Curso de Jurimetria

Associação Brasileira de Jurimetria

2016-08-07

Contents

1 Introdução **5**

1.1 Definição de Jurimetria 5

2 Ferramental de trabalho da ABJ **7**

2.1 Exemplo trabalhado 8

2.2 Data tidying 8

2.3 Pipe 9

2.4 Pacote lubridate para trabalhar com datas 10

2.5 Pacote stringr para trabalhar com textos 11

2.6 Pacotes dplyr e tidyr 17

2.7 Pacotes httr, xml2 e rvest 27

2.8 Web scraping 30

2.9 Buscar documentos 31

2.10 Coletar processos 36

2.11 A importância da criação de APIs públicos nos tribunais 44

2.12 Visualização de dados com ggplot2 47

2.13 Exemplo visualização 62

2.14 Ferramentas de visualização com shiny 66

2.15 Fazendo mais com o shiny 68

3 Simulação **69**

3.1 Primeiros passos 69

3.2 Estruturação 70

3.3 Dados 71

3.4 Implementação 71

3.5 Análise dos resultados 72

3.6 Exemplo - Cadastro Nacional de Adoção 72

4 Referências **89**

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, que atualmente possui diversas ferramentas que ajudam nessas atividades.

Os pacotes utilizados nessa parte são httr, xml2, rvest, dplyr, tidyr, purrr, lubridate, stringr e ggplot2. Também utilizamos um pacote chamado abjutils, construído para atender algumas necessidades frequentes da 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 Exemplo trabalhado

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, e nas quatro Câmaras Extraordinárias do Tribunal de Justiça de São Paulo. Todas as informações foram obtidas através de *web scraping* 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.

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.

No curso, vamos replicar o estudo das câmaras para 2015, passando por todas as fases! Como são muitos processos, vamos trabalhar com uma amostra de apenas mil casos para as visualizações finais.

2.2 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).

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 tidyverse, um universo de pacotes contemporâneos do R que são intuitivos, 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.

2.3 Pipe

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

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

Para usar o operador %>%, primeiramente devemos instalar o pacote magrittr

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

e carregá-lo com a função library()

```
library(magrittr)
```

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

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

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

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

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

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

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

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

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

2.4 Pacote lubridate para trabalhar com datas

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

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

Outras funções importantes

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

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

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

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

Mais informações: ver aqui e aqui.

2.5 Pacote stringr para trabalhar com textos

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

As regras básicas do pacote são:

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

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

2.5.1 Expressões regulares

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

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

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

```
library(stringr)
testes <- c('ban', 'banana', 'abandonado', 'pranab anderson', 'BANANA', 'ele levou ban')
expressoes <- list(
  'ban', # reconhece tudo que tenha "ban", mas não ignora case
```

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

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

2.5.1.1 Quantificadores

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

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

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

2.5.1.2 Conjuntos

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

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

2.5.1.3 Miscelânea

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

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

2.5.2 Funções do stringr

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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


str_split(string, ', ')
```

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

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

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

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

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

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

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

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

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

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

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

```
## [1] "só"
```

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

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

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

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

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

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

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

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

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

2.5.3 Exemplo: decisões das câmaras

Suponha que temos o seguinte vetor de textos de decisões:

```
d_decisoes <- readRDS('data-raw/d_decisoes.rds')


condicao <- str_length(d_decisoes$decisao) < 200 & d_decisoes$situacao == 'Julgado'
set.seed(1247) # reprodutibilidade
decisoes <- d_decisoes$decisao[condicao] %>% sample(10, replace = FALSE)
decisoes
```

```
##  [1] "Negaram provimento ao recurso. V. U."
##  [2] "NEGARAM PROVIMENTO ao recurso. V.U."
##  [3] "Negaram provimento ao apelo do réu.V.U."
##  [4] "Deram provimento ao apelo ministerial para anular o julgamento pelo Tribunal do Júri, devendo a outro ser submetido
##  [5] "Declararam prejudicado o exame do recurso.V.U."
##  [6] "DERAM PROVIMENTO PARCIAL ao recurso para corrigir a pena de multa para 87 dias-multa, mantida, no mais, a
##  [7] "Rejeitaram as preliminares arguidas e negaram provimento à apelação. V.U."
##  [8] "Negaram provimento ao recurso. V. U."
##  [9] "Negaram provimento ao recurso. V. U."
## [10] "Negaram provimento aos recursos defensivos e deram provimento ao recurso ministerial nos termos do V. Acórdão. V.
```

```
negaram <- regex('negaram', ignore_case = TRUE)
parcial <- regex('parcial', ignore_case = TRUE)
deram <- regex('deram', ignore_case = TRUE)


tipos_decisao <- function(decisoes) {
  ifelse(
    str_detect(decisoes, negaram), 'negado', ifelse(
      str_detect(decisoes, parcial), 'parcial', ifelse(
        str_detect(decisoes, deram), 'provido', 'outros'
    ))
  )
}
```

Resultados:

```
tibble::tibble(tipo_decisao = tipos_decisao(decisoes), decisao = decisoes_min)
```

```
## # A tibble: 10 x 2
##    tipo_decisao
##           <chr>
## 1       negado
## 2       negado
## 3       negado
## 4      provido
## 5       outros
## 6      parcial
## 7       negado
## 8       negado
## 9       negado
## 10      negado
## # ... with 1 more variables: decisao <chr>
```

2.6 Pacotes dplyr e tidyr

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.

2.6.1 Trabalhando com tibbles

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

Vamos assumir que temos a seguinte base de dados:

```
d_cjsg
```

```
## # A tibble: 184,250 x 14
##                                                arq    id cd_acordao
##                                              <chr> <chr>       <chr>
## 1  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     1    9381267
## 2  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     2    8671548
## 3  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     3    8634338
## 4  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     4    8536605
## 5  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     5    8509346
## 6  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     6    8490681
## 7  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     7    8466583
## 8  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     8    8449087
## 9  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     9    8429536
## 10 /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html    10    8331899
## # ... with 184,240 more rows, and 11 more variables: n_processo <chr>,
## #   comarca <chr>, data_julgamento <chr>, data_registro <chr>,
## #   ementa <chr>, orgao_julgador <chr>, outros_numeros <chr>,
## #   relatora <chr>, classe_assunto <chr>, txt_ementa <chr>, result <chr>
```

2.6.2 As cinco funções principais do dplyr

- filter
- mutate
- select
- arrange
- summarise

2.6.3 Características

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

2.6.4 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.

2.6.5 select

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

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

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

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

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

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

```
## # A tibble: 184,250 x 4
##                  n_processo                relator data_julgamento
##                       <chr>                  <chr>           <chr>
## 1  0081568-68.2012.8.26.0050          Encinas Manfré      29/01/2015
## 2  2132739-15.2014.8.26.0000          Edison Brandão      27/01/2015
## 3  2204759-04.2014.8.26.0000            Ivan Sartori      27/01/2015
## 4  2179832-71.2014.8.26.0000         Walter da Silva      22/01/2015
## 5  2055376-49.2014.8.26.0000 Guilherme de Souza Nucci      27/01/2015
## 6  0014476-05.2014.8.26.0050         Roberto Solimene      29/01/2015
## 7  0067859-48.2014.8.26.0000           Poças Leitão      29/01/2015
## 8  0054802-60.2014.8.26.0000            Ivan Sartori      27/01/2015
## 9  2128810-71.2014.8.26.0000          Willian Campos      29/01/2015
## 10 2002343-13.2015.8.26.0000     Luis Soares de Mello      27/01/2015
## # ... with 184,240 more rows, and 1 more variables: data_registro <chr>
```

2.6.6 filter

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

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

```
## # A tibble: 41,488 x 4
##       id cd_acordao   comarca              relator
##    <chr>      <chr>     <chr>                <chr>
## 1      1    9381267 São Paulo       Encinas Manfré
## 2      6    8490681 São Paulo     Roberto Solimene
## 3      9    8429536 São Paulo       Willian Campos
## 4     10    8331899 São Paulo  Luis Soares de Mello
## 5     12    8317638 São Paulo Sérgio Mazina Martins
## 6     13    8314983 São Paulo        Péricles Piza
## 7     14    8314866 São Paulo        Péricles Piza
## 8     17    8293299 São Paulo       Ivo de Almeida
## 9     19    8284691 São Paulo       Edison Brandão
## 10    21    8274010 São Paulo  Cesar Mecchi Morales
## # ... with 41,478 more rows
```

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

```
## # A tibble: 6,262 x 5
##       id cd_acordao  comarca data_julgamento                relator
##    <chr>      <chr>    <chr>           <chr>                  <chr>
## 1     33    8221266 Sorocaba      29/01/2015 Alcides Malossi Junior
## 2    136    8211543 Campinas      29/01/2015        Walter da Silva
## 3    174    8211235 Sorocaba      22/01/2015        Walter da Silva
## 4    188    8211081 Sorocaba      29/01/2015        Walter da Silva
```

```
## 5   190   8211053 Campinas     29/01/2015      Walter da Silva
## 6   229   8206214 Sorocaba     29/01/2015    Ricardo Tucunduva
## 7   312   8194316 Sorocaba     29/01/2015       Poças Leitão
## 8   354   8193920 Campinas     29/01/2015       Sérgio Ribas
## 9   417   8193329 Campinas     29/01/2015       Poças Leitão
## 10  427   8193274 Sorocaba     29/01/2015       Poças Leitão
## # ... with 6,252 more rows
```

2.6.7 mutate

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

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

```
## # A tibble: 184,250 x 7
##       id              n_processo           comarca data_julgamento
##    <chr>                   <chr>             <chr>          <chr>
## 1      1 0081568-68.2012.8.26.0050         São Paulo     29/01/2015
## 2      2 2132739-15.2014.8.26.0000         Guarulhos     27/01/2015
## 3      3 2204759-04.2014.8.26.0000           Osasco     27/01/2015
## 4      4 2179832-71.2014.8.26.0000       Adamantina     22/01/2015
## 5      5 2055376-49.2014.8.26.0000          Limeira     27/01/2015
## 6      6 0014476-05.2014.8.26.0050         São Paulo     29/01/2015
## 7      7 0067859-48.2014.8.26.0000           Suzano     29/01/2015
## 8      8 0054802-60.2014.8.26.0000 Presidente Prudente     27/01/2015
## 9      9 2128810-71.2014.8.26.0000         São Paulo     29/01/2015
## 10    10 2002343-13.2015.8.26.0000         São Paulo     27/01/2015
## # ... with 184,240 more rows, and 3 more variables: ano_julgamento <dbl>,
```

```
## #   ano_proc <dbl>, tempo_anos <dbl>
```

2.6.8 arrange

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

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

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

2.6.9 summarise

- Retorna um vetor de tamanho 1 a partir de uma conta com as variáveis.
- Geralmente é utilizado em conjunto com group_by.

- Algumas funções importantes: n(), n_distinct().

```
d_cjsg %>%
  select(id, n_processo, comarca, data_julgamento) %>%
  mutate(ano_julgamento = year(dmy(data_julgamento)),
         ano_proc = str_sub(n_processo, 12, 15),
         ano_proc = as.numeric(ano_proc)) %>%
  mutate(tempo_anos = ano_julgamento - ano_proc) %>%
  arrange(desc(tempo_anos)) %>%
  group_by(comarca) %>%
  summarise(n = n(),
            media_anos = mean(tempo_anos),
            min_anos = min(tempo_anos),
            max_anos = max(tempo_anos)) %>%
  filter(n > 5) %>%
  arrange(desc(media_anos))
```

```
## # A tibble: 275 x 5
##                       comarca     n media_anos min_anos max_anos
##                         <chr> <int>      <dbl>    <dbl>    <dbl>
## 1      F.D. PAULÍNIA/CAMPINAS     7   9.142857        4       13
## 2                      Queluz    63   3.984127        0        7
## 3                      Rosana   126   3.865079        0       14
## 4                   Nhandeara   144   3.777778        0       15
## 5  Santa Rita do Passa Quatro   128   3.765625        0       12
## 6             Pindamonhangaba   582   3.762887        0       19
## 7                   Rancharia   293   3.744027        0       29
## 8                    Bananal    36   3.611111        0       11
## 9                   Paraibuna    61   3.524590        0       14
## 10    Mirante do Paranapanema    85   3.517647        0       19
## # ... with 265 more rows
```

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

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

2.6.10 gather

- "Empilha" o banco de dados

```
library(tidyr)
d_cjsg %>%
  select(cd_acordao:data_registro) %>%
  gather(key, value, -cd_acordao) %>%
  arrange(cd_acordao)
```

```
## # A tibble: 737,000 x 3
##    cd_acordao           key                 value
##         <chr>         <chr>                 <chr>
## 1     8137574     n_processo 0209050-18.2013.8.26.0000
## 2     8137574        comarca              São Paulo
## 3     8137574 data_julgamento             22/01/2015
## 4     8137574  data_registro             22/01/2015
## 5     8137600     n_processo 0016319-58.2014.8.26.0000
## 6     8137600        comarca              São Paulo
## 7     8137600 data_julgamento             22/01/2015
```

```
## 8     8137600   data_registro              22/01/2015
## 9     8137628     n_processo 0104838-68.2005.8.26.0050
## 10    8137628         comarca              São Paulo
## # ... with 736,990 more rows
```

2.6.11 spread

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

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

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

2.6.12 Funções auxiliares

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

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

```
## # A tibble: 30 x 2
##                                  classe     n
##                                   <chr> <int>
## 1                             Apelação 88322
## 2                      Habeas Corpus 54237
## 3           Agravo de Execução Penal 25388
## 4             Embargos de Declaração  6125
## 5         Recurso em Sentido Estrito  4038
## 6                   Revisão Criminal  3675
## 7                Mandado de Segurança   891
## 8  Embargos Infringentes e de Nulidade   485
## 9                 Reexame Necessário   305
## 10                 Agravo Regimental   217
## # ... with 20 more rows
```

2.6.13 Um pouco mais de transformação de dados

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

2.7 Pacotes httr, xml2 e rvest

Esses são os três pacotes mais modernos do R utilizados para fazer web scraping. O pacote xml2 tem a finalidade de estruturar arquivos HTML ou XML de forma eficiente, tornando possível a obtenção de *tags* e seus atributos dentro de um arquivo. Já o pacote httr é responsável por realizar requisições web para obtenção das páginas de interesse, buscando reduzir ao máximo a complexidade da programação. O pacote rvest é escrito **sobre** os dois anteriores e por isso eleva ainda mais o nível de especialização para raspagem de dados.

As características dos pacotes implicam na seguinte regra de bolso. Para trabalhar com páginas simples, basta carregar o rvest e utilizar suas funcionalidades. Caso o acesso à página exija ações mais complexas e/ou artifícios de ferramentas web, será necessário utilizar o httr. O xml2 só será usado explicitamente nos casos raros em que a página está em XML, que pode ser visto como uma generalização do HTML.

Esses pacotes não são suficientes para acessar todo tipo de conteúdo da web. Um exemplo claro disso são páginas em que o conteúdo é produzido por javascript, o que acontece em muitos sites modernos. Para trabalhar com esses sites, é necessário realmente "simular" um navegador que acessa a página web. Uma das melhores ferramentas para isso é o selenium. Não discutiremos selenium nesse curso, mas caso queira se aprofundar, acesse aqui.

2.7.1 Sessões e cookies

No momento que acessamos uma página web, nosso navegador baixa alguns arquivos que "identificam" nosso acesso à página. Esses arquivos são chamados cookies e são usados pelos sites para realizar diversas atividades, como carregar uma página pré-definida pelo usuário caso este acesse o site pela segunda vez.

O httr e por consequência o rvest já guardam esses cookies de forma automática, de forma que o usuário não precise se preocupar com isso. Em casos raros, para construir o web scraper é necessário modificar esses cookies. Nesses casos, estude a função cookies() do httr.

2.7.2 GET e POST

Uma requisição GET envia uma url ao servidor, possivelmente com alguns parâmetros nessa url (que ficam no final da url depois do ?). O servidor, por sua vez, recebe essa url, processa os parâmetros e retorna uma página HTML para o navegador[2].

A requisição POST, no entanto, envia uma url não modificada para o servidor, mas envia também uma lista de dados preenchidos pelo usuário, que podem ser números, textos ou até imagens. Na maioria dos casos, ao submeter um formulário de um site, fazemos uma requisição POST.

O httr possui os métodos GET e POST implementados e são muito similares. A lista de parâmetros enviados pelo usuário pode ser armazenado numa list nomeada, e adicionado ao GET pelo parâmetro query ou no POST pelo parâmetro body. Veremos exemplos disso mais adiante.

[2]para entender sobre server side e user side, acesse server side e user side.

2.7.3 Outras funções do httr

Outras funções úteis:

- write_disk() para escrever uma requisição direto em disco, além de guardar na memória RAM.
- config() para adicionar configurações adicionais. Por exemplo, quando acessar uma página https com certificados inadequados numa requisição GET, rode GET('https://www...', config(ssl_verifypeer=F)).
- oauth_app() para trabalhar com APIs. Não discutiremos conexão com APIs nesse curso, mas é um importante conceito a ser estudado.

2.7.4 Principais funções do rvest

```
library(rvest)
```

```
## Loading required package: xml2
```

Para acessar páginas da web:

- html_session() abre uma sessão do usuário (baixa página, carrega cookies etc).
- follow_link(), jump_to() acessa uma página web a partir de um link (tag <a>) ou url.
- html_form() carrega todos os formulários contidos numa página.
- set_value() atribui valores a parâmetros do formulário.
- submit_form() submete um formulário obtido em html_form.

Para trabalhar com arquivos HTML:

- read_html() lê o arquivo HTML de forma estruturada e facilita impressão.
- html_nodes() cria uma lista com os nós identificados por uma busca em CSS path ou XPath. html_node() é um caso especial que assume que só será encontrado um resultado.
- html_text() extrai todo o conteúdo de um objeto e retorna um texto.
- html_table() extrai o conteúdo de uma <table> e transforma em um data_frame.
- html_attr() extrai um atributo de uma tag, por exemplo href da tag <a>.

2.7.5 CSS path e XPath

O CSS path e o XPath são formas distintas de buscar tags dentro de um documento HTML. O CSS path é mais simples de implementar e tem uma sintaxe menos verborrágica, mas o XPath é mais poderoso. A regra de bolso é tentar fazer a seleção primeiro em CSS e, caso não seja possível, implementar em XPath.

Esses paths serão mostrados *en passant* durante o curso, mas não serão abordados em detalhe. Caso queira se aprofundar no assunto, comece pela ajuda da função ?html_nodes.

2.7.6 APIs com httr

O httr foi criado pensando-se nas modernas APIs que vêm sendo desenvolvidas nos últimos anos. O httr já tem métodos apropriados para trabalhar com Facebook, Twitter e Google, entre outros.

Para um guia completo de como utilizar APIs no R, acesse esse tutorial. Um exemplo de pacote que utiliza API usando esse tutorial melhores práticas pode ser acessado aqui.

2.8 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.

2.8.1 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.

2.8.2 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.

2.9 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 2016-08-07) 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_session <- function() {
  rvest::html_session('http://esaj.tjsp.jus.br/cjsg/consultaCompleta.do')
}
```

```
cjsg <- function(s, parms = cjsg_parms(s), path = './cjsg',
                 min_pag = 1, max_pag = 10, overwrite = FALSE,
                 verbose = TRUE, p = .05) {
  suppressWarnings(dir.create(path, recursive = TRUE))
  if (!file.exists(path)) stop(sprintf('Pasta não "%s" pôde ser criada', path))
  r0 <- s %>% rvest::submit_form(parms)
  n_pags <- if (is.na(max_pag) || is.infinite(max_pag)) cjsg_npags(r0) else max_pag
  abjutils::dvec(cjsg_pag, 1:n_pags, path = path, ow = overwrite, s = s)
}
```

```
cjsg_pag <- function(pag, path, ow, s) {
  Sys.sleep(1)
  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(arq) || ow) {
    httr::GET(sprintf(u, pag), httr::write_disk(arq, overwrite = ow), handle = s$handle)
    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(s, livre = '', data_inicial = NULL, data_final = NULL, secoes = '') {
  secoes <- paste(secoes, collapse = ',')
  dt_inicial <- ''
  if (!is.null(data_inicial)) {
    dt_inicial <- sprintf('%02d/%02d/%d', lubridate::day(data_inicial),
                          lubridate::month(data_inicial),
                          lubridate::year(data_inicial))
  }
  dt_final <- ''
  if (!is.null(data_final)) {
    dt_final <- sprintf('%02d/%02d/%d', lubridate::day(data_final),
                        lubridate::month(data_final),
                        lubridate::year(data_final))
  }
  suppressWarnings({
    s %>%
      rvest::html_form() %>%
```

```
    dplyr::first() %>%
    rvest::set_values('dados.buscaInteiroTeor' = livre,
                      'secoesTreeSelection.values' = secoes,
                      'dados.dtJjulgamentoInicio' = dt_inicial,
                      'dados.dtJjulgamentoFim' = dt_final)
  })
}
```

