**Study Guide Unit 2 Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
DATA 110 Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Saidi**

**Handling Data**

## Where to get data

## Reproducibility

## Handling data

## R packages

## Filter, Sort, Group, Summarize

## Work with dates

## Join datasets

## Make heatmap and treemap

## What is data?

Before we leap into making charts and maps, we will consider the nature of data, and some basic principles that will help you to “interview” datasets to find and tell stories. We will review some fundamental statistical concept as well.

Data visualization and statistics provide a view of the world that we cannot otherwise obtain. They give us a framework to make sense of daunting and otherwise meaningless masses of information. The “lies” that data and graphics can tell arise when people misuse statistics and visualization methods, not when they are used correctly.

The best data journalists understand that statistics and graphics go hand-in-hand. Just as numbers can be made to lie, graphics may misinform if the designer is ignorant of or abuses basic statistical principles. You do not have to be an expert statistician to make effective charts and maps, but understanding some basic principles will help you to tell a convincing and compelling story — enlightening rather than misleading your audience.

I hope you will get hooked on the power of a statistical way of thinking. As data artist [Martin Wattenberg](http://paldhous.github.io/www.bewitched.com/) of Google has said: [“Visualization is a gateway drug to statistics.”](http://www.newscientist.com/blogs/culturelab/2011/02/data-artists-visualisation-as-a-gateway-drug.html) Source: Peter Aldhous

## Where to get data? Here are some potential sources:

1. <https://ourworldindata.org/>
2. Open Data Network Through Socrata:  <https://dev.socrata.com/data/>
3. <https://www.kaggle.com/datasets>
4. <https://www.cdc.gov/>
5. <https://www.data.gov/>
6. [http://data.un.org](http://data.un.org/)
7. [World Bank Database](https://data.worldbank.org/indicator)  (https://data.worldbank.org/indicator)
8. <https://www.ipums.org/>
9. <https://knoema.com/atlas>
10. Open Data from Montgomery County: <https://data.montgomerycountymd.gov/>
11. https://www.icpsr.umich.edu/icpsrweb/content/about/thematic-collections.html
12. [R Datasets](http://vincentarelbundock.github.io/Rdatasets/datasets.html)   (You can browse those sets with this guide:  [R Data Guide](https://bb-montgomerycollege.blackboard.com/bbcswebdav/pid-4295118-dt-content-rid-25785804_1/xid-25785804_1))
13. WHO data: <http://www.who.int/>
14. <https://data.mendeley.com/>
15. Here is an entire page of hundreds of datasets, categorized in useful ways. Documents with descriptions of the datasets are included as well.    
    <http://vincentarelbundock.github.io/Rdatasets/datasets.html>
16. Baltimore City has made its government data open: <https://data.baltimorecity.gov/>

And many more sites….. Some sites are better than others for accessing data on geography, sports, government and politics.

## R Basics (an introduction to R from Peter Aldhous)

### The data we will use today

Download the data for this session and save it to your datasets folder. It contains the following files, used in reporting [this story](https://www.newscientist.com/article/dn18806-revealed-pfizers-payments-to-censured-doctors/), which revealed that some of the doctors paid as “experts” by the drug company Pfizer had troubling disciplinary records:

* pfizer.csv Payments made by Pfizer to doctors across the United States in the second half on 2009. Contains the following variables:
  + org\_indiv Full name of the doctor, or their organization.
  + first\_plus Doctor’s first and middle names.
  + first\_name last\_name. First and last names.
  + city state City and state.
  + category of payment Type of payment, which include Expert-led Forums, in which doctors lecture their peers on using Pfizer’s drugs, and `Professional Advising.
  + cash Value of payments made in cash.
  + other Value of payments made in-kind, for example puschase of meals.
  + total value of payment, whether cash or in-kind.
* fda.csv Data on warning letters sent to doctors by the U.S. Food and Drug Administration, because of problems in the way in which they ran clinical trials testing experimental treatments. Contains the following variables:
  + name\_last name\_first name\_middle Doctor’s last, first, and middle names.
  + issued Date letter was sent.
  + office Office within the FDA that sent the letter.

[Rpubs document](http://rpubs.com/rsaidi/498266)

Unit 2 Handling Data

*Rachel Saidi*

*May 21, 2019*

Reproducibility: Save your scripts

Data journalism should ideally be fully documented and reproducible. R makes this easy, as every operation performed can be saved in a script, and repeated by running that script.

Any code we type in here can be run in the console. Hitting Run will run the line of code on which the cursor is sitting. To run multiple lines of code, highlight them and click Run.

Set your working directory

Set the working directory to this folder by selecting from the top menu Session>Set Working Directory>Choose Directory. Then select the folder where you are keeping all your datasets for this class. By doing this, we can load the files in this directory without having to refer to the full path for their location, and anything we save will be written to this folder.

