Collecting, Classifying, and Analyzing Textual Data Using R

Alison Blaine and Markus Wust
NCSU Libraries

Welcome!

Schedule

10:00-11:10	class
11:10-11:20	break*
11:20-12:30	
12:30-1:45	lunch
1:45-2:55	
2:55-3:05	
3:05-4:15	

VM installation help

*Please take additional breaks as needed

Restrooms

single-stall: 1st floor, near elevators

multi-stall: this floor, near elevators

lactation room: 1st floor, rm 1202

All public restrooms throughout the building are ADA-accessible.

Food & Coffee

- Cafe (1st floor)
- Dining Hall (On the Oval)
- Food trucks
  - Thursday: Baguetteaboutit
  - Friday: American Meltdown
  - main campus drive

Learning environment

- Questions are welcomed
- Requests to repeat/re-explain are welcomed
- Collaborators in learning - help your neighbor if you can
- We get imposter syndrome too

Agenda

Thursday

Introduction

VM Installation

R Basics

Web scraping

Data parsing

Introduction to APIs

API data harvesting activity

Friday

Intro to Textual Analysis

Textual Analysis Activities

- Sentiment analysis
- Word Frequencies
- TF-IDF
- Bigrams
- Topic Modeling

Don't Panic!

Installation stuff

Goals

Learn how to harvest text data using R from the internet

Do some web scraping

Get some data into R using an Application Programming Interface (API)

Use an R library to harvest data

Learn & implement some R functions for preprocessing text data

Learn and implement some text analysis techniques using R

Know how to seek help & resources in the future

Tip: Getting code working is the first battle, understanding fully what it means comes later

“Good bye”

