REGULAR EXPRESSIONS

Get → **Clean** → **Transform** → Visualize → Model → Communicate

Understand

Program

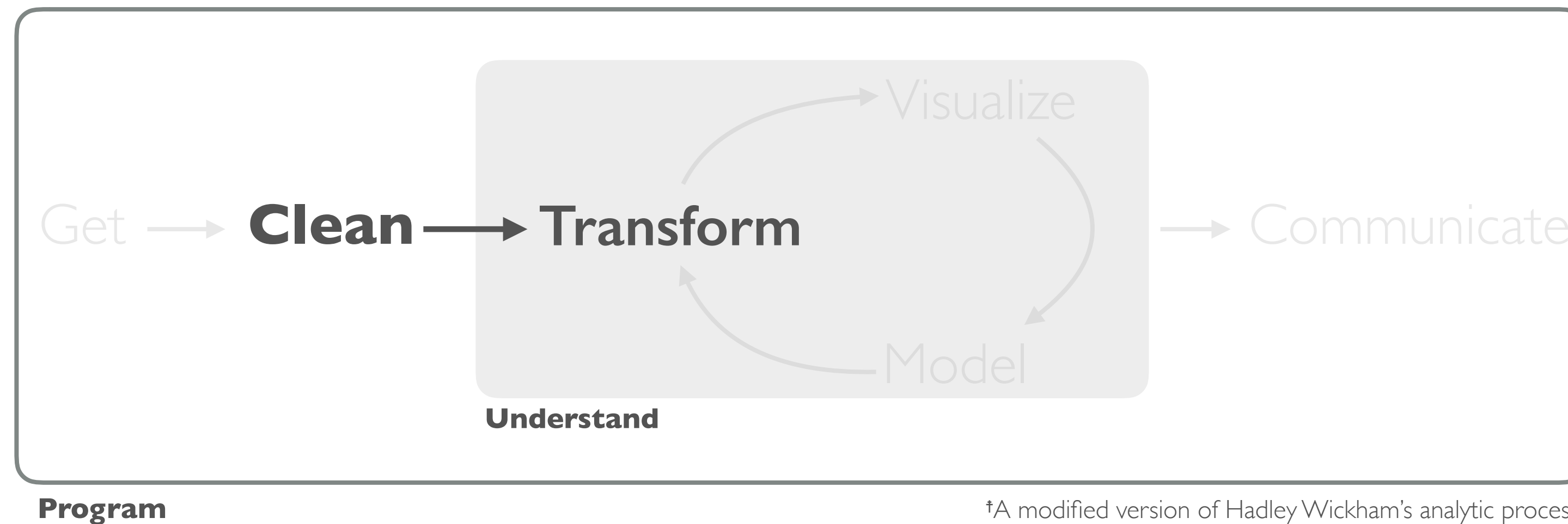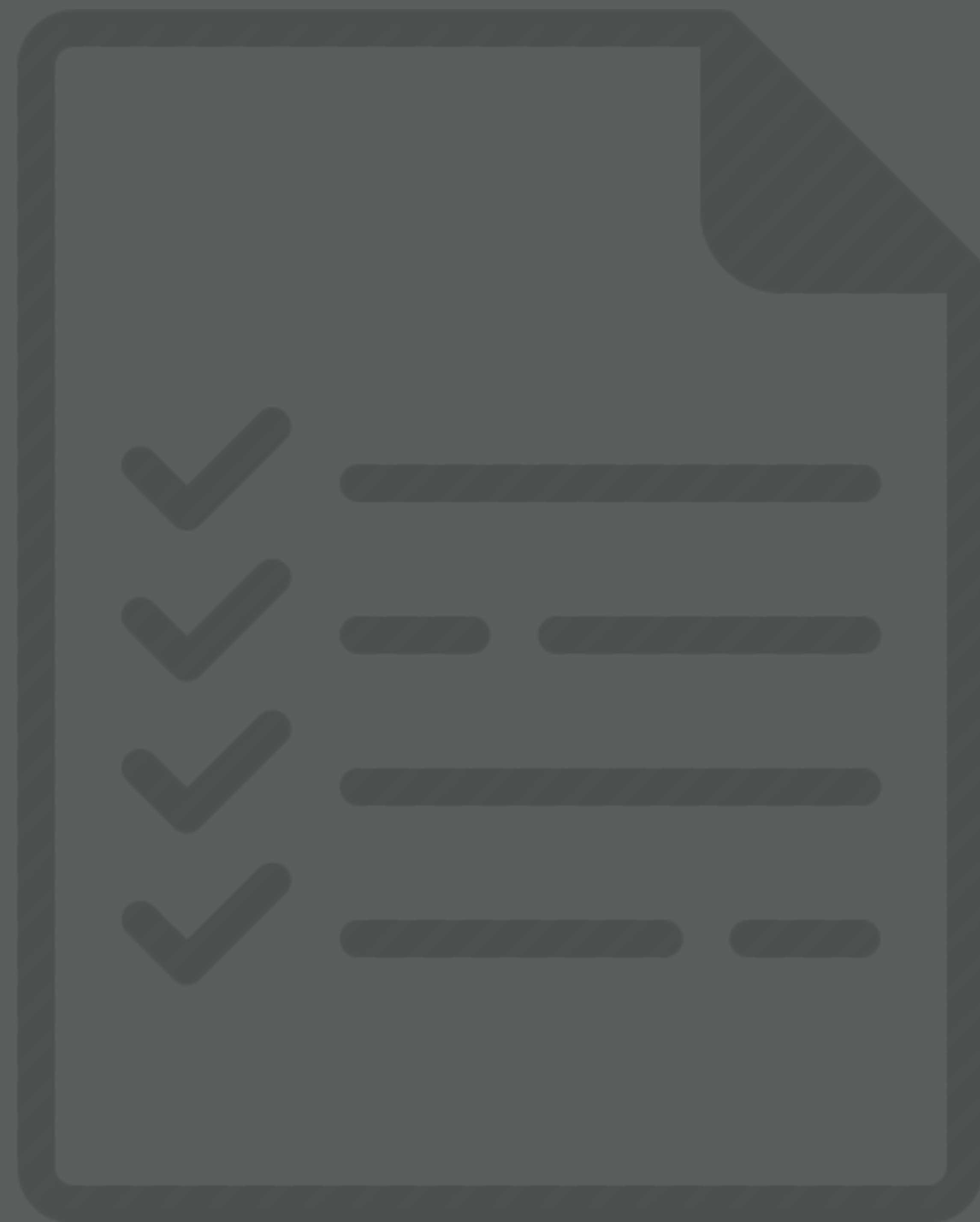"Analysts are often trained to handle tabular or rectangular data that are mostly numeric, but much of the data proliferating today are unstructured and text-heavy."

