London Business School

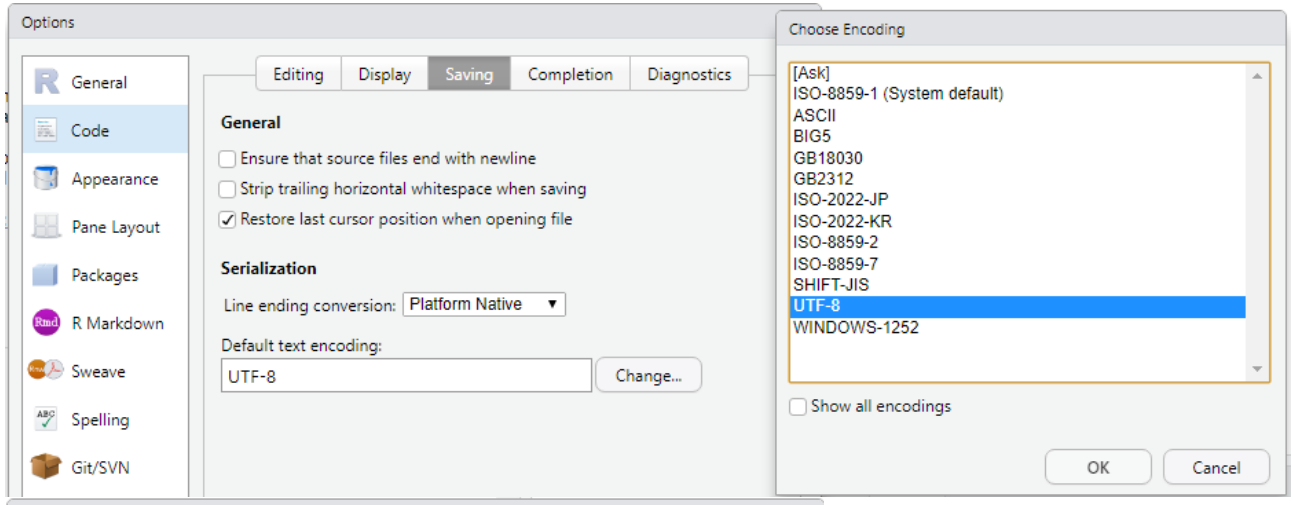
Session 2: Data Science Basics

Kostis Christodoulou
London Business School

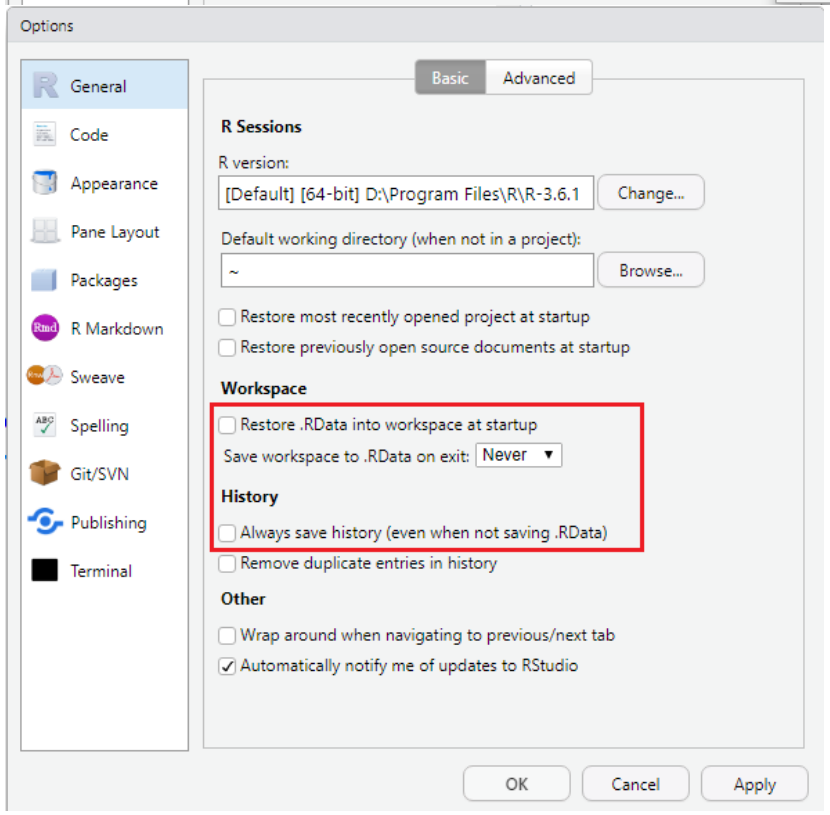
London Business School

R Studio Preferences

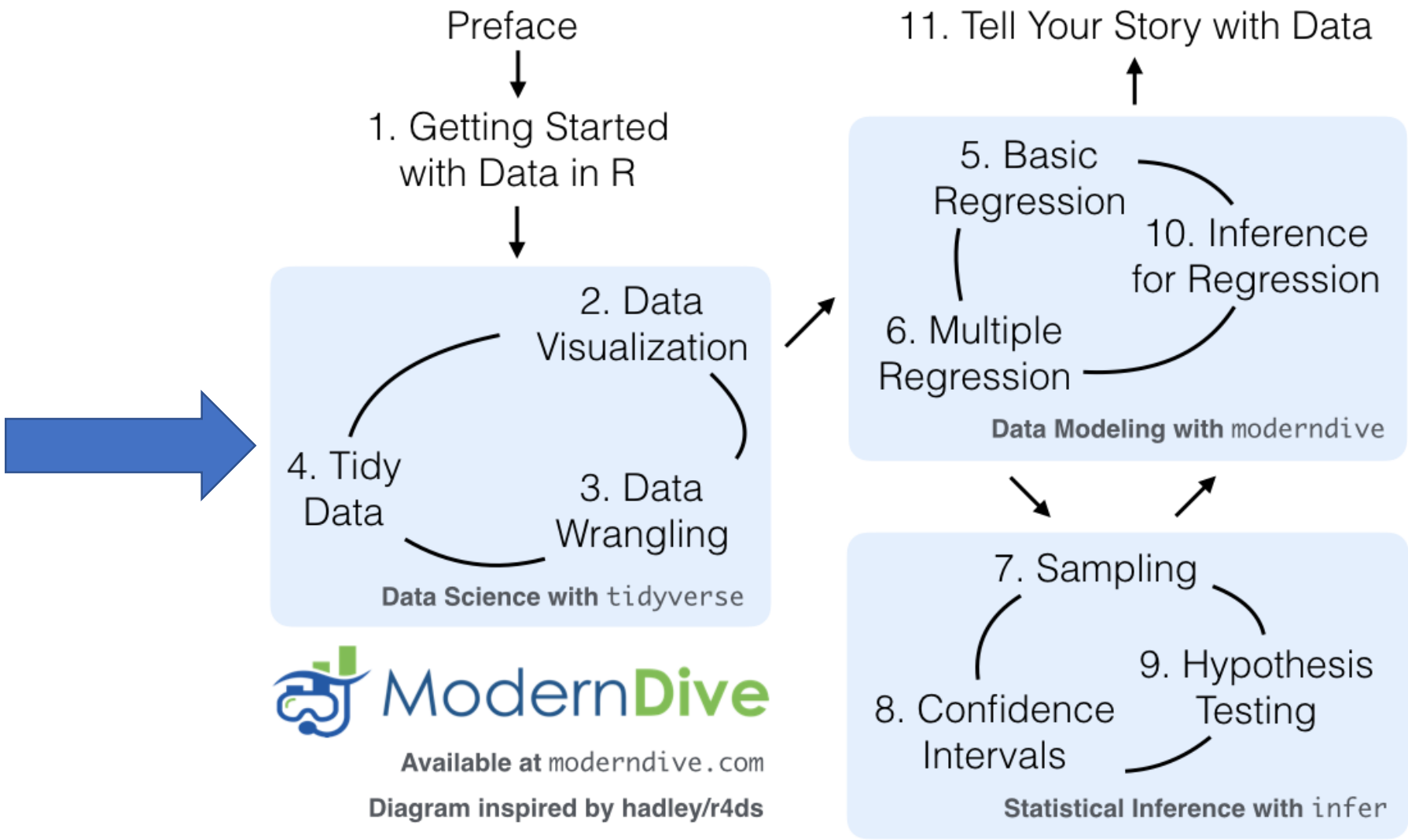
Use UTF-8 encoding



Start with a clean slate



Contents


Preface
1. Getting Started with Data in R
2. Data Visualization
3. Data Wrangling
4. Tidy Data
Data Science with tidyverse
5. Basic Regression
6. Multiple Regression
10. Inference for Regression
Data Modeling with moderndive
7. Sampling
8. Confidence Intervals
9. Hypothesis Testing
Statistical Inference with infer
11. Tell Your Story with Data
ModernDive
Available at moderndive.com
Diagram inspired by hadley/r4ds

I, ICE

Import Data

Inspect Data

Clean Data

Explore Data

London Business School

Import and Inspect data

Import Data

CSV data: ***mydata <- read_csv(datafile.csv)*** -- Requires readr package, https://readr.tidyverse.org/

Excel data: ***mydata <- read_excel(datafile.xls)*** -- Requires readxl package, https://readxl.tidyverse.org/

Fast import: use ***vroom::vroom()*** or ***datatable::fread()***

Inspect Data

- Look at the first seven rows and variables ***head(mydata, n = 7)***
- Look at the last seven rows and variables ***tail(mydata, n = 7)***
- Look at the variable (column) names ***colnames(mydata)***
- Look at all variables ***glimpse(mydata)***
- Get basic descriptives for each variable ***summary(mydata)***
- Skim variables and basic descriptives ***skimr::skim(mydata)***
- Look at your data in an Excel-like window ***View(mydata)***

London Business School

Manipulate data with *dplyr*

Examine specific variables
Sort data by values
View and create summary statistics
Create new, or remove some, variables from data

dplyr package, https://dplyr.tidyverse.org/

- `filter()`: pick rows
- `select()`: pick columns
- `arrange()`: order rows
- `mutate()`: add new or redefine existing columns
- `group_by()`: create partitions of rows and perform operations "by group"
- `summarise()`: aggregate across rows; reduces multiple values down to a single summary
- `*_join()`: merge tables
- `pivot_*()`: reshape tables

London Business School

The pipe operator— %>%

RStudio keyboard shortcut:

- Ctrl + Shift + M (Windows)
- Cmd + Shift + M (Mac).

