Computational Political Science

Session 1

David Broska
Zeppelin University
February 2, 2021

Outline for today

1. **Intro**
   - What is this course?
   - What am I going to learn in this course?
2. **Quantitative Text Analysis principles**
   - What is QTA?
   - Motivation and assumptions
   - Example analyses
3. **Review of the R language**
   - Anatomy of code
   - Data types and structures
   - Operators
   - Functions and libraries

Intro

Text as data

AGAPETI PAPÆ I
EPISTOLÆ.

EPISTOLA JUSTINIANI
AD AGAPETUM.

More majorum suorum apud pontificem Romanum recens electum fidei suæ professionem edit, eandem quam supra ad Joannem papam II miserat.

In nomine Domini nostri Jesu Christi Dei imperator Cæsar Flavius Justinianus, Alemanicus, Gothicus, Francicus, Germanicus, Anticus, Alanicus, Vandalicus, Africanus, Pius, Felix, Inclytus, Victor, ac Triumphator semper Augustus, Agapeto sanctissimo archiepiscopo almæ urbis Romæ et patriarchæ.

Ante tempus in hac regia urbe nostra quorumdam de causa fidei exstitit morbosa contentio; quam nos congrue respuentes interposito edicto repressimus. Et quia studii nostri est emergentes hujus-

Reddentes honorem apostolicæ sedi et vestræ sanctitati, quod semper nobis in voto fuit, et est, ut decet patrem, honorantes vestram beatitudinem, omnia, quæ ad Ecclesiarum statum pertinent, festinamus ad notitiam deferre vestræ sanctitatis: quoniam semper magnum nobis fuit studium unitatem vestræ apostolicæ sedis, et statum sanctarum Dei Ecclesiarum custodire, quæ hactenus obtinet, et incommote permanet, nulla intercedente contrarietate. Petimus ergo vestrum paternum affectum, ut vestris ad nos destinatis litteris, et ad sanctissimum episcopum hujus almæ urbis et patriarcham vestrum fratrem, quoniam et ipse per eosdem scripsit ad vestram sanctitatem, festinans in omnibus consequi sedem apostolicam beatitudinis vestræ, manifestum nobis faciatis, quod omnes qui prædictam fidem recte

Course schedule

Session	Date	Topic	Assignment	Due date
1	Feb 02	Overview and key concepts	-	-
2	Feb 09	Preprocessing and descriptive statistics	Formative	Feb 22 23:59:59
3	Feb 16	Dictionary methods	-	-
4	Feb 23	Machine learning for texts: Classification I	Summative 1	Mar 08 23:59:59
5	Mar 02	Machine learning for texts: Classification II	-	-
6	Mar 09	Supervised and unsupervised scaling	Summative 2	Mar 15 23:59:59
7	Mar 16	Similarity and clustering	-	-
8	Mar 23	Topic models	Summative 3	Apr 12 23:59:59
-	-	*Break*	-	-
9	Apr 13	Retrieving data from the web	-	-
10	Apr 20	Published applications	-	-
11	Apr 27	Project Presentations	-	-

Course objectives

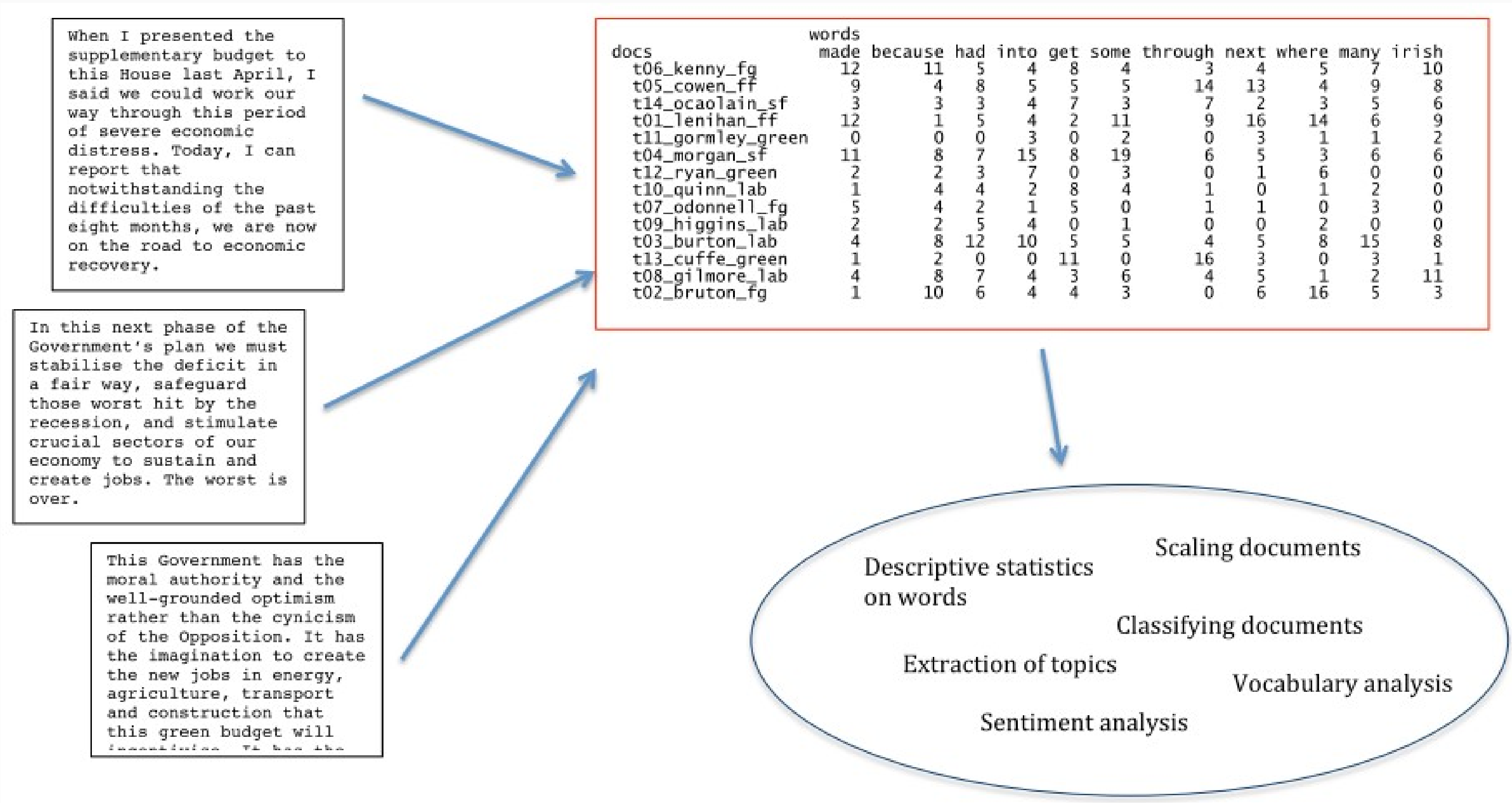
- learning the fundamentals of computational methods, particularly for text analysis
- ability to apply statistical and machine learning methods for text in R
- evaluating the strengths and weaknesses of these techniques for answering political science questions
- enhanced understanding of published applications
- conduct independent empirical research using quantitative text analysis

Tell us about yourself

- What do you study?
- What do you expect from this course?
- What is your experience with R and/or quantitative text analysis?
- Do you have a topic in mind that you want to explore?

