempo.

O documento foi construído com base na experiência em web scrapers simples, contruídos para acessar listas de páginas pré-definidas. Isto é muito diferente de web crawlers não supervisionados, como o do Google, que vão passeando pelas páginas de forma indefinida. Por conta disso, o nome crawler poderia até ser um pouco inadequado, mas estamos mantendo por falta de um nome melhor para definir essa tarefa.

Também é importante mencionar que só estamos trabalhando com páginas que são acessíveis publicamente. Caso tenha interesse e "raspar" páginas que precisam de autenticação, recomendamos que estude os termos de uso do site.

Para ilustrar este texto, usaremos como exemplo o código utilizado no trabalho das câmaras, que acessa o site do Tribunal de Justiça de São Paulo para obter informações de processos judiciais. Trabalharemos principalmente com a Consulta de Jurisprudência e a Consulta de de Processos de Segundo Grau.

3.4 Informações iniciais

Antes de tudo, verifique se existe alguma forma mais fácil de conseguir os dados que necessita. Construir um web scraper do zero é muitas vezes uma tarefa dolorosa e, caso o site seja atualizado, pode ser que boa parte do trabalho seja inútil. Se os dados precisarem ser extraídos apenas uma vez, verifique com o pessoal que mantém o site se eles não podem fazer a extração que precisa. Se os dados precisarem ser atualizados, verifique se a entidade não possui uma API para acesso aos dados.

Ao escrever um web scraper, as primeiras coisas que devemos pensar são

- Como o site a ser acessado foi contruído, se tem limites de requisições, utilização de cookies, states, etc.
- Como e com que frequência o site é atualizado, tanto em relação à sua interface como em relação aos dados que queremos extrair.
- Como conseguir a lista das páginas que queremos acessar.
- Qual o caminho percorrido para acessar uma página específica.

Sugerimos como melhores práticas dividir todas as atividades em três tarefas principais: i) buscar; ii) coletar e iii) processar. Existem casos em que a etapa de busca é desnecessária (por exemplo, se já sabemos de antemão quais são as URLs que vamos acessar).

Na maior parte dos casos, deixar os algoritmos de *download* e *parsing* dos dados em funções distintas é uma boa prática pois aumenta o controle sobre o que as ferramentas estão fazendo, facilita o debug e a atualização. Em alguns casos, no entanto, isso pode tornar o código mais ineficiente e os arquivos obtidos podem ficar pesados.

3.4.1 Diferença entre procurar, baixar e processar.

Procurar documentos significa, de uma forma geral, utilizar ferramentas de busca (ou acessar links de um site) para obter informações de uma nova requisição a ser realizada. Ou seja, essa etapa do scraper serve para "procurar links" que não sabíamos que existiam previamente. Isso será resolvido através da função `cjsg`.

Baixar documentos, no entando, significa simplesmente acessar páginas pré-estabelecidas e salvá-las em disco. Em algumas situações, os documentos baixados (depois de limpos) podem conter uma nova lista de páginas a serem baixadas, formando iterações de coletas. A tarefa de baixar documentos pré-estabelecidos será realizada pela função `cposg`.

Finalmente, processar documentos significa carregar dados acessíveis em disco e transformar os dados brutos uma base *tidy*. Não existe um limite para a profundidade dessa estruturação de dados. Geralmente, no entanto, separamos a etapa de estruturação em duas atividades: i) transformar arquivos não-estruturados em um arquivos semi-estruturados (e.g. um arquivo HTML em uma tabela mais um conjunto de textos livres) e ii) transformar arquivos semi-estruturados em uma base analítica (estruturada). A tarefa de processar as páginas será realizada pelas funções `parse_cjsg` e `parse_cpopg`.

Como veremos no decorrer do documento, no caso do TJSP, teremos um fluxo "look for" -> "collect" -> "scrape" -> "collect" -> "scrape" para conseguir nossos dados.

3.5 Procurar documentos

A tarefa de listar os documentos que queremos obter geralmente pode ser realizada de duas formas: i) utilizar uma ferramenta de busca do site e ii) acessar as páginas a partir do resultado de uma pesquisa. Dependendo do caso, será necessário realizar:

- Uma busca e uma paginação;
- Uma busca e muitas paginações;
- Muitas buscas e uma paginação por busca;
- Muitas buscas e muitas paginações por busca.

No exemplo de ilustração, nosso caso é de *uma busca e muitas paginações*. Acesse a página do e-SAJ e clique em "Consultar" para ter uma ideia de como é essa página. A página (acessada no dia 2016-07-28) é uma ferramenta de busca com vários campos, que permite pesquisa com dados em branco. Na parte de baixo o site mostra uma série de documentos, organizados em páginas de dez em dez resultados.

Para resolver o problema, precisaremos de duas funções principais, uma que faz a busca e outra que acessa uma página específica (que será repetida várias vezes). Utilizaremos as funções `look_for` e `paginate` para cada um desses problemas.

3.5.1 Search docs

```
cjpg_url <- function() {
  u <- 'https://esaj.tjsp.jus.br/cjpg/pesquisar.do'
  u
}
```

A função `search_docs` precisa ser capaz de realizar uma pesquisa e retornar a resposta do servidoe que contém a primeira página dos resultados. Para isso, ela recebe uma lista com dados da busca (do formulário) a url base e um método para realizar a requisição, podendo ser 'get' ou 'post'. Caso a pesquisa seja mais complicada, é possível adicionar também uma função que sobrepõe a busca padrão.

Futuro: A função também realizará algumas tarefas conhecidas de forma automática. Primeiramente acessa a página inicial, verifica se ela contém certos tipos de tags ocultas, como '___VIEWSTATE' para páginas em aspx ou 'javax.server.faces' para páginas em java faces e adicionará esses parâmetros automaticamente na requisição, quando esta for do tipo 'POST'.

No nosso caso, a requisição é um simples 'GET', mas com muitos parâmetros. Assim, o nosso pacote, dentro do esquema do pacotes `crawlr`, ficaria algo como:

```
cjpg_search_data <- list(
  'dadosConsulta.nuProcesso' = '',
  'dadosConsulta.pesquisaLivre' = 'danos morais',
  'dadosConsulta.dtInicio' = '01/10/2014',
  'dadosConsulta.dtFim' = '01/11/2014'
)

cjpg_search_result <- cjpg_search_data %>%
  search_docs(url = cjpg_url(), method = 'get')

cjpg_search_result
```

Também é possível incluir os parâmetros diretamente na função `search_docs`

```
cjpg_search_result <- search_docs(
  url = cjpg_url(),
  'dadosConsulta.nuProcesso' = '',
  'dadosConsulta.pesquisaLivre' = 'danos morais',
  'dadosConsulta.dtInicio' = '01/10/2014',
  'dadosConsulta.dtFim' = '01/11/2014',
  method = 'get'
)
```

No RStudio, é possível visualizar a página baixada com a função `visualize`. O documento aparecerá na

```
visualize(cjpg_search_result)
```

OBS: A imagem fica "feia" pois está sem a folha de estilos e as imagens.

Em alguns casos ser uma boa prática criar funções que facilitam a entrada de parâmetros de busca. No nosso exemplo, existem parâmetros necessários na requisição que não precisam ser preenchidos, e parâmetros que precisam ser preenchidos de uma maneira específica, como as datas, que precisam ser inseridas no formato '%d/%m/%Y'. Assim, incluimos uma função de "ajuda".

```
cjpg_parms <- function(livre = '', classes = '', assuntos = '',
                       data_inicial = '', data_final = '', varas = '') {
  classes = paste0(classes, collapse = ',')
  assuntos = paste0(assuntos, collapse = ',')
  varas = paste0(varas, collapse = ',')
  if(data_inicial != '' & data_final != '') {
    # aqui eu uso o pacote lubridate para construir a data no formato
    # que o e-SAJ exige.
    cod_data_inicial <- paste(lubridate::day(data_inicial),
                              lubridate::month(data_inicial),
                              lubridate::year(data_inicial) ,
                              sep = '/')
    cod_data_final <- paste(lubridate::day(data_final),
                            lubridate::month(data_final),
                            lubridate::year(data_final) ,
                            sep = '/')
  }
  parms <- list('dadosConsulta.nuProcesso' = '',
                'dadosConsulta.pesquisaLivre' = livre,
                'classeTreeSelection.values' = classes,
                'assuntoTreeSelection.values' = assuntos,
                'varasTreeSelection.values' = varas,
```

```
                'dadosConsulta.dtInicio' = cod_data_inicial,
                'dadosConsulta.dtFim' = cod_data_final)
  parms
}
```

Dessa forma, a chamada ficaria um pouco mais padronizada.

```
cjpg_search_data <- cjpg_parms(livre = 'danos morais',
                               data_inicial = '2014-10-01',
                               data_final = '2014-11-01')

# por default method = "get"
cjpg_search_result <-  cjpg_search_data %>% search_docs(cjpg_url())
```

É possível utilizr a função `cjpg_parms` com dados incluídos diretamente na função através do parâmetro `parm_fun`:

```
cjpg_search_result <- search_docs(url = cjpg_url(),
                                  livre = 'danos morais',
                                  data_inicial = '2014-10-01',
                                  data_final = '2014-11-01',
                                  parm_fun = cjpg_parms)
```

Por fim, é uma boa prática criar uma função que extrai o número de páginas a serem acessadas pela paginação. Geralmente esse número existe pois as ferramentas de busca usualmente mostram o número de resultados.

```
cjpg_npag <- function(r) {
  val <- xml2::read_html(httr::content(r, 'text')) %>%
    xml2::xml_find_all(".//*[@id = 'resultados']//td") %>%
    `[[`(1) %>%
    xml2::xml_text() %>%
    stringr::str_trim() %>%
    stringr::str_match('de ([0-9]+)')
  num <- as.numeric(val[1, 2])
  num
}

cjpg_npag(cjpg_search_result$result)
```

Podemos adicionar essa função como parâmetro `npag_fun` de nossa função `search_docs`.

```
cjpg_search_result <- search_docs(url = cjpg_url(),
                                  livre = 'danos morais',
                                  data_inicial = '2014-10-01',
                                  data_final = '2014-11-01',
                                  parm_fun = cjpg_parms,
                                  npag_fun = cjpg_npag)
```

O objeto retornado pela função `search_docs` é um objeto do tipo `searchdoc`, que guarda, além da resposta da requisição web, a url base utilizada, a lista com os parâmetros, e o número de páginas, estes últimos somente se os parâmetros `parm_fun` e `npag_fun` forem informados.