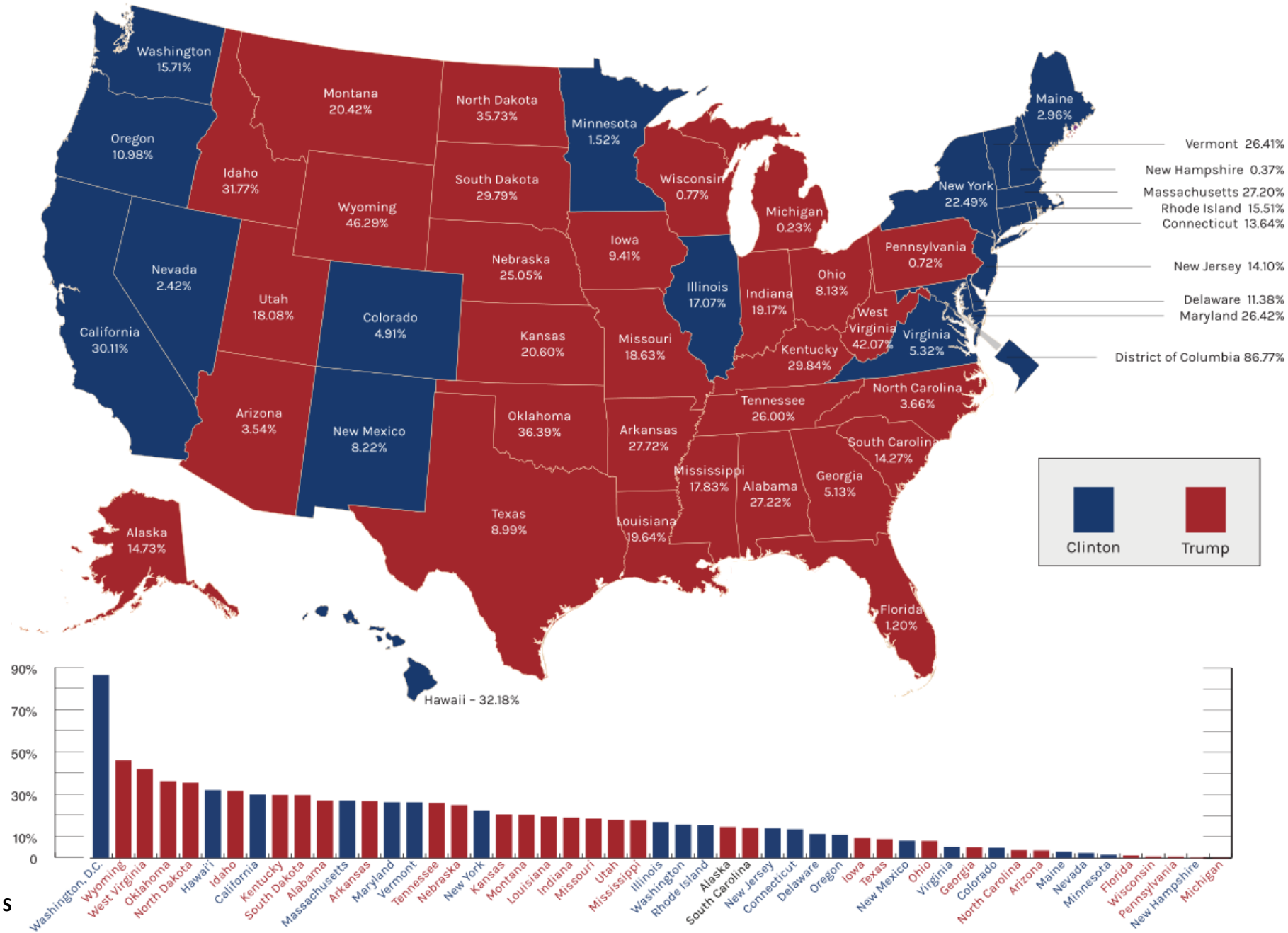
The pipe operator takes the thing on the left-hand-side ***AND THEN*** passes it to right-hand-side – literally, drops it in as the first argument.

Three ways to draw a sample 100 N(0,1) observations and calculate the interquartile range (IQR: difference between the 75th and 25th percentiles).

```
# 1. Save each intermediate step
draw <- rnorm(100)
end_points <- quantile(draw, probs = c(0.25, 0.75))
diff(end_points)

# 2. Nesting makes code difficult to read
diff(quantile(rnorm(100), probs = c(0.25, 0.75)))

# 3. use the pipe operator
rnorm(100) %>%
       quantile(probs = c(0.25, 0.75)) %>%
       diff()
```

London Business School

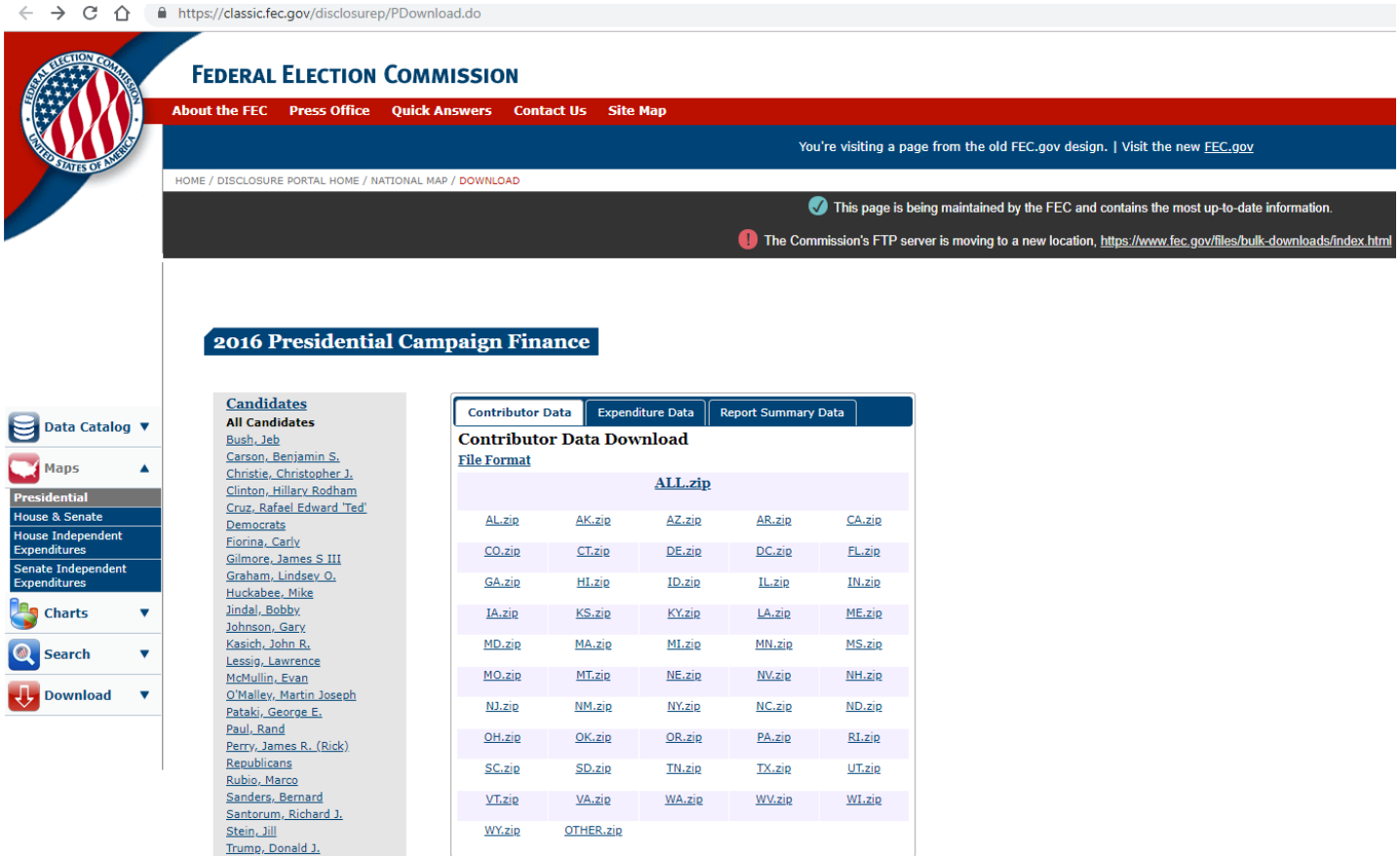
2016 Presidential Popular Vote Differential



