Curso de Jurimetria

Associação Brasileira de Jurimetria

2016-08-04

Contents

Chapter 1

Introdução

Essa é a primeira iteração do curso de Jurimetria da ABJ. Nesse curso abordamos aspectos teóricos e práticos da Jurimetria, essenciais para um profissional da Estatística que tenha interesse em trabalhar nessa área de pesquisa.

O curso é voltado para estatísticos e está organizado em duas partes principais. Na primeira parte, apresentamos as ferramentas de trabalho do laboratório de Jurimetria, que permitem a execução de nossas pesquisas. Na segunda parte, apresentamos diversas aplicações da Jurimetria, focando em metodolodia de pesquisa, técnicas estatísticas e conceitos fundamentais de Jurimetria.

Para a primeira parte do curso, será necessário ter conhecimentos do software estatístico R, como a lógica de programação, sintaxe do R e ambientação com o $\mathrm{RStudio}$. Para as partes seguintes, será necessário ter conhecimentos básicos de estatística, como probabilidade, verossimilhança, inferência e processos estocásticos.

1.1 Definição de Jurimetria

Podemos definir Jurimetria como a *disciplina do conhecimento que utiliza a metodologia estatística para investigar o funcionamento de uma ordem jurídica*. A partir dela, fica claro que a Jurimetria se distingue das demais disciplinas jurídicas, tanto pelo objeto como pela metodologia empregada nas análises.

A Jurimetria tem três pilares operacionais: jurídico, estatístico e computacional. É claro que a reunião dessas especialidades em uma só pessoa é atualmente rara. Por isso, a prática da Jurimetria vem sendo desenvolvida em um ambiente laboratorial em que bacharéis em direito, estatísticos e cientistas da computação unem esforços na resolução de problemas.

A Jurimetria propõe um giro epistemológico, análogo àquele proposto pelo *realismo jurídico*, deslocando o

centro de interesse da pesquisa do plano abstrato para o plano concreto. O conceito norteador deste giro é que o direito efetivo, aquele capaz de afetar a relação entre sujeitos, correspondente às sentenças, acórdãos, contratos e demais ordens jurídicas produzidas no plano concreto.

A lei é uma declaração de intenções do legislador, que muitas vezes se mostra plurívoca, contraditória e lacunosa. Para a jurimetria, é no plano concreto que o Direito se revela, sendo a lei apenas um dos fatores - ao lado dos valores pessoais, religião, empatia, experiência pessoal de vida e outros tantos -, capaz de influenciar o processo de concretização das normas do Direito. Por tal razão, o Direito não pode ser reduzido a um conjunto de normas editado por autoridades competentes e deve ser visto, sim, como um aparato de solução de conflitos, no qual a lei desempenha um papel importante, porém não suficiente.

Estudar de forma concreta significa **situar o objeto de estudo no tempo e no espaço**. Uma vantagem desse tipo de abordagem é que há clareza na construção do escopo das pesquisas, o que pode levar a conclusões mais diretas. Para tornar os estudos factíveis, no entanto, também é necessário definir claramente o arcabouço de suposições no qual as conclusões são válidas.

Utilizar a abordagem concreta não significa que pensar de forma abstrata é ruim. Para construção de modelos capazes de mensurar certas quantidades de forma adequada, ou para elaborar soluções para um problema na lei, é necessário desconstruir o fenômeno de maneira ampla, utilizar criatividade e intuição. A concretude ajuda ao direcionar as abstrações e ligá-las a um objetivo pragmático.

A jurimetria também assume como ponto de partida a possibilidade modelar certos fenômenos do direito como eventos aleatórios. Essa abordagem contrasta com a abordagem clássica, que usa o determinismo fatalista como realidade do direito. As vantagens em assumir aleatoriedade em modelos jurimétricos confundem-se com as vantagens da utilização da metodologia estatística em geral. Ao aceitar que há incerteza nos modelos e que não somos em geral capazes de dar respostas definitivas às nossas perguntas, ganhamos a habilidade de afirmar mais sobre o desconhecido, associando possíveis resultados a suas respectivas verossimilhanças.

Neste curso, abordaremos aplicações dentro dos tribunais, na administração e para a sociedade. Após estudar este material, o estatístico terá base suficiente para planejar e executar estudos jurimétricos, que poderão auxiliar na elaboração de políticas públicas, na gestão judiciária e administração de tribunais.

Chapter 2

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 lapidadas para obtenção de informação útil.

O jurimetrista trabalha com dados sujos e desorganizados, mas gera muito valor ao extrair suas informações. Por isso, o profissional precisa dominar o ferramental
de extração, transformação e visualização de dados, e é sobre isso que discutiremos nesta primeira parte do curso. Utilizaremos como base o software estatístico R,

Os pacotes utilizados são httr, xml2, rvest, dplyr, tidyr, purrr, lubridate, stringr e 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*[1]. É possível instalar todos esses pacotes de uma vez rodando

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

[1] Baixe aqui.

2.1 Exemplos trabalhados

2.1.1 Câmaras de gás

Uma das principais questões que surgem quando o tema é impunidade e que motivou esse trabalho 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 levantamento 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 dezesseis Câmaras de Direito Criminal do Estado de São Paulo, e nas quatro Câmaras Extraordinárias. Todas as informações foram obtidas através de ferramentas computacionais a partir de bases de dados disponíveis 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.

2.1.2 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 com 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.

2.2 Pipe

O operador *pipe* foi uma das grandes revoluções recentes do R, tornando a leitura de códigos muito mais lógica, fácil e compreensível. Este operador foi introduzido por Stefan Milton Bache no pacote $\mathrm{magrittr}$ 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 devemos instalar o pacote $\mathrm{magrittr}$

```
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 uma dentro das outras, 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"), "farinha", até = "maci
```

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 se parece com uma receita de bolo.

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

2.3 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 $\mathrm{data\_frame}$ separado (possivelmente com chaves de associação).

Nessa parte do curso, vamos trabalhar com *arrumação de dados*, que consiste em trabalhar com ferramentas de extração, consolidação e transformação de dados. As ferramentas utilizadas fazem parte do chamado $\mathrm{tidyverse}$, um universo de pacotes contemporâneos do R que são intuitivas, eficientes e úteis.

O objetivo em *arrumação de dados* 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.

2.4 Pacotes dplyr e tidyr

(falta adicionar exemplos)

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.

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.

2.4.1 Trabalhando com tibbles

Vamos assumir que temos a seguinte base de dados na forma tidy.

```
4
```

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

2.4.2 As cinco funções principais do dplyr

- filter
- mutate
- select
- arrange
- summarise

2.4.3 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 %>%

2.4.4 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.

2.4.5 select

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

2.4.6 filter

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

2.4.7 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.

2.4.8 arrange

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

2.4.9 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().

2.4.10 spread

- "Joga" uma variável nas colunas

2.4.11 gather

- "Empilha" o banco de dados

2.4.12 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.

2.4.13 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.

2.5 Pacotes lubridate e stringr

...intro...

Agora, vamos assumir que temos a base de dados em formato tidy, mas ao invés do resultado do acórdão, temos os textos das decisões.

2.6 Pacotes httr, xml2 e rvest

(...)

2.7 Web scraping

Esta seção 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 jurimetrista a desenvolver programas que sejam fáceis de adaptar no tempo.

É importante ressaltar 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 do TJSP.