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, parms = NULL) {
  if (!is.null(parms)) req <- req %>% rvest::submit_form(parms)
  num <- req$response %>%
    httr::content('text') %>%
    xml2::read_html() %>%
    rvest::html_node('#nomeAba-A') %>%
    rvest::html_text() %>%
    tidyr::extract_numeric()
  (num %/% 20) + 1
}
```

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(tibble::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 projeto das câmaras, rodamos o seguinte código:

```
library(magrittr)
library(tjsp)

sec <- list_secoes_2inst() %>%
  dplyr::filter(stringr::str_detect(secao, '[Cc]rim'),
                stringr::str_detect(pai, 'CRIM')) %>%
  with(cod)

session <- cjsg_session()
parms <- session %>%
  cjsg_parms(secoes = sec, data_inicial = '2015-01-01', data_final = '2015-12-31')

# numero de paginas a serem baixadas
session %>% cjsg_npags(parms)

d_result <- session %>%
  cjsg(parms, path = 'data-raw/cjsg', max_pag = 100)
```

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.

2.10 Coletar processos

Antes de coletar os processos, é necessário ler os arquivos HTML baixados na etapa anterior.

```
parse_cjsg_um <- function(i, nodes) {
  node <- nodes[[i]]
  trim <- stringr::str_trim
  id <- node %>%
    rvest::html_node('.ementaClass') %>%
    rvest::html_text() %>%
    trim() %>%
    stringr::str_replace_all('[^0-9]', '')
  infos <- node %>%
    rvest::html_node('.downloadEmenta') %>% {
      tibble::tibble(n_processo = trim(rvest::html_text(.)),
                     cd_acordao = rvest::html_attr(., 'cdacordao'))
    }
  ca <- node %>%
    rvest::html_node('.assuntoClasse') %>%
    rvest::html_text() %>%
    trim()
  tsf <- node %>%
    rvest::html_node('textarea') %>%
    rvest::html_text()
  tab_infos <- node %>%
    rvest::html_nodes('.ementaClass2') %>%
```

```
    rvest::html_text() %>%
    stringr::str_split_fixed(':', 2) %>%
    data.frame(stringsAsFactors = FALSE) %>%
    magrittr::set_names(c('key', 'val')) %>%
    dplyr::mutate_all(dplyr::funs(trim(.))) %>%
    dplyr::mutate(key = tolower(abjutils::rm_accent(key)),
                  key = stringr::str_replace_all(key, ' +', '_'),
                  key = stringr::str_replace_all(key, '[^a-z_]', ''),
                  key = stringr::str_replace_all(key, '_d[eo]_', '_')) %>%
    tidyr::spread(key, val) %>%
    dplyr::bind_cols(infos) %>%
    dplyr::mutate(id = id, classe_assunto = ca, txt_ementa = tsf) %>%
    dplyr::select(id, cd_acordao, n_processo, dplyr::everything(), txt_ementa)
  tab_infos
}

parse_cjsg_arq <- function(arq) {
  itens <- xml2::read_html(arq, encoding = 'UTF-8') %>%
    rvest::html_nodes('.fundocinza1')
  abjutils::dvec(parse_cjsg_um, 1:length(itens), nodes = itens, verbose = FALSE) %>%
    dplyr::select(-item)
}

#' Parser do CJSG
#'
#' Parser dos arquivos HTML baixados pela função \code{\link{cjsg}}.
#'
#' @param arqs vetor de arquivos (caminho completo) a serem lidos.
#'
#' @return tibble com as colunas
#' \itemize{
#'   \item \code{arq} nome do arquivo lido.
#'   \item \code{id} id contido na página lida.
```

```
#'   \item \code{cd_acordao} código único do acórdão.
#'   \item \code{n_processo} número do processo (pode repetir).
#'   \item \code{comarca} nome da comarca.
#'   \item \code{data_julgamento} data de julgamento em formato \%d/\%m/\%Y.
#'   \item \code{data_registro} data de registro no sistem em formato \%d/\%m/\%Y.
#'   \item \code{ementa} ementa do acórdão (muitos vazios).
#'   \item \code{orgao_julgador} câmara julgadora do recurso.
#'   \item \code{outros_numeros} números antigos / complementares.
#'   \item \code{relatora} Nome do relator ou relatora do recurso.
#'   \item \code{classe_assunto} Classe / assunto, separados por " / ".
#'   \item \code{txt_ementa} Texto da ementa sem formatação.
#' }
#' @export
parse_cjsg <- function(arqs) {
  abjutils::dvec(parse_cjsg_arq, arqs) %>%
    dplyr::rename(arq = item)
}
```

Rodando a função criada.

```
arqs <- dir('data-raw/cjsg', full.names = TRUE)
d_cjsg <- parse_cjsg(arqs)
saveRDS(d_cjsg, 'data-raw/d_cjsg.rds')
```

```
d_cjsg <- readRDS('data-raw/d_cjsg.rds')
d_cjsg
```

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

```
## 7  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     7    8466583
## 8  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     8    8449087
## 9  /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     9    8429536
## 10 /home/storage/abj/raw/TJSP/cjsg/cjsg_mes01/00001.html     10    8331899
## # ... with 184,240 more rows, and 11 more variables: n_processo <chr>,
## #   comarca <chr>, data_julgamento <chr>, data_registro <chr>,
## #   ementa <chr>, orgao_julgador <chr>, outros_numeros <chr>,
## #   relatora <chr>, classe_assunto <chr>, txt_ementa <chr>, result <chr>
```

Agora criamos a função que baixa processos.

```
dados_cposg <- function(p) {
  list('conversationId' = '',
       'paginaConsulta' = '1',
       'localPesquisa.cdLocal' = '-1',
       'cbPesquisa' = 'NUMPROC',
       'tipoNuProcesso' = 'UNIFICADO',
       'numeroDigitoAnoUnificado' = stringr::str_sub(p, 1, 11),
       'foroNumeroUnificado' = stringr::str_sub(p, -4, -1),
       'dePesquisaNuUnificado' = p,
       'dePesquisaNuAntigo' = '')
}

#' Baixa um processo.
#'
#' Baixa um processo na consulta de processos de segundo grau do TJSP. Deve ser usado internamente.
#'
#' @param p número do processo (string apenas com os números).
#' @param path caminho da pasta onde será salvo o arquivo HTML.
#' @param ow sobrescrever o arquivo HTML?
#'
cposg_um <- function(p, path, ow) {
  Sys.sleep(1)
  arq <- sprintf('%s/%s.html', path, p)
  if (!file.exists(arq) || ow) {
```

```
    httr::GET('https://esaj.tjsp.jus.br/cpo/sg/search.do',
              query = dados_cposg(p),
              config = httr::config(ssl_verifypeer = FALSE),
              httr::write_disk(arq, overwrite = ow))
    tibble::tibble(result = 'OK')
  } else {
    tibble::tibble(result = 'já existe')
  }
}

#' Baixa processos
#'
#' Baixa processos na consulta de processos de segundo grau do TJSP.
#'
#' @param processos número do processo (string apenas com os números).
#' @param path caminho da pasta onde os arquivos HTML serão salvos.
#' @param overwrite sobrescrever os arquivos HTML?
#'
#' @export
cposg <- function(processos, path = 'data-raw/cposg', overwrite = FALSE) {
  suppressWarnings(dir.create(path, recursive = TRUE))
  processos <- gsub('[^0-9]', '', processos)
  abjutils::dvec(cposg_um, processos, path = path, ow = overwrite)
}
```

Rodando a função criada.

```
d_cjsg %>%
  distinct(n_processo) %>%
  with(n_processo) %>%
  cposg()
```

Extraindo as partes do arquivo HTML.

```
partes_cposg_um <- function(arq) {
  h <- arq %>% xml2::read_html(encoding = 'UTF-8')
```

```
todas_partes <- h %>% rvest::html_nodes('#tableTodasPartes') %>% length()
if (todas_partes > 0) {
  nodes <- h %>%
    rvest::html_nodes('#tableTodasPartes > .fundoClaro')
} else {
  nodes <- h %>%
    rvest::html_nodes('#tablePartesPrincipais > .fundoClaro')
}
purrr::map_df(seq_along(nodes), function(i) {
  node <- nodes[[i]]
  titulos <- node %>%
    rvest::html_nodes('.mensagemExibindo') %>%
    rvest::html_text() %>%
    stringr::str_trim() %>%
    stringr::str_replace_all(' ', '')
  tirar <- paste(titulos, collapse = '|')
  nomes <- titulos %>%
    tolower() %>%
    abjutils::rm_accent() %>%
    stringr::str_replace_all('[^a-z]', '') %>%
    paste(sprintf('%02d', 1:length(.)), sep = '_')
  node %>%
    rvest::html_text() %>%
    stringr::str_trim() %>%
    stringr::str_replace_all(' ', '') %>%
    stringr::str_replace_all(tirar, '') %>%
    stringr::str_trim() %>%
    stringr::str_split('[\n\t\r ]{2,}', simplify = TRUE) %>%
    data.frame(stringsAsFactors = FALSE) %>%
    setNames(nomes) %>%
    tidyr::gather() %>%
    tibble::as_data_frame() %>%
    tidyr::separate(key, c('tipo', 'id_tipo'), sep = '_') %>%
```

```
    dplyr::mutate(id = i) %>%
    dplyr::select(id, id_tipo, tipo, nome = value) %>%
    dplyr::mutate(result = 'OK')
  })
}

partes_cposg <- function(arqs, verbose = FALSE) {
  abjutils::dvec(partes_cposg_um, arqs, verbose = verbose) %>%
    rename(arq = item)
}
```

```
arqs <- dir('data-raw/cposg', full.names = TRUE)
d_partes <- arqs %>% partes_cposg()
saveRDS(d_partes, 'data-raw/d_partes.rds')
d_partes
```

```
## # A tibble: 3,689 x 6
##                                    arq    id id_tipo     tipo
##                                  <chr> <int>   <chr>    <chr>
## 1  data-raw/cposg/00000021520118260412.html     1      01 apelante
## 2  data-raw/cposg/00000021520118260412.html     1      02 advogado
## 3  data-raw/cposg/00000021520118260412.html     2      01  apelado
## 4  data-raw/cposg/00000049620148260244.html     1      01 apelante
## 5  data-raw/cposg/00000049620148260244.html     2      01  apelado
## 6  data-raw/cposg/00000049620148260244.html     2      02 advogada
## 7  data-raw/cposg/00000071020138260075.html     1      01 apelante
## 8  data-raw/cposg/00000071020138260075.html     1      02 advogado
## 9  data-raw/cposg/00000071020138260075.html     2      01  apelado
## 10 data-raw/cposg/00000085120138260218.html     1      01 apelante
## # ... with 3,679 more rows, and 2 more variables: nome <chr>, result <chr>
```

```
decisoes_cposg_um <- function(arq) {
  html <- xml2::read_html(arq, encoding = 'UTF-8')
  xpath <- '(//table[@width="98%" and @align="center"])[last()]'
  r <- rvest::html_node(html, xpath = xpath)
```

```
  tab <- rvest::html_table(r)
  names(tab) <- c('data', 'situacao', 'decisao')
  tab$result <- 'OK'
  return(tab)
}
decisoes_cposg <- function(arqs, verbose = FALSE) {
  abjutils::dvec(decisao_cposg_um, arqs, verbose = verbose) %>%
    dplyr::rename(arq = item)
}
```

```
arqs <- dir('data-raw/cposg', full.names = TRUE)
d_decisoes <- decisoes_cposg(arqs)
saveRDS(d_decisoes, 'data-raw/d_decisoes.rds')
d_decisoes
```