Etymology: late 16th century: contraction of *God be with you!*

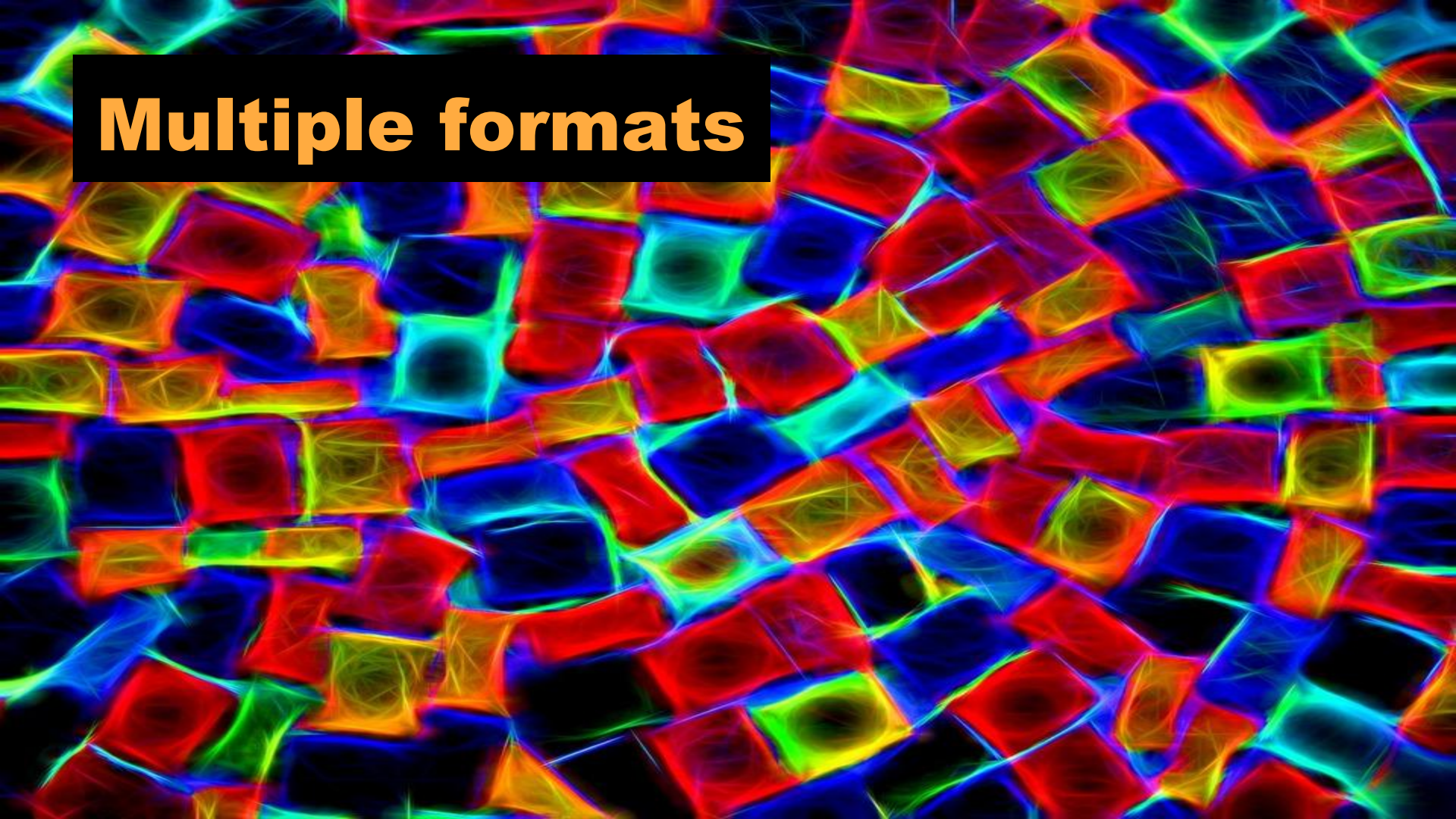
Course Resources

go.ncsu.edu/textr


Collecting Text Data

Data on the Web

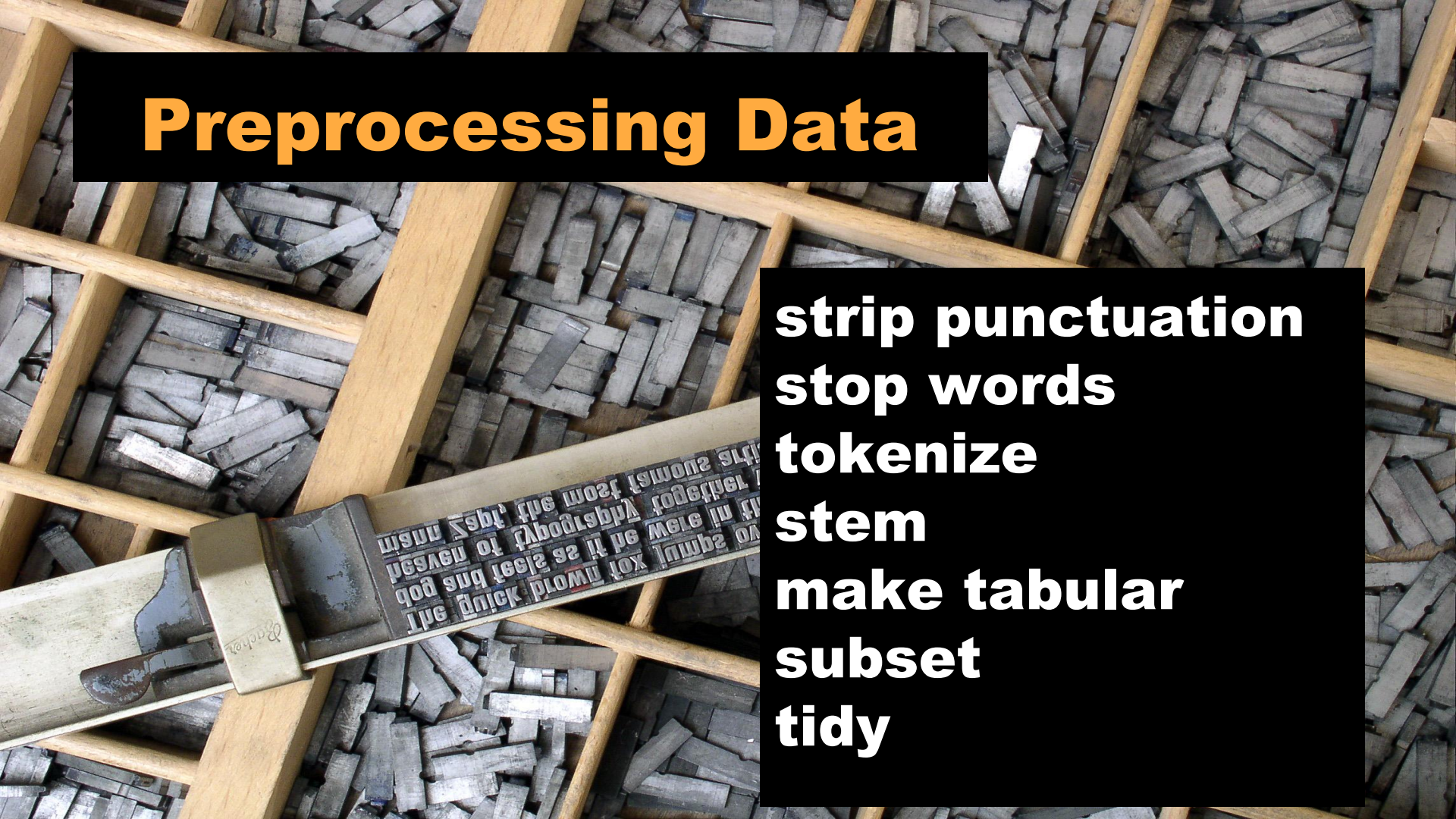

Messy


Multiple formats


Dynamic


Unstructured

Preprocessing Data

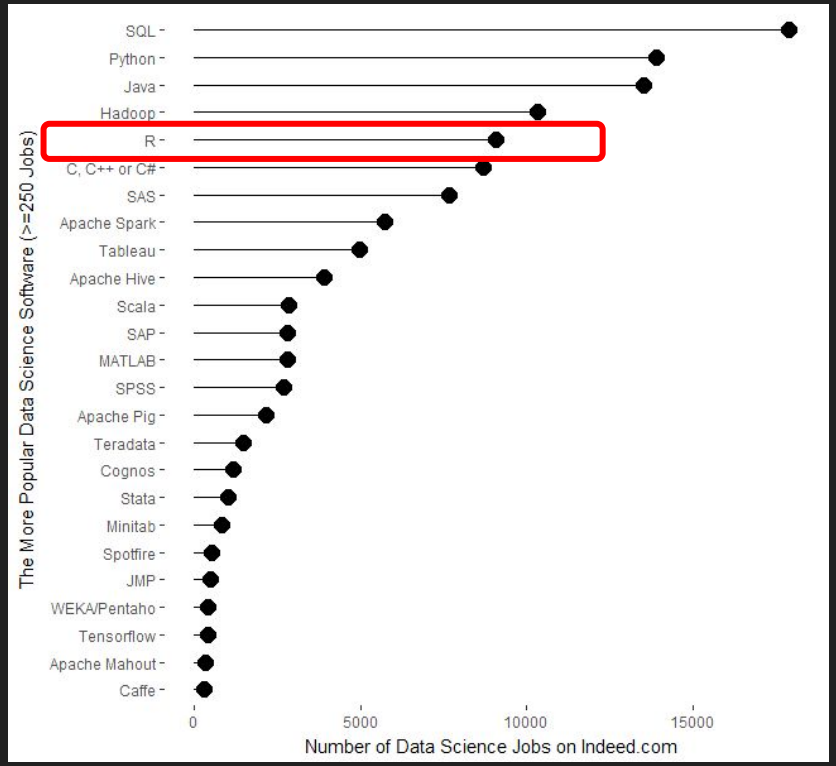


strip punctuation
stop words
tokenize
stem
make tabular
subset
tidy

Analyzing Data

word frequencies
topic modeling
n-grams (bi-, tri-)
part of speech tagging
sentiment analysis

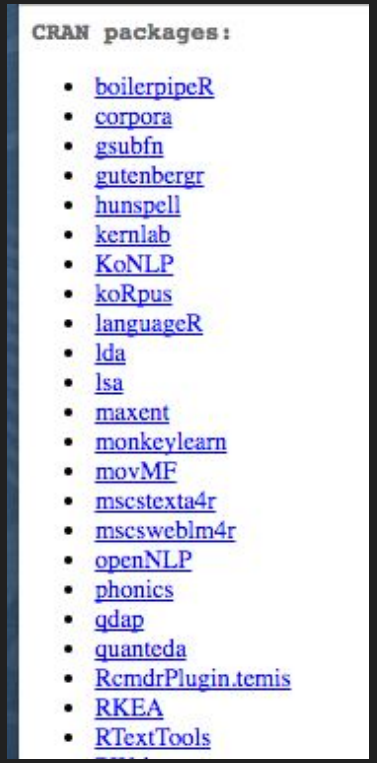
Popularity of R



source

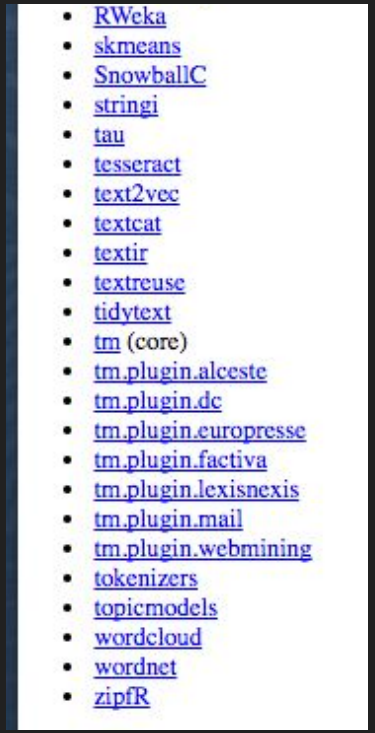
R Packages for Text Analysis

There are more than this, but this is a pretty good list!



CRAN packages:

- boilerpipeR
- corpora
- gsubfn
- gutenbergr
- hunspell
- kernlab
- KoNLP
- koRpus
- languageR
- lda
- lsa
- maxent
- monkeylearn
- movMF
- mscstexta4r
- mscsweblm4r
- openNLP
- phonics
- qdap
- quanteda
- RcmdrPlugin.temis
- RKEA
- RTextTools



- RWeka
- skmeans
- SnowballC
- stringi
- tau
- tesseract
- text2vec
- textcat
- textir
- textreuse
- tidytext
- tm (core)
- tm.plugin.alceste
- tm.plugin.dc
- tm.plugin.europresse
- tm.plugin.factiva
- tm.plugin.lexisnexis
- tm.plugin.mail
- tm.plugin.webmining
- tokenizers
- topicmodels
- wordcloud
- wordnet
- zipfR

R basics

R-objects

Vector

ALL ELEMENTS ARE THE SAME DATA TYPE!

["raspberry", "blackberry", "strawberry", "cherry"]

List

[“raspberry”, “blackberry”, “strawberry”, 1, 2, 3, 5, 6, 7]

