40 Days and 40 Nights

Martin Morgan[1] L. Shawn Matott[2]

2020-05-22

[1]Roswell Park Comprehensive Cancer Center, Martin.Morgan@RoswellPark.org
[2]Roswell Park Comprehensive Cancer Center

Contents

Motivation

This is a WORK IN PROGRESS.

This course was suggested and enabled by Adam Kisailus and Richard Hershberger. It is available for Roswell Park graduate students.

Introduction

The word 'quarantine' is from the 1660's and refers to the fourty days (Italian *quaranta giorni*) a ship suspected of carrying disease was kept in isolation.

What to do in a quarantine? The astronaut Scott Kelly spent nearly a year on the International Space Station. In a New York Times opinion piece he says, among other things, that 'you need a hobby', and what better hobby than a useful one? Let's take the opportunity provided by COVID-19 to learn R for statistical analysis and comprehension of data. Who knows, it may be useful after all this is over!

What to expect

We'll meet via zoom twice a week, Mondays and Fridays, for one hour. We'll use this time to make sure everyone is making progress, and to introduce new or more difficult topics. Other days we'll have short exercises and activities that hopefully provide an opportunity to learn at your own speed.

We haven't thought this through much, but roughly we might cover:

- Week 1: We'll start with the basics of installing and using R. We'll set up *R* and *RStudio* on your local computer, or if that doesn't work use a cloud-based RStudio. We'll learn the basics of *R* – numeric, character, logical, and other vectors; variables; and slightly more complicated representations of 'factors' and dates. We'll also use *RStudio* to write a script that allows us to easily re-create an analysis, illustrating the power concept of *reproducible research*.

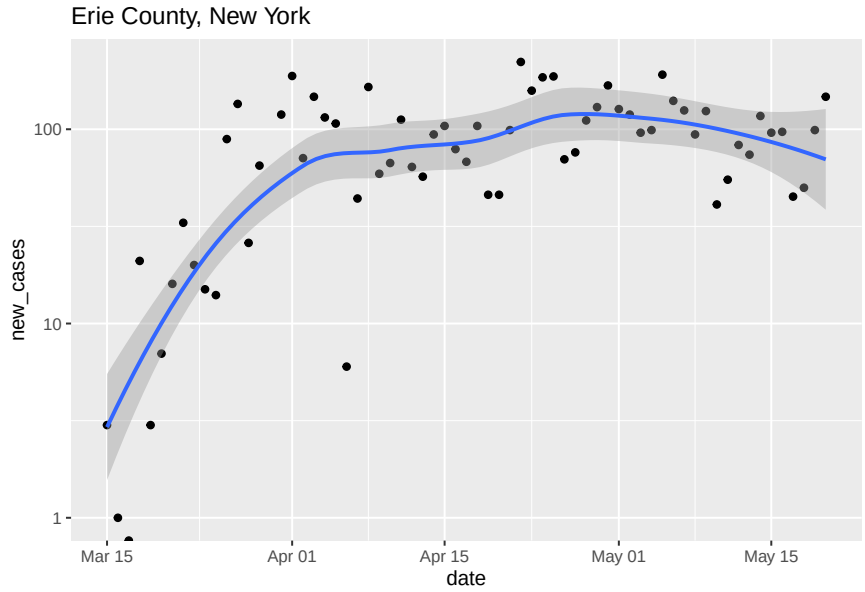
```
activity <- c("check e-mail", "breakfast", "conference call", "webinar", "walk")
minutes_per_activity <- c(20, 30, 60, 60, 60)
minutes_per_activity >= 60
## [1] FALSE FALSE  TRUE  TRUE  TRUE
activity[minutes_per_activity >= 60]
## [1] "conference call" "webinar"         "walk"
```

- Week 2: The `data.frame`. This week is all about *R*'s `data.frame`, a versatile way of representing and manipulating a table (like an Excel spreadsheet) of data. We'll learn how to create, write, and read a `data.frame`; how to go from data in a spreadsheet in Excel to a `data.frame` in *R*; and how to perform simple manipulations on a `data.frame`, like creating a subset of data, summarizing values in a column, and summarizing values in one column based on a grouping variable in another column.

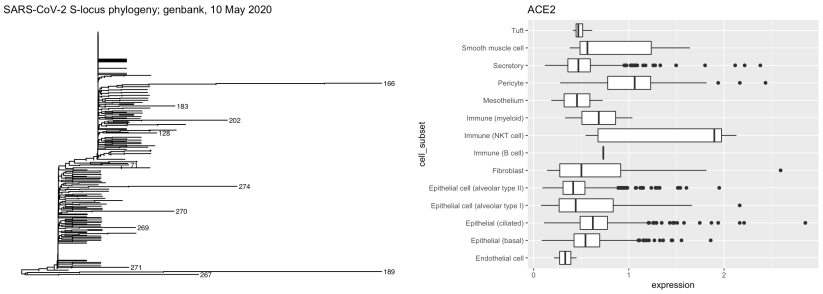
```
url = "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.
cases <- read.csv(url)
erie <- subset(cases, county == "Erie" & state == "New York")
tail(erie)
##              date county    state  fips cases deaths
## 148952 2020-05-16   Erie New York 36029  4867    428
## 151883 2020-05-17   Erie New York 36029  4954    438
## 154819 2020-05-18   Erie New York 36029  4993    444
## 157757 2020-05-19   Erie New York 36029  5037    450
## 160706 2020-05-20   Erie New York 36029  5131    455
## 163659 2020-05-21   Erie New York 36029  5270    463
```

- Week 3: Packages for extending *R*. A great strength of *R* is its extensibility through packages. We'll learn about CRAN, and install and use the 'tidyverse' suite of packages. The tidyverse provides us with an alternative set of tools for working with tabular data, and We'll use publicly available data to explore the spread of COVID-19 in the US. We'll read, filter, mutate (change), and select subsets of the data, and group data by one column (e.g., 'state') to create summaries (e.g., cases per state). We'll also start to explore data visualization, creating our first plots of the spread of COVID-19.

```
library(dplyr)
library(ggplot2)
## ...additional commands
```



- Week 4: Machine learning. This week will develop basic machine learning models for exploring data.
- Week 5: Bioinformatic analysis with Bioconductor. *Bioconductor* is a collection of more than 1800 *R* packages for the statistical analysis and comprehension of high-throughput genomic data. We'll use *Bioconductor* to look at COVID-19 genome sequences, and to explore emerging genomic data relevant to the virus.



- Week 6: COVID-19 has really shown the value of open data and collaboration. In the final week of our quarantine, we'll explore collaboration. We'll learn about writing 'markdown' vignettes (reports to) share our results with others, such as our lab colleagues. We'll write and document functions so that we can easily re-do steps in an analysis. And we will synthesize the vignettes and functions into a package for documenting and sharing our work.

Chapter 1

Basics

1.1 Day 1 (Monday) Zoom orientation

1.1.1 Logistics (10 minutes)

Course material

- Available at https://mtmorgan.github.io/QuaRantine

Cadence

- Monday and Friday group zoom sessions – these will review and troubleshoot previous material, and outline goals for the next set of independent activities.
- Daily independent activities – most of your learning will happen here!

Communicating

- We'll use Microsoft Teams (if most participants have access to the course)
- Visit Microsoft Teams and sign in with your Roswell username (e.g., `MA38727@RoswellPark.org`) and the password you use to check email, etc. Join the 'QuaRantine' team.

1.1.2 Installing *R* and *RStudio* (25 minutes, Shawn)

What is R?

- A programming language for statistical computing, data analysis and scientific graphics.
- Open-source with a large (and growing) user community.
- Currently in the top 10 most popular languages according to the tiobe index.

What is RStudio?

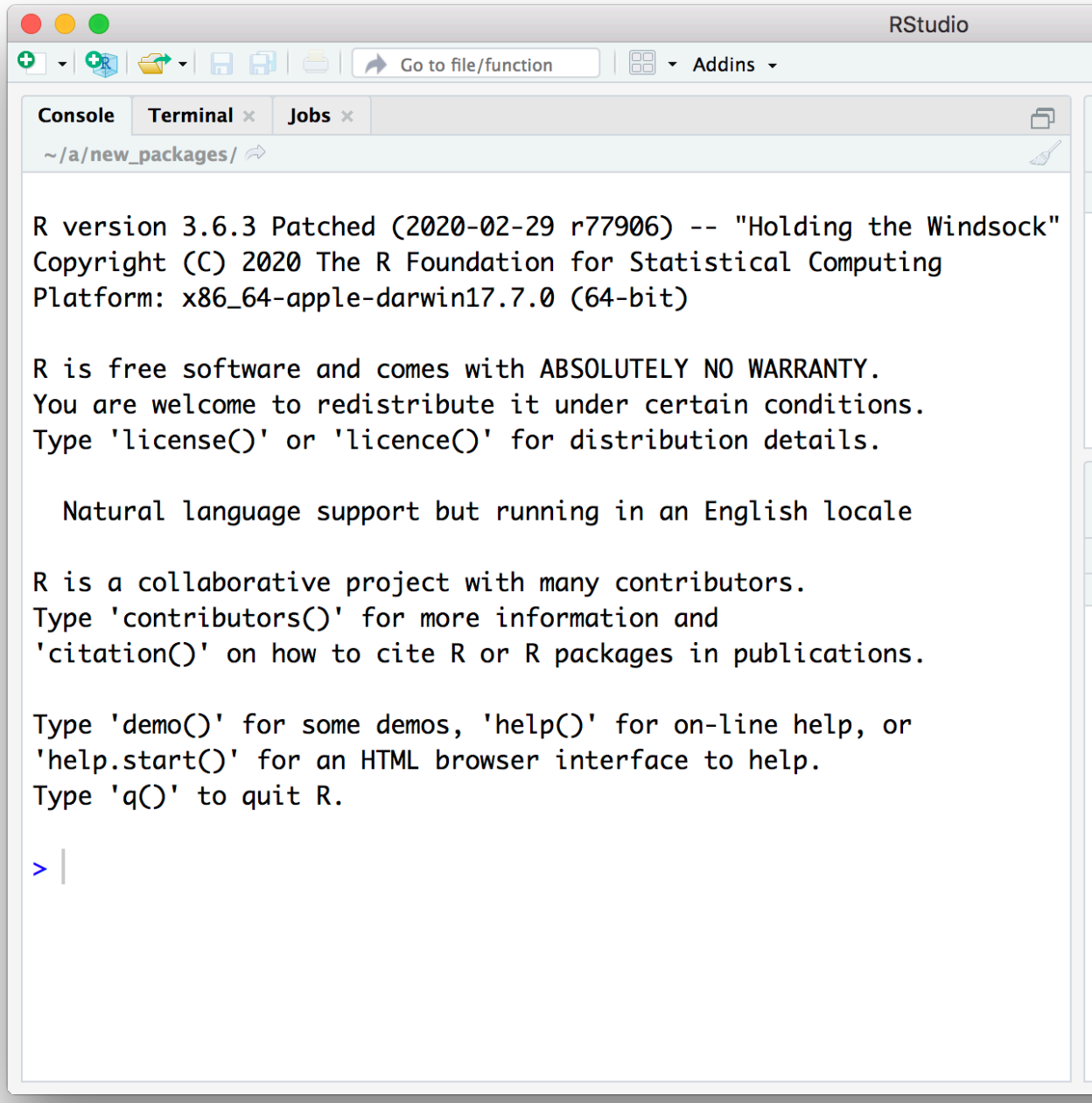
- RStudio provides an integrated editor and shell environment to make R programming easier. Some of the more useful features include:
  - Syntax highlighting and color coding
  - Easy switching between shell and editor
  - Dynamic help and docs

Installing *R* and *RStudio*

- Two ways to "get" RStudio:
  - Install on your laptop or desktop
    - Download the free desktop installer here
  - Use the rstudio.cloud resource
    - Visit rstudio.cloud, sign-up, and sign-on

The preferred approach for this course is to try to install R and RStudio on your own computer

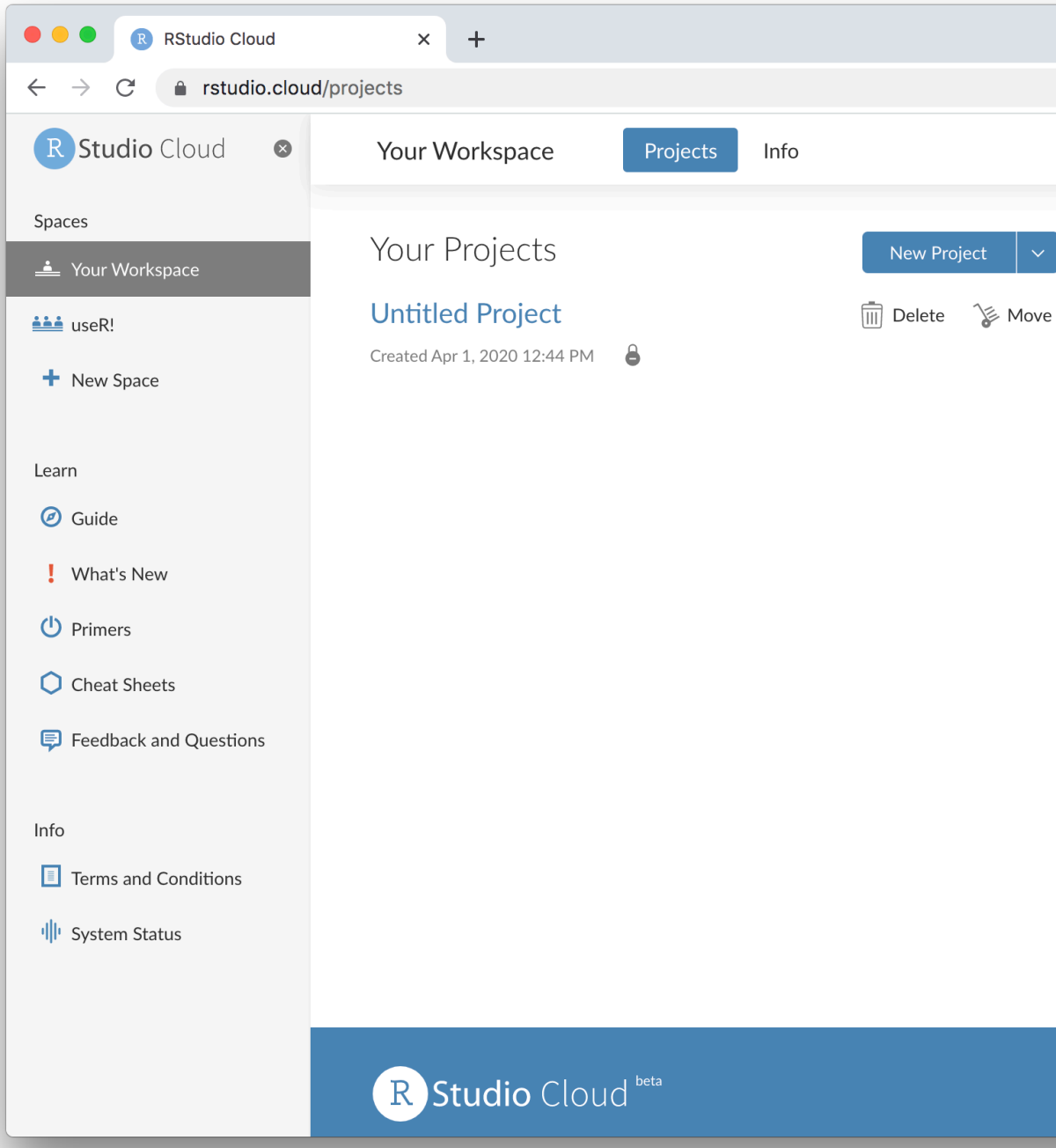
- Windows Users:
  - Download R for Windows and run the installer. Avoid, if possible, installing as administrator.
  - Download RStudio for Windows and run the installer.
  - Test the installation by launching RStudio. You should end up with a window like the screen shot below.
- Mac Users:
  - Download R for macOS (OS X 10.11, El Capitan, and later) or older macOS and run the installer.
  - Download RStudio for macOS and run the installer.
  - Test the installation by launching RStudio. You should end up with a window like the screen shot below.