– Julia Silge and David Robinson

PREREQUISITES

PACKAGE PREREQUISITE

```r
library(tidyverse)

if (packageVersion("devtools") < 1.6) {
  install.packages("devtools")
}

devtools::install_github("bradleyboehmke/harrypotter")
library(harrypotter)
```

REGULAR EXPRESSIONS

- Regular expressions (regex) are strings to identify patterns in text

- Two areas of focus:

  i. regex functions

  ii. regex syntax

- The `stringr` package provides us a convenient approach to regex text mining

- We'll explore dealing with regex in both character strings and data frames

REGEX FUNCTIONS

Dealing with character strings

DATA PREREQUISITE

philosophers_stone

 [1] "THE BOY WHO LIVED   Mr. and Mrs. Dursley, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much. They were the last people you'd expect to be involved in anything strange or mysterious, because they just didn't hold with such nonsense.   Mr. Dursley was the director of a firm called Grunnings, which made drills. He was a big, beefy man with hardly any neck, although he did have a very large mustache. Mrs. Dursley was thin and blonde and had nearly twice the usual amount of neck, which came in very useful as she spent so much of her time craning over garden fences, spying on the neighbors. The Dursleys had a small son called Dudley and in their opinion there was no finer boy anywhere.   The Dursleys had everything they wanted, but they also had a secret, and their greatest fear was that somebody would discover it. They didn't think they could bear it if anyone found out about the Potters. Mrs. Potter was Mrs. Dursley's sister, but they h... <truncated>

 [2] "THE VANISHING GLASS   Nearly ten years had passed since the Dursleys had woken up to find their nephew on the front step, but Privet Drive had hardly changed at all. The sun rose on the same tidy front gardens and lit up the brass number four on the Dursleys' front door; it crept into their living room, which was almost exactly the same as it had

str_*

```
str_*(string, pattern)
```

string: character vector

pattern: regex pattern to look for

str_*

```
str_detect(philosophers_stone, "Harry")
 [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[10] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

str_detect: does the expression exist?

str_*

```
str_count(philosophers_stone, "Harry")
 [1]   20   79   68   48  147  117   63   49   70   82   68
[12]   99   62   45   93  116   99
```

str_count: how many instances are there?

str_*

```
str_detect(philosophers_stone, "Harry")

str_count(philosophers_stone, "Harry")

str_extract(philosophers_stone, "Harry")
 [1] "Harry" "Harry" "Harry" "Harry" "Harry"
 [6] "Harry" "Harry" "Harry" "Harry" "Harry"
[11] "Harry" "Harry" "Harry" "Harry" "Harry"
[16] "Harry" "Harry"
```

str_extract: extract the **first** instance

str_*

```
str_detect(philosophers_stone, "Harry")

str_count(philosophers_stone, "Harry")

str_extract(philosophers_stone, "Harry")

str_extract_all(philosophers_stone, "Harry")
[[1]]
 [1] "Harry" "Harry" "Harry" "Harry" "Harry"
 [6] "Harry" "Harry" "Harry" "Harry" "Harry"
[11] "Harry" "Harry" "Harry" "Harry" "Harry"
[16] "Harry" "Harry" "Harry" "Harry" "Harry"


[[2]]
 [1] "Harry" "Harry" "Harry" "Harry" "Harry"
 [6] "Harry" "Harry" "Harry" "Harry" "Harry"
[11] "Harry" "Harry" "Harry" "Harry" "Harry"
[16] "Harry" "Harry" "Harry" "Harry" "Harry"
```

str_extract_all: extract **all** instances

str_*

```
str_detect(philosophers_stone, "Harry")

str_count(philosophers_stone, "Harry")

str_extract(philosophers_stone, "Harry")

str_extract_all(philosophers_stone, "Harry")

str_locate_all(philosophers_stone, "Harry")
[[1]]
       start    end
 [1,]   5243   5247
 [2,]   5798   5802
 [3,]   5868   5872
 [4,]  10231  10235
 [5,]  18057  18061
 [6,]  18190  18194
 [7,]  18521  18525
```

str_locate_all: locate the position of **all** instances

YOUR TURN!

Take 5 minutes to explore the various `str_*` *functions*

REGEX SYNTAX

Dealing with character strings

([a-z][^a-z0-9

```
str_count(philosophers_stone, "Harry  Potter")