```
d_decisoes <- readRDS('data-raw/d_decisoes.rds')
d_decisoes
```

```
## # A tibble: 1,120 x 5
##                                    arq       data situacao
##                                  <chr>      <chr>     <chr>
## 1  data-raw/cposg/00000021520118260412.html 15/12/2014  Julgado
## 2  data-raw/cposg/00000049620148260244.html 10/12/2015  Julgado
## 3  data-raw/cposg/00000071020138260075.html 18/05/2015  Julgado
## 4  data-raw/cposg/00000085120138260218.html 11/11/2015  Julgado
## 5  data-raw/cposg/00000094520148260624.html 26/02/2015  Julgado
## 6  data-raw/cposg/00000104920148260165.html 12/05/2015  Julgado
## 7  data-raw/cposg/00000109420148260247.html 24/09/2015  Julgado
## 8  data-raw/cposg/00000109420148260635.html 15/12/2015  Julgado
## 9  data-raw/cposg/00000125120108260038.html 03/09/2015  Julgado
## 10 data-raw/cposg/00000135920148260082.html 17/08/2015  Julgado
## # ... with 1,110 more rows, and 2 more variables: decisao <chr>,
## #   result <chr>
```

2.11 A importância da criação de APIs públicos nos tribunais

(texto extraído do Portal da Jurimetria).

Neste texto fazemos uma apresentação sobre a prática do Jurimetrista, mostrando com um exemplo de como isso é divertido, e também como pode fazer com que passemos por alguns caminhos um tanto tortuosos. Esperamos que o leitor não entenda esse post como um desincentivo à prática da Jurimetria, mas sim como uma motivação para que cada vez mais as pesquisas empíricas no Direito sejam facilitadas.

2.11.1 Busca e extração

Jurimetristas são aquelas pessoas que, remando contra a maré do abstrato e do determinismo, tentam entender a realidade olhando a realidade. Pessoas que não se contentam em estudar a lei, mas querem ver o que está acontecendo no mundo. Nessa viagem, já encontramos diversos personagens: advogados, juristas, estatísticos, economistas, cientistas sociais, cientistas políticos, psicólogos, sociólogos, e por aí vai. Existem até estes raros profissionais que transitam em duas ou mais áreas de forma exemplar. Muito bonita a ideia da Jurimetria.

Uma necessidade que temos nessa área (e qualquer ciência) é tentar verificar ou invalidar nossas hipóteses com evidências obtidas através de dados. Para isso, tentamos aprender mais sobre o Direito e seus mecanismos utilizando como ferramenta complementar a estatística. Com metodologias adequadas de pesquisa e com os dados em mãos, construímos e ajustamos modelos que buscam explicar, com erros, a realidade. Com os dados em mãos...

Pois é, nós precisamos de dados. Muitas vezes, nosso objeto de estudo são processos dentro de um determinado escopo (intervalo de tempo, região geográfica, com determinadas características, etc.), seja para estudar seus valores, seus resultados, ou seus tempos de tramitação. Para conseguir isso, só precisamos de um bom repositório de dados, de onde poderemos extrair o que queremos analisar.

Pela Lei 12.527, a Lei de Acesso à Informação, (quase) todos os processos são públicos! Fantástico, pois assim teremos facilidade em obter nossos dados...

No entanto, não é bem isso o que observamos na realidade.

2.11.2 Como o profissional do Direito acessa seus dados

Chegou um processo no escritório de advocacia. Precisamos preencher dados sobre ele no sistema jurídico. O que fazemos? Entramos no site do tribunal (TJSP, TRT2, ou seja lá qual for), digitamos o número CNJ do processo (todos os processos judiciais do Brasil possuem uma regra de numeração padronizada!), e voilá, encontramos

todas as informações que precisamos. Perfeito! Se precisamos de jurisprudência, fazemos uma pesquisa por palavras-chave no TJ, e encontramos os argumentos que precisamos. Aqui não funciona tão bem, mas ainda é tranquilo.

Existem inúmeras ferramentas jurídicas para busca e recuperação de informações. Só alguns exemplos: Digesto, JusBrasil, sistemas para escritórios de advogados, LexML, TJ's, Justiça Aberta, Diário Oficial, e muitos outros. Com eles, podemos buscar as informações que precisamos, dispondo de uma infinidade de alternativas. Desse modo, se um pesquisador tiver em mãos o número do processo, ele achará informações do processo. Se ele quiser uma lista de processos, ainda poderá fazer isso usando as ferramentas de busca, e nessas listas ele usualmente encontrará o que precisa. Concluindo: os sistemas jurídicos até que são eficazes, mas são todos voltados para a busca de informações.

2.11.3 Como o estatístico acessa seus dados

Um estatístico chora de alegria quando acessa sites como o do PNUD, ou quando tem a possibilidade de exportar para planilhas os dados de acordo com alguma definição de população. Para estatísticos também temos muitos exemplos de sites, como IpeaData, Datasus, IBGE, entre outros. O estatístico mais corajoso ainda poderá utilizar APIs (Application Programming Interfaces) para obter dados de tweets, posts no facebook, e por aí vai. O importante é notar que os sistemas voltados para estatísticos são em sua maioria voltados para extração de informações, no nível de microdados. Os dados chegam praticamente prontos para análise, e não somente para consulta. Claro que muitas vezes é necessário fazer uma limpeza aqui e ali, mas isso também é nosso trabalho.

2.11.4 Como o jurimetrista acessa seus dados

O jurimetrista então se encontra numa situação engraçada: ele quer dados da população toda (ou pelo menos uma amostra aleatória), com linhas e colunas, numa planilha toda padronizada, e tudo o que consegue encontrar são documentos individuais, listagens de processos, páginas web e arquivos PDF. Nesse contexto, com certeza vale aquela regra de que o estatístico passa 80% do tempo arrumando a base de dados e 20% fazendo análise, sendo essa uma estimativa otimista. Sem sombra de dúvidas este desafio é enfrentado por muitas outras áreas do conhecimento, mas é curioso olhar a Justiça, que é aberta, e verificar que para esse fim, na prática ela não está tão aberta assim.

2.11.5 Como melhorar?

Tapando o sol com a peneira:

Não é só no Direito que esse problema fica evidente. Muitos cientistas sociais e cientistas políticos, economistas, entre outros, encontram esse tipo de problema. Muitas vezes as páginas web estão lá, disponíveis, mas levaríamos uma eternidade para buscar manualmente todas as páginas que precisamos. Para resolver isso, construímos web crawlers / scrapers, que a grosso modo são robôs que passeiam pelas páginas e baixam suas informações automaticamente (crawling) e depois tranformam essas páginas baixadas em dados (scraping).

Os web crawlers / scrapers são muito mais comuns do que imaginamos. O JusBrasil, o Digesto, a AASP, e muitos outros sites que se baseiam nos diários oficiais utilizam essas ferramentas. No entanto, nesses casos, as ferramentas são usadas para baixar e processar as páginas, para então indexar e permitir, mais uma vez, que o usuário faça buscas. Para utilizar esses sites como ferramenta analítica (para fazer uma análise estatística, por exemplo), precisaríamos construir web crawlers/scrapers dessas páginas...

Para o jurimetrista, saber utilizar esse tipo de ferramenta (ou saber quem sabe usar), por conta das circunstâncias, é essencial. Devo admitir que as pesquisas da ABJ foram fortemente impulsionadas por esses brinquedos, e que sem eles não teríamos metade dos dados que temos hoje. Só para se ter uma ideia do poder da ferramenta, mostrarei um exemplo baseado no sistema Justiça Aberta, que é o que dá o nome do artigo.

Mas o problema é mais embaixo: Nem todo mundo sabe criar e usar web crawlers / scrapers e, mesmo na estatística, são raros os profissionais que dominam essa arte. Nesse sentido, até estamos construindo algumas funções básicas de pesquisa em um pacote do R para facilitar algumas pesquisas mais simples (aguardem novidades...), de ordem acadêmica. No entanto, essas ferramentas não são capazes de resolver qualquer problema. Só a existência dos captchas (aqueles textos que vêm em imagem para verificar se você não é um robô), por exemplo, que são feitos para bloquear essas ferramentas, já atrapalha bastante. E pior, existem coisas que não são possíveis de fazer, simplesmente porque os sites não dão acesso. Tente, por exemplo, no TJSP, encontrar uma lista de todos os processos em andamento. Ou então listar todos os processos julgados no TRT2, em primeira instância. São inúmeros os exemplos em que simplesmente não conseguimos as informações que precisamos, o que é irônico pois todos os processos (salvo os que correm em segredo de justiça) são públicos. Nesses casos, talvez o único jeito de driblar o problema seria utilizar a Lei de Acesso à Justiça, que é fantástica, mas faz com que nos tornemos agentes passivos nas pesquisas.

2.11.6 Solução a longo prazo:

Mais interessante do que caçar dados por toda a eternidade e fazer disso uma profissão, seria resolver o problema na sua raiz: modificar os sites dos Tribunais, permitindo extrações, e construir APIs que permitam que pesquisadores busquem as informações públicas de maneira tranquila, segura e organizada. Esse tipo de ferramenta é surpreendentemente simples de se construir (uma rápida pesquisa no google já mostra dezenas de tutoriais), e não causaria

grandes impactos na infraestrutura dos tribunais. Ao permitir que os dados sejam baixados de forma "oficial", seria possível controlar melhor o volume de dados transferido por unidade de tempo, evitando que os servidores dos tribunais fiquem sobrecarregados.

2.11.7 Vantagens de permitir o acesso aos dados

Não é difícil pensar como uma estrutura para armazenamento e extração de dados pode ajudar a todos, profissionais do Direito, estatísticos, jurimetristas. Aqui, convido o leitor a sugerir soluções e vantagens, colocando apenas um pequeno cardápio:

Jurimetristas terão mais bases de dados para analisar! A pesquisa empírica no Direito teria um grande avanço. Análise de texto, decisões judiciais, tempo processual, volume processual, tudo isso seria possível. Ao construir APIs para download dos dados, os tribunais poderão obter informações das pessoas que estão utilizando a ferramenta, e quais consultas fizeram. Essas informações podem ser valiosas para os tribunais. Seria maior o feedback a respeito dos dados públicos, o que teria como consequência a melhora da documentação, novas padronizações, etc.

Advogados poderiam falar em números com maior responsabilidade, e discutir com maior propriedade (aqui, sempre tomando cuidado para não mentir com a estatística!). Juízes, aliados de um pessoal de TI e estatística, poderiam gerir seus processos e decidir com maior eficiência (aqui, sem entrar nos méritos de decisões "mais justas" por conta dos dados). Ferramentas jurídicas em escritórios poderão preencher alguns campos automaticamente, apenas com o número do processo (já existem muitas que fazem isso, mas o caminho é tortuoso).

2.11.8 Expectativas

Acreditamos fortemente que as preces serão atendidas e que no futuro o Brasil será referência mundial no que diz respeito a estruturação de dados jurídicos, e que isso será muito benéfico para o país. Processos nós temos (e muitos!); só falta analisar.

2.12 Visualização de dados com ggplot2

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

2.12.1 Construindo gráficos

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

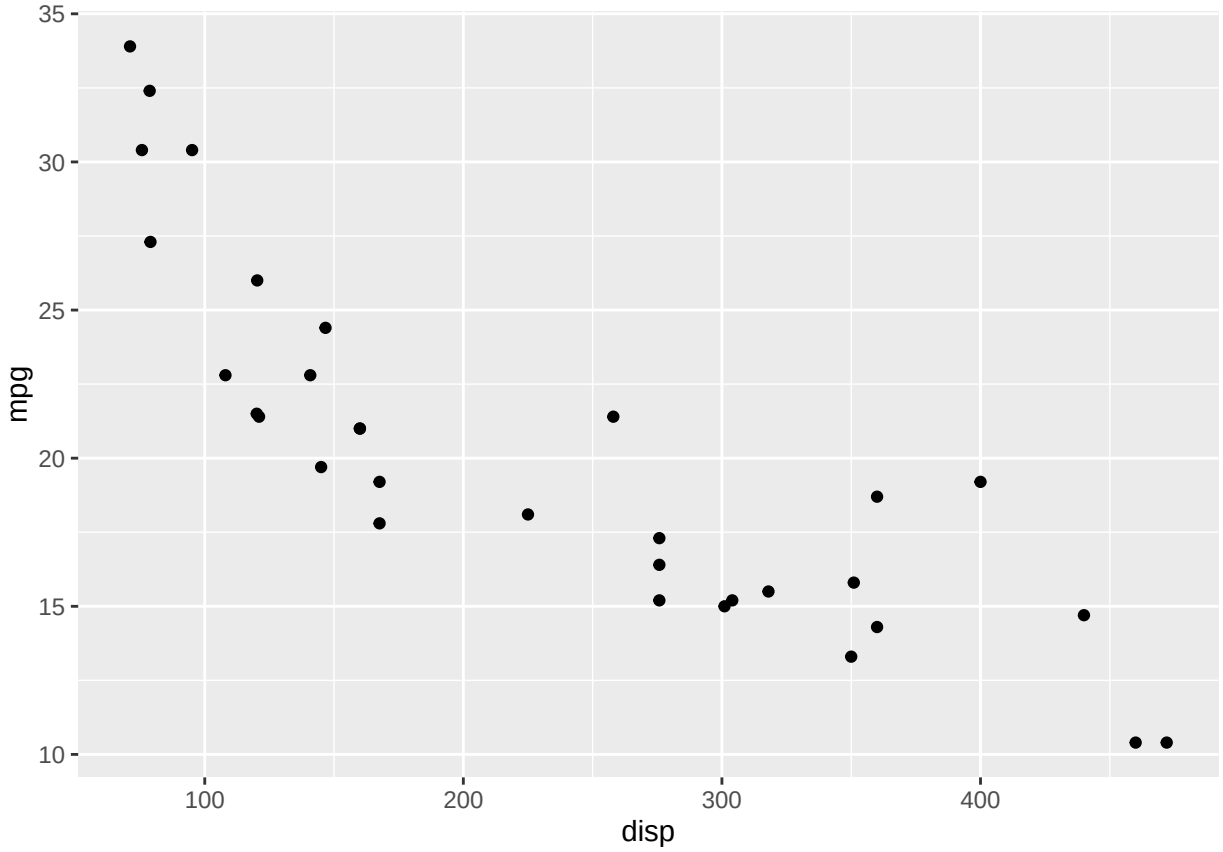
```
head(mtcars)
```

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

2.12.2 As camadas de um gráfico

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

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



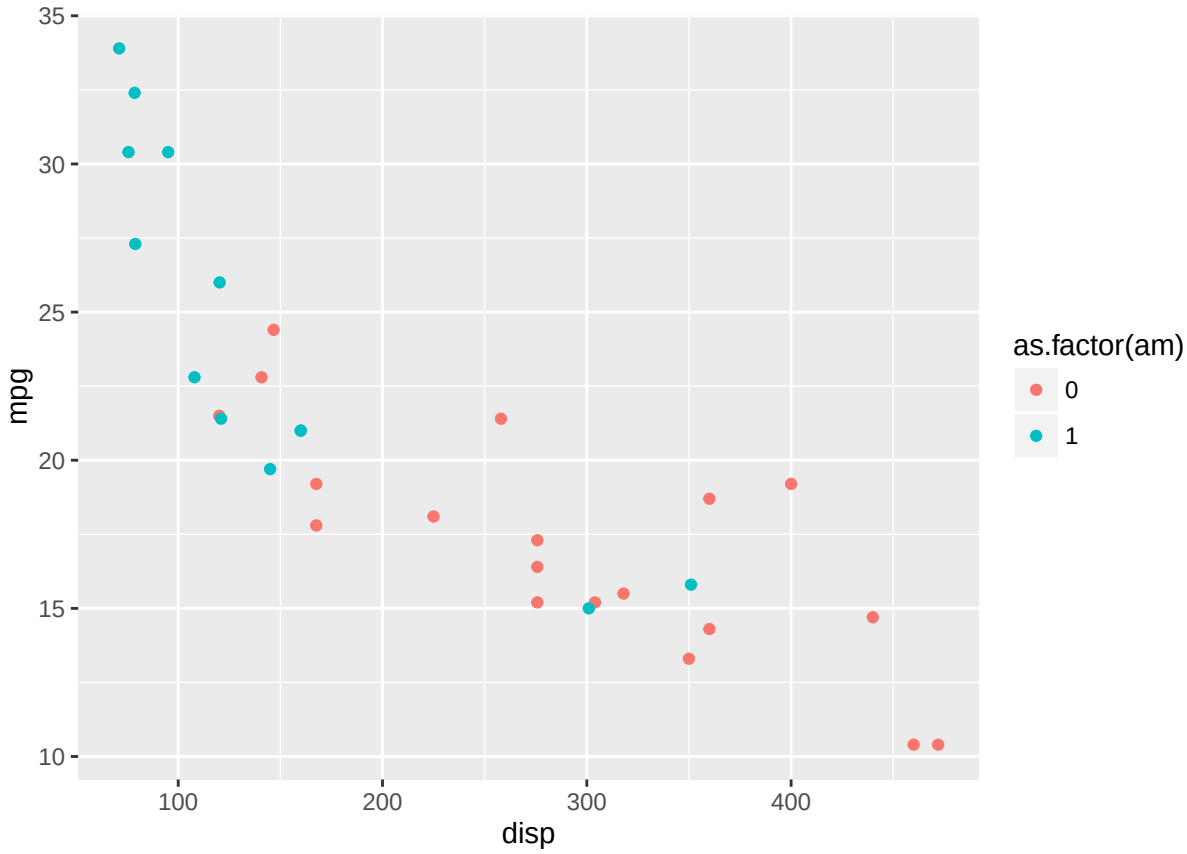
Observe que o primeiro argumento da função ggplot é um data frame. A função aes() descreve como as variáveis são mapeadas em aspectos visuais de formas geométricas definidas pelos *geoms*. Aqui, essas formas geométricas são pontos, selecionados pela função geom_point(), gerando, assim, um gráfico de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

2.12.2.1 Aesthetics

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

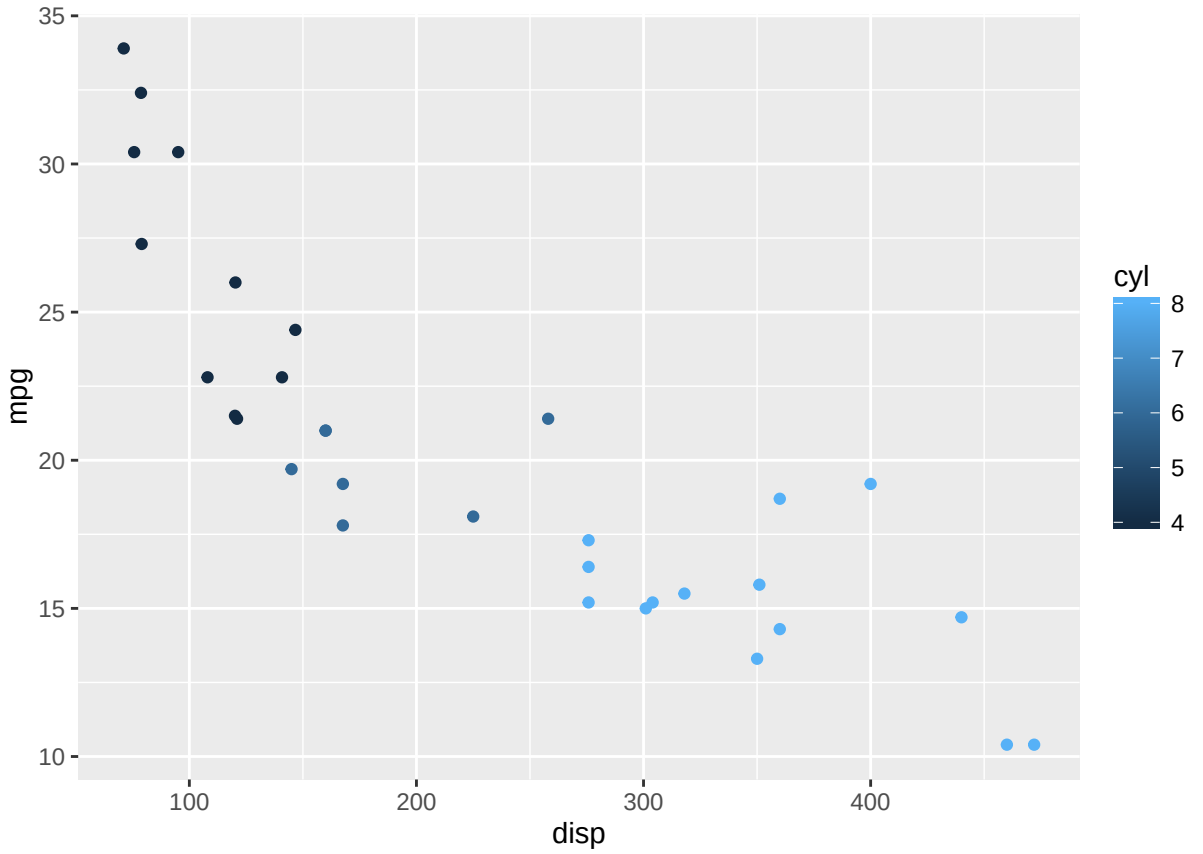
No exemplo acima, atribuímos aspectos de posição: ao eixo y mapeamos a variável mpg (milhas por galão) e ao eixo x a variável disp (cilindradas). Outro aspecto que pode ser mapeado nesse gráfico é a cor dos pontos

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



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

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

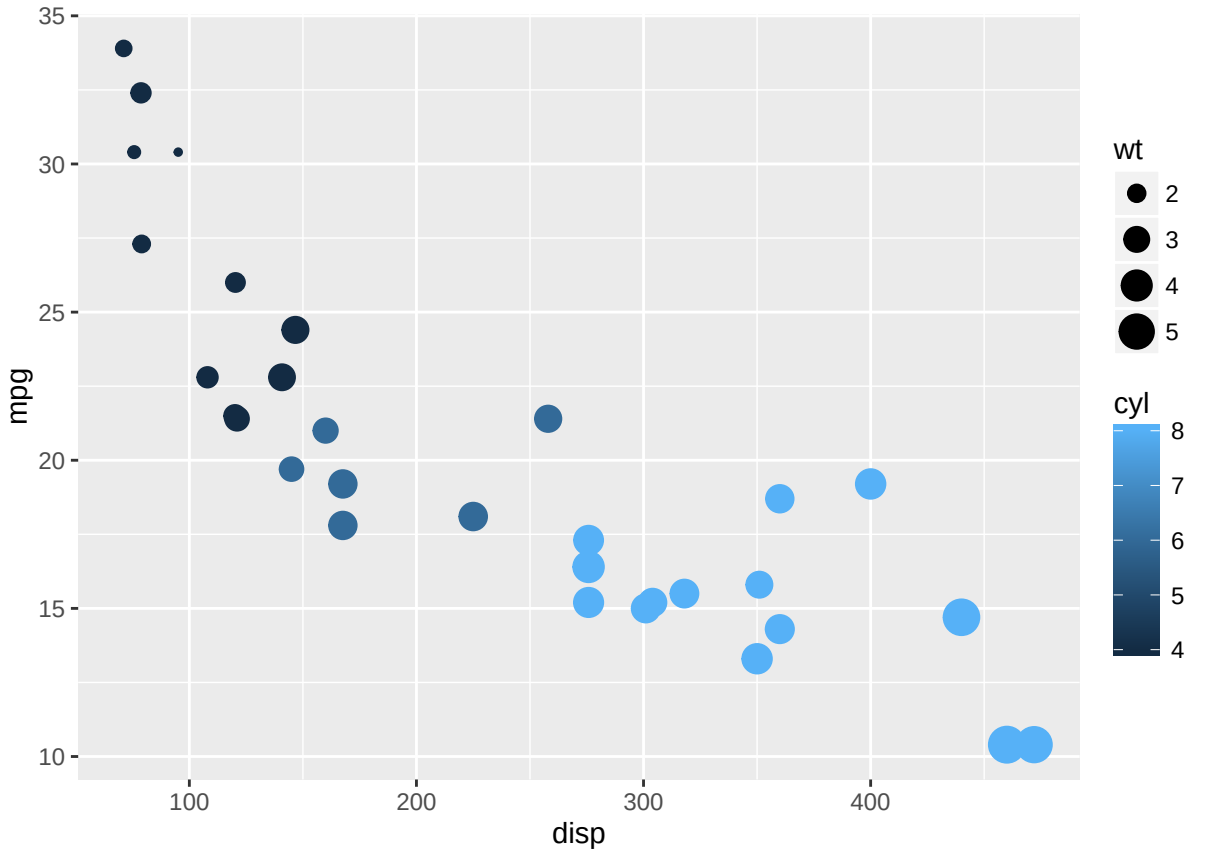


Aqui, o número de cilindros, cyl, é representado pela tonalidade da cor azul.

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

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

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



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

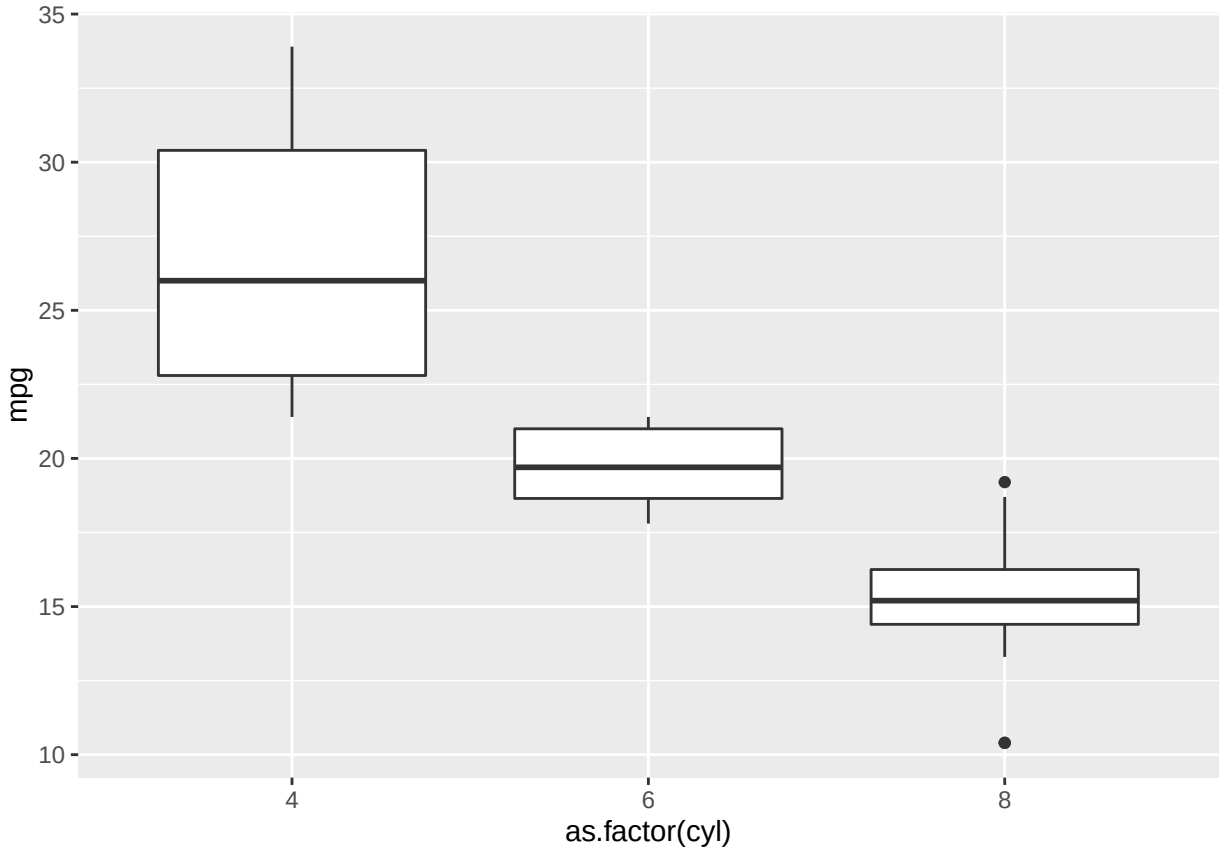
2.12.2.2 Geoms

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

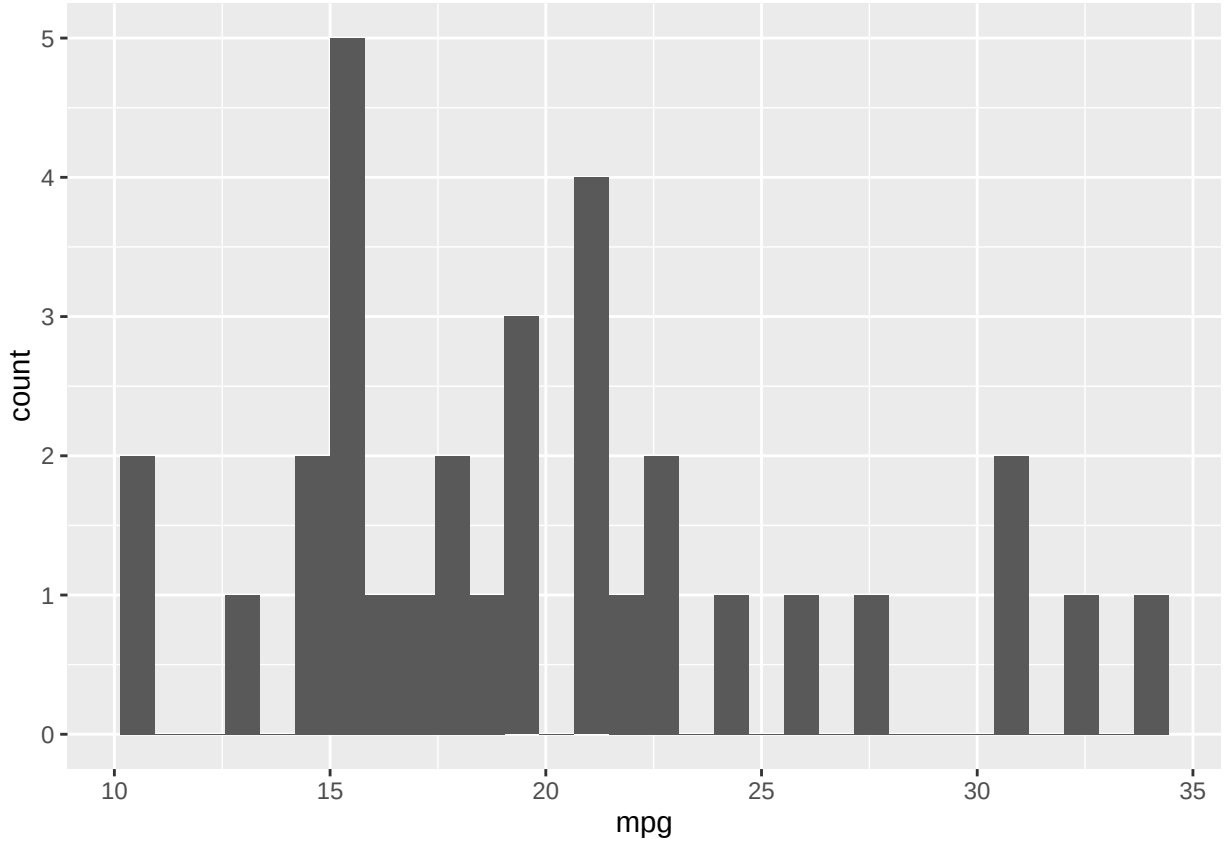
Veja a seguir como é fácil gerar diversos gráficos diferentes utilizando a mesma estrutura do gráfico de dispersão acima:

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

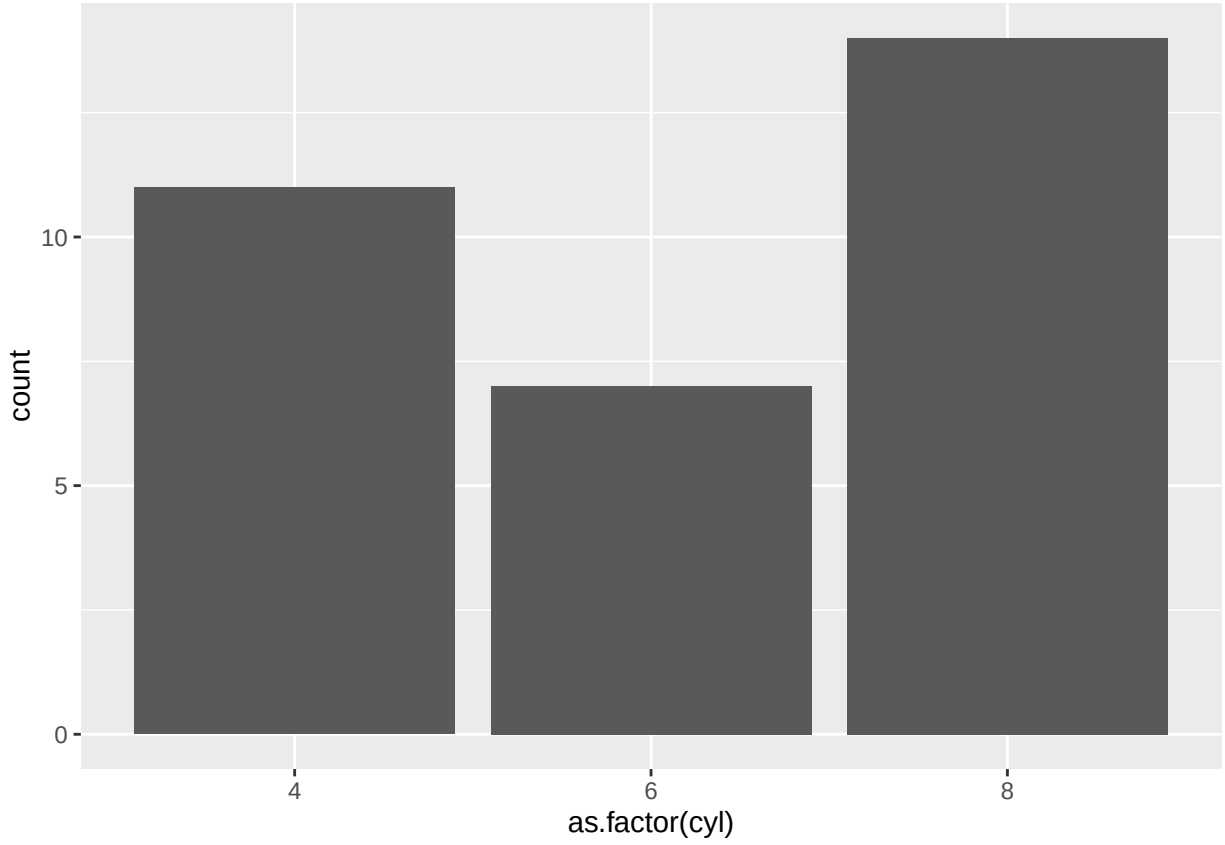