Fundamentals of quantitative text analysis

Text to document-feature matrix



docs	words made	because	had	into	get	some	through	next	where	many	irish
t06_kenny_fg	12	11	5	4	8	4	3	4	5	7	10
t05_cowen_ff	9	4	8	5	5	5	14	13	4	9	8
t14_ocaolain_sf	3	3	3	4	7	3	7	2	3	5	6
t01_lenihan_ff	12	1	5	4	2	11	9	16	14	6	9
t11_gormley_green	0	0	0	3	0	2	0	3	1	1	2
t04_morgan_sf	11	8	7	15	8	19	6	5	3	6	6
t12_ryan_green	2	2	3	7	0	3	0	1	6	0	0
t10_quinn_lab	1	4	4	2	8	4	1	0	1	2	0
t07_odonnell_fg	5	4	2	1	5	0	1	1	0	3	0
t09_higgins_lab	2	2	5	4	0	1	0	0	2	0	0
t03_burton_lab	4	8	12	10	5	5	4	5	8	15	8
t13_cuffe_green	1	2	0	0	11	0	16	3	0	3	1
t08_gilmore_lab	4	8	7	4	3	6	4	5	1	2	11
t02_bruton_fg	1	10	6	4	4	3	0	6	16	5	3

Roadmap for QTA projects

1. Selecting texts: Defining the **corpus**
2. **Conversion** of texts into a common electronic format
3. **Defining documents**: deciding what will be the documentary unit of analysis
4. **Defining features**. These can take a variety of forms, including tokens, equivalence classes of tokens (dictionaries), selected phrases, human-coded segments (of possibly variable length), linguistic features, and more.
5. Conversion of textual features into a **quantitative matrix**
6. A **quantitative or statistical procedure** to extract information from the quantitative matrix
7. **Summary** and interpretation of the quantitative results

Why quantitative text analysis?

Justin Grimmer's haystack metaphor

Analyzing a straw of hay: understanding the meaning of a sentence

→ Humans are great! But computer struggle.

Organizing the haystack: *describing, classifying, scaling texts*

→ Humans struggle. But computers are great! (What this course is about)

Principles of quantitative text analysis (Grimmer and Stewart 2013)

1. All quantitative models are wrong - but some are useful
2. Quantitative methods for text *amplify* resources and *augment* humans
3. There is no globally best method for automated text analysis
4. Validate, validate, validate

Quantitative text analysis requires

1. Texts represent an observable implication of some **underlying characteristic** of interest
    - An attribute of the author
    - A sentiment or emotion
    - Salience of a political issue
2. Texts can be represented through **extracting their features**
    - most common is the bag of words assumption
    - many other possible definitions of “features” (e.g. word embeddings)
3. A document-feature matrix can be analyzed using quantitative methods to produce **meaningful and valid estimates** of the underlying characteristic of interest

Overview of text as data methods

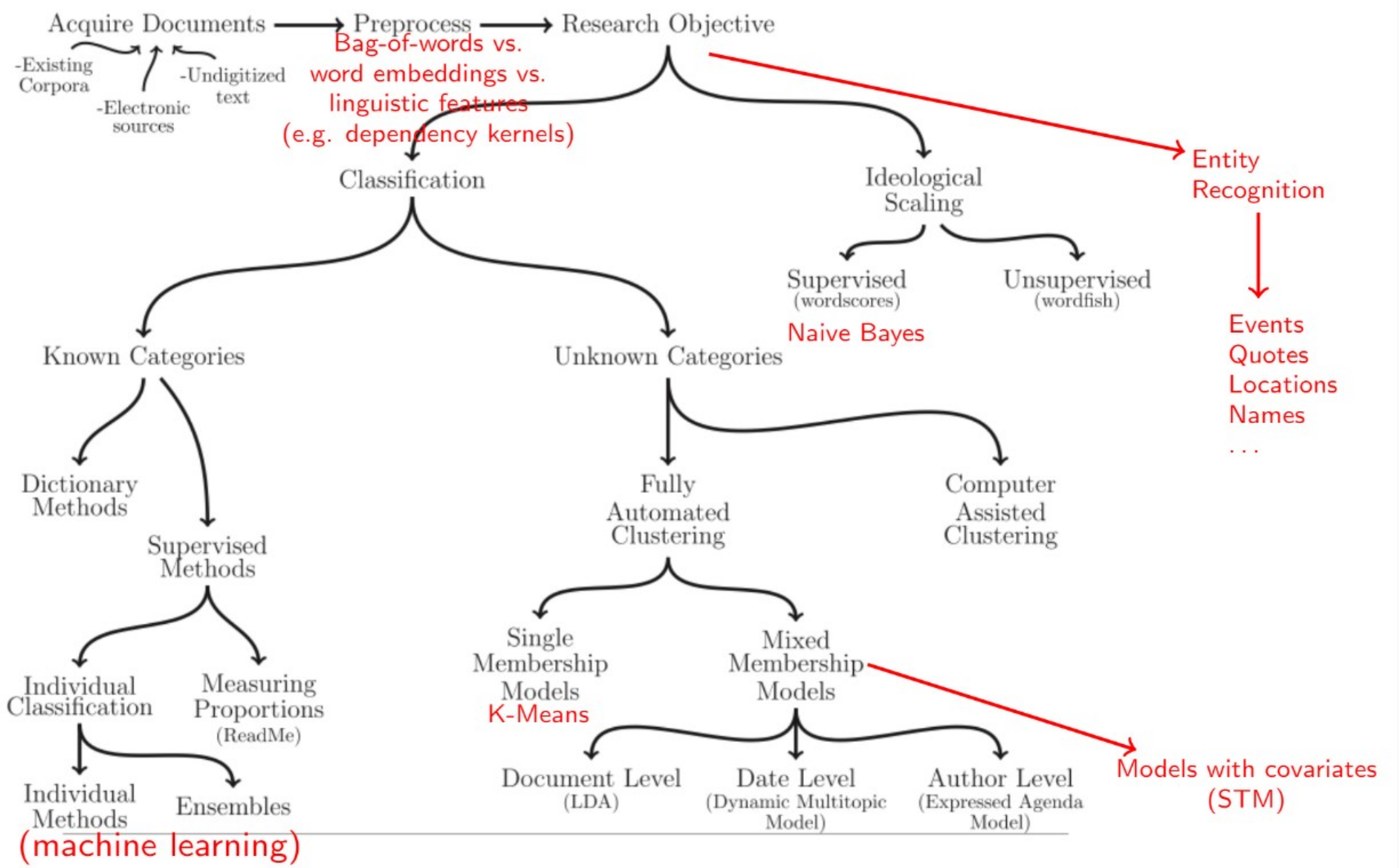
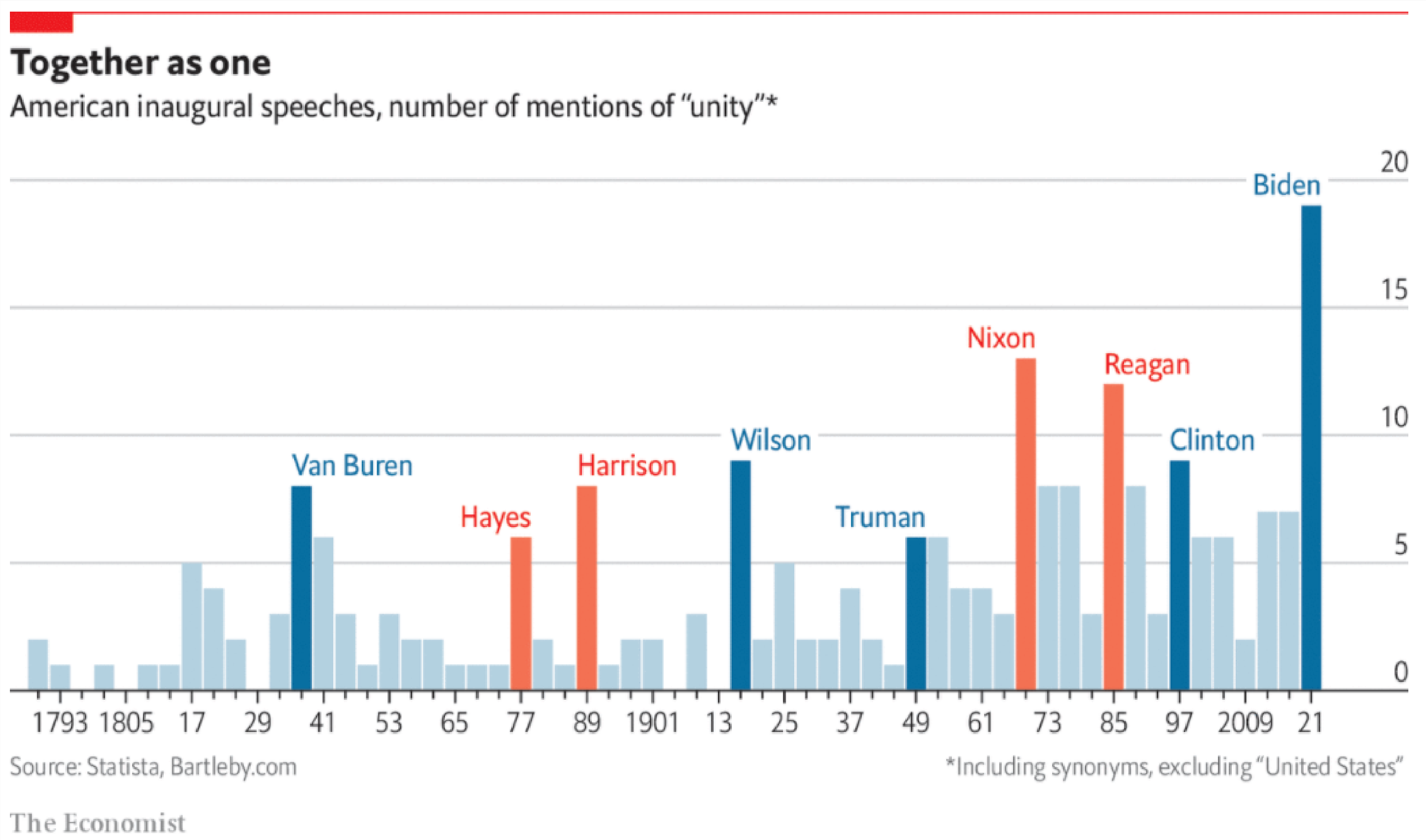