[1] 5 2 0 2 3 5 1 1 0 0 0 0 0 1 5 0 3
```

Phrases

MULTIPLE WORDS / CASE SENSITIVE

```
str_count(philosophers_stone, "Harry  Potter")
```

```
str_count(philosophers_stone, "Harry|Potter")
```

 [1]   28   57   56   30 109   76   50   44   55   67   52   73   52

[14]   31   70   80   61

"Harry" or **"Potter"**

```
str_count(philosophers_stone, "Harry  Potter")

str_count(philosophers_stone, "Harry | Potter")

str_count(philosophers_stone, "ye(s|ah)")

 [1] 17  5  3  7 25  9 13  5  8  4  5 10  1  4  8 12  8
```

"yes" or "yeah"

```
str_count(philosophers_stone, "Harry  Potter")

str_count(philosophers_stone, "Harry | Potter")

str_count(philosophers_stone, "ye(s|ah)")

str_count(philosophers_stone, "boy")                                    ← default

 [1]  9  7  2  5 15 34  5  1  7  1  0  3  0  0  3  2  6


str_count(philosophers_stone, regex("boy", ignore_case = TRUE))         ← ignore case

 [1] 10  7  2  5 15 34  5  1  7  1  0  3  0  0  3  2  6
```

YOUR TURN!

How many times are "Mr" and "Mrs" used in `philosophers_stone`*?*

SOLUTION

```
str_count(philosophers_stone, "Mr\\.\\s|Mrs\\.\\s") %>% sum
[1] 126
```

ANCHORS

```
str_count(deathly_hallows, "^Harry")
 [1] 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0
[28] 1 0 0 0 0 1 0 0 0 0
```

^: Identify patterns at the **beginning** of an element

ANCHORS

```
str_count(deathly_hallows, "^Harry")

str_count(philosophers_stone,
          regex("end$", ignore_case = TRUE))
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

$: Identify patterns at the **end** of an element

YOUR TURN!

Extract all elements in `deathly_hallows` *that start with* *"Harry"*

SOLUTION

```
deathly_hallows[str_detect(deathly_hallows, "^Harry")]
```
 [1] "Harry was bleeding. Clutching his right hand in his left and swearing under his
breath, he shouldered open his bedroom door. There was a crunch of breaking china. He had
trodden on a cup of cold tea that had been sitting on the floor outside his bedroom door.
\"What the ¨C?\"He looked around, the landing of number four, Privet Drive, was deserted.
Possibly the cup of tea was Dudley's idea of a clever booby trap. Keeping his bleeding hand
elevated, Harry scraped the fragments of cup together with the other hand and threw them
into the already crammed bin just visible inside his bedroom door. Then he tramped across
to the bathroom to run his finger under the tap.It was stupid, pointless, irritating beyond
belief that he still had four days left of being unable to perform magic...but he had to
admit to himself that this jagged cut in his finger would have defeated him. He had never
learned how to repair wounds, and now he came to think of it ¨C particularly in light of
his immediate ... <truncated>

 [2] "Harry ran back upstairs to his bedroom, arriving at the window just in time to see
the Dursleys' car swinging out of the drive and off up the road. Dedalus's top hat was
visible between Aunt Petunia and Dudley in the backseat. The car turned right at the end of
Privet Drive, its windows burned scarlet for a moment in the now setting sun, and then it
was gone.Harry picked up Hedwig's cage, his Firebolt, and his rucksack, gave his
```

# SPECIAL PATTERNS

```
str_extract(philosophers_stone, "Harry.")
 [1] "Harry\"" "Harry "  "Harry "   "Harry!"
 [5] "Harry "  "Harry'"  "Harry'"   "Harry "
 [9] "Harry "  "Harry "  "Harry "   "Harry "
[13] "Harry "  "Harry,"  "Harry'"   "Harry "
[17] "Harry."
```

.: wild card - **any character**

SPECIAL PATTERNS

```
str_extract(philosophers_stone, "Harry.")

str_extract(philosophers_stone, "\\d")
 [1] NA  NA  "4" "1" "0" "1" NA  "3" "3" NA  "1" NA
[13] "1" "1" NA  "1" NA
```

\\d: digits

SPECIAL PATTERNS

```
str_detect(philosophers_stone, "Harry")

str_extract(philosophers_stone, "\\d")

str_extract(philosophers_stone, "[1|4]")
 [1] NA   NA   "4" "1" "1" "1" NA   "1" NA   NA   "1" NA
[13] "1" "1" NA   "1" NA
```

[d|d]: specified digits

SPECIAL PATTERNS

```
str_detect(philosophers_stone, "Harry")

str_count(philosophers_stone, "Harry")

str_extract(philosophers_stone, "[1|4]")

str_extract(philosophers_stone, ".[yz].")
 [1] "ey," "ly " "azi" "ey " "ry " "ry'" "ry " " yo"
 [9] "ry " "oy " "ey " "oze" "ry " "ey'" "dy " " ye"
[17] "ry."
```

[a-zA-Z]: specified letters

YOUR TURN!

How many times is the word "Harry" get followed by a word that starts with a vowel in `philosophers_stone`*?*

SOLUTION

```
str_count(philosophers_stone, "Harry.[aeiou]") %>% sum
str_count(philosophers_stone, "Harry\\s[aeiou]") %>% sum
[1] 156
```

This is the safer option since **\\s** specifically identifies white space

REPETITION

```
str_extract(philosophers_stone, "[aeiou]{4}")

str_extract(philosophers_stone, "[aeiou]{3,}")

str_extract(philosophers_stone, "[aeiou]{3,4}")
 [1] "iou" "uie" "iou" "uee" "iou" "iou" "uie" "uea"
 [9] "uie" "iou" "iou" "iou" "eei" "iou" "iou" "iou"
[17] "uie"
```

{n}: find n repetitions

{n,}: find n or more repetitions

{n,m}: find n to m repetitions

YOUR TURN!

1. *Without computer support, what is this finding:*

   ```
   str_count(philosophers_stone, regex("((no[[:punct:]])[ ]){3}", ignore_case = TRUE))
   ```

2. *Extract the 25 characters that precede and follow the use of "Harry" in* **philosophers_stone**

SOLUTION

```
str_count(philosophers_stone, regex("((no[[:punct:]])[ ]){3}", ignore_case = TRUE))
```
Looking for 3 repeated uses of "no" with any form of punctuation and a space following each use of "no". (i.e. "No, no, no!")

