zeppelin universität

zwischen
Wirtschaft Kultur Politik

Data Analytics and Visualisation

II Data Manipulation — The Heart of Data Science

David Zimmermann

2017-04-22

Zeppelin University

Why do we care about data manipulation?

> *It's an absolute myth that you can send an algorithm over raw data and have insights pop up [. . .] Data scientists [. . .] spend 50-80% of their time mired in this more mundane labor of collecting and preparing unruly digital data, before it can be explored for useful nuggets.*

https://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html

Outline

1. Recap
2. Pipes
3. `hadleyverse`
4. Tibbles
5. Data Import & Export
6. `tidy`-data
7. Data Manipulation
8. Data Merging

– Appendix –

9. Version Control
10. Dates and Times

Recap

Data Types & Structures

What have we done yesterday?

- Math: `+`, `-`, `*`, `/`, `^`, `sqrt()`, `log()`, `exp()`, and `%%`
- Boolean: `==`, `<`, `<=`, `>`, `>=`, `!`, and `%in%`. Also `&` and `|` (as well as `&&` and `||`)
- Variables: `x <- 123`
- Data types: `numeric`, `character`, `logical`, `integer`, and `complex`
- Additional: `NAs`, `Inf`, `-Inf`, `NaN`, etc.
- If-Statements: `if (test) {...} else if (test2) {...} else {...}`, or `ifelse()`
- For-loops: `for (var in vector) {...}`

Data Types & Structures cont'd

- Data Structures:
  - Vector: `c()`, `length()`, and `[]`. Also: `seq()` and `rep()`
- Functions:

```
foo <- function(arg1, arg2 = 123) {
  ... # do something
  return(var)
}
foo(arg1, arg2)
```

- Libraries: once `install.packages("pkg_name")` then each session or top of script `library(pkg_name)`
- Style: `just.Dont_fuckUp.YouRStyle` and `use_common_sense` (i.e., underscore) and consistency...
- Organization: common sense and consistency ...

Debugging & Troubleshooting

What now?

0. Read the error message and RTFM!
1. Rubber-Ducky-Method (-> Rubber Duck Debugging)!
2. Recreate the error with a Minimum Working Example (MWE)!
3. Google is your friend!!!!11
4. Ask friends/collegues/supervisors

MWE:

- https://stackoverflow.com/help/mcve
- https://stackoverflow.com/a/5963610/3048453

The hadleyverse!

Hadley Wickham

Source: http://hadley.nz/

Chief Scientist RStudio & Adjunct Prof. of Statistics

Among many others books: author of "R for Data Science" (http://r4ds.had.co.nz/)

Author of many packages...

Hadleyverse

- `tibble`: improved data structures
- `readr`: improved "read and write" data
- `dplyr`: improved data manipulation
- `tidyr`: improved data tidying
- `ggplot2`: improved graphics
- `lubridate`: improved dates
- `stringr`: improved string manipulation
- `devtools`: improved development
- `roxygen2`: improved package development
- ...

http://adolfoalvarez.cl/the-hitchhikers-guide-to-the-hadleyverse/

Piping

Piping 101

So far: nested code

```
mean(abs(rnorm(1000)))
```

```
## [1] 0.7854796
```

Read: Take the mean of absolutes of random values, 1000

Using pipes (%>%, read as "then") it becomes

```
library(dplyr) # pipes need to be loaded
# alternatively library(magrittr)
1000 %>% rnorm %>% abs %>% mean
```

```
## [1] 0.7854796
```

Read left to right: Take 1000 (then) random values, then take the absolute, then compute the mean

Piping cont'd

Take what is on the left side of the pipe-operator and paste it into the right side function (in the first argument or use '.' to specify location).

`log(100)` is equivalent to `100 %>% log`

To specify location use '.':

```
"David" %>% paste("I am", .) %>%
  rep(3) %>% print
```

```
## [1] "I am David" "I am David" "I am David"
```

Equivalent to

```
print(rep(paste("I am", "David"), 3))
```

Piping cont'd

With piping:

- `f(x)` becomes `x %>% f` or `x %>% f()`
- `f(x, y)` becomes `x %>% f(y)`
- `h(g(f(x)))` becomes `x %>% f %>% g %>% h`

Placeholders:

- `f(y, x)` becomes `x %>% f(y, .)`
- `f(y, z = x)` becomes `x %>% f(y, z = .)`

Additional Infos:

https://github.com/smbache/magrittr/tree/dev

```
vignette("magrittr")
```

Exercises: Piping

Data.frames and data_frames and tibbles

data.frames vs. data_frames vs. tibbles

Goal: have one data type per column, but different column types

Solution: `data.frame` or the more advanced `data_frame` (belongs to `tibble`)

- base-r provides data.frames. Hadley Wickham created data_frames and tibbles (later two are almost identical).
- We are going to focus on data_frames (tibbles) as they are more user friendly and more performant.
- If you have the issue of very large data and performance drags you down, use `data.table`.

Tibbles

```
library(tibble)
# tibble or data_frame
df <- tibble(
  id = 1:3,
  value = rnorm(3),
  group = c("A", "B", "C")
)
df
```

```
## # A tibble: 3 × 3
##      id       value group
##   <int>       <dbl> <chr>
## 1     1 -0.99579872     A
## 2     2 -1.03995504     B
## 3     3 -0.01798024     C
```

See also:

```
vignette("tibble")
```

Tibbles cont'd

Dimension, structure, and summary statistics of a tibble

```
dim(df); glimpse(df)
```

```
## [1] 3 3
```

```
## Observations: 3
## Variables: 3
## $ id    <int> 1, 2, 3
## $ value <dbl> -0.99579872, -1.03995504, -0.01798024
## $ group <chr> "A", "B", "C"
```

Tibbles cont'd

```
summary(df)
```

```
##        id          value              group
##  Min.   :1.0   Min.   :-1.03995   Length:3
##  1st Qu.:1.5   1st Qu.:-1.01788   Class :character
##  Median :2.0   Median :-0.99580   Mode  :character
##  Mean   :2.0   Mean   :-0.68458
##  3rd Qu.:2.5   3rd Qu.:-0.50689
##  Max.   :3.0   Max.   :-0.01798
```

