

Text as Data para Ciências Sociais
guiia prático com aplicações

Davi Moreira

Criado em 02-08-2019. Atualizado em: 09-09-2020

Contents

Prefácio	5
Objetivo	5
Sobre o autor/organizador	5
Licença	6
Agradecimentos	6
1 Introdução	7
1.1 O R e o RStudio	7
1.2 O Pacote <code>txt4cs</code> e outros	9
1.3 Material de apoio	10
2 Text as data: o texto como dado	11
2.1 Panorama da área	11
2.2 Oportunidades	11
2.3 Quadro geral de metodologias	15
2.4 O processo de análise do texto como dado	16
3 R e o Processamento de Linguagem Natural	19
3.1 Encoding - Codificação de caracteres	19
3.2 Encoding para remover acentos	20
4 Strings no R	21
4.1 Strings e vetores	22
4.2 Strings e matrizes	25
4.3 Strings e data.frames	26
4.4 Strings e listas	26
4.5 Processamento básico	27
4.6 O pacote <code>stringr</code>	29
4.7 Regular Expressions no R	31
5 Obtenção de conteúdo	35
5.1 word, excel ou .pdf	35
5.2 Webscraping	36
5.3 Web Services	39

5.4	Download de arquivos da web	45
5.5	Twitter	47
5.6	Imagens	47
5.7	Áudio Transcrição	49
6	Processamento dos dados	51
6.1	Tokens	51
6.2	Corpus	51
6.3	Tokens e Corpus	51
6.4	DFM: Matriz de documentos e termos	51
6.5	Stemming	51
6.6	FCM: Matriz de co-ocorrência de termos	51
7	Mineração e estatísticas básicas	53
7.1	Análise de frequênciā	53
7.2	Nuvem de palavras	53
7.3	tf-idf	53
7.4	Rede de n-grams	53
7.5	Correlação pareada	53
7.6	Diversidade lexical	53
7.7	Similaridade entre documentos/termos	53
7.8	KEYNESS: Análise de Frequênciā Relativa	53
8	Escalonamento	55
8.1	Wordscore	55
8.2	Wordfish	55
9	Classificação	57
9.1	Método de dicionário: Análise de sentimento	57
9.2	Naive Bayes	57
9.3	LDA: Latent Dirichlet Allocation	57
9.4	STM: Structed Topic Model	57

Prefácio

txt4cs|

A partir da produção de material para o curso *Text as Data: análise automatizada de conteúdo* que ministrei no [MQ-UFMG](#) em 2019 e no artigo que publiquei em coautoria com [Maurício Izumi](#) (Izumi and Moreira, 2018), esse livro tem como propósito difundir nas ciências sociais e humanidades técnicas e métodos de análise automatizada de conteúdo usando a linguagem R.

Objetivo

O principal objetivo do livro é ser tutorial prático de uso e aplicação de técnicas e métodos de análise automatizada de conteúdo na língua portuguesa através da linguagem R .

Sobre o autor/organizador

Davi Moreira é Professor Adjunto da Escola de Relações Internacionais da Fundação Getulio Vargas. Ph.D. em Ciência Política pela Universidade de São Paulo (USP) e vencedor do Prêmio CAPES de tese 2017 na área de Ciência Política e Relações Internacionais. Atua nas seguintes áreas: políticas públicas, estudos legislativos, métodos quantitativos em ciências sociais e análise automatizada de conteúdo.

Para mais informações:

- [Página pessoal.](#)
- [GitHub](#)
- [Google Scholar.](#)

Licença

Este livro é distribuído de acordo com a licença Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License ([CC BY-NC-SA 4.0](#)).

Agradecimentos

Agradeço aos organizadores do MQ-UFGM 2019 pela oportunidade de ministrar o curso e assim me estimular a empreender esse projeto. Também agradeço aos amigos [Manoel Galdino](#), [Rafael Magalhães](#), Lincon Ribeiro e [Umberto Mignozzetti](#) pelo apoio e incentivo ao longo de toda minha trajetória como cientista. Por fim, agradeço à fantástica cientista de dados [Mônica Rocabado](#), sem a qual esse projeto continuaria na gaveta.

Este livro é escrito com o uso do pacote **bookdown** ([Xie, 2019](#)), através do R **Markdown** e **knitr** ([Xie, 2015](#)).

Chapter 1

Introdução

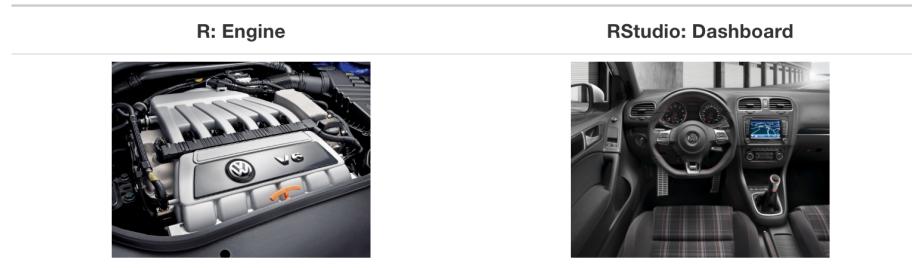
1.1 O R e o RStudio

Com o objetivo de ser um tutorial prático de uso e aplicação de técnicas e métodos de análise automatizada de conteúdo para ciências sociais e humanidades este livro fará uso da linguagem R.

R é uma linguagem de programação e também um ambiente de desenvolvimento integrado para cálculos estatísticos e gráficos. Ele pode ser facilmente instalado através do link: <https://cran.r-project.org/>.

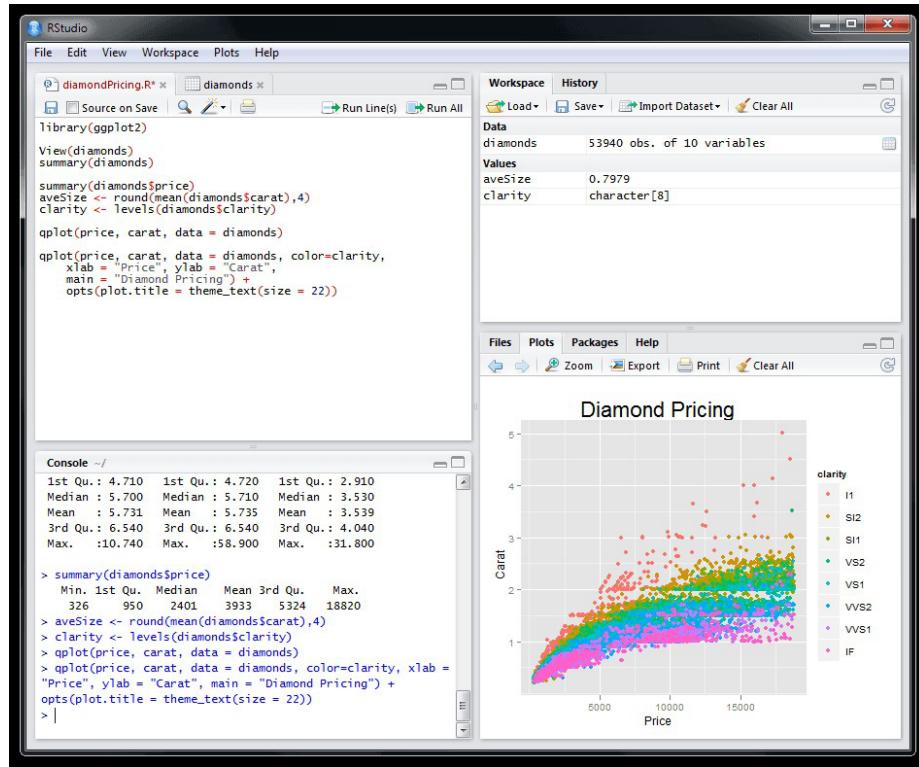
Para auxiliar no desenvolvimento das análises, este livro incentiva o uso do **RStudio**. Trata-se de um software livre de ambiente de desenvolvimento integrado (IDE) para o R¹.

De forma ilustrativa, o R e o RStudio operam como a figura abaixo:



Com o RStudio, você estará diante do seguinte dashboard:

¹IDE, do inglês *Integrated Development Environment*, é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo.