RStudio
Go to file/function
Addins
Console
Terminal
Jobs
~/a/new_packages/
R version 3.6.3 Patched (2020-02-29 r77906) -- "Holding the Windsock"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin17.7.0 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>

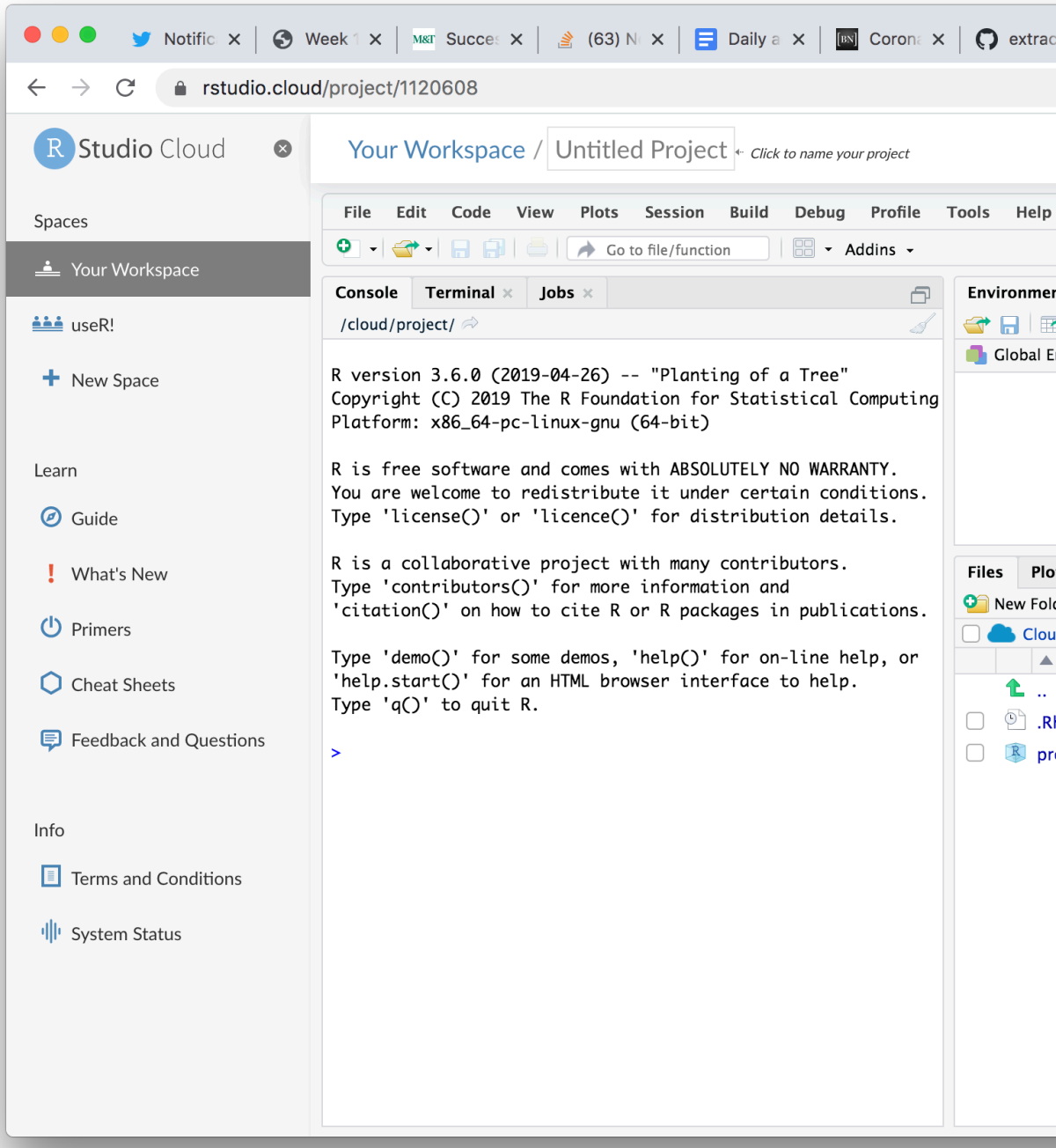
An ALTERNATIVE, if installing on your own computer does not work:

- Do the following only if you are NOT ABLE TO INSTALL R and RStudio.

- Visit rstudio.cloud. Click the 'Get Started' button, and create an account (I used my gmail account…). You should end up at a screen like the following.


RStudio Cloud
rstudio.cloud/projects
Studio Cloud
Spaces
Your Workspace
useR!
New Space
Learn
Guide
What's New
Primers
Cheat Sheets
Feedback and Questions
Info
Terms and Conditions
System Status
Your Workspace
Projects
Info
Your Projects
New Project
Untitled Project
Delete
Move
Created Apr 1, 2020 12:44 PM
Studio Cloud beta

- Click on the 'New Project' button, to end up with a screen like the one below. Note the 'Untitled Project' at the top of the screen; click on it to name your project, e.g., 'QuaRantine'.


rstudio.cloud/project/1120608
Studio Cloud
Your Workspace / Untitled Project
Click to name your project
Spaces
Your Workspace
useR!
New Space
Learn
Guide
What's New
Primers
Cheat Sheets
Feedback and Questions
Info
Terms and Conditions
System Status
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Console
Terminal
Jobs
/cloud/project/
R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Environmer
Global E
Files
Plo
New Fold
Clou

Breakout Room

At this point you should have RStudio running either via your desktop installation or through rstudio.cloud. If not, please let us know via the chat window and we'll invite you to a breakout room to troubleshoot your installation.

1.1.3 Basics of *R* (25 minutes)

R as a simple calculator

```
1 + 2
## [1] 3
```

R Console Output

Enter this in the console:

```
2 + 3 * 5
## [1] 17
```

Q: what's the `[1]` all about in the output?

A: It's the index of the first entry in each line.

This is maybe a better example:

```
1:30
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30
```

Displaying help in the R Console

`? <command-name>`

- Some examples:

  ```
  ? cat
  ? print
  ```

Variables

Naming variables in *R*

- A variable name can contain letters, numbers, and the dot `.` or underline `_` characters. Variables should start with a letter.
- Try entering these in the console:

  ```
  y = 2
  ```

  ```
  try.this = 33.3
  ```

  ```
  oneMoreTime = "woohoo"
  ```

- Now try these:

  ```
  2y = 2
  ```

  ```
  _z = 33.3
  ```

  ```
  function = "oops, my bad"
  ```

R is case sensitive (R != r)

```
R = 2
r = 3
R == r
## [1] FALSE
```

Variable Assignment

- You may use `=` or `<-` (and even `->`) to assign values to a variable.

  ```
  x <- 2 + 3 * 5
  y =  2 + 3 * 6
  2 + 3 * 7 -> z
  cat(x, y, z)
  ## 17 20 23
  ```

R's four basic 'atomic' data types

- Numeric (includes integer, double, etc.)
  - `3.14`, `1`, `2600`
- Character (string)
  - `"hey, I'm a string"`
  - `'single quotes are ok too'`
- Logical
  - `TRUE` or `FALSE` (note all caps)
- `NA`
  - not assigned (no known value)

Use `class()` to query the class of data:

```
a <- 5
class(a)
## [1] "numeric"
```

Use `as.` to coerce a variable to a specific data type

```
a <- as.integer(5)
class(a)
## [1] "integer"
```

```
d <- as.logical(a)
d
## [1] TRUE
class(d)
## [1] "logical"
```

Using Logical Operators

Equivalence test (`==`):

```
1 == 2
## [1] FALSE
```

Not equal test (`!=`):

```
1 != 2
## [1] TRUE
```

less-than (`<`) and greater-than (`>`):

```
18 > 44
## [1] FALSE
3 < 204
## [1] TRUE
```

Logical Or (`|`):

```
(1 == 2) | (2 == 2)
## [1] TRUE
```

Logical And (`&`):

```
(1 == 2) & (2 == 2)
## [1] FALSE
```

Objects and Vectors in R

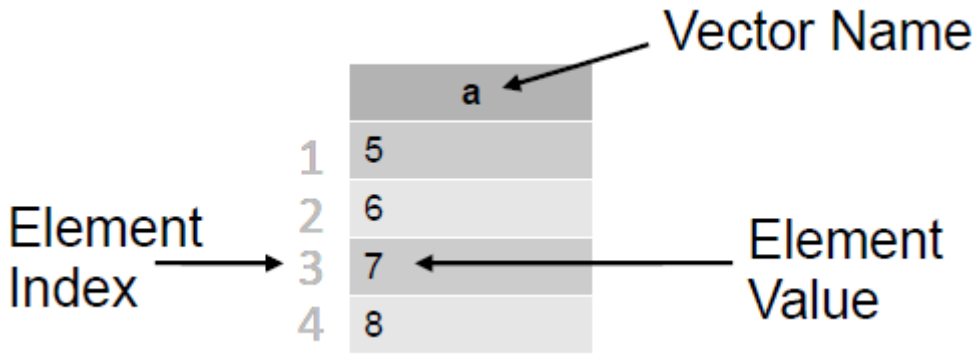
Objects

- R stores everything, variables included, in 'objects'.

```
x <- 2.71

# print the value of an object
print(x)
## [1] 2.71

# determine class or internal type of an object
class(x)
## [1] "numeric"

# TRUE if an object has not been assigned a value
is.na(x)
## [1] FALSE
```

Vectors

- 'Vectors' and 'data frames' are the bread and butter of R
- Vectors consist of several elements of the same class
  - e.g. a vector of heart rates, one per patient



Data frames (`data.frame`)

- Data frames are structures that can contain columns of various types
  - e.g. height, weight, age, heart rate, etc.
  - Handy containers for experimental data
  - Analogous to spreadsheet data

  - More on Data Frames throughout the week!

Working with Vectors

Creating a Vector

- Use the `c()` function

```
name <- c("John Doe", "Jane Smith", "MacGillicuddy Jones", "Echo Shamus")
age <- c(36, 54, 82, 15)
favorite_color <- c("red", "orange", "green", "black")

## print the vectors
name
## [1] "John Doe"            "Jane Smith"          "MacGillicuddy Jones"
## [4] "Echo Shamus"
age
## [1] 36 54 82 15
favorite_color
## [1] "red"    "orange" "green"  "black"
```

Accessing vector data

- Use numerical indexing
- R uses 1-based indexing
  - 1st vector element has index of 1
  - 2nd has an index of 2
  - 3rd has an index of 3
  - and so on

```
name[1]
## [1] "John Doe"
age[3]
## [1] 82
```

- R supports "slicing" (i.e. extracting multiple items)

```
favorite_color[c(2, 3)]
## [1] "orange" "green"
```

- Negative indices are omitted

```
age[-2]
## [1] 36 82 15
```

Some Useful Vector Operations

- `length()`: number of elements
- `sum()`: sum of all element values
- `unique()`: distinct values
- `sort()`: sort elements, omitting NAs

- `order()`: indices of sorted elements, NAs are last
- `rev()`: reverse the order
- `summary()`: simple statistics

```
a <- c(5, 5, 6, 7, 8, 4)
sum(a)
## [1] 35
length(a)
## [1] 6
unique(a)
## [1] 5 6 7 8 4
sort(a)
## [1] 4 5 5 6 7 8
order(a)
## [1] 6 1 2 3 4 5
a[order(a)]
## [1] 4 5 5 6 7 8
rev(a)
## [1] 4 8 7 6 5 5
summary(a)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.000   5.000   5.500   5.833   6.750   8.000
```

Handling Missing Data

- First consider the reason(s) for the missing data
  - e.g. concentrations that are below detectable levels?
- Sometimes NAs in data require special statistical methods
- Other times we can safely discard / ignore NA entries
- To remove NAs prior to a calculation:

```
y = c(1,NA,3,2,NA)
sum(y, na.rm=TRUE)
## [1] 6
```

Wrapping up day 1

The goal for today was to rapidly cover some of the essential aspects of R programming. For the remainder of the week you'll work at your own pace to get more of a hands-on deep dive into this material. If you run into trouble please don't hesiate to ask for help via Teams (QuaRantine Team), slack (QuaRantine Course), or email (Drs. Matott and Morgan) — whatever works best for you!

1.2 Day 2: Vectors and variables

Our overall goal for the next few days is to use *R* to create a daily log of quarantine activities.

Our goal for today is to become familiar with *R* vectors. Along the way we'll probably make data entry and other errors that will start to get us comfortable with *R*.

If you run into problems, reach out to the slack channel for support!

The astronaut Scott Kelly said that to survive a year on the International Space Station he found it essential to

- Follow a schedule – plan your day, and stick to the plan
- Pace yourselves – you've got a long time to accomplish tasks, so don't try to get everything done in the first week.
- Go outside – if Scott can head out to space, we should be able to make it to the back yard or around the block!
- Get a hobby – something not work related, and away from that evil little screen. Maybe it's as simple as rediscovering the joy of reading.
- Keep a journal
- Take time to connect – on a human level, with people you work with and people you don't!
- Listen to experts – Scott talked about relying on the mission controllers; for us maybe that's watching webinars or taking courses in new topics!
- Wash your hands!

I wanted to emphasize 'follow a schedule' and 'keep a journal'. How can *R* help? Well, I want to create a short record of how I spend today, day 2 of my quarantine.

My first goal is to create *vectors* describing things I plan to do today. Let's start with some of these. To get up to speed, type the following into the *R* console, at the `>` prompt

```
1 + 2
```

Press the carriage return and remind yourself that *R* is a calculator, and knows how to work with numbers!