Tibbles cont'd

Names and overviews

```
names(df)
```

```
## [1] "id"    "value" "group"
```

```
# head() shows the first n rows (default = 6)
# show the last n rows with: tail(df)
head(df, 2)
```

```
## # A tibble: 2 × 3
##      id      value group
##   <int>      <dbl> <chr>
## 1     1 -0.9957987     A
## 2     2 -1.0399550     B
```

Accessing a Tibble

Access using [row#, col#] or [col#]

```
# first row, all columns
df[1, ]
```

```
## # A tibble: 1 × 3
##      id      value group
##   <int>      <dbl> <chr>
## 1     1 -0.9957987     A
```

```
# first variable (column)
# same as df[1]
df[, 1]
```

```
## # A tibble: 3 × 1
##      id
##   <int>
## 1     1
## 2     2
## 3     3
```

```
# first row, first column
df[1, 1]
```

```
## # A tibble: 1 × 1
##      id
##   <int>
## 1     1
```

Accessing a Tibble cont'd

Avoid numbers use variable names if possible using `[var]` or `$var`

```
# returns a tibble
# same as df["group"]
df[, "group"]
```

```
## # A tibble: 3 × 1
##   group
##   <chr>
## 1     A
## 2     B
## 3     C
```

```
df$group # returns a vector
```

```
## [1] "A" "B" "C"
```

Accessing a Tibble cont'd

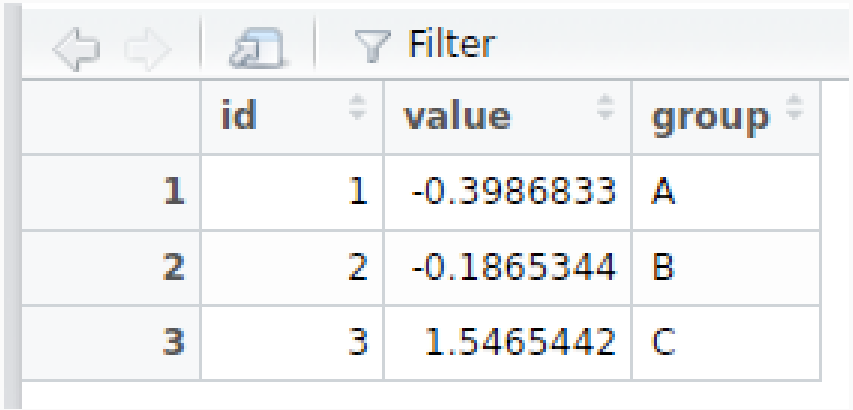
```
selected_variables <- c("id", "value")
# same as df[, selected_variables]
df[selected_variables]
```

```
## # A tibble: 3 × 2
##      id       value
##   <int>       <dbl>
## 1     1 -0.99579872
## 2     2 -1.03995504
## 3     3 -0.01798024
```

Accessing a Tibble cont'd

If you want to see all data in a spreadsheet-format use

```
View(df)
```



	id	value	group
1	1	-0.3986833	A
2	2	-0.1865344	B
3	3	1.5465442	C

Exercises: Tibbles

Data Import & Export

Data Import & Export

Realistic case: share data via Dropbox/GitHub/other service.

Best format: plain text, i.e., `.csv` (readable by every program (try text-editor...))

R: possible: `read.csv/write.csv` but better using Hadley's library `readr` and its `read_csv/write_csv`

Readr

```r
library(readr)

# read data
df <- read_csv("myfile.csv")
# also read_csv("https://...") possible

# write file
write_csv(df, "myfile.csv")
```

More information: https://github.com/hadley/readr

Use RStudio projects, or `setwd()`, and path navigation `folder/file.csv` (relativ path from current working directory) to find the file (try also `file.exists("file.csv")`).

Other formats

Everything that is not saved in plain-text or comes from a non open-source tool

- **MS Excel**:
  - `readxl`: https://github.com/hadley/readxl
- **Stata**, **SAS**, **SPSS**:
  - `haven`: https://github.com/tidyverse/haven

Exercises: Data IO

tidy-data

Tidy Data

Two different data concepts: Wide vs. Long data formats

A Wide Table

stock	2010	2011
GOOGL	297.28	323.27
AAPL	42.18	52.96

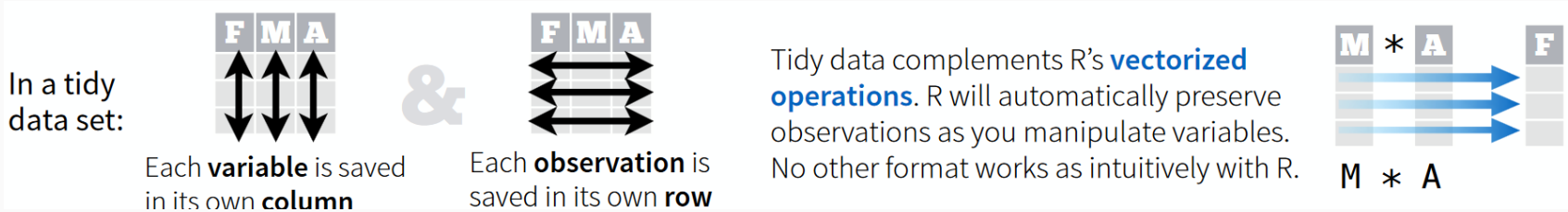
A Long Table

ticker	year	price
GOOGL	2010	297.28
GOOGL	2011	323.27
AAPL	2010	42.18
AAPL	2011	52.96

- Wide data: ???
- Long data: Each variable in a column, each observation in a row

Tidy Data

- Advantages wide data: more dense (saves space and data)
- Advantages long data: easier for computation, manipulation, etc.