You can always check what directory you are in with the code: getwd()

Once you set your working directory for where you will access your stored data, load the “tidyverse” package, then read in the data. More information on these processes will follow:

getwd()

## [1] "C:/Users/rsaidi/Dropbox/Rachel/MontColl/DATA110/Notes"

setwd("C:/Users/rsaidi/Dropbox/Rachel/MontColl/Datasets/Datasets")

**library**(tidyverse)

## -- Attaching packages ------------------------------------------- tidyverse 1.2.1 --

## v ggplot2 3.1.0 v purrr 0.3.0

## v tibble 2.0.1 v dplyr 0.8.0.1

## v tidyr 0.8.3 v stringr 1.4.0

## v readr 1.3.1 v forcats 0.4.0

## -- Conflicts ---------------------------------------------- tidyverse\_conflicts() --

## x dplyr::filter() masks stats::filter()

## x dplyr::lag() masks stats::lag()

**library**(psych)

##

## Attaching package: 'psych'

## The following objects are masked from 'package:ggplot2':

##

## %+%, alpha

pfizer <- read\_csv("pfizer.csv")

## Parsed with column specification:

## cols(

## org\_indiv = col\_character(),

## first\_plus = col\_character(),

## first\_name = col\_character(),

## last\_name = col\_character(),

## city = col\_character(),

## state = col\_character(),

## category = col\_character(),

## cash = col\_double(),

## other = col\_double(),

## total = col\_double()

## )

fda <- read\_csv("fda.csv")

## Parsed with column specification:

## cols(

## name\_last = col\_character(),

## name\_first = col\_character(),

## name\_middle = col\_character(),

## issued = col\_character(),

## office = col\_character()

## )

Notice that the Environment now contains two objects, of the type tbl\_df, a variety of the standard R object for holding tables of data, known as a data frame.

Save your script

As with any working document, be sure to save your script continually.

Comment your code

Anything that appears on a line after # will be treated as a comment, and will be ignored when the code is run. Get into the habit of commenting your code: Don’t trust yourself to remember what it does!

R packages

Much of the power of R comes from the thousands of “packages” written by its community of open source contributors. These are optimized for specific statistical, graphical or data-processing tasks. To see what packages are available in the basic distribution of R, select the Packages tab in the panel at bottom right. To find packages for particular tasks, try searching Google using appropriate keywords and the phrase “R package.”

In this unit, we will work with two incredibly useful packages developed by Hadley Wickham, chief scientist at RStudio:

. **readr** For reading and writes csv and other text files.

. **dplyr** For processing and manipulating data.

These and several other useful packages have been combined into a super-package called **tidyverse**, which you learned to install and load in the last unit.

Each time you start R, it’s a good idea to click on **Update** in the Packages panel to update all your installed packages to the latest versions. Installing a package makes it available to you, but to use it in any R session you need to load it. You can do this by checking its box in the Packages tab. However, we will enter the following code into our script, then highlight these lines of code and run them:

## What is Piping Anyway?

## The Pipe Operator in R: Introduction (From Data Camp)

To understand what the pipe operator in R is and what you can do with it, it's necessary to learn the history behind it. Where does this weird combination of symbols come from? And why was it made like this? might be on top of your mind.   
Now, you can look at the history from three perspectives: from a mathematical point of view, from a holistic point of view of programming languages, and from the point of view of the R language itself. You'll cover all three in what follows!

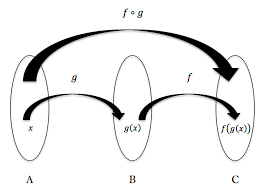
### History of the Pipe Operator in R

#### Mathematical History

If you have two functions, f: B→C and  g: A→B you can chain these functions together by taking the output of one function and inserting it into the next. In short, "chaining" means that you pass an intermediate result onto the next function.

For example, you can say, f(g(x)): g(x) serves as an input for f(), while x serves as input to g(). This is often called a composite function, where f is ***composed*** of g.

If you would want to note this down, you will use the notation f◦g, which reads as "f follows g", or “f of g”. Alternatively, you can visually represent this as:



#### Pipes in R

The history of this operator in R starts in 2012 when [Hadley Wickham](http://hadley.nz/) started the dplyr package on GitHub, which is based off of F# (pronounced F Sharp, as in Visual F# Programming Language, which is an open source, cross platform compiler, which can generate JavaScript and graphics processing unit (GPU) code. The question started:

*How can you implement F#'s forward pipe operator in R? The operator makes it possible to easily chain a sequence of calculations. For example, when you have an input data and want to call functions foo and bar in sequence, you can write data |> foo |> bar?*

"%>%" <- function(x,f) do.call(f,list(x))  
pi %>% sin  
[1] 1.224606e-16  
pi %>% sin %>% cos  
[1] 1  
cos(sin(pi))  
[1] 1

#### What is a Pipe?

In R, the pipe operator is, as you have already seen, %>%. You can think of this operator as being similar to the + in a ggplot2 statement. It takes the output of one statement and makes it the input of the next statement. When describing it, you can think of it as a "THEN".

This is one of the most powerful things about the Tidyverse. In fact, having a standardized chain of processing actions is called "a pipeline". Making pipelines for a data format is great, because you can apply that pipeline to incoming data that has the same formatting and have it output in a ggplot2 friendly format, for example.

### Why Use It?

R is a functional language, which means that your code often contains a lot of parenthesis, ( and ). When you have complex code, this often will mean that you will have to nest those parentheses together. This makes your R code hard to read and understand. Here's where %>% comes in to the rescue!

Take a look at the following example, which is a typical example of nested code:

# Initialize `x`  
x <- c(0.109, 0.359, 0.63, 0.996, 0.515, 0.142, 0.017, 0.829, 0.907)  
#Compute the logarithm of `x`, return suitably lagged and iterated differences, compute the exponential function and round the result  
round(exp(diff(log(x))), 1)

With the help of %>%, you can rewrite the above code as follows:

# Import `magrittr`  
library(magrittr)  
# Perform the same computations on `x` as above  
x %>% log() %>%  
 diff() %>%  
 exp() %>%  
 round(1)

In short, here are four reasons why you should be using pipes in R:

* You'll structure the sequence of your data operations from left to right, as opposed to from inside and out;
* You'll avoid nested function calls;
* You'll minimize the need for local variables and function definitions; And
* You'll make it easy to add steps anywhere in the sequence of operations.