Now type an activity in your day, for instance I often start with

```
"check e-mail"
```

Now try assigning that to a variable, and displaying the variable, e.g.,

```
activity <- "check e-mail"
activity
## [1] "check e-mail"
```

OK, likely you have several activities scheduled. Create a *vector* of a few of these by concatenating individual values

```
c("check e-mail", "breakfast", "conference call", "webinar", "walk")
## [1] "check e-mail"    "breakfast"       "conference call" "webinar"
## [5] "walk"
```

Assign these to a variable

```
activity <- c("check e-mail", "breakfast", "conference call", "webinar", "walk")
activity
## [1] "check e-mail"    "breakfast"       "conference call" "webinar"
## [5] "walk"
```

Create another vector, but this time the vector should contain the minutes spent on each activity

```
minutes <- c(20, 30, 60, 60, 60)
minutes
## [1] 20 30 60 60 60
```

So I spent 20 minutes checking email, 30 minutes having breakfast and things like that, I was in a conference call for 60 minutes, and then attended a webinar where I learned new stuff for another 60 minutes. Finally I went for a walk to clear my head and remember why I'm doing things.

Apply some basic functions to the variables, e.g., use `length()` to demonstrate that you for each `activity` you have recorded the `minutes`.

```
length(activity)
## [1] 5
length(minutes)
## [1] 5
```

Use `tail()` to select the last two activities (or `head()` to select the first two…)

```
tail(activity, 2)
## [1] "webinar" "walk"
tail(minutes, 2)
## [1] 60 60
```

R has other types of vectors. Create a logical vector that indicates whether each activity was 'work' activity' or something you did for your own survival. We'll say that checking email is a work-related activity!

```
is_work <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
is_work
## [1]  TRUE FALSE  TRUE  TRUE FALSE
```

1.3 Day 3: `factor()`, `Date()`, and `NA`

Yesterday we learned about `character`, `numeric`, and `logical` vectors in *R* (you may need to revisit previous notes and re-create these variables)

```
activity
## [1] "check e-mail"    "breakfast"       "conference call" "webinar"
## [5] "walk"
```

```
minutes
## [1] 20 30 60 60 60
is_work
## [1]  TRUE FALSE  TRUE  TRUE FALSE
```

Today we will learn about slightly more complicated vectors.

We created the logical vector `is_work` to classify each `activity` as either work-related or not. What if we had several different categories? For instance, we might want to classify the activities into categories inspired by astronaut Kelly's guidance. Categories might include: `connect` with others; go outside and `exercise`; `consult` experts; get a `hobby`; and (my own category, I guess) perform `essential` functions like eating and sleeping. So the values of `activity` could be classified as

```
classification <-
    c("connect", "essential", "connect", "consult", "exercise")
```

I want to emphasize a difference between the `activity` and `classification` variables. I want `activity` to be a character vector that could contain any description of an activity. But I want `classification` to be terms only from a limited set of possibilities. In *R*, I want `classification` to be a special type of vector called a `factor`, with the *values* of the vector restricted to a set of possible *levels* that I define. I create a factor by enumerating the possible *levels* that the factor can take on

```
levels <- c("connect", "exercise", "consult", "hobby", "essential")
```

And then tell *R* that the vector `classification` should be a factor with values taken from a particular set of levels

```
classification <- factor(
    c("connect", "essential", "connect", "consult", "exercise"),
    levels = levels
)
classification
## [1] connect   essential connect   consult   exercise
## Levels: connect exercise consult hobby essential
```

Notice that activity (a character vector) displays differently from classification (a factor)

```
activity
## [1] "check e-mail"    "breakfast"       "conference call" "webinar"
## [5] "walk"
classification
## [1] connect   essential connect   consult   exercise
## Levels: connect exercise consult hobby essential
```

Also, some of the levels (e.g., hobby) have not been part of our schedule yet, but the factor still 'knows' about the level.

Notice also what happens when I try to use a value (disconnect) that is not a level of a factor

```
factor(c("connect", "disconnect"), levels = levels)
## [1] connect <NA>
## Levels: connect exercise consult hobby essential
```

The value with the unknown level is displayed as NA, for 'not known'. NA values can be present in any vector, e.g.,

```
c(1, 2, NA, 4)
## [1]  1  2 NA  4
c("walk", "talk", NA)
## [1] "walk" "talk" NA
c(NA, TRUE, FALSE, TRUE, TRUE)
## [1]    NA  TRUE FALSE  TRUE  TRUE
```

This serves as an indication that the value is simply not available. Use NA rather than adopting some special code (e.g., '-99') to indicate when a value is not available.

One other type of vector we will work a lot with are dates. All of my activities are for today, so I'll start with a character vector with the same length as my activity vector, each indicating the date in a consistent month-day-year format

```
dates <- c("04-14-2020", "04-14-2020", "04-14-2020", "04-14-2020", "04-14-2020")
dates
## [1] "04-14-2020" "04-14-2020" "04-14-2020" "04-14-2020" "04-14-2020"
```

Incidentally, I could do this more efficiently using the replicate function

```
rep("04-14-2020", 5)
## [1] "04-14-2020" "04-14-2020" "04-14-2020" "04-14-2020" "04-14-2020"
```

And even better use length() to know for sure how many times I should replicate the character vector

```
rep("04-14-2020", length(activity))
## [1] "04-14-2020" "04-14-2020" "04-14-2020" "04-14-2020" "04-14-2020"
```

dates is a character vector, but it has specially meaning as a calendar date, *R* has a Date *class* that knows how to work with dates, for instance to calculate the number of days between two dates. We will *coerce* date to an object of class Date using a function as.Date. Here's our first attempt…

```
as.Date(dates)
```

… but this results in an error:

```
Error in charToDate(x) :
  character string is not in a standard unambiguous format
```

R doesn't know the format (month-day-year) of the dates we provide. The solution is to add a second argument to `as.Date()`. The second argument is a character vector that describes the date format. The format we use is `"%m-%d-%Y"`, which says that we provide the `%`month first, then a hyphen, then the `%`day, another hyphen, and finally the four-digit `%`Year.

```
as.Date(dates, format = "%m-%d-%Y")
## [1] "2020-04-14" "2020-04-14" "2020-04-14" "2020-04-14" "2020-04-14"
```

Notice that the format has been standardized to year-month-day. Also notice that although the original value of `date` and the return from `as.Data()` look the same, they are actually of different *class*.

```
class(date)
## [1] "function"
class(as.Date(dates, format = "%m-%d-%Y"))
## [1] "Date"
```

R will use the information about class to enable specialized calculation on dates, e.g., to sort them or to determine the number of days between different dates. So here's our `date` vector as a `Date` object.

```
dates <- rep("04-14-2020", length(activity))
date <- as.Date(dates, format = "%m-%d-%Y")
date
## [1] "2020-04-14" "2020-04-14" "2020-04-14" "2020-04-14" "2020-04-14"
```

OK, time for a walk! See you tomorrow!

1.4 Day 4: Working with variables

Remember that *R* can act as a simple calculator, and that one can create new variables by assignment

```
x <- 1
x + 1
## [1] 2
y <- x + 1
y
## [1] 2
```

Let's apply these ideaas to our `minutes` vector from earlier in the week.

```
minutes <- c(20, 30, 60, 60, 60)
```

We can perform basic arithmetic on vectors. Suppose we wanted to increase the time of each activity by 5 minutes

```
minutes + 5
## [1] 25 35 65 65 65
```

or to increase the time of the first two activities by 5 minutes, and the last three activities by 10 minutes

```
minutes + c(5, 5, 10, 10, 10)
## [1] 25 35 70 70 70
```

R has a very large number of *functions* that can be used on vectors. For instance, the average time spent on activities is

```
mean(minutes)
## [1] 46
```

while the total amount of time is

```
sum(minutes)
## [1] 230
```

Explore other typical mathematical transformations, e.g., `log()`, `log10()`, `sqrt()` (square root), … Check out the help pages for each, e.g., `?log`.

Explore the consequences of `NA` in a vector for functions like `mean()` and `sum()`.

```
x <- c(1, 2, NA, 3)
mean(x)
## [1] NA
```

R is saying that, since there is an unknown (`NA`) value in the vector, it cannot possibly know what the mean is! Tell *R* to remove the missing values before performing the calculation by adding the `na.rm = TRUE` argument

```
mean(x, na.rm = TRUE)
## [1] 2
```

Check out the help page `?mean` to find a description of the `na.rm` and other arguments.

It's possible to perform logical operations on vectors, e.g., to ask which activities lasted 60 minutes or more

```
minutes >= 60
## [1] FALSE FALSE  TRUE  TRUE  TRUE
```

Here's our `activity` vector

```
activity <- c("check e-mail", "breakfast", "conference call", "webinar", "walk")
```

The elements of this vector are numbered from 1 to 5. We can create a new vector that is a subset of this vector using `[` and an integer index, e.g., the second activity is

```
activity[2]
## [1] "breakfast"
```

The index can actually be a vector, so we could choose the second and fourth activity as

```
index <- c(2, 4)
activity[index]
## [1] "breakfast" "webinar"
```

In fact, we can use logical vectors for subsetting. Consider the activities that take sixty minutes or longer:

```
index <- minutes >= 60
activity[index]
## [1] "conference call" "webinar"         "walk"
```

We had previously characterized the activities as 'work' or otherwise.

```
is_work <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
```

Use `is_work` to subset `activity` and identify the work-related activities

```
activity[is_work]
## [1] "check e-mail"    "conference call" "webinar"
```

How many minutes were work-related?

```
work_minutes <- minutes[is_work]
sum(work_minutes)
## [1] 140
```

What about not work related? `!` negates logical vectors, so

```
is_work
## [1]  TRUE FALSE  TRUE  TRUE FALSE
!is_work
## [1] FALSE  TRUE FALSE FALSE  TRUE
non_work_minutes <- minutes[!is_work]
sum(non_work_minutes)
## [1] 90
```

Note that it doesn't make sense to take the `mean()` of a character vector like `activity`, and *R* signals a warning and returns `NA`

```
mean(activity)
## Warning in mean.default(activity): argument is not numeric or logical: returning
## NA
## [1] NA
```

Nonetheless, there are many functions that *do* work on character vectors, e.g.,

the number of letters in each element `nchar()`, or transformation to upper-case

```
nchar(activity)
## [1] 12  9 15  7  4
toupper(activity)
## [1] "CHECK E-MAIL"    "BREAKFAST"       "CONFERENCE CALL" "WEBINAR"
## [5] "WALK"
```

1.5 Day 5 (Friday) Zoom check-in

1.5.1 Logistics

- Please join Microsoft Teams! Need help? Contact Adam.Kisailus at RoswellPark.org.

1.5.2 Review and trouble shoot (25 minutes; Martin)

Data representations

'Atomic' vectors

```
activity <- c("check e-mail", "breakfast", "conference call", "webinar", "walk")
minutes <- c(20, 30, 60, 60, 60)
is_work <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
```

`factor()` and `date()`

```
levels <- c("connect", "exercise", "consult", "hobby", "essential")
classification <- factor(
    c("connect", "essential", "connect", "consult", "exercise"),
    levels = levels
)

dates <- rep("04-14-2020", length(activity))
date <- as.Date(dates, format = "%m-%d-%Y")
```

Missing values

```
x <- c(1, 3, NA, 5)
sum(x)
## [1] NA
sum(x, na.rm = TRUE)
## [1] 9

factor(c("connect", "disconnect"), levels = levels)
## [1] connect <NA>
## Levels: connect exercise consult hobby essential
```

Functions and logical operators

```
x <- c(1, 3, NA, 5)
sum(x)
## [1] NA
sum(x, na.rm = TRUE)
## [1] 9

minutes >= 60
## [1] FALSE FALSE  TRUE  TRUE  TRUE
```

Subsetting vectors

- 1-basaed numeric indexes

```
activity
## [1] "check e-mail"    "breakfast"       "conference call" "webinar"
## [5] "walk"

idx <- c(1, 3, 1)
activity[idx]
## [1] "check e-mail"    "conference call" "check e-mail"
```

- logical index

```
is_work
## [1]  TRUE FALSE  TRUE  TRUE FALSE
activity[is_work]
## [1] "check e-mail"    "conference call" "webinar"

sum(minutes[is_work])
## [1] 140
```

- Maybe more interesting...

```
short <- minutes < 60
short
## [1]  TRUE  TRUE FALSE FALSE FALSE
minutes[short]
## [1] 20 30
activity[short]
## [1] "check e-mail" "breakfast"
```

Other fun topics

`%in%`: a *binary operator*

- is each of the vector elements on the left-hand side *in* the set of elements on the right hand side

```
fruits <- c("banana", "apple", "grape", "orange", "kiwi")
c("apple", "orange", "hand sanitizer") %in% fruits
## [1]  TRUE  TRUE FALSE
```

named vectors (see Annual Estimates… table from census.gov)

- Define a named vector

```
state_populations <- c(
    Alabama = 4903185, Alaska = 731545, Arizona = 7278717, Arkansas = 3017804,
    California = 39512223, Colorado = 5758736, Connecticut = 3565287,
    Delaware = 973764, `District of Columbia` = 705749, Florida = 21477737,
    Georgia = 10617423, Hawaii = 1415872, Idaho = 1787065, Illinois = 12671821,
    Indiana = 6732219, Iowa = 3155070, Kansas = 2913314, Kentucky = 4467673,
    Louisiana = 4648794, Maine = 1344212, Maryland = 6045680, Massachusetts = 6892503,
    Michigan = 9986857, Minnesota = 5639632, Mississippi = 2976149,
    Missouri = 6137428, Montana = 1068778, Nebraska = 1934408, Nevada = 3080156,
    `New Hampshire` = 1359711, `New Jersey` = 8882190, `New Mexico` = 2096829,
    `New York` = 19453561, `North Carolina` = 10488084, `North Dakota` = 762062,
    Ohio = 11689100, Oklahoma = 3956971, Oregon = 4217737, Pennsylvania = 12801989,
    `Rhode Island` = 1059361, `South Carolina` = 5148714, `South Dakota` = 884659,
    Tennessee = 6829174, Texas = 28995881, Utah = 3205958, Vermont = 623989,
    Virginia = 8535519, Washington = 7614893, `West Virginia` = 1792147,
    Wisconsin = 5822434, Wyoming = 578759
)
```

- Computations on named vectors

```
## US population
sum(state_populations)
## [1] 328239523

## smallest states
head(sort(state_populations))
##              Wyoming              Vermont District of Columbia
##               578759               623989               705749
##               Alaska         North Dakota         South Dakota
##               731545               762062               884659

## largest states
head(sort(state_populations, decreasing = TRUE))
##   California        Texas      Florida     New York Pennsylvania     Illinois
##     39512223     28995881     21477737     19453561     12801989     12671821

## states with more than 10 million people
big <- state_populations[state_populations > 10000000]
big
```

```
##     California        Florida        Georgia       Illinois       New York
##       39512223       21477737       10617423       12671821       19453561
## North Carolina           Ohio   Pennsylvania          Texas
##       10488084       11689100       12801989       28995881
names(big)
## [1] "California"     "Florida"        "Georgia"        "Illinois"
## [5] "New York"       "North Carolina" "Ohio"           "Pennsylvania"
## [9] "Texas"
```

- Subset by name

```
## populations of California and New York
state_populations[c("California", "New York")]
## California   New York
##   39512223   19453561
```

1.5.3 Weekend activities (25 minutes; Shawn)

Writing *R* scripts

R scripts are convenient text files that we can use to save one or more lines of *R* syntax. Over the weekend you will get some experience working with *R* scripts. The example below will help you be a bit more prepared.