3.5.2 Paginate

Após conseguir os resultados pela ferramenta de busca e acessar os resultados, o próximo desafio é baixar as páginas dos resultados. Realizar a paginação nada mais é do que repetir a tarefa de acessar uma página

diversas vezes, mudando somente o parâmetro da página.

Algumas vezes, é possível que a URL base para acessar a paginação seja diferente da ferramenta de busca. Esse é o caso do nosso exemplo. Nesses casos, podemos criar uma nova função para guardar essa URL.

```
cjpg_url_pag <- function() {
  u <- 'https://esaj.tjsp.jus.br/cjpg/trocarDePagina.do'
  u
}
```

A função `paginate` recebe como parâmetros

- `.sch`. Um objeto de classe `searchdoc`.
- `pags`. As páginas que serão acessadas. O valor padrão é 'all', indicando que todas as páginas devem ser baixadas. Como alternativa, é possível informar ou um vetor nomeado `c(from = min, to = max)`, onde `min` e `max` são os extremos do intervalo de páginas que se deseja acessar, ou ainda um vetor numérico com os índices desejados.
- `pag_parm`, indicando o nome do parâmetro que identifica a página no site.

Alguns parâmetros opcionais a serem incluídos são

- `method` o método para acessar a página, entre 'get' e 'post', caso este seja diferente do método utilizado na função `search_docs`.
- `url` a url para acessar a página, caso esta seja diferente da utilizada na função `search_docs`.
- `name_fun` uma função para nomear os arquivos salvos. Mostraremos um exemplo em seguida.
- `wait` tempo de espera entre cada requisição. Pode ser útil caso o site tenha algum limitador.
- `path`, o caminho para salvar os arquivos. Por default é o próprio diretório.

Utilização básica de `paginate`

```
cjpg_search_result %>%
  paginate(pags = 1:30, parm_pag= 'pagina',
           url = cjpg_url_pag(), path = 'data-raw/cjpg')
```

```
dir('data-raw/cjpg')
```

No exemplo, fizemos o download de 30 páginas a partir do resultado da pesquisa. Os arquivos salvos são arquivos `.rds` que guardam os resultados das requisições. Eles devem ser lidos com a função `readRDS`.

Com isso, conseguimos baixar as páginas que **listam** os itens que queremos baixar. Em muitos casos, esse

Futuro: Infelizmente, alguns sites não permitem a inclusão do número de uma página como parâmetro para realizar a paginação. Muitas vezes esses sites possuem somente um botão "Next". No futuro vamos adaptar a função `paginate` para esses casos.

Onde guardar os dados? Ao construir um pacote que utiliza o pacote `crawlr`, pode fazer sentido guardar os dados baixados dentro do próprio pacote, para reprodutibilidade. Nesse caso, o melhor lugar para guardar esses dados é na pasta `data-raw`, como sugerido por Hadley Wickham no livro r-pkgs. No entanto, Se os dados forem muito volumosos, colocá-los dentro do pacote pode ser ruim para colocar no GitHub e no CRAN. Por isso, pode ser necessário colocar esses documentos numa pasta externa ao pacote. Para garantir a reprodutibilidade, recomendo que criem um pacote no R cujo objetivo é guardar somente esses dados, e coloque esse pacote em um repositório na nuvem (Dropbox, por exemplo). No pacote que contém as funções de extração, guarde os dados já processados (se couberem) num arquivo `.rda` dentro da pasta `data` do pacote.

3.6 Processar

3.7 TODO

- Mais exemplos.
- Melhor documentação.
- Adicionar métodos para usar selenium.

3.8 Manipulação de dados com dplyr

A manipulação 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.

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.

> "(...) 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

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`.

3.8.1 Trabalhando com tibbles

```
pnud_muni <- tbl_df(pnud_muni)
pnud_muni
```

```
## # A tibble: 16,695 x 238
##       uf   ano codmun6 codmun7             municipio espvida fectot mort1
## *  <dbl> <dbl>   <dbl>   <dbl>                 <chr>   <dbl>  <dbl> <dbl>
## 1     11  1991  110001 1100015 ALTA FLORESTA D'OESTE   62.01   4.08 45.58
## 2     11  1991  110002 1100023             ARIQUEMES   66.02   3.72 32.39
## 3     11  1991  110003 1100031                CABIXI   63.16   3.89 41.52
## 4     11  1991  110004 1100049                CACOAL   65.03   3.81 35.37
## 5     11  1991  110005 1100056            CEREJEIRAS   62.73   3.55 43.00
## 6     11  1991  110006 1100064     COLORADO DO OESTE   64.46   3.38 37.19
## 7     11  1991  110007 1100072            CORUMBIARA   59.32   3.95 56.02
## 8     11  1991  110008 1100080         COSTA MARQUES   62.76   4.19 42.90
## 9     11  1991  110009 1100098       ESPIGÃO D'OESTE   64.18   3.84 38.09
## 10    11  1991  110010 1100106         GUAJARÁ-MIRIM   64.71   4.19 36.41
## # ... with 16,685 more rows, and 230 more variables: mort5 <dbl>,
## #   razdep <dbl>, sobre40 <dbl>, sobre60 <dbl>, t_env <dbl>,
## #   e_anosestudo <dbl>, t_analf11a14 <dbl>, t_analf15a17 <dbl>,
## #   t_analf15m <dbl>, t_analf18a24 <dbl>, t_analf18m <dbl>,
## #   t_analf25a29 <dbl>, t_analf25m <dbl>, t_atraso_0_basico <dbl>,
## #   t_atraso_0_fund <dbl>, t_atraso_0_med <dbl>, t_atraso_1_basico <dbl>,
## #   t_atraso_1_fund <dbl>, t_atraso_1_med <dbl>, t_atraso_2_basico <dbl>,
## #   t_atraso_2_fund <dbl>, t_atraso_2_med <dbl>, t_fbbas <dbl>,
```

```
## #   t_fbfund <dbl>, t_fbmed <dbl>, t_fbpre <dbl>, t_fbsuper <dbl>,
## #   t_flbas <dbl>, t_flfund <dbl>, t_flmed <dbl>, t_flpre <dbl>,
## #   t_flsuper <dbl>, t_freq0a3 <dbl>, t_freq11a14 <dbl>,
## #   t_freq15a17 <dbl>, t_freq18a24 <dbl>, t_freq25a29 <dbl>,
## #   t_freq4a5 <dbl>, t_freq4a6 <dbl>, t_freq5a6 <dbl>, t_freq6 <dbl>,
## #   t_freq6a14 <dbl>, t_freq6a17 <dbl>, t_freqfund1517 <dbl>,
## #   t_freqfund1824 <dbl>, t_freqfund45 <dbl>, t_freqmed1824 <dbl>,
## #   t_freqmed614 <dbl>, t_freqsuper1517 <dbl>, t_fund11a13 <dbl>,
## #   t_fund12a14 <dbl>, t_fund15a17 <dbl>, t_fund16a18 <dbl>,
## #   t_fund18a24 <dbl>, t_fund18m <dbl>, t_fund25m <dbl>, t_med18a20 <dbl>,
## #   t_med18a24 <dbl>, t_med18m <dbl>, t_med19a21 <dbl>, t_med25m <dbl>,
## #   t_super25m <dbl>, corte1 <dbl>, corte2 <dbl>, corte3 <dbl>,
## #   corte4 <dbl>, corte9 <dbl>, gini <dbl>, pind <dbl>, pindcri <dbl>,
## #   pmpob <dbl>, pmpobcri <dbl>, ppob <dbl>, ppobcri <dbl>,
## #   pren10ricos <dbl>, pren20 <dbl>, pren20ricos <dbl>, pren40 <dbl>,
## #   pren60 <dbl>, pren80 <dbl>, prentrab <dbl>, r1040 <dbl>, r2040 <dbl>,
## #   rdpc <dbl>, rdpc1 <dbl>, rdpc10 <dbl>, rdpc2 <dbl>, rdpc3 <dbl>,
## #   rdpc4 <dbl>, rdpc5 <dbl>, rdpct <dbl>, rind <dbl>, rmpob <dbl>,
## #   rpob <dbl>, theil <dbl>, cpr <dbl>, emp <dbl>, p_agro <dbl>,
## #   p_com <dbl>, p_constr <dbl>, ...
```

3.8.2 Filosofia do Hadley para análise de dados

3.8.3 As cinco funções principais do dplyr

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

3.8.4 Características

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

3.8.5 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`.

3.8.6 select

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

```
# por indice (nao recomendavel!)
pnud_muni %>%
  select(1:10)
```

```
## # A tibble: 16,695 x 10
##       uf   ano codmun6 codmun7             municipio espvida fectot mort1
## *  <dbl> <dbl>   <dbl>   <dbl>                 <chr>   <dbl>  <dbl> <dbl>
## 1     11  1991  110001 1100015 ALTA FLORESTA D'OESTE   62.01   4.08 45.58
## 2     11  1991  110002 1100023             ARIQUEMES   66.02   3.72 32.39
## 3     11  1991  110003 1100031                CABIXI   63.16   3.89 41.52
## 4     11  1991  110004 1100049                CACOAL   65.03   3.81 35.37
## 5     11  1991  110005 1100056            CEREJEIRAS   62.73   3.55 43.00
## 6     11  1991  110006 1100064     COLORADO DO OESTE   64.46   3.38 37.19
## 7     11  1991  110007 1100072            CORUMBIARA   59.32   3.95 56.02
## 8     11  1991  110008 1100080         COSTA MARQUES   62.76   4.19 42.90
## 9     11  1991  110009 1100098       ESPIGÃO D'OESTE   64.18   3.84 38.09
## 10    11  1991  110010 1100106         GUAJARÁ-MIRIM   64.71   4.19 36.41
## # ... with 16,685 more rows, and 2 more variables: mort5 <dbl>,
## #   razdep <dbl>
```

```
# especificando nomes (maneira mais usual)
pnud_muni %>%
  select(ano, ufn, municipio, idhm)
```

```
## # A tibble: 16,695 x 4
##      ano      ufn             municipio  idhm
## *  <dbl>    <chr>                 <chr> <dbl>
## 1   1991 Rondônia ALTA FLORESTA D'OESTE 0.329
## 2   1991 Rondônia             ARIQUEMES 0.432
## 3   1991 Rondônia                CABIXI 0.309
## 4   1991 Rondônia                CACOAL 0.407
## 5   1991 Rondônia            CEREJEIRAS 0.386
## 6   1991 Rondônia     COLORADO DO OESTE 0.376
## 7   1991 Rondônia            CORUMBIARA 0.203
## 8   1991 Rondônia         COSTA MARQUES 0.425
## 9   1991 Rondônia       ESPIGÃO D'OESTE 0.388
## 10  1991 Rondônia         GUAJARÁ-MIRIM 0.468
## # ... with 16,685 more rows
```

```
# intervalos e funcoes auxiliares (para economizar trabalho)
pnud_muni %>%
  select(ano:municipio, starts_with('idhm'))
```

```
## # A tibble: 16,695 x 8
##      ano codmun6 codmun7             municipio  idhm idhm_e idhm_l idhm_r
## *  <dbl>   <dbl>   <dbl>                 <chr> <dbl>  <dbl>  <dbl>  <dbl>
## 1   1991  110001 1100015 ALTA FLORESTA D'OESTE 0.329  0.112  0.617  0.516
## 2   1991  110002 1100023             ARIQUEMES 0.432  0.199  0.684  0.593
## 3   1991  110003 1100031                CABIXI 0.309  0.108  0.636  0.430
## 4   1991  110004 1100049                CACOAL 0.407  0.171  0.667  0.593
## 5   1991  110005 1100056            CEREJEIRAS 0.386  0.167  0.629  0.547
## 6   1991  110006 1100064     COLORADO DO OESTE 0.376  0.151  0.658  0.536
## 7   1991  110007 1100072            CORUMBIARA 0.203  0.039  0.572  0.373
## 8   1991  110008 1100080         COSTA MARQUES 0.425  0.220  0.629  0.553
## 9   1991  110009 1100098       ESPIGÃO D'OESTE 0.388  0.159  0.653  0.561
## 10  1991  110010 1100106         GUAJARÁ-MIRIM 0.468  0.247  0.662  0.625
## # ... with 16,685 more rows
```