S

Federal Election Commission

The Federal Election Commission (FEC) is an independent regulatory agency created by US Congress in 1975 to administer and enforce the Federal Elections Campaign Act. The FEC is responsible for disclosing campaign finance information, enforcing limits and prohibitions on contributions, and the overseeing the public funding of presidential elections. We will look at some data from California contributors in the 2016 Presidential Election



Source: https://classic.fec.gov/disclosurep/PDownload.do

session2_wrangling_CA_contributions.R

```
> glimpse(CA_contributors_2016)
Rows: 1,292,843
Columns: 4
$ cand_nm           <chr> "Clinton, Hillary Rodham", "Clinton, Hillary Rodham", "Clinton, Hillary Rodham", "Trump, Don
$ contb_receipt_amt <dbl> 50.00, 200.00, 5.00, 48.33, 40.00, 244.34, 35.00, 100.00, 25.00, 40.00, -40.00, -4.00, -4.00,
$ zip               <dbl> 94939, 93428, 92337, 95334, 93011, 95826, 90278, 90278, 92084, 92637, 92656, 92011, 92011, 9
$ contb_date        <date> 2016-04-26, 2016-04-20, 2016-04-02, 2016-11-21, 2016-03-04, 2016-11-24, 2016-03-05, 2016-03-
>
> CA_contributors_2016 %>%
+   select(contb_receipt_amt) %>%
+   skim()
-- Data Summary ------------------------
                           Values
Name                       Piped data
Number of rows             1292843
Number of columns          1
_______________________
Column type frequency:
  numeric                  1
_______________________
Group variables            None

-- Variable type: numeric ---------------------------------------------------------------------------------------
# A tibble: 1 x 11
  skim_variable     n_missing complete_rate  mean    sd    p0   p25   p50   p75  p100 hist
* <chr>                 <int>         <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
1 contb_receipt_amt         0             1  120.  389. -8300    15    27    94 10800 ___▇___
```

- Who made the largest donation?
- What was the average donation for a candidate, say Trump?
- What was the average donation for all candidates, ranked in descending order?
- How quickly did candidates raise money?
- Which cities raised the most money for the two candidates?

dplyr

dplyr rules for functions

1. First argument is always a data frame
2. Subsequent arguments, using the pipe operator **%>%**, say what to do with that data frame
3. Always return a data frame

- `select()` the columns (or variables) of interest
  - *select(cand_nm, contbr_nm, contbr_city, contbr_occupation, contb_receipt_amt)*
- `filter()` out irrelevant data to keep rows of interest
  - *filter(cand_nm == 'Trump, Donald J.')* to test equality, use **==** , not just one **=**

- `arrange()` the rows in a data set
  - *arrange(data, contb_receipt_amt)* by default, it sorts in increasing order
  - *arrange(data, **desc**(contb_receipt_amt))* to arrange in descending order, we add **desc**

- `group_by()`: create partitions of rows and perform operations "by group"
- `summarise()` the data (e.g., calculate mean, median, maximum, etc).
  - *summarise(data, avg_contribution=mean(contb_receipt_amt))*

- `mutate()` a data set by adding columns or redefining existing columns

Largest donation?

dplyr rules for functions

1. First argument is always a data frame
2. Subsequent arguments, using the pipe operator **%>%**, say what to do with that data frame
3. Always return a data frame

Verbs we use to describe how we manipulate the dataframe

- **select** the columns of interest
- **filter** out irrelevant data to keep rows of interest
- **mutate** a data set by adding more columns
- **arrange** the rows in a data set
- **summarise** the data (e.g., collapse values to a simple calculated mean, median, maximum, etc).

Largest donation?

```
> glimpse(CA_contributors_2016)
Observations: 1,292,843
Variables: 4
$ cand_nm           <chr> "Clinton, Hillary Rodham", "Clinton, Hillary Rodham", "Clinton, Hillary Rodham", "Trump, Donald...
$ contb_receipt_amt <dbl> 50.00, 200.00, 5.00, 48.33, 40.00, 244.34, 35.00, 100.00, 25.00, 40.00, -40.00, -4.00, -4.00, -...
$ zip               <dbl> 94939, 93428, 92337, 95334, 93011, 95826, 90278, 90278, 92084, 92637, 92656, 92011, 92011, 9201...
$ contb_date        <date> 2016-04-26, 2016-04-20, 2016-04-02, 2016-11-21, 2016-03-04, 2016-11-24, 2016-03-05, 2016-03-06...
```

Largest donation?

dplyr rules for functions

1. First argument is always a data frame
2. Subsequent arguments, using the pipe operator **%>%**, say what to do with that data frame
3. Always return a data frame

Verbs we use to describe how we manipulate the dataframe

- **select** the columns of interest
- **filter** out irrelevant data to keep rows of interest
- **mutate** a data set by adding more columns
- **arrange** the rows in a data set
- **summarise** the data (e.g., collapse values to a simple calculated mean, median, maximum, etc).