```
str_extract_all(philosophers_stone, ".{25}Harry.{25}")
 [1] " I heard yes, their son, Harry\"   Mr. Dursley stopped de"
 [2] "ter who had a son called Harry. Come to think of it, he"
 [3] "re his nephew was called Harry. He'd never even seen th"
 [4] "in? Howard, isn't it?\"   \"Harry. Nasty, common name, if "
 [5] "o kill the Potter's son, Harry. But -- he couldn't. He "
 [6] "at when he couldn't kill Harry Potter, Voldemort's powe"
 [7] "n the name of heaven did Harry survive?\"   \"We can only "
 [8] "s?\"   \"I've come to bring Harry to his aunt and uncle. T"
 [9] "t, screaming for sweets. Harry Potter come and live her"
[10] "ed if today was known as Harry Potter day in the future"
[11] "] he books written about Harry   every child in our wo"
```

REGEX

Doing similar stuff with a data frame

DATA PREREQUISITE

```
airbnb <- read_rds("data/airbnb.rds")

airbnb
# A tibble: 3,585 × 95
          id                           listing_url      scrape_id last_scraped
       <int>                                 <chr>            <dbl>       <date>
1   12147973 https://www.airbnb.com/rooms/12147973 2.016091e+13   2016-09-07
2    3075044  https://www.airbnb.com/rooms/3075044 2.016091e+13   2016-09-07
3       6976      https://www.airbnb.com/rooms/6976 2.016091e+13   2016-09-07
4    1436513  https://www.airbnb.com/rooms/1436513 2.016091e+13   2016-09-07
5    7651065  https://www.airbnb.com/rooms/7651065 2.016091e+13   2016-09-07
6   12386020 https://www.airbnb.com/rooms/12386020 2.016091e+13   2016-09-07
7    5706985  https://www.airbnb.com/rooms/5706985 2.016091e+13   2016-09-07
8    2843445  https://www.airbnb.com/rooms/2843445 2.016091e+13   2016-09-07
9     753446   https://www.airbnb.com/rooms/753446 2.016091e+13   2016-09-07
10    849408   https://www.airbnb.com/rooms/849408 2.016091e+13   2016-09-07
```

DATA FRAME REGEX MADE EASY

BASICS

```
airbnb %>%
  select(name) %>%
  mutate(character_count = str_count(name))
# A tibble: 3,585 × 2
                                      name character_count
                                     <chr>           <int>
1                Sunny Bungalow in the City              26
2            Charming room in pet friendly apt          33
3            Mexican Folk Art Haven in Boston           32
4  Spacious Sunny Bedroom Suite in Historic Home        45
5                         Come Home to Boston            19
6               Private Bedroom + Great Coffee           30
7            New Lrg Studio apt 15 min to Boston         35
8               "Tranquility" on "Top of the Hill"       34
9            6 miles away from downtown Boston!         34
10              Perfect & Practical Boston Rental        33
# ... with 3,575 more rows
```

- We can use **str_count** to count the number of characters in a character field

BASICS

```
airbnb %>%
  select(name) %>%
  mutate(first_five = str_sub(name, start = 1, end = 5),
         last_five = str_sub(name, start = -5))
# A tibble: 3,585 × 3
                                    name first_five last_five
                                   <chr>      <chr>     <chr>
1              Sunny Bungalow in the City      Sunny      City
2             Charming room in pet friendly apt      Charm     y apt
3                Mexican Folk Art Haven in Boston      Mexic     oston
4   Spacious Sunny Bedroom Suite in Historic Home      Spaci      Home
5                      Come Home to Boston      Come     oston
6             Private Bedroom + Great Coffee      Priva     offee
7            New Lrg Studio apt 15 min to Boston      New L     oston
8               "Tranquility" on "Top of the Hill"      "Tran     Hill"
9               6 miles away from downtown Boston!      6 mil     ston!
10                Perfect & Practical Boston Rental      Perfe     ental
# ... with 3,575 more rows
```

- We can use **str_sub** with start and end arguments to take out a substring

BASICS

```
airbnb %>%
  select(host_name) %>%
  mutate(lower_case = str_to_lower(host_name),
         upper_case = str_to_upper(host_name))
# A tibble: 3,585 × 3
   host_name lower_case upper_case
       <chr>      <chr>      <chr>
1   Virginia   virginia   VIRGINIA
2     Andrea     andrea     ANDREA
3       Phil       phil       PHIL
4     Meghna     meghna     MEGHNA
5      Linda      linda      LINDA
6    Deborah    deborah    DEBORAH
7     Juliet     juliet     JULIET
8    Marilyn    marilyn    MARILYN
9       Sami       sami       SAMI
10     Damon      damon      DAMON
# ... with 3,575 more rows
```

- We can use **str_to_lower** and **str_to_upper** to normalize text case

YOUR TURN!

1. *What is the average number of characters used in the* **name** *column? What about the* **description** *column?*

2. *What is the most common name in the* **host_name** *column?*

SOLUTION