Fig. 1 in Grimmer and Stuart (2013)

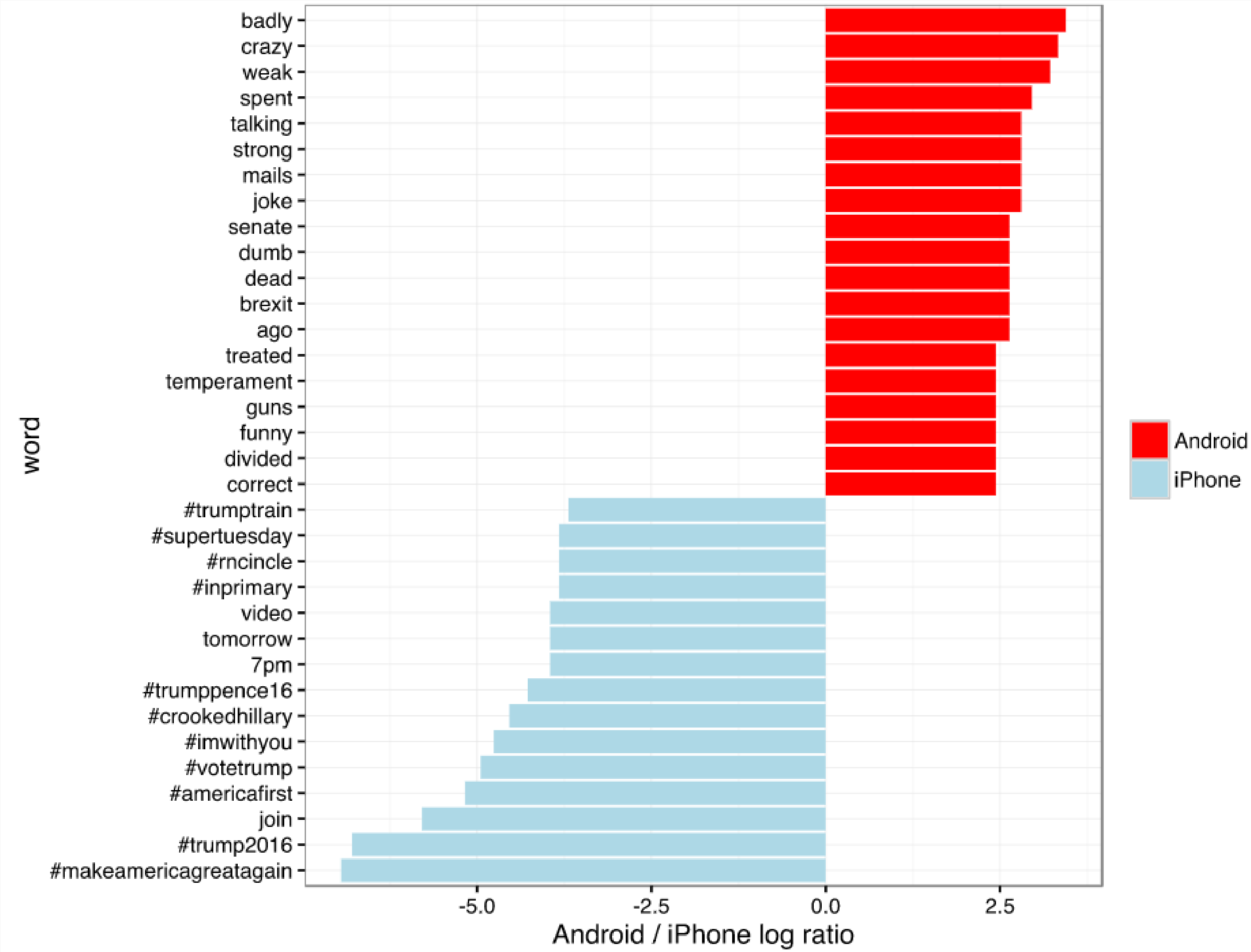
Examples

Compare documents



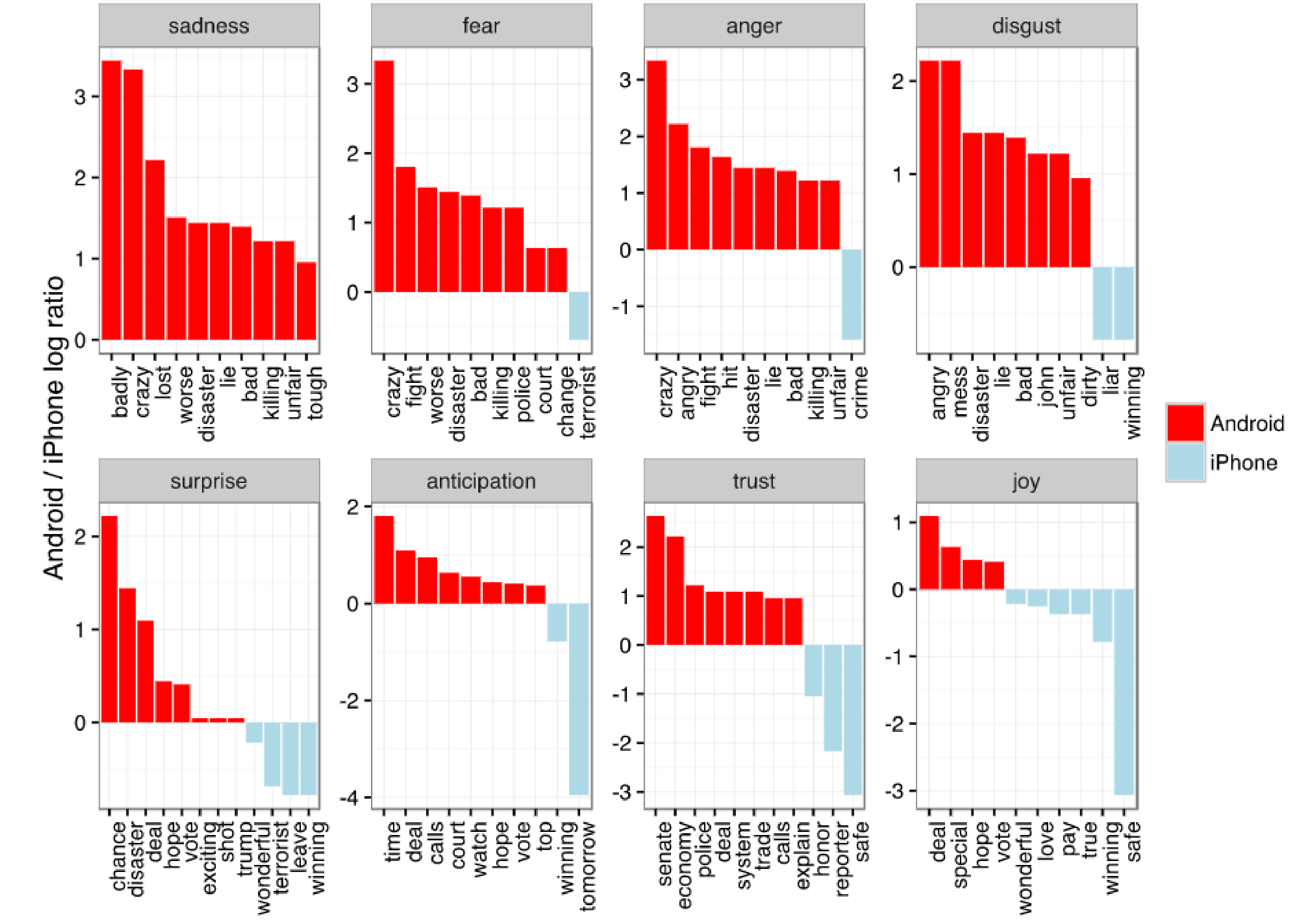