Source: Data Wrangling Cheatsheet

Conversion Wide to Long

```
library(tidyr)
wide_df <- tibble(stock = c("GOOGL", "AAPL"),
                  "2010" = c(297.28, 42.18),
                  "2011" = c(323.27, 52.96))

long_df <- gather(wide_df, key = year, value = price, -stock)
# -stock indicates the variables for which we do the reshape
# can be multiple variables
long_df
```

```
## # A tibble: 4 × 3
##   stock  year  price
##   <chr> <chr>  <dbl>
## 1 GOOGL  2010 297.28
## 2  AAPL  2010  42.18
## 3 GOOGL  2011 323.27
## 4  AAPL  2011  52.96
```

Computing Returns

More on the syntax in the next chapter

Long format:

```
library(dplyr)
long_df %>%
  group_by(stock) %>%
  mutate(ret = price / lag(price) - 1)
```

```
## Source: local data frame [4 x 4]
## Groups: stock [2]
##
##   stock  year  price       ret
##   <chr> <chr>  <dbl>     <dbl>
## 1 GOOGL  2010 297.28        NA
## 2  AAPL  2010  42.18        NA
## 3 GOOGL  2011 323.27 0.0874260
## 4  AAPL  2011  52.96 0.2555714
```

Conversion Long to Wide

```
wide_df2 <- spread(long_df, key = year, value = price)
wide_df2
```

```
## # A tibble: 2 × 3
##   stock `2010` `2011`
## * <chr>  <dbl>  <dbl>
## 1  AAPL  42.18  52.96
## 2 GOOGL 297.28 323.27
```

Separating Columns

Separate one column into multiple columns I.e., create a month, year, and date column

```
year <- 2000
month <- 1:12
day <- 1
df <- tibble(date_mess = paste(month, day, year, sep = "_"))
```

```
df %>% head
```

```
## # A tibble: 6 × 1
##   date_mess
##       <chr>
## 1  1_1_2000
## 2  2_1_2000
## 3  3_1_2000
## 4  4_1_2000
## 5  5_1_2000
## 6  6_1_2000
```

Separating Columns cont'd

```
df <- df %>% separate(col = date_mess,
                      into = c("month", "day", "year"),
                      sep = "_")
```

```
df %>% head
```

```
## # A tibble: 6 × 3
##   month   day   year
##   <chr> <chr> <chr>
## 1     1     1  2000
## 2     2     1  2000
## 3     3     1  2000
## 4     4     1  2000
## 5     5     1  2000
## 6     6     1  2000
```

Uniting Columns

Combining the three date-related columns into a proper date column

```
df <- df %>% unite(col = "date_tidy",
                   year, month, day, sep = "-")
```

```
df %>% head
```

```
## # A tibble: 6 × 1
##   date_tidy
##       <chr>
## 1  2000-1-1
## 2  2000-2-1
## 3  2000-3-1
## 4  2000-4-1
## 5  2000-5-1
## 6  2000-6-1
```

Additional Links

More Information on reshaping the data, how to use `spread` and `gather`

- http://www.cookbook-r.com/Manipulating_data/Converting_data_between_wide_and_long_format/
- http://r4ds.had.co.nz/tidy.html
- https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

Exercises: Tidy Data

Data Manipulation

Data Manipulation using `dplyr`

`dplyr`: library to make data manipulation easier and more logically.

Main functions:

1. filter and select data
2. compute new variables
3. summarise data by a grouping variable
4. merge datasets

More information: https://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html

```
# install.packages("dplyr")
library(dplyr)
```

Dataset:

Figure 1: NYC Flights 2013

Data Overview

```
library(nycflights13)
?flights
```

> *On-time data for all flights that departed NYC (i.e. JFK, LGA or EWR) in 2013.*

Data Overview cont'd

```
flights %>% dim
```

```
## [1] 336776     19
```

```
flights %>% names
```

```
##  [1] "year"           "month"          "day"
##  [4] "dep_time"       "sched_dep_time" "dep_delay"
##  [7] "arr_time"       "sched_arr_time" "arr_delay"
## [10] "carrier"        "flight"         "tailnum"
## [13] "origin"         "dest"           "air_time"
## [16] "distance"       "hour"           "minute"
## [19] "time_hour"
```

Data Overview cont'd

```
# flights %>% head # same as
flights %>% top_n(5)
```

```
## Selecting by time_hour
```

```
## # A tibble: 5 × 19
##    year month   day dep_time sched_dep_time dep_delay
##   <int> <int> <int>    <int>          <int>     <dbl>
## 1  2013    12    31       13           2359        14
## 2  2013    12    31       18           2359        19
## 3  2013    12    31     2328           2330        -2
## 4  2013    12    31     2355           2359        -4
## 5  2013    12    31     2356           2359        -3
## # ... with 13 more variables: arr_time <int>,
## #   sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>,
```

Data Overview cont'd

```
flights %>% glimpse
```

```
## Observations: 336,776
## Variables: 19
## $ year           <int> 2013, 2013, 2013, 2013, 201...
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ dep_time       <int> 517, 533, 542, 544, 554, 55...
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 55...
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3...
## $ arr_time       <int> 830, 850, 923, 1004, 812, 7...
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 7...
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 1...
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL...
## $ flight         <int> 1545, 1714, 1141, 725, 461,...
## $ tailnum        <chr> "N14228", "N24211", "N619AA...
## $ origin         <chr> "EWR", "LGA", "JFK", "JFK",...
## $ dest           <chr> "IAH", "IAH", "MIA", "BQN",...
## $ air_time       <dbl> 227, 227, 160, 183, 116, 15...
## $ distance       <dbl> 1400, 1416, 1089, 1576, 762...
## $ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, ...
## $ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0...
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-...
```

Data Overview cont'd

```
flights %>% summary
```

```
##       year          month             day
##  Min.   :2013   Min.   : 1.000   Min.   : 1.00
##  1st Qu.:2013   1st Qu.: 4.000   1st Qu.: 8.00
##  Median :2013   Median : 7.000   Median :16.00
##  Mean   :2013   Mean   : 6.549   Mean   :15.71
##  3rd Qu.:2013   3rd Qu.:10.000   3rd Qu.:23.00
##  Max.   :2013   Max.   :12.000   Max.   :31.00
##
##     dep_time    sched_dep_time   dep_delay
##  Min.   :   1   Min.   : 106   Min.   : -43.00
##  1st Qu.: 907   1st Qu.: 906   1st Qu.:  -5.00
##  Median :1401   Median :1359   Median :  -2.00
##  Mean   :1349   Mean   :1344   Mean   :  12.64
##  3rd Qu.:1744   3rd Qu.:1729   3rd Qu.:  11.00
##  Max.   :2400   Max.   :2359   Max.   :1301.00
##  NA's   :8255                  NA's   :8255
##     arr_time    sched_arr_time   arr_delay
##  Min.   :   1   Min.   :   1   Min.   : -86.000
##  1st Qu.:1104   1st Qu.:1124   1st Qu.: -17.000
##  Median :1535   Median :1556   Median :  -5.000
##  Mean   :1502   Mean   :1536   Mean   :   6.895
##  3rd Qu.:1940   3rd Qu.:1945   3rd Qu.:  14.000
##  Max.   :2400   Max.   :2359   Max.   :1272.000
```