- In RStudio, click `File --> New File --> R Script` to create a new script file and open it in the editor.

If you've followed the daily coding activities throughout the week, you should have some R code that keeps track of your daily activities.

- If so, enter that code into your *R* script now.
- Otherwise, feel free to use the code below. Look for a `copy to clipboard` icon in the top-right of the code block. To copy the code block to your *R* script:
  - Click on the `copy to clipboard` icon
  - Place your cursor in your *R* script
  - Click `Edit --> Paste`:

```
## ========================================================================
## day 1 information
## ========================================================================
day1_activity = c("breakfast",
                  "check e-mail",
                  "projects",
                  "conference call",
                  "teams meeting",
                  "lunch",
```

```
                  "conference call",
                  "webinar")
day1_is_work = c(FALSE,TRUE,TRUE,TRUE,TRUE,FALSE,TRUE,TRUE)
day1_minutes = c(30, 75, 120, 30, 60, 30, 60, 120)

n = length(day1_activity)
day1_total_hours = sum(day1_minutes) / 60
day1_work_hours = sum(day1_minutes[day1_is_work == TRUE]) / 60

cat("Total time recorded for day 1 : ", day1_total_hours,
    "hours, over", n, "activities\n")
cat("Total time working  for day 1 : ", day1_work_hours, "hours \n\n")
```

Recall the discussion of factors and levels in Day 3; the code below leverages this but adds another level named `independent work`.

- If you've already got code to assign factors and levels to your daily activity, enter that code into your *R* script now.
- Otherwise, feel free to use the code below via the `copy to clipboard` procedure outlined above:

```
## =========================================================================
## Kelly, Morgan, and Matott's classification strategy
## =========================================================================
kmm_levels = c("connect",
               "exercise",
               "consult",
               "hobby",
               "essential",
               "independent work")
## manually map day 1 activity to appropriate kmm_levels
day1_classes = factor(
    c("essential", "connect", "independent work",
      "connect", "connect",  "essential",
      "connect", "consult"),
    levels = kmm_levels
)
```

On day 3 you also got some experience working with dates. The code below stamps our day 1 activity data with an appropriately formatted date.

- If you've already got code to assign dates your daily activity, enter that code into your R script now.
- Otherwise, feel free to use the code below via the `copy to clipboard` procedure outlined above:

```
## ========================================================================
## Assign dates
## ========================================================================
day1_dates = rep("04-13-2020",length(day1_activity))
day1_dates = as.Date(day1_dates,format = "%m-%d-%Y")
```

[OPTIONAL ADVANCED MATERIAL]

Earlier today Dr. Morgan touched on named vectors. We can leverage named vectors to create a more general mapping between activities and levels. The code for this is given below. Try it and compare the result to your manual mapping!

```
## kmm_map is a named vector that maps activities to categories
kmm_map = c("breakfast"       = "essential",
            "check e-mail"    = "connect",
            "projects"        = "independent work",
            "requests"        = "independent work",
            "conference call" = "connect",
            "teams meeting"   = "connect",
            "lunch"           = "essential",
            "webinar"         = "consult",
            "walk"            = "exercise")
day1_classes = factor(kmm_map[day1_activity], levels = kmm_levels)
```

Saving *R* scripts

If you've been following along you should now have an *R* script that contains a bunch of code for keeping track of your daily activity log. Let's save this file:

- In RStudio, place your cursor anywhere in the script file
- click `File --> Save` (or press `CTRL+S`)
  - Name your file something like `daily_activity.R`.

Running *R* scripts

Now that we've created an *R* script you may be wondering "How do I run the code in the script?" There's actually a few ways to do this:

Option #1 (Run)

- Highlight the first block of the code (e.g the part where you recorded day 1 activity and maybe calculated amount of time worked).
- Click the `--> Run` icon in the top-right portion of the script editor window.
  - This will run the highlighted block of code. The output will appear in the RStudio console window along with an echo of the code itself.

Option #2 (Source)

- Click on the `--> Source` icon just to the right of the `--> Run` icon.
- This will run the entire script.
- Equivalent to entering into the console

```
source("daily_activity.R")
```

- Only the output generated by `print()` and `cat()` will appear in the RStudio console (i.e. the code in the script is not echoed to the console).

Option #3 (Source with Echo)

- Click on the downward pointing arrowhead next to the source buton to open a dropdown menu
- In the dropdown menu, sectect `Source with Echo`
- This will run the entire script and the code in the script will be echoed to the RStudio console along with any output generated by `print()` and `cat()`.
- The echoed source and the normal output are not color-coded like they are when using the `--> Run` button.
- Equivalent to running

```
source("daily_activity.R", echo = TRUE, max = Inf)
```

Saving data

It can be useful to save objects created in an *R* script as a data file. These data files can be loaded or re-loaded into a new or existing *R* session.

For example, let's suppose you had an *R* script that mined a trove of Twitter feeds for sentiment data related to government responses to COVID-19. Suppose you ran the script for serveral weeks and collected lot's of valuable data into a bunch of vectors. Even though the *R* code is saved as a script file, the data that the script is collecting would be lost once script stops running. Furthermore, due to the temporal nature of Twitter feeds, you wouldn't be able to collect the same data by simply re-running the script. Luckily, *R* provides several routines for saving and loading objects. Placing the appropriate code in your *R* script will ensure that your data is preserved even after the script stops running.

Saving individual *R* objects

R supports storing a single *R* object as an `.rds` file. For example, the code below saves the `day1_activity` vector to an `.rds` file. The `saveRDS()` function is the workhorse in this case and the `setwd()`, `getwd()`, and `file.path()` commands allow us to conveniently specify a name and location for the data file:

```
## ========================================================================
## creating .rds data files (for saving individual objects)
## ========================================================================
setwd("C:/Matott/MyQuarantine")
my_rds_file = file.path(getwd(), "day1_activity.rds")
my_rds_file # print value -- sanity check
saveRDS(day1_activity, my_rds_file)
```

Loading individual *R* objects

The complement to the `saveRDS()` function is the `readRDS()` function. It loads the *R* object stored in the specified file. In the example below a data file is loaded and stored as an object named `day1_activity_loaded`. Compare this object to the existing `day1_activity` object - they should be the same!

```
## ========================================================================
## Reading .rds data files (for loading individual objects)
## ========================================================================
setwd("C:/Matott/MyQuarantine")
my_rds_file = file.path(getwd() ,"day1_activity.rds")
my_rds_file
day1_activity_loaded = readRDS(my_rds_file) # now load from disk
```

Saving multiple *R* objects

The `save()` function will save one or more objects into a `.Rdata` file (these are also known as `session` files). The example below saves various `day1` and related factor-level objects to an `.Rdata` file.

```
## ========================================================================
## creating .RData files (for saving multiple objects)
## ========================================================================
setwd("C:/Matott/MyQuarantine")
my_rdata_file = file.path(getwd(), "day1.rdata")
save(kmm_levels, kmm_map,
     day1_activity, day1_classes, day1_dates, day1_is_work,
     day1_minutes, day1_total_hours, day1_work_hours,
     file = my_rdata_file)
```

If you have many objects that you want to save, listing them all can be tedious. Fortunately, the `ls()` command provides a list of all objects in the current *R* session. The results of `ls()` can be passed along to the `save()` command and this will result in all objects being saved. An example of the required syntax is given below.

```
setwd("C:/Matott/MyQuarantine")
my_rdata_file = file.path(getwd(), "day1.rdata")
```

```
save(list = ls(), file = my_rdata_file)
```

Loading multiple *R* objects

The complement to the `save()` function is the `load()` function. This will load all objects stored in an `.Rdata` file into the current *R* session. Example syntax is given below:

```
## =========================================================================
## Reading .RData files (for loading multiple objects)
## =========================================================================
setwd("C:/Matott/MyQuarantine")
my_rdata_file = file.path(getwd(), "daily_activity.rdata")
load(my_rdata_file) # reload
```

It is also possible to load an `.Rdata` file using the RStudio interface.

- Click `Session --> Load Workspace ...`
- A file browser dialog will open
- Navigate to the .rdata file and select

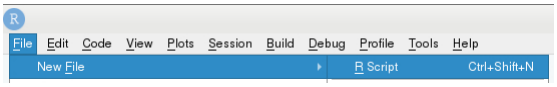
Wrapping up day 5

Today we reviewed the concepts that you worked with throughout the week during your independent activity. We also troubleshooted any problems or questions that may have come up during this time. Finally, we previewed the creation and use of *R* scripts and learned about saving and loading objects. Over the weekend you will gain some more experience with these topics.

1.6 Day 6: *R* scripts

Some of you may have already started saving your R commands as script files. As the material gets more complicated (and more interesting) everyone will want to start doing this. Here is an example to get you started:

- Recall that we can create a script file in RStudio, click "File –> New File –> R Script" to create a new script file and open it in the editor



- By convention, R scripts have a .R exstension (e.g. my_script.R)
  - In RStudio, click into your untitled script and click "File –> Save"
  - Name your file something fun like `my_first_script.R` and save it
- Use the `#` character for comments. Enter the following into your R Script file:

```
## This is my first R script
```

- Enter each command on a separate line. It's also possible to enter multiple (short!) commands on a single line, separated by a semi-colon ;

```
x = "Hello world!"
y = 'Today is'; d = format(Sys.Date(), "%b %d, %Y")
cat(x, y, d)
```

- Use the "Run" button in RStudio to run the highlighted portion of an R script file. Try this on your simple R Script.


```
x = "Hello world!"; y = 'Today is'; d = format(Sys.Date(),"%b %d, %Y")
cat(x, y, d, "\n")
## Hello world! Today is May 22, 2020
```

- Alternatively, use "Run –> Run All" to run an entire script file.

For today's exercise, create a script file that summarizes your quarantine activities over several days. Use comments, white space (blank lines and spaces), and variable names to summarize each day. Here's what I've got…

```
## 'classification' factor levels
levels <- c("connect", "exercise", "consult", "hobby", "essential")

## Quarantine log, day 1

activity_day_1 <-
    c("check e-mail", "breakfast", "conference call", "webinar", "walk")
minutes_day_1 <- c(20, 30, 60, 60, 60)
is_work_day_1 <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
classification_day_1 <- factor(
    c("connect", "essential", "connect", "consult", "exercise"),
    levels = levels
)
date_day_1 <- as.Date(rep("04-14-2020", length(activity_day_1)), "%m-%d-%Y")

## Quarantine log, day 2

activity_day_2 <-
    c("check e-mail", "breakfast", "conference call", "webinar", "read a book")
minutes_day_2 <- c(20, 30, 60, 60, 60)
is_work_day_2 <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
classification_day_2 <- factor(
    c("connect", "essential", "connect", "consult", "hobby"),
    levels = levels
```

```
)
date_day_2 <- as.Date(rep("04-15-2020", length(activity_day_2)), "%m-%d-%Y")

## Quarantine log, day 3

activity_day_3 <-
    c("check e-mail", "breakfast", "webinar", "read a book")
minutes_day_3 <- c(20, 30, 60, 60)
is_work_day_3 <- c(TRUE, FALSE, TRUE, FALSE)
classification_day_3 <- factor(
    c("connect", "essential", "connect", "consult", "hobby"),
    levels = levels
)
date_day_3 <- as.Date(rep("04-16-2020", length(activity_day_3)), "%m-%d-%Y")
```

Try concatenating these values, e.g.,

```
activity <- c(activity_day_1, activity_day_2, activity_day_3)
activity
##  [1] "check e-mail"    "breakfast"       "conference call" "webinar"
##  [5] "walk"            "check e-mail"    "breakfast"       "conference call"
##  [9] "webinar"         "read a book"     "check e-mail"    "breakfast"
## [13] "webinar"         "read a book"
```

Save your script, quit *R* and *RStudio*, and restart *R*. Re-open and run the script to re-do your original work.

Think about how this makes your work *reproducible* from one day to the next, and how making your scientific work reproducible would be advantageous.

1.7 Day 7: Saving data

We've defined these variables

```
activity <- c("check e-mail", "breakfast", "conference call", "webinar", "walk")
minutes <- c(20, 30, 60, 60, 60)
is_work <- c(TRUE, FALSE, TRUE, TRUE, FALSE)

levels <- c("connect", "exercise", "consult", "hobby", "essential")
classification <- factor(
    c("connect", "essential", "connect", "consult", "exercise"),
    levels = levels
)

dates <- rep("04-14-2020", length(activity))
date <- as.Date(dates, format = "%m-%d-%Y")
```

Individual variables can be saved to a file.

- Define the *path* to the file. The file extension is, by convention, '.rds'. We'll use a temporary location

```
temporary_file_path <- tempfile(fileext = ".rds")
```

  ...but we could have chosen the destination interactively

```
interactive_file_path <- file.choose(new = TRUE)
```

  ...or provided path relative to the 'current working directory', or an absolute file path (use '/' to specify paths on all operating systems, including Windows)

```
getwd()
relative_file_path <- "my_activity.rds"
absolute_file_path_on_macOS <- "/Users/ma38727/my_activity.rda"
```

- use `saveRDS()` to save a single object to a file

```
saveRDS(activity, temporary_file_path)
```

- use `readRDS()` to read the object back in

```
activity_from_disk <- readRDS(temporary_file_path)
activity_from_disk
## [1] "check e-mail"    "breakfast"       "conference call" "webinar"
## [5] "walk"
```

Use `save()` and `load()` to save and load several objects.

- Use `.RDaata` as the file extension.Usually we would NOT save to a temporary location, because the temporary location would be deleted when we ended our *R* session.

```
temporary_file_path <- tempfile(fileext = ".RData")
save(activity, minutes, file = temporary_file_path)
```

- Remove the objects from the *R* session, and verify that they are absent

```
rm(activity, minutes)
try(activity) # fails -- object not present
## Error in try(activity) : object 'activity' not found
```

- Load the saved objects

```
load(temporary_file_path)
activity
## [1] "check e-mail"    "breakfast"       "conference call" "webinar"
## [5] "walk"
```

As an exercise...