2.7.1 Informações iniciais

Antes de iniciar um programa de web scraping, 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 os responsáveis pela manutenção do site se eles 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*. Quando já sabemos de antemão quais são as URLs que vamos acessar, a etapa de busca é desnecessária.

Na maior parte dos casos, deixar os algoritmos de *coleta* e *processamento* 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. Por outro lado, em alguns casos isso pode tornar o código mais ineficiente e os arquivos obtidos podem ficar pesados.

2.7.2 Diferença entre buscar, baixar e processar.

Buscar 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*. Usualmente separamos a estruturação em duas etapas: 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 HTML será realizada pelas funções $\mathrm{parse\_cjsg}$ e $\mathrm{parse\_cpopg}$.

Na pesquisa das câmaras, seguimos o fluxo

$\mathrm{buscar} \rightarrow \mathrm{coletar} \rightarrow \mathrm{processar} \rightarrow \mathrm{coletar} \rightarrow \mathrm{processar}$

para conseguir nossos dados.

2.8 Buscar documentos

A tarefa de listar os documentos de interesse é realizada acessando resultados de um formulário. Dependendo do site, 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 TJSP temos *uma busca e muitas paginações*. Acesse a página do e-SAJ, digite "acordam" no campo "Pesquisa Livre" e clique em "Pesquisar", para ter uma ideia de como é essa página.

A página (acessada no dia 2016-08-04) é uma ferramenta de busca com vários campos, que não permite pesquisa com dados em branco. Na parte de baixo o site mostra uma série de documentos, organizados em páginas de vinte em vinte resultados.

Para realizar a coleta, precisamos de duas funções principais, uma que faz a busca e outra que acessa uma página específica (que será executada várias vezes). Utilizaremos as funções cjsg e cjsg_pag.

```
cjsg <- function(parms = cjsg_parms(), path = './cjsg',
                 min_pag = 1, max_pag = 10, overwrite = FALSE,
                 verbose = TRUE, p = .05) {
  suppressWarnings(dir.create(path))
  if (!file.exists(path)) stop(sprintf('Pasta não "%s" pôde ser criada', path))
  conf <- httr::config(ssl_verifypeer = FALSE)
  r0 <- httr::POST('http://esaj.tjsp.jus.br/cjsg/resultadoCompleta.do',
                   body = parms, config = conf)
  if (is.na(max_pag) || is.infinite(max_pag)) n_pags <- cjsg_n_pags(r0)
  abjutils::dvec(cjsg_pag, 1:n_pags, path = path, overwrite = overwrite)
}
```

```
cjsg_pag <- function(pag, path, ow) {
  u <- 'http://esaj.tjsp.jus.br/cjsg/trocaDePagina.do?tipoDeDecisao=A&pagina=%d'
  u_pag <- sprintf(u, pag)
  arq <- sprintf('%s/%05d.html', path, pag)
  if (!file.exists(pag) || ow) {
    httr::GET(sprintf(u, pag), httr::write_disk(arq, overwrite = ow))
    tibble::data_frame(result = 'OK')
  } else {
    tibble::data_frame(result = 'já existe')
  }
}
```

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.

É possível visualizar a página baixada com a função BROWSE do pacote httr.

```
arqs <- dir('data-raw/cjsg', full.names = TRUE)
httr::BROWSE(arqs[1])
```

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

Note que criamos uma função que facilita 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".

```
cjsg_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
}
```

Também foi necessário realizar um pequeno processamento na primeira requisição, quando o usuário não souber a priori quantas páginas deseja baixar. Nesse caso, a função cjsg_npags identifica o número de paginações necessárias.

```
cjsg_npags <- function(req) {
  val <- xml2::read_html(httr::content(req, '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
}
```

A função dvec é uma função genérica que ajuda a aplicar uma função a cada elemento de determinados itens, como um lapply, mas que o faz de forma mais verborrágica e não resulta em erro caso um elemento dê erro.

```
#' Vetorizando scrapers
#'
#' Vetoriza um scraper (função) para um vetor de itens
#'
#' @param fun função a ser aplicada em cada arquivo.
#' @param itens character vector dos caminhos de arquivos a serem transformados.
#' @param ... outros parâmetros a serem passados para \code{fun}
#' @param verbose se \code{TRUE} (default), mostra o item com probabilidade p.
#' @param p probabilidade de imprimir mensagem.
#'
#' @export
```

```
dvec <- function(fun, itens, ..., verbose = TRUE, p = .05) {
  f <- dplyr::failwith(data_frame(result = 'erro'), fun)
  tibble::data_frame(item = itens) %>%
    dplyr::distinct(item) %>%
    dplyr::group_by(item) %>%
    dplyr::do({
      if (runif(1) < p && verbose) print(.$item)
      d <- f(.$item, ...)
      if (tibble::has_name(d, 'result')) d$result <- 'OK'
      d
    }) %>%
    dplyr::ungroup()
}
```

No trabalho das câmaras, utilizamos a seguinte chamada da função:

```
d_result <- cjsg_parms() %>%
  cjsg(path = 'data-raw/cjsg')
```

Onde guardar os dados? Ao construir um scraper, é importante guardar os dados brutos na máquina ou num servidor, para reprodutibilidade e manutenção do scraper. Se estiver construindo um pacote do R, o melhor lugar para guardar esses dados é na pasta data-raw, como sugerido no livro r-pkgs. Se os dados forem muito volumosos, pode ser necessário colocar esses documentos numa pasta externa ao pacote. Para garantir a reprodutibilidade, recomendamos a criação de um pacote no R cujo objetivo é somente baixar e processar esses dados, além da criação de 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.

2.9 Coletar processos

(...)

2.10 Visualização de dados

(ainda nada feito)

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.