The Economist (2021)

Authorship identification



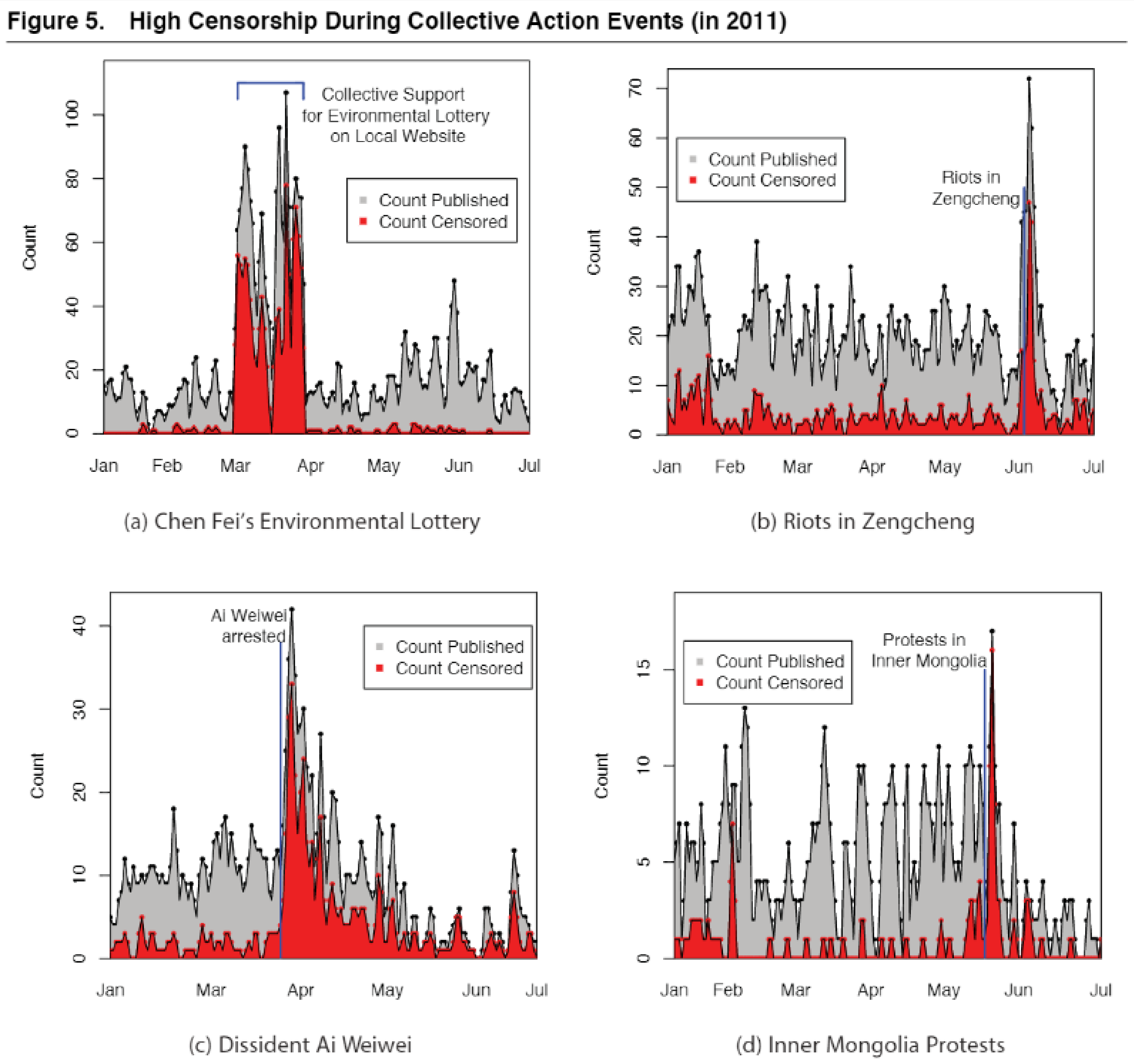
Robinson (2016)