3.8.7 filter

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

```
# somente estado de SP, com IDH municipal maior que 80% no ano 2010
pnud_muni %>%
  select(ano, ufn, municipio, idhm) %>%
  filter(ufn=='São Paulo', idhm > .8, ano == 2010)
```

```
## # A tibble: 21 x 4
##      ano       ufn           municipio  idhm
##    <dbl>     <chr>               <chr> <dbl>
## 1   2010 São Paulo  ÁGUAS DE SÃO PEDRO 0.854
## 2   2010 São Paulo           AMERICANA 0.811
## 3   2010 São Paulo          ARARAQUARA 0.815
## 4   2010 São Paulo               ASSIS 0.805
## 5   2010 São Paulo               BAURU 0.801
## 6   2010 São Paulo            CAMPINAS 0.805
## 7   2010 São Paulo       ILHA SOLTEIRA 0.812
## 8   2010 São Paulo             JUNDIAÍ 0.822
## 9   2010 São Paulo        PIRASSUNUNGA 0.801
## 10  2010 São Paulo PRESIDENTE PRUDENTE 0.806
## # ... with 11 more rows
```

```
# mesma coisa que o anterior
pnud_muni %>%
  select(ano, ufn, municipio, idhm) %>%
  filter(ufn=='São Paulo' & idhm > .8 & ano == 2010)
```

```
## # A tibble: 21 x 4
##      ano       ufn           municipio  idhm
##    <dbl>     <chr>               <chr> <dbl>
## 1   2010 São Paulo  ÁGUAS DE SÃO PEDRO 0.854
## 2   2010 São Paulo           AMERICANA 0.811
## 3   2010 São Paulo          ARARAQUARA 0.815
## 4   2010 São Paulo               ASSIS 0.805
## 5   2010 São Paulo               BAURU 0.801
## 6   2010 São Paulo            CAMPINAS 0.805
## 7   2010 São Paulo       ILHA SOLTEIRA 0.812
## 8   2010 São Paulo             JUNDIAÍ 0.822
## 9   2010 São Paulo        PIRASSUNUNGA 0.801
## 10  2010 São Paulo PRESIDENTE PRUDENTE 0.806
## # ... with 11 more rows
```

```
# !is.na(x)
pnud_muni %>%
  select(ano, ufn, municipio, idhm, pea) %>%
  filter(!is.na(pea))
```

```
## # A tibble: 11,130 x 5
##      ano      ufn             municipio  idhm   pea
##    <dbl>    <chr>                 <chr> <dbl> <dbl>
## 1   2000 Rondônia ALTA FLORESTA D'OESTE 0.483 12670
## 2   2000 Rondônia             ARIQUEMES 0.556 33705
## 3   2000 Rondônia                CABIXI 0.488  3227
## 4   2000 Rondônia                CACOAL 0.567 34206
```

```
## 5   2000 Rondônia            CEREJEIRAS 0.542  8407
## 6   2000 Rondônia     COLORADO DO OESTE 0.545  9576
## 7   2000 Rondônia            CORUMBIARA 0.401  3729
## 8   2000 Rondônia         COSTA MARQUES 0.486  3686
## 9   2000 Rondônia       ESPIGÃO D'OESTE 0.501 10428
## 10  2000 Rondônia         GUAJARÁ-MIRIM 0.573 15802
## # ... with 11,120 more rows
```

```
# %in%
pnud_muni %>%
  select(ano, ufn, municipio, idhm) %>%
  filter(municipio %in% c('CAMPINAS', 'SÃO PAULO'))
```

```
## # A tibble: 6 x 4
##     ano       ufn municipio  idhm
##   <dbl>     <chr>     <chr> <dbl>
## 1  1991 São Paulo  CAMPINAS 0.618
## 2  1991 São Paulo SÃO PAULO 0.626
## 3  2000 São Paulo  CAMPINAS 0.735
## 4  2000 São Paulo SÃO PAULO 0.733
## 5  2010 São Paulo  CAMPINAS 0.805
## 6  2010 São Paulo SÃO PAULO 0.805
```

3.8.8 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`.

```
pnud_muni %>%
  select(ano, ufn, municipio, idhm) %>%
  filter(ano == 2010) %>%
  mutate(idhm_porc = idhm * 100,
         idhm_porc_txt = paste(idhm_porc, '%'))
```

```
## # A tibble: 5,565 x 6
##      ano      ufn             municipio  idhm idhm_porc idhm_porc_txt
##    <dbl>    <chr>                 <chr> <dbl>     <dbl>         <chr>
## 1   2010 Rondônia ALTA FLORESTA D'OESTE 0.641      64.1        64.1 %
## 2   2010 Rondônia             ARIQUEMES 0.702      70.2        70.2 %
## 3   2010 Rondônia                CABIXI 0.650      65.0          65 %
## 4   2010 Rondônia                CACOAL 0.718      71.8        71.8 %
## 5   2010 Rondônia            CEREJEIRAS 0.692      69.2        69.2 %
## 6   2010 Rondônia     COLORADO DO OESTE 0.685      68.5        68.5 %
## 7   2010 Rondônia            CORUMBIARA 0.613      61.3        61.3 %
## 8   2010 Rondônia         COSTA MARQUES 0.611      61.1        61.1 %
## 9   2010 Rondônia       ESPIGÃO D'OESTE 0.672      67.2        67.2 %
## 10  2010 Rondônia         GUAJARÁ-MIRIM 0.657      65.7        65.7 %
## # ... with 5,555 more rows
```

```
# media de idhm_l e idhm_e
pnud_muni %>%
  select(ano, ufn, municipio, starts_with('idhm')) %>%
  filter(ano == 2010) %>%
  mutate(idhm2 = (idhm_e + idhm_l) / 2)
```

```
## # A tibble: 5,565 x 8
```

```
##      ano      ufn             municipio  idhm idhm_e idhm_l idhm_r  idhm2
##    <dbl>    <chr>                 <chr> <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1   2010 Rondônia ALTA FLORESTA D'OESTE 0.641  0.526  0.763  0.657 0.6445
## 2   2010 Rondônia             ARIQUEMES 0.702  0.600  0.806  0.716 0.7030
## 3   2010 Rondônia                CABIXI 0.650  0.559  0.757  0.650 0.6580
## 4   2010 Rondônia                CACOAL 0.718  0.620  0.821  0.727 0.7205
## 5   2010 Rondônia           CEREJEIRAS 0.692  0.602  0.799  0.688 0.7005
## 6   2010 Rondônia     COLORADO DO OESTE 0.685  0.584  0.814  0.676 0.6990
## 7   2010 Rondônia            CORUMBIARA 0.613  0.473  0.774  0.630 0.6235
## 8   2010 Rondônia         COSTA MARQUES 0.611  0.493  0.751  0.616 0.6220
## 9   2010 Rondônia       ESPIGÃO D'OESTE 0.672  0.536  0.819  0.691 0.6775
## 10  2010 Rondônia         GUAJARÁ-MIRIM 0.657  0.519  0.823  0.663 0.6710
## # ... with 5,555 more rows
```

```
# errado
pnud_muni %>%
  select(ano, ufn, municipio, starts_with('idhm')) %>%
  filter(ano == 2010) %>%
  mutate(idhm2 = mean(c(idhm_e, idhm_l)))

# uma alternativa (+ demorada)
pnud_muni %>%
  select(ano, ufn, municipio, starts_with('idhm')) %>%
  filter(ano == 2010) %>%
  rowwise() %>%
  mutate(idhm2 = mean(c(idhm_e, idhm_l)))
```