Matrix

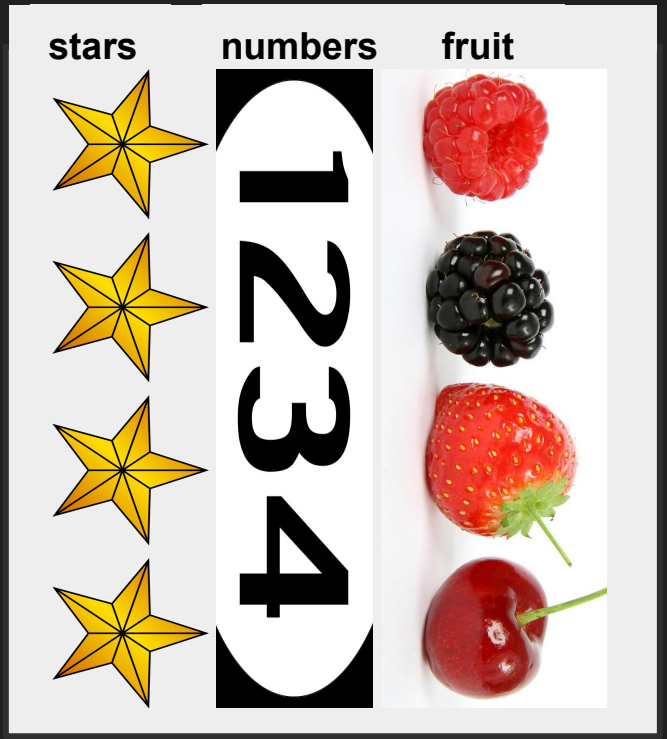
	col1	col2	col3
row1	2	4	3
row2	1	5	7

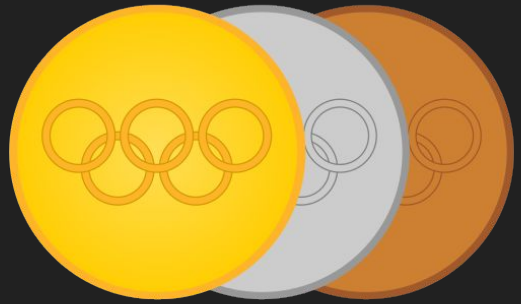
Array

Like a Matrix but can have multiple dimensions! Or not.

Data Frame

Tabular. All items must have the same length! Can be vectors, Arrays or matrices





Factor

```
> x <- c(1, 1, 1, 1, 1, 2, 2, 2, 2, 9, 9, 9, 9)
> factor(x)
 [1] 1 1 1 1 1 2 2 2 2 9 9 9 9
Levels: 1 2 9
```

Ordered categories. Factors make it easier to work with categorical variables in modeling & graphing!

R packages

Installation

Only have to install a package once!

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

Loading

Have to load the Package in each R session!

```
library(ggplot2)
```

Popular packages for working with data

dplyr

dee-PLY-ur

select() - select data

filter() - filter data

arrange() - order data

mutate() - create variables

count() - count

group_by() - group

tidyr

TY-dee-ur

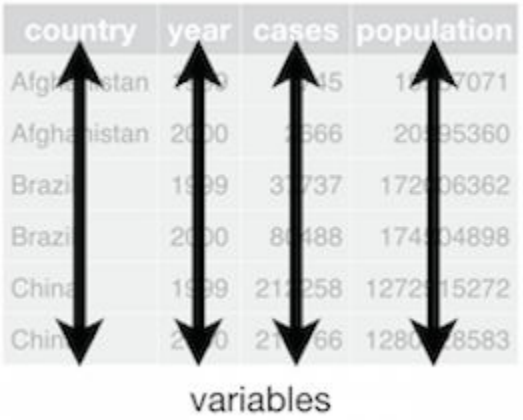
gather() - columns into rows

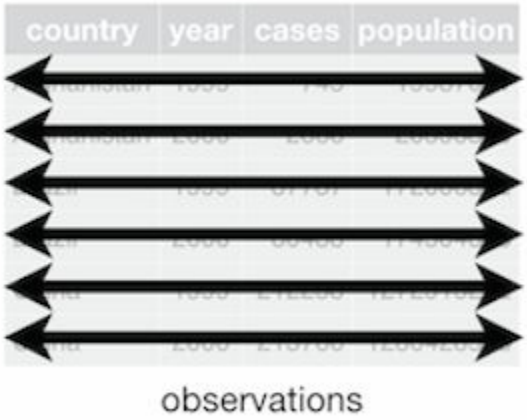
spread() - rows into columns

separate() - one column into several

unite() - several columns into one

Tidy data


country
year
cases
population
variables


country
year
cases
population
observations


country
year
cases
population
values

gather() - columns into rows

tidyr

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

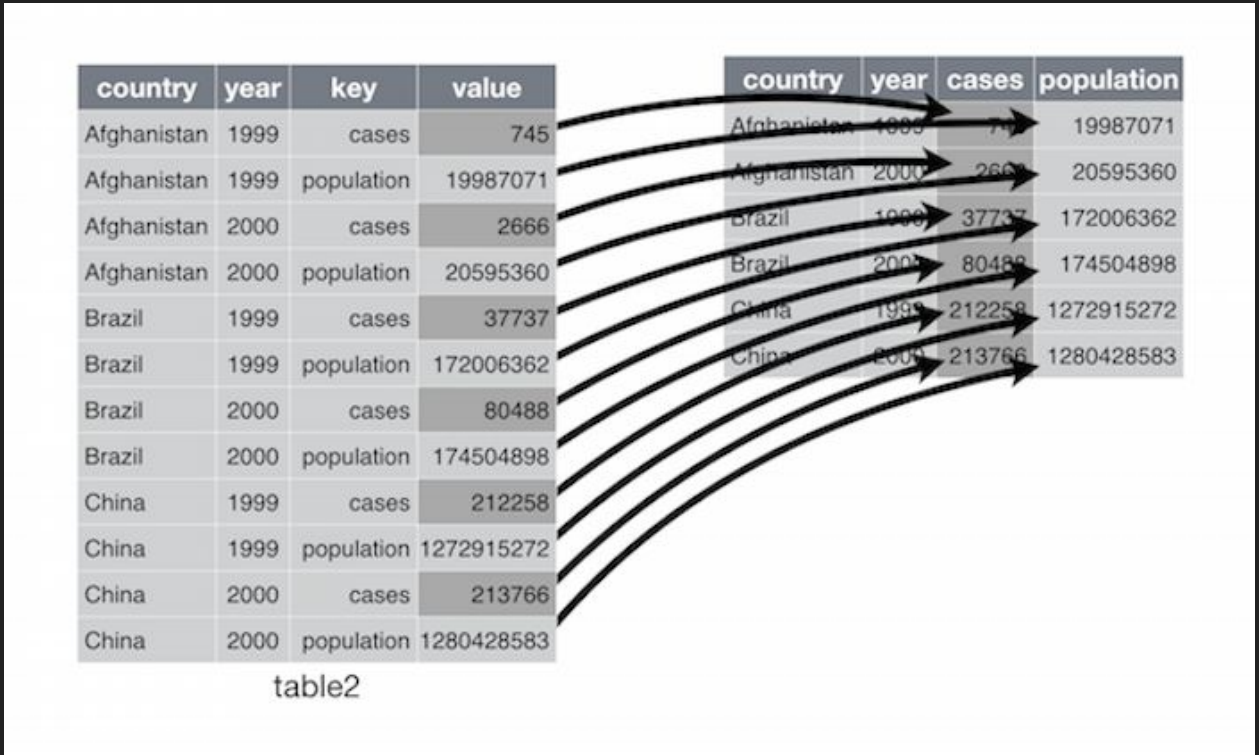
country	1999	2000
[illegible]	[illegible]	2666
Brazil	37737	80488
China	212258	213766

