Strings and regular expressions

Week 5

AEM 2850 / 5850 : R for Business Analytics

Cornell Dyson

Fall 2025

Acknowledgements: R4DS (2e)

Announcements

Prelim 1 will be next Thursday, October 2 at 7:30pm in Warren 173 and 175

- Will cover everything so far **including this week's material**
- Tuesday, Sep 30: review session in class
- Thursday, Oct 2: no class
- Extra office hours TBD
- We will provide more guidance via canvas
- Please contact me and SDS **as soon as possible** if you are eligible for testing accommodations and are not yet registered

Questions before we get started?

Plan for this week

Tuesday

Working with strings in R

Regular expressions

Working with regular expressions in R

Thursday

example-05

Working with strings in R

Strings are nothing new

```
flights |>
  select(carrier, tailnum, origin, dest)
```

```
## # A tibble: 336,776 × 4
##    carrier tailnum origin dest
##    <chr>   <chr>   <chr>  <chr>
##  1 UA      N14228  EWR    IAH
##  2 UA      N24211  LGA    IAH
##  3 AA      N619AA  JFK    MIA
##  4 B6      N804JB  JFK    BQN
##  5 DL      N668DN  LGA    ATL
##  6 UA      N39463  EWR    ORD
##  7 B6      N516JB  EWR    FLL
##  8 EV      N829AS  LGA    IAD
##  9 B6      N593JB  JFK    MCO
## 10 AA      N3ALAA  LGA    ORD
## # ℹ 336,766 more rows
```

```
read_csv("homework-1-survey.csv") |>
  select(Concentration)
```

```
## # A tibble: 113 × 1
##    Concentration
##    <chr>
##  1 Finance
##  2 Business Analytics and Accounting
##  3 Business Analytics
##  4 Entrepreneurship
##  5 Entrepreneurship
##  6 Finance
##  7 Strategy
##  8 Business Analytics
##  9 Business Analytics
## 10 Human Resource Management
## # ℹ 103 more rows
```

Strings in R

Strings are also referred to as "characters" (abbreviated `chr`)

Strings can be stored in many ways:

- Vectors
- **Data frame columns**
- Elements in a list

So far we have used them as we would any other data

But now we'll learn to filter on, modify, or analyze "functions" of strings

The stringr package

`stringr` is loaded as part of the core tidyverse

All `stringr` functions have intuitive names that start with `str_`

We will cover a bunch of handy functions this week:

1. `str_length`
2. `str_to_upper` and `str_to_lower`
3. `str_c` and `str_glue`
4. `str_detect`
5. `str_count`
6. `str_replace`

See `vignette("stringr")` for more

We'll use data from The Office

The schrute package contains transcripts of all episodes of The Office (US)

```
library(schrute)
theoffice # this data frame is an object from the schrute package
```

```
## # A tibble: 55,130 × 12
##    index season episode episode_name director   writer           character text
##    <int>  <int>   <int> <chr>        <chr>      <chr>            <chr>     <chr>
##  1     1      1       1 Pilot        Ken Kwapis Ricky Gervais;S… Michael   All …
##  2     2      1       1 Pilot        Ken Kwapis Ricky Gervais;S… Jim       Oh, …
##  3     3      1       1 Pilot        Ken Kwapis Ricky Gervais;S… Michael   So y…
##  4     4      1       1 Pilot        Ken Kwapis Ricky Gervais;S… Jim       Actu…
##  5     5      1       1 Pilot        Ken Kwapis Ricky Gervais;S… Michael   All …
##  6     6      1       1 Pilot        Ken Kwapis Ricky Gervais;S… Michael   Yes,…
##  7     7      1       1 Pilot        Ken Kwapis Ricky Gervais;S… Michael   I've…
##  8     8      1       1 Pilot        Ken Kwapis Ricky Gervais;S… Pam       Well…
##  9     9      1       1 Pilot        Ken Kwapis Ricky Gervais;S… Michael   If y…
## 10    10      1       1 Pilot        Ken Kwapis Ricky Gervais;S… Pam       What?
## # ℹ 55,120 more rows
## # ℹ 4 more variables: text_w_direction <chr>, imdb_rating <dbl>,
## #   total_votes <int>, air_date <chr>
```