3.8.9 arrange

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

```
pnud_muni %>%
  select(ano, ufn, municipio, idhm) %>%
  filter(ano == 2010) %>%
  mutate(idhm_porc = idhm * 100,
         idhm_porc_txt = paste(idhm_porc, '%')) %>%
  arrange(idhm)
```

```
## # A tibble: 5,565 x 6
##      ano      ufn          municipio  idhm idhm_porc idhm_porc_txt
##    <dbl>    <chr>              <chr> <dbl>     <dbl>         <chr>
## 1   2010     Pará            MELGAÇO 0.418      41.8        41.8 %
## 2   2010 Maranhão    FERNANDO FALCÃO 0.443      44.3        44.3 %
## 3   2010 Amazonas   ATALAIA DO NORTE 0.450      45.0          45 %
## 4   2010 Maranhão     MARAJÁ DO SENA 0.452      45.2        45.2 %
## 5   2010  Roraima           UIRAMUTÃ 0.453      45.3        45.3 %
## 6   2010     Pará             CHAVES 0.453      45.3        45.3 %
## 7   2010     Acre             JORDÃO 0.469      46.9        46.9 %
## 8   2010     Pará              BAGRE 0.471      47.1        47.1 %
## 9   2010     Pará CACHOEIRA DO PIRIÁ 0.473      47.3        47.3 %
## 10  2010 Amazonas          ITAMARATI 0.477      47.7        47.7 %
## # ... with 5,555 more rows
```

```
pnud_muni %>%
  select(ano, ufn, municipio, idhm) %>%
  filter(ano == 2010) %>%
  mutate(idhm_porc = idhm * 100,
         idhm_porc_txt = paste(idhm_porc, '%')) %>%
  arrange(desc(idhm))
```

```
## # A tibble: 5,565 x 6
##      ano              ufn          municipio  idhm idhm_porc idhm_porc_txt
##    <dbl>            <chr>              <chr> <dbl>     <dbl>         <chr>
## 1   2010        São Paulo SÃO CAETANO DO SUL 0.862      86.2        86.2 %
## 2   2010        São Paulo ÁGUAS DE SÃO PEDRO 0.854      85.4        85.4 %
## 3   2010   Santa Catarina      FLORIANÓPOLIS 0.847      84.7        84.7 %
## 4   2010   Espírito Santo            VITÓRIA 0.845      84.5        84.5 %
## 5   2010   Santa Catarina BALNEÁRIO CAMBORIÚ 0.845      84.5        84.5 %
## 6   2010        São Paulo             SANTOS 0.840      84.0          84 %
## 7   2010   Rio de Janeiro           NITERÓI 0.837      83.7        83.7 %
## 8   2010   Santa Catarina           JOAÇABA 0.827      82.7        82.7 %
## 9   2010 Distrito Federal          BRASÍLIA 0.824      82.4        82.4 %
## 10  2010           Paraná          CURITIBA 0.823      82.3        82.3 %
## # ... with 5,555 more rows
```

3.8.10 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()`.

```
pnud_muni %>%
  filter(ano == 2010) %>%
  group_by(ufn) %>%
  summarise(n = n(),
            idhm_medio = mean(idhm),
            populacao_total = sum(popt)) %>%
  arrange(desc(idhm_medio))
```

```
## # A tibble: 27 x 4
##                   ufn     n idhm_medio populacao_total
##                 <chr> <int>      <dbl>           <dbl>
## 1    Distrito Federal     1  0.8240000         2541714
## 2           São Paulo   645  0.7395271        40915379
## 3      Santa Catarina   293  0.7316485         6199947
## 4   Rio Grande do Sul   496  0.7135302        10593371
## 5      Rio de Janeiro    92  0.7089130        15871447
## 6              Paraná   399  0.7019599        10348247
## 7               Goiás   246  0.6949837         5934769
## 8      Espírito Santo    78  0.6921923         3477471
## 9         Mato Grosso   141  0.6842908         2961982
## 10 Mato Grosso do Sul    78  0.6797051         2404631
## # ... with 17 more rows
```

```
pnud_muni %>%
  filter(ano == 2010) %>%
  count(ufn)
```

```
## # A tibble: 27 x 2
##                 ufn     n
##               <chr> <int>
## 1              Acre    22
## 2           Alagoas   102
## 3             Amapá    16
## 4          Amazonas    62
## 5             Bahia   417
## 6             Ceará   184
## 7  Distrito Federal     1
## 8    Espírito Santo    78
## 9             Goiás   246
## 10         Maranhão   217
## # ... with 17 more rows
```

```
pnud_muni %>%
  group_by(ano, ufn) %>%
  tally() %>%
  head # nao precisa de parenteses!
```

```
## Source: local data frame [6 x 3]
## Groups: ano [1]
##
##     ano      ufn     n
##   <dbl>    <chr> <int>
## 1  1991     Acre    22
## 2  1991  Alagoas   102
## 3  1991    Amapá    16
## 4  1991 Amazonas    62
## 5  1991    Bahia   417
## 6  1991    Ceará   184
```

3.8.11 Data Tidying com tidyr

3.8.12 spread

- "Joga" uma variável nas colunas

```
pnud_muni %>%
  group_by(ano, ufn) %>%
  summarise(populacao = sum(popt)) %>%
  ungroup() %>%
  spread(ano, populacao)
```

```
## # A tibble: 27 x 4
##                 ufn     1991     2000     2010
## *             <chr>    <dbl>    <dbl>    <dbl>
## 1              Acre   414609   519639   690774
## 2           Alagoas  2448544  2611271  3045853
## 3             Amapá   280599   453547   652768
## 4          Amazonas  1977073  2543710  3301220
## 5             Bahia 11522516 12286822 13755196
## 6             Ceará  6255097  6995427  8317603
## 7  Distrito Federal  1551869  2001728  2541714
## 8    Espírito Santo  2562362  3048681  3477471
```

```
## 9             Goiás  3931474  4887131  5934769
## 10         Maranhão  4803825  5258529  6317986
## # ... with 17 more rows
```

3.8.13 gather

- "Empilha" o banco de dados

```
pnud_muni %>%
  filter(ano == 2010) %>%
  select(ufn, municipio, starts_with('idhm_')) %>%
  gather(tipo_idh, idh, starts_with('idhm_'))
```

```
## # A tibble: 16,695 x 4
##         ufn             municipio tipo_idh   idh
##       <chr>                 <chr>    <chr> <dbl>
## 1  Rondônia ALTA FLORESTA D'OESTE   idhm_e 0.526
## 2  Rondônia             ARIQUEMES   idhm_e 0.600
## 3  Rondônia                CABIXI   idhm_e 0.559
## 4  Rondônia                CACOAL   idhm_e 0.620
## 5  Rondônia            CEREJEIRAS   idhm_e 0.602
## 6  Rondônia     COLORADO DO OESTE   idhm_e 0.584
## 7  Rondônia             CORUMBIARA   idhm_e 0.473
## 8  Rondônia          COSTA MARQUES   idhm_e 0.493
## 9  Rondônia       ESPIGÃO D'OESTE   idhm_e 0.536
## 10 Rondônia         GUAJARÁ-MIRIM   idhm_e 0.519
## # ... with 16,685 more rows
```

3.8.14 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.

3.8.15 Um pouco mais de manipulação 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`.

3.8.16 Acessando páginas web

3.8.17 Armazenamento de documentos web

3.8.18 Trabalhando com arquivos pdf

3.9 Text mining

3.9.1 Expressões regulares

3.9.2 Trabalhando com datas

Chapter 4

O pacote 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) de 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.

4.1 Instalação

O `ggplot2` não faz parte dos pacotes base do R. Assim, antes de usá-lo, você precisa baixar e instalar o pacote. Para isso, é necessário ter pelo menos a versão 2.8 do R, pois o `ggplot2` não é compatível com versões anteriores.

Para baixar e instalar o pacote, utilize a seguitne linha de código:

```
install.packages("ggplot2")
```

Não se esqueça de carregar o pacote antes de utilizá-lo:

```
library(ggplot2)
```

Chapter 5

Construindo gráficos

A seguir, vamos discutir os aspcetos básicos para a construção de gráficos com o pacote `gglplot2`. Para isso, utilizaremos o banco de dados contido no objeto `mtcars`. Para visualizar as primeiras linhas deste banco, utilize o comando:

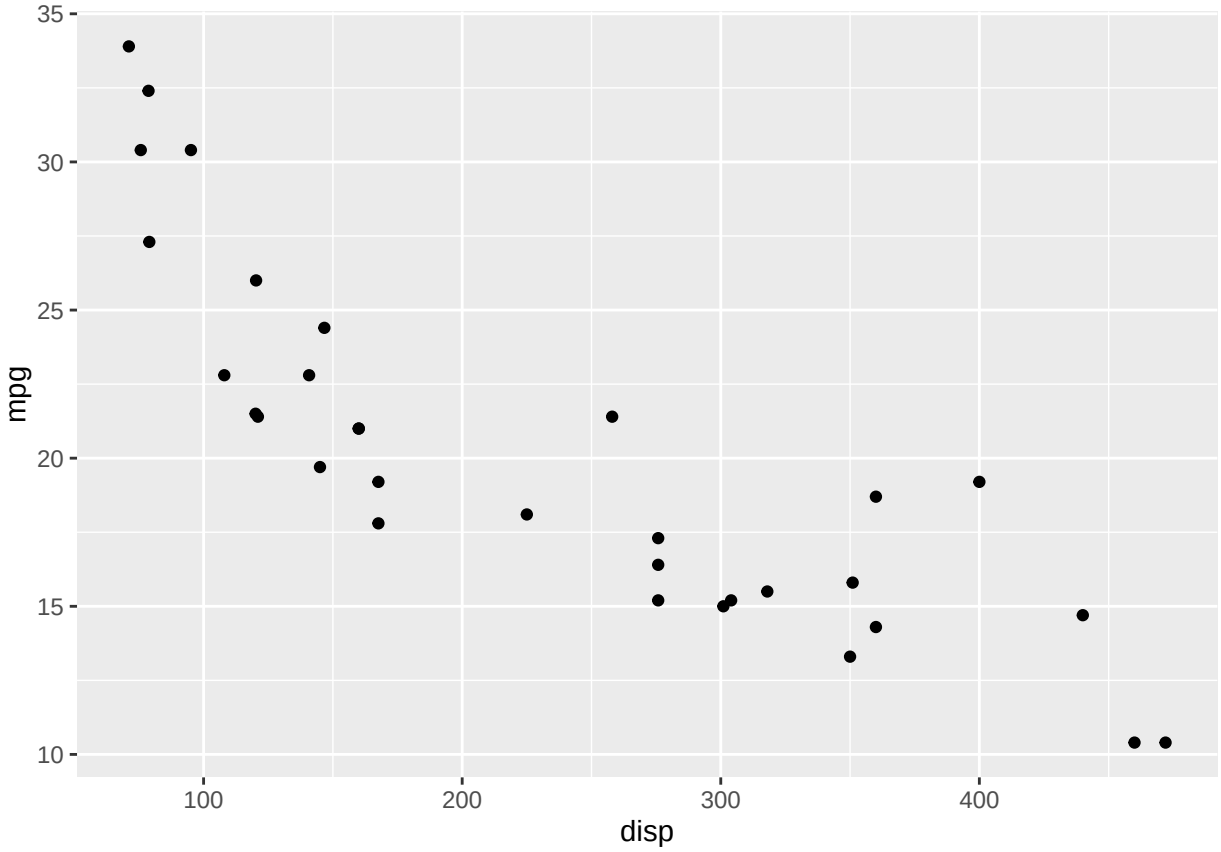
```
head(mtcars)
```

```
##                   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 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.

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



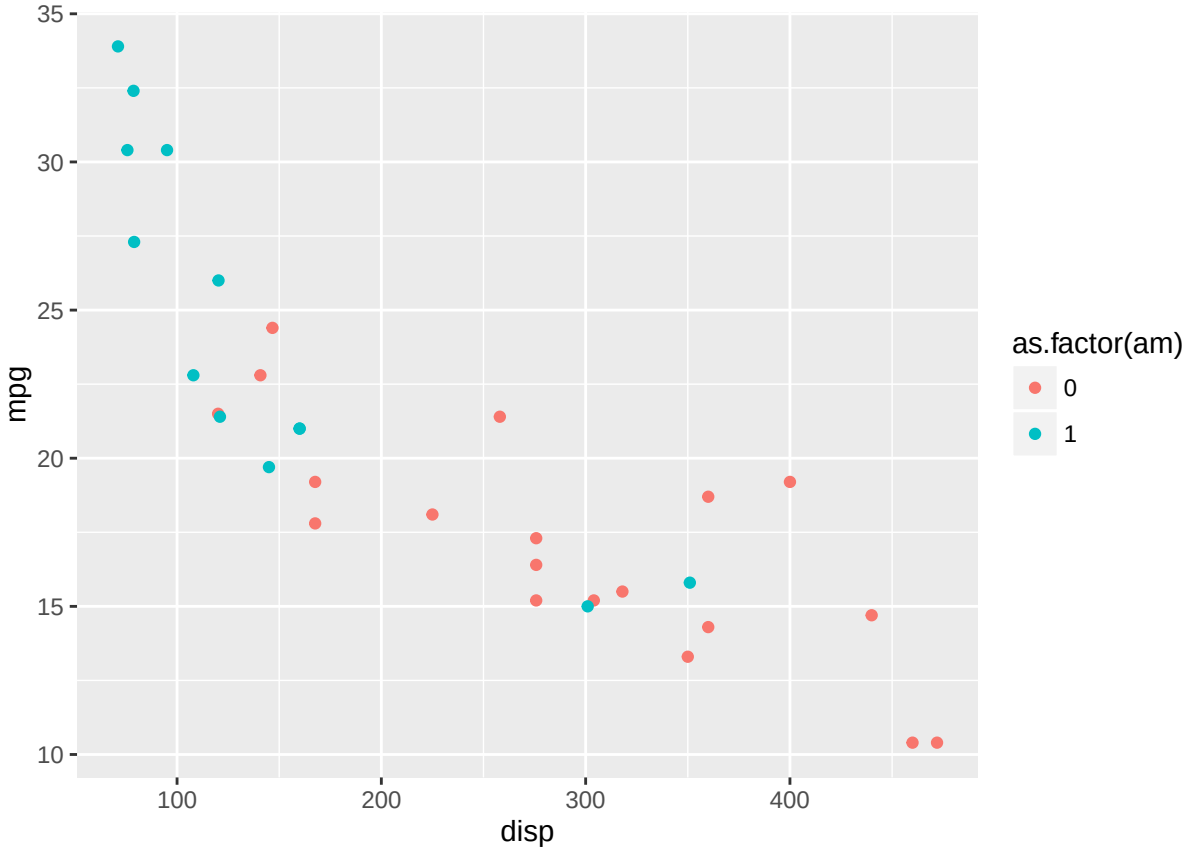
Observe que o primeiro argumento da função `ggplot` é um data frame. A função `aes()` descreve como as variáveis são mapeadas em aspectos visuais de formas geométricas definidas pelos *geoms*. Aqui, essas formas geométricas são pontos, selecionados pela função `geom_point()`, gerando, assim, um gráfico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

5.1.1 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.

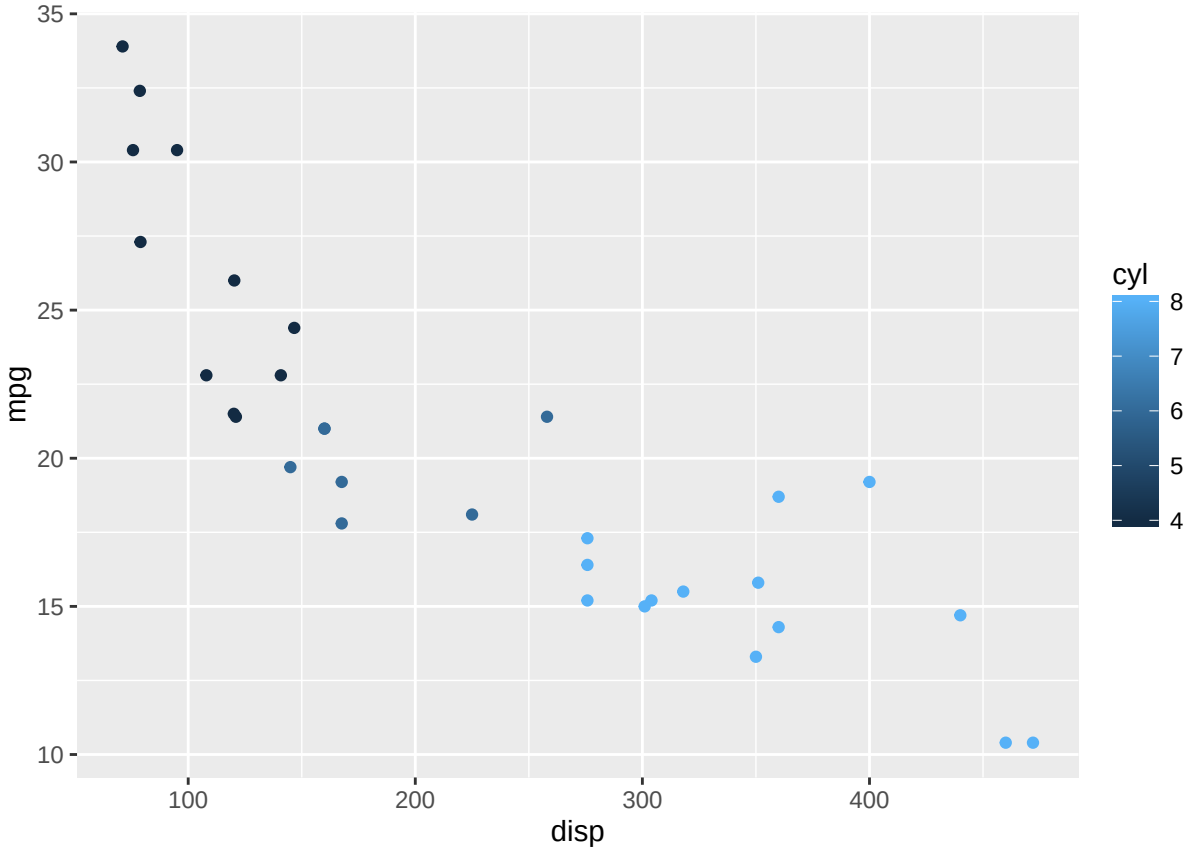
No exemplo acima, atribuímos aspectos de posição: ao eixo y mapeamos a variável `mpg` (milhas por galão) e ao eixo x a variável `disp` (cilindradas). Outro aspecto que pode ser mapeado nesse gráfico é a cor dos pontos