Sentiment analysis



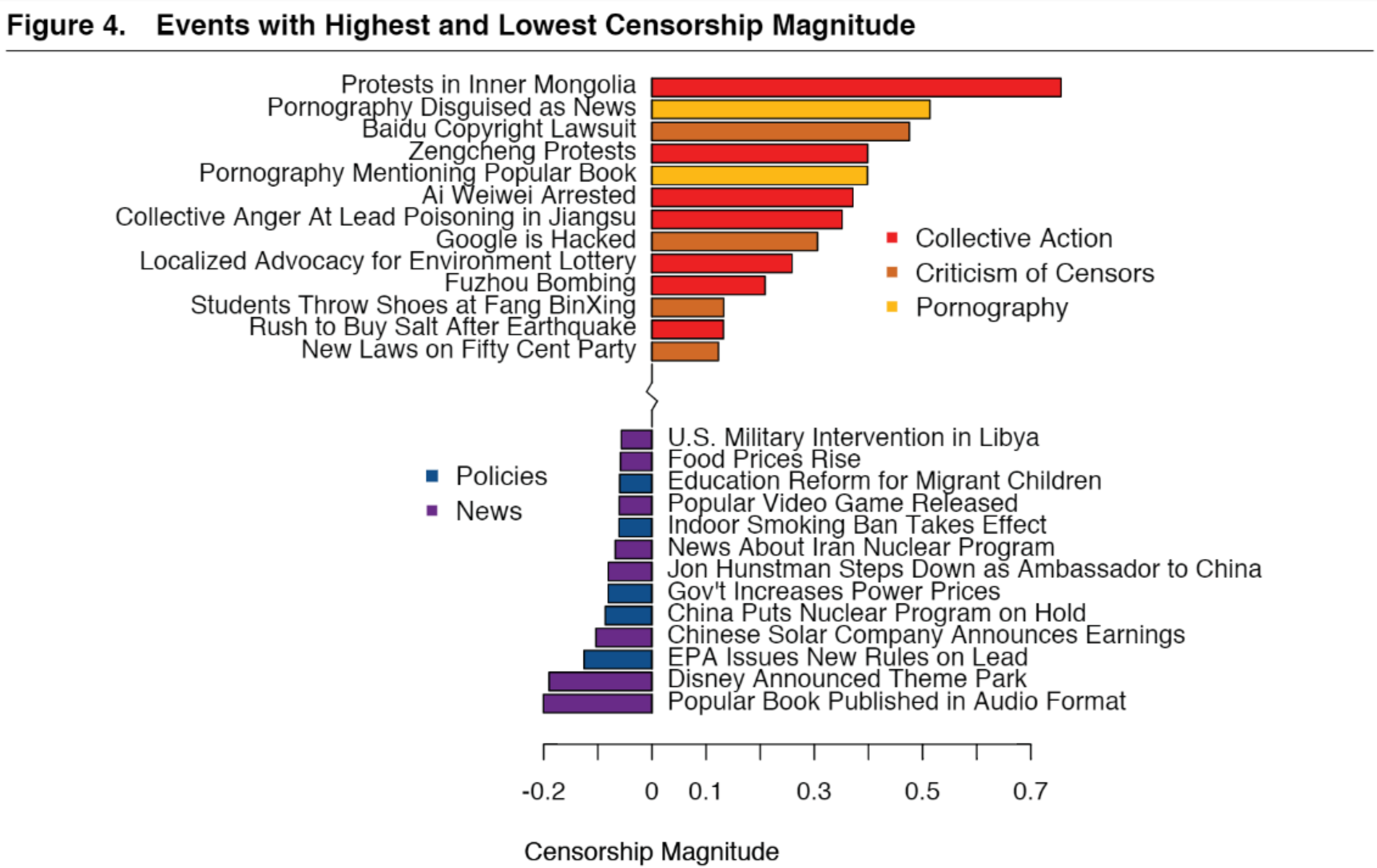
Robinson (2016)

Track censorship



Track censorship

What is the agenda of the censorship program of the Chinese government?



King, Pan and Roberts (2013)

The R language

R and RStudio

R is a free software environment for statistical computing and graphics (and a programming language)

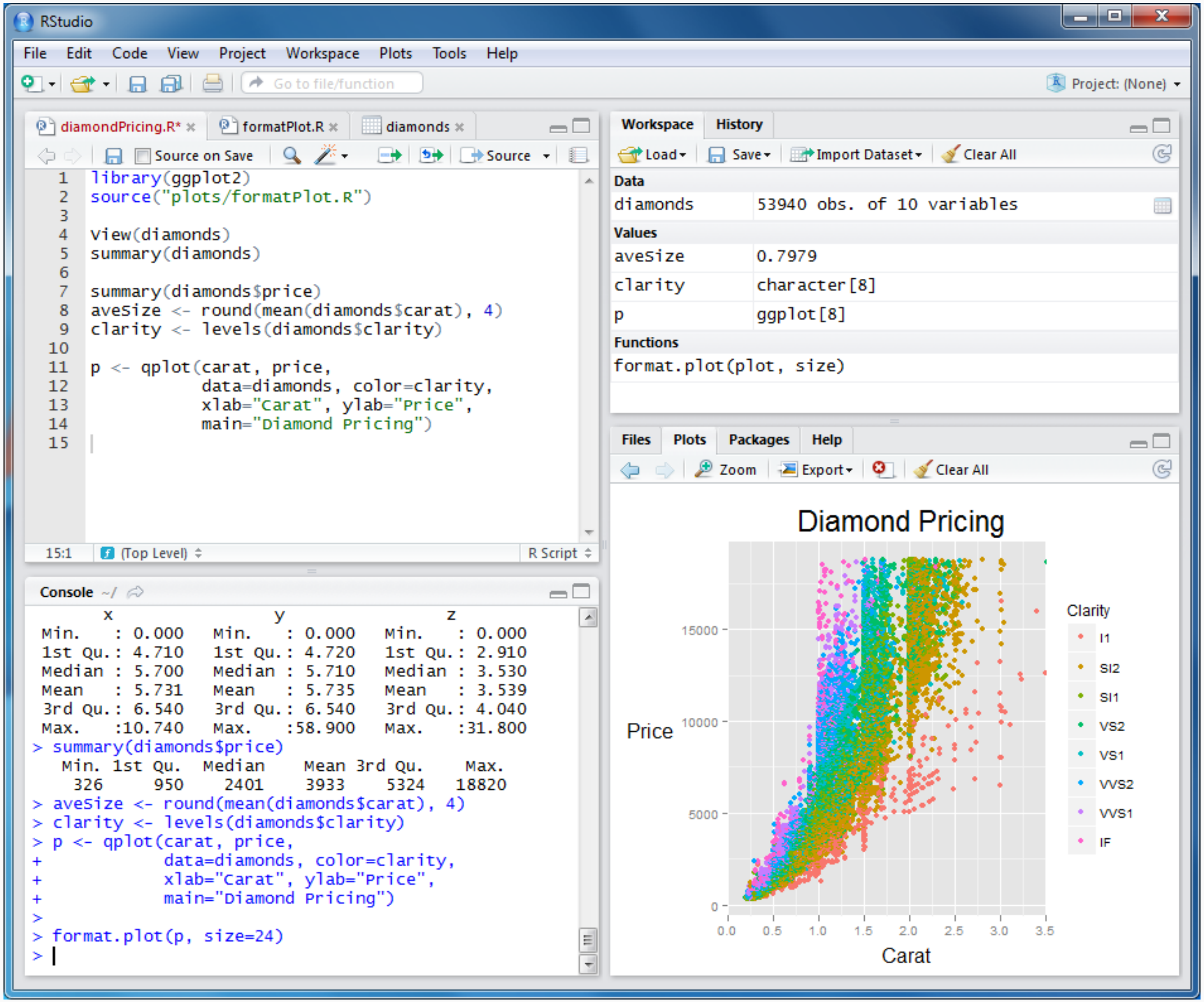
RStudio is an integrated development environment (IDE) for R

Loosely speaking, it makes using R more convenient!