- Chose a location to save your data, e.g., in the current working direcotry

```
getwd()    # Where the heck are we?
## [1] "/Users/ma38727/a/github/QuaRantine"
my_file_path <- "my_quaRantine.RData"
```

- Save the data

```
save(activity, minutes, is_work, classification, date, file = my_file_path)
```

- Now the moment of truth. Quit *R* without saving your workspace

```
quit(save = FALSE)
```

- Start a new session of *R*, and verify that your objects are not present

```
ls() # list objects available in the '.GlobalEnv' -- there should be none
## character(0)
try(activity) # nope, not there...
## Error in try(activity) : object 'activity' not found
```

- Create a path to the saved data file

```
my_file_path <- "my_quaRantine.RData"
```

- Load the data and verify that it is correct

```
load(my_file_path)
activity
## [1] "check e-mail"    "breakfast"       "conference call" "webinar"
## [5] "walk"
minutes
## [1] 20 30 60 60 60
is_work
## [1]  TRUE FALSE  TRUE  TRUE FALSE
date
## [1] "2020-04-14" "2020-04-14" "2020-04-14" "2020-04-14" "2020-04-14"
classification
## [1] connect   essential connect   consult   exercise
## Levels: connect exercise consult hobby essential
```

See you in zoom on Monday!

Chapter 2

The data frame

2.1 Day 8 (Monday) Zoom check-in

Logistics

- Remember to use the QuaRantine Microsoft Team. Your Roswell credentials are required, and you must have been invited (by Adam.Kisailus at RoswellPark.org)
- We're thinking of having a 'networking' hour after Friday's class (so 3pm and after) where we'll break into smaller groups (if necessary) and provide an opportunity for people to turn on their video and audio so that we can increase the amount of intereaction. Likely the first networking hour will be a round of introductions / what you hope to get out of the course / etc., and maybe brief discussion of topics arising.

Review and troubleshoot (15 minutes)

Saving and loading objects

Scripts

The data frame (40 minutes)

Concept

Recall from Day 1:

- Data frames are handy containers for experimental data.
- Like a spreadsheet, a data frame has rows and columns
- The columns of a data frame contain measurements describing each individual

  - Each measurement could refer to a different type of object (numeric, string, etc.)
  - Measurements could be physical observations on samples, e.g., `height`, `weight`, `age`, `minutes` an activity lasts, etc.
  - Measurements might also describe how the row is classified, e.g., `activity`, `is work?`, `classification`, `date`, etc.

- The rows of a data frame represent a 'tuple' of measurements corresponding to an experimental observation, e.g.,

  - Note: you must ensure units are consistent across tuples!

- Rows and columns can be assigned names.

Create a simple data frame

```
heights <- c(72, 65, 68)
weights <- c(190, 130, 150)
ages <-  c(44, 35, 37)
df <- data.frame(heights, weights, ages)
df
##   heights weights ages
## 1      72     190   44
## 2      65     130   35
## 3      68     150   37
```

It's possible to update the column names, and to provide row names…

```
named_df <- data.frame(heights, weights, ages)
colnames(named_df) <- c("hgt_inches", "wgt_lbs", "age_years")
rownames(named_df) <- c("John Doe", "Pat Jones", "Sara Grant")
named_df
##             hgt_inches wgt_lbs age_years
## John Doe            72     190        44
## Pat Jones           65     130        35
## Sara Grant          68     150        37
```

…but it's often better practice to name columns at time of creation, and to store all information as columns (rather than designating one column as a 'special' row name)

- Here's our first attempt

```
data.frame(
    person = c("John Doe", "Pat Jones", "Sara Grant"),
    hgt_inches = heights, wgt_lbs = weights, age_years = ages
)
##       person hgt_inches wgt_lbs age_years
## 1   John Doe         72     190        44
```

```
## 2  Pat Jones         65     130        35
## 3 Sara Grant         68     150        37
```

- It's unsatisfactory because by default *R* treats character vectors as `factor`. We'd like them to plain-old character vectors. To accomplish this, we add the `stringsAsFactors = FALSE` argument

```
df <- data.frame(
    person = c("John Doe", "Pat Jones", "Sara Grant"),
    hgt_inches = heights, wgt_lbs = weights, age_years = ages,
    stringsAsFactors = FALSE
)
df
##       person hgt_inches wgt_lbs age_years
## 1   John Doe         72     190        44
## 2  Pat Jones         65     130        35
## 3 Sara Grant         68     150        37
```

Adding and deleting rows

Adding rows

- Add a row with `rbind()`

```
more_people <- c("Bob Kane", "Kari Patra", "Sam Groe")
more_heights <- c(61, 68, 70)
more_weights <- c(101, 134, 175)
more_ages <-  c(13, 16, 24)
more_df <- data.frame(
    person = more_people,
    hgt_inches = more_heights, wgt_lbs = more_weights, age_years = more_ages,
    stringsAsFactors = FALSE
)

df_all <- rbind(df, more_df)
df_all
##       person hgt_inches wgt_lbs age_years
## 1   John Doe         72     190        44
## 2  Pat Jones         65     130        35
## 3 Sara Grant         68     150        37
## 4   Bob Kane         61     101        13
## 5 Kari Patra         68     134        16
## 6   Sam Groe         70     175        24
```

- *R* often has more than one way to perform an operation. We'll see `add_rows()` later in the course.

Delete rows using a logical vector…

- Create a logical or numeric index indicating the rows to be deleted

```
## suppose the study has some dropouts ....
dropouts <- c("Bob Kane", "John Doe")

## create a logical vector indicating which rows should be dropped
drop <- df_all$person %in% dropouts

## ...but we actually want to know which rows to `keep`
keep <- !drop
```

- Subset the data frame with the logical vector indicating the rows we would like to keep

```
df_all[keep,]
##       person hgt_inches wgt_lbs age_years
## 2  Pat Jones         65     130        35
## 3 Sara Grant         68     150        37
## 5 Kari Patra         68     134        16
## 6   Sam Groe         70     175        24
```

...or a numeric vector

- Create a vector containing the rows to be deleted

```
# suppose the study has some dropouts ....
dropouts = c(2, 3)
```

- Use a minus sign - to indicated that these rows should be *dropped*, rather than kept

```
df_all # referesh my memory about df contents ....
##       person hgt_inches wgt_lbs age_years
## 1   John Doe         72     190        44
## 2  Pat Jones         65     130        35
## 3 Sara Grant         68     150        37
## 4   Bob Kane         61     101        13
## 5 Kari Patra         68     134        16
## 6   Sam Groe         70     175        24
df_remaining <- df_all[-dropouts, ]

df_remaining # items 2 and 3 are dropped!
##       person hgt_inches wgt_lbs age_years
## 1   John Doe         72     190        44
## 4   Bob Kane         61     101        13
## 5 Kari Patra         68     134        16
## 6   Sam Groe         70     175        24
```

Some useful data frame operations

Try these out on your simple data frames `df` and `named_df`:

- `str(df)` `# structure (NOT string!)(sorry Python programmers ;)`
- `dim(df)` `# dimensions`
- `View(df)` `# open tabular view of data frame`
- `head(df)` `# first few rows`
- `tail(df)` `# last few rows`
- `names(df)` `# column names`
- `colnames(df)` `# column names`
- `rownames(df)` `# row names`

Writing, reading, and spreadhseets

Saving a `data.frame`

- We *could* save the `data.frame` as an *R* object, using the methods from quarantine day 7
- Often, better practice (e.g., to make it easy to share data with others in our lab) is to save data as a text file
- A 'csv' file is one example
  - A plain text file
  - The first line contains column names
  - Each line of the text file represents a row of the data frame
  - Columns within a row are separated by a comma, `,`
- Example: save `df_all` to a temporary file location

```
file <- tempfile() # temporary file
## file <- file.choose()
## file <- "df_all.csv"
## file <- "/Users/ma38737/MyQuarantine/df_all.csv"
write.csv(df_all, file, row.names = FALSE)
```

- now, read the data back in from the temporary location

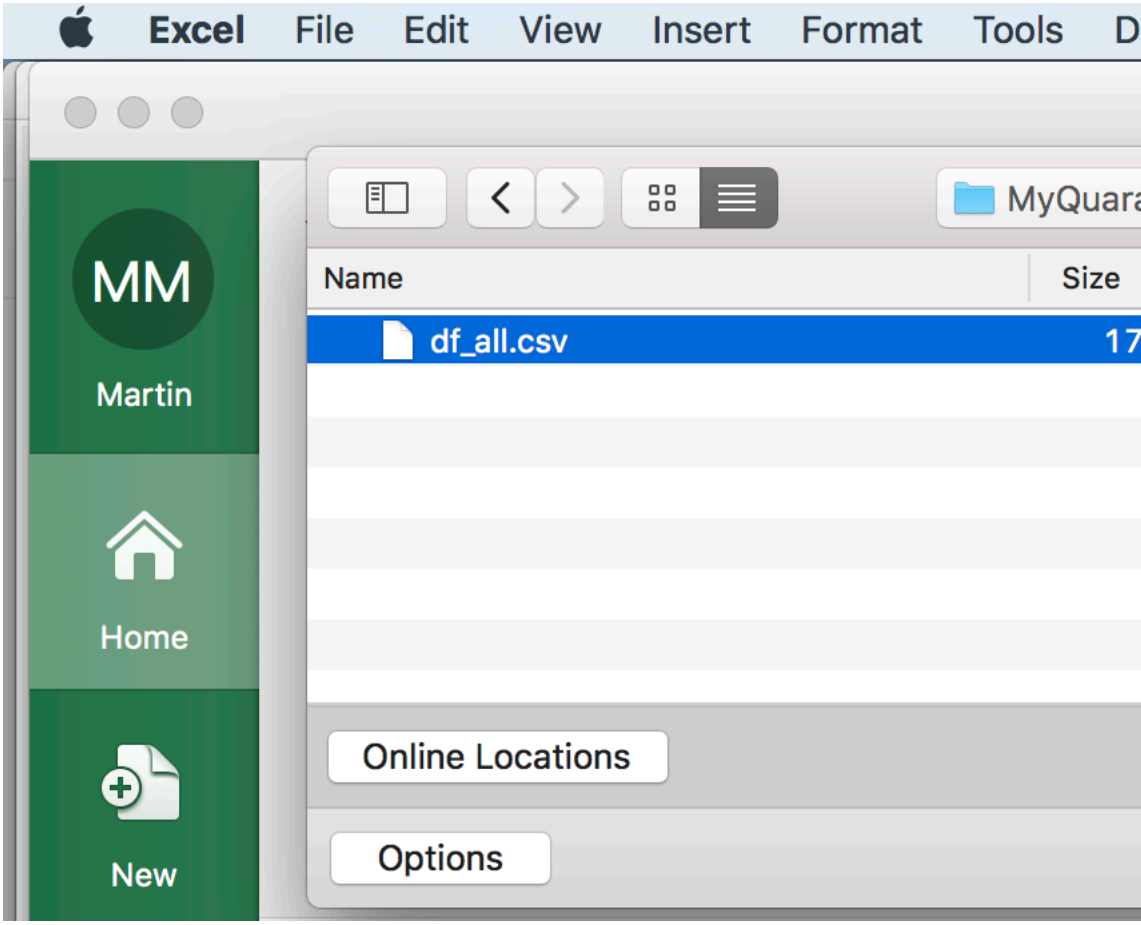
```
df_all_from_file <- read.csv(file, stringsAsFactors = FALSE)
df_all_from_file
##       person hgt_inches wgt_lbs age_years
## 1   John Doe         72     190        44
## 2  Pat Jones         65     130        35
## 3 Sara Grant         68     150        37
## 4   Bob Kane         61     101        13
## 5 Kari Patra         68     134        16
## 6   Sam Groe         70     175        24
```

R and spreadsheets

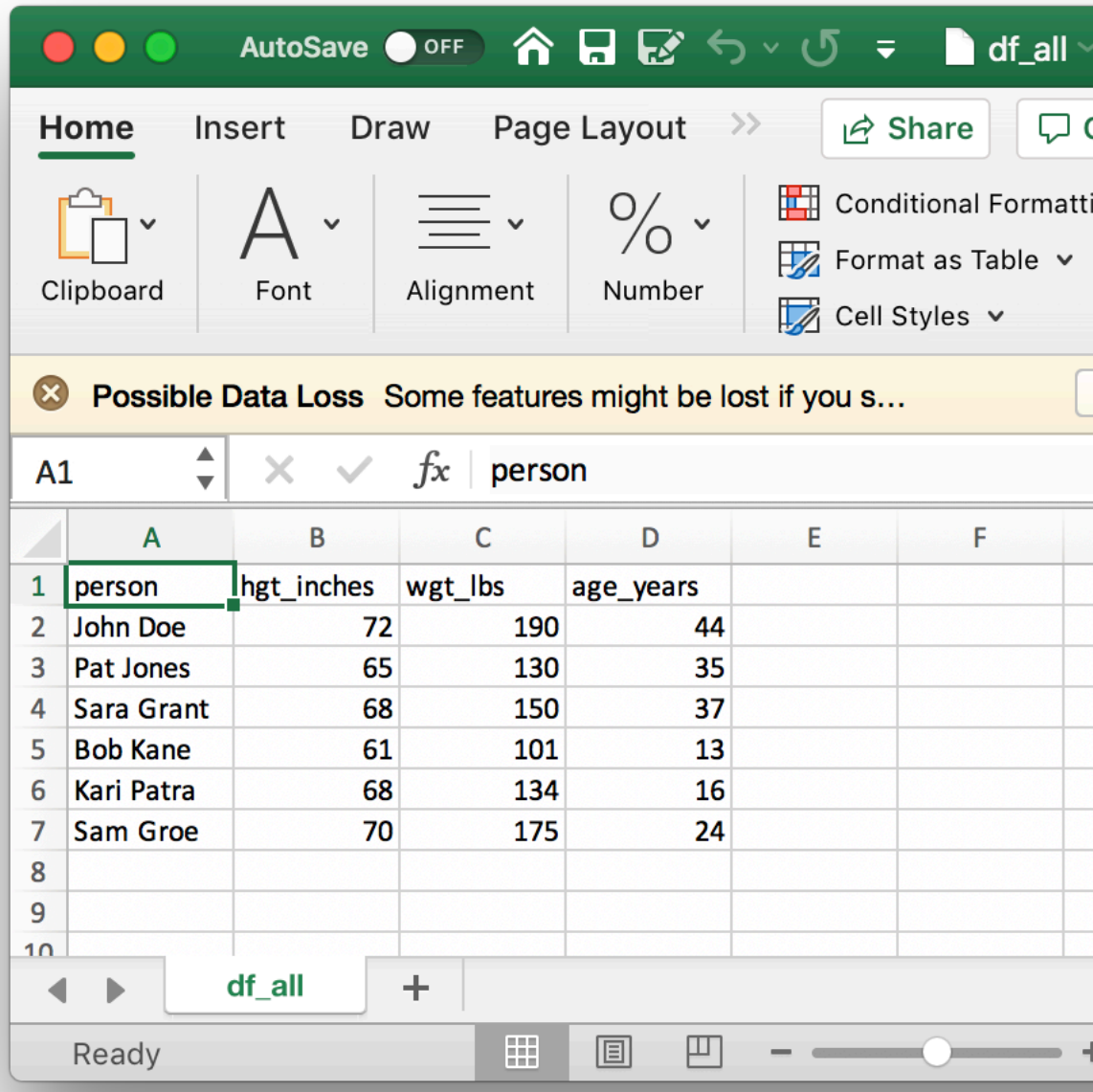
- A CSV file is a common way to move data from *R* to a spreadsheet, and vice versa. Following along with the example above, write `df_all` to a CSV file.

```
file <- "/Users/ma38727/MyQuarantine/df_all.csv" # a location on (my) disk
write.csv(df_all, file, row.names = FALSE)
```