### Additional Pipes

Even though %>% is the (main) pipe operator of the magrittr package, there are other operators that you should know and that are part of the same package:

## The compound assignment operator %<>%;

# Initialize `x`   
x <- rnorm(100)  
# Update value of `x` and assign it to `x`  
x %<>% abs %>% sort

## How to Use Pipes in R

Now you know how the %>% operator originated, what it actually is and why you should use it. It is time for you to discover how you can actually use it to your advantage. You will see that there are quite some ways in which you can use it!

### Basic Piping

Before you go into the more advanced usages of the operator, it's good to first take a look at the most basic examples that use the operator. In essence, you'll see that there are 3 rules that you can follow when you're first starting out:

* f(x) can be rewritten as x %>% f

In short, this means that functions that take one argument, function(argument), can be rewritten as follows: argument %>% function(). Take a look at the following, more practical example to understand how these two are equivalent:

# Compute the logarithm of `x`   
x %>% log()

### Compound Assignment Pipe Operations

There are situations where you want to overwrite the value of the left-hand side, just like in the example right below. Intuitively, you will use the assignment operator <- to do this.  
  
# Load in the Iris data

iris <- read.csv(url("http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"), header = FALSE)

# Add column names to the Iris data

names(iris) <- c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width", "Species")

# Compute the square root of `iris$Sepal.Length` and assign it to the variable

iris$Sepal.Length <-   
 iris$Sepal.Length %>%  
 sqrt()

However, there is a compound assignment pipe operator, which allows you to use a shorthand notation to assign the result of your pipeline immediately to the left-hand side:

# Compute the square root of `iris$Sepal.Length` and assign it to the variable  
iris$Sepal.Length %<>% sqrt

# Return `Sepal.Length`  
iris$Sepal.Length

**Note** that the compound assignment operator %<>% needs to be the first pipe operator in the chain for this to work. This is completely in line with what you just read about the operator being a shorthand notation for a longer notation with repetition, where you use the regular <- assignment operator.

As a result, this operator will assign a result of a pipeline rather than returning it.

Manipulate the pfizer and fda data

Recall that the pfizer data contains information about Pfizer payments to doctors and warning letters sent by food and drug administration

Examine the data

We can **View** data at any time by clicking on its table icon in the Environment tab in the **Grid view.**

Alternatively, you can use code we learned in the last unit. Click on the console output in each square that comes up after your chunk of code.

str(pfizer)

## Classes 'spec\_tbl\_df', 'tbl\_df', 'tbl' and 'data.frame': 10087 obs. of 10 variables:

## $ org\_indiv : chr "3-D MEDICAL SERVICES LLC" "AA DOCTORS, INC." "ABBO, LILIAN MARGARITA" "ABBO, LILIAN MARGARITA" ...

## $ first\_plus: chr "STEVEN BRUCE" "AAKASH MOHAN" "LILIAN MARGARITA" "LILIAN MARGARITA" ...

## $ first\_name: chr "STEVEN" "AAKASH" "LILIAN" "LILIAN" ...

## $ last\_name : chr "DEITELZWEIG" "AHUJA" "ABBO" "ABBO" ...

## $ city : chr "NEW ORLEANS" "PASO ROBLES" "MIAMI" "MIAMI" ...

## $ state : chr "LA" "CA" "FL" "FL" ...

## $ category : chr "Professional Advising" "Expert-Led Forums" "Business Related Travel" "Meals" ...

## $ cash : num 2625 1000 0 0 1800 ...

## $ other : num 0 0 448 119 0 0 47 0 0 396 ...

## $ total : num 2625 1000 448 119 1800 ...

## - attr(\*, "spec")=

## .. cols(

## .. org\_indiv = col\_character(),

## .. first\_plus = col\_character(),

## .. first\_name = col\_character(),

## .. last\_name = col\_character(),

## .. city = col\_character(),

## .. state = col\_character(),

## .. category = col\_character(),

## .. cash = col\_double(),

## .. other = col\_double(),

## .. total = col\_double()

## .. )

head(pfizer)

## # A tibble: 6 x 10

## org\_indiv first\_plus first\_name last\_name city state category cash

## <chr> <chr> <chr> <chr> <chr> <chr> <chr> <dbl>

## 1 3-D MEDI~ STEVEN BR~ STEVEN DEITELZW~ NEW ~ LA Profess~ 2625

## 2 AA DOCTO~ AAKASH MO~ AAKASH AHUJA PASO~ CA Expert-~ 1000

## 3 ABBO, LI~ LILIAN MA~ LILIAN ABBO MIAMI FL Busines~ 0

## 4 ABBO, LI~ LILIAN MA~ LILIAN ABBO MIAMI FL Meals 0

## 5 ABBO, LI~ LILIAN MA~ LILIAN ABBO MIAMI FL Profess~ 1800

## 6 ABDULLAH~ ABDULLAH ABDULLAH RAFFEE FLINT MI Expert-~ 750

## # ... with 2 more variables: other <dbl>, total <dbl>

describe(pfizer)

