Week 3 Cohort 4: R4DS Book Club

Chapters 4 & 5

Workflow: basics; Data transformation

Collin K. Berke

Twitter: @BerkeCollin

Last updated: 2020-12-30

5-minute ice breaker

- What's the last TV show/movie you watched?

Quick housekeeping/reminders

- Video camera is optional, but encouraged.
- If we need to slow down and discuss, let me know.
    - Most likely someone has the same question.
- Take time to learn the theory.
- Please attempt the chapter exercises.
- Please plan on teaching one of the lessons.

Tonight's discussion

- Chapter 4: Workflow basics
- Chapter 5: Data transformations
    - `filter()` rows
    - `arrange()` data
    - `select()` columns by their names
    - `mutate()` new variables
    - `group_by()` and `summarise()` a new calculation
    - `%>%` to chain commands together

Workflow basics

- R can be used like a calculator.

```
1 / 200 * 30
```

```
## [1] 0.15
```

```
(59 + 73 + 2) / 3
```

```
## [1] 44.66667
```

```
sin(pi / 2)
```

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

Creating objects

- R is an object based programming language.

- Assign new objects with the `<-` (Alt + -), avoid using `=`

```
# Avoid
x = 3 * 4

# object_name <- value
# "Object name gets value"
x <- 3 * 4
```

- What would happen if we type `x` in the terminal?

```
x
#> [1] 12
```

- See all objects in the Environment pane

Naming rules

- Object names must start with a letter.

```
# Bad
2data

# Bad
_data

# Bad
%data

# Good
data
```

Naming rules (cont.)

- Only contain letters, numbers, `_`, and `.`.

```
# Bad
flights_dep_&_arr

# Good
flights.filtered.data

# Good, though not very descriptive
flights_data_2
```

Naming rules (cont.)

- Use descriptive names.

```
# Bad
flights_2_data

# Good
flights_filtered_data
```

Naming rules (cont.)

- Choose a case and style, stay consistent.
  - Think about others reading/using your code.
  - Think about your future self reading/using your code.

```
# Examples from book
i_use_snake_case
otherPeopleUseCamelCase
some.people.use.periods
And_aFew.People_RENOUNCEconvention
```

Naming rules (cont.)

- Typos matter
- Case matters

> "There’s an implied contract between you and R: it will do the tedious computation for you, but in return, you must be completely precise in your instructions. Typos matter. Case matters. ~Hadley Wickham & Garrett Grolemund (authors)"

```
y
#> Error: object 'y' not found
```

- You won't always get an error from a typo, so keep an eye out.

```
# Functions used for importing data, we will discuss later
read_csv() # from readr

# vs.
read.csv() # from utils
```