table4

Figure 12.2: Gathering `table4` into a tidy form.

spread() - rows into columns

tidyr



country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

table2

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

magrittr

ma-GRIT-ur

%>%

pipe operator

you can put one variable through multiple functions

Function for washing a car using magrittr

```
dirty_car %>%
soap() %>%
rinse() %>%
dry()
```

Example using dplyr and magrittr

"books" is a data frame with the following information:

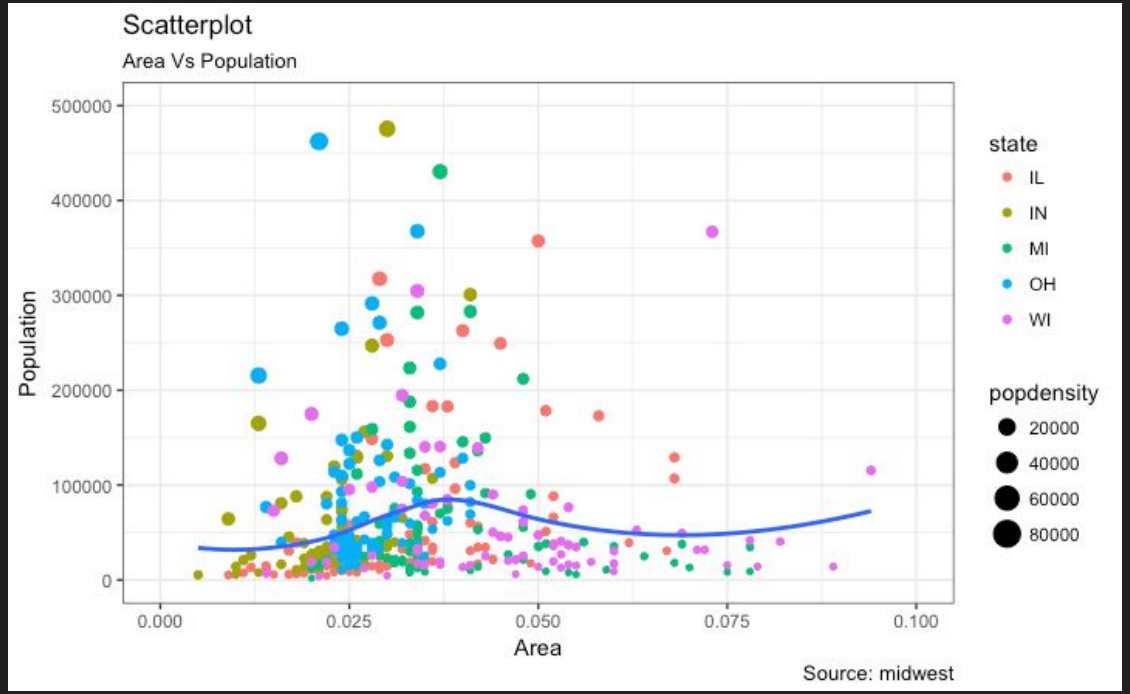
title	author	edition	date_published	type
A Day in the Life of Squid	O. Isles	first	10/4/15	not banned
Recipes for Disaster	Y. Not	second	1/28/17	banned

s

```
banned_books <- books %>%
   filter(type == "banned")
```

What does banned_books contain?

ggplot2


Scatterplot
Area Vs Population
Population
500000
400000
300000
200000
100000
0
0.000
0.025
0.050
0.075
0.100
Area
Source: midwest
state
IL
IN
MI
OH
WI
popdensity
20000
40000
60000
80000

The tidyverse

Components

Cheat sheets

- data wrangling
- ggplot2

R Studio

~/machine-learning-test - RStudio

Go to file/function | Addins | machine-learning-test

Ex_1.R | Ex_2.R

Source on Save | Run | Source

```
1  install.packages("ISLR")
2  install.packages("randomForest")
3  library(ISLR)
4  library("randomForest")
5  library(ggplot2)
6  library(caret)
7
8  # Know more about the Wage dataset
9  data(Wage)
10 summary(Wage)
11 dim(Wage)
12
13
14 # Some feature engineering, we don't need logwage
15 Wage<- subset(Wage, select=- c(logwage))
16
17 # Some Exploratory data analysis a.k.a. visualization
18 qplot(age, wage, data=Wage, colour = race)
19 qplot(age, wage, data=Wage, colour=education)
20
21 Wage_x <- data.frame(as.numeric(Wage$age), as.numeric(Wage$education), as.numeric(Wage$jobclass),as.nu
22 cor(Wage_x, Wage$wage)
23
24
25 inTrain<- createDataPartition(y = Wage$wage, p=0.7, list= FALSE)
26
27 training<- Wage[inTrain,]
```