```
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()
```

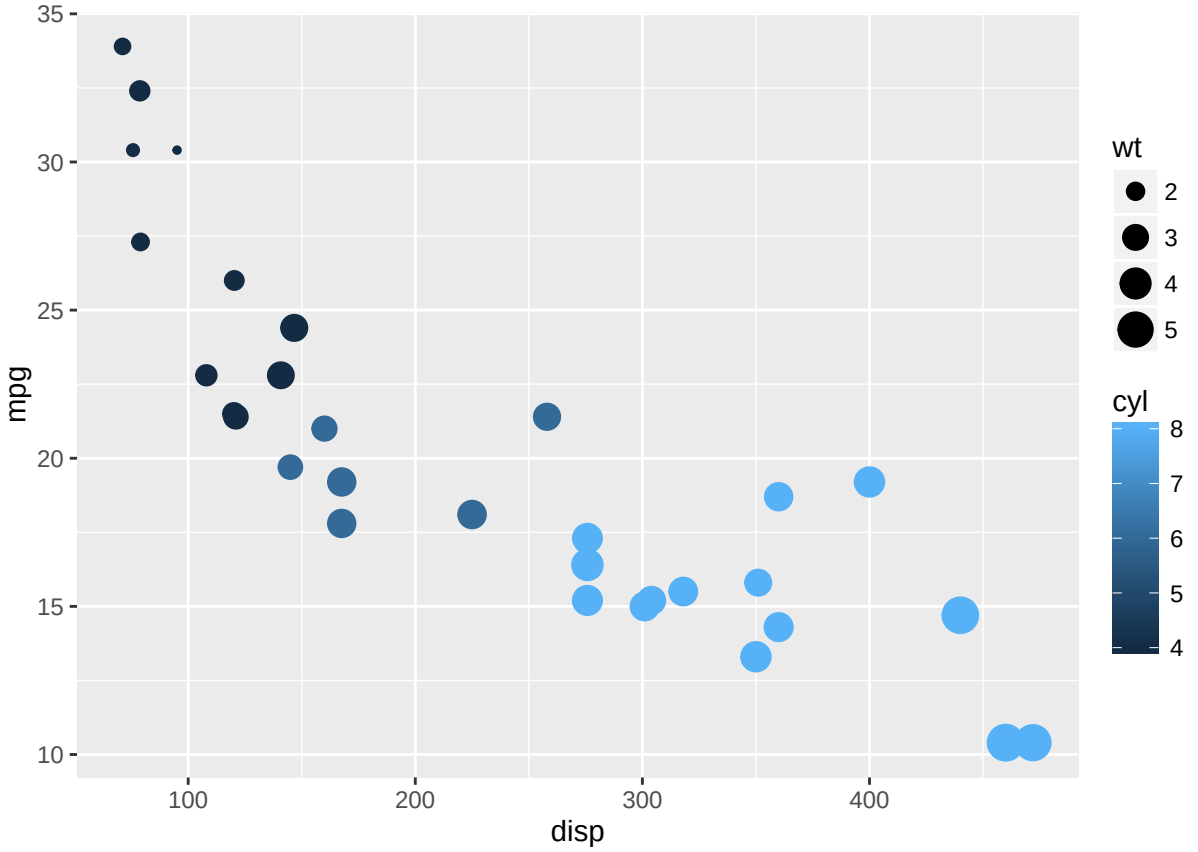


Aqui, o número de cilindros, `cyl`, é representado pela tonalidade da cor azul.

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.

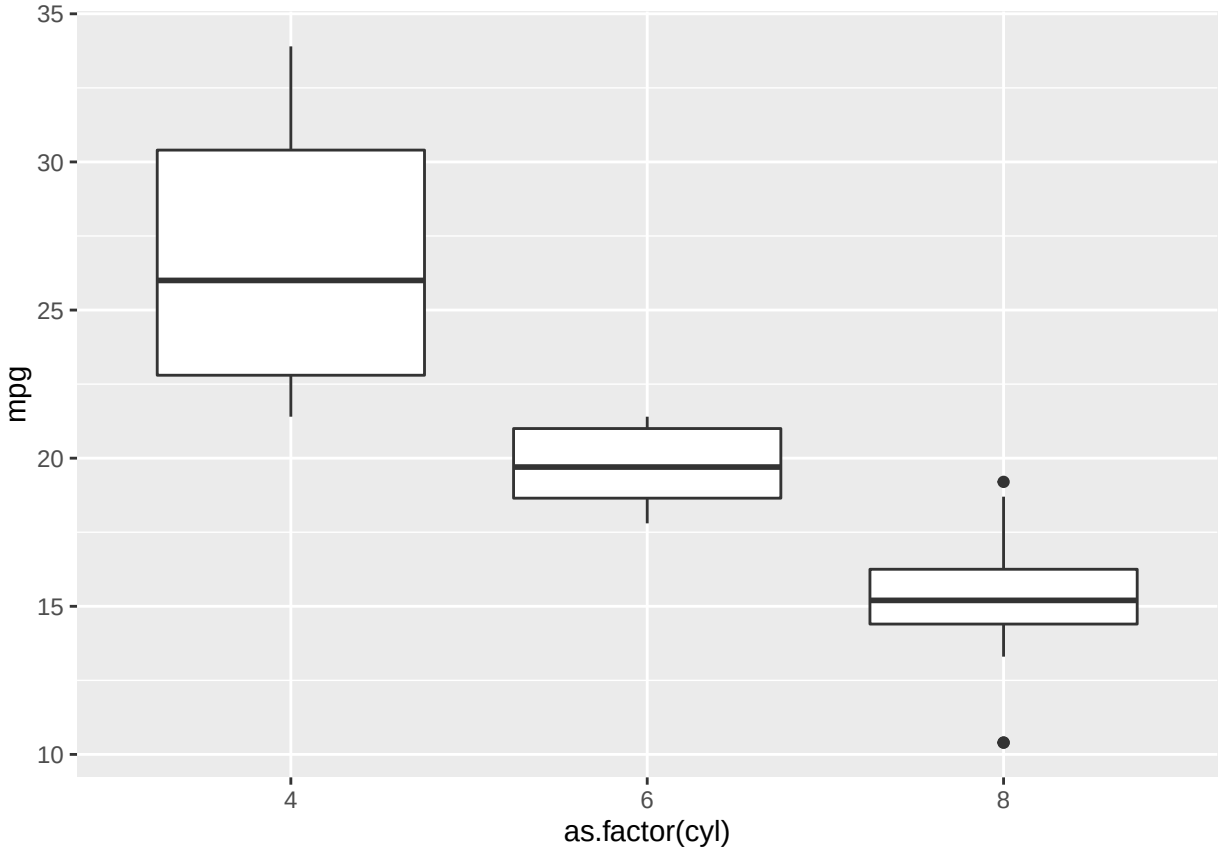
5.1.2 Geoms

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

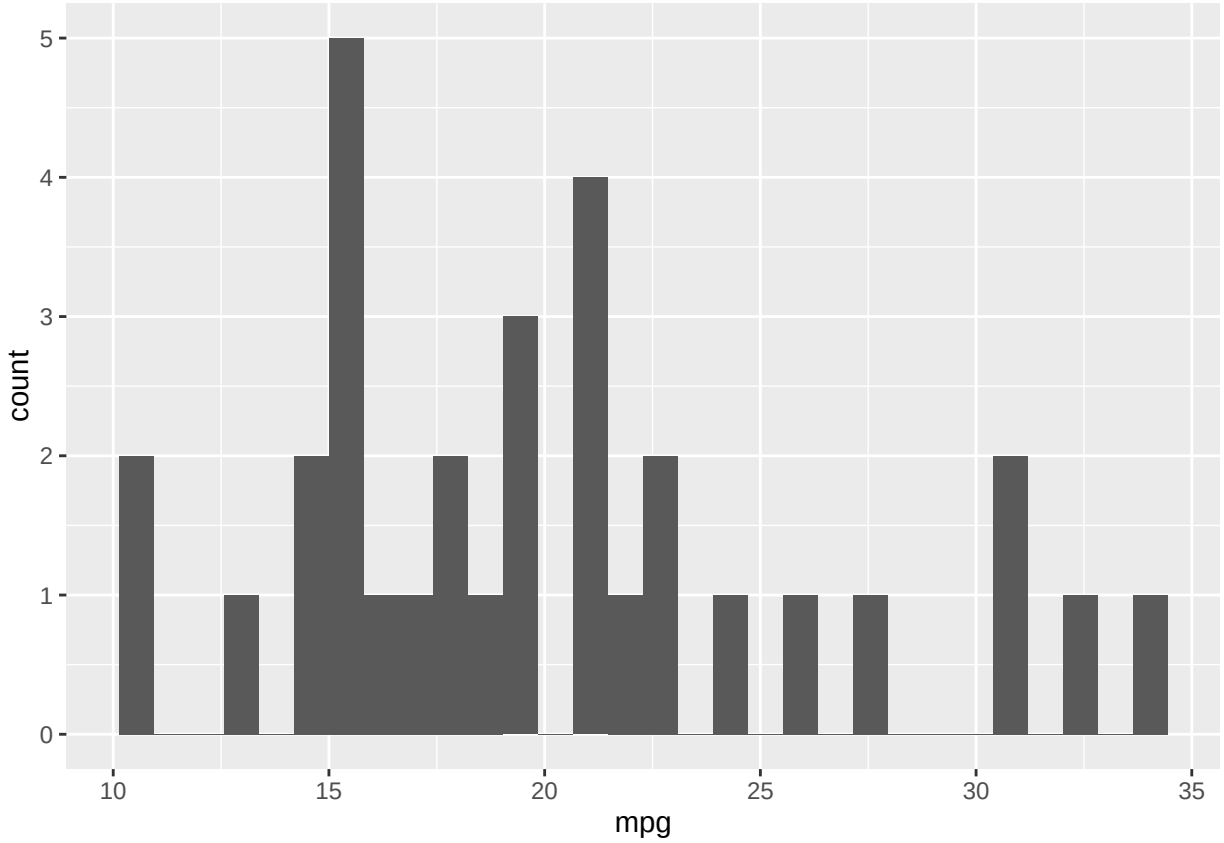
Veja a seguir como é fácil gerar diversos gráficos diferentes utilizando a mesma estrutura do gráfico de dispersão acima:

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

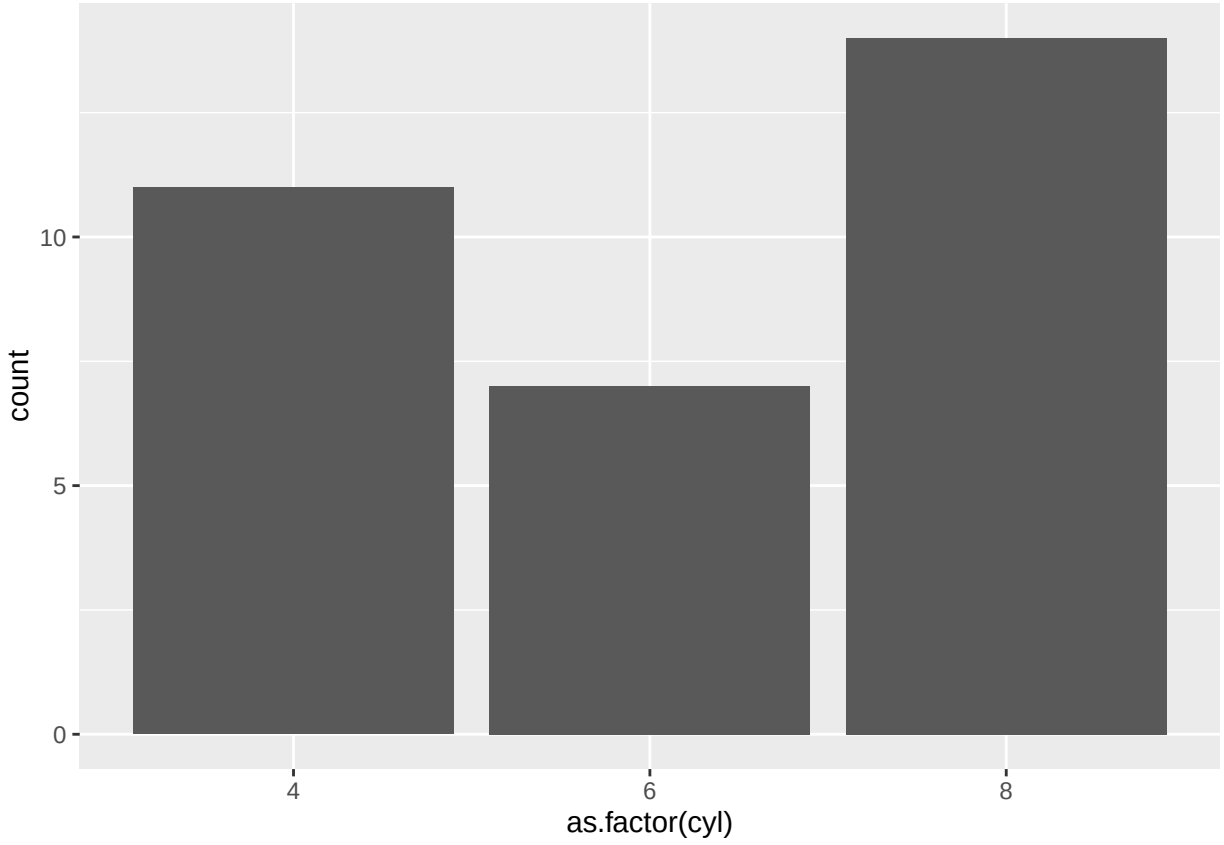