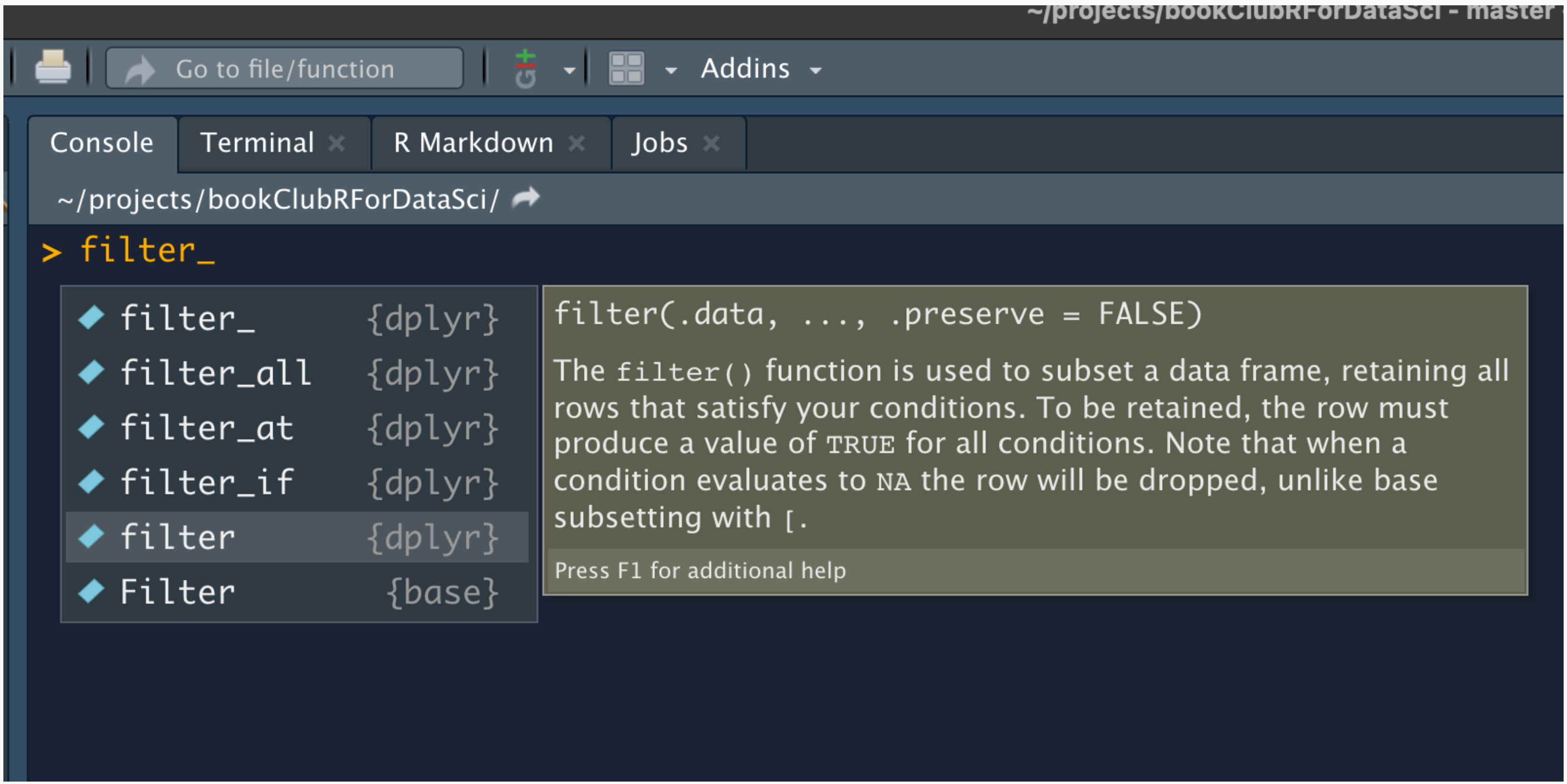
Naming rules, examples

tidyverse style guide

Google's R style guide

Calling functions

- Give your fingers a break, use the tab key.


~/projects/bookClubRForDataSci - master
Go to file/function
Addins
Console
Terminal
R Markdown
Jobs
~/projects/bookClubRForDataSci/

```
> filter_
```

```
filter_       {dplyr}
filter_all    {dplyr}
filter_at     {dplyr}
filter_if     {dplyr}
filter        {dplyr}
Filter         {base}
```

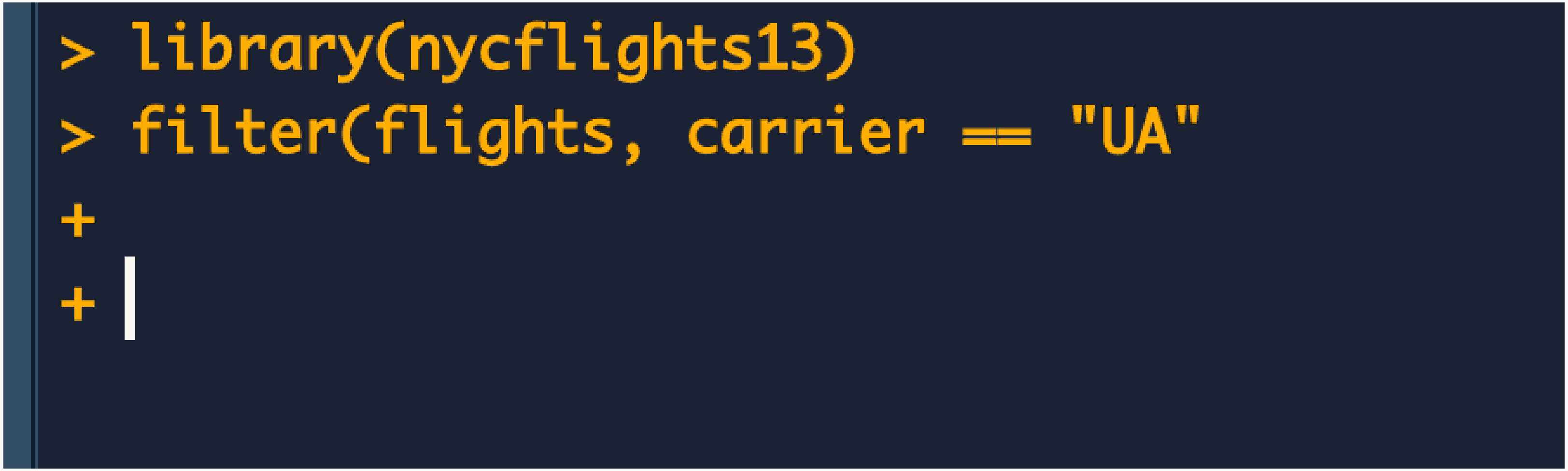
```
filter(.data, ..., .preserve = FALSE)
```