2.10.1 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)
```

2.11 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:

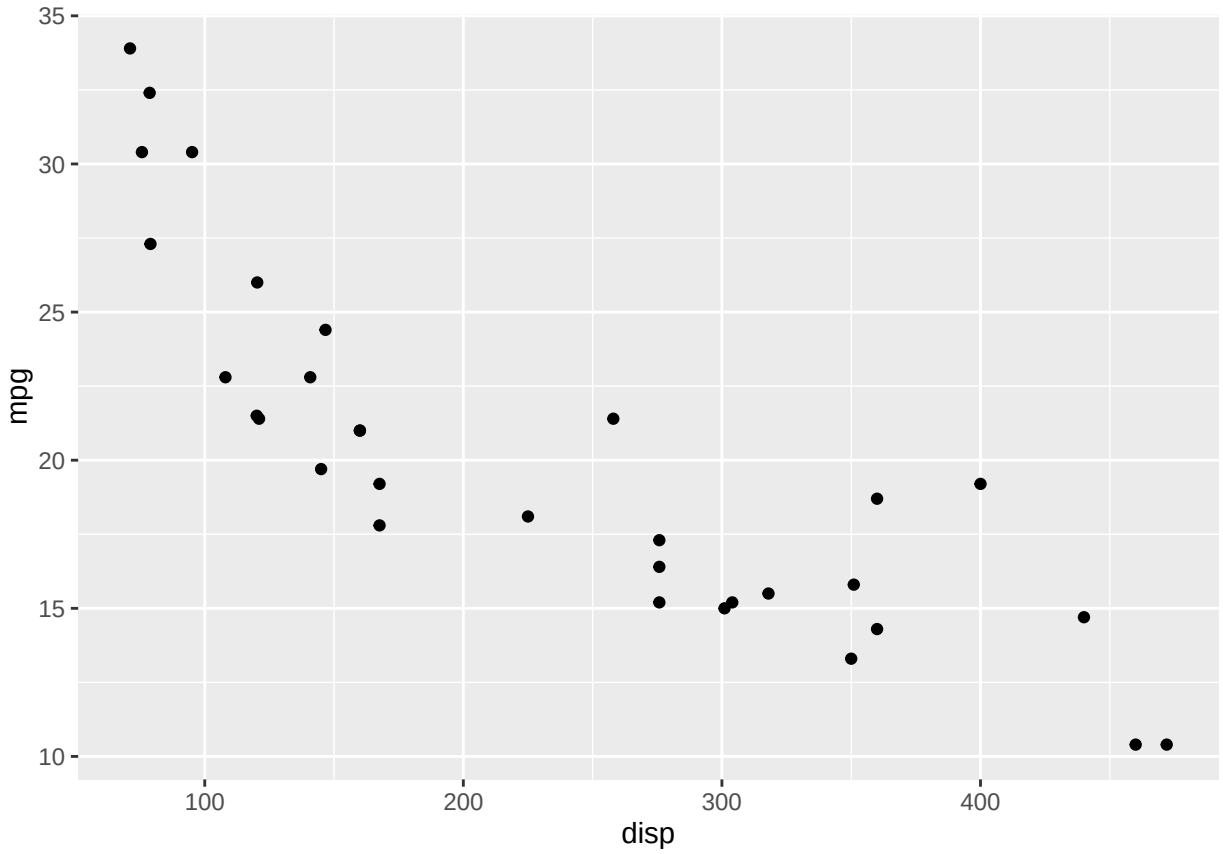
```
head(mtcars)
```

```
##                   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
```

2.11.1 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.

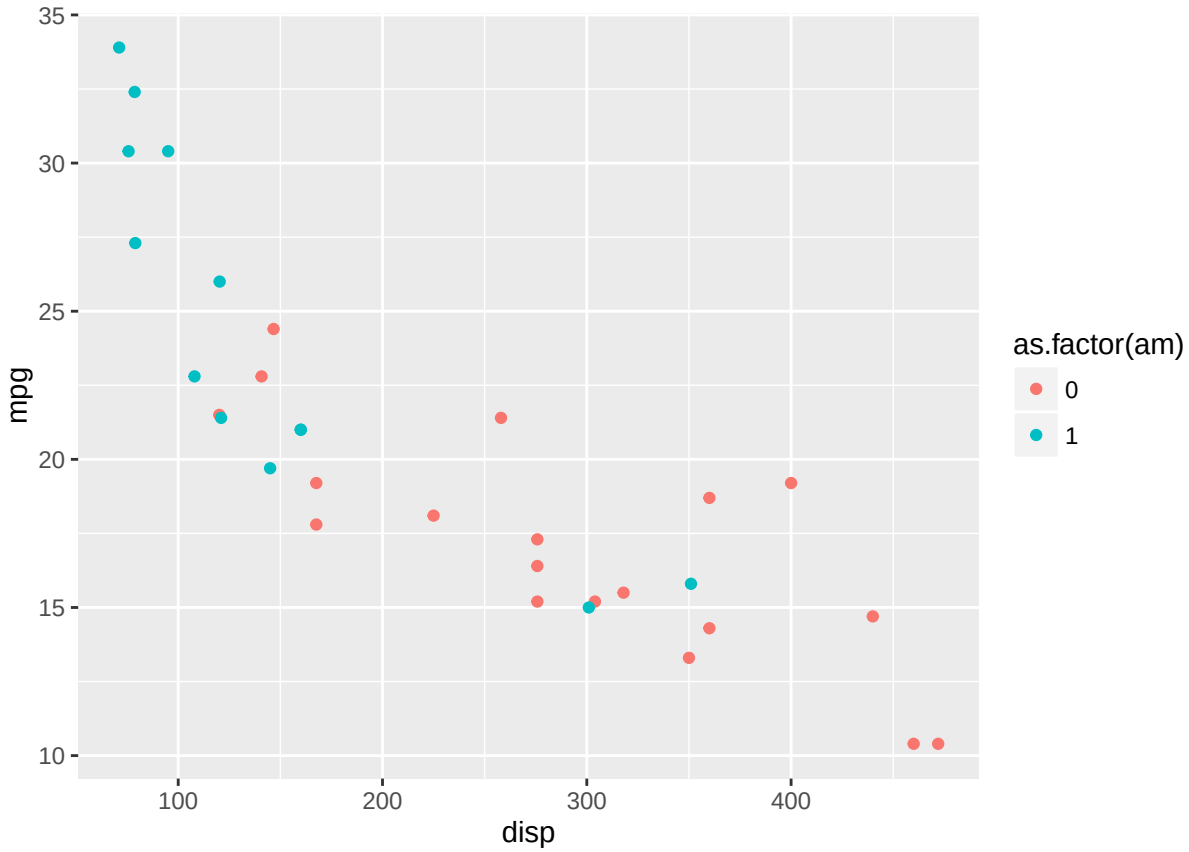
2.11.1.1 Aesthetics

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.

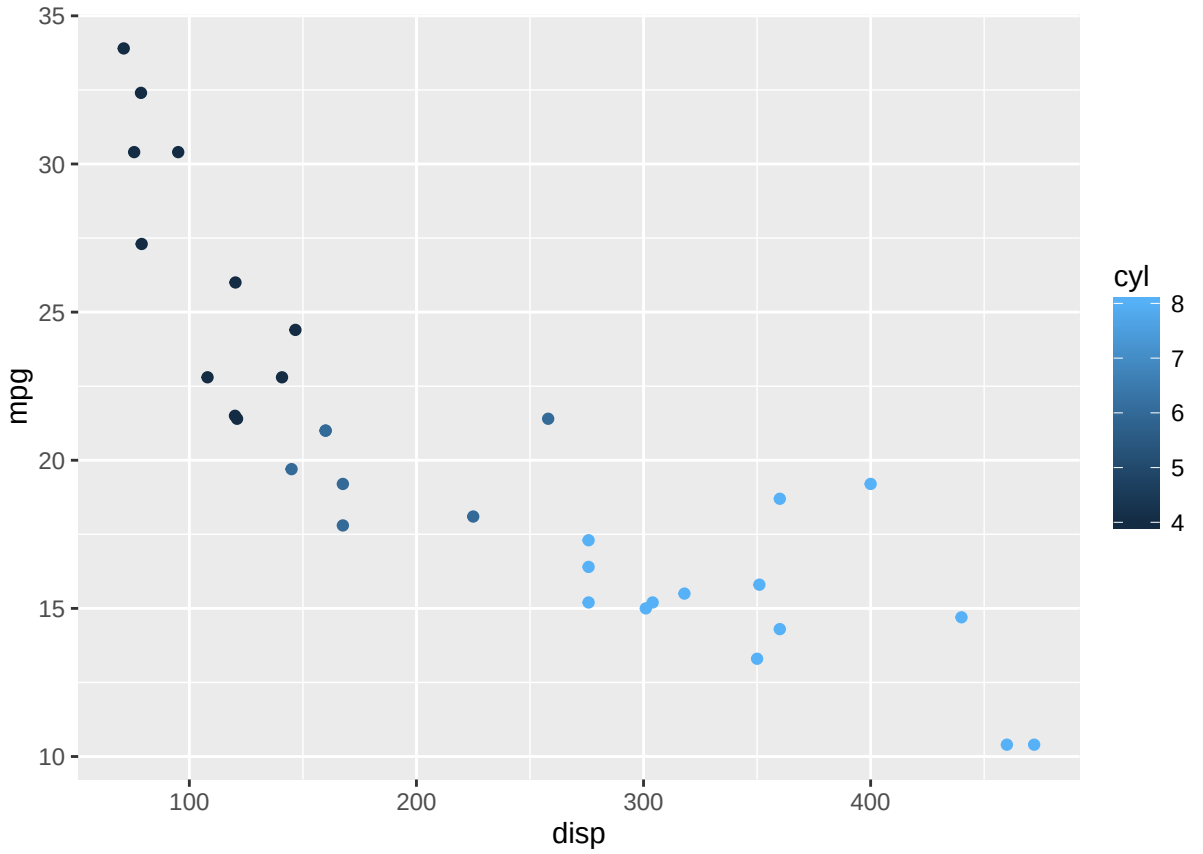
No exemplo acima, atribuímos aspectos de posição: ao eixo y mapeamos a variável mpg (milhas por galão) e ao eixo x a variável disp (cilindradas). Outro aspecto que pode ser mapeado nesse gráfico é a cor dos pontos