Largest donation?

```
# Highest Individual Contribution -----------
CA_contributors_2016 %>%
  arrange(desc(contb_receipt_amt)) %>%
  View()
```

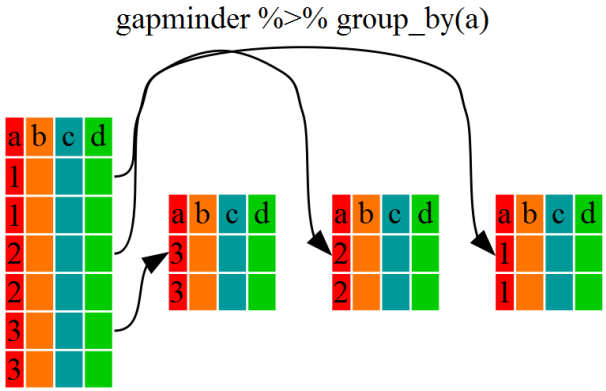
Average donation for Trump?
Average donation for all candidates?

dplyr rules for functions

1. First argument is always a data frame
2. Subsequent arguments, using the pipe operator **%>%**, say what to do with that data frame
3. Always return a data frame

Verbs we use to describe how we manipulate the dataframe

- **select** the columns of interest
- **filter** out irrelevant data to keep rows of interest
- **mutate** a data set by adding more columns
- **arrange** the rows in a data set
- **summarise** the data (e.g., collapse values to a simple calculated mean, median, maximum, etc).



London Business School

Average donation for Trump?
Average donation for all candidates?

dplyr rules for functions

1. First argument is always a data frame
2. Subsequent arguments, using the pipe operator **%>%**, say what to do with that data frame
3. Always return a data frame

Verbs we use to describe how we manipulate the dataframe

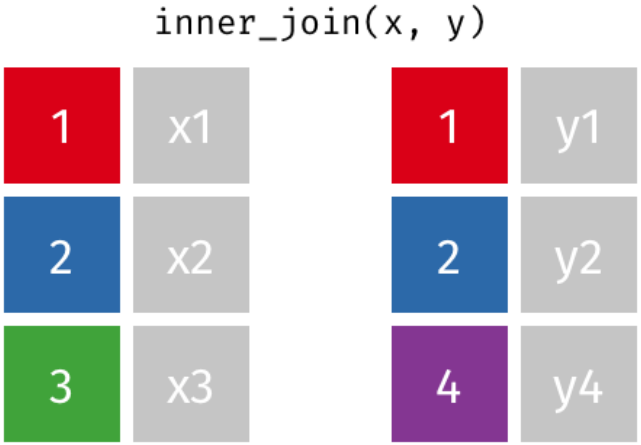
- **select** the columns of interest
- **filter** out irrelevant data to keep rows of interest
- **mutate** a data set by adding more columns
- **arrange** the rows in a data set
- **summarise** the data (e.g., collapse values to a simple calculated mean, median, maximum, etc).

Who raised the most amount of money in CA

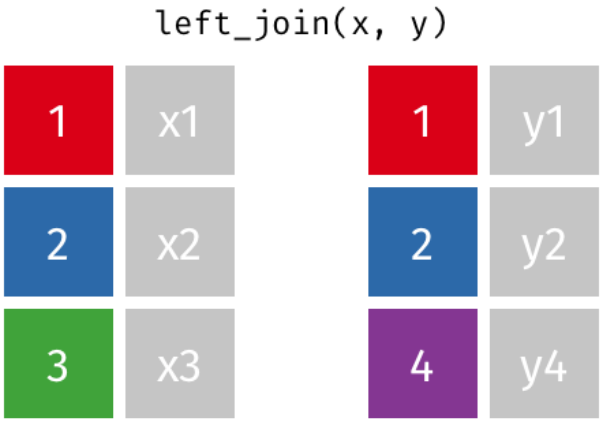
Joining dataframes (1/2)

A mutating join allows you to combine variables from two dataframes. It first matches observations by their keys, then copies across variables from one table to the other.

Inner Join
All rows from x where there are matching values in y, and all columns from x and y.



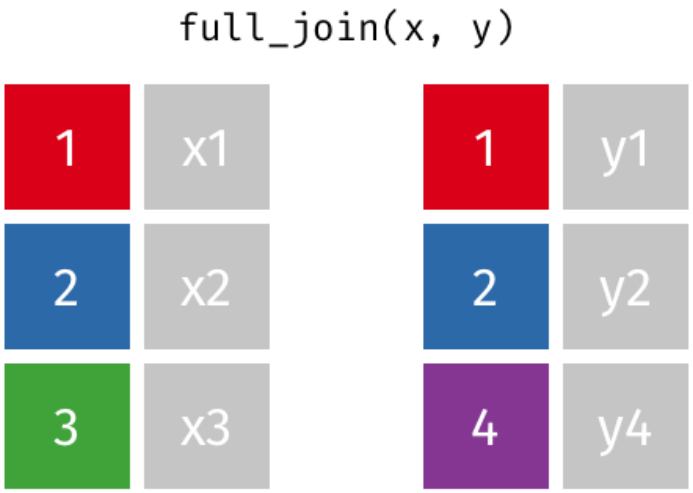
Left Join
All rows from x, and all columns from x and y. Rows in x with no match in y will have NA values in the new columns.



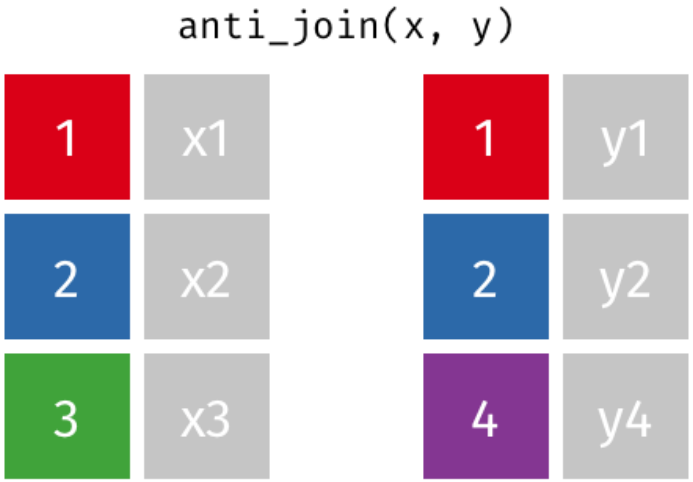
https://github.com/gadenbuie/tidyexplain#mutating-joins

Joining dataframes (2/2)

Full Join
All rows and all columns from both x and y. Where there are not matching values, returns NA for the one missing.