```
ggplot(mtcars, aes(x = mpg)) +
  geom_histogram()
```

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




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

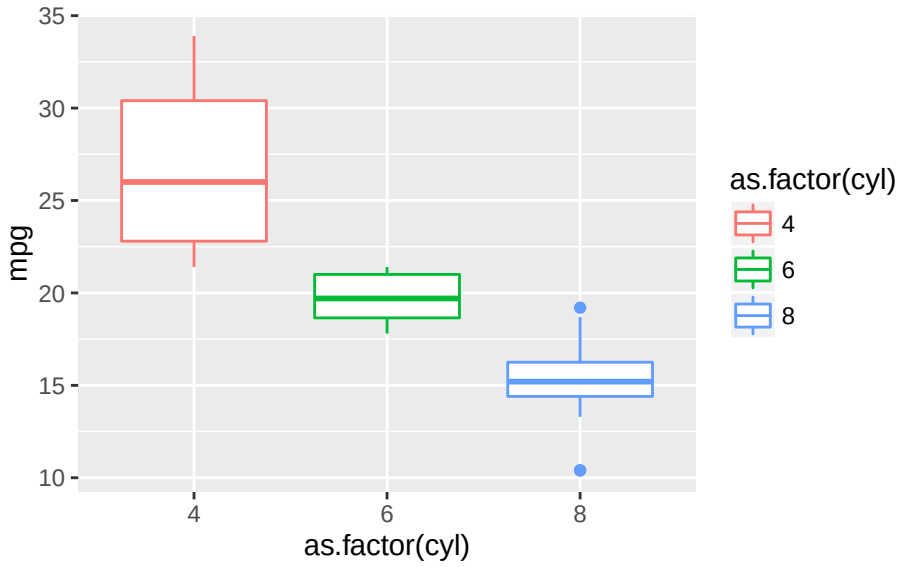


Para fazer um boxplot para cada grupo, precisamos passar para o aspecto x do gráfico uma variável do tipo fator. ### Personalizando os gráficos

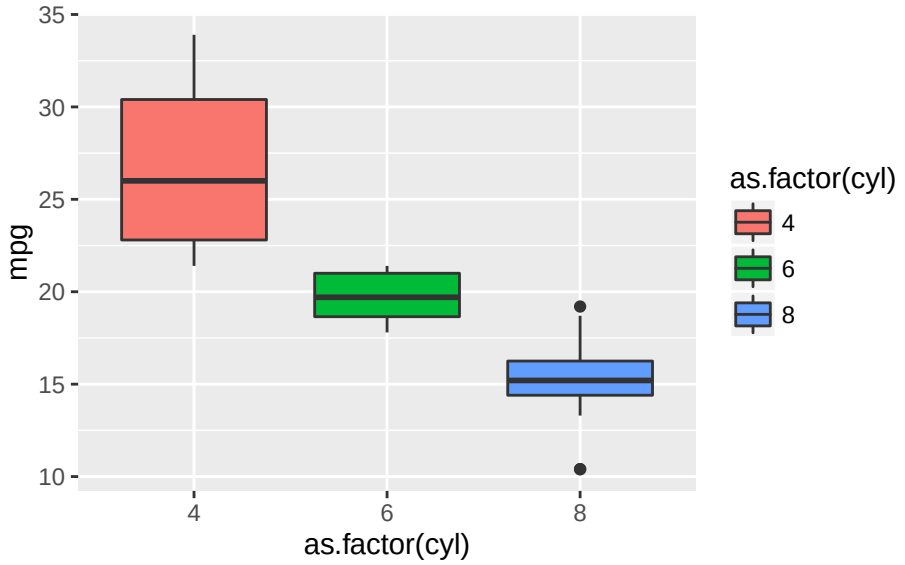
2.12.2.3 Cores

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

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

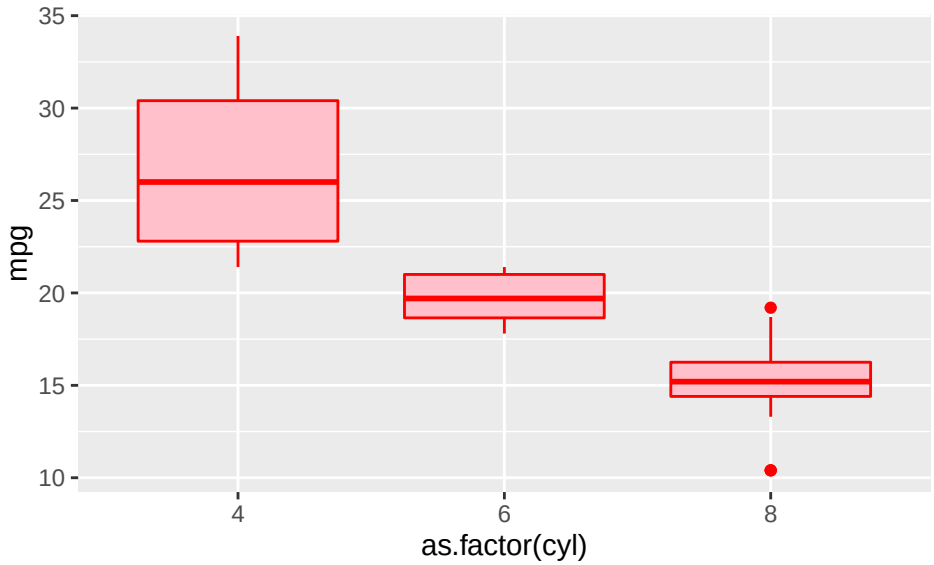