1) str_length()

`str_length` tells you the number of characters in a string

```
str_length("supercalifragilisticexpialidocious")
```

```
## [1] 34
```

```
theoffice |>
  distinct(character) |>
  slice_head(n = 5) |>
  mutate(name_length = str_length(character))
```

```
## # A tibble: 5 × 2
##   character name_length
##   <chr>           <int>
## 1 Michael             7
## 2 Jim                 3
## 3 Pam                 3
## 4 Dwight              6
## 5 Jan                 3
```

2) str_to_lower() and str_to_upper()

`str_to_lower` converts to lower case

```
str_to_lower("I went to Cornell, you ever heard of it?")
```

```
## [1] "i went to cornell, you ever heard of it?"
```

`str_to_upper` converts to upper case

```
str_to_upper("I went to Cornell, you ever heard of it?")
```

```
## [1] "I WENT TO CORNELL, YOU EVER HEARD OF IT?"
```

Similar functionality for `str_to_title()` and `str_to_sentence()`

These functions are locale dependent (e.g., "en_GB" vs "en_US")

3) str_c()

We have seen `c` combine arguments to make a **longer** vector or list:

```
abc <- c("a", "b", "c")
c(abc, "s")
```

```
## [1] "a" "b" "c" "s"
```

```
c(abc, abc)
```

```
## [1] "a" "b" "c" "a" "b" "c"
```

```
c(abc, NA)
```

```
## [1] "a" "b" "c" NA
```

By contrast, `str_c` combines arguments to make a **wider** character vector:

```
str_c(abc, "s")
```

```
## [1] "as" "bs" "cs"
```

```
str_c(abc, abc)
```

```
## [1] "aa" "bb" "cc"
```

```
str_c(abc, NA)
```

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

We can use `str_c` within data frames to combine strings at scale

3) str_c()

Here's an example of multiple columns in a data frame being combined into one:

```
theoffice |> slice_head(n = 1) |>
  select(character, text)
```

```
## # A tibble: 1 × 2
##   character text
##   <chr>     <chr>
## 1 Michael   All right Jim. Your quarterlies look very good. How are things at t…
```

```
theoffice |> slice_head(n = 1) |>
    mutate(line = str_c(character, " said: ", text))
```

```
## # A tibble: 1 × 1
##   line
##   <chr>
## 1 Michael said: All right Jim. Your quarterlies look very good. How are things …
```

3) str_c()

`str_c` will automatically recycle fixed arguments like `" said: "` that are shorter than `character` and `text`:

```
theoffice |> slice_head(n = 3) |>
    mutate(line = str_c(character, " said: ", text))
```

```
## # A tibble: 3 × 1
##   line
##   <chr>
## 1 Michael said: All right Jim. Your quarterlies look very good. How are things …
## 2 Jim said: Oh, I told you. I couldn't close it. So...
## 3 Michael said: So you've come to the master for guidance? Is this what you're …
```

3) str_glue()

`str_glue()` provides similar functionality, but different syntax:

```
theoffice |> slice_head(n = 3) |>
  mutate(line = str_glue("{character} said: {text}")) # note the different syntax
```

```
## # A tibble: 3 × 1
##   line
##   <glue>
## 1 Michael said: All right Jim. Your quarterlies look very good. How are things …
## 2 Jim said: Oh, I told you. I couldn't close it. So...
## 3 Michael said: So you've come to the master for guidance? Is this what you're …
```

Items inside `{}` are evaluated as if they are outside the quotes

This can be handy when combining many fixed and variable strings

`str_c()` and `str_glue()` work well with `mutate()` because their output is the same length as their inputs

Regular expressions

Regular expressions

What are regular expressions?