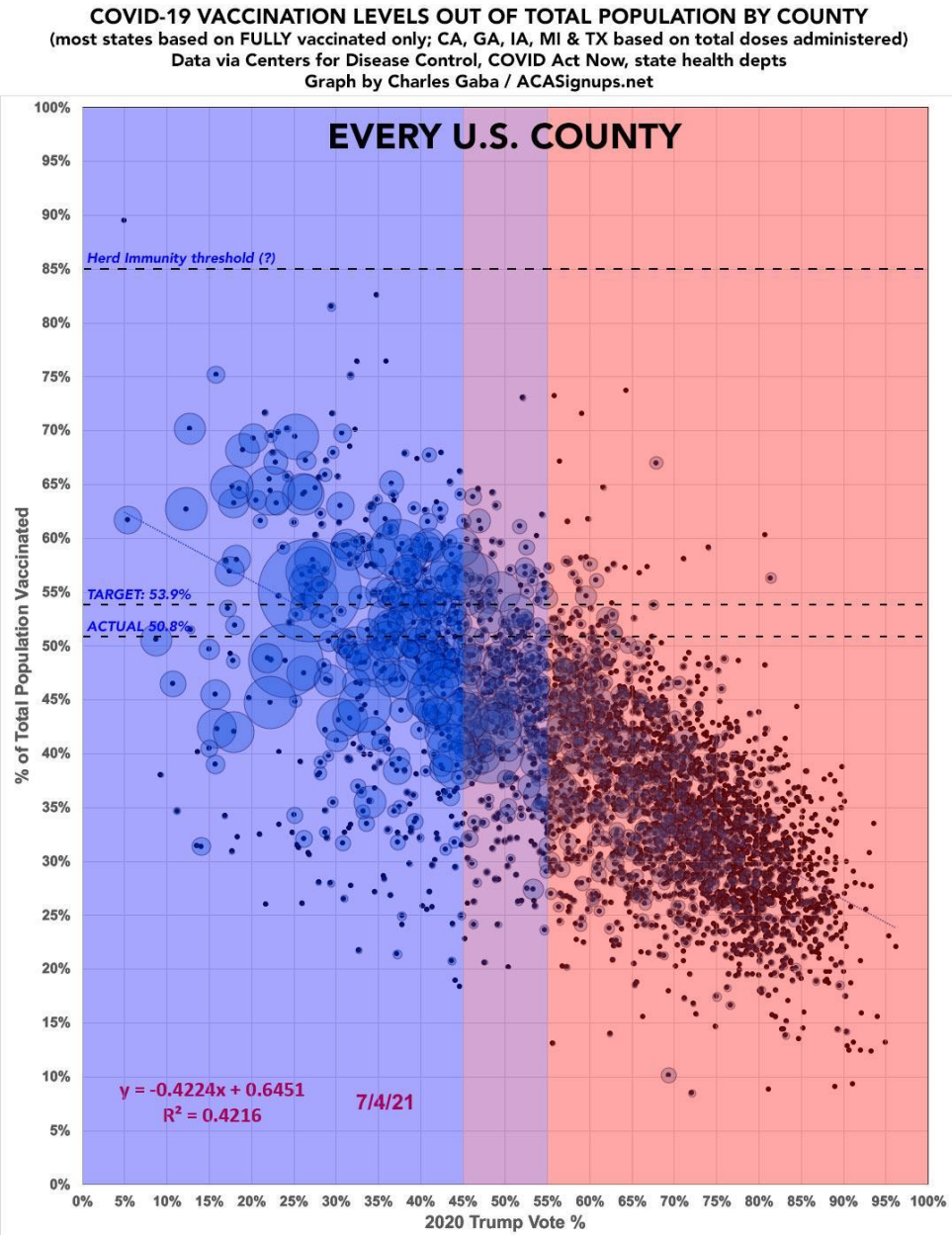


Anti-Join
All rows from x where there are not matching values in y, keeping just columns from x.



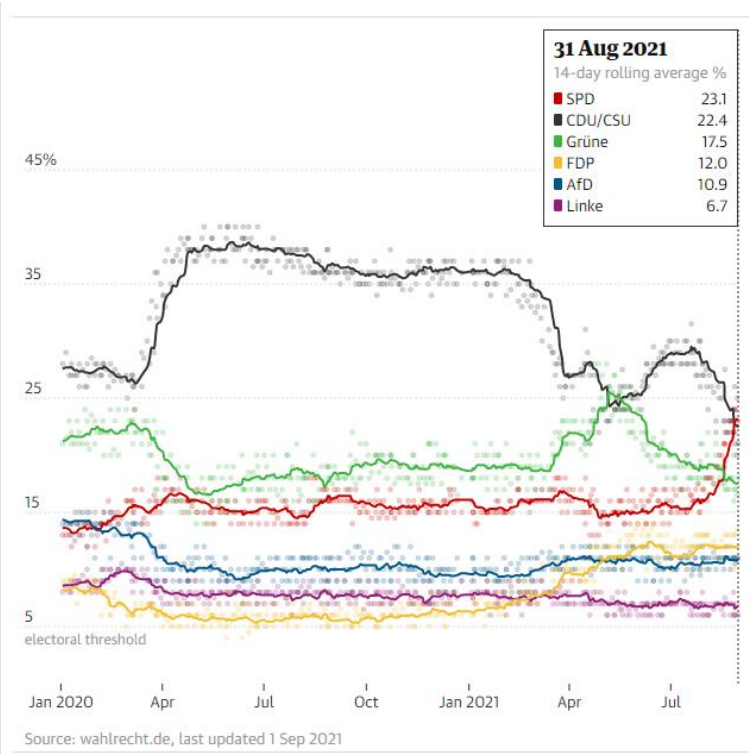
https://github.com/gadenbuie/tidyexplain#mutating-joins