```
ggplot(mtcars, aes(x = as.factor(cyl), y = mpg, fill = as.factor(cyl))) + geom_boxplot()
```



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

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

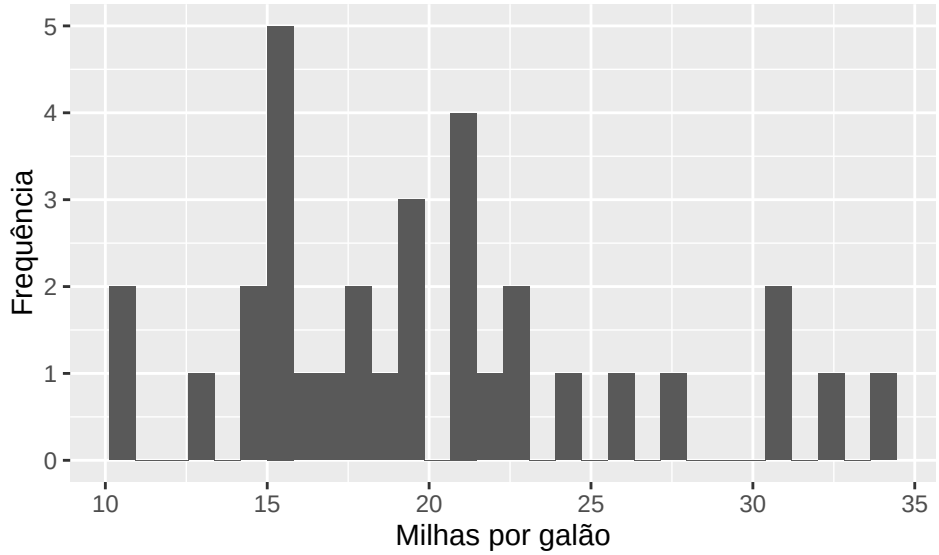


2.12.2.4 Eixos

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

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

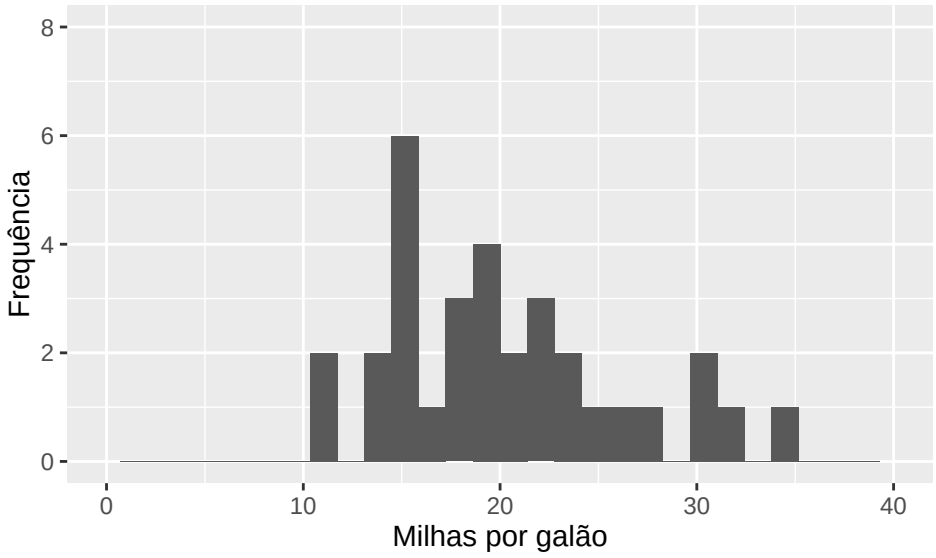
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



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

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

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

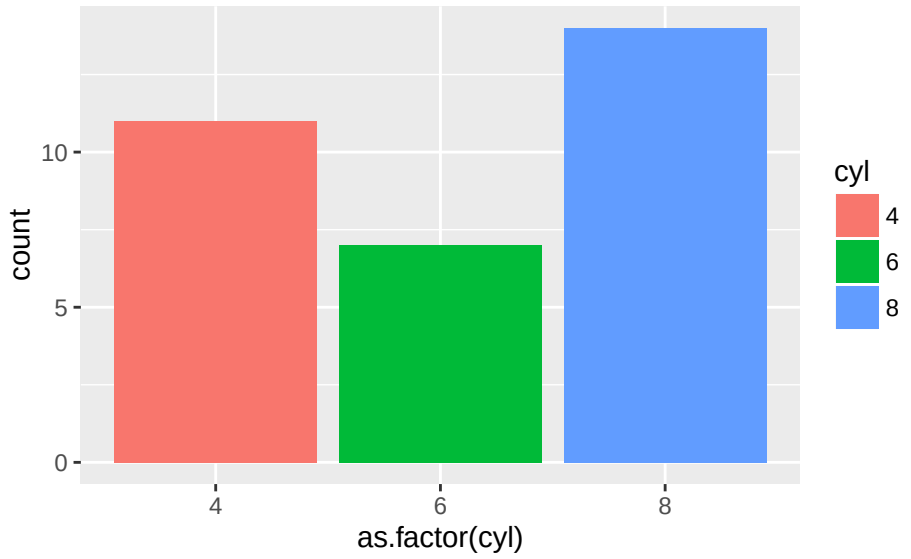


2.12.2.5 Legendas

A legenda de um gráfico pode ser facilmente personalizada.

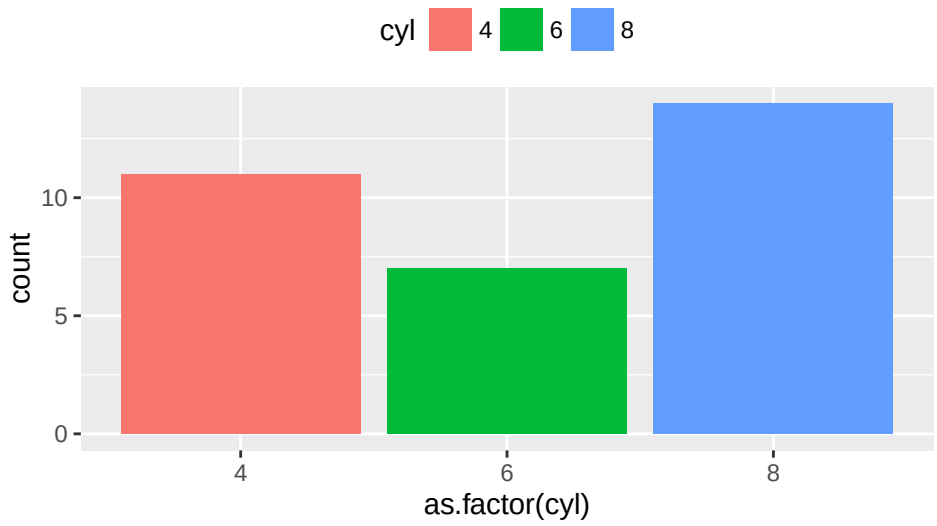
Para trocar o *label* da leganda:

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



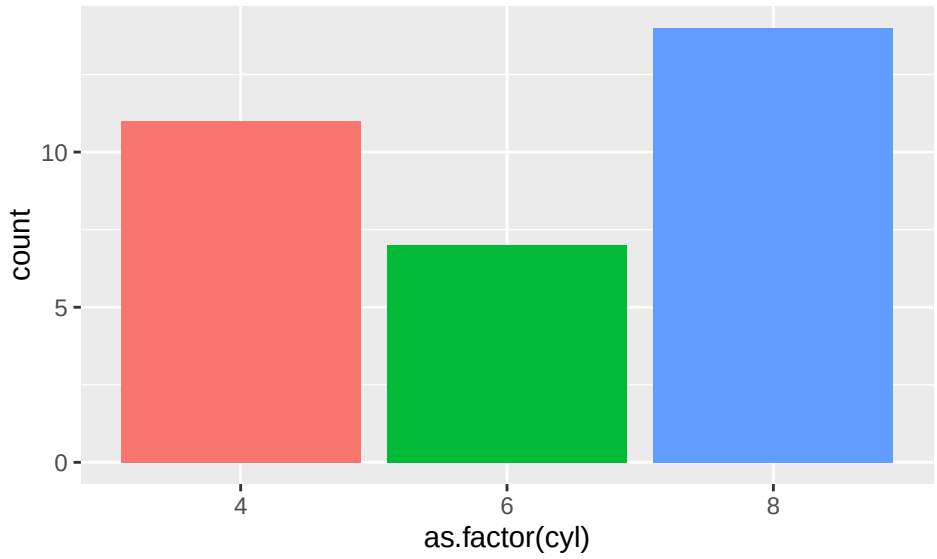
Para trocar a posição da legenda:

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



Para retirar a legenda:

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

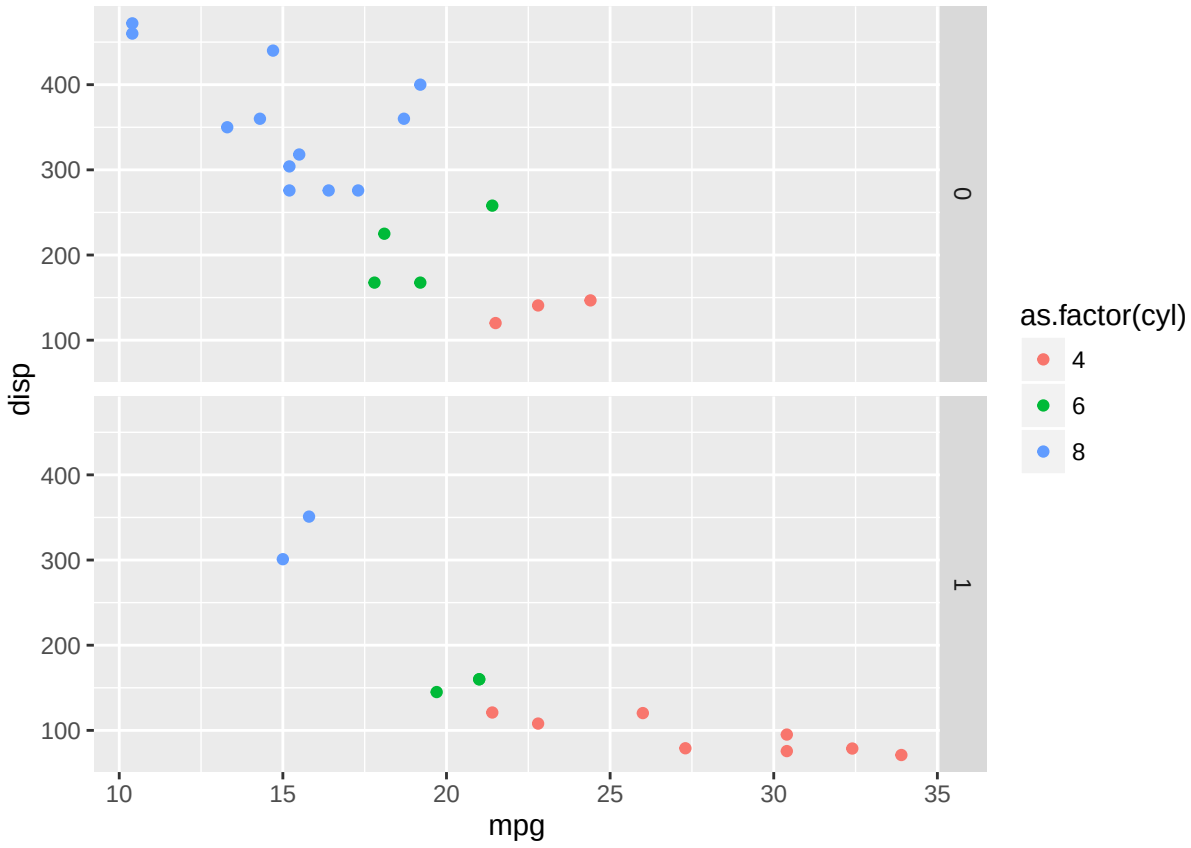


Veja mais opções de personalização aqui!

2.12.2.6 Facets

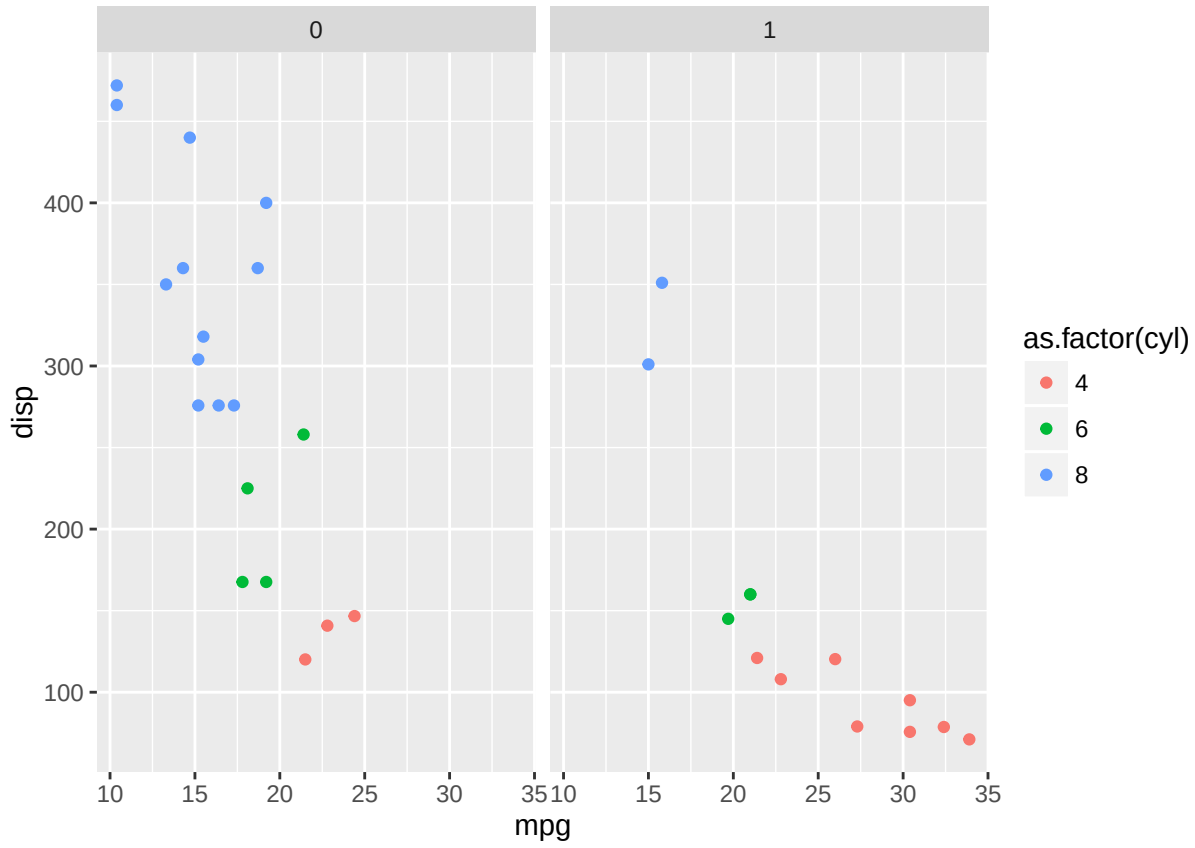
Outra funcionalidade muito importante do ggplot é o uso de *facets*.

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



Podemos colocar os graficos lado a lado também:

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

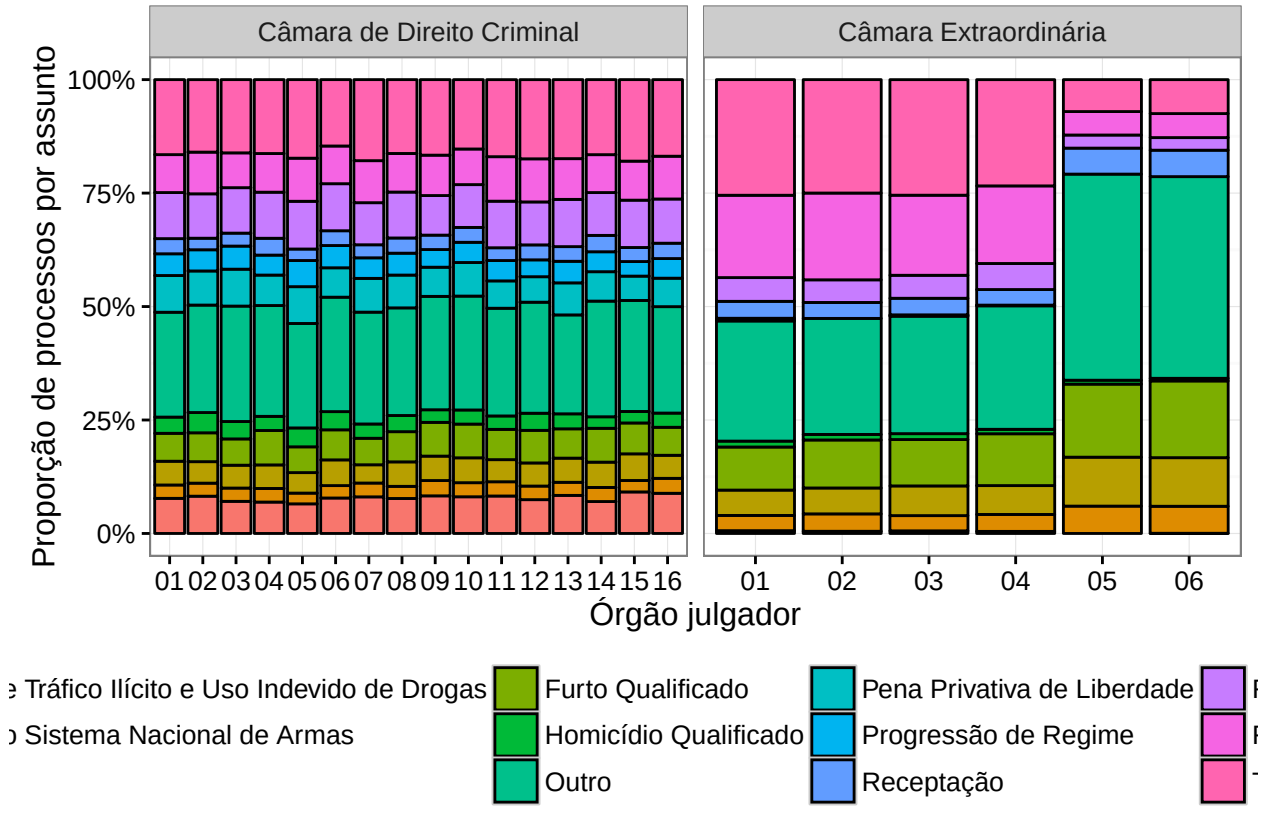


2.13 Exemplo visualização

No exemplo das câmaras, vamos fazer três gráficos. O primeiro mostra a proporção de processos por assunto em cada câmara.

```
d_cjsg %>%
  separate(classe_assunto, c('classe', 'assunto'), sep = ' / ',
           extra = 'merge', fill = 'right') %>%
  group_by(assunto) %>%
  mutate(n_assunto = n()) %>%
  ungroup() %>%
  mutate(assunto = ifelse(n_assunto < 5000, 'Outro', assunto)) %>%
  count(orgao_julgador, assunto) %>%
  mutate(ntot = sum(n), prop = n / ntot) %>%
  ungroup %>%
  filter(ntot > 1000) %>%
```

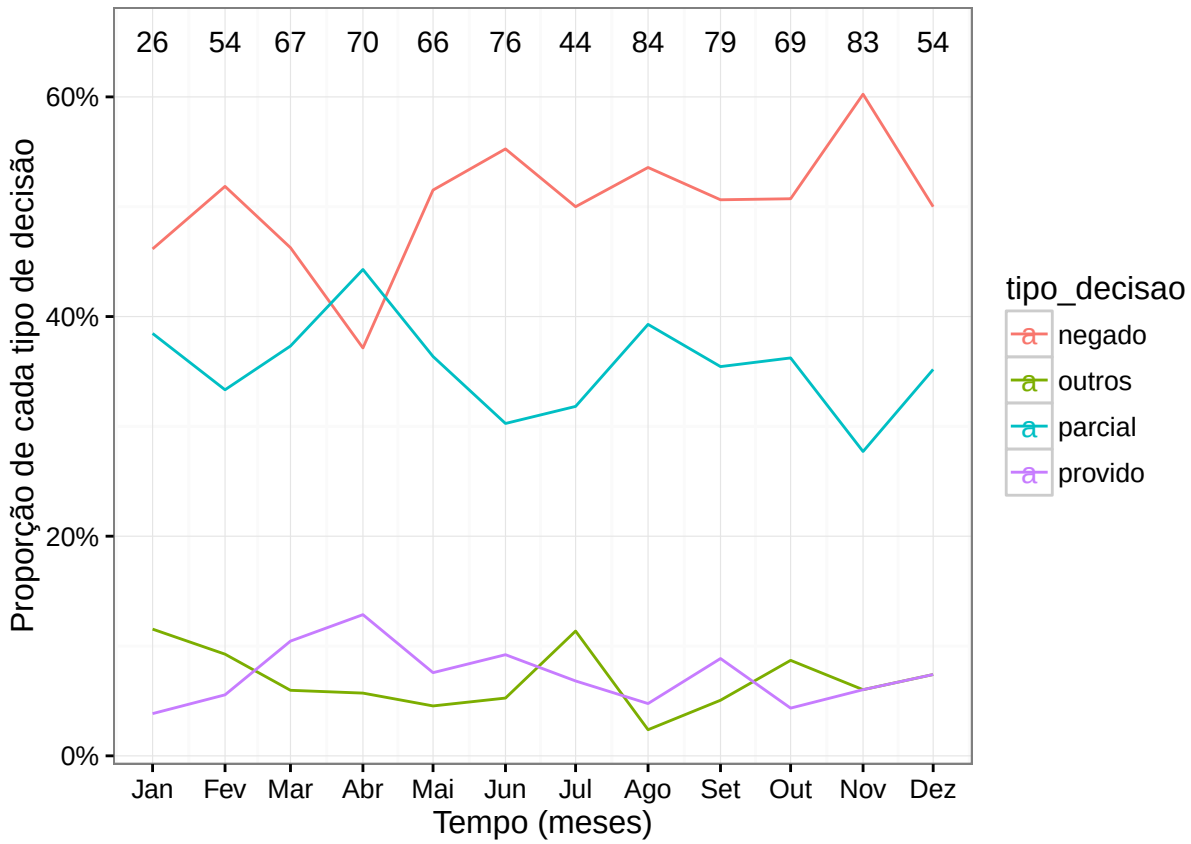
```
  mutate(num = extract_numeric(orgao_julgador),
         num = sprintf('%02d', num)) %>%
  mutate(extra = str_detect(orgao_julgador, 'Extra'),
         extra = ifelse(extra, 'Câmara Extraordinária',
                        'Câmara de Direito Criminal')) %>%
  ggplot(aes(x = num, fill = assunto, y = prop)) +
  geom_bar(stat = 'identity', colour = 'black') +
  facet_wrap(~extra, scales = 'free_x') +
  theme_bw() +
  scale_y_continuous(labels = scales::percent) +
  xlab('Órgão julgador') +
  ylab('Proporção de processos por assunto') +
  theme(legend.position = "bottom")
```



O segundo mostra a proporção de decisões favoráveis no tempo.