The `filter()` function is used to subset a data frame, retaining all rows that satisfy your conditions. To be retained, the row must produce a value of `TRUE` for all conditions. Note that when a condition evaluates to `NA` the row will be dropped, unlike base subsetting with `[`.

Press F1 for additional help

Calling functions (cont.)

- Match `()` and `""`.
  - If you see `+`, `R` is waiting for you to complete the expression.
  - Finish the expression or hit the `esc` key.



```
> library(nycflights13)
> filter(flights, carrier == "UA"
+
+ 
```

Chapter 5 - Data transformation

- **Note:** This chapter has lots of good information and many useful application examples. One session may not be enough to fully discuss each function and their many uses.

The verbs of data manipulation

Perform some action with our data

- `dplyr` functions:
  - `filter()`
  - `arrange()`
  - `select()`
  - `mutate()`
  - `group_by()` and `summarise()`

`dplyr` verbs never change our original data

- Get comfortable using the assignment operator, `←`.

The `nycflights13` data

```r
library(nycflights13)

# more info, enter ?flights into console

names(flights)
```

```
##  [1] "year"           "month"          "day"            "dep_time"
##  [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_ti
##  [9] "arr_delay"      "carrier"        "flight"         "tailnum"
## [13] "origin"         "dest"           "air_time"       "distance"
## [17] "hour"           "minute"         "time_hour"
```

The `nycflights13` data types

```r
# Quick view of the data types
glimpse(flights)
```

```
## Rows: 336,776
## Columns: 19
## $ year           <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, …
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …
## $ dep_time       <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558,…
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600,…
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, -…
## $ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849…
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851…
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, -…
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", …
## $ flight         <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301, …
## $ tailnum        <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N39…
## $ origin         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA"…
## $ dest           <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD"…
## $ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, …
## $ distance       <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733,…
## $ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, …
## $ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59, …
```

The `filter()` function

```
# data = data you want to filter
# filter exp = the expression used to filter data
filter(<data>, <filter exp>)
```

- Allows you to subset observations based on their values.
- Subset our data to return flights on Jan. 3, 2013

```
(jan1 <- filter(flights, month == 1, day == 1))
```

```
## # A tibble: 842 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
```

The `filter()` function (cont.)

Effective filtering requires:

- The use of comparison operators.
  - '>' greater than
  - '>=' greater than or equal to
  - '<' less than
  - '<=' less than or equal to
  - '!=' not equal
  - '==' equal

- The application of logical operators.
  - '&' is "and"
  - '|' is "or"
  - '!' is "not"

- The use of the `%in%` operator

The `filter()` function, examples

- Using the `flights` data (you give it a try):
    - How many flights were to the major airports in Chicago (i.e., ORD, MDW) in 2013?
    - *There are multiple ways to get the answer.*

```r
# One solution
filter(flights, dest == "ORD" | dest == "MDW")

# Another solution
filter(flights, dest %in% c("ORD", "MDW"))
```

The `arrange()` function

```
# data = data you want to use
# col = column name you want to arrange by
arrange(<data>, <col>, desc(<col>))
```

- `arrange()` - changes the order of the rows based on a variable(s).
- Orders rows in ascending (default) or descending (`desc()`) order.
- Multiple variables are used as tie-breakers.
- `NA` values are always sorted at the end.