Challenge 1



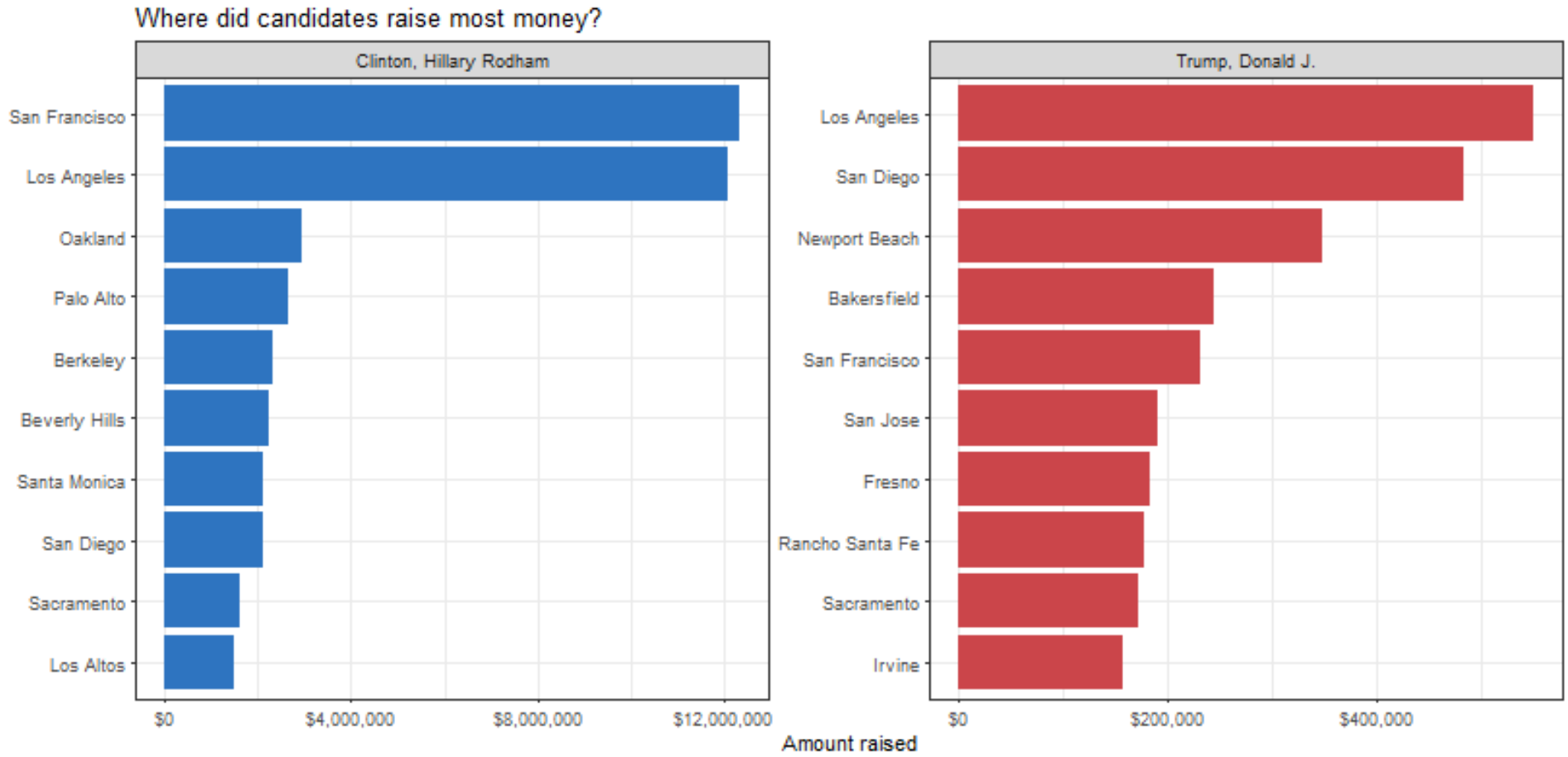

```{r, echo=FALSE, cache=TRUE}

# Download CDC vaccination by county
cdc_url <- "https://data.cdc.gov/api/views/8xkx-amqh/rows.csv?accessType=DOWNLOAD"
vaccinations <- vroom(cdc_url) %>%
  janitor::clean_names() %>%
  filter(fips != "UNK") # remove counties that have an unknown (UNK) FIPS code

# Download County Presidential Election Returns
# https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/VOQCHQ
election2020_results <- vroom(here::here("data", "countypres_2000-2020.csv")) %>%
  janitor::clean_names() %>%

  # just keep the results for the 2020 election
  filter(year == "2020") %>%

  # change original name county_fips to fips, to be consistent with the other two files
  rename (fips = county_fips)

# Download county population data
population_url <- "https://www.ers.usda.gov/webdocs/DataFiles/48747/PopulationEstimates.csv?v=2232"
population <- vroom(population_url) %>%
  janitor::clean_names() %>%

  # select the latest data, namely 2019
  select(fips = fip_stxt, pop_estimate_2019) %>%

  # pad FIPS codes with leading zeros, so they are always made up of 5 characters
  mutate(fips = stringi::stri_pad_left(fips, width=5, pad = "0"))
```

Challenge 2




```{r, scrape_wikipedia_polling_data, warnings= FALSE, message=FALSE}
url <- "https://en.wikipedia.org/wiki/Opinion_polling_for_the_2021_German_federal_election"
# https://www.economist.com/graphic-detail/who-will-succeed-angela-merkel
# https://www.theguardian.com/world/2021/jun/21/german-election-poll-tracker-who-will-be-the-next-chancellor


# get tables that exist on wikipedia page
tables <- url %>%
  read_html() %>%
  html_nodes(css="table")


# parse HTML tables into a dataframe called polls
# Use purr::map() to create a list of all tables in URL
polls <- map(tables, . %>%
             html_table(fill=TRUE)%>%
             janitor::clean_names())


# list of opinion polls
german_election_polls <- polls[[1]] %>% # the first table on the page contains the list of all opinions polls
  slice(2:(n()-1)) %>%  # drop the first row, as it contains again the variable names and last row that contains 2017 results
  mutate(
         # polls are shown to run from-to, e.g. 9-13 Aug 2021. We keep the last date, 13 Aug here, as the poll date
         # and we extract it by picking the last 11 characters from that field
         end_date = str_sub(fieldwork_date, -11),

         # end_date is still a string, so we convert it into a date object using lubridate::dmy()
         end_date = dmy(end_date),

         # we also get the month and week number from the date, if we want to do analysis by month- week, etc.
         month = month(end_date),
         week = isoweek(end_date)
         )
```