```r
# problem 1
airbnb %>%
  select(name, description) %>%
  mutate(
    name_char = str_count(name),
    desc_char = str_count(description)
    ) %>%
  summarise(
    name_char = mean(name_char, na.rm = TRUE),
    desc_char = mean(desc_char, na.rm = TRUE)
    )
# A tibble: 1 × 2
  name_char desc_char
      <dbl>     <dbl>
1  32.34728  768.5431
```

SOLUTION

```
# problem 2
airbnb %>%
  select(host_name) %>%
  mutate(host_name = str_to_lower(host_name)) %>%
  count(host_name, sort = TRUE)
# A tibble: 1,334 × 2
   host_name      n
      <chr>  <int>
1      kara    138
2  seamless     79
3      mike     71
4  flatbook     58
5    alicia     50
6     marie     42
7     jason     35
8     sarah     26
```

FILTERING

```
airbnb %>%
  select(name) %>%
  mutate(charming = str_detect(name, regex("charming", ignore_case = TRUE)))
# A tibble: 3,585 × 2
                                         name charming
                                        <chr>    <lgl>
1                   Sunny Bungalow in the City    FALSE
2                 Charming room in pet friendly apt    TRUE
3                 Mexican Folk Art Haven in Boston    FALSE
4   Spacious Sunny Bedroom Suite in Historic Home    FALSE
5                        Come Home to Boston    FALSE
6               Private Bedroom + Great Coffee    FALSE
7             New Lrg Studio apt 15 min to Boston    FALSE
8               "Tranquility" on "Top of the Hill"    FALSE
9              6 miles away from downtown Boston!    FALSE
10             Perfect & Practical Boston Rental    FALSE
# ... with 3,575 more rows
```

- We can use str_detect to see if the word "charming" exists in the name
- Since str_detect supplies a logical response we can use this for filtering…

FILTERING

```
airbnb %>%
  select(name) %>%
  filter(str_detect(name, regex("charming", ignore_case = TRUE)))
# A tibble: 92 x 1
   name
   <chr>
 1 Charming room in pet friendly apt
 2 Cozy room in a charming villa.
 3 Charming Gambrel on a sweet street
 4 Charming 3 bedroom-15 min to Boston
 5 Charming new house-15 min to Boston
 6 Queen room in a charming villa
 7 Charming sunlit house in Boston
 8 Charming Victorian near T
 9 Charming 2BD Across from Arboretum
10 Charming Boston Apartment
# ... with 82 more rows
```

- We can use str_detect to see if the word "charming" exists in the name

- Since str_detect supplies a logical response we can use this for filtering…

FILTERING

- We can use different approaches to get to the same results

- What do you expect these to return????

```
airbnb %>%
  select(name) %>%
  filter(str_detect(name, "(C|c)harming|(C|c)ute"))
```

```
airbnb %>%
  select(name) %>%
  mutate(name = str_to_lower(name)) %>%
  filter(str_detect(name, "charming|cute"))
```

FILTERING

- We can use different approaches to get to the same results

```
airbnb %>%
  select(name) %>%
  filter(str_detect(name, "(C|c)harming|(C|c)ute"))
# A tibble: 105 x 1
   name
   <chr>
 1 Charming room in pet friendly apt
 2 Cozy room in a charming villa.
 3 Charming Gambrel on a sweet street
 4 Charming 3 bedroom-15 min to Boston
 5 Charming new house-15 min to Boston
 6 Queen room in a charming villa
 7 Charming sunlit house in Boston
 8 Charming Victorian near T
 9 Charming 2BD Across from Arboretum
```

```
airbnb %>%
  select(name) %>%
  mutate(name = str_to_lower(name)) %>%
  filter(str_detect(name, "charming|cute"))
# A tibble: 105 x 1
   name
   <chr>
 1 Charming room in pet friendly apt
 2 Cozy room in a charming villa.
 3 Charming Gambrel on a sweet street
 4 Charming 3 bedroom-15 min to Boston
 5 Charming new house-15 min to Boston
 6 Queen room in a charming villa
 7 Charming sunlit house in Boston
 8 Charming Victorian near T
```

YOUR TURN!

1. *Using the* `house_rules` *column, how many observations (aka hosts) advocate for "no shoes"?*

2. *How would you filter out these observations?*

SOLUTION

```
# problem 1
airbnb %>%
  tally(str_detect(house_rules, regex("no shoes", ignore_case = TRUE)))
# A tibble: 1 × 1
  no_shoes
     <int>
1       67


# problem 2
airbnb %>%
  filter(!str_detect(house_rules, regex("no shoes", ignore_case = TRUE)))
# A tibble: 2,326 × 95
        id                             listing_url    scrape_id last_scraped
     <int>                                   <chr>        <dbl>       <date>
1 12147973 https://www.airbnb.com/rooms/12147973 2.016091e+13   2016-09-07
2  3075044  https://www.airbnb.com/rooms/3075044 2.016091e+13   2016-09-07
3     6976      https://www.airbnb.com/rooms/6976 2.016091e+13   2016-09-07
```

CLEANING