- Now open a spreadsheet application like Excel and navigate to the directory containing the file

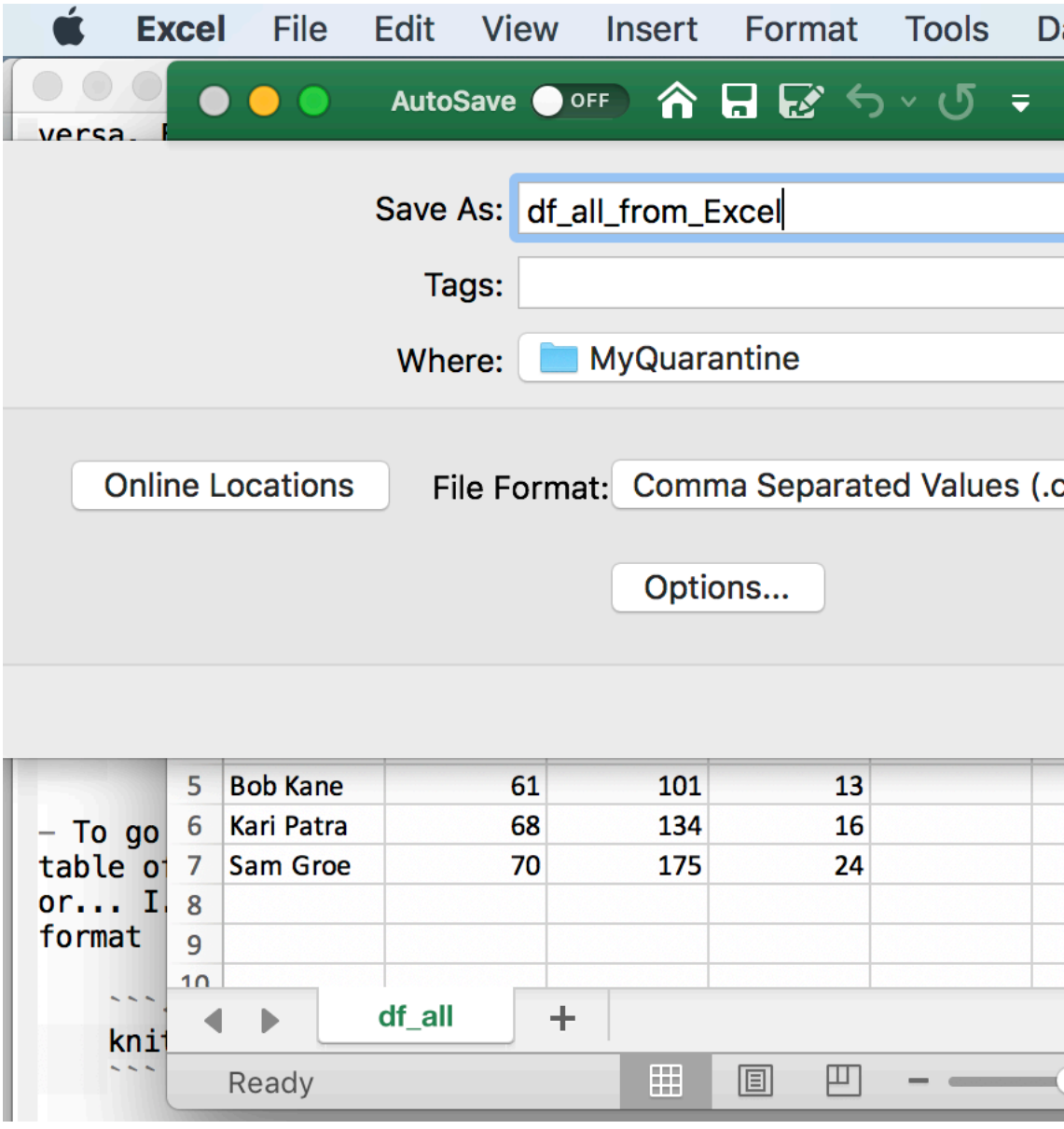


- … and open the file



	A	B	C	D
1	person	hgt_inches	wgt_lbs	age_years
2	John Doe	72	190	44
3	Pat Jones	65	130	35
4	Sara Grant	68	150	37
5	Bob Kane	61	101	13
6	Kari Patra	68	134	16
7	Sam Groe	70	175	24
8				
9				

- To go from Excel to *R*, make sure your spreadsheet is a simple rectangular table of rows and columns, without 'merged' cells or fancy column formatting, or... i.e., your spreadsheet should be as simple as the one we imported from *R*. Then save the file in CSV format



- ...and import it into *R*

```
df_all_from_Excel <- read.csv(
    "/Users/ma38727/MyQuarantine/df_all_from_Excel.csv",
    stringsAsFactors = FALSE
)
```

An alternative way of working with `data.frame()`

- `with()`: column selection and computation
- `within()`: update or add columns
- `subset()`: row and column subset
- Our quarantine log, day 1

```
activity <- c("check e-mail", "breakfast", "conference call", "webinar", "walk")
minutes <- c(20, 30, 60, 60, 60)
is_work <- c(TRUE, FALSE, TRUE, TRUE, FALSE)

levels <- c("connect", "exercise", "consult", "hobby", "essential")
classification <- factor(
    c("connect", "essential", "connect", "consult", "exercise"),
    levels = levels
)

dates <- rep("04-14-2020", length(activity))
date <- as.Date(dates, format = "%m-%d-%Y")

log <- data.frame(
    activity, minutes, is_work, classification, date,
    stringsAsFactors = FALSE
)
log
##          activity minutes is_work classification       date
## 1    check e-mail      20    TRUE        connect 2020-04-14
## 2       breakfast      30   FALSE      essential 2020-04-14
## 3 conference call      60    TRUE        connect 2020-04-14
## 4         webinar      60    TRUE        consult 2020-04-14
## 5            walk      60   FALSE       exercise 2020-04-14
```

Summarization

- Use `with()` to simplify variable reference
- Create a new `data.frame()` containing the summary

```
with(log, {
    data.frame(
        days_in_quarantine = length(unique(date)),
```

```
        total_minutes = sum(minutes),
        work_activities = sum(is_work),
        other_activities = sum(!is_work)
    )
})
##   days_in_quarantine total_minutes work_activities other_activities
## 1                  1           230               3                2
```

Summarization by group

- aggregate()

```
## minutes per day spent on each activity, from the quarantine_log
aggregate(minutes ~ activity, log, sum)
##          activity minutes
## 1       breakfast      30
## 2    check e-mail      20
## 3 conference call      60
## 4            walk      60
## 5         webinar      60

## minutes per day spent on each clasasification
aggregate(minutes ~ classification, log, sum)
##   classification minutes
## 1        connect      80
## 2       exercise      60
## 3        consult      60
## 4      essential      30

## non-work activities per day
aggregate(!is_work ~ date, log, sum)
##         date !is_work
## 1 2020-04-14        2
```

This week's activities (5 minutes)

Goal: retrieve and summarize COVID 19 cases in Erie county and nationally

2.2 Day 9: Creation and manipulation

Creation

Last week we created vectors summarizing our quarantine activities

```
activity <- c("check e-mail", "breakfast", "conference call", "webinar", "walk")
minutes <- c(20, 30, 60, 60, 60)
is_work <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
```

```
levels <- c("connect", "exercise", "consult", "hobby", "essential")
classification <- factor(
    c("connect", "essential", "connect", "consult", "exercise"),
    levels = levels
)

dates <- rep("04-14-2020", length(activity))
date <- as.Date(dates, format = "%m-%d-%Y")
```

Each of these vectors is the same length, and are related to one another in a specific way – the first element of `activity`, 'check e-mail', is related to the first element of `minutes`, '20', and to `is_work`, etc.

Use `data.frame()` to construct an object containing each of these vectors

- Each argument to `data.frame()` is a vector representing a column
- The `stringsAsFactors = FALSE` argument says that character vectors should NOT be automatically coerced to factors

```
activities <- data.frame(
    activity, minutes, is_work, classification, date,
    stringsAsFactors = FALSE
)
activities
##          activity minutes is_work classification       date
## 1    check e-mail      20    TRUE        connect 2020-04-14
## 2       breakfast      30   FALSE      essential 2020-04-14
## 3 conference call      60    TRUE        connect 2020-04-14
## 4         webinar      60    TRUE        consult 2020-04-14
## 5            walk      60   FALSE       exercise 2020-04-14
```

- We can query the object we've created for its `class()`, `dim()`ensions, take a look at the `head()` or `tail()` of the object, etc. `names()` returns the column names.

```
class(activities)
## [1] "data.frame"
dim(activities)     # number of rows and columns
## [1] 5 5
head(activities, 3) # first three rows
##          activity minutes is_work classification       date
## 1    check e-mail      20    TRUE        connect 2020-04-14
## 2       breakfast      30   FALSE      essential 2020-04-14
## 3 conference call      60    TRUE        connect 2020-04-14
names(activities)
## [1] "activity"       "minutes"        "is_work"        "classification"
## [5] "date"
```

Column selection

Use `[` to select rows and columns

- `activities` is a two-dimensional object
- Subset the data to contain the first and third rows and the first and fourth columns

```
activities[c(1, 3), c(1, 4)]
##          activity classification
## 1    check e-mail        connect
## 3 conference call        connect
```

- Subset columns by name

```
activities[c(1, 3), c("activity", "is_work")]
##          activity is_work
## 1    check e-mail    TRUE
## 3 conference call    TRUE
```

- Subset only by row or only by column by omiting the subscript index for that dimension

```
activities[c(1, 3), ]                  # all columns for rows 1 and 3
##          activity minutes is_work classification       date
## 1    check e-mail      20    TRUE        connect 2020-04-14
## 3 conference call      60    TRUE        connect 2020-04-14
activities[, c("activity", "minutes")] # all rows for columns 1 and 2
##          activity minutes
## 1    check e-mail      20
## 2       breakfast      30
## 3 conference call      60
## 4         webinar      60
## 5            walk      60
```

- Be careful when selecting a single column!
  - By default, *R* returns a *vector*

```
activities[, "classification"]
## [1] connect   essential connect   consult   exercise
## Levels: connect exercise consult hobby essential
```

  - Use `drop = FALSE` to return a `data.frame`

```
activities[, "classification", drop = FALSE]
##   classification
## 1        connect
```

```
## 2      essential
## 3        connect
## 4        consult
## 5       exercise
```

Use `$` or `[[` to select a column

- Selection of individual columns as vectors is easy

```
activities$classification
## [1] connect   essential connect   consult   exercise
## Levels: connect exercise consult hobby essential
```

- An alternative, often used in scripts, is to use `[[`, which requires the name of a variable provided as a character vector

```
activities[["classification"]]
## [1] connect   essential connect   consult   exercise
## Levels: connect exercise consult hobby essential

colname <- "classification"
activities[[colname]]
## [1] connect   essential connect   consult   exercise
## Levels: connect exercise consult hobby essential
```

Column selection and subsetting are often combined, e.g., to create a `data.frame` of work-related activities, or work-related activities lasting 60 minutes or longer

```
work_related_activities <- activities[ activities$is_work == TRUE, ]
work_related_activities
##          activity minutes is_work classification       date
## 1    check e-mail      20    TRUE        connect 2020-04-14
## 3 conference call      60    TRUE        connect 2020-04-14
## 4         webinar      60    TRUE        consult 2020-04-14

row_idx <- activities$is_work & (activities$minutes >= 60)
activities[row_idx,]
##          activity minutes is_work classification       date
## 3 conference call      60    TRUE        connect 2020-04-14
## 4         webinar      60    TRUE        consult 2020-04-14
```

Adding or updating columns

Use `$` or `[` or `[[` to add a new column,

```
activities$is_long_work <- activities$is_work & (activities$minutes >= 60)
activities
```

```
##          activity minutes is_work classification       date is_long_work
## 1    check e-mail      20    TRUE        connect 2020-04-14        FALSE
## 2       breakfast      30   FALSE      essential 2020-04-14        FALSE
## 3 conference call      60    TRUE        connect 2020-04-14         TRUE
## 4         webinar      60    TRUE        consult 2020-04-14         TRUE
## 5            walk      60   FALSE       exercise 2020-04-14        FALSE

## ...another way of doing the same thing
activities[["is_long_work"]] <- activities$is_work & (activities$minutes >= 60)

## ...and another way
activities[,"is_long_work"] <- activities$is_work & (activities$minutes >= 60)
```

Columns can be updated in the same way

```
activities$activity <- toupper(activities$activity)
activities
##          activity minutes is_work classification       date is_long_work
## 1    CHECK E-MAIL      20    TRUE        connect 2020-04-14        FALSE
## 2       BREAKFAST      30   FALSE      essential 2020-04-14        FALSE
## 3 CONFERENCE CALL      60    TRUE        connect 2020-04-14         TRUE
## 4         WEBINAR      60    TRUE        consult 2020-04-14         TRUE
## 5            WALK      60   FALSE       exercise 2020-04-14        FALSE
```

Reading and writing

Create a file path to store a 'csv' file. From day 7, the path could be temporary, chosen interactively, a relative path, or an absolute path

```
## could be any of these...
##
## interactive_file_path <- file.choose(new = TRUE)
## getwd()
## relative_file_path <- "my_activity.rds"
## absolute_file_path_on_macOS <- "/Users/ma38727/my_activity.rda"
##
## ... but we'll use
temporary_file_path <- tempfile(fileext = ".csv")
```

Use `write.csv()` to save the data.frame to disk as a plain text file in 'csv' (comma-separated value) format. The `row.names = FALSE` argument means that the row indexes are not saved to the file (row names are created when data is read in using `read.csv()`).

```
write.csv(activities, temporary_file_path, row.names = FALSE)
```

If you wish, use RStudio File -> Open File to navigate to the location where

you saved the file, and open it. You could also open the file in Excel or other spreadsheet. Conversely, you can take an Excel sheet and export it as a csv file for reading into *R*.