Se está começando a usar o R para análise de dados, recomendo o seguinte material:

1. [R for Data Science](#);
2. [Modern Dive - Statistical Inference via Data Science](#);
3. [Curso R](#);
4. [Usando R: Um Guia para Cientistas Políticos](#);

Em caso de dúvidas, use e abuse de fóruns como o [Stackoverflow](#). Para aprimorar seu código e otimizar o desenvolvimento de suas análises, os guias de estilo do [Google](#) e do [RStudio](#) são ótimas referências.

1.2 O Pacote `txt4cs` e outros

`txt4cs|`

Este livro conta com o pacote `txt4cs`. Ele traz consigo funções específicas e bases de dados utilizadas nos exemplos apresentados. Um dos acervos de exemplo se refere ao conteúdo proferido em 17 de abril de 2016, dia de aprovação do impeachment da então Presidenta Dilma Rousseff na Câmara dos Deputados.



Figure 1.1: Fonte: Empresa Brasil de Comunicação - EBC

Para instalação, use os comandos abaixo:

```
if(require(devtools) == F) install.packages('devtools'); require(devtools);
devtools::install_github("davi-moreira/txt4cs-pkg")
require(txt4cs)
```

Ademais, os seguintes pacotes são essenciais para o desenvolvimento da análise

automatizada de conteúdo com o R. Conforme forem necessários, serão apresentados no livro.

```
install.packages("tidyverse")
install.packages("stringr")
install.packages("quanteda")
install.packages("readtext")
install.packages("stringi")
install.packages("tm")
```

1.3 Material de apoio

Este livro não é feito do zero e resulta de inspiração em diferentes fontes. As principais são:

1.3.1 Referências para processamento de sequências de caracteres com o R

1. [Handling and Processing Strings in R](#) e [Handling Strings with R](#)
2. [R Wikibook: Programming and Text Processing](#)
3. [stringr: modern, consistent string processing](#)

1.3.2 Referências em análise de conteúdo com o R:

1. [Quanteta Tutorials](#)
2. [Text Mining with R](#)

Chapter 2

Text as data: o texto como dado

2.1 Panorama da área

A análise de conteúdo possui grande relevância para as ciências sociais. Contudo, sua abordagem manual sempre limitou o volume de documentos sob análise. São raros os projetos como o *Manifesto Research Group* que, desde os anos 1970, analisa a ênfase temática de manifestos partidários ou o *Comparative Agendas Project*, que coleta e analisa dados sobre agendas de políticas públicas em diferentes países.

O avanço tecnológico e científico permitiu que técnicas automatizadas de análise do conteúdo fossem desenvolvidas e aplicadas de forma simples a grandes acervos. Este avanço não foi realizado sem a contribuição das ciências sociais. Só a *Political Analysis*, principal revista de métodos em ciência política, possui dois *special issues* dedicados ao tema (*Special Issue*, *Virtual Issue*).

2.2 Oportunidades

Com o desenvolvimento de métodos para análise automatizada de conteúdo, hoje o leque de oportunidades as ciências sociais é diverso e promissor. Agora, é possível:

- **Analizar grandes acervos** de forma ágil e barata, otimizando o trabalho do pesquisador.
- **Pesquisar novos acervos** para inferir o conteúdo presente e assim guiar pesquisas através de atalhos informacionais.



Figure 2.1: Biblioteca Florestan Fernandes - FFLCH - USP

Figure 2.2: Acervo da CIA: <<https://www.cia.gov/library/readingroom/advanced-search-view>>

- Analisar processos políticos contemporâneos.



Figure 2.3: Trecho de fala do Deputado Federal Glauber Braga (PSOL-RJ) durante seu voto no processo de impeachment da então Presidenta da República Dilma Rousseff em 2016.

- Redes sociais.
- Fake news!



- **Olhar o passado com as lentes do presente.** Questões que antes não podiam ser enunciadas agora podem ser respondidas! Processos políticos conhecidos podem ganhar novas interpretações através do uso de métodos e técnicas contemporâneas de análise automatizada de conteúdo.
- **Contribuir socialmente:** Retórica Parlamentar - Projeto experimental



Figure 2.4: Foto de Pedro Ladeira, Folha de São Paulo, maio de 2019.



Figure 2.5: Liberdade Guiando o Povo - Eugène Delacroix - 1830

desenvolvido no primeiro Hackathon da Câmara dos Deputados em 2013 por Davi Moreira, Manoel Galdino e Luis Carli. Posteriormente incubado pelo Laboratório Hacker da Câmara dos Deputados.



2.3 Quadro geral de metodologias

Dada a complexidade da linguagem, o processo de geração, produção e seleção de dados que resultam na comunicação humana é ainda um mistério para a ciência (Izumi and Moreira, 2018; Grimmer and Stewart, 2013). Logo, modelos estatísticos desenvolvidos falham na tarefa de prover um relato preciso do processo de geração de dados utilizados na produção de conteúdo e, principalmente, em seu significado.

Os modelos de análise de conteúdo, portanto, não devem ser avaliados pelo quanto explicam do processo de geração dos dados. Transformar palavras em números não substitui a leitura cuidadosa e atenta de documentos. Reconhecendo que “métodos de análise automatizada de conteúdo são modelos incorretos de linguagem” (Grimmer and Stewart, 2013, p. 2), a performance de qualquer método automatizado não é garantida sem a consideração de ao menos quatro princípios:

1. Todos os modelos quantitativos de análise de conteúdo estão errados, mas alguns são úteis;
2. Métodos quantitativos de análise de conteúdo amplificam a capacidade humana, mas não a substitui;

3. Não há um método global para a análise automatizada de conteúdo;
4. Validar, validar, validar.

A escolha do modelo, da família de modelos ou de eventuais combinações a serem utilizadas é resultado dos objetivos almejados. Há uma variedade de modelos disponíveis e nenhum deles se sobrepõe aos demais.

Além de estatísticas e outras informações que podem ser obtidas através da mineração do texto enquanto dados, nesse livro será dado foco aos métodos de escalonamento e classificação de conteúdo. Assim, como indicado pelo quadro de Grimmer e Stewart (2013) métodos de análise supervisionada e não supervisionada serão abordados.

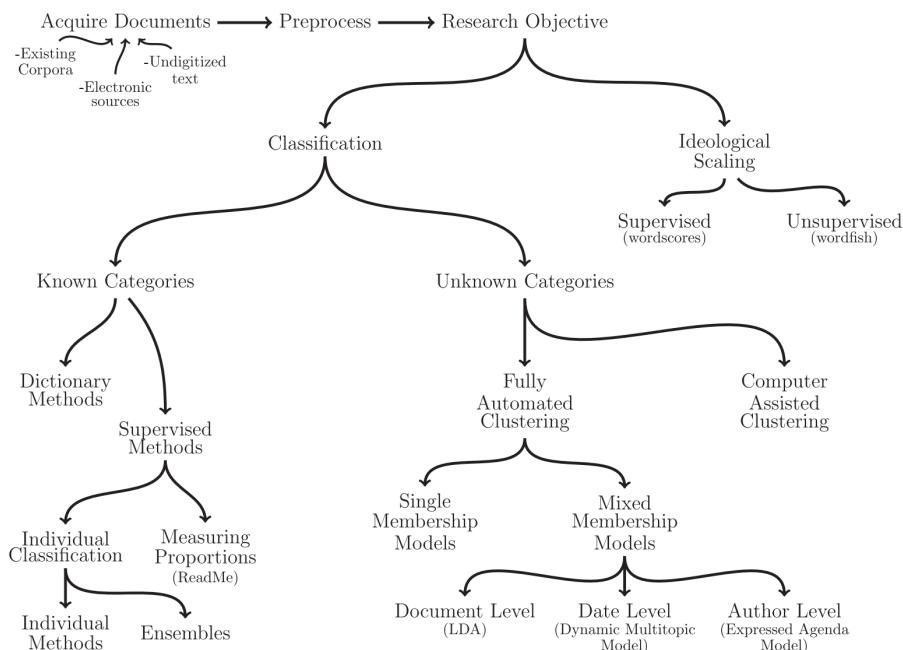


Figure 2.6: Quadro geral de metodologias para análise automatizada de conteúdo (Grimmer e Stewart, 2013)