Data Overview cont'd

```
flights %>% slice(20:25)
```

```
## # A tibble: 6 × 19
##    year month   day dep_time sched_dep_time dep_delay
##   <int> <int> <int>    <int>          <int>     <dbl>
## 1  2013     1     1      601            600         1
## 2  2013     1     1      602            610        -8
## 3  2013     1     1      602            605        -3
## 4  2013     1     1      606            610        -4
## 5  2013     1     1      606            610        -4
## 6  2013     1     1      607            607         0
## # ... with 13 more variables: arr_time <int>,
## #   sched_arr_time <int>, 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>
```

Filter

Filter Rows

Using boolean operators ==, !=, <=, %in%, etc.

Q: Which flights on christmans had more than 60 mins delay?

```
flights %>% filter(month == 12, day == 24, arr_delay > 60)
```

```
## # A tibble: 17 × 19
##     year month   day dep_time sched_dep_time dep_delay
##    <int> <int> <int>    <int>          <int>     <dbl>
## 1   2013    12    24      640            551        49
## 2   2013    12    24      812            701        71
## 3   2013    12    24     1022            800       142
## 4   2013    12    24     1026            900        86
## 5   2013    12    24     1034            947        47
## 6   2013    12    24     1035            835       120
## 7   2013    12    24     1206           1100        66
## 8   2013    12    24     1349           1215        94
## 9   2013    12    24     1413           1310        63
## 10  2013    12    24     1630           1455        95
## 11  2013    12    24     1739           1600        99
```

Filter Rows cont'd

Nested checks using & and | also possible!

Q: Which flights on christmans or on new years', had more than 60 mins delay and (thus) arrived after 8?

```
flights %>% filter(month == 12, day == 24 | day == 31,
                   arr_delay > 60 & arr_time > 2000)
```

```
## # A tibble: 13 × 19
##     year month   day dep_time sched_dep_time dep_delay
##    <int> <int> <int>    <int>          <int>     <dbl>
## 1   2013    12    24     1750           1535       135
## 2   2013    12    24     1801           1350       251
## 3   2013    12    24     1932           1715       137
## 4   2013    12    24     2016           1530       286
## 5   2013    12    24     2059           1729       210
## 6   2013    12    31     1649           1535        74
## 7   2013    12    31     1819           1505       194
## 8   2013    12    31     1853           1805        48
## 9   2013    12    31     1901           1730        91
```

Arrange Rows

Sort the dataset

Q: Which flights departed earliest?

```
flights %>% arrange(dep_delay)
```

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay
##    <int> <int> <int>    <int>          <int>     <dbl>
## 1   2013    12     7     2040           2123       -43
## 2   2013     2     3     2022           2055       -33
## 3   2013    11    10     1408           1440       -32
## 4   2013     1    11     1900           1930       -30
## 5   2013     1    29     1703           1730       -27
## 6   2013     8     9      729            755       -26
## 7   2013    10    23     1907           1932       -25
## 8   2013     3    30     2030           2055       -25
## 9   2013     3     2     1431           1455       -24
## 10  2013     5     5      934            958       -24
## #     with 336,766 more rows, and 13 more variables:
```

Arrange Rows cont'd

Q: Which flights departed latest?

```
flights %>% arrange(desc(dep_delay))
```

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay
##    <int> <int> <int>    <int>          <int>     <dbl>
## 1   2013     1     9      641            900      1301
## 2   2013     6    15     1432           1935      1137
## 3   2013     1    10     1121           1635      1126
## 4   2013     9    20     1139           1845      1014
## 5   2013     7    22      845           1600      1005
## 6   2013     4    10     1100           1900       960
## 7   2013     3    17     2321            810       911
## 8   2013     6    27      959           1900       899
## 9   2013     7    22     2257            759       898
## 10  2013    12     5      756           1700       896
## # ... with 336,766 more rows, and 13 more variables:
## #   arr_time <int>, sched_arr_time <int>,
```

Select


I DON'T ALWAYS
QUERY DATA
BUT WHEN I DO, I
SELECT *
memeshappen.com

Select Variables

```
flights %>% select(dep_delay, arr_delay, airline = carrier)
```

```
## # A tibble: 336,776 × 3
##    dep_delay arr_delay airline
##        <dbl>     <dbl>   <chr>
## 1          2        11      UA
## 2          4        20      UA
## 3          2        33      AA
## 4         -1       -18      B6
## 5         -6       -25      DL
## 6         -4        12      UA
## 7         -5        19      B6
## 8         -3       -14      EV
## 9         -3        -8      B6
## 10        -2         8      AA
## # ... with 336,766 more rows
```

Deselect Variables & Renaming

```
flights %>% select(-year, -month, -day)
```

```
## # A tibble: 336,776 × 16
##    dep_time sched_dep_time dep_delay arr_time
##       <int>          <int>     <dbl>    <int>
## 1       517            515         2      830
## 2       533            529         4      850
## 3       542            540         2      923
## 4       544            545        -1     1004
## 5       554            600        -6      812
## 6       554            558        -4      740
## 7       555            600        -5      913
## 8       557            600        -3      709
## 9       557            600        -3      838
## 10      558            600        -2      753
## # ... with 336,766 more rows, and 12 more variables:
## #   sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>
```

Select Variables cont'd

Q: Select all variables related to delay.