A concise, powerful way for describing patterns within strings

Regular expressions are a generic tool, not something specific to R

Let's use the names of some characters from The Office as examples:

```
names <- theoffice |> distinct(character) |> slice_head(n = 10) |> pull(character)
names
```

```
##  [1] "Michael"     "Jim"         "Pam"         "Dwight"      "Jan"
##  [6] "Michel"      "Todd Packer" "Phyllis"     "Stanley"     "Oscar"
```

Pattern basics

The simplest patterns consist of literal characters

```
names
```

```
##  [1] "Michael"     "Jim"         "Pam"         "Dwight"      "Jan"
##  [6] "Michel"      "Todd Packer" "Phyllis"     "Stanley"     "Oscar"
```

`str_view` is a handy classroom utility for class to see how patterns match:

```
str_view(names, pattern = "J")
```

```
## [2] │ <J>im
## [5] │ <J>an
```

Pattern basics

Literal pattern matches are case-sensitive by default

```
names
```

```
##  [1] "Michael"     "Jim"         "Pam"         "Dwight"      "Jan"
##  [6] "Michel"      "Todd Packer" "Phyllis"     "Stanley"     "Oscar"
```

What do you think this will return?

```
str_view(names, pattern = "M")
```

```
## [1] │ <M>ichael
## [6] │ <M>ichel
```

What do you think this will return?

```
str_view(names, pattern = "m")
```

```
## [2] │ Ji<m>
## [3] │ Pa<m>
```

Note that these matches depend on **patterns**, not position

Meta-characters

Punctuation characters like `.`, `+`, `*`, `[`, `]`, and `?` are **meta-characters** with special meanings

The most common one is `.`, which will match any character

What do you think these statements will return?

```
str_view(names, pattern = "J.m")
```

```
## [2] │ <Jim>
```

```
str_view(names, pattern = "J.n")
```

```
## [5] │ <Jan>
```

Meta-characters

What do you think these statements will return?

```
str_view(names, pattern = "J..")
```

```
## [2] │ <Jim>
## [5] │ <Jan>
```

```
str_view(names, pattern = "J...")
```

Quantifiers

Quantifiers control how many times a pattern can match:

- `?` makes a pattern optional -- it matches 0 or 1 times
- `+` lets a pattern repeat -- it matches at least once
- `*` lets a pattern be optional or repeat

What do you think this statement will return?

```
str_view(names, "M.*l") # match strings with M, then any number of any characters, then l
```

```
## [1] │ <Michael>
## [6] │ <Michel>
```

Note that quantifiers modify the pattern they follow: we used `.*`, not `*.`

Character classes

`[]` lets you match a set of characters

```
str_view(names, "[aeiou]") # vowels
```

```
##  [1] │ M<i>ch<a><e>l
##  [2] │ J<i>m
##  [3] │ P<a>m
##  [4] │ Dw<i>ght
##  [5] │ J<a>n
##  [6] │ M<i>ch<e>l
##  [7] │ T<o>dd P<a>ck<e>r
##  [8] │ Phyll<i>s
##  [9] │ St<a>nl<e>y
## [10] │ Osc<a>r
```

`^` inverts character class matches

```
str_view(names, "[^aeiou]") # NOT vowels
```

```
##  [1] │ <M>i<c><h>ae<l>
##  [2] │ <J>i<m>
##  [3] │ <P>a<m>
##  [4] │ <D><w>i<g><h><t>
##  [5] │ <J>a<n>
##  [6] │ <M>i<c><h>e<l>
##  [7] │ <T>o<d><d>< ><P>a<c><k>e<r>
##  [8] │ <P><h><y><l><l>i<s>
##  [9] │ <S><t>a<n><l>e<y>
## [10] │ <O><s><c>a<r>
```

Alternation

Last one! Hang in there!

Alternation, `|`, allows you to search for one or more alternative patterns

This should seem familiar...

