Data Analysis with R

Master the Tidyverse!

Before we start:

Download R:

- Go to https://cran.rstudio.com/ and then click on Windows or (Mac) OS X.
- *For Mac:* Click on and download the newest .pkg (under “Latest release” – R-3.5.2.pkg)
- *For Windows:* click the hyperlink that says: "install R for the first time." When redirected, click on “Download R 3.5.2 for Windows.”

Download RStudio:

- Go to https://www.rstudio.com/products/rstudio/download/
- Click RStudio Desktop: Open Source License (FREE)
- Under “Installers for Supported Platforms,” choose the platform you have (Windows or Mac)
- *For Windows:* RStudio 1.1.463 - Windows Vista/7/8/10
- *For Mac:* RStudio 1.1.463 - Mac OS X 10.6+ (64-bit)
- After following your machine’s installer instructions, double-click on the RStudio icon to open it.

Before we start:

If you aren't too comfortable with R, not to worry! We have TAs ready to give you more focused attention.

If you're planning to code alongside our presentation (recommended), please open RStudio and create a new R Markdown file. ("File" → "New File" → "R Markdown")
If it asks if you want to install the rmarkdown package, click "Yes."

The Tidyverse:

Most of the packages we'll be dealing with today belong to the Tidyverse, a package suite developed by Hadley Wickham, chief scientist at RStudio.

As the official website says: "All packages share an underlying design philosophy, grammar, and data structures." They make data analysis in R a faster, more cohesive process, and advocate a **tidy data** approach.

In tidy data:
1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

Installing and loading the Tidyverse:

Use the following to install and load the Tidyverse. These are the standard commands for installing and loading CRAN packages.

```
install.packages("tidyverse")
library(tidyverse)
```

`tidyverse_packages()` will give you a list of the packages you just imported. Of those, we'll be working with `readr`, `tidyr`, `dplyr`, `stringr` and `ggplot2` today.

readr:

`readr` is a package designed to read rectangular data. Some frequently used `read_` functions are:

- `read_csv()`: for files with comma-separated values.
- `read_tsv()`: tab-separated files
- `read_table()`: whitespace-separated files
- `read_delim()`: a file delimited by any other specified symbol

tidyr:

`tidyr` helps create tidy data. The two fundamental functions of `tidyr` are:

- `spread()`: makes “long” data “wide”
- `gather()`: makes “wide” data “long”

Other functions: `separate()`, `unite()`. Here’s the RStudio tidyr cheatsheet!

stringr:

The stringr package provide a cohesive set of functions designed to make working with strings as easy as possible.
All functions in stringr start with `str_` and take a vector of strings as the first argument.

Most string functions work with **regular expressions**, a concise language for describing patterns of text. For example, the regular expression "`[aeiou]`" matches any single character that is a vowel.

Some resources for working with strings in R:

- A stringr guide to regular expressions.
- RStudio's stringr cheatsheet.

dplyr:

`dplyr` is used for data wrangling. Its commands are usually verbs, and they solve various common data manipulation problems.
Some important `dplyr` commands:

- `mutate()`: create new variables that are functions of existing ones
- `select()`: picks certain variables (columns) based on names
- `filter()`: picks certain observations (rows) based on their values

Here is RStudio's `dplyr` cheatsheet. (It's the same as the `tidyr` one.)

`dplyr` also imports the pipe `%>%` from the `magrittr` package, which allows for clear, readable code by eliminating the need for nested functions.

ggplot2:

ggplot2 is quite hard to explain, but we're going to try.

Adapted from the ggplot2 website: "In most cases you start with ggplot(), supply a dataset and aesthetic mapping (*in which you define what goes on your axes, and what variables, if any, define color, fill, transparency and size, among other things*). You then add on layers (*this specifies the type of plot you want – like bar, scatter, histogram*), scales (*to modify elements you defined in aes()*), faceting specifications (*splitting your plot into multiple plots based on a variable – we'll show you*) and coordinate systems (*flipping coordinates and such*).