Also: `starts_with()`, `ends_with()`, `one_of()`, `matches()`, ...

```
flights %>% select(contains("delay"))
```

```
## # A tibble: 336,776 × 2
##    dep_delay arr_delay
##        <dbl>     <dbl>
## 1          2        11
## 2          4        20
## 3          2        33
## 4         -1       -18
## 5         -6       -25
## 6         -4        12
## 7         -5        19
## 8         -3       -14
## 9         -3        -8
## 10        -2         8
## #     with 336,766 more rows
```

Select Distinct Variables

Q: What carriers operated at JFK?

```
flights %>% filter(origin == "JFK") %>% select(carrier) %>%
  distinct
```

```
## # A tibble: 10 × 1
##    carrier
##      <chr>
## 1       AA
## 2       B6
## 3       UA
## 4       DL
## 5       US
## 6       VX
## 7       MQ
## 8       9E
## 9       HA
## 10      EV
```

Create New Variables

Q: What where the fastest flights?

```
flights %>%
  mutate(avg_speed = 1.609 * distance / air_time * 60,
         gain = dep_delay - arr_delay) %>%
  select(year, month, day, carrier, avg_speed, gain) %>%
  arrange(avg_speed)
```

```
## # A tibble: 336,776 × 6
##     year month   day carrier avg_speed  gain
##    <int> <int> <int>   <chr>     <dbl> <dbl>
## 1   2013     1    28      US  123.5712   -51
## 2   2013     6    29      B6  136.2515   -91
## 3   2013     8    28      9E  148.7666   -33
## 4   2013     1    30      9E  153.8095   -39
## 5   2013    11    27      US  154.4640   -33
## 6   2013     5    21      US  154.4640   -26
## 7   2013    12     9      US  157.0820   -19
## 8   2013     6    10      US  157.7066   -96
```

Saving the output

```
# if you want to save the new variable
flights_new <- flights %>%
  mutate(avg_speed = 1.609 * distance / air_time * 60) %>%
  select(year, month, day, carrier, avg_speed) %>%
  arrange(avg_speed)
```

```
flights_new
```

```
## # A tibble: 336,776 × 5
##     year month   day carrier avg_speed
##    <int> <int> <int>   <chr>     <dbl>
## 1   2013     1    28      US  123.5712
## 2   2013     6    29      B6  136.2515
## 3   2013     8    28      9E  148.7666
## 4   2013     1    30      9E  153.8095
## 5   2013    11    27      US  154.4640
## 6   2013     5    21      US  154.4640
## 7   2013    12     9      US  157.0820
```

Summarise Variables

Summarising the table is easiest done using `summarise`

Q: What was the average (standard deviation) of departure delays?

```
flights %>%
  summarise(avg_delay = mean(dep_delay, na.rm = TRUE),
            sd_delay = sd(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 1 × 2
##   avg_delay sd_delay
##       <dbl>    <dbl>
## 1  12.63907 40.21006
```

Summarise Grouped Variables

Q: What was the average departure/arrival delay by carrier (+ no. of flights)?

```
flights %>%
  group_by(carrier) %>%
  summarise(n_flights = n(),
            avg_arr_delay = mean(arr_delay, na.rm = T),
            avg_dep_delay = mean(dep_delay, na.rm = T)) %>%
  arrange(avg_arr_delay)
```

```
## # A tibble: 16 × 4
##    carrier n_flights avg_arr_delay avg_dep_delay
##      <chr>     <int>         <dbl>         <dbl>
## 1       AS       714    -9.9308886      5.804775
## 2       HA       342    -6.9152047      4.900585
## 3       AA     32729     0.3642909      8.586016
## 4       DL     48110     1.6443409      9.264505
## 5       VX      5162     1.7644644     12.869421
## 6       US     20536     2.1295951      3.782418
```

Mutate Grouped Variables

Q: Select the flights where the flight operator had the highest standard deviation in delays.

```
flights %>%
  group_by(carrier) %>%
  mutate(sd_arr_delay = sd(arr_delay)) %>%
  select(carrier, dep_time, arr_time, arr_delay, sd_arr_delay) %>%
  arrange(sd_arr_delay)
```

```
## Source: local data frame [336,776 x 5]
## Groups: carrier [16]
##
##    carrier dep_time arr_time arr_delay sd_arr_delay
##      <chr>    <int>    <int>     <dbl>        <dbl>
## 1       HA      857     1516       -14     75.12942
## 2       HA      909     1525        -5     75.12942
## 3       HA      914     1504       -26     75.12942
## 4       HA      900     1516       -14     75.12942
## 5       HA      858     1519       -11     75.12942
```

Sample Data

In some cases, you might want to look at only a fraction of the data

```
set.seed(123) # for reproducibility
flights %>% sample_frac(0.1, replace = T)
```

```
## # A tibble: 33,678 × 19
##     year month   day dep_time sched_dep_time dep_delay
##    <int> <int> <int>    <int>          <int>     <dbl>
## 1   2013    12    15     2124           2128        -4
## 2   2013     7    17      652            700        -8
## 3   2013     3     2     1637           1645        -8
## 4   2013     8    19     1059            755       184
## 5   2013     9     9     1252           1246         6
## 6   2013     1    18     1259           1300        -1
## 7   2013     4    14     1502           1508        -6
## 8   2013     8    22     1531           1416        75
## 9   2013     4    22     1957           1810       107
## 10  2013     3    19     2003           2015       -12
```

Sample Data cont'd

. . . or you want to sample a precise number of observations

```
flights %>% sample_n(100, replace = T)
```

```
## # A tibble: 100 × 19
##     year month   day dep_time sched_dep_time dep_delay
##    <int> <int> <int>    <int>          <int>     <dbl>
## 1   2013    10    30     1417           1425        -8
## 2   2013     9    29     1153           1200        -7
## 3   2013     1    12     1258           1300        -2
## 4   2013     5     8       NA           2130        NA
## 5   2013    10    11     1723           1700        23
## 6   2013     5     4     1124           1125        -1
## 7   2013     4    25     1718           1529       109
## 8   2013     2     5     1046           1050        -4
## 9   2013    12    10     1717           1655        22
## 10  2013    12     5      927            930        -3
## # ... with 90 more rows, and 13 more variables:
## #   arr_time <int>, sched_arr_time <int>,
```

Recap: Data Wrangling using `dplyr`

- Piping (%>%) and `dplyr` make your life easier
- `filter` for filtering rows
- `arrange` for arranging rows
- `select` for selecting variables
- `distinct` for selecting distinct rows
- `mutate` for creating new variables (columns)
- `summarise` for summarising the data
- `group_by` for grouping operations