The `arrange()` function, examples

- What will the result of this code be?

```r
arrange(flights, desc(dep_delay))
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     9      641            900      1301     1242           1530
##  2  2013     6    15     1432           1935      1137     1607           2120
##  3  2013     1    10     1121           1635      1126     1239           1810
##  4  2013     9    20     1139           1845      1014     1457           2210
##  5  2013     7    22      845           1600      1005     1044           1815
##  6  2013     4    10     1100           1900       960     1342           2211
##  7  2013     3    17     2321            810       911      135           1020
##  8  2013     6    27      959           1900       899     1236           2226
##  9  2013     7    22     2257            759       898      121           1026
## 10  2013    12     5      756           1700       896     1058           2020
## # … with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

The `arrange()` function, examples (cont.)

- What will be the result of this code be?

```
filter(flights, is.na(dep_time))
```

```
## # A tibble: 8,255 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1       NA           1630        NA       NA           1815
##  2  2013     1     1       NA           1935        NA       NA           2240
##  3  2013     1     1       NA           1500        NA       NA           1825
##  4  2013     1     1       NA            600        NA       NA            901
##  5  2013     1     2       NA           1540        NA       NA           1747
##  6  2013     1     2       NA           1620        NA       NA           1746
##  7  2013     1     2       NA           1355        NA       NA           1459
##  8  2013     1     2       NA           1420        NA       NA           1644
##  9  2013     1     2       NA           1321        NA       NA           1536
## 10  2013     1     2       NA           1545        NA       NA           1910
## # … with 8,245 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

The `select()` function

`select()` subsets the data with specific operations based on the names of the variables.

```
# data = data you want to use
# col = column(s) you want to include in your selection
select(<data>, <col>, <col>)
```

- Function is useful when you have datasets with many variables.

The `select()` function (cont.)

- `select()` can be combined with **helper functions** to select any combination of columns.

- `starts_with()` - matches names from the start.
- `ends_with()` - matches names from the end.
- `contains()` - matches names anywhere.

- `matches()` - match based on a regular expression.
- `num_range()` - matches based on a sequence of numbers.
- `everything()` - select all non-specified variables.

General applications of `select()`

```
# Select by name
select(flights, year, month, day)

# Select a range of columns (inclusive)
select(flights, year:day)

# Select columns excluding a range (inclusive)
select(flights, -(year:day))
```

Combining helper functions and `select()`

5.4.1.1. Brainstorm as many ways as possible to select `dep_time`, `dep_delay`, `arr_time`, and `arr_delay` from flights? (Anyone want to share?)

```r
select(flights, dep_time, dep_delay, arr_time, arr_delay)

select(flights, ends_with(c("time", "delay")))

select(flights, starts_with(c("dep", "arr")))

select(flights, matches(c("^dep", "^arr")))

# Works, but it is ambiguous
select(flights, 4, 6, 7, 9)

select(flights, 4, 6:7, 9)

vars <- c("dep_time", "dep_delay", "arr_time", "arr_delay")

select(flights, all_of(vars))
```

Renaming and the use of `select()`

- Renaming can be done using select.
- However, the book suggests using `rename()` instead.

```
# Using rename(), keeps all the columns
# data = data you want to use
# new name = the new name you want to apply to the column
# old name = the old name of the column you want to change
rename(<data>, <new name> = <old name>)

rename(flights, tail_num = tailnum)

# Using select to rename and retain the one column
select(flights, tail_num = tailnum)
```

The `mutate()` function

`mutate()` adds new columns that are functions of existing columns.

- Always adds new columns at the end of the dataset.
- Only want the calculated columns?
  - Use `transmute()`.
- Utilize creation functions to calculate new columns.
  - **Must be vectorised** (i.e., takes a vector as input, returns a vector with the same number of values as output)

'Useful' creation functions

- Arithmetic operators
  - `+`, `-`, `*`, `/`, `^`
- Modular arithmetic
  - `%/%`, `%%`
- Logs
  - `log()`, `log2()`, `log10()`
- Offsets
  - `lead()`, `lag()`
- Cumulative and rolling aggregates
  - `cumsum()`, `cumprod()`, etc.
- Logical comparisons
- Ranking
  - `min_rank()`, `percent_rank()`
- **Note:** not an exhaustive list.