2.4 O processo de análise do texto como dado

O processo de trabalho para análise quantitativa de texto é muito similar a qualquer tipo de fluxo de trabalho para análise de dados em geral. Como indicado no livro [Text Mining with R: a tidy approach](#) (Silge and Robinson, 2017), o seguinte fluxograma será adotado nesse livro:

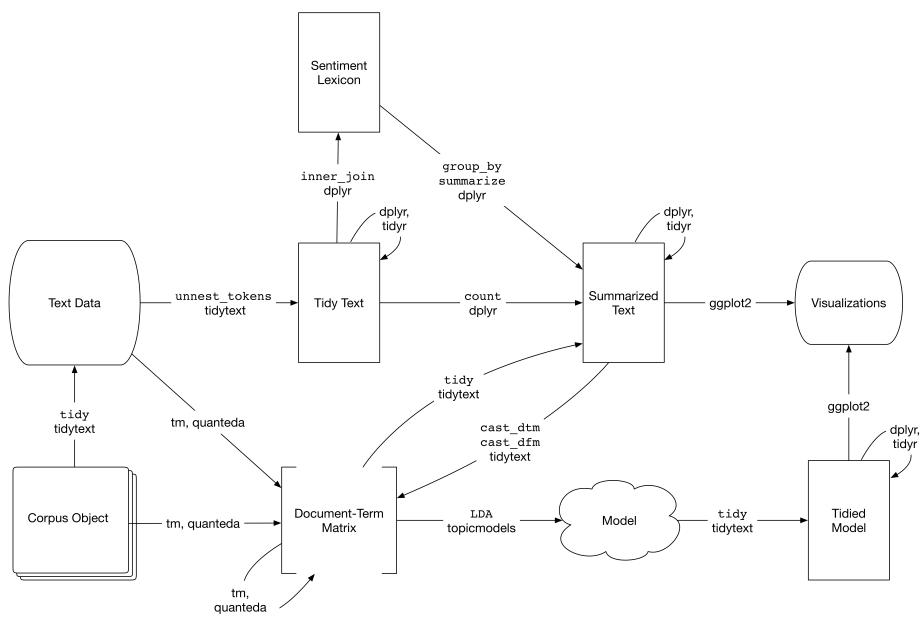


Figure 2.7: Fonte: Text Mining with R

Chapter 3

R e o Processamento de Linguagem Natural

O [processamento de linguagem natural \(NLP\)](#) é um subcampo da ciência da computação relacionado às interações entre computadores e a linguagem humana. O R dispõe de uma [série de pacotes](#) dedicados a essa área e apresenta grande potencial ao conectar o processamento de linguagem natural a todo seu arcabouço de pacotes estatísticos¹.

3.1 Encoding - Codificação de caracteres

Um repertório de caracteres é representado por algum tipo de sistema de codificação ([Wiki](#)). Exemplo comum de sistema de codificação é o código Morse que codifica as letras do alfabeto latino e os numerais como sequências de pulsos elétricos de longa e curta duração. Outro exemplo é o sistema de codificação [UTF-8](#), capaz de codificar todos os 1.112.064 pontos de código válidos em Unicode usando até 8 bits.

O R fornece funções para lidar com diferentes sistemas de codificação. Isso é útil se você lida com arquivos de texto que foram criados com outro sistema operacional e especialmente se o idioma não for o inglês e tiver muitos acentos e caracteres específicos. Por exemplo, o esquema de codificação padrão no Linux é [UTF-8](#), enquanto o esquema de codificação padrão no Windows é [Latin1](#).

A função `Encoding()` retorna a codificação de uma sequência de caracteres. Por sua vez, a função `iconv()` é usado para converter a codificação. Vejamos um exemplo de identificação do encoding de uma sequência de caracteres:

¹Este capítulo tem inspiração nesse [Wikibook](#).

```
chr <- "olê, olê, olê, olá, Lula, Lula"
Encoding(chr) <- "UTF-8"
Encoding(chr)

## [1] "UTF-8"
```

Utilizando o resultado do código do bloco acima, vamos agora converter o sistema de codificação para [Latin1](#):

```
chr <- iconv(chr, from = "UTF-8", to = "latin1")
Encoding(chr)

## [1] "latin1"
```

Para conhecer a lista de sistemas de codificação de seu computador, use a função `iconvlist()`.

3.2 Encoding para remover acentos

Conhecer o sistema de codificação e como utilizá-lo é útil se você lida com arquivos de texto criados com outro sistema operacional e/ou em idiomas que utilizam acentos e caracteres específicos. A depender da análise que deseja fazer, pode ser do seu interesse remover os acentos de uma sequência de caracteres. Nesse caso, vejamos um exemplo com o uso do pacote `stringi`:

```
library(stringi)
chr <- "olê, olê, olê, olá, Lula, Lula"
stri_trans_general(chr, "Latin-ASCII")

## [1] "ole, ole, ole, ola, Lula, Lula"
```

No exemplo acima, removemos os acentos da sequência de caracteres utilizando o American Standard Code for Information Interchange - [ASCII](#).

Se desejar uma solução caseira, o pacote `txt4cs`, que acompanha o livro, possui a função `remove_accent()`. Abaixo a sua aplicação:

```
library(txt4cs)
chr <- "olê, olê, olê, olá, Lula, Lula"
remove_accent(chr)
```

Chapter 4

Strings no R

Na ciência da computação chamamos uma sequência de caracteres de **string**. Para o desenvolvimento de análises automatizadas de conteúdo, é necessário saber como processar esse tipo especial de dado (o texto como dado)¹. Nesse sentido, três coisas são importantes de serem lembradas aqui:

1. Computadores não interpretam letras. No limite, todos os caracteres são transformados em sequências compostas por zeros e uns. Logo, é através de padrões que caracteres são interpretados e os computadores armazenam os dados que retornam a nossos olhos.
2. Programar é escrever! Não é à toa que chamamos as formas de escrita em programação de linguagens de programação. Nesse livro, por exemplo, usamos a linguagem R. Sabendo disso, o desafio de se trabalhar com o texto como dado é o desafio de fazer com que o computador diferencie **código escrito** do "**texto como dado**" que ele precisará processar de acordo com os interesses do analista.
3. Como nós brasileiros lemos, o código e texto é processado pelo computador no seguinte sentido: da esquerda para a direita e de cima para baixo. Logo, ao desenvolver seu **script** é importante ter atenção em relação à ordem de escrita para que o computador possa desempenhar corretamente suas tarefas.

É possível utilizar toda a versatilidade de estruturas de dados no R (vetores, matrizes, listas, data.frame, etc.) para processar sequências de caracteres. Como trabalhar com **strings** no R, portanto?

¹Esse capítulo do material é inspirado no livro [Handling Strings with R](#)

4.1 Strings e vetores

Para declarar uma string, utilizamos aspas simples ' ou aspas dupla """ . Vejamos o caso dos dois vetores abaixo, ambos recebendo a letra "a".

```
# Vetores de caracteres
caracter1 <- "a"
caracter2 <- 'A'

class(caracter1)

## [1] "character"
class(caracter2)

## [1] "character"
```

Ambos são da classe character.

4.1.1 O R é case sensitive

O R diferencia letras maiúsculas de letras minúsculas. Se compararmos os dois objetos criados acima, temos:

```
caracter1 == caracter2

## [1] FALSE
```

4.1.2 Sequências de caracteres

```
# string
txt <- "uma string é uma sequência de caracteres"
txt <- 'também pode ser utilizada com aspas simples'

txt <- "no caso de aspas dupla, usa-se 'aspas simples' na string"
txt <- 'no caso de aspas simples, usa-se "aspas dupla" na string'

txt <- "para usar \"aspas dupla\" na string é necessário usar \\"
cat(txt)

## para usar "aspas dupla" na string é necessário usar \
```

O R armazena a sequência de caracteres conforme ela é apresentada. Porém, é possível fazer uso de caracteres especiais para que o computador interprete e apresente o texto de forma adequada. Como vimos acima, o objeto txt armazena a

string conforme foi redigida, mas com o uso da função `cat()` podemos apresentá-lo de forma adequada. Perceba a diferença entre o resultado e a sequência de caracteres que, de fato, foi armazenada no objeto `txt`.