```
airbnb %>%
  select(name) %>%
  count(str_extract(name, "^[^A-Za-z0-9]+"), sort = TRUE)
# A tibble: 20 x 2
   `str_extract(name, "^[^A-Za-z0-9]+")`      n
   <chr>                                  <int>
 1 NA                                      3468
 2 [                                         79
 3 $                                          8
 4 "\""                                       4
 5 (                                          3
 6 #                                          3
 7 *                                          2
 8 **                                         2
 9 ^                                          2
10 【                                         1
11 "* "                                       1
12 "*** "                                     1
```

- Sometimes we need to do some cleaning. For example, if we wanted to look for the most common first words used in names, we may want to clean up non-alphanumeric characters.
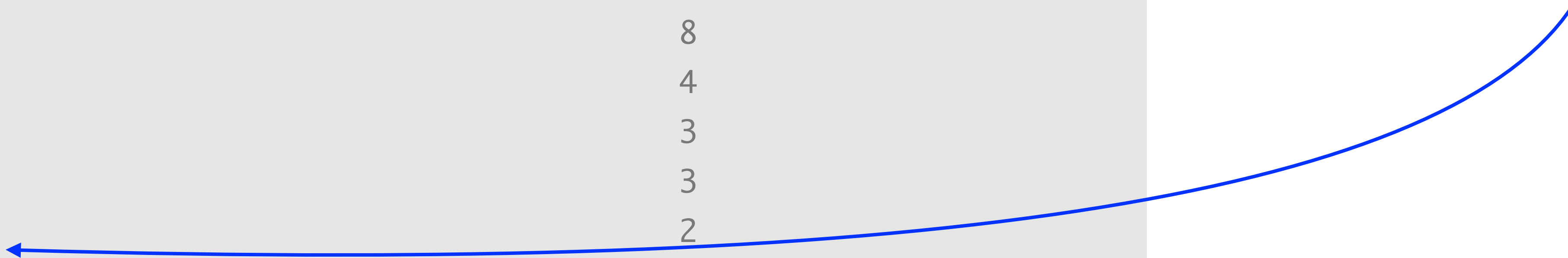
CLEANING

```
airbnb %>%
  select(name) %>%
  mutate(
    name = str_replace_all(name, "[^A-Za-z0-9]+", " "),
    name = str_replace_all(name, "[[:punct:]]+", " "),
    name = str_trim(name),
    name = str_to_lower(name)
  ) %>%
  count(str_extract(name, "^[A-Za-z0-9]+"), sort = TRUE)
# A tibble: 626 x 2
  `str_extract(name, "^[A-Za-z0-9]+")`      n
  <chr>                                 <int>
1 cozy                                    183
2 private                                 169
3 beautiful                               120
4 spacious                                114
5 lux                                     104
```

- Remove all non-alphanumerics
- Remove punctuations
- Remove extra white spaces
- Standardize to lowercase
- Extract and count first alphanumeric words

CHALLENGE

CHALLENGE

In the Kaggle competition for predicting Titanic survivors, the most important predictor variable ended up being the passenger's title (i.e. Mr., Mrs., Miss., Master).

Using the `titanic::titanic_train` *data, extract the passengers title and create a new feature named "Title".*

SOLUTION

```
titanic <- titanic::titanic_train %>% as_tibble()