`mutate()` function, example

- What will be the result of the following code?

```r
# What is happening here?
flights_sml <- select(flights,
  year:day,
  ends_with("delay"),
  distance,
  air_time
)

# What is the result?
mutate(flights_sml,
  gain = dep_delay - arr_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours # What concept is happening here?
)
```

- Notice how we can refer to previously created variables in the same `mutate()` function.

The `%>%` operator

The `%>%` (pipe) allows us to chain commands together.

> As suggested by this reading, a good way to pronounce %>% when reading code is "then". ~Hadley Wickham & Garrett Grolemund (authors).

Why should we use the `%>%`?

```r
# filter data for flights into Atlanta
flights_dest_atl <- filter(flights, dest == "ATL")

# Which flights had the longest arrival delay?
flights_atl_arr_delay <- arrange(flights_dest_atl, desc(arr_delay))

# I only care about carriers and arrival delays, use select
flights_carrier_delay <- select(flights_atl_arr_delay, carrier, arr_delay, arr_time)
```

```r
# Same result, but with the `%>%`
# Use ctrl + shift + m on keyboard
flights_carrier_delay <- flights %>%
  filter(dest == "ATL") %>%
  arrange(desc(arr_delay)) %>%
  select(arr_time, arr_delay, carrier)
```

- It makes the code easier to read.

- Keeps us from cluttering up our environment with objects.

- What other benefits does the pipe provide?

The `summarise()` function

`summarise()` collapses a data frame to a single row.

```
# data = data you want to create a new variable from
# new var name = the name your new variable will be
# calc exp = the calculation you want to perform
summarise(<data>, <new var name> = <calc exp>)
```

```
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   delay
##   <dbl>
## 1  12.6
```

`summarise()` and `group_by()`

When paired with `group_by()`, it changes the unit of analysis from the whole data to specific subsets.

```
# Average departure delay by day
flights %>%
  group_by(year, month, day) %>%
  summarise(delay = mean(dep_delay, na.rm = TRUE))
```

```
## `summarise()` regrouping output by 'year', 'month' (override with `.groups` argument)
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [12]
##     year month   day delay
##    <int> <int> <int> <dbl>
##  1  2013     1     1 11.5 
##  2  2013     1     2 13.9 
##  3  2013     1     3 11.0 
##  4  2013     1     4  8.95
##  5  2013     1     5  5.73
##  6  2013     1     6  7.15
##  7  2013     1     7  5.42
##  8  2013     1     8  2.55
```

Useful summary functions

- Measures of location
  - `mean()`, `median()`
- Measures of spread
  - `sd()`, `IQR()`, `mad()`
- Measures of rank
  - `min()`, `max()`, `quantile()`
- Measures of position
  - `first()`, `nth()`, `last()`
- Counts
  - `n()`, `sum(!is.na(x))`, `n_distinct()`, `count()`
- Counts and proportions of logical values
  - e.g. `sum(x > 10)`, `mean(y == 0)`

Grouping by multiple variables

- Each summary peels off one level of the grouping.
- This works with sums and counts. **Be careful with rank-based statistics**.

```r
# Daily flights
daily <- group_by(flights, year, month, day)
(per_day   <- summarise(daily, flights = n()))

# Flights per month
(per_month <- summarise(per_day, flights = sum(flights)))

# Flights per year
(per_year  <- summarise(per_month, flights = sum(flights)))
```

Ungrouping

- Removing grouping is easy, just use `ungroup()`.
- Useful when you need to do more data wrangling.

```
daily ← group_by(flights, year, month, day)

daily
```

```
## # A tibble: 336,776 x 19
## # Groups:   year, month, day [365]
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # … with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
```

Grouped mutates (and filters)

- The book suggests avoiding these, except in cases of quick manipulations.

- It's hard to check that the manipulation was done correctly.

- Book suggests checking out window functions `vignette("window-functions)`.

```r
# Return only destinations with 365 flights
popular_dests <- flights %>%
  group_by(dest) %>%
  filter(n() > 365)

popular_dests
```

Questions/Discussion

- What examples/exercises did you find most useful from the reading?
- What examples/exercises gave you the most trouble?
- What is the most useful thing you took away from this chapter?