Further Reading and Useful Links

- Dplyr Vignettes: https://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html
- Data Wrangling Cheatsheet: https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

Exercises: Data Manipulation using dplyr

Data Merging

What is a Join

Table 1: Superheroes

superhero	alignment	publisher
Batman	good	DC
Joker	bad	DC
Xavier	good	Marvel
Magneto	bad	Marvel

Table 2: Publishers

publisher	address
DC	Burbank (CA)
Marvel	NYC (NY)

Merge on *"publisher"*

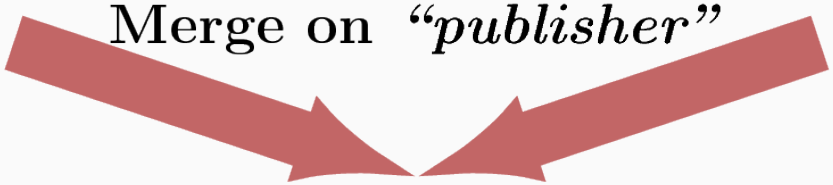


Table 3: Superheroes expanded

superhero	alignment	publisher	address
Batman	good	DC	Burbank (CA)
Joker	bad	DC	Burbank (CA)
Xavier	good	Marvel	NYC (NY)
Magneto	bad	Marvel	NYC (NY)

Expanding the Data

Table 1: Superheroes

superhero	alignment	publisher
Batman	good	DC
Joker	bad	DC
Xavier	good	Marvel
Magneto	bad	Marvel
Hellboy	good	Dark Horse

Table 2: Publishers

publisher	address
DC	Burbank (CA)
Marvel	NYC (NY)
Image Comics	Berkeley (CA)

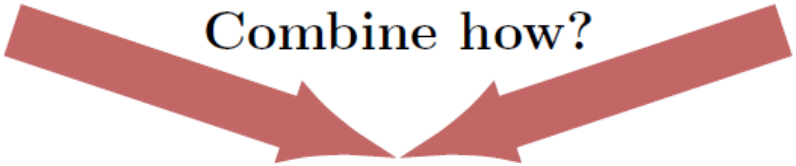


Table 3: Superheroes expanded

superhero	alignment	publisher	address
Batman	good	DC	Burbank (CA)
Joker	bad	DC	Burbank (CA)
Xavier	good	Marvel	NYC (NY)
Magneto	bad	Marvel	NYC (NY)
???	???	???	???

Left Join

Table 1: Superheroes

superhero	alignment	publisher
Batman	good	DC
Joker	bad	DC
Xavier	good	Marvel
Magneto	bad	Marvel
Hellboy	good	Dark Horse

Table 2: Publishers

publisher	address
DC	Burbank (CA)
Marvel	NYC (NY)
Image Comics	Berkeley (CA)

Table 3: Superheroes expanded

superhero	alignment	publisher	address
Batman	good	DC	Burbank (CA)
Joker	bad	DC	Burbank (CA)
Xavier	good	Marvel	NYC (NY)
Magneto	bad	Marvel	NYC (NY)
Hellboy	good	Dark Horse	NA

Right Join

Table 1: Superheroes

superhero	alignment	publisher
Batman	good	DC
Joker	bad	DC
Xavier	good	Marvel
Magneto	bad	Marvel
Hellboy	good	Dark Horse

Table 2: Publishers

publisher	address
DC	Burbank (CA)
Marvel	NYC (NY)
Image Comics	Berkeley (CA)

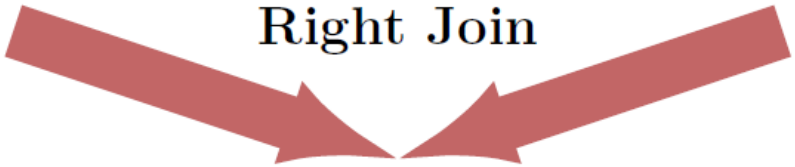


Table 3: Superheroes expanded

superhero	alignment	publisher	address
Batman	good	DC	Burbank (CA)
Joker	bad	DC	Burbank (CA)
Xavier	good	Marvel	NYC (NY)
Magneto	bad	Marvel	NYC (NY)
NA	NA	Image Comics	Berkeley (CA)

Inner Join

Table 1: Superheroes

superhero	alignment	publisher
Batman	good	DC
Joker	bad	DC
Xavier	good	Marvel
Magneto	bad	Marvel
Hellboy	good	Dark Horse

Table 2: Publishers

publisher	address
DC	Burbank (CA)
Marvel	NYC (NY)
Image Comics	Berkeley (CA)

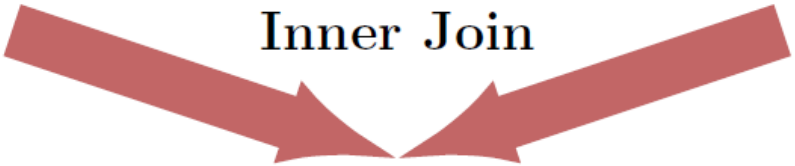


Table 3: Superheroes expanded

superhero	alignment	publisher	address
Batman	good	DC	Burbank (CA)
Joker	bad	DC	Burbank (CA)
Xavier	good	Marvel	NYC (NY)
Magneto	bad	Marvel	NYC (NY)

Outer Join

Table 1: Superheroes

superhero	alignment	publisher
Batman	good	DC
Joker	bad	DC
Xavier	good	Marvel
Magneto	bad	Marvel
Hellboy	good	Dark Horse

Table 2: Publishers

publisher	address
DC	Burbank (CA)
Marvel	NYC (NY)
Image Comics	Berkeley (CA)