```
partes_apelacoes <- d_partes %>%
  filter(tipo == 'apelado', str_detect(nome, '[Mm]inist')) %>%
  mutate(n_processo = str_replace_all(arq, '[^0-9]', '')) %>%
```

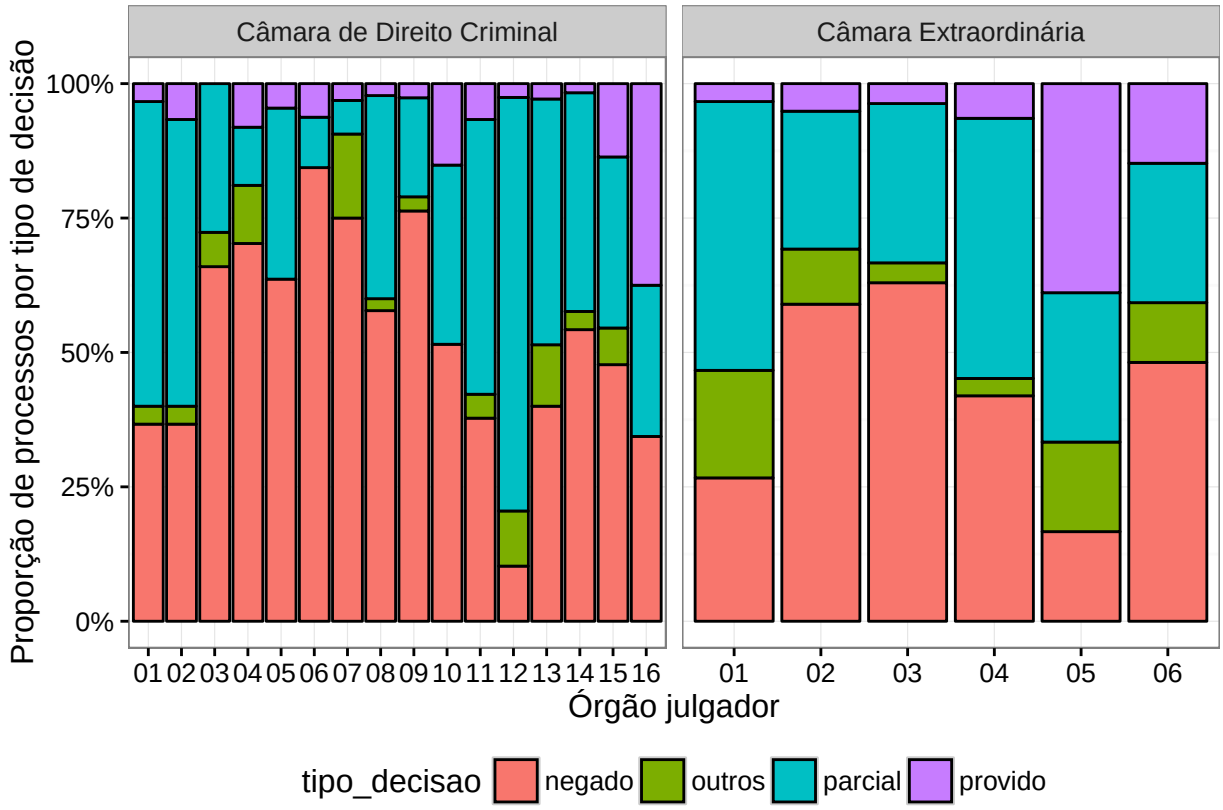
```
  select(n_processo)


decisoes <- d_decisoes %>%
  mutate(n_processo = str_replace_all(arq, '[^0-9]', '')) %>%
  semi_join(partes_apelacoes, 'n_processo') %>%
  filter(situacao == 'Julgado') %>%
  distinct(n_processo, decisao) %>%
  mutate(tipo_decisao = tipos_decisao(decisao)) %>%
  select(n_processo, tipo_decisao)



aux <- d_cjsg %>%
  mutate(n_processo = str_replace_all(n_processo, '[^0-9]', '')) %>%
  inner_join(decisoes, 'n_processo') %>%
  mutate(data = dmy(data_julgamento)) %>%
  mutate(ano_mes = floor_date(data, 'month'))

aux %>%
  count(ano_mes, tipo_decisao) %>%
  mutate(prop = n/sum(n)) %>%
  ungroup %>%
  ggplot(aes(x = ano_mes, y = prop, colour = tipo_decisao)) +
  geom_line() +
  geom_text(aes(y = 0.65, label = n, colour = NULL),
            data = count(aux, ano_mes)) +
  scale_x_date(breaks = scales::date_breaks('1 month'),
               labels = scales::date_format("%b")) +
  scale_y_continuous(labels = scales::percent) +
  xlab('Tempo (meses)') +
  ylab('Proporção de cada tipo de decisão') +
  theme_bw()
```



O terceiro mostra a proporção de cada tipo de decisão em cada câmara.

```
d_cjsg %>%
  mutate(n_processo = str_replace_all(n_processo, '[^0-9]', '')) %>%
  inner_join(decisoes, 'n_processo') %>%
  count(orgao_julgador, tipo_decisao) %>%
  mutate(ntot = sum(n), prop = n / ntot) %>%
  ungroup() %>%
  filter(ntot > 10) %>%
  mutate(num = extract_numeric(orgao_julgador),
         num = sprintf('%02d', num)) %>%
  mutate(extra = str_detect(orgao_julgador, 'Extra'),
         extra = ifelse(extra, 'Câmara Extraordinária',
                        'Câmara de Direito Criminal')) %>%
  ggplot(aes(x = num, fill = tipo_decisao, y = prop)) +
  geom_bar(stat = 'identity', colour = 'black') +
  facet_wrap(~extra, scales = 'free_x') +
```

```
theme_bw() +
scale_y_continuous(labels = scales::percent) +
xlab('Órgão julgador') +
ylab('Proporção de processos por tipo de decisão') +
theme(legend.position = "bottom")
```



2.14 Ferramentas de visualização com shiny

O Shiny é um sistema para desenvolvimento de aplicações web usando o R, um pacote do R (shiny) e um servidor web (shiny server). O Shiny não é uma página web não é um substituto para sistemas mais gerais, como Ruby on Rails e Django e não é uma ferramenta gerencial, como o Tableau.

Para entender sobre Shiny, é necessário entender primeiro o que é server side e user side. Quando surfamos na web, nos *comunicamos* com servidores do mundo inteiro, geralmente através do protocolo HTTP.

No server side, processamos requisições e dados do cliente, estrutura e envia páginas web, interage com banco de dados, etc. Linguagens server side comuns são PHP, C#, Java, R etc (virtualmente qualquer linguagem de programação).

No user side, criamos interfaces gráficas a partir dos códigos recebidos pelo servidor, envia e recebe informações do servidor etc. As "linguagens" mais usuais nesse caso são HTML, CSS e JavaScript.

Mas onde está o Shiny nisso tudo? O código de uma aplicação shiny fica no *server side*. O shiny permite que um computador (servidor) envie páginas web, receba informações do usuário e processe dados, utilizando apenas o R. Para rodar aplicativos shiny, geralmente estruturamos a parte relacionada ao HTML, JavaScript e CSS no arquivo ui.R, e a parte relacionada com processamento de dados e geração de gráficos e análises no arquivo server.R. Os arquivos ui.R e server.R ficam no servidor! Atualmente é possível construir aplicativos em um arquivo só, mas vamos manter a estrutura de ui.R e server.R.

O pacote shiny do R possui internamente um servidor web básico, geralmente utilizado para aplicações locais, permitindo somente uma aplicação por vez. O shiny server é um programa que roda somente em Linux que permite o acesso a múltiplas aplicações simultaneamente.

2.14.1 Começando com um exemplo

```
shiny::runGitHub('abjur/vistemplate', subdir='exemplo_01_helloworld')
```

O Shiny utiliza como padrão o bootstrap css do Twitter, que é bonito e responsivo (lida bem com várias plataformas, como notebook e mobile). Note que criamos páginas básicas com pageWithSidebar. Páginas mais trabalhadas são criadas com fluidPage, fluidRow, column. Pesquise outros tipos de layouts no shiny. É possível criar páginas web customizadas direto no HTML.

Para estudar os *widgets* (entradas de dados para o usuário), acesse este link ou rode

```
shiny::runGitHub('garrettgman/shinyWidgets')
```

2.14.1.1 Exercício

- Criar um pageWithSideBar com dois wellPanel, um dateInput, um checkboxGroup e um textInput.
- Aprender fluidRow e column.

2.14.2 Criando outputs

Imagine que para cada função xxOutput('foo', ...) do ui.R você pode colocar um código do tipo output$foo <- renderXX(...) no server.R. A função no arquivo ui.R determina a localização e identificação do elemento. Crie gráficos com plotOutput e renderPlot e exiba dados com dataTableOutput e renderDataTable.

2.14.2.1 Exercício

- Criar um output de gráfico contento $\mathrm{pairs}(\mathrm{mtcars}[1{:}3])$ e um output de dados contendo $\mathrm{cor}(\mathrm{mtcars}[1{:}3])$.

2.15 Fazendo mais com o shiny

2.15.1 Shiny Server Pro

- Licença comercial do Shiny-server
- Possui algumas características a mais, como autenticação e suporte.

2.15.2 shinyapps.io

- Para compartilhar um aplicativo shiny, geralmente precisamos ter um servidor Linux (geralmente utilizando algum serviço na cloud como AWS ou DigitalOcean) com o shiny server instalado.
- Isso pode ser doloroso.
- O shinyapps.io é um sistema (que envolve tanto pacote do R como uma página web) que permite que o usuário coloque sua aplicação shiny na web sem muito esforço.
- O serviço está sendo desenvolvido pela RStudio Inc. e terá contas grátis e pagas.

2.15.3 Ainda mais!

- Ferramenta em amplo desenvolvimento.
- Grande oportunidade na área acadêmica e profissional.
- Potencial de revolucionar as formas atuais de comunicação.

Chapter 3

Simulação

Esta parte do curso tratará do uso de simulação de eventos discretos e modelagem de fenômenos do mundo real em processos estocásticos. Este tipo de simulação é apenas uma das muitas existentes, considerando que até mesmo experimentos sociais com voluntários podem ser encaixados nessa categoria.

O que motiva a construção e o estudo de simulações no contexto da jurimetria é a natureza complexa do objeto de estudo. Frequentemente, nossas investigações objetivam avaliar o impacto de certa medida, que já ocorreu ou que ainda ocorrerá, ou embasar concretamente uma determinada decisão. Nesses casos, uma saída sofisticada e eficiente para a viabilidade do estudo é a construção de um procedimento computacional que copie as propriedades do sistema de interesse, de forma que seja possível replicar "em laboratório" o que acontece (ou aconteceria) nas situações reais.

Como exemplo de aplicação bem sucedida desta metodologia, podemos citar Allen and Bernshteyn (2008). Neste estudo, conclui-se que, devido à má alocação de máquinas de votação nos Estados Unidos, 20000 votantes deixaram de votar nas eleições de 2008 por conta das filas serem longas demais. Como agravante, verificaram que a maior parte destes votantes seriam afrodescendentes. A má alocação introduziria um viés racial na eleição. Com isso em mente, na elição de 2008, métodos de simulação foram utilizados para alocar as máquinas de votação de forma a minimizar o número de pessoas que deixam de votar.

3.1 Primeiros passos

Segundo Karnon (2012), simulações computacionais são particularmente úteis quando:

1. O problema envolve recursos restritos ou limitados. Embora muitos problemas desse tipo possam ser tratadas utilizando métodos de otimização linear, Bertsimas (1997), ou otimização não linear, Sodolov e

Ismailov (2007), situações muito complicadas podem dificultar a aplicação de técnicas bem consolidadas na literatura.

2. O problema envolve avaliar a interação de muitas variáveis, como por exemplo um conjunto de pessoas numa fila ou andamento de processos judiciais. Também se encaixam nessa categoria problemas com poucas variáveis e relações de dependência complicadas.

3. O problema envolve considerar a evolução no tempo de um conjunto de variáveis muito dependentes.

Uma vez que decide-se usar uma simulação para resolver um problema, o próximo passo é modelá-lo de forma que a simulação seja possível. Esta fase é muito importante pois nela são feitas suposições que tornam viável o modelo de simulação. Suposições mal escolhidas implicam em resultados que não representam bem a situação modelada. Na seção seguinte, descreveremos um conjunto de passos que podem auxiliar este procedimento.

3.2 Estruturação

A construção de um modelo para simulação pode ser feita respondendo às seguintes perguntas:

1. Quais são as quantidades de interesse?
2. Quais são as informações necessárias para calcular as quantidades de interesse?
3. Qual a relação entre elas?
4. As quantidades de interesse variam no tempo? Como?

É importante observar que as respostas às perguntas acima não são independentes e não é necessário evitar repetições nas respostas. Na verdade, é interessante que as respostas sejam bem completas, fornecendo mais insumos à modelagem.

Para simplificar, vamos chamar as quantidades listadas no item 1 de variáveis. As informações necessárias para o cálculo de uma variável, quantidades listadas no item 2, podem depender, ou não, de uma outra variável. Quando não dependem, chamaremos esta informação de parâmetro. Quanto uma variável é necessária para o cálculo de outra, precisaremos notificar esta dependência no item 3 e incluí-la no nosso modelo na forma de uma equação ou algoritmo.

A distinção entre parâmetros e variáveis é importante pois, no geral, gostaríamos de realizar as simulações para verificar o efeito da variação de um parâmetro (quantidade que não depende de nenhuma outra informação) numa variável (quantidade que precisa de outras para ser calculada).

Por serem muito abertas, as respostas à essas perguntas podem tomar muito tempo. Podemos construir modelos super complexos, o que torna todas as fases subsequentes mais difíceis, ou simplicar o problema intro-

duzindo muitas hipóteses, o que pode diminuir a precisão dos resultados. Conclui-se, então, que modelar de maneira simples e eficiente é uma tarefa que exige muito conhecimento sobre o assunto estudado, pois depende de compreender quais informações são importantes e quais são supérfluas. Por isso, é importante conhecer muitos estudos sobre a questão de interesse antes de prosseguir com a construção de um modelo.

As respostas às perguntas 3. e 4. auxiliam na construção dos procedimentos que conduzem a evolução das variáveis. Esse procedimento de "evolução" pode ser pensado para acontecer no tempo, como por exemplo um número de processos que aumenta ou diminui ao longo dos anos, ou num determinado instante, como quando cálculos complicados ou sorteios são realizados utilizando como insumo um conjunto de variáveis ou de parâmetros.

Por fim, complementando a pergunta 3., é comum que representem relações ou dependências entre as variáveis/parâmetros através de equações ou distribuições conjuntas de probabilidade.

3.3 Dados

Para maior precisão e confiabilidade da simulações, é recomendado o uso de séries históricas ou outras fontes de dados como insumos.

Por exemplo, um certo parâmetro descrito na fase de estruturação do modelo pode ter seu valor aproximado utilizando um conjunto de observações passadas. Em outras situações, pode ser interessante inserir quantidades aleatórias para considerar variabilidades intrínsecas em certos fenômenos. Neste caso, a distribuição das quantidades aleatórias pode ser obtida a partir dos dados reais.

3.4 Implementação

Nesta fase, o resultado das seções anteriores é traduzido num programa de computador que efetivamente produzirá as simulações.

Não é necessário ater-se aos detalhes da delimitação de variáveis e parâmetros quando planeja-se o programa. Basta que todas as informações necessárias para análise estejam disponíveis em algum formato. Por exemplo, se uma quantidade de interesse é uma contagem de processos, mas os processos estão guardados na coluna de uma tabela, não há necessidade de guardar o número de linhas em uma nova variável.

O principal cuidado a ser tomado durante a implementação é garantir que a produção dos cenários de interesse seja realizada de forma simples e em tempo hábil, possibilitando a avaliação desejada.

3.5 Análise dos resultados

Uma vez que o procedimento computacional foi implementado, chega a hora de analisar os resultados produzidos pelas simulações. Antes de prosseguir, é interessante responder às seguintes questões:

1. Desejamos analisar o impacto de quais parâmetros?
2. Quais são as variáveis sobre as quais desejamos analisar o impacto?
3. O que a alteração dos parâmetros de interesse deve causar em cada variável?

A resposta à essas perguntas é importante pois fornece diretrizes para a elaboração de um relatório de pesquisa pós simulação. Em linhas gerais, desejamos avaliar, com relação às variáveis listadas no item 2, simulações que modifiquem os valores para os parâmetros listados no item 1. Essa avaliação deve ser feita à luz do que se espera obter, informação descrita no item 3.

O principal cuidado a se tomar nesta fase de análise dos resultados deve-se ao fato de, em algumas situações, alguns parâmetros inseridos no modelo servirem apenas para deixá-lo mais verossímil, sem que se tenha interesse em analisar o seu impacto. Nessa situação, é interessante caracterizar o impacto dos parâmetros de interesse eliminando a parte deste que pode ser atribuída aos parâmetros que não importam.

Essa "eliminação" é feita verificando qual se o mesmo efeito de um parâmetro "relevante" é observado para vários valores de parâmetros "irrelevantes".

3.6 Exemplo - Cadastro Nacional de Adoção

Nesta sessão, estudaremos um modelo de simulação proposto pela Associação Brasileira de Jurimetria num relatório da série Justiça Pesquisa sobre os tempos de processos de adoção no Brasil.

O modelo em questão tinha como interesse estudar o impacto de uma redução na duração de processos de adoção no Cadastro Nacional de Adoção (CNA) do Conselho Nacional de Justiça.

Argumenta-se que, caso ocorresse uma diminuição na duração dos processos de adoção, a idade de entrada nas crianças no CNA também diminuiria. Como a maior parte dos pretendentes prefere adotar crianças mais jovens, o número de crianças adotadas tende a aumentar. Além disso, espera-se obter uma diminuição no número de crianças que atingem a maioridade.

A partir do contexto descrito acima, a modelagem desse problema pode ser realizada seguindo o roteiro proposto anteriormente.

3.6.1 Primeiros passos

Primeiramente justificamos o uso de um modelo de simulação, já que o problema envolve analisar o que acontece com o CNA no decorrer tempo. Além disso, como número de crianças adotadas depende de pareamentos entre pretendentes e crianças disponíveis no CNA, podemos estudar o impacto de diferentes estratégias de pareamento realizando poucas alterações no procedimento de simulação, o que torna essa abordagem computacionalmente atraente.

3.6.2 Estruturação

A segunda parte da modelagem consiste na resposta do questionário proposto anteriormente:

1. Quais são as quantidades de interesse?

O número de crianças disponíveis no CNA e suas respectivas idades, o número de crianças que atingem a maioridade, o número de crianças adotadas num determinado período e o número de pretendentes à adoção e as idades máximas preferidas por cada um deles.

2. Quais são as informações necessárias para calcular as quantidades de interesse?

O número de crianças que atingem a maioridade pode ser calculado a partir da idade das crianças cadastradas no período anterior.

O número de crianças adotadas pode ser obtido comparando as idades das crianças do período anterior com as idades máximas preferidas por cada pretendente no período anterior.

O número de pretendentes e suas preferências dependem da quantidade de pretendentes que se cadastram periodicamente no CNA e da distribuição desses novos pretendentes com relação à idades máximas preferidas.

O número de crianças disponíveis no CNA depende do número de crianças que foram adotadas, do número de crianças que são cadastradas periodicamente no CNA e da distribuição de idade dessas novas crianças.

4. As quantidades de interesse variam no tempo? Como?

A simulação precisa atualizar os seus valores periodicamente, de forma que o período seguinte utilize informação de período anterior.

Os períodos analisados podem ser anos ou semestres.

3. Qual a relação entre as quantidades de interesse?

Vamos definir, para cada período t, as seguintes quantidades:

$N(t) =$ número de crianças disponíveis para adoção no instante t

$M(t) =$ número de crianças adotadas no instante t

$D(t) =$ número de crianças que atingiram a maioridade no instante t

$K(t) =$ número de crianças que entram no cadastro no instante t

$P(t) =$ número de pretendentes cadastrados no instante t

$P_i(t) =$ número de pretendentes cadastrados no instante t

que preferem crianças de idade até i

$N_i(t) =$ número de crianças cadastradas de idade

$$0 \leq i < 18$$

A relação fundamental entre essas quantidades é

$$N(t) = N(t-1) - M(t-1) - D(t-1) + K(t)$$

$$N(t) = \sum_{i=0}^{17} N_i(t)$$

$$P(t) = \sum_{i=0}^{17} P_i(t)$$

O número de adoções $M(t)$ é calculado utilizando as idades das crianças adotadas, $N_i(t)$, e as preferências dos pretendentes cadastrados no sistema, $M_i(t)$, através de uma estratégia de pareamento.

Para viabilizar a aplicação do modelo vamos fazer algumas hipóteses sobre as quantidades descritas acima:

1. $K(t)$ é dado por uma constante K.

2. A distribuição de idades das $K(t)$ crianças é a mesma encontrada nos dados.
3. $P(t)$ é dado por uma constante P.
4. A distribuição de preferências dos $P(t)$ pretendentes é a mesma encontrada nos dados.
5. A estratégia de pareamento de crianças disponíveis e pretendentes é maximizar o número de adotados priorizando crianças mais velhas.

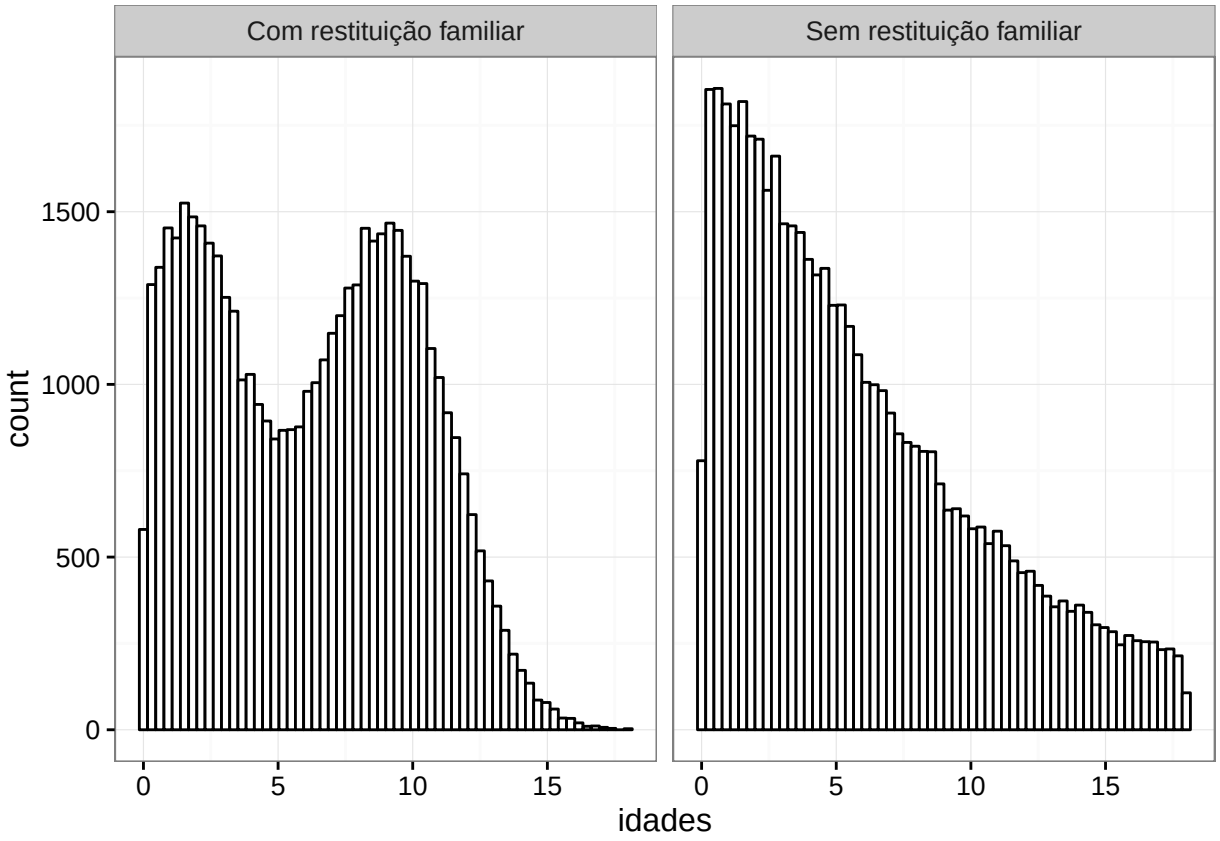
3.6.3 Dados

Anteriormente citamos que a distribuição de idades e de preferências seria obtida através de uma análise dos dados.

A distribuição observada de preferências de idades máximas, segundo um levantamento da ABJ, está descrita na tabela abaixo.

Idade	Proporção
0	14.78%
1	18.33%
2	19.74%
3	18.79%
4	10.56%
5	9.81%
6	3.62%
7	1.76%
8	0.95%
9	0.32%
10	0.66%
11	0.15%
12	0.2%
13	0.07%
14	0.05%
15	0.06%
16	0.03%
17	0.12%

Outra importante informação obtida através da análise de dados é a distribuição de idade das crianças cadastradas no CNA. Analisando a idade de entrada dos cadastrados, obtivemos as seguintes distribuições:



Note que, como cada uma destas distriuições está associada a uma causa de cadastro no CNA, vamos incluir a proporção de crianças registradas devido a processos de restituição familiar como um parâmetro do modelo. Por simplicidade, fixaremos este valor em 15%, equivalente à proporção observada no relatório supra citado.

O impacto de variações na distribuição de idades das crianças cadastradas no CNA será resumido no parâmetro que controla a localização da segunda "corcova" da distribuição de idades das crianças cadastradas no CNA após processos com restituição familiar.

Por fim, precisamos fixar os valores de K e P. Para isso, fixaremos a razão $\frac{K}{P}$ em $3,52$, pois este número representa a razão do número total de crianças cadastradas no CNA, ativas ou inativas, pelo número total de pretendentes cadastrados no CNA, ativos ou inativos. Como $P = 3,52K$, vamos nos preocupar apenas variar o parâmetro K.

3.6.4 Implementação

As simulações propriamente ditas serão realizadas utilizando programas desenvolvidos no software R.

Em linhas gerais, a implementação utilizará dois vetores. Um deles conterá as idades das crianças cadastradas e o outro conterá as idades máximas preferidas por cada pretendente cadastrado. A cada iteração, o vetor das crianças é atualizado retirando as crianças que foram adotadas ou que atingiram a maioridade e adicionando as novas crianças cadastradas no CNA. O vetor dos pretendentes é atualizado retirando aqueles que adotaram alguma criança e adicionando nos novos cadastrados.

Primeiramente, construíremos funções que sorteiem as idades das crianças que são registradas no CNA e preferências dos pretendenentes registrados no CNA.