20:1 (Top Level) | R Script

script

Console ~/machine-learning-test/

```
307460    281.74597 128.97951
156087    134.70538 100.35238
12157      75.35568  82.67135
10832     118.88436 101.01245
160170     27.14058  76.07576
453712    182.02062 147.03898
305387    141.77517 146.90206
380031    135.59806 130.24479
154617     96.37065 128.80519
447647     54.59815  96.90569
451555     96.86670  92.74230
153403     95.23071 109.92425
```

output

Environment | History

Import Dataset | List

Global Environment

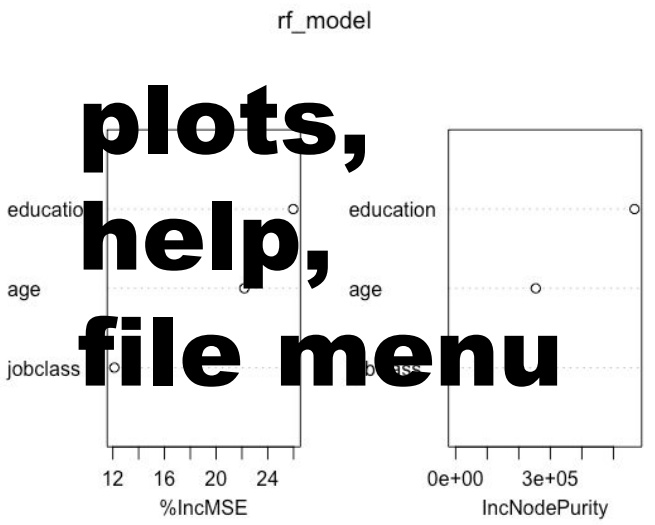
Data

compare_result	898 obs. of 2 variables
emp_data	20 obs. of 2 variables
earnings: num	1 [illegible]
s_rating: num	50 80 80 75 50 75 75 50 50 85 ...
inTrain	int [1:2102, 1] 1 2 5 6 7 10 11 12 14 17 ...
test	7 obs. of 2 variables
testing	[illegible] variables
training	[illegible] variables
Wage	3000 obs. of 11 variables

session info

Files | Plots | Packages | Help | Viewer

Zoom | Export | Publish



plots, help, file menu

Web scraping

Web Scraping

- Accessing one or more webpages and extracting data
- What you do if there is no API (Application Programming Interface) to get data
- For this to work, webpages should use consistent HTML markup

Web Scraping: How Does It Work?

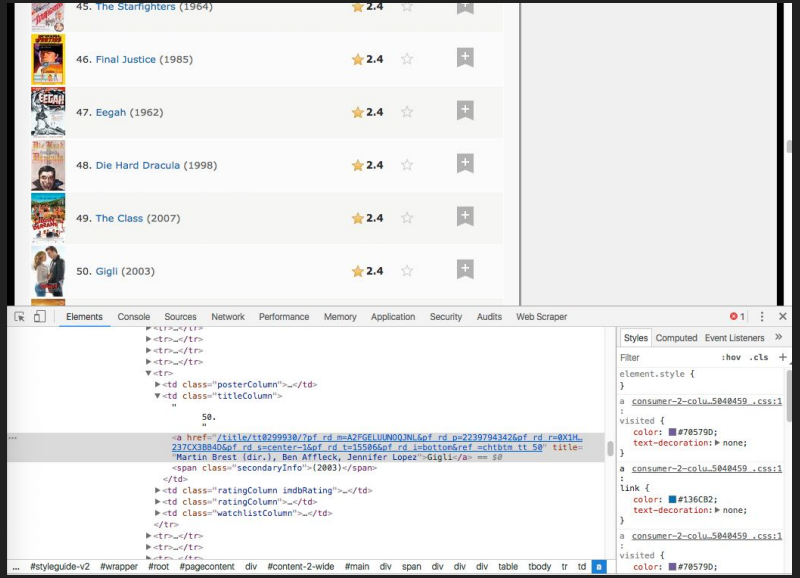
Webpages consist of content (text, images, etc.) and HTML markup:

```
<html>
<body>
<div id="content">
<p id="descripton">Lorem ipsum...</p>
<p id="summary">Lorem ipsum...</p>
</div>
</body>
</div>
```

Web Scraping: How Does It Work?

- You have to identify the HTML elements that contain the data you are interested in
- You can select elements by:
  - The elements' names
  - IDs
  - Classes
  - Their parents (the elements that they are nested in)
  - Their siblings (elements that are at the same level)
  - All of the above!

Web Scraping: How Does It Work?



Note:

- This example uses Google Chrome, but other browsers offer similar functionality out of the box.
- This is different from the browser's "View Page Source" function.

Web Scraping: How Does It Work?