Use `read.csv()` to import a plain text file formatted as csv

```
imported_activities <- read.csv(temporary_file_path, stringsAsFactors = FALSE)
imported_activities
##           activity minutes is_work classification       date is_long_work
## 1    CHECK E-MAIL      20    TRUE        connect 2020-04-14        FALSE
## 2       BREAKFAST      30   FALSE      essential 2020-04-14        FALSE
## 3 CONFERENCE CALL      60    TRUE        connect 2020-04-14         TRUE
## 4         WEBINAR      60    TRUE        consult 2020-04-14         TRUE
## 5            WALK      60   FALSE       exercise 2020-04-14        FALSE
```

Note that some information has not survived the round-trip – the `classification` and `date` columns are plain character vectors.

```
class(imported_activities$classification)
## [1] "character"
class(imported_activities$date)
## [1] "character"
```

Update these to be a `factor()` with specific levels, and a `Date`. ‘

```
levels <- c("connect", "exercise", "consult", "hobby", "essential")
imported_activities$classification <- factor(
    imported_activities$classification,
    levels = levels
)

imported_activities$date <- as.Date(imported_activities$date, format = "%Y-%m-%d")

imported_activities
##           activity minutes is_work classification       date is_long_work
## 1    CHECK E-MAIL      20    TRUE        connect 2020-04-14        FALSE
## 2       BREAKFAST      30   FALSE      essential 2020-04-14        FALSE
## 3 CONFERENCE CALL      60    TRUE        connect 2020-04-14         TRUE
## 4         WEBINAR      60    TRUE        consult 2020-04-14         TRUE
## 5            WALK      60   FALSE       exercise 2020-04-14        FALSE
```

Reading from a remote file (!)

- Visit the New York Times csv file daily tally of COVID-19 cases in all US counties.
- Read the data into an *R* `data.frame`

```
url <-
  "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv"
us <- read.csv(url, stringsAsFactors = FALSE)
```

- Explore the data

```
class(us)
## [1] "data.frame"
dim(us)
## [1] 164886      6
head(us)
##         date    county      state  fips cases deaths
## 1 2020-01-21 Snohomish Washington 53061     1      0
## 2 2020-01-22 Snohomish Washington 53061     1      0
## 3 2020-01-23 Snohomish Washington 53061     1      0
## 4 2020-01-24      Cook   Illinois 17031     1      0
## 5 2020-01-24 Snohomish Washington 53061     1      0
## 6 2020-01-25    Orange California  6059     1      0
```

- Subset the data to only New York state or Erie county

```
ny_state <- us[us$state == "New York",]
dim(ny_state)
## [1] 3813    6

erie <- us[(us$state == "New York") & (us$county == "Erie"), ]
erie
##              date county    state  fips cases deaths
## 2569   2020-03-15   Erie New York 36029     3      0
## 3028   2020-03-16   Erie New York 36029     6      0
## 3544   2020-03-17   Erie New York 36029     7      0
## 4141   2020-03-18   Erie New York 36029     7      0
## 4870   2020-03-19   Erie New York 36029    28      0
## 5717   2020-03-20   Erie New York 36029    31      0
## 6711   2020-03-21   Erie New York 36029    38      0
## 7805   2020-03-22   Erie New York 36029    54      0
## 9003   2020-03-23   Erie New York 36029    87      0
## 10310  2020-03-24   Erie New York 36029   107      0
## 11741  2020-03-25   Erie New York 36029   122      0
## 13345  2020-03-26   Erie New York 36029   134      2
## 15081  2020-03-27   Erie New York 36029   219      6
## 16912  2020-03-28   Erie New York 36029   354      6
## 18839  2020-03-29   Erie New York 36029   380      6
## 20878  2020-03-30   Erie New York 36029   443      8
## 23003  2020-03-31   Erie New York 36029   438      8
## 25193  2020-04-01   Erie New York 36029   553     12
```

```
## 27445  2020-04-02   Erie New York 36029   734     19
## 29765  2020-04-03   Erie New York 36029   802     22
## 32150  2020-04-04   Erie New York 36029   945     26
## 34579  2020-04-05   Erie New York 36029  1059     27
## 37047  2020-04-06   Erie New York 36029  1163     30
## 39558  2020-04-07   Erie New York 36029  1163     36
## 42109  2020-04-08   Erie New York 36029  1205     38
## 44685  2020-04-09   Erie New York 36029  1362     46
## 47294  2020-04-10   Erie New York 36029  1409     58
## 49946  2020-04-11   Erie New York 36029  1472     62
## 52616  2020-04-12   Erie New York 36029  1571     75
## 55297  2020-04-13   Erie New York 36029  1624     86
## 57993  2020-04-14   Erie New York 36029  1668     99
## 60705  2020-04-15   Erie New York 36029  1751    110
## 63430  2020-04-16   Erie New York 36029  1850    115
## 66171  2020-04-17   Erie New York 36029  1929    115
## 68926  2020-04-18   Erie New York 36029  1997    115
## 71690  2020-04-19   Erie New York 36029  2070    146
## 74463  2020-04-20   Erie New York 36029  2109    153
## 77241  2020-04-21   Erie New York 36029  2147    161
## 80028  2020-04-22   Erie New York 36029  2233    174
## 82826  2020-04-23   Erie New York 36029  2450    179
## 85627  2020-04-24   Erie New York 36029  2603    184
## 88436  2020-04-25   Erie New York 36029  2773    199
## 91248  2020-04-26   Erie New York 36029  2954    205
## 94071  2020-04-27   Erie New York 36029  3021    208
## 96904  2020-04-28   Erie New York 36029  3089    216
## 99747  2020-04-29   Erie New York 36029  3196    220
## 102596 2020-04-30   Erie New York 36029  3319    227
## 105454 2020-05-01   Erie New York 36029  3481    233
## 108317 2020-05-02   Erie New York 36029  3598    243
## 111186 2020-05-03   Erie New York 36029  3710    250
## 114062 2020-05-04   Erie New York 36029  3802    254
## 116938 2020-05-05   Erie New York 36029  3891    264
## 119821 2020-05-06   Erie New York 36029  4008    338
## 122719 2020-05-07   Erie New York 36029  4136    350
## 125624 2020-05-08   Erie New York 36029  4255    356
## 128534 2020-05-09   Erie New York 36029  4337    368
## 131444 2020-05-10   Erie New York 36029  4453    376
## 134355 2020-05-11   Erie New York 36029  4483    387
## 137266 2020-05-12   Erie New York 36029  4530    395
## 140183 2020-05-13   Erie New York 36029  4606    402
## 143100 2020-05-14   Erie New York 36029  4671    411
## 146023 2020-05-15   Erie New York 36029  4782    417
## 148952 2020-05-16   Erie New York 36029  4867    428
```

```
## 151883 2020-05-17   Erie New York 36029  4954    438
## 154819 2020-05-18   Erie New York 36029  4993    444
## 157757 2020-05-19   Erie New York 36029  5037    450
## 160706 2020-05-20   Erie New York 36029  5131    455
## 163659 2020-05-21   Erie New York 36029  5270    463
```

2.3 Day 10: `subset()`, `with()`, and `within()`

`subset()`

`subset()`ing a `data.frame`

- Read the New York Times csv file summarizing COVID cases in the US.

```
url <-
  "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv"
us <- read.csv(url, stringsAsFactors = FALSE)
```

- Create subsets, e.g., to include only New York state, or only Erie county

```
ny_state <- subset(us, state == "New York")
dim(ny_state)
## [1] 3813    6
tail(ny_state)
##              date      county    state  fips cases deaths
## 163698 2020-05-21      Warren New York 36113   241     28
## 163699 2020-05-21  Washington New York 36115   222     11
## 163700 2020-05-21       Wayne New York 36117   103      1
## 163701 2020-05-21 Westchester New York 36119 32672   1438
## 163702 2020-05-21     Wyoming New York 36121    79      3
## 163703 2020-05-21       Yates New York 36123    34      6

erie <- subset(us, (state == "New York") & county == "Erie")
dim(erie)
## [1] 68  6
tail(erie)
##              date county    state  fips cases deaths
## 148952 2020-05-16   Erie New York 36029  4867    428
## 151883 2020-05-17   Erie New York 36029  4954    438
## 154819 2020-05-18   Erie New York 36029  4993    444
## 157757 2020-05-19   Erie New York 36029  5037    450
## 160706 2020-05-20   Erie New York 36029  5131    455
## 163659 2020-05-21   Erie New York 36029  5270    463
```

with()

Use `with()` to simplify column references

- Goal: calculate maximum number of cases in the Erie county data subset
- First argument: a `data.frame` containing data to be manipulated – `erie`
- Second argument: an *expression* to be evaluated, usually referencing columns in the data set – `max(cases)`
  - E.g., Calculate the maximum number of cases in the `erie` subset

```
with(erie, max(cases))
## [1] 5270
```

Second argument can be more complicated, using `{}` to enclose several lines.

- E.g., Calculate the number of new cases, and then reports the average number of new cases per day. We will use `diff()`
  - `diff()` calculates the difference between successive values of a vector

```
x <- c(1, 1, 2, 3, 5, 8)
diff(x)
## [1] 0 1 1 2 3
```

  - The length of `diff(x)` is one less than the length of `x`

```
length(x)
## [1] 6
length(diff(x))
## [1] 5
```

- `new_cases` is the `diff()` of successive values of `cases`, with an implicit initial value equal to 0.

```
with(erie, {
    new_cases <- diff(c(0, cases))
    mean(new_cases)
})
## [1] 77.5
```

within()

Adding and updating columns `within()` a `data.frame`

- First argument: a `data.frame` containing data to be updated – `erie`
- Second argument: an expression of one or more variable assignments, the assignments create new columns in the `data.frame`.
- Example: add a `new_cases` column

```
erie_new_cases <- within(erie, {
    new_cases <- diff(c(0, cases))
})
head(erie_new_cases)
##            date county    state  fips cases deaths new_cases
## 2569 2020-03-15   Erie New York 36029     3      0         3
## 3028 2020-03-16   Erie New York 36029     6      0         3
## 3544 2020-03-17   Erie New York 36029     7      0         1
## 4141 2020-03-18   Erie New York 36029     7      0         0
## 4870 2020-03-19   Erie New York 36029    28      0        21
## 5717 2020-03-20   Erie New York 36029    31      0         3
```

2.4 Day 11: aggregate() and an initial work flow

aggregate() for summarizing columns by group

Goal: summarize maximum number of cases by county in New York state

Setup

- Read and subset the New York Times data to contain only New York state data

```
url <- "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties
us <- read.csv(url, stringsAsFactors = FALSE)
```

```
ny_state <- subset(us, state == "New York")
```

`aggregate()`

- First argument: a *formula* – `cases ~ county`
  - Right-hand side: the variable to be used to subset (group) the data – `county`
  - Left-hand side: the variable to be used in the aggregation function – `cases`
- Second argument: source of data – `ny_state`
- Third argument: the function to be applied to each subset of data – `max`
- Maximum number of cases by county:

```
max_cases_by_county <- aggregate( cases ~ county, ny_state, max )
```

Exploring the data summary

- Subset to some interesting 'counties'

```
head(max_cases_by_county)
##        county cases
## 1      Albany  1700
## 2    Allegany    44
## 3      Broome   451
## 4 Cattaraugus    71
## 5      Cayuga    72
## 6  Chautauqua    58
subset(
    max_cases_by_county,
    county %in% c("New York City", "Westchester", "Erie")
)
##           county  cases
## 14          Erie   5270
## 29 New York City 200507
## 57   Westchester  32672
```

Help: `?aggregate.formula`

An initial work flow

Data input

- From a remote location

```
url <- "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv"
us <- read.csv(url, stringsAsFactors = FALSE)
```

```
class(us)
## [1] "data.frame"
dim(us)
## [1] 164886      6
head(us)
##         date    county      state  fips cases deaths
## 1 2020-01-21 Snohomish Washington 53061     1      0
## 2 2020-01-22 Snohomish Washington 53061     1      0
## 3 2020-01-23 Snohomish Washington 53061     1      0
## 4 2020-01-24      Cook   Illinois 17031     1      0
## 5 2020-01-24 Snohomish Washington 53061     1      0
## 6 2020-01-25    Orange California  6059     1      0
```

Cleaning

- `date` is a plain-old `character` vector, but should be a `Date`.

```
class(us$date) # oops, should be 'Date'
## [1] "character"
```

- Update, method 1

```
us$date <- as.Date(us$date, format = "%Y-%m-%d")
head(us)
##         date    county      state  fips cases deaths
## 1 2020-01-21 Snohomish Washington 53061     1      0
## 2 2020-01-22 Snohomish Washington 53061     1      0
## 3 2020-01-23 Snohomish Washington 53061     1      0
## 4 2020-01-24      Cook   Illinois 17031     1      0
## 5 2020-01-24 Snohomish Washington 53061     1      0
## 6 2020-01-25    Orange California  6059     1      0
```

- Update, method 2

```
us <- within(us, {
    date = as.Date(date, format = "%Y-%m-%d")
})
head(us)
##         date    county      state  fips cases deaths
## 1 2020-01-21 Snohomish Washington 53061     1      0
## 2 2020-01-22 Snohomish Washington 53061     1      0
## 3 2020-01-23 Snohomish Washington 53061     1      0
## 4 2020-01-24      Cook   Illinois 17031     1      0
## 5 2020-01-24 Snohomish Washington 53061     1      0
## 6 2020-01-25    Orange California  6059     1      0
```

Subset to only Erie county, New York state

- Subset, method 1

```
row_idx <- (us$county == "Erie") & (us$state == "New York")
erie <- us[row_idx,]
dim(erie)
## [1] 68  6
```

- Subset, method 2

```
erie <- subset(us, (county == "Erie") & (state == "New York"))
dim(erie)
## [1] 68  6
```

Manipulation

- Goal: calculate `new_cases` as the difference between succesive days, using `diff()`
- Remember use of `diff()`

```
## example: `diff()` between successive numbers in a vector
x <- c(1, 1, 2, 3, 5, 8, 13)
```

```
diff(x)
## [1] 0 1 1 2 3 5
```

- Update, methods 1 & 2 (prepend a `0` when using `diff()`, to get the intial number of new cases)

```
## one way...
erie$new_cases <- diff( c(0, erie$cases) )

## ...or another
erie <- within(erie, {
    new_cases <- diff( c(0, cases) )
})
```

Summary: calculate maximum (total) number of cases per county in New York state

- For Erie county, let's see how to calculate the maximum (total) number of cases

```
max(erie$cases)        # one way...
## [1] 5270
with(erie, max(cases)) # ... another
## [1] 5270
```

- Subset US data to New York state

```
ny_state <- subset(us, state == "New York")
```

- Summarize each county in the state using `aggregate()`.
  - First argument: summarize `cases` grouped by `county` – `cases ~ county`
  - Second argument: data source – `ny_state`
  - Third argument: function to apply to each subset – `max`

```
max_cases_by_county <- aggregate( cases ~ county, ny_state, max)
head(max_cases_by_county)
##         county cases
## 1       Albany  1700
## 2    Allegany    44
## 3      Broome   451
## 4 Cattaraugus    71
## 5      Cayuga    72
## 6  Chautauqua    58
```

- `subset()` to select counties

```
subset(
    max_cases_by_county,
    county %in% c("New York City", "Westchester", "Erie")
)
##           county  cases
## 14          Erie   5270
## 29 New York City 200507
## 57   Westchester  32672
```