```
ggplot(mtcars, aes(x = mpg)) +
  geom_histogram()
```

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




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



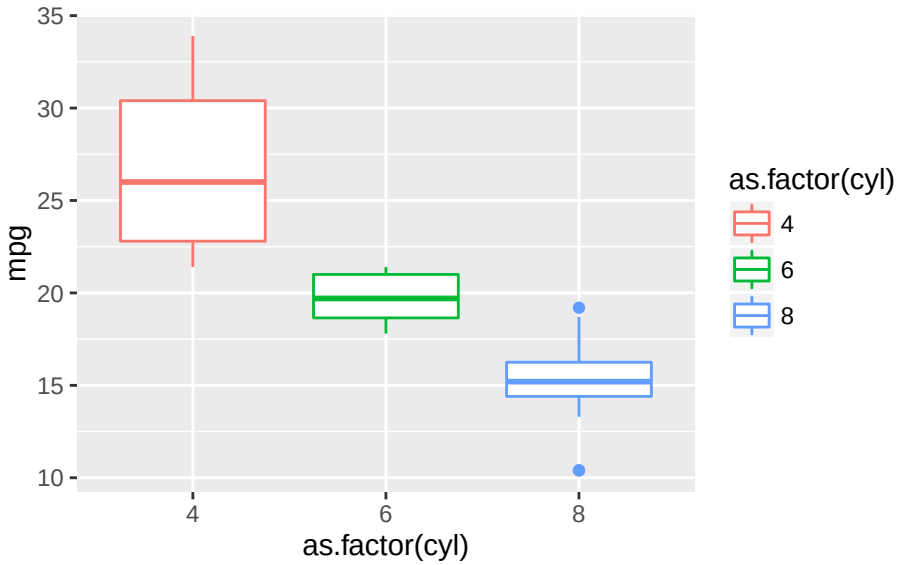
Para fazer um boxplot para cada grupo, precisamos passar para o aspecto x do gráfico uma variável do tipo fator.

5.2 Personalizando os gráficos

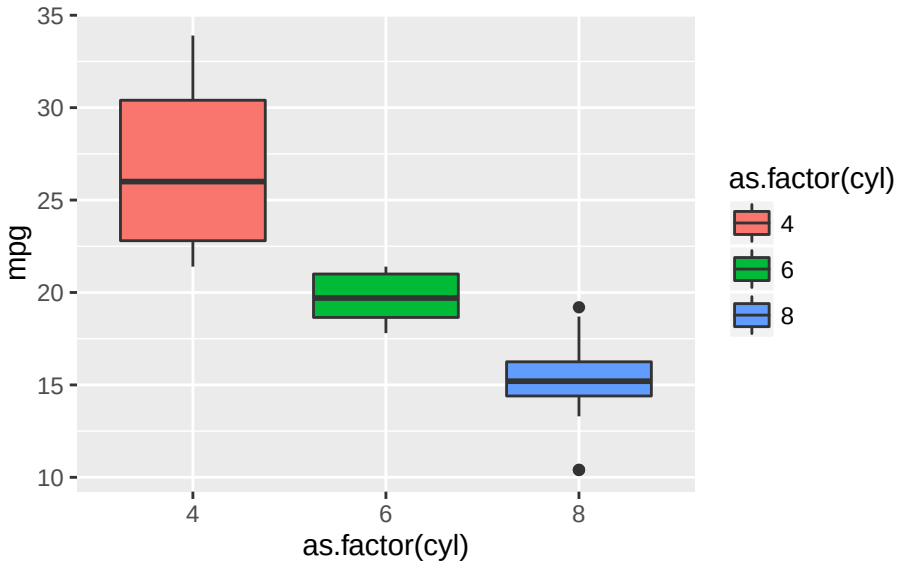
5.2.1 Cores

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()
```

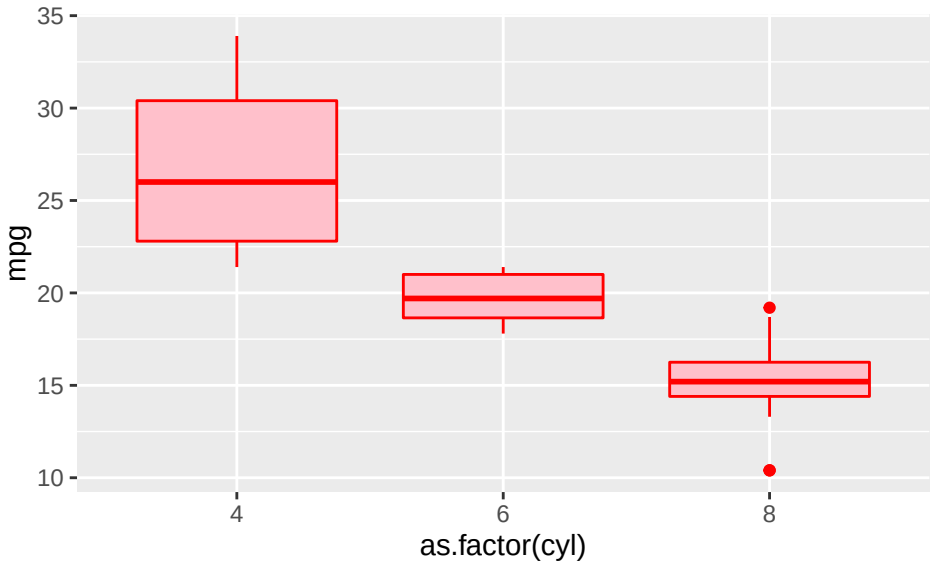



```
ggplot(mtcars, aes(x = as.factor(cyl), y = mpg, fill = as.factor(cyl))) + geom_boxplot()
```



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")
```

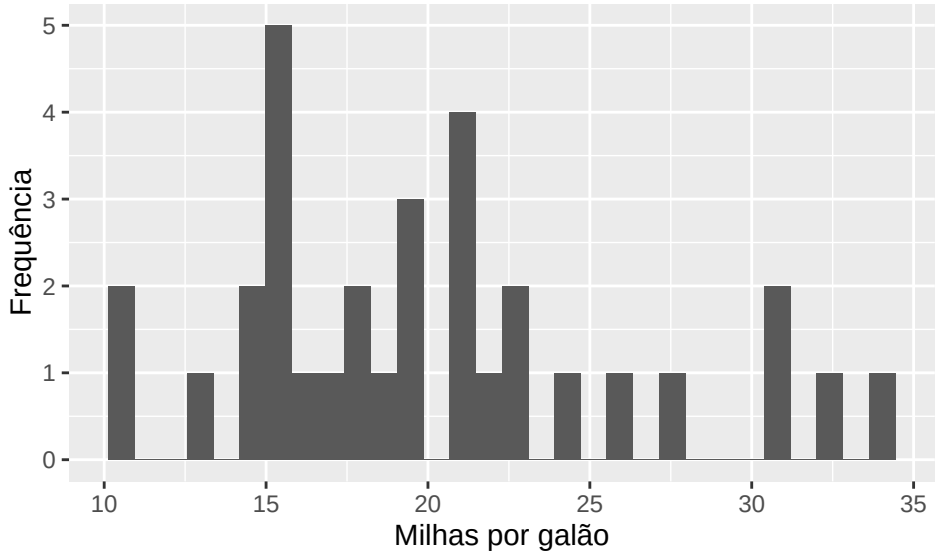


5.2.2 Eixos

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`.
```

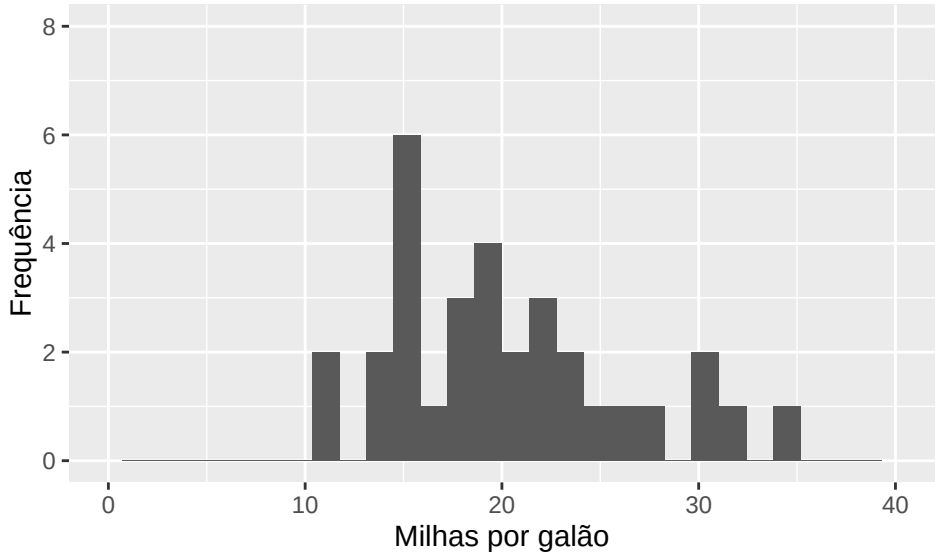


Para alterar os limites dos gráficos usamos as funções `xlim()` e `ylim()`.

```
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`.
```

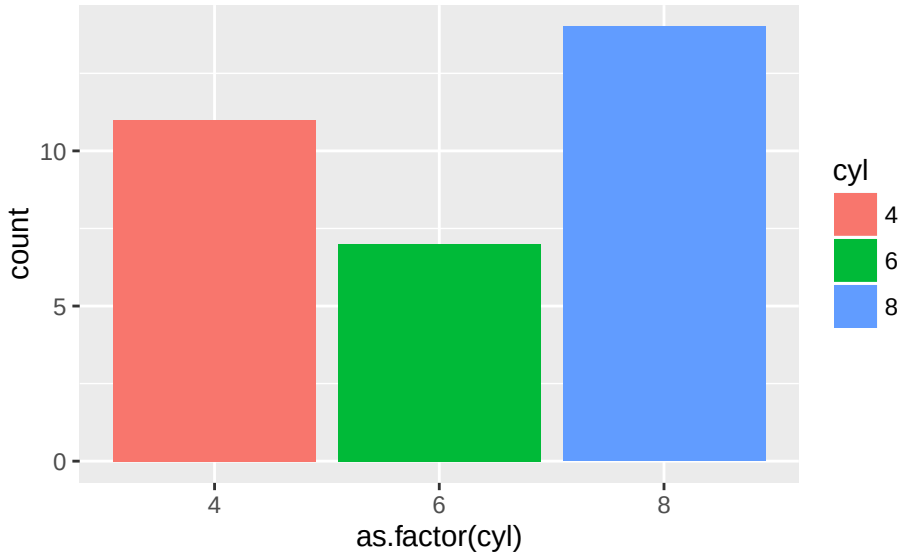


5.2.3 Legendas

A legenda de um gráfico pode ser facilmente personalizada.