titanic %>%
  mutate(Title = str_replace_all(Name, "(.*, )|(\\..*)", ""))
```
```
# A tibble: 891 x 13
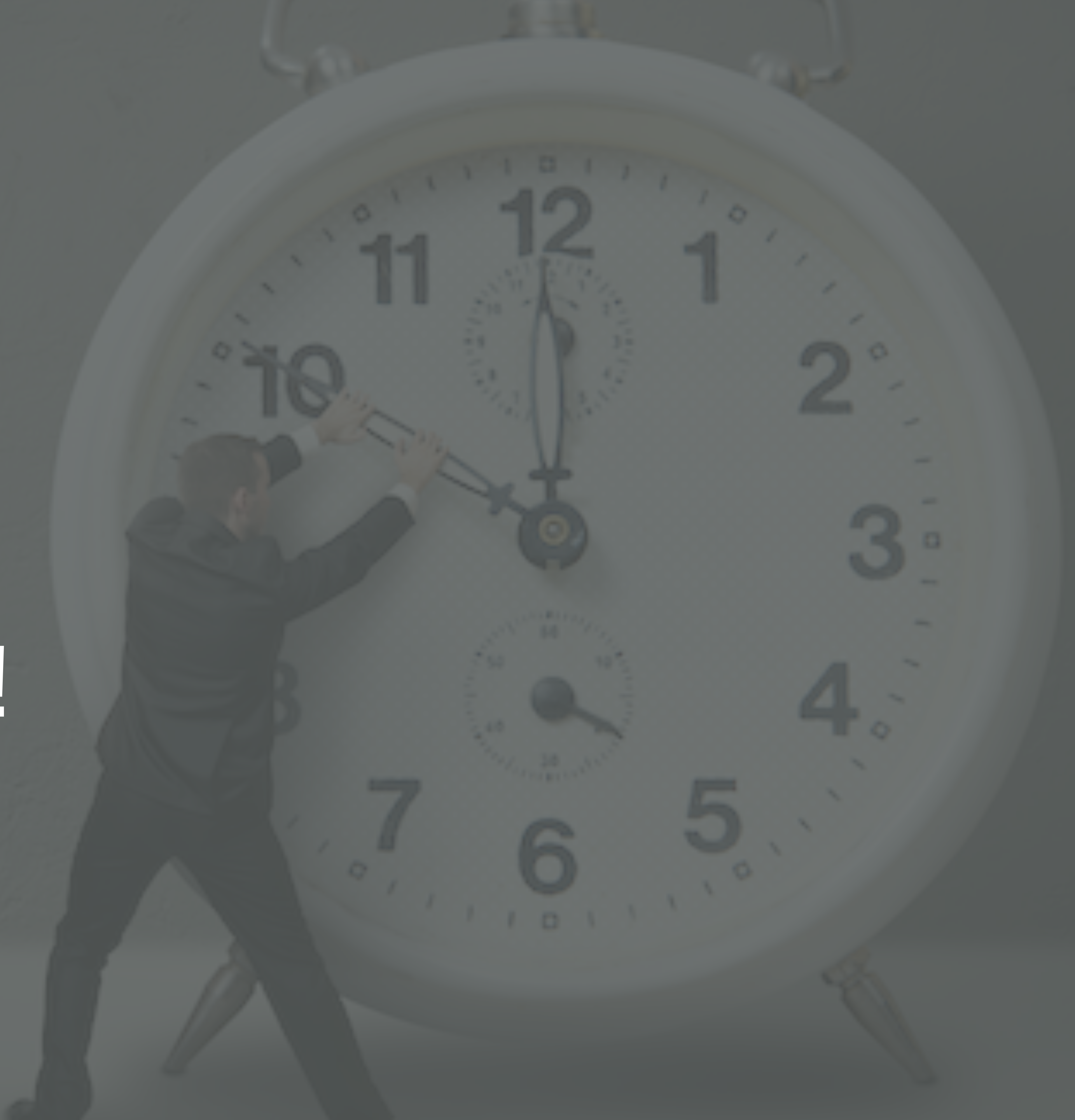   PassengerId Survived Pclass Name                          Sex      Age SibSp Parch Ticket        Fare Cabin Embarked Title
         <int>    <int>  <int> <chr>                         <chr>  <dbl> <int> <int> <chr>        <dbl> <chr> <chr>    <chr>
 1           1        0      3 Braund, Mr. Owen Harris       male    22.0     1     0 A/5 21171     7.25 ""    S        Mr
 2           2        1      1 Cumings, Mrs. John Bradley (Floren… female 38.0     1     0 PC 17599     71.3  C85   C        Mrs
 3           3        1      3 Heikkinen, Miss. Laina        female 26.0     0     0 STON/O2. 31…  7.92 ""    S        Miss
 4           4        1      1 Futrelle, Mrs. Jacques Heath (Lily… female 35.0     1     0 113803       53.1  C123  S        Mrs
 5           5        0      3 Allen, Mr. William Henry      male    35.0     0     0 373450        8.05 ""    S        Mr
 6           6        0      3 Moran, Mr. James              male      NA     0     0 330877        8.46 ""    Q        Mr