Please download and install base R and RStudio Desktop before the next session (see tutorial).

Parts of RStudio

1. **Scripts**: Recipe of what to do
2. **Console**: executes commands
3. **Workspace**: Memory of what is currently loaded
4. **Files/Plots/Help**: Miscellaneous functions such as displaying files, plots, and help files

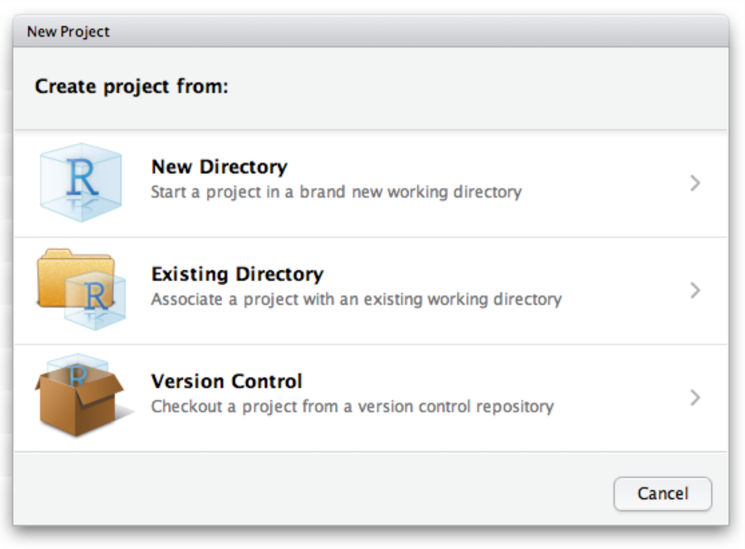


Projects in RStudio

Where did R just save the file to? Its "working directory".

Each project should have its own encapsulation using an RStudio project

Upper right corner of RStudio, click new Projects



Then the working directory is set where the project (`.Rproj`) is saved.

Fore more, read this blog

Outline Intro R

- Anatomy of Code: How does code look like what are components of it?
- R as a (fancy) Calculator: Simple maths with R
- Comparison and Logical Operations: How can we compare data?
- Data Types: What kind of data does R recognize?
- Data Structures: How can we store data?
- Conditional Statements: Managing workflow
- Anatomy of a Function: What are the elements of a function?
- Libraries: How can we use predefined functions

We need to have a basic understanding of highlighted topics before we start with QTA in R!

Anatomy of Code

- Comments
- Variables
- Conditionals
- Functions
- Libraries

```r
# Load data, run a regression, and export the results
library(tidyverse)
library(texreg)

# should the result be saved to a file?
save_to_file <- TRUE

# loads the data from file
cars_data <- read_csv("data/cars_data.csv")

# performs the regression
model <- lm(mpg ~ wt + cyl + hp, data = cars_data)

# converts the regression model to a tex-string
tex_string <- texreg(model)

# check if the result should be saved to a file
if (save_to_file) {
  # saves the tex-string to a file
  write_lines(tex_string, "written/tables/cars_regression.tex")
}
```

Anatomy of Code

- Comments
- Variables
- Conditionals
- Functions
- Libraries

```
# Load data, run a regression, and export the results
library(tidyverse)
library(texreg)

# should the result be saved to a file?
save_to_file <- TRUE

# loads the data from file
cars_data <- read_csv("data/cars_data.csv")

# performs the regression
model <- lm(mpg ~ wt + cyl + hp, data = cars_data)

# converts the regression model to a tex-string
tex_string <- texreg(model)

# check if the result should be saved to a file
if (save_to_file) {
  # saves the tex-string to a file
  write_lines(tex_string, "written/tables/cars_regression.tex")
}
```

Anatomy of Code

- Comments
- Variables
  - Assignment
  - Usage
- Conditionals
- Functions
- Libraries

```
# Load data, run a regression, and export the results
library(tidyverse)
library(texreg)

# should the result be saved to a file?
save_to_file <- TRUE

# loads the data from file
cars_data <- read_csv("data/cars_data.csv")

# performs the regression
model <- lm(mpg ~ wt + cyl + hp, data = cars_data)

# converts the regression model to a tex-string
tex_string <- texreg(model)

# check if the result should be saved to a file
if (save_to_file) {
  # saves the tex-string to a file
  write_lines(tex_string, "written/tables/cars_regression.tex")
}
```

Anatomy of Code

- Comments
- Variables
  - Assignment
  - Usage
- Conditionals
- Functions
- Libraries

```r
# Load data, run a regression, and export the results
library(tidyverse)
library(texreg)

# should the result be saved to a file?
save_to_file <- TRUE

# loads the data from file
cars_data <- read_csv("data/cars_data.csv")

# performs the regression
model <- lm(mpg ~ wt + cyl + hp, data = cars_data)

# converts the regression model to a tex-string
tex_string <- texreg(model)

# check if the result should be saved to a file
if (save_to_file) {
  # saves the tex-string to a file
  write_lines(tex_string, "written/tables/cars_regression.tex")
}
```

Anatomy of Code

- Comments
- Variables
- Conditionals
- Functions
- Libraries

```r
# Load data, run a regression, and export the results
library(tidyverse)
library(texreg)

# should the result be saved to a file?
save_to_file <- TRUE

# loads the data from file
cars_data <- read_csv("data/cars_data.csv")

# performs the regression
model <- lm(mpg ~ wt + cyl + hp, data = cars_data)

# converts the regression model to a tex-string
tex_string <- texreg(model)

# check if the result should be saved to a file
if (save_to_file) {
  # saves the tex-string to a file
  write_lines(tex_string, "written/tables/cars_regression.tex")
}
```

Anatomy of Code

- Comments
- Variables
- Conditionals
- Functions
  - Function Call
  - Function Parameters
- Libraries

```
# Load data, run a regression, and export the results
library(tidyverse)
library(texreg)

# should the result be saved to a file?
save_to_file <- TRUE

# loads the data from file
cars_data <- read_csv("data/cars_data.csv")

# performs the regression
model <- lm(mpg ~ wt + cyl + hp, data = cars_data)

# converts the regression model to a tex-string
tex_string <- texreg(model)

# check if the result should be saved to a file
if (save_to_file) {
  # saves the tex-string to a file
  write_lines(tex_string, "written/tables/cars_regression.tex"
}
```

Anatomy of Code

- Comments
- Variables
- Conditionals
- Functions
  - Function Call
  - Function Parameters
- Libraries

```r
# Load data, run a regression, and export the results
library(tidyverse)
library(texreg)

# should the result be saved to a file?
save_to_file <- TRUE

# loads the data from file
cars_data <- read_csv("data/cars_data.csv")

# performs the regression
model <- lm(mpg ~ wt + cyl + hp, data = cars_data)

# converts the regression model to a tex-string
tex_string <- texreg(model)

# check if the result should be saved to a file
if (save_to_file) {
  # saves the tex-string to a file
  write_lines(tex_string, "written/tables/cars_regression.tex"
}
```