```
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()
```

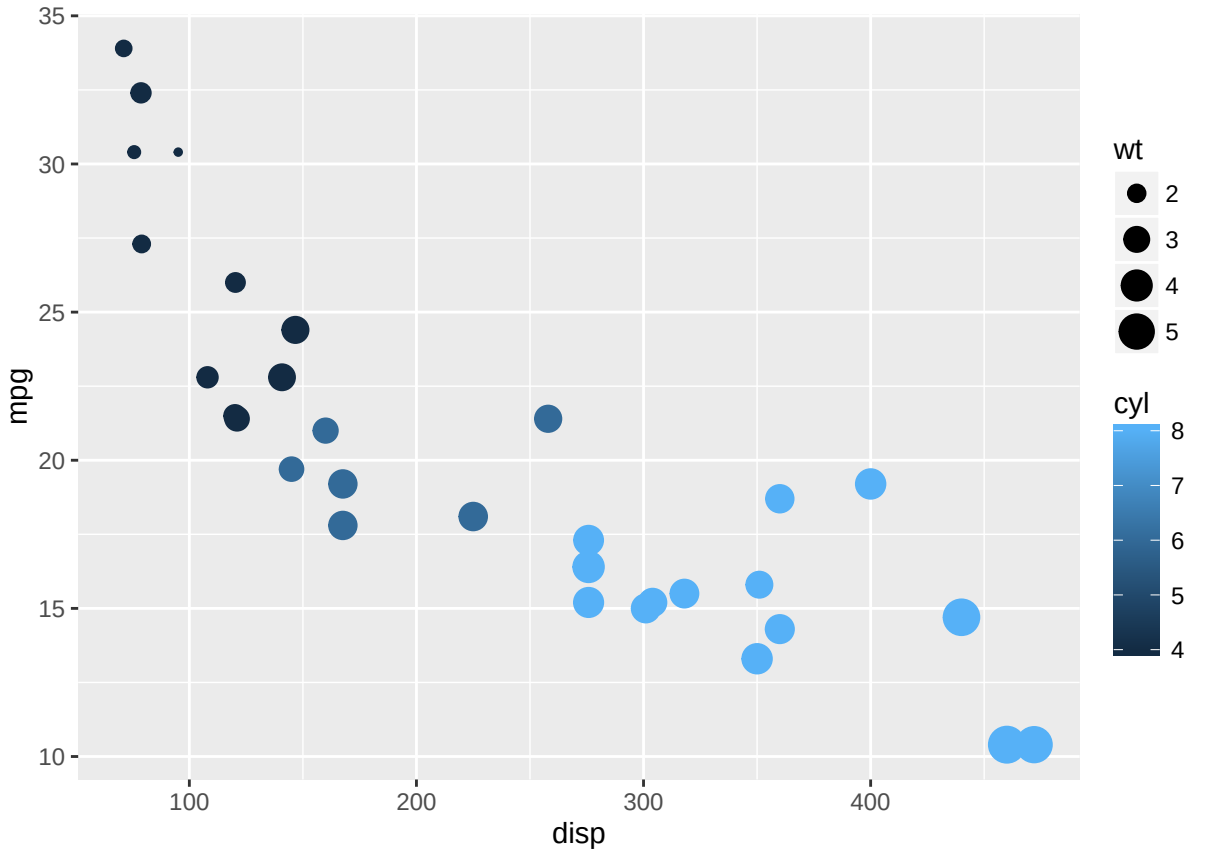


Aqui, o número de cilindros, cyl, é representado pela tonalidade da cor azul.

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.

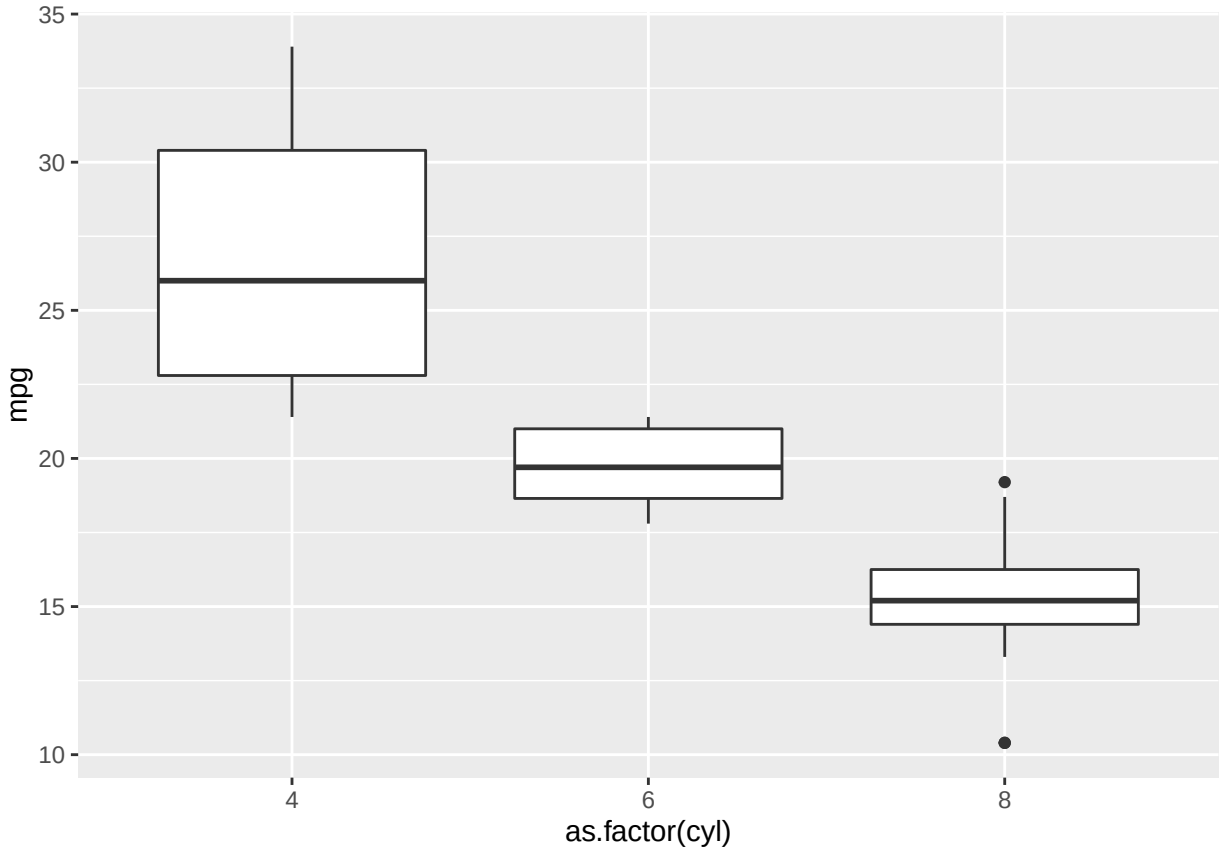
2.11.1.2 Geoms

Os *geoms* definem qual forma geométrica será utilizada para a visualização dos dados no gráfico. Como já vimos, a função $\mathrm{geom\_point()}$ 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

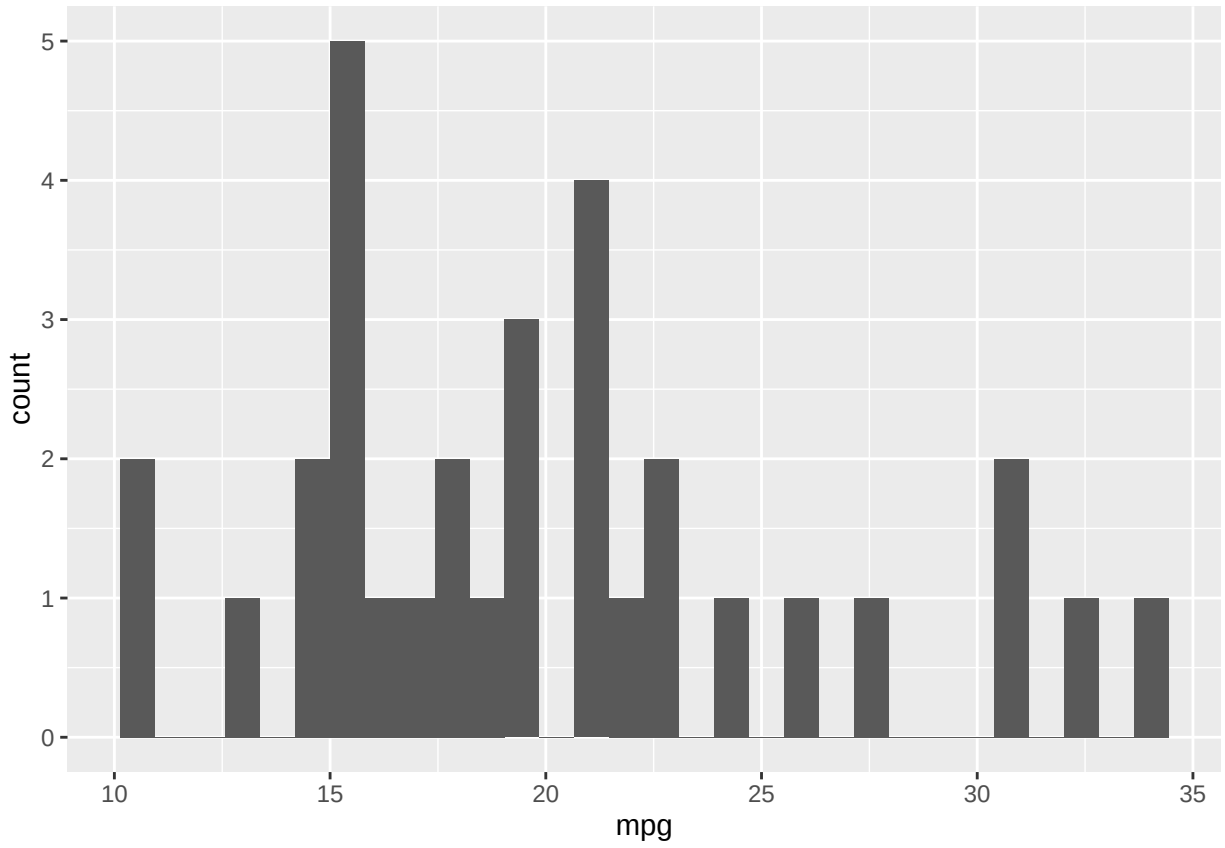
Veja a seguir como é fácil gerar diversos gráficos diferentes utilizando a mesma estrutura do gráfico de dispersão acima:

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

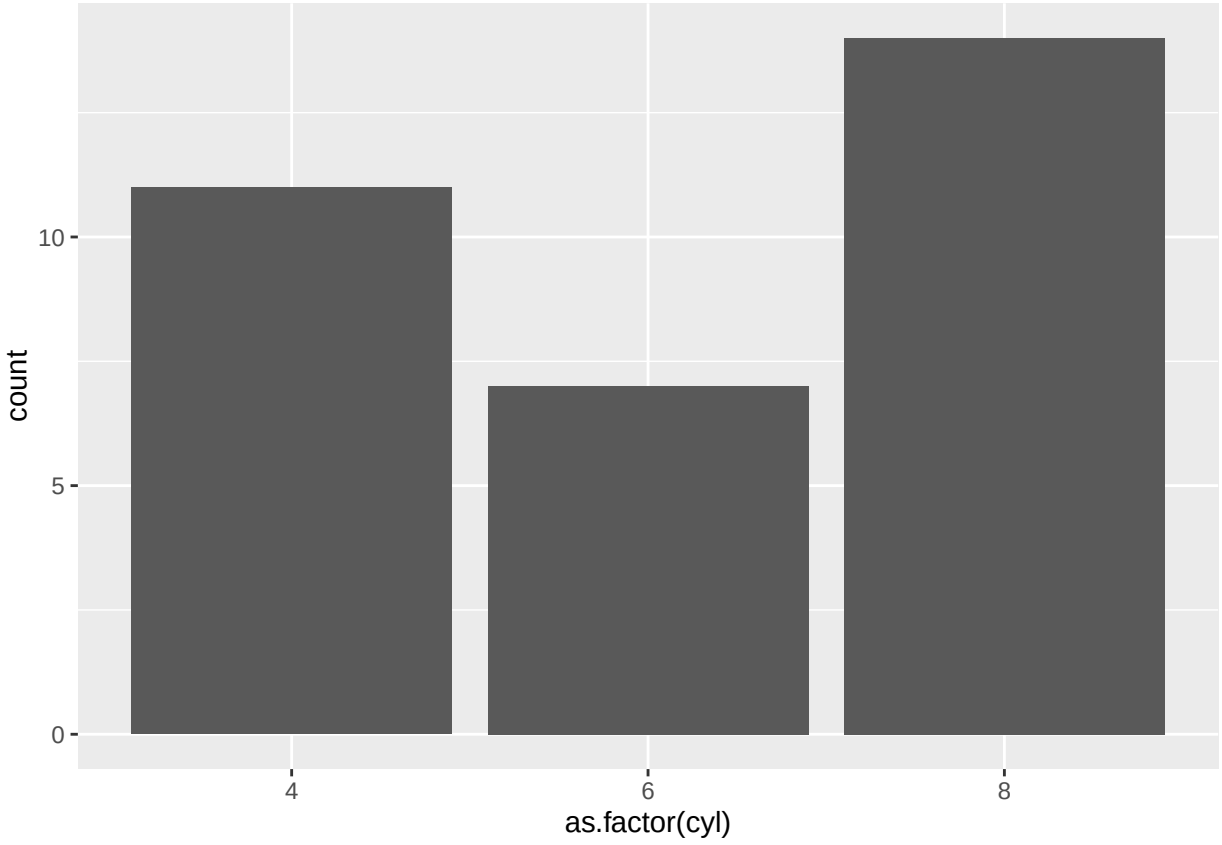