4.1.3 Operações básicas com vetores de strings

É possível declarar um vetor de caracteres vazio.

```
# vetor de caracteres com 5 strings vazias
palmeiras <- character(5)
palmeiras

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

Vejamos seu tamanho:

```
length(palmeiras) # verificando o tamanho do vetor

## [1] 5
```

Vemos que o objeto `palmeiras` possui 5 elementos, todos sem qualquer conteúdo, mas da classe `character`.

Será que é possível inserir conteúdo em elementos específicos do vetor? Vejamos:

```
# incluindo string no primeiro e terceiro elementos do vetor
palmeiras[1] <- "Quando surge o alviverde imponente"
palmeiras[3] <- "Sabe bem o que vem pela frente"
palmeiras

## [1] "Quando surge o alviverde imponente" ""
## [3] "Sabe bem o que vem pela frente"      ""
## [5] "
```

Ótimo! Significa que podemos ter um vetor no com o [Hino do Palmeiras](#), sendo cada um de seus elementos um verso dessa bela poesia.

E seria possível ter um vetor cujos elementos fossem os hinos (sequências de caracteres/strings) de todos os times do país? Sim!

4.1.3.1 Atenção

Um vetor com uma string vazia é diferente de um vetor sem strings

```
# Atenção:
str_vazia <- "" # string vazia
char_vazio <- character(0) # caracter vazio

length(str_vazia)
```

```
## [1] 1
length(char_vazio)

## [1] 0
```

4.1.4 Caracteres e outros tipos de dados

É importante saber como o R processa o texto como dado (`character`) em conjunto com outros formatos.

```
frase <- "Campeonatos Brasileiros vencidos pelo Palmeiras."
is.numeric(frase)
```

```
## [1] FALSE
is.character(frase)

## [1] TRUE
```

Acima verificamos que a classe do objeto `frase` é de tipo `character`.

```
quantidade <- 5 + 5
quantidade
```

```
## [1] 10
is.numeric(quantidade)

## [1] TRUE
is.character(quantidade)
```

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

Acima verificamos que a classe do objeto `quantidade` é de tipo `numeric`. Seria possível converter de um tipo para outro?

```
# convertendo quantidade
quantidade <- as.character(quantidade)
quantidade
```

```
## [1] "10"
is.character(quantidade)
```

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

Sim! Veja que agora o valor 10 aparece entre aspas, pois o objeto `quantidade` foi convertido para a classe `character`.

E se um vetor possuir números e caracteres em diferentes elementos, como o R interpreta a classe desse vetor?

```
# vetor com números e strings
brasileiros <- c(10, "Campeonatos Brasileiros vencidos pelo Palmeiras.")
brasileiros

## [1] "10"
## [2] "Campeonatos Brasileiros vencidos pelo Palmeiras."
class(brasileiros)

## [1] "character"
```

Perceba que o vetor é declarado com o número 10 no primeiro elemento e uma string no segundo elemento. Contudo, o R adota um critério de coerção de dados para que o vetor seja da classe `character`. Por isso, o número 10 é automaticamente convertido como caracter.

O R segue duas regras básicas de coerção de tipos de dados:

1. Se uma cadeia de caracteres estiver presente em um vetor, todo o resto do vetor será convertido em cadeias de caracteres.
2. Se um vetor tiver apenas elementos lógicos e números, os elementos lógicos serão convertidos em números; Valores TRUE se tornam 1 e os valores FALSE se tornam 0.

4.2 Strings e matrizes

No R matrizes são estruturas de dados que suportam apenas um tipo de classe de dados. Logo, assim como no caso do vetor visto anteriormente, ao constatar a presença de alguma entrada de classe `character` automaticamente todos os elementos da matriz são convertidos.

```
# Matrices ----
m <- rbind(c(1:5), letters[1:5])
m

##      [,1] [,2] [,3] [,4] [,5]
## [1,] "1"  "2"  "3"  "4"  "5"
## [2,] "a"  "b"  "c"  "d"  "e"
class(m)

## [1] "matrix"
```

4.3 Strings e data.frames

`data.frames` são as estruturas de dados mais utilizadas no R. Sua versatilidade permite ter no mesmo objeto dados de classes diferentes num formato de matriz (matriz de dados). Vejamos:

```
# Data Frames ----
df1 <- data.frame(numeros = 1:5, letras = letters[1:5])
str(df1)
```

```
## 'data.frame': 5 obs. of  2 variables:
##   $ numeros: int  1 2 3 4 5
##   $ letras : Factor w/ 5 levels "a","b","c","d",...: 1 2 3 4 5
```

Como padrão da função `data.frame()` strings são transformadas em fatores. Para manter strings como caracteres deve-se usar o argumento: `stringsAsFactors = FALSE`.

```
df1 <- data.frame(numeros = 1:5, letras = letters[1:5], stringsAsFactors = FALSE)
str(df1)
```

```
## 'data.frame': 5 obs. of  2 variables:
##   $ numeros: int  1 2 3 4 5
##   $ letras : chr  "a" "b" "c" "d" ...
```

4.4 Strings e listas

Das estruturas de objetos mais populares no R, listas são as mais complexas. Sua grande vantagem em relação às demais estruturas é permitir uma organização hierárquica dos dados independente de sua classe e tamanho. Vejamos um exemplo:

```
# Listas ----
# listas contemplam qualquer tipo de estrutura de dados
ls <- list(1:10, letters[1:5], rnorm(5), m)
ls
```

```
## [[1]]
## [1] 1 2 3 4 5 6 7 8 9 10
##
## [[2]]
## [1] "a" "b" "c" "d" "e"
##
## [[3]]
## [1] -0.8921771 -1.5965511  0.2751271 -1.5293649 -0.2948023
##
## [[4]]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,] "1"  "2"  "3"  "4"  "5"
## [2,] "a"  "b"  "c"  "d"  "e"
```

No exemplo acima, o objeto `ls` é composto por quatro elementos que contêm, cada um, diferentes tamanhos e diferentes estruturas de dados.

4.5 Processamento básico

4.5.1 Contando caracteres

A função `nchar()` é um forma ágil e fácil de se obter o número de caracteres de uma string ou de strings de um vetor.

```
nchar(c("Quantos", "caracteres?"))
```

```
## [1] 7 11
nchar("Quantos caracteres?")
```

```
## [1] 19
```

No exemplo acima, perceba que a função contabiliza o espaço entre palavras como caracter. Por isso, a soma do total de caracteres do primeiro caso ($7+11 = 18$ caracteres) não é igual ao total de caracteres do segundo (19 caracteres).

4.5.2 `toupper()`, `tolower()`

Sendo o R *case sensitive*, para o processamento do texto como dado, pode ser de interesse do pesquisador harmonizar o conteúdo sob análise com o objetivo de ter todos os caracteres em formato maiúsculo ou minúsculo. As funções `toupper()` e `tolower()` desempenham bem esse papel.

```
tolower(c("TUdo eM MinúsCuLA", "ABCDE"))
```

```
## [1] "tudo em minúscula" "abcde"
```

```
toupper(c("TUdo eM mAiúsCula", "ABCDE"))
```

```
## [1] "TUDO EM MAIÚSCULA" "ABCDE"
```

4.5.2.1 Recortando strings: `substr()`, `substring()`.

Para o processamento do texto como dado, também pode ser de interesse do pesquisador a seleção de trechos de uma sequência de caracteres. Isso pode

ser facilmente feito com as funções `substr()` e `substring()` indicando como parâmetros a posição nas quais a string deve ser recortada.

```
substr("O Palmeiras é o time da virada, o Palmeiras é o time do amor.", 1, 30)

## [1] "O Palmeiras é o time da virada"

substring("O Palmeiras é o time da virada, o Palmeiras é o time do amor.", 33, 60)

## [1] "o Palmeiras é o time do amor"
```

4.5.2.2 União, Intersecção, Diferença, Igualdade

Operações com vetores de forma geral podem ser aplicadas a vetores com strings. Podemos, por exemplo, unir diferentes vetores.