## vars n mean sd median trimmed mad min max

## org\_indiv\* 1 10087 NaN NA NA NaN NA Inf -Inf

## first\_plus\* 2 9884 NaN NA NA NaN NA Inf -Inf

## first\_name\* 3 9884 NaN NA NA NaN NA Inf -Inf

## last\_name\* 4 10087 NaN NA NA NaN NA Inf -Inf

## city\* 5 10087 NaN NA NA NaN NA Inf -Inf

## state\* 6 10087 NaN NA NA NaN NA Inf -Inf

## category\* 7 10086 NaN NA NA NaN NA Inf -Inf

## cash 8 10086 3241.12 21815.80 0 814.66 0.00 0 1185466

## other 9 10084 266.47 861.06 41 121.95 60.79 0 27681

## total 10 10087 3506.57 21792.20 750 1111.64 947.38 0 1185466

## range skew kurtosis se

## org\_indiv\* -Inf NA NA NA

## first\_plus\* -Inf NA NA NA

## first\_name\* -Inf NA NA NA

## last\_name\* -Inf NA NA NA

## city\* -Inf NA NA NA

## state\* -Inf NA NA NA

## category\* -Inf NA NA NA

## cash 1185466 30.20 1280.49 217.23

## other 27681 12.27 234.62 8.57

## total 1185466 30.26 1284.45 216.98

You do not need the “describe” command for the fda data, because there are no continuous variables.

str(fda)

## Classes 'spec\_tbl\_df', 'tbl\_df', 'tbl' and 'data.frame': 272 obs. of 5 variables:

## $ name\_last : chr "ADELGLASS" "ADKINSON" "ALLEN" "AMSTERDAM" ...

## $ name\_first : chr "JEFFREY" "N." "MARK" "DANIEL" ...

## $ name\_middle: chr "M." "FRANKLIN" "S." NA ...

## $ issued : chr "5/25/1999" "4/19/2000" "1/28/2002" "11/17/2004" ...

## $ office : chr "Center for Drug Evaluation and Research" "Center for Biologics Evaluation and Research" "Center for Devices and Radiological Health" "Center for Biologics Evaluation and Research" ...

## - attr(\*, "spec")=

## .. cols(

## .. name\_last = col\_character(),

## .. name\_first = col\_character(),

## .. name\_middle = col\_character(),

## .. issued = col\_character(),

## .. office = col\_character()

## .. )

head(fda)

## # A tibble: 6 x 5

## name\_last name\_first name\_middle issued office

## <chr> <chr> <chr> <chr> <chr>

## 1 ADELGLASS JEFFREY M. 5/25/1999 Center for Drug Evaluation an~

## 2 ADKINSON N. FRANKLIN 4/19/2000 Center for Biologics Evaluati~

## 3 ALLEN MARK S. 1/28/2002 Center for Devices and Radiol~

## 4 AMSTERDAM DANIEL <NA> 11/17/20~ Center for Biologics Evaluati~

## 5 AMSTUTZ HARLAN C. 7/19/2004 Center for Devices and Radiol~

## 6 ANDERSON C. JOSEPH 2/25/2000 Center for Devices and Radiol~

Notice that issued has been recognized as a Date variable. Other common data types include num, for numbers that may contain decimals and **POSIXct** for full date and time.

To specify an individual column use the name of the data frame and the column name, separated by $. Type this into your script and run:

The output will be the first 10,000 values for that column.

If you need to change the data type for any column, use the following functions:

. **as.character** converts to a text string

. **as.numeric** converts to a number

. **as.factor** converts to a categorical variable

. **as.integer** converts to an integer

. **as.Date** converts to a date

. **as.POSIXct** convets to a full date and time

(Conversions to full dates and times can get complicated, because of time zones. Now add the following code to your script to convert total in the pfizer data to a numeric variable (which would allow it to hold decimal values, if we had any). Later, we will use the package, **Lubridate**, to help sort dates in varying formats.

pfizer$total <- as.numeric(pfizer$total) *# convert total to numeric variable*

str(pfizer)

## Classes 'spec\_tbl\_df', 'tbl\_df', 'tbl' and 'data.frame': 10087 obs. of 10 variables:

## $ org\_indiv : chr "3-D MEDICAL SERVICES LLC" "AA DOCTORS, INC." "ABBO, LILIAN MARGARITA" "ABBO, LILIAN MARGARITA" ...

## $ first\_plus: chr "STEVEN BRUCE" "AAKASH MOHAN" "LILIAN MARGARITA" "LILIAN MARGARITA" ...

## $ first\_name: chr "STEVEN" "AAKASH" "LILIAN" "LILIAN" ...

## $ last\_name : chr "DEITELZWEIG" "AHUJA" "ABBO" "ABBO" ...

## $ city : chr "NEW ORLEANS" "PASO ROBLES" "MIAMI" "MIAMI" ...

## $ state : chr "LA" "CA" "FL" "FL" ...

## $ category : chr "Professional Advising" "Expert-Led Forums" "Business Related Travel" "Meals" ...

## $ cash : num 2625 1000 0 0 1800 ...

## $ other : num 0 0 448 119 0 0 47 0 0 396 ...

## $ total : num 2625 1000 448 119 1800 ...

## - attr(\*, "spec")=

## .. cols(

## .. org\_indiv = col\_character(),

## .. first\_plus = col\_character(),

## .. first\_name = col\_character(),

## .. last\_name = col\_character(),

## .. city = col\_character(),

## .. state = col\_character(),

## .. category = col\_character(),

## .. cash = col\_double(),

## .. other = col\_double(),

## .. total = col\_double()

## .. )

The summary function will run a quick statistical summary of a data frame, calculating mean, median and quartile values for continuous variables:

summary(pfizer) *# summary of pfizer data*

## org\_indiv first\_plus first\_name

## Length:10087 Length:10087 Length:10087

## Class :character Class :character Class :character

## Mode :character Mode :character Mode :character

## last\_name city state

## Length:10087 Length:10087 Length:10087

## Class :character Class :character Class :character

## Mode :character Mode :character Mode :character

## category cash other total

## Length:10087 Min. : 0 Min. : 0.0 Min. : 0

## Class :character 1st Qu.: 0 1st Qu.: 0.0 1st Qu.: 191

## Mode :character Median : 0 Median : 41.0 Median : 750

## Mean : 3241 Mean : 266.5 Mean : 3507

## 3rd Qu.: 2000 3rd Qu.: 262.0 3rd Qu.: 2000

## Max. :1185466 Max. :27681.0 Max. :1185466

## NA's :1 NA's :3

Manipulate and analyze data

Now we will use **dplyr** to manipulate the data, using operations and functions:

. **Sort**: Largest to smallest, oldest to newest, alphabetical etc.

. **Filter:** Select a defined subset of the data.

. **Join:** Merging entries from two or more datasets based on common field(s), e.g. unique ID number, last name and first name.

. **select** - Keeps and/or drops specified variables. Also reorders variables left to right

. **filter** - Keeps and/or drops records based on value of specified variables

. **arrange** - Sort the data, by size for continuous variables, by date, or alphabetically.

. **group\_by** - Group the data by a categorical variable.

. **mutate** - Create new column(s) in the data, or change existing column(s).

Conditional mutate() with ifelse()

General syntax for ifelse() function, usually paired with mutate:   
 ifelse(condition tested with logical operator, output if condition is true, output if condition is false)   
 EX: mpg1 <-mpg %>% mutate(size= ifelse(displ < 4, “small”, “big”)

. **bind\_rows** - Merge two data frames into one, combining data from columns with the same name.

. **summarize** - Summarize, or aggregate (for each group if following group\_by). Often used in conjunction with functions including:

o **mean** Calculate the mean, or average

o **median** Calculate the median

o **max** Find the maximum value

o **min** Find the minimum value

o **sum** Add all the values together

o ***n*** Count the number of records

There are also various functions to join data, which we will explore below.

Filter and sort data

Filter and sort the data in specific ways. For each of the following examples, copy the code that follows into your script, and view the results. Notice how we create a new objects to hold the processed data.

Find doctors in California paid $10,000 or more by Pfizer to run “Expert-Led Forums.”

*# doctors in California who were paid $10,000 or more by Pfizer to run "Expert-Led Forums."*

ca\_expert\_10000 <- pfizer %>%

filter(state == "CA" & total >= 10000 & category == "Expert-Led Forums")

Notice the use of == to find values that match the specified text, >= for greater than or equal to, and the Boolean operator &.

Now add a sort to the end of the code to list the doctors in descending order by the payments received:

*# doctors in California who were paid $10,000 or more by Pfizer to run "Expert-Led Forums."*

ca\_expert\_10000 <- pfizer %>%

filter(state == "CA" & total >= 10000 & category == "Expert-Led Forums") %>%

arrange(desc(total))

If you arrange without the desc function, the sort will be from smallest to largest.

*# Find doctors in California or New York who were paid $10,000 or more by Pfizer to run "Expert-Led Forums."*

ca\_ny\_expert\_10000 <- pfizer %>%

filter((state == "CA" | state == "NY") & total >= 10000 & category == "Expert-Led Forums") %>%

arrange(desc(total))

Notice the use of the | Boolean operator, and the brackets around that part of the query. This ensures that this part of the query is run first. See what happens if you exclude them.

*# Find doctors in states other than California who were paid $10,000 or more by Pfizer to run "Expert-Led Forums."*

not\_ca\_expert\_10000 <- pfizer %>%

filter(state != "CA" & total >= 10000 & category=="Expert-Led Forums") %>%

arrange(desc(total))

Notice the use of the != operator to exclude doctors in California.

*# Find the 20 doctors across the four largest states (CA, TX, FL, NY) who were paid the most for professional advice.*

ca\_ny\_tx\_fl\_prof\_top20 <- pfizer %>%

filter((state=="CA" | state == "NY" | state == "TX" | state == "FL") & category == "Professional Advising") %>%

arrange(desc(total)) %>%

head(20)

Notice the use of head, which grabs a defined number of rows from the start of a data frame. Here, it is crucial to run the sort first! See what happens if you change the order of the last two lines.

Filter the data for all payments for running Expert-Led Forums or for Professional Advising, and arrange alphabetically by doctor (last name, then first name)

*# Filter the data for all payments for running Expert-Led Forums or for Professional Advising, and arrange alphabetically by doctor (last name, then first name)*

expert\_advice <- pfizer %>%

filter(category == "Expert-Led Forums" | category == "Professional Advising") %>%

arrange(last\_name, first\_name)

Notice that you can sort by multiple variables, separated by commas. Use pattern matching to filter text.

The following code uses the grepl function to find values containing a particular string of text. This can simplify the code used to filter based on text.