What do you think these statements will return?

```
str_view(names, "J.m|P.m")
```

```
## [2] │ <Jim>
## [3] │ <Pam>
```

```
str_view(names, "S.*|O.*")
```

```
##  [9] │ <Stanley>
## [10] │ <Oscar>
```

More patterns

See Chapter 15 of R4DS (2e) for more on:

- **escaping**: matching meta-characters as if they were literal strings
- **anchors**: match the start or end of a strong
- **character classes**: (continued)
- **quantifiers** (continued)
- **operator precedence**: parentheses, etc.
- **grouping**: back references, etc.

Working with regular expressions in R

4) str_detect()

`str_detect` can be used to match patterns and return a logical vector

```
first_4_characters
```

```
## [1] "Michael" "Jim"     "Pam"     "Dwight"
```

```
str_detect(first_4_characters, "Dwight")
```

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

What do you think these statements will return?

```
str_detect(first_4_characters, "a")
```

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

```
str_detect(first_4_characters, "[aeiou]")
```

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

How could we fit this into our current workflow?

4) str_detect()

`str_detect` is a powerful way to `filter` a data frame

```
theoffice |> select(season, episode, character, text) |>
  filter(str_detect(text,    # where to match a pattern: the column named "text"
                    "sale")) # what pattern to match: the pattern "sale"
```

```
## # A tibble: 370 × 4
##    season episode character text
##     <int>   <int> <chr>     <chr>
##  1      1       2 Jim       This is my biggest sale of the year. They love me o…
##  2      1       2 Jim       Mr. Decker, we didn't lose your sale today, did we?…
##  3      1       3 Jim       That is a great offer. Thank you. I really think I …
##  4      1       3 Jan       From sales?
##  5      1       4 Michael   Look, look, look. I talked to corporate, about prot…
##  6      1       5 Michael   All right, time, time out. Come on, sales, over her…
##  7      1       6 Jan       Alan and I have created an incentive program to inc…
##  8      1       6 Jan       We've created an incentive program to increase sale…
##  9      1       6 Jim       Plus you have so much more to talk to this girl abo…
## 10      1       6 Stanley   I thought that was the incentive prize for the top …
## # ℹ 360 more rows
```

4) str_detect()

Literal pattern matches with `str_detect` are case-sensitive

```
theoffice |> select(season, episode, character, text) |>
  filter(str_detect(text,
                    "Sale")) # sale and Sale produce different output
```

```
## # A tibble: 28 × 4
##    season episode character          text
##     <int>   <int> <chr>              <chr>
##  1      2      11 Michael            No, no. Salesmen and profit centers.
##  2      2      14 Michael            Old fashioned raid. Sales on Accounting. Y…
##  3      2      14 Michael and Dwight Ahhhh! Whoo hoo! Come on, come on, come on…
##  4      2      14 Michael            Oh, and I'm not? Why would you say that? B…
##  5      2      17 Jim                Dwight was the top salesman of the year at…
##  6      2      17 Michael            Speaker at the Sales Convention. Been ther…
##  7      2      17 Dwight             Saleswoman has a v*g1n*.
##  8      2      17 Speaker            Next, I'd like to introduce the Dunder Mif…
##  9      2      17 Dwight             Salesman of Northeastern Pennsylvania, I a…
## 10      3       5 Angela             Sales take a long time.
## # ℹ 18 more rows
```

4) str_detect()

You could use multiple calls to `str_detect`, or use alternation:

```
theoffice |> select(season, episode, character, text) |>
  filter(str_detect(text,
                    "sale|Sale")) # look for sale OR Sale
```

```
## # A tibble: 392 × 4
##    season episode character text
##     <int>   <int> <chr>     <chr>
##  1      1       2 Jim       This is my biggest sale of the year. They love me o…
##  2      1       2 Jim       Mr. Decker, we didn't lose your sale today, did we?…
##  3      1       3 Jim       That is a great offer. Thank you. I really think I …
##  4      1       3 Jan       From sales?
##  5      1       4 Michael   Look, look, look. I talked to corporate, about prot…
##  6      1       5 Michael   All right, time, time out. Come on, sales, over her…
##  7      1       6 Jan       Alan and I have created an incentive program to inc…
##  8      1       6 Jan       We've created an incentive program to increase sale…
##  9      1       6 Jim       Plus you have so much more to talk to this girl abo…
## 10      1       6 Stanley   I thought that was the incentive prize for the top …
## # ℹ 382 more rows
```