```
#Sorteia n_pretendentes a partir de uma distribuição desejada. Por default, utiliza a distribuição disponível no relatório so

distr <- c(0.1478,0.1833,0.1974,0.1879,0.1056,0.0981,0.0362,
           0.0176,0.0095,0.0032,0.0066,0.0015,0.0020,0.0007,
           0.0005,0.0006,0.0003,0.0012)
sorteia_preferencias <- function(n_pretendentes, distribuicao = distr){
  sapply(runif(n_pretendentes), function(x) {which.max(x < distribuicao) - 1})
}

#Sorteia n valores de uma mistura de normais limitada ao intervalo de 0 a 18.

tnorMix <- function(n,fit){
  x <- nor1mix::rnorMix(n,fit)
  while(x < 0 | x > 18){
    x <- nor1mix::rnorMix(n,fit)
  }
  return(x)
}

#Sorteia n valores de uma distribuição gama limitada ao intervalo de 0 a 18.

trgamma <- function(n,shape,rate){
  x <- rgamma(n,shape,rate)
  while(x < 0 | x > 18){
    x <- rgamma(n,shape,rate)
  }
```

```
  return(x)
}

#Sorteia as idades de n_criancas_para_adocao a partir das distribuições acima, utilizando um conjunto de parâmetros.

sorteia_idades <- function(n_criancas_para_adocao, shape, rate, mu1, mu2, sigma, p, peso = 0.5){
  distr <- nor1mix::norMix(mu = c(mu1,mu2), sigma = rep(sigma,2), w = c(1-peso,peso))
  sapply(runif(n_criancas_para_adocao), function(x){ifelse(x < p, ifelse(runif(1) < 0.5,trgamma(1,shape,rate),rexp(1)), tn
}
```

A partir dessas funções, podemos inicializar os nossos vetores:

As listas inicializadas seguem abaixo

Table 3.2: Exemplo de conjunto de crianças simulado

idades	id
0.2006772	1
9.7013161	2
3.6083768	3
2.0634004	4
4.1576919	5
1.5176869	6
5.6406676	7
3.2431550	8
5.6340124	9
4.5060209	10

Table 3.3: Exemplo de conjunto de pretendentes simulado

idade_maxima_preferida	id
0	1
0	2
0	3
0	4

idade_maxima_preferida	id
0	5
0	6
0	7
0	8
0	9
0	10

Para completar a inicialização e finalizar a implementação, precisamos desenvolver um algoritmo que maximize o número de matchs a partir de um conjunto de idades e preferências. O algoritmo abaixo cumpre esse papel, realizando uma varredura das idades ordenadas para checar a viabilidade da adoção de cada criança.

```
# Arredonda a idade das crianças cadastradas para baixo e ordena as idades arredondadas em ordem decrescente.
criancas_floor <- floor(sort(idades, decreasing = T))

# Ordena as idades máximas preferidas de cada pretendente em ordem decrescente.
pretendentes <- sort(pretendentes, decreasing = T)

# Ordena a idade das crianças cadastradas em ordem decrescente.
criancas <- sort(idades, decreasing = T)

# Contador que percorre o vetor de idades.
i = 1

# Inicialização do número de pareamentos.
num_match = 0

while(i <= length(criancas)){

  #Se a idade arredondada da criança de maior idade for menor que a maior idade máxima tolerada por um pretendente,
  if(criancas_floor[i] <= pretendentes[1]){

    #Remove a criança adotada do vetor de crianças
    criancas <- criancas[-i]
```

```
    #Remove a criança adotada do vetor de idades arredondadas
    criancas_floor <- criancas_floor[-i]

    #Remove o pretendente que adotou a criança dos pretendentes disponíveis
    pretendentes <- pretendentes[-1]

    #Conta um novo pareamento
    num_match = num_match + 1

    #O contador recua uma posição por conta da remoção da criança adotada
    i <- i-1
  }

  #Continua a contagem
  i <- i + 1
}
```

Na verdade, este algoritmo pode ser melhorado se realizarmos uma adaptação na regra de pareamento. Não é necessário que a criança de maior idade seja pareada com o pretendente de maior idade máxima tolerada. Embora seja improvável, essa regra de pareamento pode produzir a adoção de uma criança de 10 anos por um pretendente que não se opõe a adotar jovens de até 17 anos. De certa forma, este tipo de pareamento é um "desperdício", já que pretendentes com idades máximas toleradas grandes são mais raros, de forma que é mais interessante pareá-los com jovens mais velhos.

A adaptação sugerida no parágrafo anterior pode ser implementada da forma que segue, notando que, na função, as duas estratégias estão disponíveis na simulação através do parâmetro "tipo".

```
matching <- function(pretendentes, criancas, tipo = 1){

  criancas_floor <- floor(sort(criancas, decreasing = T))
  pretendentes <- sort(pretendentes, decreasing = T)
  criancas <- sort(criancas, decreasing = T)

  i = 1
  num_match = 0
```

```
  while(i <= length(criancas)){

    if(criancas_floor[i] <= pretendentes[1]){

      j = 1

      while(tipo == 2 & j != length(pretendentes) & criancas_floor[i]<=pretendentes[ifelse(j < length(pretendentes), j+1, le
        j = j + 1
      }

      criancas <- criancas[-i]
      criancas_floor <- criancas_floor[-i]

      pretendentes <- pretendentes[-j]
      num_match = num_match + 1
      i <- i-1
    }
    i <- i + 1
  }

  return(list(criancas,pretendentes, num_match))
}
```

O exemplo abaixo ilustra o funcionamento dessas estratégias de pareamento.

```
p <- sorteia_preferencias(30)
idades <- sorteia_idades(30, 1.1, 0.15, 1, 9, 2.8, 0.1, 0.5)

print(sort(round(idades, 2), decreasing = T))
```

```
##  [1] 13.22 12.96 12.72 12.41 10.82 10.44  9.79  9.30  8.93  8.71  7.30
## [12]  7.13  7.03  6.66  6.59  6.51  5.29  4.98  4.66  3.55  3.32  3.01
## [23]  2.72  2.14  1.31  1.21  1.09  0.65  0.42  0.42
```

```
print(sort(round(p, 2), decreasing = T))
```

```
##  [1] 2 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
#Lista contendo o vetor de crianças e de pretendentes restantes.
pareamentos_estrategia_1 <- matching(p, idades, 1)
pareamentos_estrategia_2 <- matching(p, idades, 2)


print(pareamentos_estrategia_1)
```

```
## [[1]]
##  [1] 13.220144 12.963619 12.720426 12.409301 10.816369 10.435139  9.794466
##  [8]  9.298955  8.934183  8.711584  7.295204  7.132229  7.025862  6.656854
## [15]  6.590421  6.507732  5.285900  4.983965  4.661280  3.550157  3.320593
## [22]  3.011672  2.139547
##
## [[2]]
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##
## [[3]]
## [1] 7
```

```
print(pareamentos_estrategia_2)
```

```
## [[1]]
##  [1] 13.220144 12.963619 12.720426 12.409301 10.816369 10.435139  9.794466
##  [8]  9.298955  8.934183  8.711584  7.295204  7.132229  7.025862  6.656854
## [15]  6.590421  6.507732  5.285900  4.983965  4.661280  3.550157  3.320593
## [22]  3.011672  2.139547
##
## [[2]]
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##
## [[3]]
## [1] 7
```

3.6.5 Análise dos resultados

Antes de proceder com a análise de resultados vamos responder às perguntas propostas anteriormente:

1. Desejamos analisar o impacto de quais parâmetros?

   A idade de entrada das crianças no CNA e as estratégias de pareamento de crianças e pretendentes.

2. Quais são as variáveis sobre as quais desejamos analisar o impacto?

   Desejamos a variação do número de crianças que atingem a maioridade ao longo do tempo, o número de crianças e pretendentes disponíveis e o número de adoções.

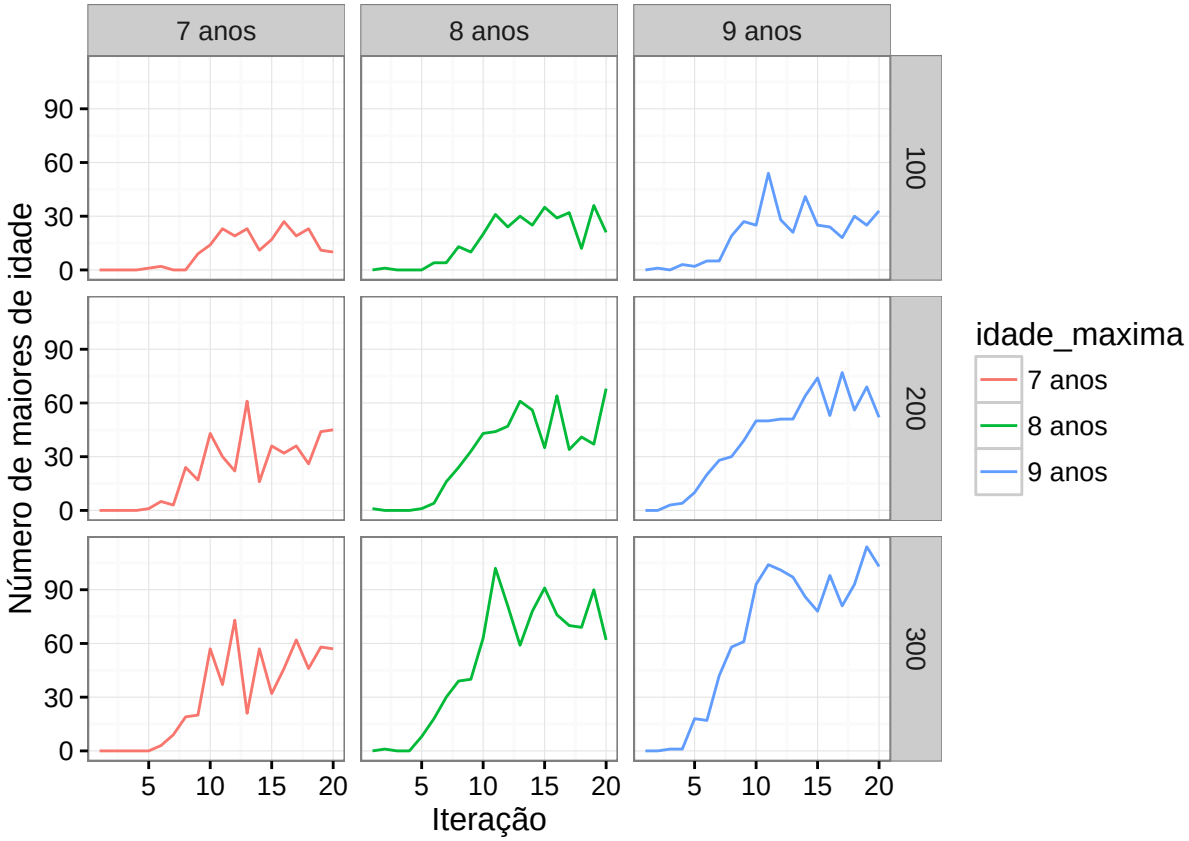
3. O que a alteração dos parâmetros de interesse deve causar em cada variável?

   Espera-se que uma menor idade de regitro no CNA diminua o número de crianças que atingem a maioridade, aumente o número de adoções e, consequentemente, diminuia o número de crianças disponíveis para adoção. Temos interesse especial em avaliar o tamanho dessas diminuições/acréscimos.

3.6.5.1 Idade de entrada no CNA

Desejamos checar se, conforme a idade de entrada no CNA diminui, o número de maiores de idade por iteração fica menor. Este é o caso, como se observa na figura abaixo.

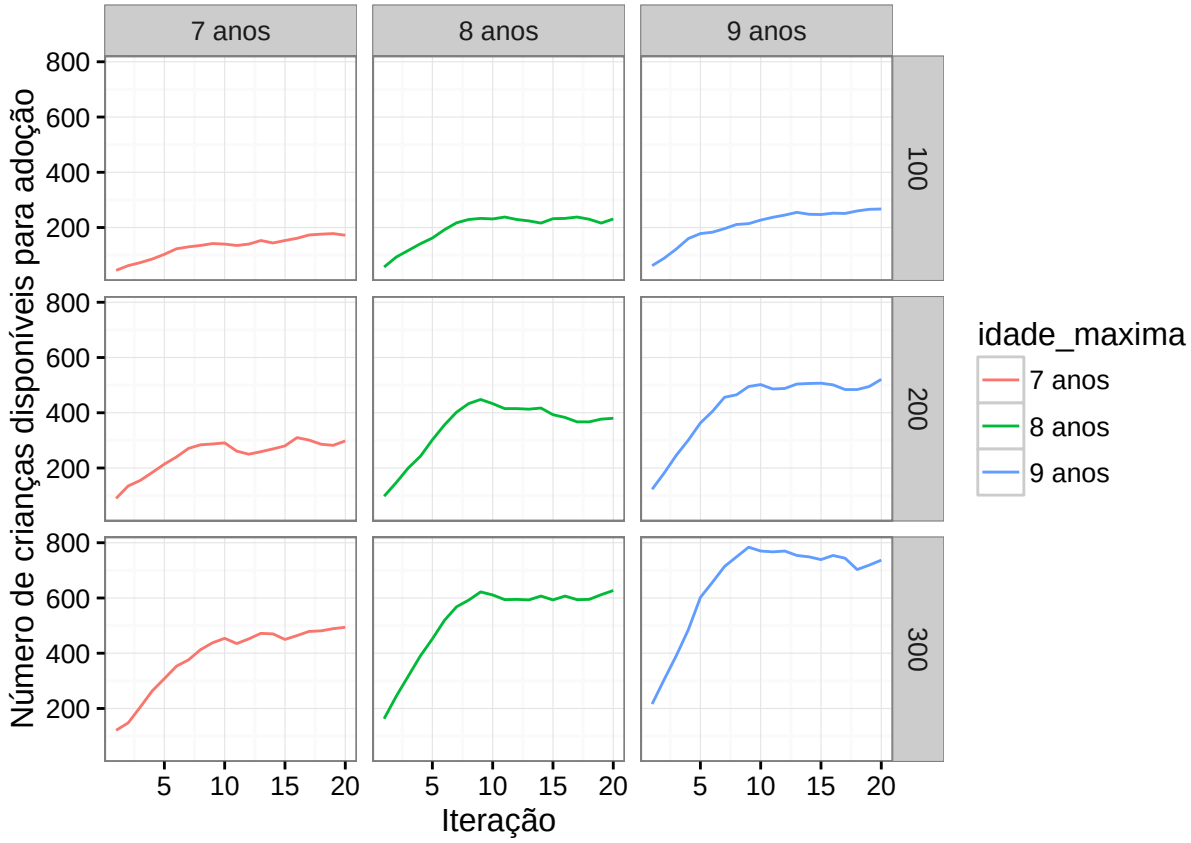
```
maiores_de_idade <- function(tipo = 1, tempos = 1:10, K = 100, p_cada_tipo = 0.1, unidade = 1, idade_maxima = 9){
   realiza_processo(tipo, tempos, K, p_cada_tipo, unidade, idade_maxima)$maiores_de_idade
}