```
# União
vec1 <- c("algumas", "palavras", "aleatória", "aqui")
vec2 <- c("e", "algumas", "palavras", "ali")
union(vec1, vec2)

## [1] "algumas"   "palavras"  "aleatória" "aqui"       "e"          "ali"
```

Verificar a intersecção entre dois vetores.

```
# Intersecção
intersect(vec1, vec2)

## [1] "algumas"  "palavras"
```

Verificar a diferença entre dois vetores.

```
# Diferença
setdiff(vec1, vec2)

## [1] "aleatória" "aqui"
```

E a igualdade de elementos entre dois vetores. No caso, entre o vetor `vec1` e ele mesmo.

```
# Igualdade
identical(vec1, vec1)

## [1] TRUE
```

4.5.2.3 Elemento contido em

Outra operação básica de interesse é a verificação se um elemento (no caso, uma sequência de caracteres) está contido num objeto. Vamos verificar abaixo se a sequência “aqui” está contida no vetor `vec1` através do operador `%in%`.

```
# Elemento contido em ----
elem <- "aqui"
elem %in% vec1

## [1] TRUE
```

É importante destacar que o exemplo acima é uma operação básica de vetores no R e não exclusiva para sequência de caracteres. Nesse sentido, ela apenas checa se no vetor `vec1` há algum elemento idêntico ao padrão “aqui”. Mais adiante verificaremos como identificar a presença de sequências de caracteres no interior de outras strings sem que tenham de ser idênticas.

4.5.2.4 Ordenação

É possível ordenar um vetor de strings em ordem alfabética ou em sentido oposto como no exemplo abaixo. Tal versatilidade pode ser útil para o ordenamento de uma matriz de dados completa com base numa variável de nomes, por exemplo.

```
# Ordenando ----
sort(vec1, decreasing = TRUE)

## [1] "palavras"    "aqui"        "algumas"      "aleatória"
```

4.6 O pacote stringr

O pacote `stringr` integra uma coleção de pacotes projetados para a ciência de dados, o `tidyverse`. Combinado ao pacote `stringi`, você terá acesso a praticamente todas as possíveis funções necessárias para o processamento de strings em mais alto nível.

Existem quatro famílias principais de funções no `stringr`:

1. Manipulação de caracteres: essas funções permitem que você manipule caracteres individuais dentro de sequências de caracteres.
2. Ferramentas de espaço em branco para adicionar, remover e manipular espaços.
3. Operações sensíveis à localização geográfica, cujas operações irão variar de local para local.
4. Funções de correspondência de padrões, sendo o mais comum as expressões regulares.

Além do que veremos neste material, é altamente recomendável a consulta ao capítulo sobre strings do R for Data Science.

```
# carregando pacote ----
library(stringr)
```

4.6.1 Verificando o tamanho de uma string²

```
str_length("O Palmeiras é o time da virada, o Palmeiras é o time do amor.")

## [1] 61
```

4.6.2 Identificando caracter numa posição específica.

```
# vetor de strings
txt <- c("O Palmeiras é o time da virada", "o Palmeiras é o time do amor.")
```

Selecionando o terceiro caracter.

```
# identificando terceira letra
str_sub(txt, 3, 3)
```

```
## [1] "P" "P"
```

Selecionando do segundo caracter de trás pra frente.

```
str_sub(txt, 2, -2)
```

```
## [1] " Palmeiras é o time da virad" " Palmeiras é o time do amor"
```

4.6.3 Incluindo caracter ou string numa posicao específica.

```
str_sub(txt, 3, 11) <- "PALMEIRAS"
txt
```

```
## [1] "O PALMEIRAS é o time da virada" "o PALMEIRAS é o time do amor."
```

Preencher uma string em tamanho fixo.

```
str_pad(txt, 50) # por padrão: left
```

```
## [1] "                                     O PALMEIRAS é o time da virada"
## [2] "                                     o PALMEIRAS é o time do amor."
```

Remove espaço extra.

```
txt <- str_pad(txt, 50) # por padrão: left
str_trim(txt)
```

² Assim como fizemos anteriormente com a função nchar().

```
## [1] "O PALMEIRAS é o time da virada" "o PALMEIRAS é o time do amor."
```

4.6.4 Recortando uma string para obter parte da sequência de caracteres.

```
str_sub(txt, start = 3, end = 11)
```

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

É possível fazer o recorte usando índices de trás pra frente.

```
str_sub(txt, start = -14, end = -1)
```

```
## [1] "time da virada" " time do amor."
```

Extração de palavras.

```
word(txt, 2)
```

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

```
word(txt, -1)
```

```
## [1] "virada" "amor."
```

4.7 Regular Expressions no R

Até aqui você viu funções básicas e intermediárias para o processamento de sequências de caracteres no R. Para avançar, é necessário aprender o uso de expressões regulares (*Regular Expressions*).

Como definido no livro [Handling Strings with R](#), uma expressão regular é um conjunto de símbolos que descreve um padrão de texto. Mais formalmente, uma expressão regular é um padrão que descreve um conjunto de cadeias de caracteres. Como o termo “expressão regular” é bastante longo, a maioria das pessoas usa a palavra **regex** para se referir à área.

Entre outras tarefas, o uso de expressões regulares pode ajudá-lo a ([Wickham and Grolemund, 2017](#)):

- Determinar cadeias de caracteres correspondentes a um padrão.
- Encontrar as posições de padrões correspondentes.
- Extraír o conteúdo de padrões correspondentes.
- Substituir o padrão correspondente por novos valores.
- Dividir uma sequência com base na correspondência de um padrão determinado.

No entanto, é preciso ter atenção, pois o uso de expressões regulares pode se tornar uma tarefa realmente complexa. Veja [esta discussão do StackOverflow](#) a respeito de seu uso para identificação de endereços de e-mail, por exemplo.

Dada a complexidade que a área pode assumir, vamos verificar o uso das *regex* em algumas funções do pacote `stringr` com base [nesse tutorial](#). Junto a ele, é recomendável que a leitura atenta do `?regex` no R.

4.7.1 Identificação e Extração de padrão

```
txt <- c("O Palmeiras é o time da virada", "o Palmeiras é o time do amor.")
str_extract(txt, "amor")

## [1] NA      "amor"
str_detect(txt, "amor")
```

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

Utilizando o operador `|` (“OU”):

```
str_detect(c("presidente", "presidencialismo", "presidencialista", "parlamentarismo"),
          ## [1] TRUE TRUE FALSE TRUE
          str_extract(c("presidente", "presidencialismo", "presidencialista", "parlamentarismo"))

## [1] "ente" "ismo" NA      "ismo"
str_extract(c("presidente", "presidencialismo", "presidencialista", "parlamentarismo"))

## [1] NA           "presidencialismo" NA           "parlamentarismo"
```

Usar o “.” corresponde a qualquer caracter exceto uma nova linha:

```
txt <- c("presidente", "presidencialismo", "presidencialista", "parlamentarismo")
str_extract(txt, "...a.....")
```

```
## [1] NA      "cialismo" "cialista" "rlamenta"
```

Para identificar o “.” de fato, usamos “\.”. Para poder usar a “\”, adicionamos mais uma e temos:

```
txt <- c("O Palmeiras é o time da virada", "o Palmeiras é o time do amor.")
str_detect(txt, "\\.")
```

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

Para identificar a “\” de fato, usamos “\\”:

```
txt <- c("O Palmeiras é o time da virada \\ o Palmeiras é o time do amor.")
writeLines(txt)
```

```
## O Palmeiras é o time da virada \ o Palmeiras é o time do amor.
str_detect(txt, "\\.")
## [1] TRUE
```

4.7.2 Substituição

```
txt <- c("O Palmeiras é o time da virada", "o Palmeiras é o time do amor.")
str_replace(txt, "Palmeiras", "PALMEIRAS")
## [1] "O PALMEIRAS é o time da virada" "o PALMEIRAS é o time do amor."
```

4.7.3 Âncoras

Por padrão, expressões regulares buscam por correspondência em qualquer parte de uma sequência de caracteres. Porém, é extremamente útil poder ancorar a busca pela correspondência no início ou no final de uma string. Podemos usar:

- “^” para coincidir com o início da string.
- “\$” para coincidir com o final da string.