```
ggplot(mtcars, aes(x = mpg)) +
  geom_histogram()
```

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




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



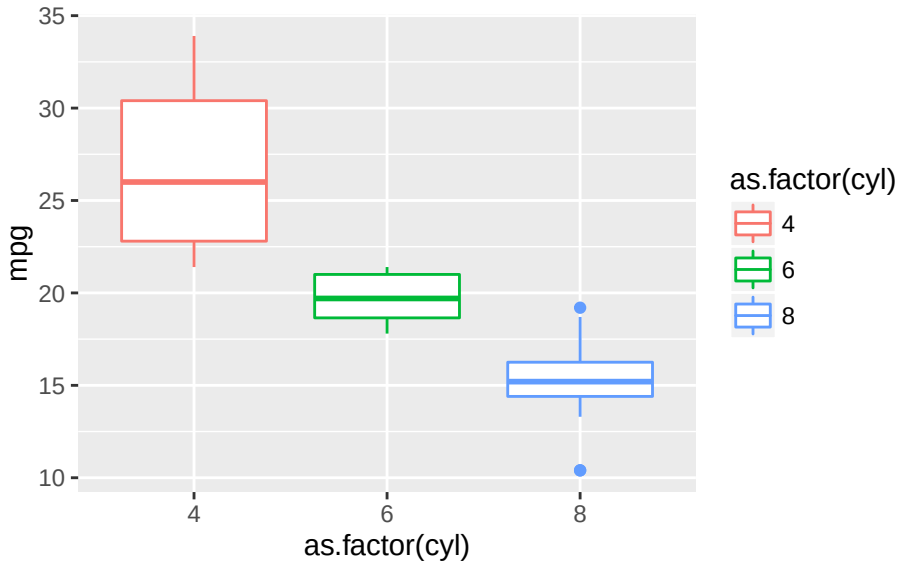
Para fazer um boxplot para cada grupo, precisamos passar para o aspecto x do gráfico uma variável do tipo fator.

2.11.2 Personalizando os gráficos

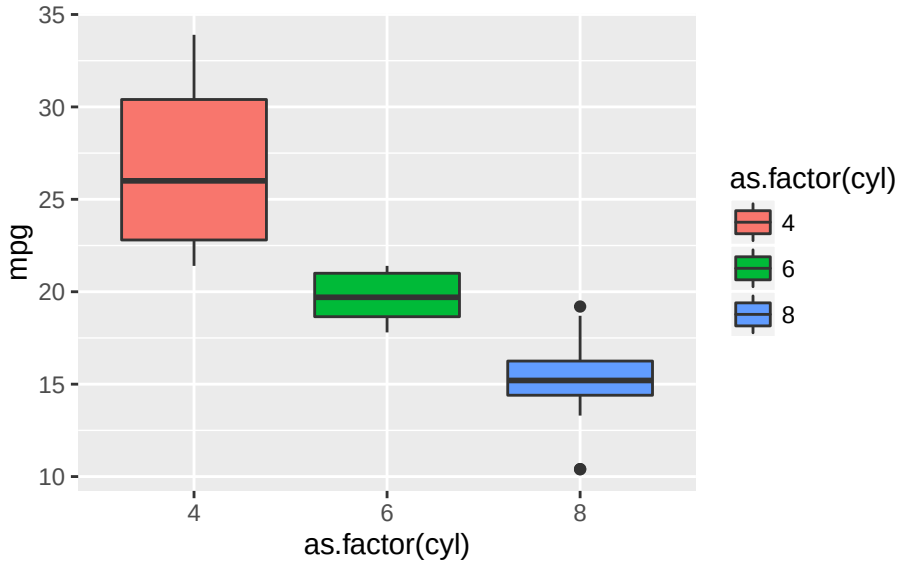
2.11.2.1 Cores

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()
```