Since we don't have much time, here are some resources that will help you begin/get ahead on your ggplot2 journey:

- DataCamp's *Data Visualization with ggplot2* interactive course
- The data viz chapter in Hadley Wickham's *R for Data Science*
- RStudio's ggplot2 cheatsheet

Applying the Tidyverse: the 2017 grades dataset

```
grades_2017 <- read_csv("grades_2017.csv")
names(grades_2017)
```

Note: read the data description first!

```
 [1] "name"          "number"        "title"         "full_name"     "term_quarter"  "term_year"     "grade_Ap"
 [8] "grade_A"       "grade_Am"      "grade_Bp"      "grade_B"       "grade_Bm"      "grade_Cp"      "grade_C"
[15] "grade_Cm"      "grade_Dp"      "grade_D"       "grade_Dm"      "grade_F"       "grade_no_pass" "grade_pass"
```

We can use dplyr's mutate() to create a variable, 'total_students':

```
grades_2017 <- grades_2017 %>%
  mutate(total_students = rowSums(grades_2017[, 7:21]))
```

Creating a variable using `separate()`

A variable that contains **subject area** (e.g. just MATH or STATS) would enable some insightful comparisons. We can create this using `tidyr`'s `separate()`, the '`name`' variable and some regular expressions!

Per this link, class numbers can either be completely numeric, or prefixed by C, M, or CM to denote classes that are concurrent (credit awarded at both undergrad and grad level), multiple-listed (listed in two or more departments) or both. Our regular expression must account for this.

A good way to begin working with a function, to make sure you're using it right, is to **consult its documentation**. RStudio lets you do this within itself. Type '`?separate`' into the console to see the function's documentation. This will help you understand why we frame the **parameters** of the function the way we do on the next slide.

```r
grades_2017 <- grades_2017 %>%
  separate(name, c("subject_area", NA),
           sep = "(M|C|CM|)[0-9]", remove = FALSE)
# the regex says "(M or C or CM or <nothing>) and [any digit (from 0-9, so any digit)]"
```

There may be a few classes numbered slightly differently, meaning the '`subject_area`' variable will be the same as the '`name`' variable (since no part of their name would have matched the regular expression). Let's check, using `dplyr`'s `filter()` function:

```r
grades_2017 %>% filter(subject_area == name)
# only one class fails to meet numbering conventions; let's take it out
grades_2017 <- grades_2017 %>% filter(subject_area != name)
# this command allows us to keep all the other rows.

grades_2017$subject_area <- trimws(grades_2017$subject_area)
# handy base function to trim white space
```

Our '`subject_area`' variable is now ready!

Using dplyr's `group_by()` and `summarize()`

We want to find the subject areas with the largest average class sizes, and visualize the results.

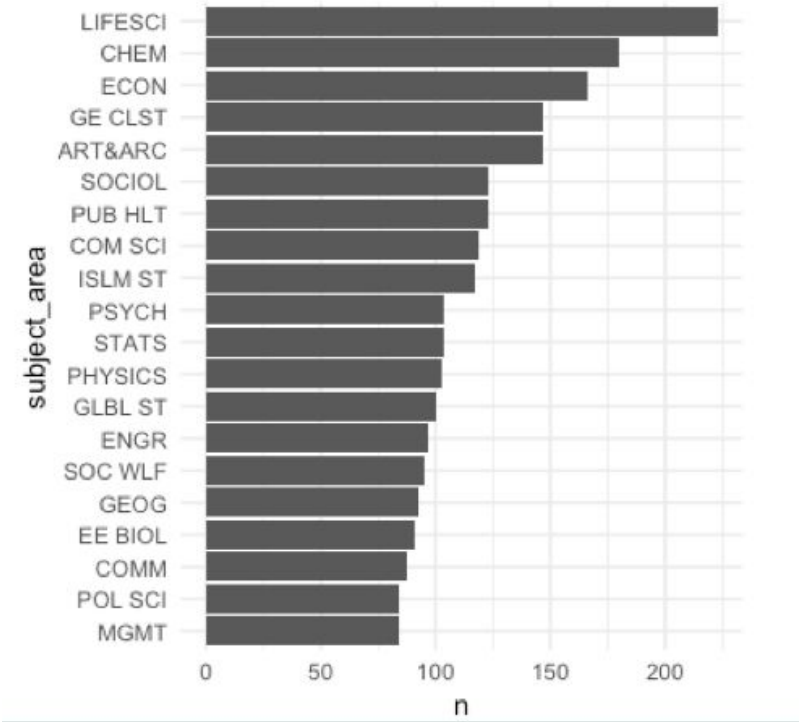
```
largest_classes_subj <- grades_2017 %>%
  group_by(subject_area) %>%
  summarise(n = mean(total_students)) %>%
  # gives us the sum of class enrollments
  # grouped by subject area.
  # replacing 'sum' with 'mean' within summarise
  # gives mean enrollments by subject area
  arrange(desc(n)) %>%
  # arrange in descending order of n
  top_n(20)
  # select top 20 for further analysis
```