4) str_detect()

You could consolidate this: regex parentheses are like in math

```
theoffice |> select(season, episode, character, text) |>
  filter(str_detect(text,
                    "(s|S)ale")) # look for sale OR Sale
```

```
## # A tibble: 392 × 4
##    season episode character text
##     <int>   <int> <chr>     <chr>
##  1      1       2 Jim       This is my biggest sale of the year. They love me o…
##  2      1       2 Jim       Mr. Decker, we didn't lose your sale today, did we?…
##  3      1       3 Jim       That is a great offer. Thank you. I really think I …
##  4      1       3 Jan       From sales?
##  5      1       4 Michael   Look, look, look. I talked to corporate, about prot…
##  6      1       5 Michael   All right, time, time out. Come on, sales, over her…
##  7      1       6 Jan       Alan and I have created an incentive program to inc…
##  8      1       6 Jan       We've created an incentive program to increase sale…
##  9      1       6 Jim       Plus you have so much more to talk to this girl abo…
## 10      1       6 Stanley   I thought that was the incentive prize for the top …
## # ℹ 382 more rows
```

4) str_detect()

Or use `regex` to ignore all cases and control other pattern matching details

```
theoffice |> select(season, episode, character, text) |>
  filter(str_detect(text,
                    regex("sale", ignore_case = TRUE))) # ignore all cases
```

```
## # A tibble: 393 × 4
##    season episode character text
##     <int>   <int> <chr>     <chr>
##  1      1       2 Jim       This is my biggest sale of the year. They love me o…
##  2      1       2 Jim       Mr. Decker, we didn't lose your sale today, did we?…
##  3      1       3 Jim       That is a great offer. Thank you. I really think I …
##  4      1       3 Jan       From sales?
##  5      1       4 Michael   Look, look, look. I talked to corporate, about prot…
##  6      1       5 Michael   All right, time, time out. Come on, sales, over her…
##  7      1       6 Jan       Alan and I have created an incentive program to inc…
##  8      1       6 Jan       We've created an incentive program to increase sale…
##  9      1       6 Jim       Plus you have so much more to talk to this girl abo…
## 10      1       6 Stanley   I thought that was the incentive prize for the top …
## # ℹ 383 more rows
```

4) str_detect()

When I say ignore all cases, I mean IGNORE ALL CASES!

```
theoffice |> select(season, episode, character, text) |>
  filter(str_detect(text,
                    regex("sale", ignore_case = TRUE))) |>
  filter(!str_detect(text, "(s|S)ale")) # find non-standard form(s)
```

```
## # A tibble: 1 × 4
##   season episode character text
##    <int>   <int> <chr>     <chr>
## 1      3       3 Dwight    I HAVE EXCELLENT SALES NUMBERS!
```

4) str_detect()

`str_detect` can be combined with familiar functions to summarize data

```
theoffice |>
  filter(str_detect(text, regex("sale", ignore_case = TRUE))) |>
  count(character, sort = TRUE)
```

```
## # A tibble: 46 × 2
##    character     n
##    <chr>     <int>
##  1 Michael      91
##  2 Dwight       81
##  3 Jim          51
##  4 Andy         31
##  5 Pam          26
##  6 Ryan         10
##  7 Clark         8
##  8 Gabe          7
##  9 David         6
## 10 Angela        5
## # ℹ 36 more rows
```

4) str_detect()