Anatomy of Code

- Comments
- Variables
- Conditionals
- Functions
- Libraries

```r
# Load data, run a regression, and export the results
library(tidyverse)
library(texreg)

# should the result be saved to a file?
save_to_file <- TRUE

# loads the data from file
cars_data <- read_csv("data/cars_data.csv")

# performs the regression
model <- lm(mpg ~ wt + cyl + hp, data = cars_data)

# converts the regression model to a tex-string
tex_string <- texreg(model)

# check if the result should be saved to a file
if (save_to_file) {
  # saves the tex-string to a file
  write_lines(tex_string, "written/tables/cars_regression.tex")
}
```

R as a (fancy) calculator

Basic Oparators

basic

- `+`, `-`, `*`, `/`

power

- `^`, `sqrt()`

logarithmic

- `log()`, `exp()`

modulus and int division

- `%%`, `%/%`

```r
1 + 1
## [1] 2
2 - 2
## [1] 0
3 * 3
## [1] 9
4 / 4
## [1] 1
5^5
## [1] 3125
```

```r
sqrt(9)
## [1] 3
log(2.7182)
## [1] 0.9999699
exp(1)
## [1] 2.718282
10 %% 3
## [1] 1
10 %/% 3
## [1] 3
```

Data Types

R knows different types of data

Name	**Description**	**Examples**
`logical`	Boolean values	`TRUE`, `FALSE`, `NA`
`integer`	Integer numbers	`1L`, `-10L`, `0L`, `NA`
`numeric`	Numeric values	`1`, `-0.3`, `1/3`, `NA`
`character`	Characters	`"hello"`, `"1.0"`, `"TRUE"`, `NA`
...	...	

Other types (that we won't really use for now) include `factors`, `complex`, ...

Comparison Operators

Operator	Name	Example	Result
`<`	Smaller	`3 < 5`	`TRUE`
`<=`	Smaller Equal	`3 <= 3`	`TRUE`
`>`	Larger	`3 > 5`	`FALSE`
`>=`	Larger Equal	`5 >= 3`	`TRUE`
`==`	Equal	`"Alice" == "Bob"`	`FALSE`
`!=`	Not Equal	`"Alice" != 5`	`TRUE`

Also `%in%` (checks if a value is in a vector), `"Alice" %in% c("Alice", "Bob")` evaluates to `TRUE`.

Logical Operators

<table>
<tr><th>Operator</th><th>Name</th><th>Example</th><th>Result</th></tr>
<tr><td>&</td><td>And</td><td>3 == 5 & 4 < 5</td><td>FALSE</td></tr>
<tr><td>|</td><td>Or</td><td>3 == 5 | 4 < 5</td><td>TRUE</td></tr>
<tr><td>!</td><td>Not</td><td>!(4 < 5)</td><td>FALSE</td></tr>
</table>

Data Structures

- Single Type
  - Vectors
  - Matrix
- Multiple Types

Using the `c()` function to combine values into a vector

```
heights <- c(186, 176, 165, 172, 187)
heights
## [1] 186 176 165 172 187
# only one type per vector
vec <- c(160, "Alice", TRUE)
vec
## [1] "160"   "Alice" "TRUE"
```

Data Structures

- Single Type
  - Vectors
  - Matrix
- Multiple Types

Creating a sequence using `:` or `seq()`

```
1:10
##  [1]  1  2  3  4  5  6  7  8  9 10
seq(1, 10, 2)
## [1] 1 3 5 7 9
seq(1, 10, length.out = 5)
## [1]  1.00  3.25  5.50  7.75 10.00
```

Data Structures

- Single Type
  - Vectors
  - Matrix
- Multiple Types

Create a matrix from a vector using `matrix()`

```
mat <- matrix(1:9, nrow = 3)
mat
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
# Only one type per matrix
mat2 <- matrix(c(1, 2L, TRUE, "Bob"), nrow = 2)
mat2
##      [,1] [,2]
## [1,] "1"  "TRUE"
## [2,] "2"  "Bob"
```

Data Structures

- Single Type
- Multiple Types
    - List
    - Data.Frames

Create a list where each element can have a different type using `list`

```
# a named vector
person <- list(name = "Alice",
               children = list(
                 list(name = "Bobby"),
                 list(name = "Charlie")
               )
)
person
## $name
## [1] "Alice"
##
## $children
## $children[[1]]
## $children[[1]]$name
## [1] "Bobby"
##
##
## $children[[2]]
## $children[[2]]$name
## [1] "Charlie"
```

Data Structures

- Single Type
- Multiple Types
  - List
  - Data.Frames

Create a data.frame from vectors, where each variable has one type (is a vector)

```
people <- data.frame(
  name = c("Alice", "Bob", "Charlie"),
  age  = c(40, 25, 15),
  num_children = c(2, 1, 0),
  has_children = c(TRUE, TRUE, FALSE)
)
people
##      name age num_children has_children
## 1   Alice  40            2         TRUE
## 2     Bob  25            1         TRUE
## 3 Charlie  15            0        FALSE
```

Basic Data Access

Using the `[]`-operator we can access elements of a vector (1D), a matrix (2D), a list (n-dimensional), a data.frame (2D).

```
heights <- c(186, 204, 176, 165, 182)
# accesses the third element
heights[3]
## [1] 176
# accesses certain elements
heights[c(2, 1, 3, 5, 4)]
## [1] 204 186 176 182 165
```

```
# compare if an element is
# larger than 180
heights > 180
## [1]  TRUE  TRUE FALSE FALSE  TRUE
# take elements that are
# larger than 180
heights[heights > 180]
## [1] 186 204 182
```

Basic Data Access

The `[]`-operator can also be used for 2D structures as `mat[row, col]`

```
mat <- matrix(1:9, nrow = 3)
mat
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9

# take the first row, third column
mat[1, 3]
## [1] 7
```

```
# take only the first row
mat[1, ]
## [1] 1 4 7

# take only the first column
mat[, 1]
## [1] 1 2 3

# take the first and second row,
# first and third column
mat[1:2, c(1, 3)]
##      [,1] [,2]
## [1,]    1    7
## [2,]    2    8
```

Conditionals

For workflow management, sometimes we need want to evaluate (=execute) code only if certain conditions are met

- if call
- Condition
- Body
- (possible) else-if
- (possible) else

```
if (condition1) {
  print("condition1 is true!")
} else if (condition2) {
  print("condition1 is not true but condition2 is true!")
} else {
  print("Neither condition1 nor condition2 is true!")
}
```

Conditionals

For workflow management, sometimes we need want to evaluate (=execute) code only if certain conditions are met

- if call
- Condition
- Body
- (possible) else-if
- (possible) else

```
if (condition1) {
  print("condition1 is true!")
} else if (condition2) {
  print("condition1 is not true but condition2 is true!")
} else {
  print("Neither condition1 nor condition2 is true!")
}
```

Conditionals

For workflow management, sometimes we need want to evaluate (=execute) code only if certain conditions are met

- if call
- Condition
- Body
- (possible) else-if
- (possible) else

```
if (condition1) {
  print("condition1 is true!")
} else if (condition2) {
  print("condition1 is not true but condition2 is true!")
} else {
  print("Neither condition1 nor condition2 is true!")
}
```

Conditionals

For workflow management, sometimes we need want to evaluate (=execute) code only if certain conditions are met