```
txt <- c("O Palmeiras é o time da virada", "o Palmeiras é o time do amor.")
str_detect(txt, "^O")
## [1] TRUE FALSE
str_detect(txt, "\\.$")
## [1] FALSE TRUE
```


Chapter 5

Obtenção de conteúdo

Davi Moreira, Mônica Rocabado

Uma das tarefas mais importantes para a análise de conteúdo consiste na sua própria busca e aquisição. O R nos ajuda nessa tarefa a partir de distintas estratégias. A seguir apresento aquelas nas quais o uso de técnicas computacionais e programação potencializa o alcance e escala de acervos a serem utilizados para pesquisas.

5.1 word, excel ou .pdf

Caso tenha realizado surveys com perguntas abertas ou possua conteúdo de texto organizado em formato de documento ou tabulado é possível utilizar o R para analisá-las. Para arquivos em formato excel pode-se usar o pacote `readxl` com a função `read_excel`, funciona de forma similar ao exemplo em de leitura de arquivos em `.csv`. Para leitura de arquivos `.txt`, `.csv`, entre outros, recomenda-se utilizar o pacote `readr`. Vejamos um exemplo:

5.1.1 .xlsx

A Câmara dos Vereadores de São Paulo publica dados de sua atividade no portal [SPLegis](#). Entre as informações disponíveis, é possível obter [relatórios dos projetos](#) com o conteúdo de todas as ementas através do download de um arquivo em formato `.xlsx`!



Figure 5.1: SPLegis

```
library(readxl)
arquivo_excel <- read_excel("Emendas Apresentadas.xlsx")
```

5.1.2 .pdf e .doc

Antes de achar que os dados de um arquivo .pdf ou .doc são um obstáculo para a abordagem do texto como dado, diferentes estratégias podem ser adotadas para sua obtenção e processamento. Com o uso do pacote `textreadr`, por exemplo, de forma simples pode-se transformar o conteúdo do arquivo .pdf num arquivo .txt. Vejamos o conteúdo do [discurso de posse do ex-presidente Luís Inácio Lula da Silva](#) realizado em 01 de janeiro de 2003 no Congresso Nacional:

```
library(textreadr)
# lendo arquivo .pdf
txt <- read_document("https://raw.githubusercontent.com/davi-moreira/txt4cs/master/data/lula_posse.pdf")
# salvando como .txt
writeLines(txt, "lula-pronunciamento-posse-cd-2003.txt")
```

5.2 Webscraping

O *Webscraping* consiste na possibilidade de uso de programação para raspagem de dados da web, ou seja a obtenção de conteúdo presente na web. Nesse sentido, suponha que ao invés de montar uma equipe que irá acessar páginas na web para coletar seu conteúdo, você desenvolverá um programa específico para realizar essa tarefa com foco sobre os objetivos de sua pesquisa.

5.2.1 Pacotes para raspagem de dados

Há diversos pacotes para raspagem de dados com o R. Abaixo segue um lista com os principais. Para referências sobre seu uso, consulte os links indicados, [este tutorial sobre o rvest](#) e [este capítulo sobre web scraping](#).

- [httr](#)
- [xml2](#)
- [rvest](#)

Como o site [Curso-R](#) destaca, esses pacotes não são suficientes para acessar todo tipo de conteúdo da web. Páginas com conteúdo produzido na linguagem [javascript](#), por exemplo, precisam de outras ferramentas para acesso a seu conteúdo. Nesses casos, é necessário “simular” um navegador que acessa a página web e realiza consultas. Uma das melhores ferramentas para isso é o [selenium](#), abaixo indicado.

- [RSelenium](#)

5.2.2 Etapas para raspagem de dados na web

O processo de raspagem dos dados consiste nas seguintes etapas:

- **Etapa 1:** Conhecer detalhadamente o caminho para acesso aos dados
 - Qual o caminho que um usuário necessita realizar para obter os dados?
 - É necessário preencher um formulário ou assinalar um Recaptcha?
- **Etapa 2:** Armazenar todos os caminhos de acesso aos dados de forma amigável ao programa
 - Caso exista um caminho para obter esse dado, deve ser registrado
 - Não é necessário realizar esse procedimento em todos as páginas que for realizar, mas é desejável.
- **Etapa 3:** Obter os dados: raspagem de fato
- **Etapa 4:** Processar os dados obtidos

5.2.3 Código fonte

Toda página na internet possui um código-fonte - muitas vezes em [html](#) - que indica e cria o conteúdo de forma visual para página. Ao clicar na página desejada com o botão direito do mouse e selecionar “código fonte” ou digitar CRTL + U, pode-se visualizá-lo. Vejamos um exemplo:

Ao acessar o código fonte dessa [página do portal da Câmara dos Deputados](#), você consegue visualizar o código [.html](#) que produz toda visualização, incluindo o



Figure 5.2: Câmara dos Deputados

conteúdo do discurso proferido pelo então deputado Jair Bolsonaro em 17 de Agosto de 2010¹.

5.2.4 Obtenção de Código Fonte - Exemplo:

Se é possível visualizar o conteúdo, é possível obtê-lo de forma automatizada. Vamos, portanto, obter [o conteúdo do discurso proferido pelo então deputado Jair Bolsonaro em 17 de Agosto de 2010²](#).

Vamos utilizar as Etapas anteriormente apresentadas:

De uma só vez, conseguimos cumprir as Etapas 1 e 2 com o código abaixo:

Etapa 1: Conhecer detalhadamente o caminho para acesso aos dados: no nosso exemplo, o código fonte do endereço virtual que apresenta o discurso também nos apresenta o conteúdo publicado.

Etapa 2: Armazenar todos os caminhos de acesso aos dados de forma amigável ao programa: nesse exemplo, trata-se de apenas um endereço que armazenamos no objeto `link`.

```
# carregando pacotes ----
library(tidyverse)
library(rvest)
library(httr)
library(xml2)

# definindo o endereço da web
link <- "https://www.camara.leg.br/internet/SitaqWeb/TextoHTML.asp?etapa=5&nuSessao=17"
```

Etapa 3: Obter os dados: Podemos facilmente obter o código fonte de um endereço na internet com o uso da função `readLines`. Aplicamos, portanto, a função no objeto `link` e atribuímos seu resultado ao objeto `conteudo`.

¹Conteúdo publicado pela Câmara dos Deputados sem revisão do autor.

²Conteúdo publicado pela Câmara dos Deputados sem revisão do autor.

```
# obtém o código fonte
conteúdo <- readLines(link)
```

Veja que o objeto `conteúdo` é um vetor cujos elementos são cada uma das linhas presentes no código fonte da página. Isso significa que não precisamos mais do acesso à internet ou do próprio endereço para processar o conteúdo obtido uma vez que já está retido no seu ambiente de trabalho no R. Nessa etapa, pode ser conveniente salvar o objeto `conteúdo` em seu formato bruto para posterior tratamento.

Etapa 4: Processar os dados obtidos:

Para finalizar nossa tarefa, uma rápida análise do objeto `conteúdo` (código fonte da página que publicou o discurso), nos mostra que o elemento 328 do vetor apresenta o conteúdo de interesse. Veja que, nesse caso, a análise do próprio código fonte da página da Câmara dos Deputados apresenta o conteúdo do discurso na linha 328 do código `html`.

Para processar os dados obtidos, vamos selecionar apenas o elemento 328 e assim concluímos nossa missão.