*# use pattern matching with grepl to filter text*

expert\_advice <- pfizer %>%

filter(grepl("Expert|Professional", category)) %>%

arrange(last\_name, first\_name)

not\_expert\_advice <- pfizer %>%

filter(!grepl("Expert|Professional", category)) %>%

arrange(last\_name, first\_name)

This code differs only by the ! Boolean operator. Notice that it has split the data into two, based on categories of payment.

Append one data frame to another.

The following code uses the bind\_rows function to append one data frame to another, here recreating the unfiltered data from the two data frames above.

*# merge/append data frames*

pfizer2 <- bind\_rows(expert\_advice, not\_expert\_advice)

Write data to a CSV file

**readr** can write data to CSV and other text files.

*# write expert\_advice data to a csv file*

write\_csv(expert\_advice, "expert\_advice.csv", na="")

**na=“”** ensures that any empty cells in the data frame are saved as blanks - R represents null values as NA, so if you don’t include this, any null values will appear as NA in the saved file.

Group and summarize data

Calculate the total payments, by state

*# calculate total payments by state*

state\_sum <- pfizer %>%

group\_by(state) %>%

summarize(sum = sum(total)) %>%

arrange(desc(sum))

Notice the use of group\_by followed by summarize to group and summarize data, here using the function sum.

Calculate some additional summary statistics, by state

*# As above, but for each state also calculate the median payment, and the number of payments*

state\_summary <- pfizer %>%

group\_by(state) %>%

summarize(sum = sum(total), median = median(total), count = n()) %>%

arrange(desc(sum))

Notice the use of multiple summary functions, sum, median, and n. (You don’t specify a variable for n because it is simply counting the number of rows in the data.)

Group and summarize for multiple categories

*# as above, but group by state and category*

state\_category\_summary <- pfizer %>%

group\_by(state, category) %>%

summarize(sum = sum(total), median = median(total), count = n()) %>%

arrange(state, category)

As for arrange, you can group\_by by multiple variables, separated by commas.

## Working with dates

Now let’s see how to work with dates, using the FDA warning letters data.

Filter the data for letters sent from the start of 2005 onwards. FDA sent warning letters from the start of 2005 onwards

You will have to fix “issued”" to be read as a date. If you look back at str(fda), it was read in as a chr (character). To coerce it to be a date, use the command,

fda$issued <- as.Date(fda$issued, "%m/%d/%Y")

Check it out by running. Alternatively use the package “lubridate”

*#install.packages("lubridate")*

**library**(lubridate)

##

## Attaching package: 'lubridate'

## The following object is masked from 'package:base':

##

## date

fda$issued <- ymd(fda$issued)

fda$issued

fda$issued

str(fda)

## Classes 'spec\_tbl\_df', 'tbl\_df', 'tbl' and 'data.frame': 272 obs. of 5 variables:

## $ name\_last : chr "ADELGLASS" "ADKINSON" "ALLEN" "AMSTERDAM" ...

## $ name\_first : chr "JEFFREY" "N." "MARK" "DANIEL" ...

## $ name\_middle: chr "M." "FRANKLIN" "S." NA ...

## $ issued : Date, format: "1999-05-25" "2000-04-19" ...

## $ office : chr "Center for Drug Evaluation and Research" "Center for Biologics Evaluation and Research" "Center for Devices and Radiological Health" "Center for Biologics Evaluation and Research" ...

## - attr(\*, "spec")=

## .. cols(

## .. name\_last = col\_character(),

## .. name\_first = col\_character(),

## .. name\_middle = col\_character(),

## .. issued = col\_character(),

## .. office = col\_character()

## .. )

post2005 <- fda %>%

filter(issued >= "2005-01-01") %>%

arrange(issued)

Notice that operators like >= can be used for dates, as well as for numbers.

Count the number of letters issued by year

*# count the letters by year*

letters\_year <- fda %>%

mutate(year = format(issued, "%Y")) %>%

group\_by(year) %>%

summarize(letters=n())

This code introduces dplyr’s mutate function to create a new column in the data. The new variable year is the four-digit year “%Y (see here for more on time and date formats in R), extracted from the issued dates using the format function. Then the code groups by year and counts the number of letters for each one.

Add columns giving the number of days and weeks that have elapsed since each letter was sent

*# add new columns showing many days and weeks elapsed since each letter was sent*

fda <- fda %>%

mutate(days\_elapsed = Sys.Date() - issued,

weeks\_elapsed = difftime(Sys.Date(), issued, units = "weeks"))

Notice in the first line that this code changes the fda data frame, rather than creating a new object. The function Sys.Date returns the current date, and if you subtract another date, it will calculate the difference in days. To calculate date and time differences using other units, use the difftime function.

Notice also that you can mutate multiple columns at one go, separated by commas.

Join data from two data frames

Here is an animation for the different types of joins: <https://github.com/gadenbuie/tidyexplain>

There are a number of join functions in **dplyr** to combine data from two data frames. Here are the most useful:

. **inner\_join()** returns values from both tables only where there is a match

. **left\_join()** returns all the values from the first-mentioned table, plus those from the second table that match

. **semi\_join()** filters the first-mentioned table to include only values that have matches in the second table

. **anti\_join()** filters the first-mentioned table to include only values that have no matches in the second table.