`str_detect` can be combined with familiar functions to summarize data

```
theoffice |>
  filter(str_detect(text,
                    regex("that's what she said", ignore_case = TRUE))) |>
  count(character, sort = TRUE)
```

```
## # A tibble: 8 × 2
##   character     n
##   <chr>     <int>
## 1 Michael      23
## 2 Dwight        3
## 3 Jim           2
## 4 Creed         1
## 5 David         1
## 6 Holly         1
## 7 Jan           1
## 8 Pam           1
```

4) str_detect()

`str_detect` with regular expressions can be very powerful

```
theoffice |> select(character, text) |>
  filter(str_detect(text, "assistant.*manager")) |>
  slice_head(n = 10)
```

```
## # A tibble: 10 × 2
##    character text
##    <chr>     <chr>
##  1 Dwight    I, but if there were, I'd be protected as assistant regional manag…
##  2 Dwight    And that's why you have an assistant regional manager.
##  3 Michael   No, I am the team manager. You can be assistant to the team manage…
##  4 Dwight    Hey, Pam, I'm assistant regional manager, and I can take care of h…
##  5 Michael   All right. Well then, you are now acting manager of Dunder Mifflin…
##  6 Dwight    Uh,... my first sale, my promotion to assistant regional manager, …
##  7 Jim       Oh, that's because at first it was a made up position for Dwight, …
##  8 Charles   So you're the assistant to the regional manager?
##  9 Darryl    Since Andy promoted me to assistant regional manager, I've been tr…
## 10 Andy      You now, Darryl, this is textbook assistant regional manager stuff…
```

5) str_count()

`str_count` can be used to count the number of matches in a string

```
theoffice |>
  distinct(character) |>
  slice_head(n = 5) |>
  mutate(
    name = str_to_lower(character), # another way to avoid case sensitivity
    m_s = str_count(name, "m"),
    i_s = str_count(name, "i")
  )
```

```
## # A tibble: 5 × 4
##   character name      m_s   i_s
##   <chr>     <chr>   <int> <int>
## 1 Michael   michael     1     1
## 2 Jim       jim         1     1
## 3 Pam       pam         1     0
## 4 Dwight    dwight      0     1
## 5 Jan       jan         0     0
```

5) str_count() with regex

```
theoffice |>
  distinct(character) |>
  slice_head(n = 5) |>
  mutate(
    name = str_to_lower(character),
    vowels = str_count(name, "[aeiou]"), # count matches of ANY of these characters
    consonants = str_count(name, "[^aeiou]") # count matches of everything EXCEPT these characters
  )
```

```
## # A tibble: 5 × 4
##   character name    vowels consonants
##   <chr>     <chr>    <int>      <int>
## 1 Michael   michael      3          4
## 2 Jim       jim          1          2
## 3 Pam       pam          1          2
## 4 Dwight    dwight       1          5
## 5 Jan       jan          1          2
```

Reminder: `[]` lets you match a set of characters; `^` inverts character class matches

6) str_replace()

As the name suggests, `str_replace` can be used to modify patterns in strings

```
names
```

```
##  [1] "Michael"     "Jim"         "Pam"         "Dwight"      "Jan"
##  [6] "Michel"      "Todd Packer" "Phyllis"     "Stanley"     "Oscar"
```

```
str_replace(names, "Dw", "Duhw") # jim's office pronunciation guide
```

```
##  [1] "Michael"     "Jim"         "Pam"         "Duhwight"    "Jan"
##  [6] "Michel"      "Todd Packer" "Phyllis"     "Stanley"     "Oscar"
```

6) str_replace()

`str_replace` replaces the first match of a pattern

```
str_replace("Phyllis", "l", "!")
```

```
## [1] "Phy!lis"
```

`str_replace_all` replaces all matches of a pattern

```
str_replace_all("Phyllis", "l", "!")
```

```
## [1] "Phy!!is"
```

These pair naturally with `mutate` just like `str_c`, `str_glue`, and `str_count`

example-05