produces

	subject_area	n
1	LIFESCI	222.58974
2	CHEM	179.71560
3	ECON	166.51190
4	GE CLST	146.63918
5	ART&ARC	146.50000
6	SOCIOL	123.21429
7	PUB HLT	123.00000
8	COM SCI	119.25000
9	ISLM ST	117.00000
10	PSYCH	104.01852
11	STATS	103.91379
12	PHYSICS	102.79808
13	GLBL ST	100.50000
14	ENGR	96.76471
15	SOC WLF	95.25000
16	GEOG	92.33333
17	EE BIOL	91.16901
18	COMM	87.12903
19	POL SCI	84.44615
20	MGMT	83.85938

Using ggplot2 to plot our findings:

```
largest_classes_subj %>%
  # DATASET
  ggplot(aes(x = subject_area, y = n)) +
  # ggplot() + AESTHETIC MAPPING (only axes in this case)
  geom_bar(stat = "identity") +
  # LAYER (barplot)
  # stat = 'identity' tells ggplot not to summarize data.
  scale_x_discrete(limits =
                     rev(largest_classes_subj$subject_area)) +
  # SCALES (orders plot from least to most)
  coord_flip() +
  # COORDINATE SYSTEMS (flipping coordinates)
  theme_minimal()
  # changing the plot's look
```