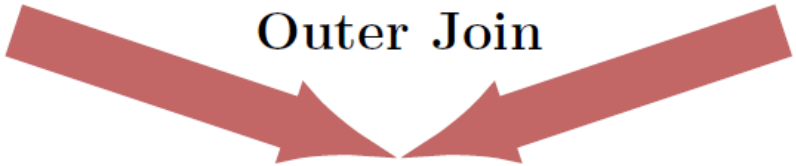
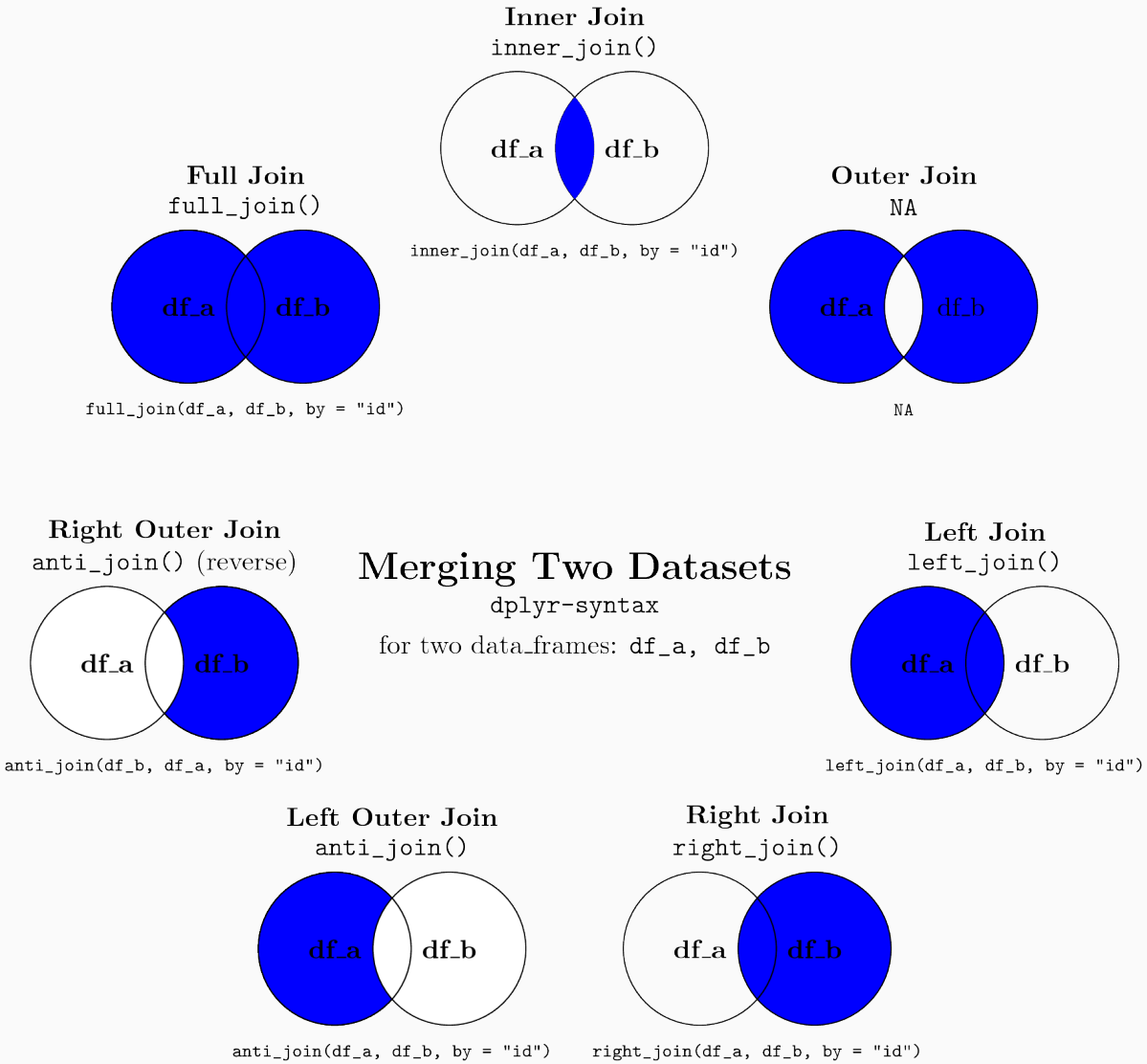


Table 3: Superheroes expanded

superhero	alignment	publisher	address
Hellboy	good	Dark Horse	NA
NA	NA	Image Comics	Berkeley (CA)

Join Overview



Preparing a Join

Taking a subset of our original data and find the geolocations for each airport

```
df_flight <- flights %>%
  select(year, month, day, dep_time, origin, dest)
df_flight
```

```
## # A tibble: 336,776 × 6
##     year month   day dep_time origin  dest
##    <int> <int> <int>    <int>  <chr> <chr>
## 1   2013     1     1      517    EWR   IAH
## 2   2013     1     1      533    LGA   IAH
## 3   2013     1     1      542    JFK   MIA
## 4   2013     1     1      544    JFK   BQN
## 5   2013     1     1      554    LGA   ATL
## 6   2013     1     1      554    EWR   ORD
## 7   2013     1     1      555    EWR   FLL
## 8   2013     1     1      557    LGA   IAD
## 9   2013     1     1      557    JFK   MCO
## 10  2013     1     1      558    LGA   ORD
## # ... with 336,766 more rows
```

Merge & Joins cont'd

Second dataset: `airports`

```
df_airports <- airports %>% select(faa, lat, lon)
df_airports
```

```
## # A tibble: 1,458 × 3
##      faa      lat        lon
##    <chr>    <dbl>      <dbl>
## 1    04G 41.13047  -80.61958
## 2    06A 32.46057  -85.68003
## 3    06C 41.98934  -88.10124
## 4    06N 41.43191  -74.39156
## 5    09J 31.07447  -81.42778
## 6    0A9 36.37122  -82.17342
## 7    0G6 41.46731  -84.50678
## 8    0G7 42.88356  -76.78123
## 9    0P2 39.79482  -76.64719
## 10   0S9 48.05381 -122.81064
## # ... with 1,448 more rows
```

Merge & Joins cont'd

Combine the two datasets on the variables `origin` and `faa`

```
df_airports_1 <- df_airports %>%
  select(faa, lat_origin = lat, lon_origin = lon)
df_flight <- left_join(df_flight, df_airports_1,
                       by = c("origin" = "faa"))
df_flight
```

```
## # A tibble: 336,776 × 8
##     year month   day dep_time origin  dest lat_origin
##    <int> <int> <int>    <int>  <chr> <chr>      <dbl>
## 1   2013     1     1      517    EWR   IAH   40.69250
## 2   2013     1     1      533    LGA   IAH   40.77725
## 3   2013     1     1      542    JFK   MIA   40.63975
## 4   2013     1     1      544    JFK   BQN   40.63975
## 5   2013     1     1      554    LGA   ATL   40.77725
## 6   2013     1     1      554    EWR   ORD   40.69250
## 7   2013     1     1      555    EWR   FLL   40.69250
## 8   2013     1     1      557    LGA   IAD   40.77725
## 9   2013     1     1      557    JFK   MCO   40.63975
## 10  2013     1     1      558    LGA   ORD   40.77725
## # ... with 336,766 more rows, and 1 more variables:
## #   lon_origin <dbl>
```