 7           7        0      1 McCarthy, Mr. Timothy J       male    54.0     0     0 17463        51.9  E46   S        Mr
 8           8        0      3 Palsson, Master. Gosta Leonard male    2.00     3     1 349909       21.1  ""    S        Mast…
 9           9        1      3 Johnson, Mrs. Oscar W (Elisabeth V… female 27.0     0     2 347742       11.1  ""    S        Mrs
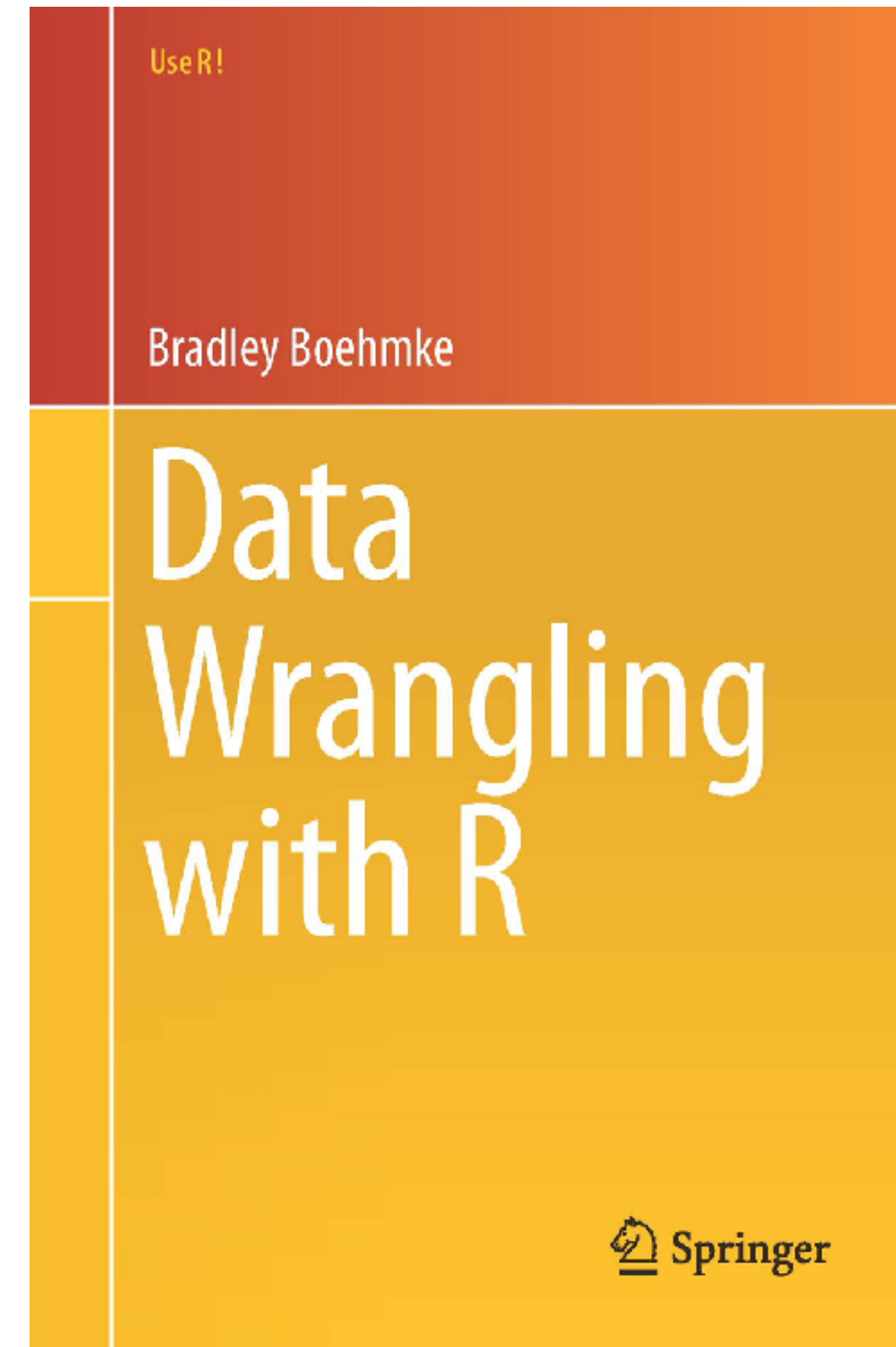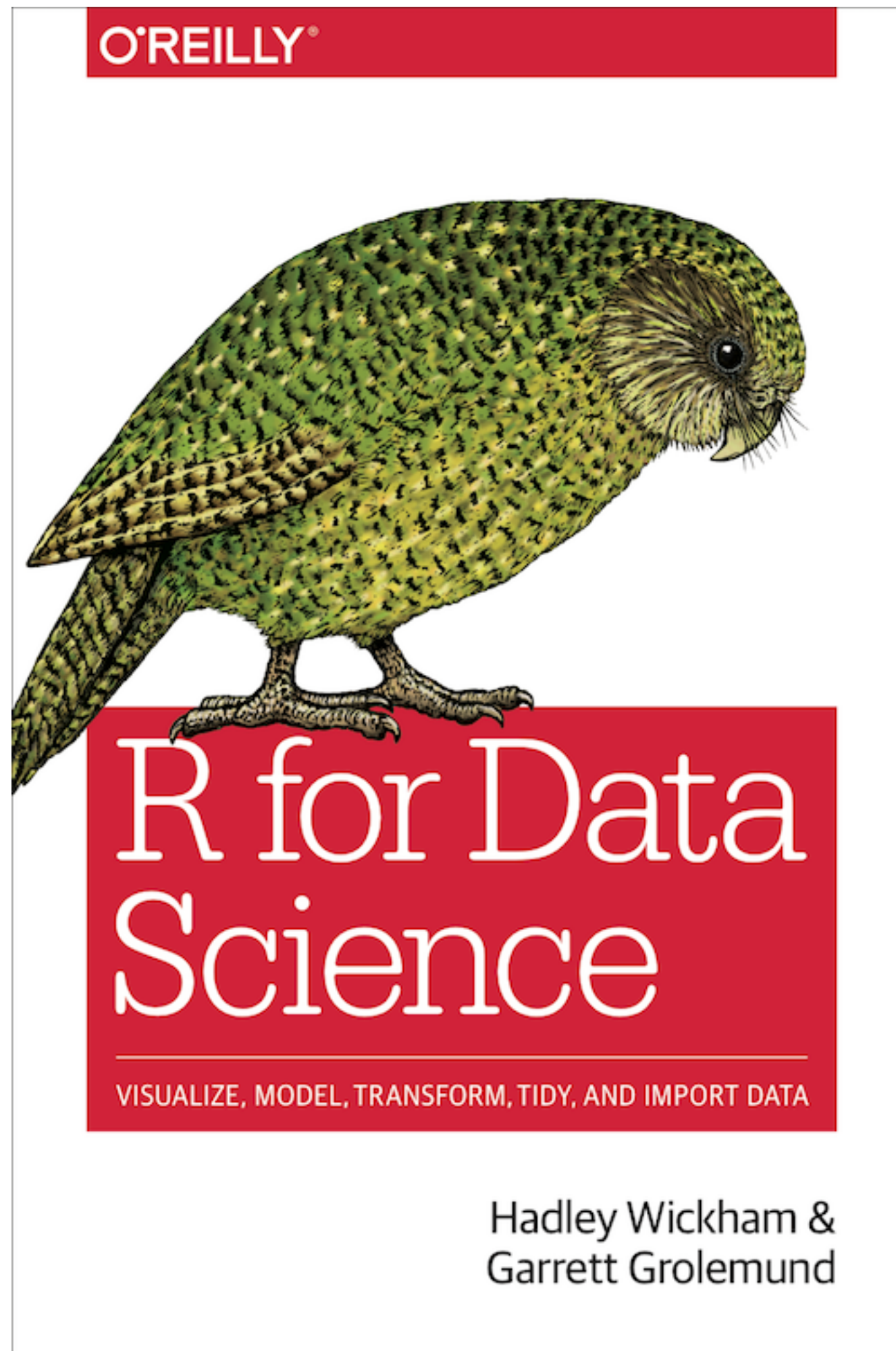10          10        1      2 Nasser, Mrs. Nicholas (Adele Achem) female 14.0     1     0 237736       30.1  ""    C        Mrs
# ... with 881 more rows
```

SO LITTLE TIME!

LEARN MORE

O'REILLY®

R for Data Science

VISUALIZE, MODEL, TRANSFORM, TIDY, AND IMPORT DATA

Hadley Wickham &
Garrett Grolemund

Use R!

Bradley Boehmke

Data Wrangling with R

Springer

WHAT TO REMEMBER

FUNCTIONS TO REMEMBER

Operator/Function	Description
`str_*`	stringr functions for regular expressions
`regex(pattern, ignore_case = TRUE)`	ignore case
"x	y" "(x|y)"	using or for finding multiple forms of regular expressions
^ $	anchors - finding regex at beginning or end of element
`., \\d, \\s, [0-9], [a-zA-Z]`	finding regex patterns
`+, {n}, {n,}, {n,m}`	finding repetitions of regex patterns