```
conteúdo <- conteúdo[328]
```

5.2.4.1 Atividade prática

Com base no exemplo acima, obtenha o código fonte da página do [Chico Buarque na Wikipédia](#).

5.3 Web Services

Os [Web services](#) são utilizados para disponibilizar serviços interativos na Web, podendo ser acessados por outras aplicações. O objetivo dos Web Services é a comunicação de aplicações através da Internet. Um dos motivos que tornam os Web Services atrativos para a obtenção de dados e conteúdo é o fato deste serviço ser desenvolvido com base em tecnologias *standards*, em particular XML e HTTP (Hypertext Transfer Protocol).

5.3.1 Obtenção de conteúdo via WS - Exemplo:

A Câmara dos Deputados do Brasil possui um excelente serviço de transparência. Estimulado pela iniciativa do [Laboratório Hacker](#), foi desenvolvido o [Web service da Câmara dos Deputados](#).

Após a realização da Primeira Maratona Hacker da Câmara dos Deputados em 2013, quando fora desenvolvido o [Projeto Retórica Parlamentar](#), o Web Service

da CD passou a disponibilizar os discursos proferidos pelos deputados federais em plenário.



Figure 5.3: September 11th - The Washington Post

Os [ataques terroistas de 11 de setembro de 2001](#) chocaram o mundo e os deputados federais brasileiros não ficaram em silêncio diante de fato tão relevante. Nossa tarefa será a de obter os dados e o conteúdo dos discursos proferidos nesse dia terrível. Em virtude da estrutura de disponibilização dos dados no WebService da Câmara dos Deputados, nossa tarefa será dividida em duas subtarefas:

- obter os meta-dados dos discursos;
- obter o conteúdo dos discursos (inteiros teor).

5.3.2 a) obter os meta-dados dos discursos

Vamos utilizar as Etapas anteriormente apresentadas:

De uma só vez, conseguimos realizar as Etapas 1 e 2:

Etapa 1: Conhecer detalhadamente o caminho para acesso aos dados:

Conhecendo exatamente como deve ser a chamada (o endereço que dá acesso aos dados), podemos simplesmente passar os parâmetros. Os parâmetros podem ser encontrados no guia do [WebService](#). No nosso exemplo, temos como parâmetros a data inicial (`dataInicial`) e a data final (`dataFinal`) de busca pelos discursos. Sendo nosso objetivo obter os discursos proferidos num mesmo dia, usamos como parâmetros “11/09/2001”.

Etapa 2: Armazenar todos os caminhos de acesso aos dados de forma amigável ao programa:

Nesse exemplo, trata-se de apenas um endereço que armazenamos no objeto `link`.

```
# pacotes
library(httr)
library(XML)
library(xml2)
library(RCurl)
library(tidyverse)
library(stringr)

# definindo parametros da chamada
dataInicial <- "11/09/2001"
dataFinal <- "11/09/2001"

# alocando endereço a objeto link
link <- paste("https://www.camara.leg.br/sitcamaraws/SessoesReunioes.asmx>ListarDiscursosPlenarios",
              "dataIni=", dataInicial,
              "&dataFim=", dataFinal,
              "&codigoSessao=&parteNomeParlamentar=&siglaPartido=&siglaUF=",
              sep = "")
```

Etapa 3: Obter os dados:

Para essa etapa, basta fazer uso da função GET. No código abaixo, armazenamos os resultados no objeto `response`.

```
response <- GET(link)
```

Etapa 4: Processar os dados obtidos:

Organizar o resultado da função GET num formato de matriz de dados

(`data.frame`) é simples. Primeiro, transformamos o resultado numa lista, na qual cada elemento será uma das sessões legislativas realizadas naquela data. Em segundo lugar, de forma interativa, usando a função `for`, alocamos os campos desejados num objeto `data.frame`³.

```
# analisa um arquivo XML ou HTML e gera uma estrutura no R.
data <- xmlParse(response, encoding = "UTF-8")

# transforma um XML nó em lista. Importante pois permite você reconhecer o caminho para
ls <- xmlToList(data)

# data frame que receberá dados dos pronunciamentos
bd <- data.frame()

for (i in 1:length(ls)){
  # obtendo quantidade de pronunciamentos de uma respectiva sessão
  quantPronunciamentos <- length(ls[i]$sessao$fasesSessao$faseSessao$discursos)

  sumario <- vector("character")
  numInserção <- vector("character")
  numQuarto <- vector("character")
  indexação <- vector("character")
  hora <- vector("character")
  uf <- vector("character")
  numOrador <- vector("character")
  nomeOrador <- vector("character")
  partido <- vector("character")

  for(j in 1:quantPronunciamentos){
    # obtendo todos os dados do pronunciamento
    # sumário
    sumario[j] <-
      str_trim(ls[i]$sessao$fasesSessao$faseSessao$discursos[[j]]$sumario)
    # inserção
    numInserção[j] <-
      str_trim(ls[i]$sessao$fasesSessao$faseSessao$discursos[[j]]$numeroInserção)
    # quarto
    numQuarto[j] <-
      str_trim(ls[i]$sessao$fasesSessao$faseSessao$discursos[[j]]$numeroQuarto)
    # indexação
    indexação[j] <-
      str_trim(ls[i]$sessao$fasesSessao$faseSessao$discursos[[j]]$txtIndexação)
    # hora
  }
}
```

³Para uma referência de como transformar objetos `xml` em `data.frame` ver: i) (<https://stackoverflow.com/questions/17198658/how-to-parse-xml-to-r-data-frame>); ii) (<https://stackoverflow.com/questions/13579996/how-to-create-an-r-data-frame-from-a-xml-file>)

```

hora[j] <-
  str_trim(ls[i]$sessao$fasesSessao$faseSessao$discursos[[j]]$horaInicioDiscurso)
# uf orador
uf[j] <-
  str_trim(ls[i]$sessao$fasesSessao$faseSessao$discursos[[j]]$orador$uf)
# numero orador
numOrador[j] <-
  str_trim(ls[i]$sessao$fasesSessao$faseSessao$discursos[[j]]$orador$numero)
# nome orador
nomeOrador[j] <-
  str_trim(ls[i]$sessao$fasesSessao$faseSessao$discursos[[j]]$orador$nome)
# partido orador
partido[j] <-
  str_trim(ls[i]$sessao$fasesSessao$faseSessao$discursos[[j]]$orador$partido)
}

# obtendo todos os dados da fase
# codigo
codigoFase <- str_trim(ls[i]$sessao$fasesSessao$faseSessao$codigo)
# descricao
descricaoFase <- str_trim(ls[i]$sessao$fasesSessao$faseSessao$descricao)
# obtendo todos os dados da sessao
# codigo
codigoSessao <- str_trim(ls[i]$sessao$codigo)
# data
dataSessao <- str_trim(ls[i]$sessao$data)
# numero
numSessao <- str_trim(ls[i]$sessao$numero)
# tipo
tipoSessao <- str_trim(ls[i]$sessao$tipo)

bdTemp <- data.frame(codigoSessao = rep(codigoSessao, length(nomeOrador)),
                      dataSessao = rep(dataSessao, length(nomeOrador)),
                      numSessao = rep(numSessao, length(nomeOrador)),
                      tipoSessao = rep(tipoSessao, length(nomeOrador)),
                      codigoFase = rep(codigoFase, length(nomeOrador)),
                      descricaoFase = rep(descricaoFase, length(nomeOrador)),
                      numInsercao = numInsercao,
                      numQuarto = numQuarto,
                      hora = hora,
                      numOrador = numOrador,
                      nomeOrador = nomeOrador,
                      uf = uf,
                      partido = partido,
                      indexacao = indexacao,

```

```

    sumario = sumario)

  bd <- rbind(bd, bdTemp)
}

```

Com o objeto `bd` acima, concluímos a primeira subetapa e podemos seguir para a segunda.

5.3.3 b) obter o conteúdo dos discursos (inteiror teor).**

Importante ressaltar que essa etapa só é possível tendo realizado o método acima de `ListarDiscursosPlenário`, devido exigência do próprio WebService.