expand.grid(K = c(100,200,300), idade_maxima = 7:9) %>%    plyr:::mdply(maiores_de_idade, tempo = 1:20) %>%
 reshape2::melt(id.vars = c('K','idade_maxima')) %>%
 select(-variable) %>%
 group_by(K, idade_maxima) %>%
 mutate(periodo = 1:n()) %>%
 ungroup() %>%
 mutate(idade_maxima = paste0(idade_maxima,' anos')) %>%
 ggplot(aes(x = periodo, y = value, color = idade_maxima))+
  geom_line()+
  facet_grid(K ~ idade_maxima)+
 theme_bw()+
 xlab('Iteração')+
 ylab('Número de maiores de idade')
```



A diminuição no número de maiores de idade implica em mais crianças sendo adotadas ao longo do tempo, de forma que devemos verificar uma diminuição nesta variável também. O resultado deste teste segue na figura abaixo.

```
criancas_disponiveis <- function(tipo = 1, tempos = 1:10, K = 100, p_cada_tipo = 0.1, unidade = 1, idade_maxima = 9)
    realiza_processo(tipo, tempos, K, p_cada_tipo, unidade, idade_maxima)$criancas_disponiveis
}


expand.grid(K = c(100,200,300), idade_maxima = 7:9) %>%    plyr:::mdply(criancas_disponiveis, tempo = 1:20) %>%
  reshape2::melt(id.vars = c('K','idade_maxima')) %>%
  select(-variable) %>%
  group_by(K, idade_maxima) %>%
  mutate(periodo = 1:n()) %>%
  ungroup() %>%
  mutate(idade_maxima = paste0(idade_maxima,' anos')) %>%
  ggplot(aes(x = periodo, y = value, color = idade_maxima))+
    geom_line()+
```

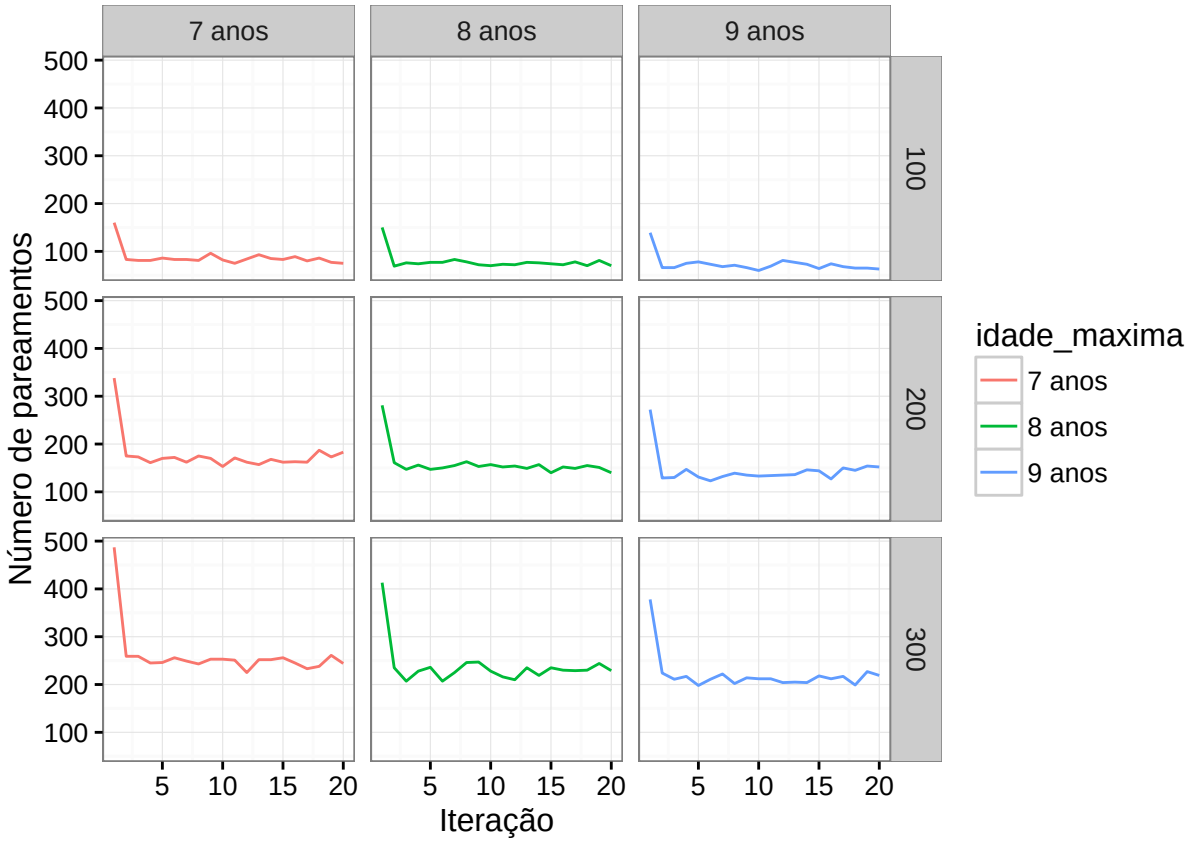
```
  facet_grid(K ~ idade_maxima)+
theme_bw()+
xlab('Iteração')+
ylab('Número de crianças disponíveis para adoção')
```



Por fim, uma diminuição na idade das crianças cadastradas no CNA deve favorecer a ocorrência de um maior número de pareamentos entre pretendentes e crianças, considerando que grande parte dos pretendentes prefere crianças mais jovens.

```
numero_de_pareamentos <- function(tipo = 1, tempos = 1:10, K = 100, p_cada_tipo = 0.1, unidade = 1, idade_maxima
  realiza_processo(tipo, tempos, K, p_cada_tipo, unidade, idade_maxima)$numero_de_pareamentos
}


expand.grid(K = c(100,200,300), idade_maxima = 7:9) %>%  plyr:::mdply(numero_de_pareamentos, tempo = 1:20) %>
 reshape2::melt(id.vars = c('K','idade_maxima')) %>%
 select(-variable) %>%
 group_by(K, idade_maxima) %>%
 mutate(periodo = 1:n()) %>%
```

```
ungroup() %>%
mutate(idade_maxima = paste0(idade_maxima,' anos')) %>%
ggplot(aes(x = periodo, y = value, color = idade_maxima))+
  geom_line()+
  facet_grid(K ~ idade_maxima)+
theme_bw()+
xlab('Iteração')+
ylab('Número de pareamentos')
```

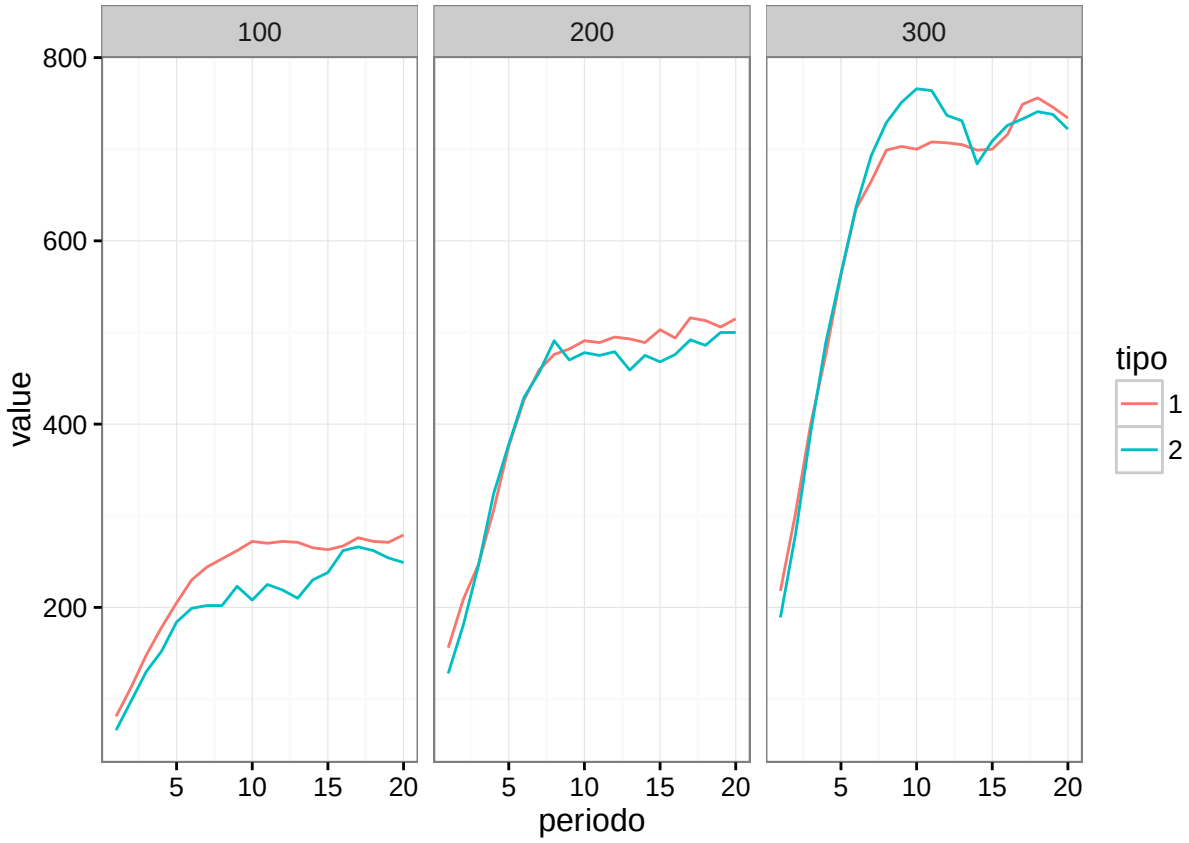


3.6.5.2 Estratégia utilizada

O número de pareamentos certamente é invariante com relação à estratégia utilizada, já que a diferença entre os dois métodos está apenas em qual pretendente será escolhido.

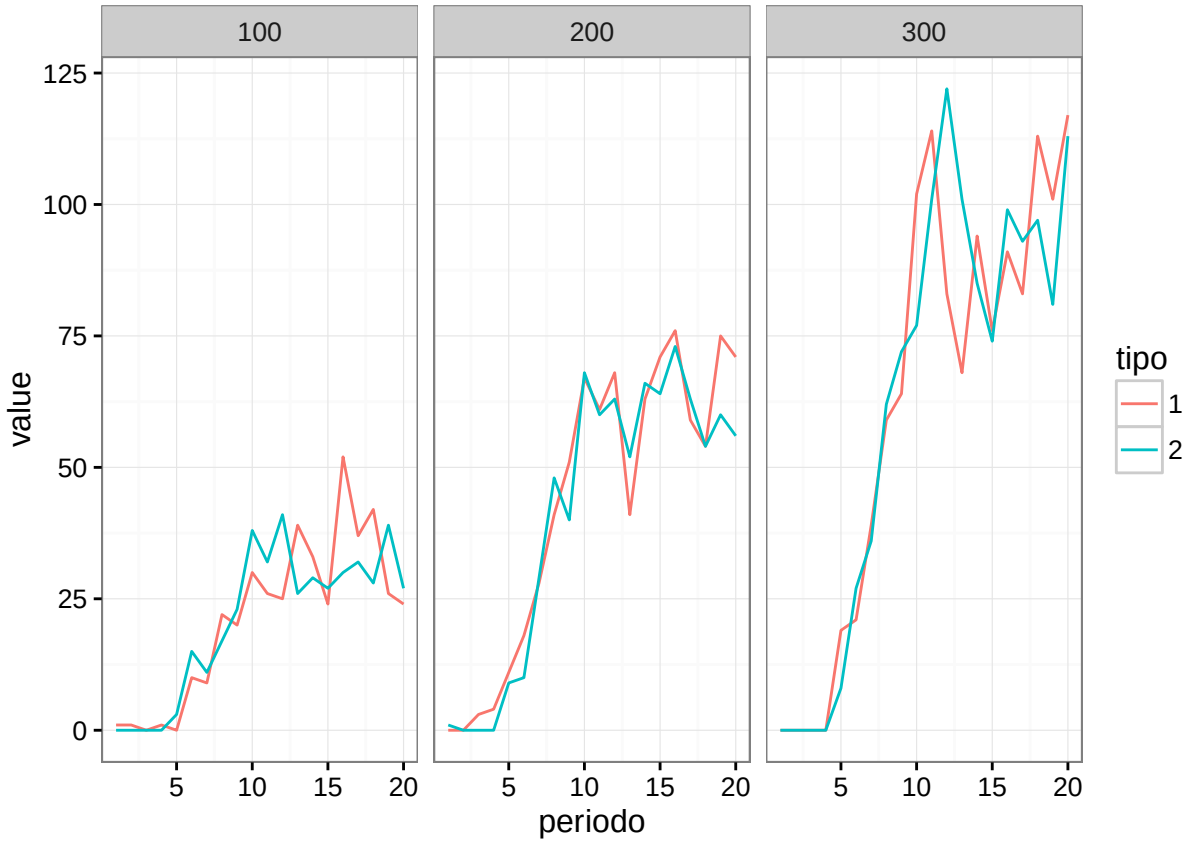
Por outro lado, o número de crianças disponíveis pode diminuir conforme pretendentes menos restritivos forem sendo preteridos. A figura abaixo sugere que este efeito não é sentido. Isso deve-se, provavelmente, a distância entre a idade máxima tolerada pelos pretendentes e as idades das crianças cadastradas.

```
expand.grid(tipo = 1:2 , K = c(100,200,300)) %>%    plyr:::mdply(criancas_disponiveis, tempo = 1:20) %>%
  reshape2::melt(id.vars = c('tipo','K')) %>%
  select(-variable) %>%
  group_by(tipo, K) %>%
  mutate(periodo = 1:n()) %>%
  ungroup() %>%
  mutate(tipo = factor(tipo)) %>%
  ggplot(aes(x = periodo, y = value, fill = tipo, color = tipo))+
  geom_line()+
    facet_wrap(~K)+
  theme_bw()
```




```
expand.grid(tipo = 1:2 , K = c(100,200,300)) %>%    plyr:::mdply(maiores_de_idade, tempo = 1:20) %>%
  reshape2::melt(id.vars = c('tipo','K')) %>%
  select(-variable) %>%
  group_by(tipo, K) %>%
  mutate(periodo = 1:n()) %>%
```

```
ungroup() %>%
mutate(tipo = factor(tipo)) %>%
ggplot(aes(x = periodo, y = value, fill = tipo, color = tipo))+
geom_line()+
  facet_wrap(~K)+
theme_bw()
```



Chapter 4

Referências

ocite

Allen TT, Bernshteyn M (2008) Helping Franklin county vote in 2008: Waiting line report to Michael Stinziano and Matthew Damschroder and The Franklin County Board of Elections http://vote.franklincountyohio.gov/assets/pdf/press-releases/PR-07302008.pdf

Karnon J, Stahl J, Brennan A, et al. Modeling using discrete event simulation: a report of the ISPOR-SMDM Modeling Good Research Practices Task Force—4. Med Decis Making. 2012; 32(5):701–11.

Dimitris Bertsimas and John N. Tsitsiklis (1997), Introduction to Linear Optimization, Athena Scientific.

Alexey Izmailov and Mikhail Solodov (2007). Optimization, Volume 2: Computational Methods. Rio de Janeiro, Brazil. Second Edition

Bibliography