To illustrate, these joins will find doctors paid by Pfizer to run expert led forums who had also received a warning letter from the FDA:

*# join to identify doctors paid to run Expert-led forums who also received a warning letter*

expert\_warned\_inner <- inner\_join(pfizer, fda, by=c("first\_name" = "name\_first", "last\_name" = "name\_last")) %>%

filter(category=="Expert-Led Forums")

expert\_warned\_semi <- semi\_join(pfizer, fda, by=c("first\_name" = "name\_first", "last\_name" = "name\_last")) %>%

filter(category=="Expert-Led Forums")

The code in by=c() defines how the **join** should be made. If instructions on how to join the tables are not supplied, **dplyr** will look for columns with matching names, and perform the **join** based on those.

The difference between the two **joins** above is that the first contains all of the columns from both data frames, while the second gives only columns from the pfizer data frame.

In practice, you may wish to inner\_join and then use **dplyr’s** select function to select the columns that you want to retain, for example:

**# as above, but select desired columns from data**

**expert\_warned <- inner\_join(pfizer, fda, by=c("first\_name" = "name\_first", "last\_name" = "name\_last")) %>%**

**filter(category=="Expert-Led Forums") %>%**

**select(first\_plus, last\_name, city, state, total, issued)**

**expert\_warned <- inner\_join(pfizer, fda, by=c("first\_name" = "name\_first", "last\_name" = "name\_last")) %>%**

**filter(category=="Expert-Led Forums") %>%**

**select(2:5,10,12)**

Notice that you can select by columns’ names, or by their positions, where 1 is the first column, 3 is the third, and so on. The code in by=c() defines how the join should be made. If instructions on how to join the tables are not supplied, dplyr will look for columns with matching names, and perform the join based on those.

The difference between the two joins above is that the first contains all of the columns from both data frames, while the second gives only columns from the pfizer data frame.

In practice, you may wish to inner\_join and then use dplyr’s select function to select the columns that you want to retain, for example:

*# as above, but select desired columns from data*

expert\_warned <- inner\_join(pfizer, fda, by=c("first\_name" = "name\_first", "last\_name" = "name\_last")) %>%

filter(category=="Expert-Led Forums") %>%

select(first\_plus, last\_name, city, state, total, issued)

expert\_warned <- inner\_join(pfizer, fda, by=c("first\_name" = "name\_first", "last\_name" = "name\_last")) %>%

filter(category=="Expert-Led Forums") %>%

select(2:5,10,12)

Notice that you can select by columns’ names, or by their positions, where 1 is the first column, 3 is the third, and so on.

[Here is a useful reference](http://stat545-ubc.github.io/bit001_dplyr-cheatsheet.html) for managing joins with **dplyr**.

## (Additional Information) Merging Two Datasets in R

We can do the same joining process in R. So now we will re-do this using R code.

This page has some simple (but easty-to-understand!) animations on how various types of joins work.

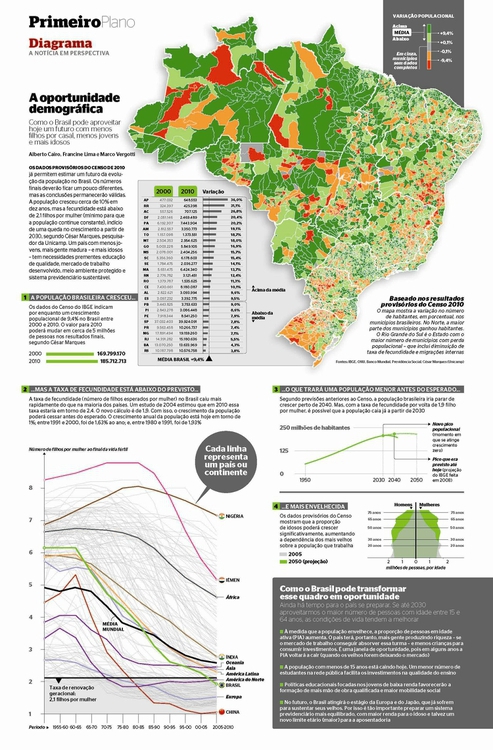
<https://github.com/gadenbuie/tidyexplain>

[Here is a useful reference](http://stat545-ubc.github.io/bit001_dplyr-cheatsheet.html) for managing joins with **dplyr**.

## Tools to Visualize Data

### Break the story down into scenes

Many stories have a step-by-step narrative, and different charts may tell different parts of the story. So think about communicating such stories through a series of graphics. This is another good reason to experiment with different chart types when exploring a new dataset. Here is a nice example of this approach, examining demographic change in Brazil:



(Source: Época, via [Visualopolis](http://visualopolis.com/en/component/content/article/55-featured-small/117-how-we-work-at-epoca.html))

### Good practice for interactives

Nowadays the primary publication medium for many news graphics is the web or apps on mobile platforms, rather than print, which opens up many possibilities for interactivity. This can greatly enhance your ability to tell a story, but it also creates new possibilities to confuse and distract your audience — think of this as interactive chart junk.

A good general approach for interactive graphics is to provide an overview first, and then allow the interested user to zoom or filter to dig deeper into the data. In such cases, the starting state for an interactive should tell a clear story: If users have to make an effort to dig into a graphic to get anything from it, few are likely to do so. Indeed, assume that much of your audience will spend only a short time interacting with the data. [“How Different Groups Spend Their Day”](http://www.nytimes.com/interactive/2009/07/31/business/20080801-metrics-graphic.html) from The New York Times is a good example of this approach.

Similarly, don’t hide labels or information essential to understanding the graphic in tooltips that are accessed only on clicks or hovers. This is where to put more detailed information for users who have sufficient interest to explore further.