```

bdDados <- bd

bd <- data.frame() # data frame que recebera os pronunciamentos

for( i in 1:dim(bdDados)[1]){
  link <- paste("https://www.camara.leg.br/SitCamaraWS/SessoesReunioes.asmx/obterInteiroTeor",
                "codSessao=", bdDados$codigoSessao[i],
                "&numOrador=", bdDados$numOrador[i],
                "&numQuarto=", bdDados$numQuarto[i],
                "&numInsercao=", bdDados$numInsercao[i],
                sep = "")

  print(link)
  response <- GET(link)
  data <- xmlParse(response, encoding = "UTF-8")
  ls <- xmlToList(data)

  bdTemp <- data.frame(nome = ls$nome,
                        partido = ls$partido,
                        uf = ls$uf,
                        horaInicioDiscurso = ls$horaInicioDiscurso,
                        discursoRTFBase64 = ls$discursoRTFBase64)
  bd <- rbind(bd, bdTemp)
  Sys.sleep(.5)
}

```

Obtivemos os discursos em seu inteiro teor! No entanto, os dados estão em formato RTF codificado em Base64. Precisamos transformá-los em formato de texto plano para possibilitar sua leitura:

```

bd$discursoRTFBase64 <- as.character(bd$discursoRTFBase64)

bd$discursoPlainTxt <- vector(mode = "character",
                                length = dim(bd)[1])

```

```
for (i in 1:dim(bd)[1]){
  bd$discursoPlainTxt[i] <- decode_rtf(bd$discursoRTFBase64[i])
  print( bd$discursoPlainTxt)
}
```

5.3.3.1 Atividade prática

Utilizando o código acima, obtenha as falas proferidas do dia da autorização do processo de impeachment da Presidenta Dilma Vana Rousseff na Câmara dos Deputados, ocorrido em 17 de abril de 2016. Salve os dados em formato .rda.

5.4 Download de arquivos da web

Além do conteúdo diretamente publicado numa página web, pode ser de interesse fazer o download de arquivos disponíveis. Em especial, no caso brasileiro, muitos órgãos públicos publicam relatórios em formato .pdf. O obstáculo proporcionado pelo formato do arquivo e o modo como o conteúdo é disponibilizado pode ser superado com o uso da linguagem R.

Como exemplo, vamos conferir o caso do [Tribunal de Contas do Estado de Pernambuco](#), que anualmente disponibiliza relatórios de gestão.



Figure 5.4: TCE-PE

Para atingir nosso objetivo, vamos utilizar as Etapas anteriormente apresentadas:

Etapa 1: Conhecer detalhadamente o caminho para acesso aos dados:

A [página do TCE-PE](#) apresenta os relatórios publicados anualmente.

```
library(rvest)
library(XML)
library(xml2)

link_tce <- "https://www.tce.pe.gov.br/internet/index.php/relatorios-de-gestao-fiscal-"
```

Etapa 2: Armazenar todos os caminhos de acesso aos dados de forma amigável ao programa:

Aqui selecionamos exatamente os endereços de download de cada um dos arquivos publicados. Todos os .pdf são um link dentro do código fonte da página, iremos, portanto, obter os links desses relatórios.

Para tanto, podemos observar que o código fonte possui um padrão que identifica os Relatórios de Desempenho anual: `rdg`. Iremos utilizá-lo para obter somente os documentos que possuam esse padrão.

```
link_relatorios <- link_tce %>%
  read_html %>% # função que irá ler o código fonte escrito em html
  html_nodes("a") %>% # nó presente no código fonte antes do pdf
  html_attr("href") # atributro do nó

# obtenção dos links do relatório através do padrão `rgd`
link_relatorios <- link_relatorios[grep("rdg", link_relatorios)]
```

Etapa 3: Obter os dados:

A obtenção dos dados se refere justamente ao download do material para armazenamento local. Logo, definimos o diretório onde os arquivos serão salvos e fazemos uso da função `download.file`. Como obtivemos mais de um link, para mais de um relatório, no código abaixo apresentamos como obter o primeiro arquivo .pdf presente em `link_relatorios[1]`.

```
download.file(link_relatorios[1], destfile = "seu_diretorio/nome_do_arquivo.pdf", mode
```

Etapa 4: Processar os dados obtidos:

O processo é similar à leitura de conteúdo em arquivo .pdf explicada na seção ??.

```
# pacotes
library(textreadr)
library(here)

## lendo arquivo .pdf
rdg2018 <- read_document("nome_arquivo.pdf")

# salvando como .txt
writeLines(rdg2018, "meu_arquivo.txt")
```

5.5 Twitter

Existem diversos pacotes que possibilitam a captura de informação do Twitter, sendo possível obter tweets ou timelines somente de usuários públicos, ou seja que não possuem um perfil privado. Como exemplo, vamos utilizar o pacote `rtweet` para obter dados dos tweets da timeline da [Deputada Federal Tabata Amaral](#). Vale ressaltar que o Twitter exige que para obtenção dos dados você possua uma conta no Twitter e autorize o app `rstats2twitter` no popup que surgirá no seu browser ao utilizar alguma das funções no console do R, isso ocorrerá somente na primeira vez de uso, criando um token que será salvo para próximas sessões.

```
library(rtweet)

tabata_timeline <- get_timeline( user = "tabataamaralsp", n = 30)
```

Além do uso de obtenção de hashtags é possível buscar termos que estão sendo tweetados ou hashtags através do `search_tweets` que segue lógica similar ao `get_timeline`.

5.6 Imagens

No caso de textos em imagem é possível utilizar o *optical character recognition* (OCR). OCR é o processo de encontrar e reconhecer texto dentro de imagens, por exemplo, de uma captura de tela, texto digitalizado, etc. A imagem abaixo tem um texto de exemplo:

Com o pacote `Tesseract` e o uso da [Interface de Programação de Aplicativos \(API\) do Google](#) é possível capturar seu conteúdo, podendo ser uma imagem presente no seu computador ou da web. Para realizá-lo em português é necessário instalar o acervo de treinamento em português com o seguinte comando `tesseract_download('por')`. No caso, vamos utilizar a imagem do Tweet Fake atribuído ao Presidente Bolsonaro para obter seu conteúdo:

```
library(tesseract)

# obtendo treinamento na língua portuguesa
tesseract_download('por')
por <- tesseract("por") # alocando treinamento na língua portuguesa

# obtendo texto da imagem
text <- tesseract::ocr("https://raw.githubusercontent.com/davi-moreira/txt4cs/master/images/tweet_fakenotreal.jpg")

# resultado
cat(text)
```

This is a lot of 12 point text to test the ocr code and see if it works on all types of file format.

The quick brown dog jumped over the lazy fox. The quick brown dog jumped over the lazy fox. The quick brown dog jumped over the lazy fox. The quick brown dog jumped over the lazy fox.

Figure 5.5: ocr



Jair M. Bolsonaro ✅
@jairbolsonaro

FAKE
NEWS

O que é ciência política?

09:26 · 06/03/2019 · Twitter for iPhone

Figure 5.6: FAKE NEWS!

Segue o resultado:

Jair M. Bolsonaro & 8 a Qjairbolsonaro Vie . S O que é ciéncia política? 09:26 - 06/03/2019 -

5.7 Áudio Transcrição

EM CONSTRUÇÃO.....

Chapter 6

Processamento dos dados

EM CONSTRUÇÃO...

6.1 Tokens

6.2 Corpus

6.3 Tokens e Corpus

6.4 DFM: Matriz de documentos e termos

6.5 Stemming

6.6 FCM: Matriz de co-ocorrência de termos

Chapter 7

Mineração e estatísticas básicas

EM CONSTRUÇÃO...

7.1 Análise de frequência

7.2 Nuvem de palavras

7.3 tf-idf

7.4 Rede de n-grams

7.5 Correlação pareada

7.6 Diversidade lexical

7.7 Similaridade entre documentos/termos

7.8 KEYNESS: Análise de Frequência Relativa

Chapter 8

Escalonamento

EM CONSTRUÇÃO...

8.1 Wordscore

8.2 Wordfish

Chapter 9

Classificação

EM CONSTRUÇÃO...

- 9.1 Método de dicionário: Análise de sentimento**
- 9.2 Naive Bayes**
- 9.3 LDA: Latent Dirichlet Allocation**
- 9.4 STM: Structed Topic Model**

Bibliography

- Grimmer, J. and Stewart, B. M. (2013). Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts. *Political Analysis*, page mps028.
- Izumi, M. Y. and Moreira, D. C. (2018). O texto como dado: desafios e oportunidades para as ciências sociais. *REVISTA BRASILEIRA DE INFORMAÇÃO BIBLIOGRÁFICA EM CIÊNCIAS SOCIAIS - BIB*, 2(86):138–174.
- Silge, J. and Robinson, D. (2017). *Text Mining with R: A Tidy Approach*. O'Reilly Media, Beijing ; Boston, edição: 1 edition.
- Wickham, H. and Grolemund, G. (2017). *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly Media, Sebastopol, CA, 1 edition edition.
- Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.
- Xie, Y. (2019). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.12.