Para trocar o *label* da leganda:

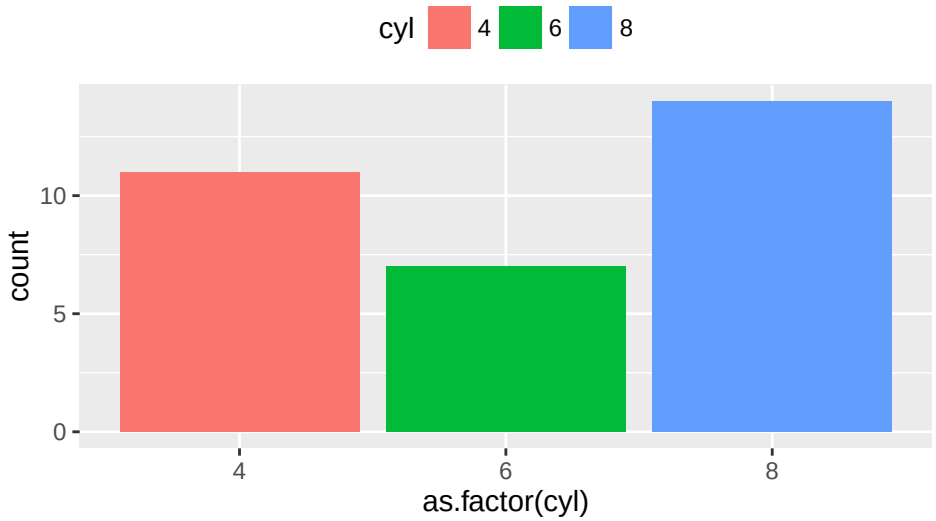
```
ggplot(mtcars, aes(x = as.factor(cyl), fill = as.factor(cyl))) +
  geom_bar() +
  labs(fill = "cyl")
```



Para trocar a posição da legenda:

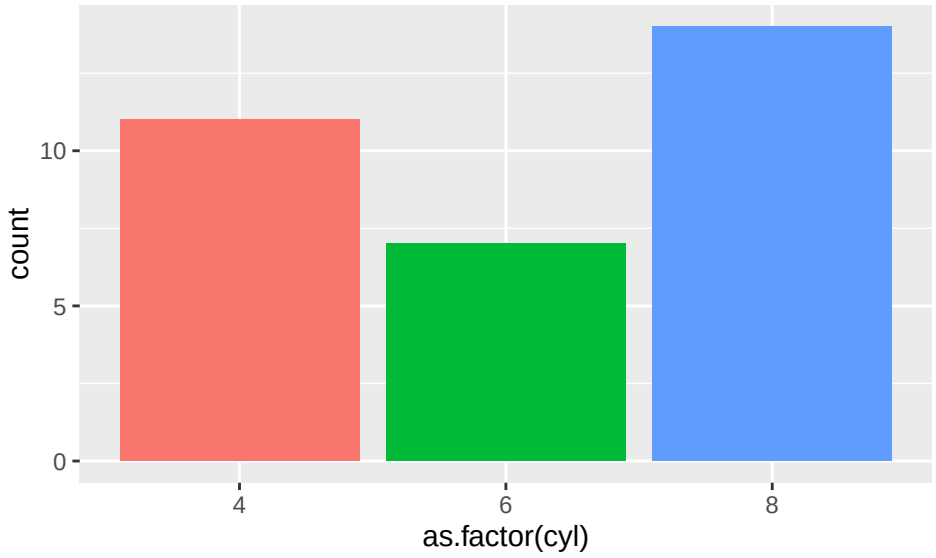
```
ggplot(mtcars, aes(x = as.factor(cyl), fill = as.factor(cyl))) +
  geom_bar() +
```

```
  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)
```

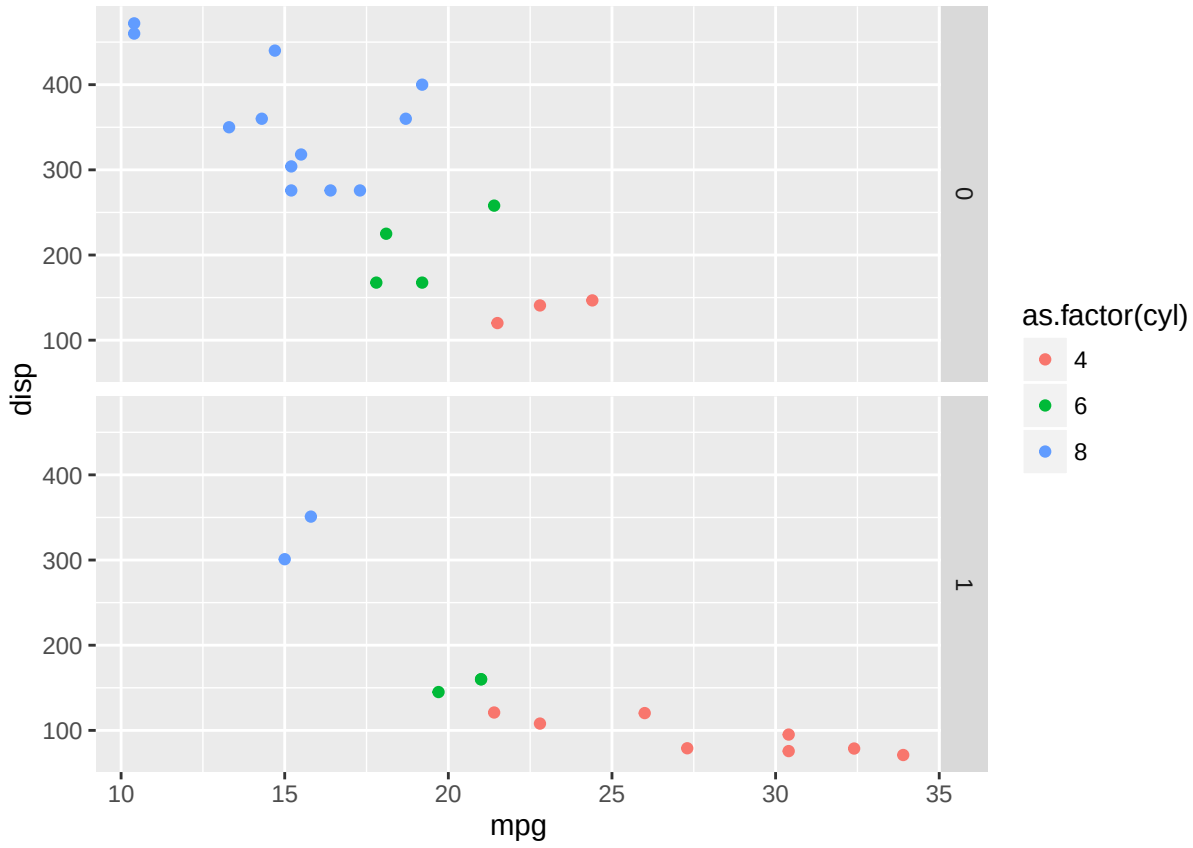


Veja mais opções de personalização aqui!

5.2.4 Facets

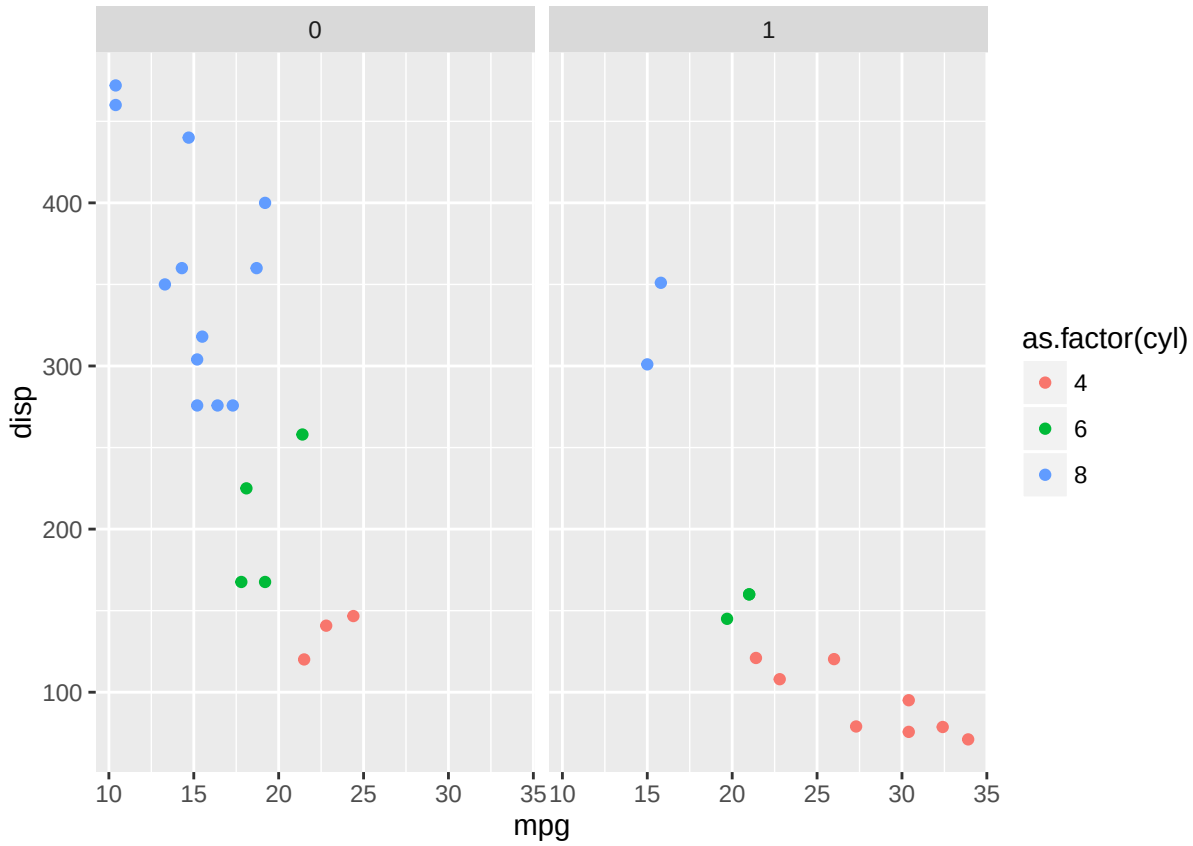
Outra funcionalidade muito importante do ggplot é o uso de *facets*.

```
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)
```


0
1
disp
400
300
200
100
10
15
20
25
30
35
mpg
as.factor(cyl)
4
6
8

Chapter 6

Applications

Some *significant* applications are demonstrated in this chapter.

6.1 Example one

6.2 Example two

Chapter 7

Final Words

We have finished a nice book.

Bibliography

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2016). *bookdown: Authoring Books with R Markdown*. R package version 0.1.3.