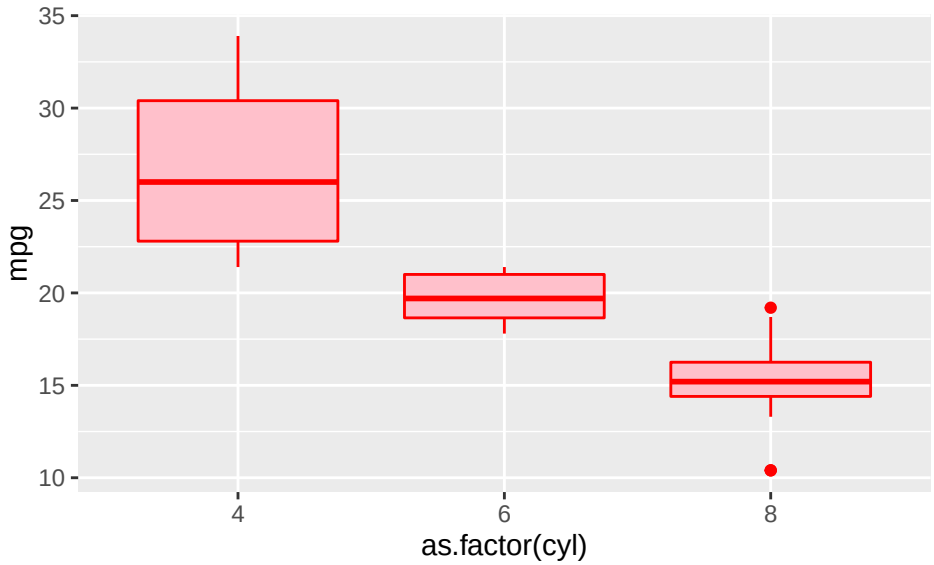



```
ggplot(mtcars, aes(x = as.factor(cyl), y = mpg, fill = as.factor(cyl))) + geom_boxplot()
```



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")
```

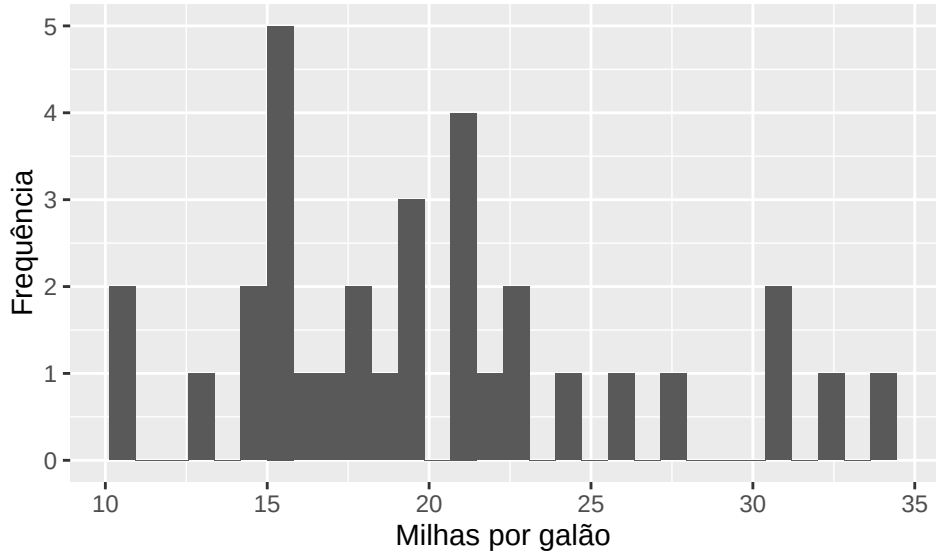


2.11.2.2 Eixos

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")
```