- if call
- Condition
- Body
- (possible) else-if
- (possible) else

```
if (condition1) {
  print("condition1 is true!")
} else if (condition2) {
  print("condition1 is not true but condition2 is true!")
} else {
  print("Neither condition1 nor condition2 is true!")
}
```

Conditionals

For workflow management, sometimes we need want to evaluate (=execute) code only if certain conditions are met

- if call
- Condition
- Body
- (possible) else-if
- (possible) else

```
if (condition1) {
  print("condition1 is true!")
} else if (condition2) {
  print("condition1 is not true but condition2 is true!")
} else {
  print("Neither condition1 nor condition2 is true!")
}
```

Conditionals

For workflow management, sometimes we need want to evaluate (=execute) code only if certain conditions are met

- if call
- Condition
- Body
- (possible) else-if
- (possible) else

```
if (condition1) {
  print("condition1 is true!")
} else if (condition2) {
  print("condition1 is not true but condition2 is true!")
} else {
  print("Neither condition1 nor condition2 is true!")
}
```

Conditionals

We can have conditional statements with only the `if`-part, ...

```
if (rains) {
  take_umbrella()
}
```

... an `if` and an `else` case, ...

```
if (rains) {
  take_umbrella
} else {
  wear_shorts()
}
```

Conditionals

…, multiple `else if`s (and multiple `else if`s and an `else` case)

```
if (rains) {
  take_umbrella()
} else if (sunny) {
  apply_sunsreen()
} else if (snows) {
  go_snowboarding()
} else if (windy) {
  go_surfing()
}
```

Q: What happens if its windy and sunny?

Q: What can we do if we want to go surfing and apply sunscreen?

Conditionals short and vectorized

The `if () {} else {}` functions evaluate only the first element of the condition.

What can we use to apply the if-else statement to a vector of values? `ifelse()`

```
ifelse(condition,
       "Value if condition is true",
       "Value if condition is false")
```

```
heights <- c(186, 204, 176, 165, 182)
ifelse(heights > 180, "above 180", "below 180")
## [1] "above 180" "above 180" "below 180" "below 180" "above 180"
```

```
ifelse(heights > 180,
       "h > 180",
       ifelse(heights > 170,
              "170 < h <= 180",
              "h <= 170"))
## [1] "h > 180"        "h > 180"        "170 < h <= 180" "h <= 170"       "h > 180"
```

Function introduction

Does this make sense?

```
print("Good morning Alice")
print("Good evening Alice")
print("Good evening Bob")
print("Good morning Bob")
print("Good day Charlie")
print("Good day Alice")
```

What happens if we want to change "Good" to something else or we find that we have some spelling error and need to find and change every part of it?

Don't repeat yourself!

Use functions!

Anatomy of a Function

- Assign
- Arguments
- Body
- Return Values
- Function Call

Function: Write once, use often

```
# define the function 'greet' once:
greet <- function(daytime, name) {
  text <- paste("Good", daytime, name)
  text
}

# use the function 'greet' as often as needed
result <- greet("morning", "Alice")
result
## [1] "Good morning Alice"
# or print result of the function directly to the console
greet("evening", "Alice")
## [1] "Good evening Alice"
```

Anatomy of a Function

- Assign
- Arguments
- Body
- Return Values
- Function Call

Assigning a function to `greet`

```
# define the function 'greet' once:
greet <- function(daytime, name) {
  text <- paste("Good", daytime, name)
  text
}

# use the function 'greet' as often as needed
result <- greet("morning", "Alice")
```

Anatomy of a Function

- Assign
- Arguments
- Body
- Return Values
- Function Call

Arguments are variables passed into the body of the functions

```
# define the function 'greet' once:
greet <- function(daytime, name) {
  text <- paste("Good", daytime, name)
  text
}

# use the function 'greet' as often as needed
result <- greet("morning", "Alice")
```

Anatomy of a Function

- Assign
- Arguments
- Body
- Return Values
- Function Call

Body is executed everytime the function is called

```
# define the function 'greet' once:
greet <- function(daytime, name) {
  text <- paste("Good", daytime, name)
  text
}

# use the function 'greet' as often as needed
result <- greet("morning", "Alice")
```

Anatomy of a Function

- Assign
- Arguments
- Body
- Return Values
- Function Call

Return: the last element that is called is returned

```
# define the function 'greet' once:
greet <- function(daytime, name) {
  text <- paste("Good", daytime, name)
  text # the last element is returned
}

# use the function 'greet' as often as needed
result <- greet("morning", "Alice")
```

The function could be also shortened to

```
greet <- function(daytime, name) {
   paste("Good", daytime, name)
}
# or even shorter
greet <- function(daytime, name) paste("Good", daytime, name)
```

Anatomy of a Function

- Assign
- Arguments
- Body
- Return Values
- Function Call

Call the function using its name and parenthesis `()` with arguments

```
# define the function 'greet' once:
greet <- function(daytime, name) {
  text <- paste("Good", daytime, name)
  text
}

# use the function 'greet' as often as needed
result <- greet("morning", "Alice")
result
```

```
## [1] "Good morning Alice"
```

```
# named function call also possible
greet(name = "Bob", daytime = "night")
## [1] "Good night Bob"
```

Default Values

We can also specify default values for function arguments in the definition of the function

```r
greet <- function(daytime = "morning", name = "Alice") {
  text <- paste("Good", daytime, name)
  text
}
```

```r
greet("evening") # no 'name' supplied, take default "Alice"
## [1] "Good evening Alice"
```

```r
greet() # neither 'daytime' nor 'name' supplied, take both default values
## [1] "Good morning Alice"
```

```r
greet(name = "Bob") # no daytime supplied, take default
## [1] "Good morning Bob"
```

Objects and Functions

> To understand computations in R, two slogans are helpful:
>
> - Everything that **exists** is an object [variable]
> - Everything that **happens** is a function call

John Chambers

Example Code

```
x <- rnorm(100)
mean(x)
```

Objects and Functions

> To understand computations in R, two slogans are helpful:
>
> - Everything that **exists** is an object [variable]
> - Everything that **happens** is a function call

John Chambers

Example Code

```
x <- rnorm(100)
mean(x)
```

Objects and Functions

> To understand computations in R, two slogans are helpful:
>
> - Everything that **exists** is an object (variable)
> - Everything that **happens** is a function call

John Chambers

Example Code

```
x <- rnorm(100)
mean(x)
```

From Functions to Libraries

When we use R and its functions (e.g., linear regression), we don't want to implement every function over and over again but want to use existing functions.

Functions can be packaged into a **Package/Library**, which is then easy to distribute and install on different machines.

Main distribution channel is called CRAN (Comprehensive R Archive Network) https://cran.r-project.org/ with over 12,000 packages.

To **install** packages from CRAN we use the `install.packages("packageName")`. Note, this needs to be done only once, thus we use the console!

In every session, we need to **load** the package using `library(packageName)`.

```r
# install the package tidyverse once (console)
install.packages("tidyverse")

# on top of each script, we write (NOT CONSOLE!)
library(tidyverse)
```

References

Grimmer, Justin, and Brandon M. Stewart. 2013. "Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts." Political Analysis 21 (3): 267–97. https://doi.org/10.1093/pan/mps028.

King, Gary, Jennifer Pan, and Margaret E. Roberts. 2013. "How Censorship in China Allows Government Criticism but Silences Collective Expression." The American Political Science Review 107 (2): 326–43.