- R has library (rvest) to support web scraping:
  - Loads webpage(s)
  - Selects content based on identifiers you provide
- Sometimes, you cannot clearly identify certain data points based on a webpage's HTML markup
- If you are looking for information that follows a particular format, such as telephone numbers, you can instead use **Regular Expressions**

Regular Expressions

Regular Expression:

"... a sequence of characters that define a search pattern" (Wikipedia)

- They help you find text that follows a specific pattern:
  - Phone numbers / Social Security Numbers
  - URLs
  - Addresses
- You can extract those data or modify them
- Note: There are different flavors of regular expressions

Regular Expressions

- All modern programming languages support regular expressions
- Regular expressions can be used in other environments:
  - Google Sheets
  - Many text editors that you would use for programming
- Since we could do a workshop just on regular expressions:
  - Excellent free tutorial: http://www.regular-expressions.info/tutorial.html
  - Free sandbox for testing: https://regex101.com

Regular Expressions: Examples

- \w <- All word characters. That includes all letters, all digits, and the underscore (_) character.
- \d <- All digits.
- \s <- Any whitespace character, i.e., a space, a tab, a line break, or a form feed.
- \W <- Anything but a word character.
- \D <- Anything but a digit.
- \S <- Anything but a whitespace character.
- \d{1,3} <- Between 1 and 3 digits.
- \d* <- 0 or more digits.

Regular Expressions

What does this regular expression find:

(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])

Regular Expressions

Answer: Email addresses!

```
(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*|
"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\
x0b\x0c\x0e-\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-
9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\
.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:(?:[\x0
1-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0
e-\x7f])+)\])
```

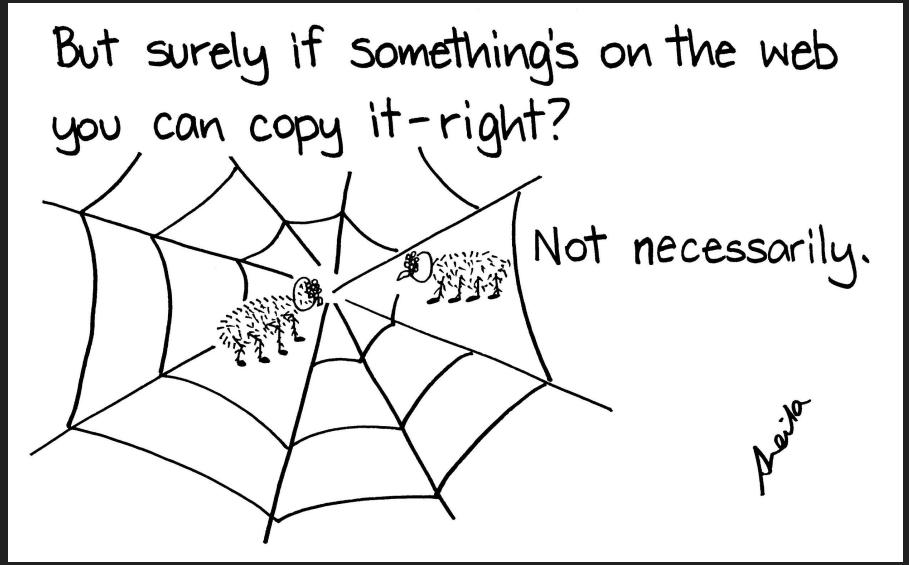
Source: http://emailregex.com/

Web Scraping: Issues

- Although data display is created with HTML markup, data is not really structured
- You can't be 100% certain that elements will always contain the same data
- Scraping becomes more difficult if a page does not use unique IDs for HTML elements

Web Scraping: Issues

- Web scraping is a quick way to get data
- Problem:
  - Just because it is made public does not mean that you can just take and use it for your own work

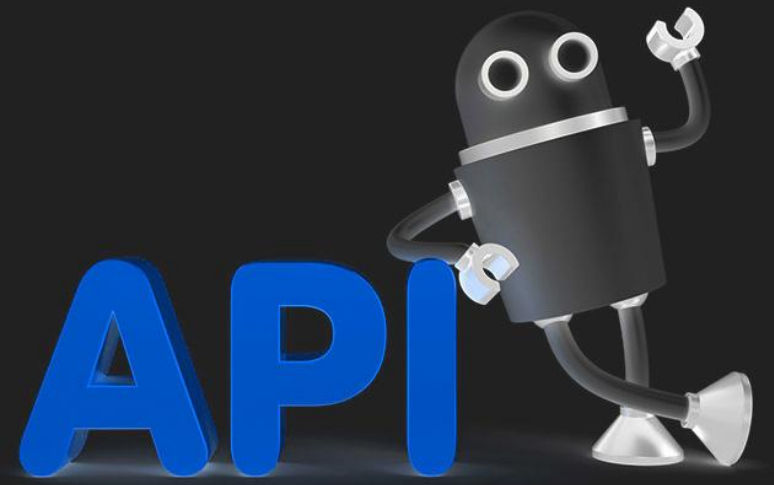


Web Scraping: Issues

- You should ask for permission if there is no explicit permission for data reuse
- That way, you can also ask whether the data are available in an alternative format, e.g., as a spreadsheet; this makes them easier to work with

API: Better Than Web Scraping

- A better way to get data is to use an **API** if one is available and accessible:
  - Legal situation is usually clear
  - Reliable data structure
  - Faster
  - Easier to use