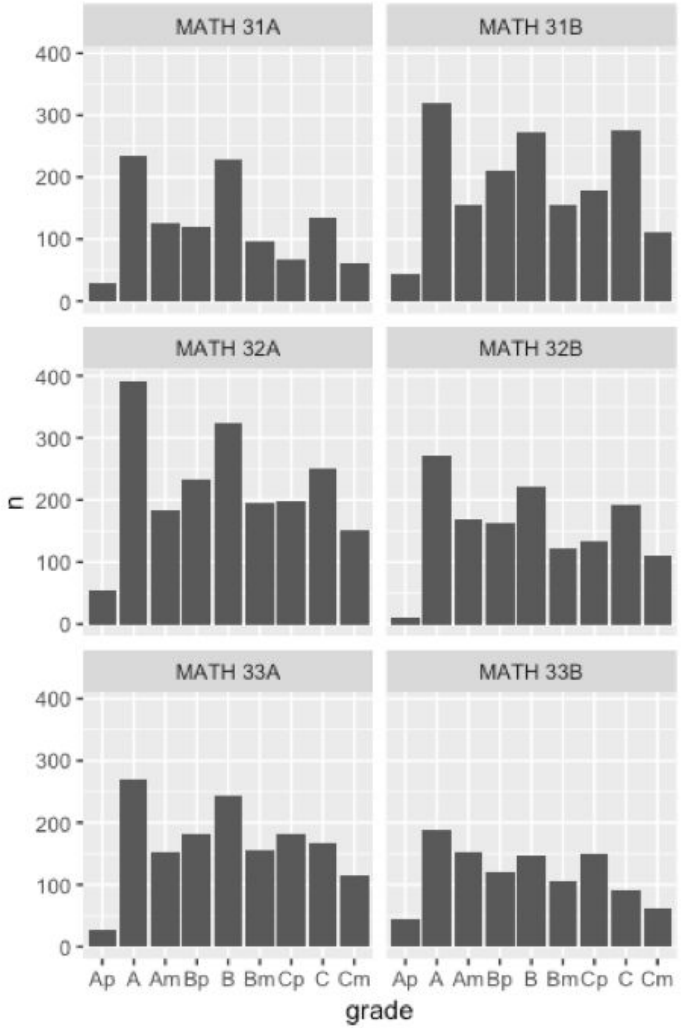


Grade distributions in Math lower-div classes:

```
math_lowerdivs <- which(str_detect(grades_2017$name, "MATH 3[123][AB]"))
# use str_detect() to detect the presence of the regex in a row
# wrap the which function around it to get the indices of those rows
# the regex says "MATH 3[either 1, 2 or 3][either A or B]"
math_lowerdivs <- grades_2017[math_lowerdivs, ]
# select only those rows
math_lowerdivs$name <- str_sub(math_lowerdivs$name, start = 1, end = 8)
# use str_sub() to obtain substrings to omit whether a class is honors
math_lowerdivs <- math_lowerdivs %>%
  gather("grade", "num_students",
         grade_Ap:grade_Cm)
# use tidyr's gather to make the wide data long
math_lowerdivs <- math_lowerdivs %>%
  group_by(name, grade) %>%
  summarise(n = sum(num_students))
# group by both name and grade to find
# the number of students per grade per class

math_lowerdivs$grade <- str_sub(math_lowerdivs$grade, start = 7)
# omit "grade_" from each grade level to just get A, Ap, Am, etc.
```

Grade distributions plotted:

```
math_lowerdivs %>%
  ggplot(aes(x = grade, y = n)) +
  # ggplot() + AESTHETIC MAPPING (only axes)
  geom_bar(stat = "identity") +
  # LAYER (barplot)
  # stat = 'identity' tells ggplot not to summarize data.
  scale_x_discrete(limits =
    c("Ap", "A", "Am", "Bp", "B", "Bm", "Cp", "C", "Cm")) +
  # SCALES (orders plot as desired)
  facet_wrap(~name, nrow = 3)
  # FACET (splits plot based on a variable)
```



Some more ggplot2 with the cereal dataset:

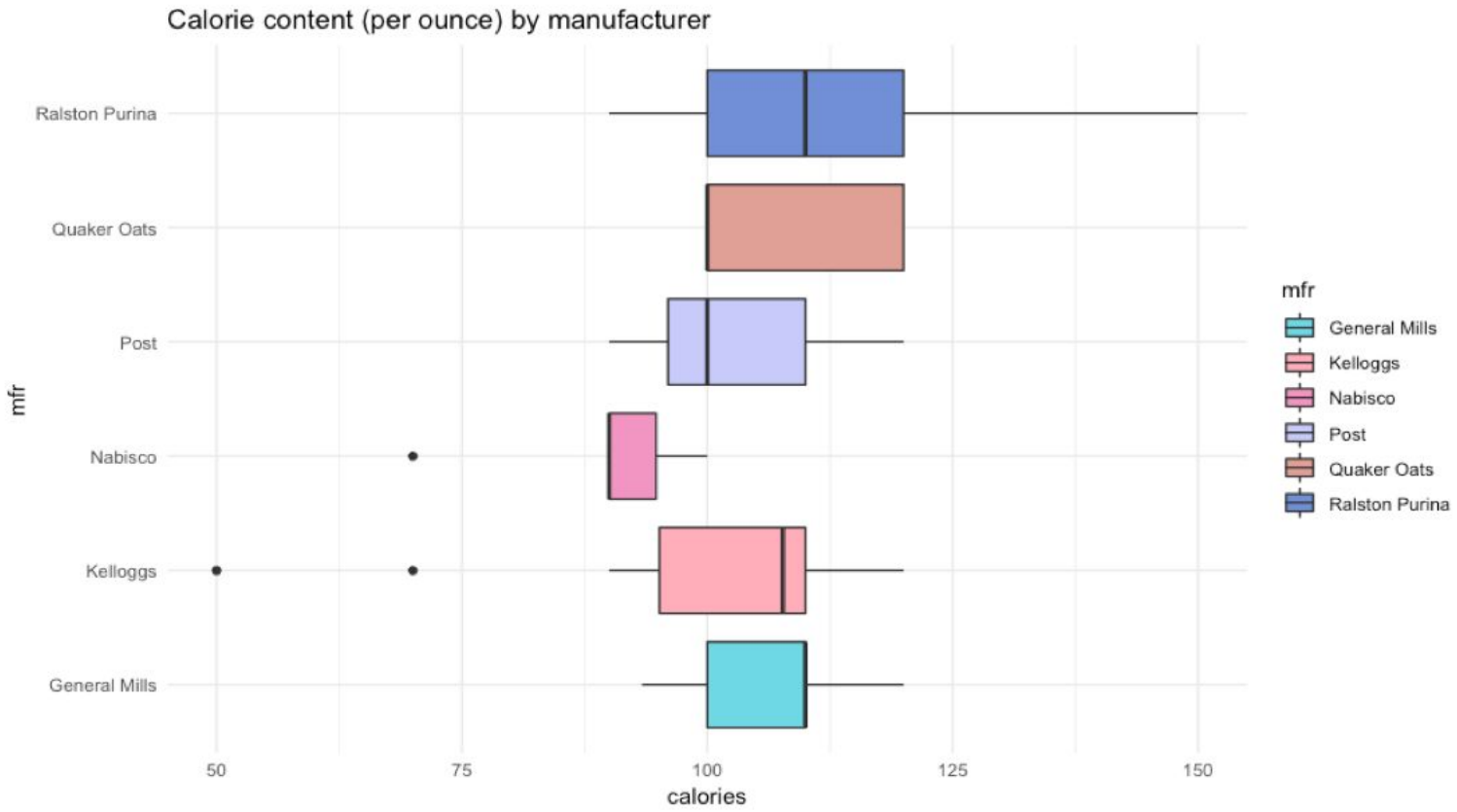
But first, let's clean it up a bit.

```
cereal <- read_csv("cereal.csv")
cereal %>%
  group_by(mfr) %>%
  count()
# these variables are pretty meaningless the way they're named.
# let's use str_replace_all() to rename.

# there are some -1s; not a valid value and will mess with calculations
# replace them with NA, using dplyr's na_if()
cereal <- na_if(cereal, -1)

cereal$mfr <- str_replace_all(cereal$mfr, "G", "General Mills")
cereal$mfr <- str_replace_all(cereal$mfr, "K", "Kelloggs")
cereal$mfr <- str_replace_all(cereal$mfr, "N", "Nabisco")
cereal$mfr <- str_replace_all(cereal$mfr, "P", "Post")
cereal$mfr <- str_replace_all(cereal$mfr, "Q", "Quaker Oats")
cereal$mfr <- str_replace_all(cereal$mfr, "R", "Ralston Purina")
```

A boxplot:

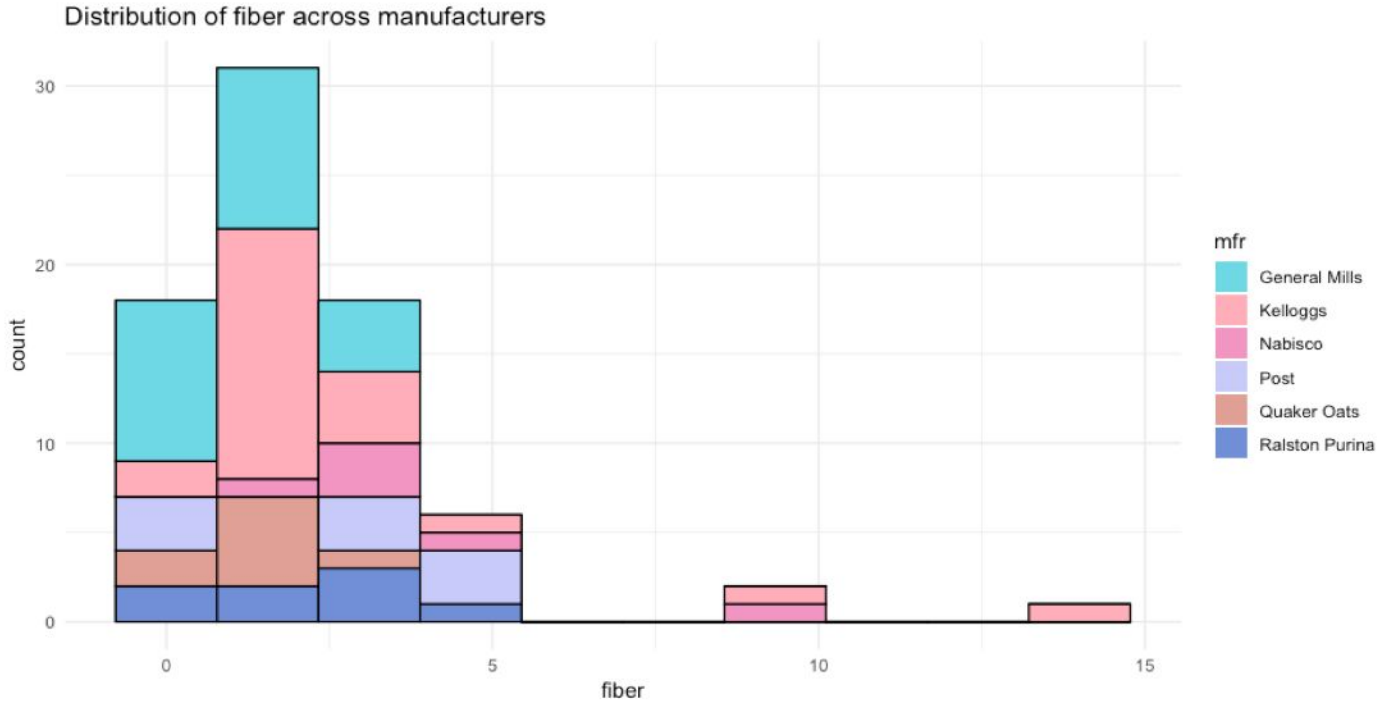
```
personal_palette <- c("#85D4E3", "#F4B5BD", "#E6A0C4",
                      "#C6CDF7", "#D8A499", "#7294D4")
# colors taken from the wesanderson R color palette package

cereal %>%
  ggplot(aes(x = mfr, y = calories, fill = mfr)) +
  geom_boxplot() +
  scale_fill_manual(values = personal_palette) +
  coord_flip() +
  theme_minimal() +
  ggtitle("Calorie content (per 100g) by manufacturer")
  # ggtitle allows you title and subtitle graphs
```

A boxplot:


Calorie content (per ounce) by manufacturer
Ralston Purina
Quaker Oats
Post
Nabisco
Kelloggs
General Mills
mfr
50
75
100
125
150
calories
mfr
General Mills
Kelloggs
Nabisco
Post
Quaker Oats
Ralston Purina

A histogram:

```
cereal %>% ggplot(aes(x = fiber, fill = mfr, color = I("black"))) + geom_histogram(bins = 10) +
  scale_fill_manual(values = personal_palette) + theme_minimal() +
  ggtitle("Distribution of fiber across manufacturers") # don't worry about I("black")
```



Presentation sources:

- This presentation draws a lot on information from the Tidyverse's website, as well as each individual package's websites.
- The grade data used was provided to the Bruinwalk dev team at the Daily Bruin by the Registrar's Office.
- The cereal data was adapted from the "80 cereals" dataset on Kaggle.