Challenge 3




```
> glimpse(CA_contributors_2016)
Observations: 1,292,843
Variables: 4
$ cand_nm           <chr> "Clinton, Hillary Rodham", "Clinton, Hillary Rodham", "Clinton, Hillary Rodham", "Trump, Donald...
$ contb_receipt_amt <dbl> 50.00, 200.00, 5.00, 48.33, 40.00, 244.34, 35.00, 100.00, 25.00, 40.00, -40.00, -4.00, -4.00, -...
$ zip               <dbl> 94939, 93428, 92337, 95334, 93011, 95826, 90278, 90278, 92084, 92637, 92656, 92011, 92011, 9201...
$ contb_date        <date> 2016-04-26, 2016-04-20, 2016-04-02, 2016-11-21, 2016-03-04, 2016-11-24, 2016-03-05, 2016-03-06...
```

Plotting in ***ggplot2***

Use this template to make graphs with ggplot2.
You may not even need to add scales and themes

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>)) +
  <SCALE_FUNCTION>() +
  <THEME_FUNCTION>() +
  labs(…) +
  facet_wrap()  or facet_grid() +
  NULL
```

London Business School

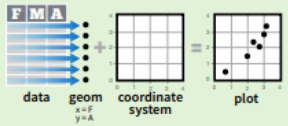
Plotting in ***ggplot2***

Data Visualization with ggplot2 Cheat Sheet

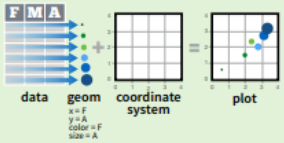
RStudio

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same few components: a **data** set, a set of **geoms**—visual marks that represent data points, and a **coordinate system.**



To display data values, map variables in the data set to aesthetic properties of the geom like **size**, **color**, and **x** and **y** locations.



Build a graph with **qplot()** or **ggplot()**

aesthetic mappings | data | geom

qplot(x = cty, y = hwy, color = cyl, data = mpg, geom = "point"**)**

Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

ggplot(data = mpg, **aes(**x = cty, y = hwy**))**

Begins a plot that you finish by adding layers to. No defaults, but provides more control than qplot().

data

```
ggplot(mpg, aes(hwy, cty)) +
 geom_point(aes(color = cyl)) +
 geom_smooth(method ="lm") +
 coord_cartesian() +
 scale_color_gradient() +
 theme_bw()
```

add layers, elements with +

layer = geom + default stat + layer specific mappings

additional elements

Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

One Variable

Continuous

a <- ggplot(mpg, aes(hwy))

a + **geom_area(stat = "bin")**
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")

a + **geom_density(**kernel = "gaussian"**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

a + **geom_dotplot()**
x, y, alpha, color, fill

a + **geom_freqpoly()**
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y = ..density..))

a + **geom_histogram(**binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(y = ..density..))

Discrete

b <- ggplot(mpg, aes(fl))

b + **geom_bar()**
x, alpha, color, fill, linetype, size, weight

Graphical Primitives

c <- ggplot(map, aes(long, lat))

c + **geom_polygon(**aes(group = group)**)**
x, y, alpha, color, fill, linetype, size

d <- ggplot(economics, aes(date, unemploy))

d + **geom_path(**lineend="butt", linejoin="round', linemitre=1**)**
x, y, alpha, color, linetype, size

d + **geom_ribbon(**aes(ymin=unemploy - 900, ymax=unemploy + 900**)**
x, ymax, ymin, alpha, color, fill, linetype, size

e <- ggplot(seals, aes(x = long, y = lat))

Two Variables

Continuous X, Continuous Y

f <- ggplot(mpg, aes(cty, hwy))

f + **geom_blank()**

f + **geom_jitter()**
x, y, alpha, color, fill, shape, size

f + **geom_point()**
x, y, alpha, color, fill, shape, size

f + **geom_quantile()**
x, y, alpha, color, linetype, size, weight

f + **geom_rug(**sides = "bl"**)**
alpha, color, linetype, size

f + **geom_smooth(**model = lm**)**
x, y, alpha, color, fill, linetype, size, weight

f + **geom_text(**aes(label = cty**)**
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))

g + **geom_bar(stat = "identity")**
x, y, alpha, color, fill, linetype, size, weight

g + **geom_boxplot()**
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight

g + **geom_dotplot(**binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill

g + **geom_violin(**scale = "area"**)**
x, y, alpha, color, fill, linetype, size, weight

Discrete X, Discrete Y

h <- ggplot(diamonds, aes(cut, color))

h + **geom_jitter()**

Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

i + **geom_bin2d(**binwidth = c(5, 0.5)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight

i + **geom_density2d()**
x, y, alpha, colour, linetype, size

i + **geom_hex()**
x, y, alpha, colour, fill size

Continuous Function

j <- ggplot(economics, aes(date, unemploy))

j + **geom_area()**
x, y, alpha, color, fill, linetype, size

j + **geom_line()**
x, y, alpha, color, linetype, size

j + **geom_step(**direction = "hv"**)**
x, y, alpha, color, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

k + **geom_crossbar(**fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, linetype, size

k + **geom_errorbar()**
x, ymax, ymin, alpha, color, linetype, size, width (also **geom_errorbarh()**)

k + **geom_linerange()**
x, ymin, ymax, alpha, color, linetype, size

k + **geom_pointrange()**
x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

l + **geom_map(**aes(map_id = state), map = map**) +**
expand_limits(x = mapSlong, y = mapSlat**)**

https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

Worked Examples: Live Coding