Make the controls for an interactive obvious — play buttons should look like play buttons, for instance. You can include a few words of explanation, but only a very few: as far as possible, how to use the interactive should be intuitive, and built into its design.

### Mobile-first’ may change your approach

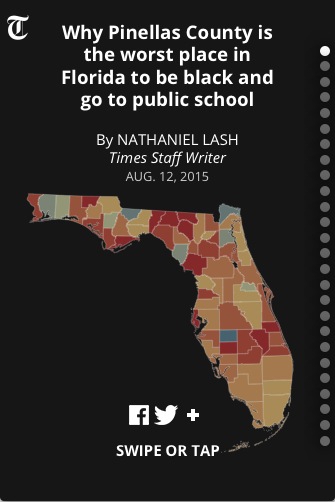
Increasingly, news content is being viewed on mobile devices with small screens

At the most basic level, this means making graphics “responsive,” so that their size adjusts to screen size. But there is more to effective design for mobile than this.

We have already discussed the value of small multiples, which can be made to reflow for different screen sizes.

[This interactive](http://chicagosmilliondollarblocks.com/), exploring spending on incarceration by block in Chicago, is a nice example of organizing and displaying the same material differently for different screen sizes. Open it up on your laptop then reduce the size of your browswer window to see how it behaves.

Again, a step-by-step narrative can be a useful device in overcoming the limitations of a small screen. [This interactive](http://www.tampabay.com/projects/2015/investigations/pinellas-failure-factories/chart-failing-black-students/), exploring school segregation by race in Florida, is a good example of this approach:

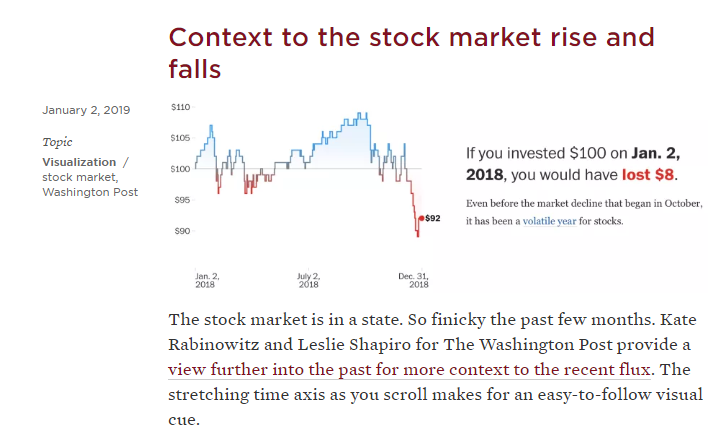


(Source: [Tampa Bay Times](http://www.tampabay.com/projects/2015/investigations/pinellas-failure-factories/chart-failing-black-students/))

[Here’s](http://www.storybench.org/using-buzzfeeds-listicle-format-tell-stories-maps-charts/) an article that includes some of the challenge of making graphics that work effectively on mobile.

Nathan Yau lists a number of data viz tools – Excel, Tableau Public (free at <https://public.tableau.com/en-us/s/>) , and the author’s site: <https://flowingdata.com/>

See the example on the home page of Flowing Data, dated January 2, 2019:



## Further reading

**[Introduction to dplyr](https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html)**

[**RStudio Data Wrangling Cheat Sheet**](https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf)  
Also introduces the [tidyr](https://blog.rstudio.org/2014/07/22/introducing-tidyr/) package, which can manage wide-to-long transformations, among other data manipulations.

[**Stack Overflow**](http://stackoverflow.com/)  
For any work involving code, this question-and-answer site is a great resource for when you get stuck, to see how others have solved similar problems. Search the site, or [browse R questions](http://stackoverflow.com/questions/tagged/r).

# Unit 2 Homework Assignment

1. (Not for a grade) Explore the links at the beginning of the notes for potential sources of data.
2. (Not for a grade) Complete copying the Markdown code from these notes to explore the pfizer and fda data (anything you did not complete in class).
3. (Worth up to 10 points) Select one of the provided articles (below) on the Reproducibility Crisis. Write a short paragraph to summarize the article, then write a paragraph response to the article, including your opinion on how statisticians/data scientists can work to eliminate this problem.

**The Reproducibility Crisis Articles**

* Article: [Fault Found (Again) with Conflicts of Interest](https://www.medpagetoday.com/publichealthpolicy/generalprofessionalissues/76832)
* Article: [ASA Statement on P-Values and P-Hacking](../Articles/Reproducibility/ASA%20StatementMoving%20to%20a%20World%20Beyond%20p%200%2005%203-19.pdf)
* Article: [Cornell's Top Food Researcher has had 13 studies retracted](../Articles/Reproducibility/p-hacking%20A%20top%20Cornell%20food%20researcher%20has%20had%2013%20studies%20retracted.docx)
* Article: [1500 Scientists could not reproduce their own studies](https://www.nature.com/news/1-500-scientists-lift-the-lid-on-reproducibility-1.19970) (in Nature News)  
  Article: [Replicability or reproducibility? On the replication crisis in computational neuroscience and sharing only relevant detail](https://link.springer.com/article/10.1007/s10827-018-0702-z) (Journal of Computational Neuroscience)

Submit all work in the Assignment Dropbox for Homework Unit 2 by **Monday, July 29th at 11:59 pm.** We will present submissions in class on Tuesday, July 30th.