Merge & Joins cont'd

Add the locations of the destination airports

```
df_airports_2 <- df_airports %>%
  select(faa, lat_dest = lat, lon_dest = lon)
df_flight <- left_join(df_flight, df_airports_2,
                       by = c("dest" = "faa"))
df_flight
```

```
## # A tibble: 336,776 × 10
##     year month   day dep_time origin  dest lat_origin
##    <int> <int> <int>    <int>  <chr> <chr>      <dbl>
## 1   2013     1     1      517    EWR   IAH   40.69250
## 2   2013     1     1      533    LGA   IAH   40.77725
## 3   2013     1     1      542    JFK   MIA   40.63975
## 4   2013     1     1      544    JFK   BQN   40.63975
## 5   2013     1     1      554    LGA   ATL   40.77725
## 6   2013     1     1      554    EWR   ORD   40.69250
## 7   2013     1     1      555    EWR   FLL   40.69250
## 8   2013     1     1      557    LGA   IAD   40.77725
## 9   2013     1     1      557    JFK   MCO   40.63975
## 10  2013     1     1      558    LGA   ORD   40.77725
## # ... with 336,766 more rows, and 3 more variables:
## #   lon_origin <dbl>, lat_dest <dbl>, lon_dest <dbl>
```

Checking Merges

```
df_flight %>% summary
```

```
##       year          month             day
##  Min.   :2013   Min.   : 1.000   Min.   : 1.00
##  1st Qu.:2013   1st Qu.: 4.000   1st Qu.: 8.00
##  Median :2013   Median : 7.000   Median :16.00
##  Mean   :2013   Mean   : 6.549   Mean   :15.71
##  3rd Qu.:2013   3rd Qu.:10.000   3rd Qu.:23.00
##  Max.   :2013   Max.   :12.000   Max.   :31.00
##
##     dep_time       origin              dest
##  Min.   :   1   Length:336776      Length:336776
##  1st Qu.: 907   Class :character   Class :character
##  Median :1401   Mode  :character   Mode  :character
##  Mean   :1349
##  3rd Qu.:1744
##  Max.   :2400
##  NA's   :8255
##    lat_origin      lon_origin        lat_dest
##  Min.   :40.64   Min.   :-74.17   Min.   :21.32
##  1st Qu.:40.64   1st Qu.:-74.17   1st Qu.:32.90
##  Median :40.69   Median :-73.87   Median :36.10
##  Mean   :40.70   Mean   :-73.95   Mean   :36.02
##  3rd Qu.:40.78   3rd Qu.:-73.78   3rd Qu.:41.41
##  Max.   :40.78   Max.   :-73.78   Max.   :61.17
```

Merge & Joins cont'd

Find the missing values

```
df_flight %>% filter(is.na(lat_dest)) %>%
  select(dest) %>% distinct
```

```
## # A tibble: 4 × 1
##    dest
##   <chr>
## 1   BQN
## 2   SJU
## 3   STT
## 4   PSE
```

Further Information

Combining (merging, joining) datasets

- https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf
- http://stat545.com/bit001_dplyr-cheatsheet.html
- https://rpubs.com/NateByers/merging

Exercises: Joins

Additional Information aka. Appendix

Beyond `tibbles` and `dplyr`

Very large datasets and speed is an issue?

Don't use `tibble` and `dplyr` but use `data.table` (currently fastest data-manipulation software, https://github.com/Rdatatable/data.table/wiki/Benchmarks-%3A-Grouping. See 100GB in memory benchmark test)

More information: https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro-vignette.html

https://github.com/Rdatatable/data.table/wiki/Getting-started

Keeping an Eye on Versions

Pro Tip: Use version control (i.e., git)

Why:

http://ellisp.github.io/blog/2016/09/16/version-control

How:

https://try.github.io/levels/1/challenges/1

https://rogerdudler.github.io/git-guide/

http://kbroman.org/github_tutorial/

https://swcarpentry.github.io/git-novice/

Times and Dates

Formatting

Source: https://xkcd.com/1179/

Dates in R

base-r

```
date1 <- as.Date("2000-01-01")
date2 <- as.Date("2000-02-01")
date2 - date1
```

```
## Time difference of 31 days
```

```
(date2 - date1) * 3
```

```
## Time difference of 93 days
```

```
as.numeric(date2 - date1)
```

```
## [1] 31
```

Dates in R cont'd

hadleyverse: `lubridate`

```
library(lubridate)
date3 <- ymd("20000101")
date3
```

```
## [1] "2000-01-01"
```

```
date4 <- ymd_hms("2000-01-01 12:00:00")
date5 <- ymd_hms("2000-01-01 12:00:00", tz = "Europe/Berlin")
date4 - date5
```

```
## Time difference of 1 hours
```

Times

```
date5 <- now()
date5
```

```
## [1] "2017-04-19 10:50:23 CEST"
```

```
month(date5)
```

```
## [1] 4
```

```
minute(date5)
```

```
## [1] 50
```

```
second(date5) <- 00
date5
```

```
## [1] "2017-04-19 10:50:00 CEST"
```

Timezones

Avoid them where possible, but if you have to

```
tz(date4) # -> ?tz
```

```
## [1] "UTC"
```

```
tz(date4) <- "America/Los_Angeles"
date4
```

```
## [1] "2000-01-01 12:00:00 PST"
```

Why? https://www.youtube.com/watch?v=-5wpm-gesOY

Further Information

```
vignette("lubridate")
```

- Vignette: https://cran.r-project.org/web/packages/lubridate/vignettes/lubridate.html
- Cheatsheet: http://blog.yhat.com/static/pdf/R_date_cheat_sheet.pdf