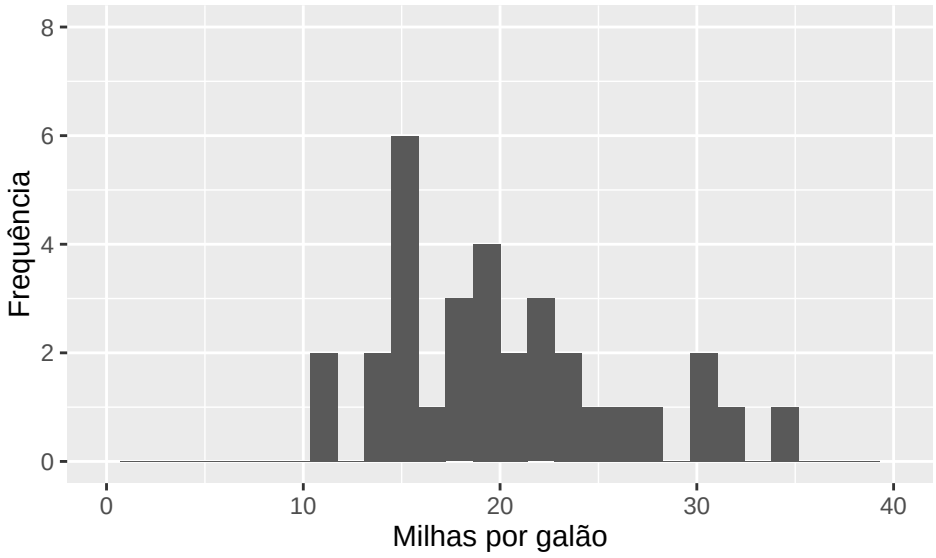
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



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`.
```

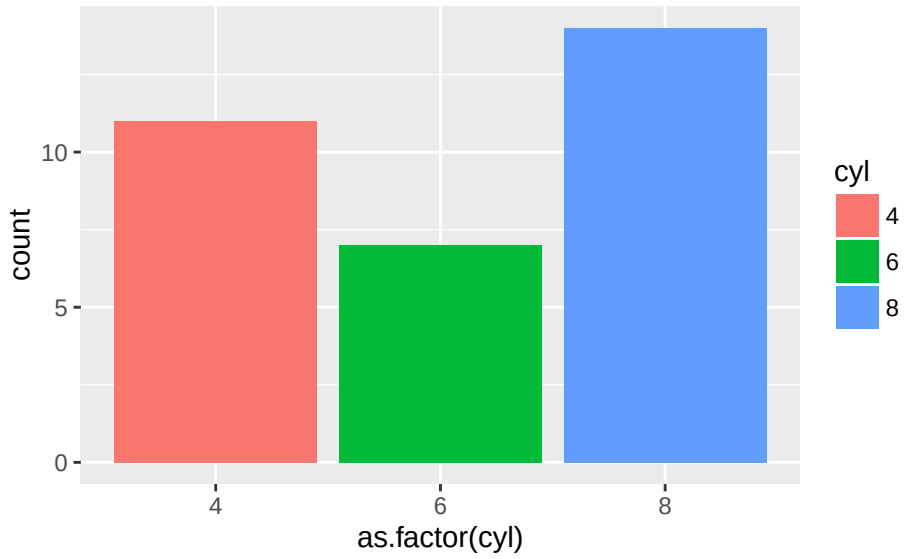


2.11.2.3 Legendas

A legenda de um gráfico pode ser facilmente personalizada.

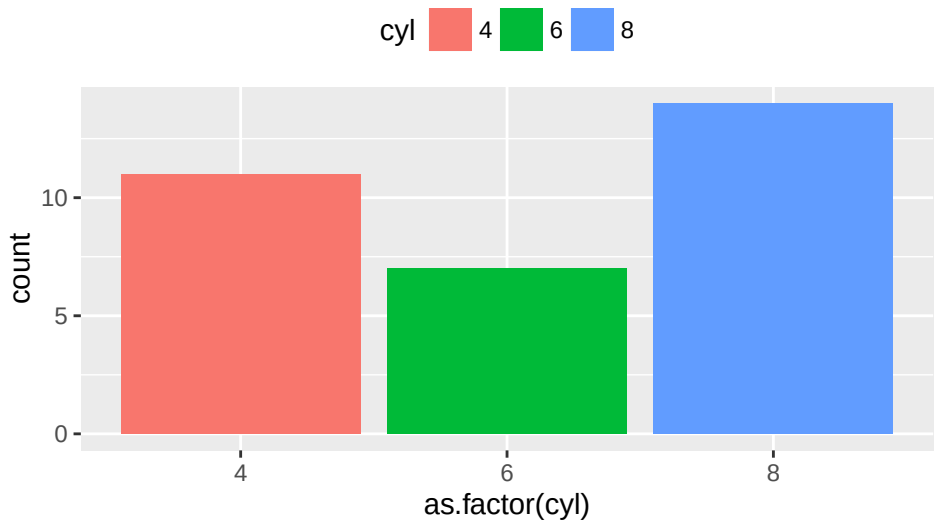
Para trocar o *label* da leganda:

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



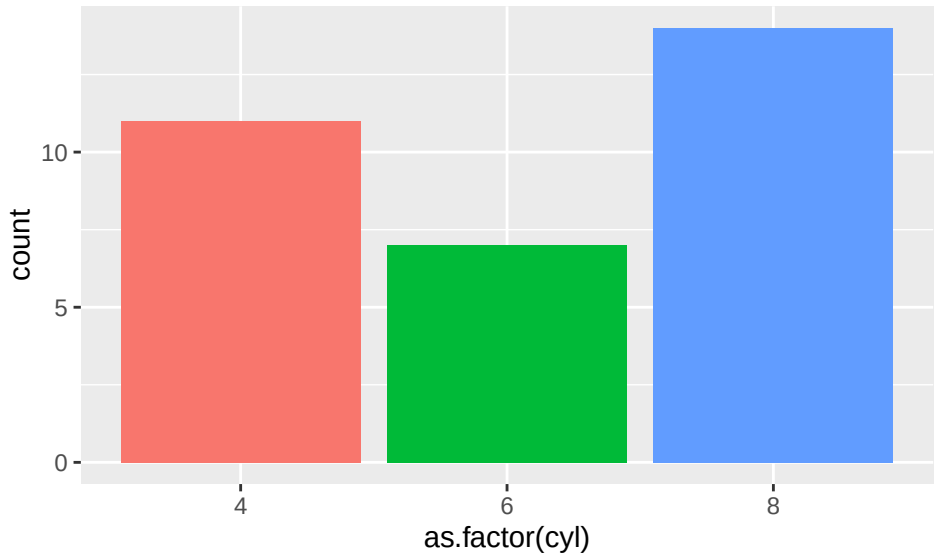
Para trocar a posição da legenda:

```
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)
```

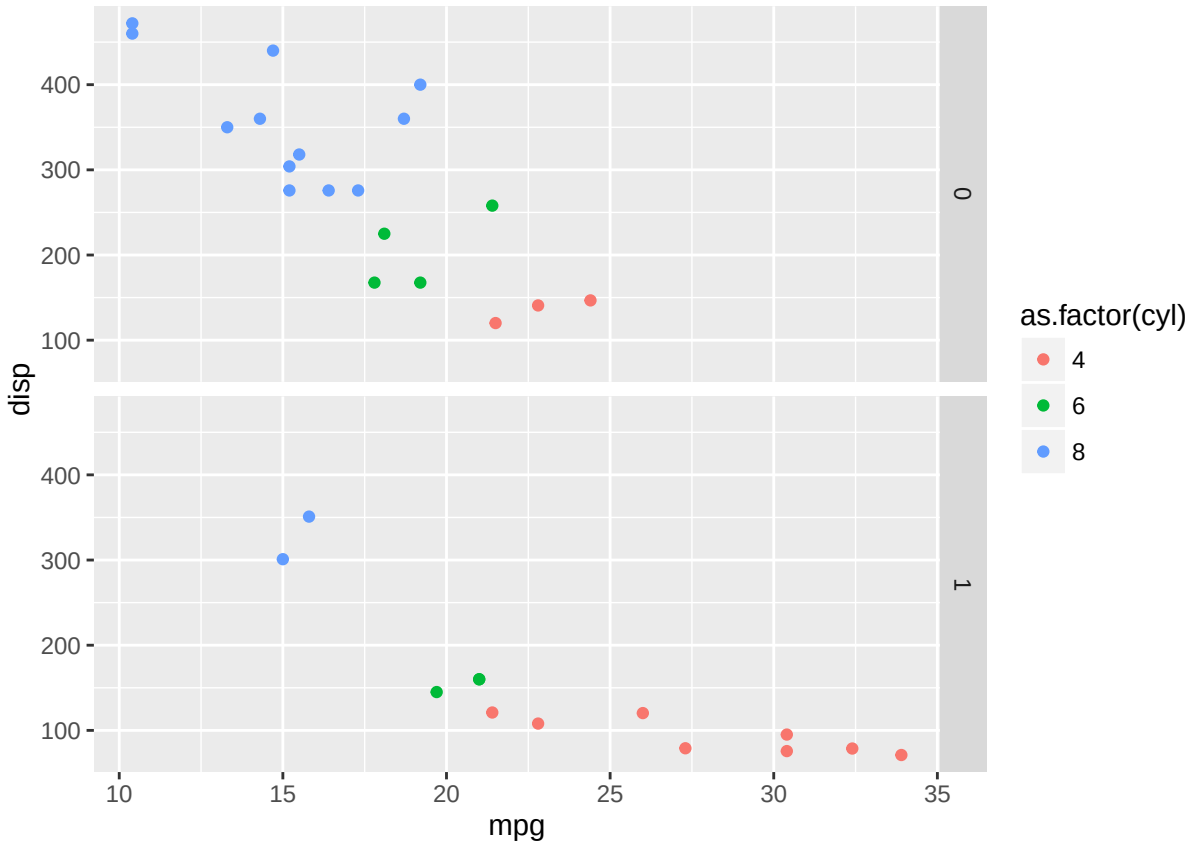


Veja mais opções de personalização aqui!

2.11.2.4 Facets

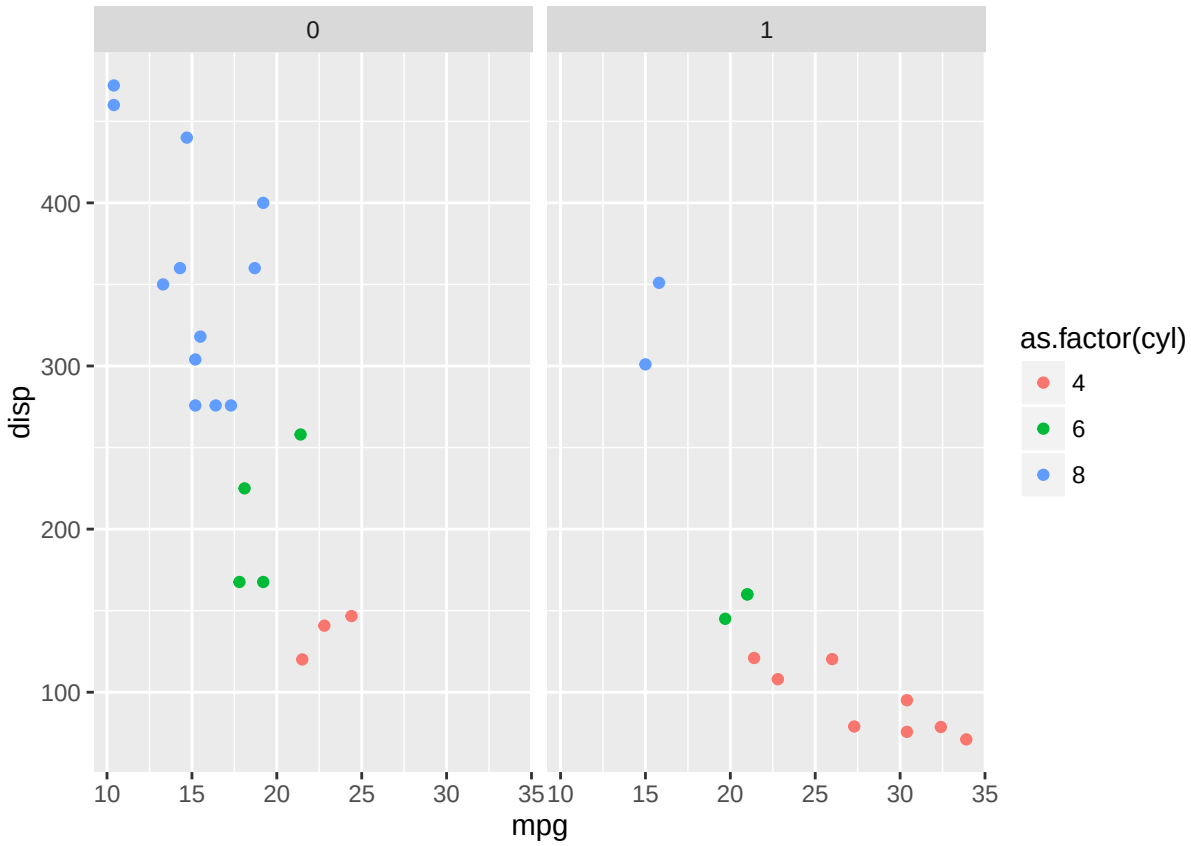
Outra funcionalidade muito importante do ggplot é o uso de *facets*.

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

Chapter 3

Aulas

Chapter 4

Applications

Some *significant* applications are demonstrated in this chapter.

4.1 Example one

4.2 Example two

Chapter 5

Final Words

We have finished a nice book.

Bibliography