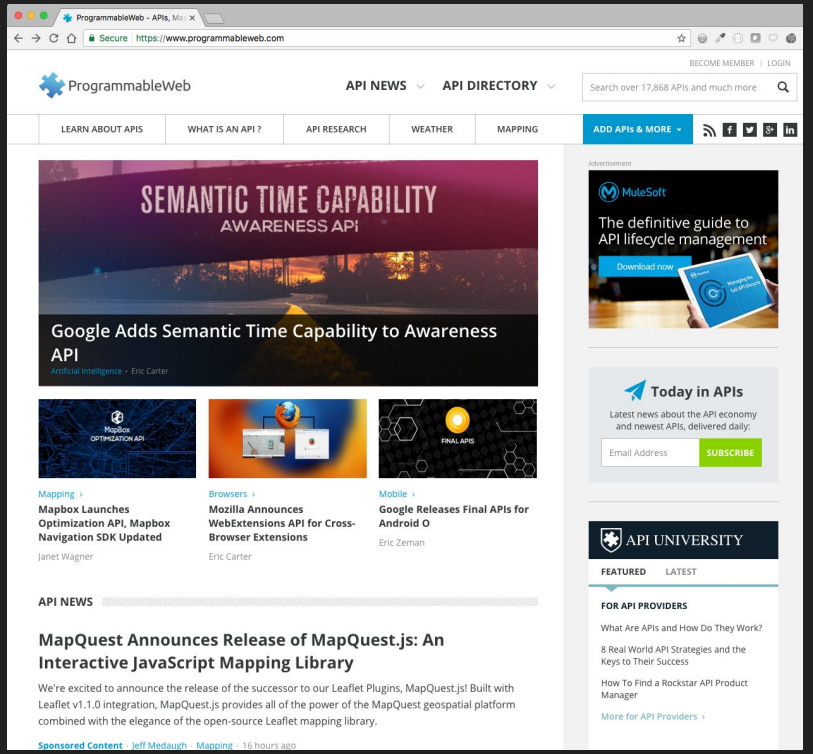


API

API <- Application Programming Interface

Web APIs:

- Allow you to send data requests to many web-based services
- Let you request individual data points or whole data sets
- Return data in a structured format that can easily be processed



API: Openness

APIs differ in their degree of openness:

- Subscription only
- Open but require API key
- Open without API key but rate-limited
  - How many requests can you send per hour?
- Open without limits

API: How To Get Data

- To get data through an API you first have to send a request
- Different technical approaches
- Some require you to first authenticate
- Some ask you to turn your request into a small data package and send that to the service
- Easiest way:
  - Construct a request URL that includes all the required information
  - Have your script send a GET request to that URL
  - API sends back data

API: How To Get Data

Example: arxiv.org (preprint archive [Physics, Math, Comp Sci, Quantitative Biology/Finance, Statistics)

http://export.arxiv.org/api/query?search_query=all:electron&start=0&max_results=20

Example: Wikipedia

https://en.wikipedia.org/w/api.php?action=opensearch&search=electron&namespace=0&format=json

To see what the data returned by these services looks like open the URLs in your browser!

API: Data Formats

- APIs also differ in the data formats they provide
- Some services only offer one format, others let you choose between formats
- The most common formats are:
  - XML <- eXtensible Markup Language
  - JSON <- JavaScript Object Notation

API: Data Formats

XML

```
<?xml version="1.0"?>
<book>
    <id>12345</id>
    <title>Beginning R</title>
    <author>Larry Pace</author>
    <published>
        <publisher>Apress</publisher>
        <year>2012</year>
    </published>
</book>
```

JSON

```
{
    "book": {
        "id": "12345",
        "title": "Beginning R",
        "author": "Larry Pace",
        "published": {
            "publisher": "Apress",
            "year": "2012"
        }
    }
}
```

API

Once the API response is received, you can then parse it and directly address the data elements you are looking for:

temp <- data["properties"]["periods"]["temperature"]

How to find correct elements:

- Look at data returned by API
- Read API documentation

API: Tools

Browser Extension:

- JSON Formatter (Chrome): Make JSON more human-readable

R Libraries

- httr: Handling API requests
- jsonlite: Parsing JSON data, creating R objects

Text Analysis

Train computers to find similarities between texts, extract themes, and analyze expressed sentiments

Text Analysis Methods

Word frequency

The mouse ate the cheese.

The 2

mouse 1

ate 1

cheese 1

Collocation

“love”

true love

my love

with love

Concordance - key word in context

‘Will you be my own true love?’ the musician sang…”
“took the dusty books with love and dusted them…”

N-grams (bi, tri-, etc)

Example: “The merry month of May”

Where n = 2:

The merry

merry month

month of

of May

These are bigrams!

Natural Language Processing

- Part of speech tagging
- Topic modeling
- Sentiment analysis
- etc

Text Analysis: Glossary

stopwords - [the, a, an, and, not, for, but, so...]

tokenize - the/cat/sat/on/the/mat/

stem - playing, played, player → play*

lemmatize -- good, better, best → good

trying, tried, tries → try

Text Analysis: Glossary

tf-idf - how important a term is in distinguishing a text from others in a collection

document-term matrix - rows: documents, columns: terms

bigram - pairs of contiguous words in a document

Workshop activities

sentiment analysis

tokenization

word frequency

document-term matrix

tf-idf

bigrams

visualization

part of speech tagging

topic modeling

Get the Data

Download “Day 2 files” folder

Unzip/extract folder

go.ncsu.edu/textr

Resources

Julia Silge and David Robinson (2017), Text Mining with R: A Tidy Approach

Matthew Jockers (2014), Text Analysis with R for Students of Literature

stackoverflow.com