Summary: calculate maximum (total) number of cases per state

- Use entire data set, `us`
- `aggregate()` cases by county *and* state – `cases ~ county + state`

```
max_cases_by_county_state <-
    aggregate( cases ~ county + state, us, max )
dim(max_cases_by_county_state)
## [1] 2975    3
head(max_cases_by_county_state)
##    county   state cases
## 1 Autauga Alabama   147
## 2 Baldwin Alabama   270
## 3 Barbour Alabama   100
## 4    Bibb Alabama    52
## 5  Blount Alabama    48
## 6 Bullock Alabama    71
```

- `aggregate()` a second time, using `max_cases_by_county_state` and aggregtaing by state

```
max_cases_by_state <-
    aggregate( cases ~ state, max_cases_by_county_state, max )
```

- Explore the data

```
head(max_cases_by_state)
##        state cases
## 1    Alabama  1874
## 2     Alaska   207
## 3    Arizona  7835
## 4   Arkansas   966
## 5 California 42037
## 6   Colorado  4948
subset(
    max_cases_by_state,
    state %in% c("California", "Illinois", "New York", "Washington")
)
##         state  cases
```

```
## 5  California  42037
## 15   Illinois  67551
## 34   New York 200507
## 52 Washington   7647
```

2.5 Day 12 (Friday) Zoom check-in

Review and troubleshoot (20 minutes)

```
## retrieve and clean the current data set
url <- "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv"
us <- read.csv(url, stringsAsFactors = FALSE)
us <- within(us, {
    date = as.Date(date, format = "%Y-%m-%d")
})

## subset
erie <- subset(us, (county == "Erie") & (state == "New York"))

## manipulate
erie <- within(erie, {
    new_cases <- diff( c(0, cases) )
})

## record of cases to date
erie
##              date county    state  fips cases deaths new_cases
## 2569   2020-03-15   Erie New York 36029     3      0         3
## 3028   2020-03-16   Erie New York 36029     6      0         3
## 3544   2020-03-17   Erie New York 36029     7      0         1
## 4141   2020-03-18   Erie New York 36029     7      0         0
## 4870   2020-03-19   Erie New York 36029    28      0        21
## 5717   2020-03-20   Erie New York 36029    31      0         3
## 6711   2020-03-21   Erie New York 36029    38      0         7
## 7805   2020-03-22   Erie New York 36029    54      0        16
## 9003   2020-03-23   Erie New York 36029    87      0        33
## 10310  2020-03-24   Erie New York 36029   107      0        20
## 11741  2020-03-25   Erie New York 36029   122      0        15
## 13345  2020-03-26   Erie New York 36029   134      2        12
## 15081  2020-03-27   Erie New York 36029   219      6        85
## 16912  2020-03-28   Erie New York 36029   354      6       135
## 18839  2020-03-29   Erie New York 36029   380      6        26
## 20878  2020-03-30   Erie New York 36029   443      8        63
## 23003  2020-03-31   Erie New York 36029   438      8        -5
```

```
## 25193  2020-04-01   Erie New York 36029   553    12    115
## 27445  2020-04-02   Erie New York 36029   734    19    181
## 29765  2020-04-03   Erie New York 36029   802    22     68
## 32150  2020-04-04   Erie New York 36029   945    26    143
## 34579  2020-04-05   Erie New York 36029  1059    27    114
## 37047  2020-04-06   Erie New York 36029  1163    30    104
## 39558  2020-04-07   Erie New York 36029  1163    36      0
## 42109  2020-04-08   Erie New York 36029  1205    38     42
## 44685  2020-04-09   Erie New York 36029  1362    46    157
## 47294  2020-04-10   Erie New York 36029  1409    58     47
## 49946  2020-04-11   Erie New York 36029  1472    62     63
## 52616  2020-04-12   Erie New York 36029  1571    75     99
## 55297  2020-04-13   Erie New York 36029  1624    86     53
## 57993  2020-04-14   Erie New York 36029  1668    99     44
## 60705  2020-04-15   Erie New York 36029  1751   110     83
## 63430  2020-04-16   Erie New York 36029  1850   115     99
## 66171  2020-04-17   Erie New York 36029  1929   115     79
## 68926  2020-04-18   Erie New York 36029  1997   115     68
## 71690  2020-04-19   Erie New York 36029  2070   146     73
## 74463  2020-04-20   Erie New York 36029  2109   153     39
## 77241  2020-04-21   Erie New York 36029  2147   161     38
## 80028  2020-04-22   Erie New York 36029  2233   174     86
## 82826  2020-04-23   Erie New York 36029  2450   179    217
## 85627  2020-04-24   Erie New York 36029  2603   184    153
## 88436  2020-04-25   Erie New York 36029  2773   199    170
## 91248  2020-04-26   Erie New York 36029  2954   205    181
## 94071  2020-04-27   Erie New York 36029  3021   208     67
## 96904  2020-04-28   Erie New York 36029  3089   216     68
## 99747  2020-04-29   Erie New York 36029  3196   220    107
## 102596 2020-04-30   Erie New York 36029  3319   227    123
## 105454 2020-05-01   Erie New York 36029  3481   233    162
## 108317 2020-05-02   Erie New York 36029  3598   243    117
## 111186 2020-05-03   Erie New York 36029  3710   250    112
## 114062 2020-05-04   Erie New York 36029  3802   254     92
## 116938 2020-05-05   Erie New York 36029  3891   264     89
## 119821 2020-05-06   Erie New York 36029  4008   338    117
## 122719 2020-05-07   Erie New York 36029  4136   350    128
## 125624 2020-05-08   Erie New York 36029  4255   356    119
## 128534 2020-05-09   Erie New York 36029  4337   368     82
## 131444 2020-05-10   Erie New York 36029  4453   376    116
## 134355 2020-05-11   Erie New York 36029  4483   387     30
## 137266 2020-05-12   Erie New York 36029  4530   395     47
## 140183 2020-05-13   Erie New York 36029  4606   402     76
## 143100 2020-05-14   Erie New York 36029  4671   411     65
## 146023 2020-05-15   Erie New York 36029  4782   417    111
```

```
## 148952 2020-05-16   Erie New York 36029  4867    428        85
## 151883 2020-05-17   Erie New York 36029  4954    438        87
## 154819 2020-05-18   Erie New York 36029  4993    444        39
## 157757 2020-05-19   Erie New York 36029  5037    450        44
## 160706 2020-05-20   Erie New York 36029  5131    455        94
## 163659 2020-05-21   Erie New York 36029  5270    463       139

## aggregate() cases in each county to find total (max) number
ny_state <- subset(us, state == "New York")
head( aggregate(cases ~ county, ny_state, max) )
##        county cases
## 1      Albany  1700
## 2    Allegany    44
## 3      Broome   451
## 4 Cattaraugus    71
## 5      Cayuga    72
## 6  Chautauqua    58
```

User-defined functions

Basic structure:

```
my_function_name <- function(arg1, arg2, ...)
{
   statements

   return(object)
}
```

A conrete example:

```
# declare a function to convert temperatures
toFahrenheit <- function(celsius)
{
  f <- (9/5) * celsius + 32
  return(f)
}

# invoke the function
temps <- c(20:25)
toFahrenheit(temps)
## [1] 68.0 69.8 71.6 73.4 75.2 77.0
```

Functions can be loaded from a separate file using the `source` command. Enter the temperature conversion function into an *R* script and save as `myFunctions.R`.

```
my_R_funcs <-
    file.path("C:\\Matott\\MyQuarantine", "myFunctions.R")
source(my_R_funcs)
```

```
toFahrenheit(c(40, 45, 78, 92, 12, 34))
## [1] 104.0 113.0 172.4 197.6  53.6  93.2
```

Statistical functions in *R*

R has many built-in statistical functions. Some of the more commonly used are listed below:

- `mean()    # average`
- `median()  # median (middle value of sorted data)`
- `range()   # max - min`
- `var()     # variance`
- `sd()      # standard deviation`
- `summary() # prints a combination of useful measures`

Plotting data

Review of Plot Types

- Pie chart
  - Display proportions of different values for some variable
- Bar plot
  - Display counts of values for categorical variables
- Histogram, density plot
  - Display counts of values for a binned, numeric variable
- Scatter plot
  - Display y vs. x
- Box plot
  - Display distributions over different values of a variable

Plotting packages

3 Main Plotting Packages

- Base graphics, lattice, and ggplot2

ggplot2

- The "Cadillac" of plotting packages.
- Part of the "tidyverse"
- Beautiful plots
- To install: `install.packages('ggplot2')`

Words of wisdom on using plotting packages

- A good approach is to learn by doing but don't start from scratch
- Find an example that is similar in appearance to what you are trying to achieve - many *R* galleries are available on the net.
- When you find something you like, grab the code and modify it to use your own data.
- Fine tune things like labels and fonts at the end, after you are sure you like the way the data is being displayed.

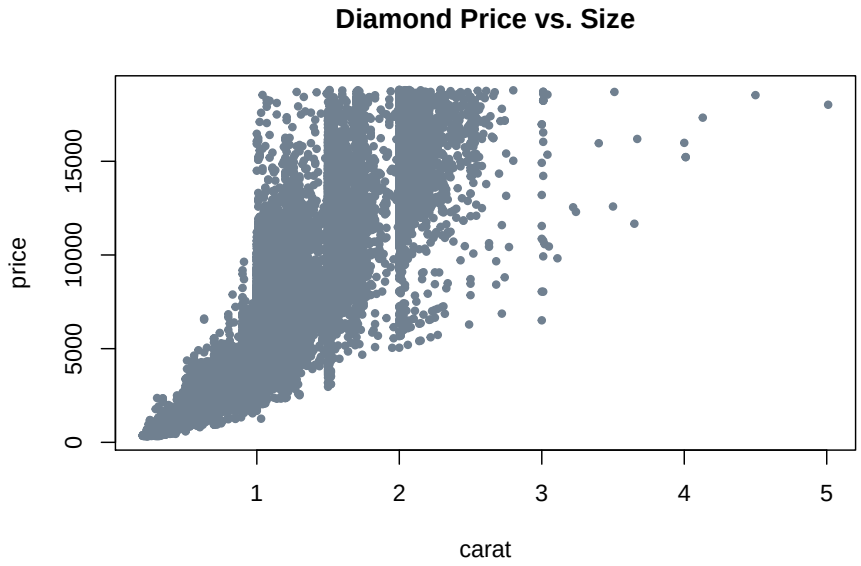
Some example plots

Many of these are based on the Diamonds dataset. Others are based on the `mtcars` dataset and this requires a bit of cleaning in preparation for the corresponding plots:

```
data("mtcars")

# convert gear to a factor-level object
mtcars$gear = factor(
    mtcars$gear,
    levels = c(3, 4, 5),
    labels = c("3 gears", "4 gears", "5 gears")
)

# convert cyl to a factor-level object
mtcars$cyl = factor(
    mtcars$cyl,
    levels = c(4, 6, 8),
    labels = c("4 cyl", "6 cyl", "8 cyl")
)
```

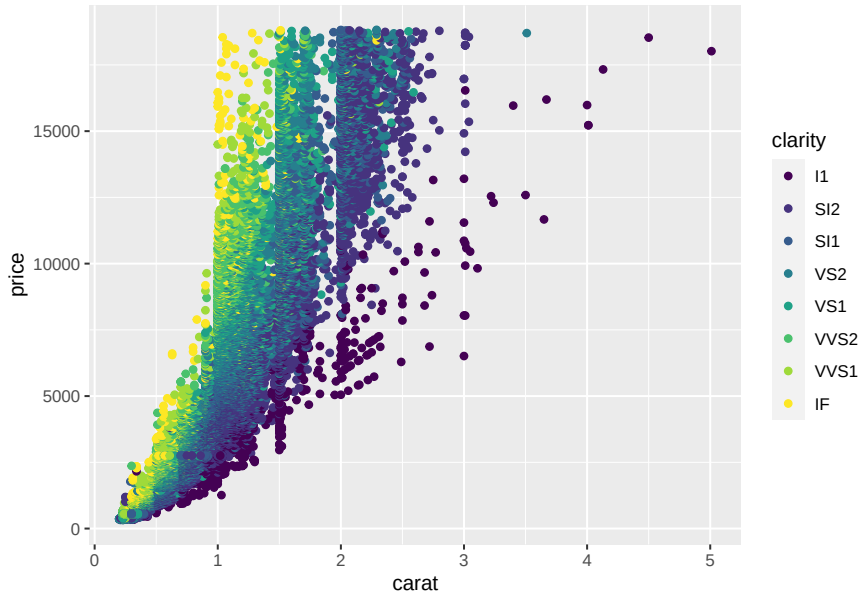
- Scatterplot using base graphics

```
library(ggplot2) # for diamonds dataset
data("diamonds")

# using base graphics plot() function
plot(formula = price ~ carat,  # price vs. carat
        data = diamonds,
        col  = "slategray",
        pch  = 20,
        main = "Diamond Price vs. Size"
)
```



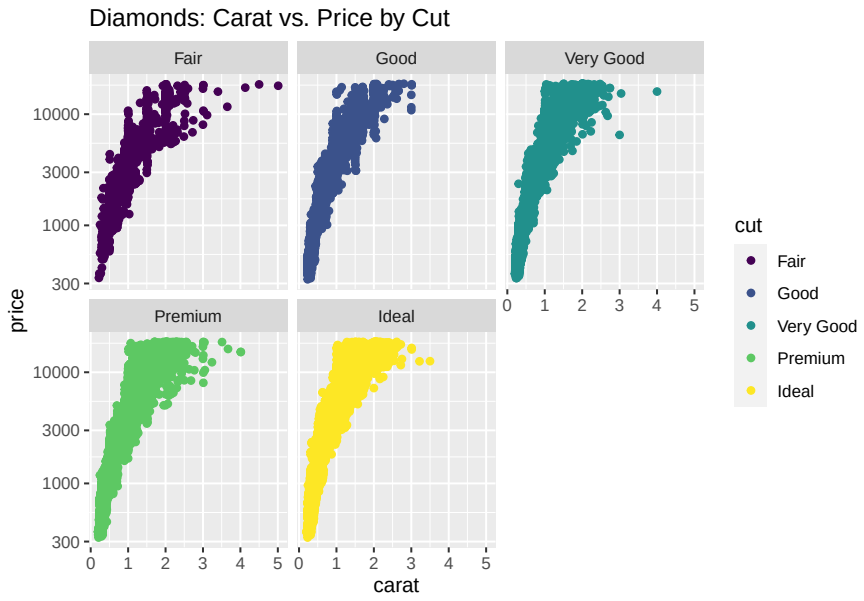
- Scatterplot using ggplot

```
library(ggplot2)
data("diamonds")
ggplot(diamonds,
    aes(x=carat, y=price, colour=clarity)) +
    geom_point()
```



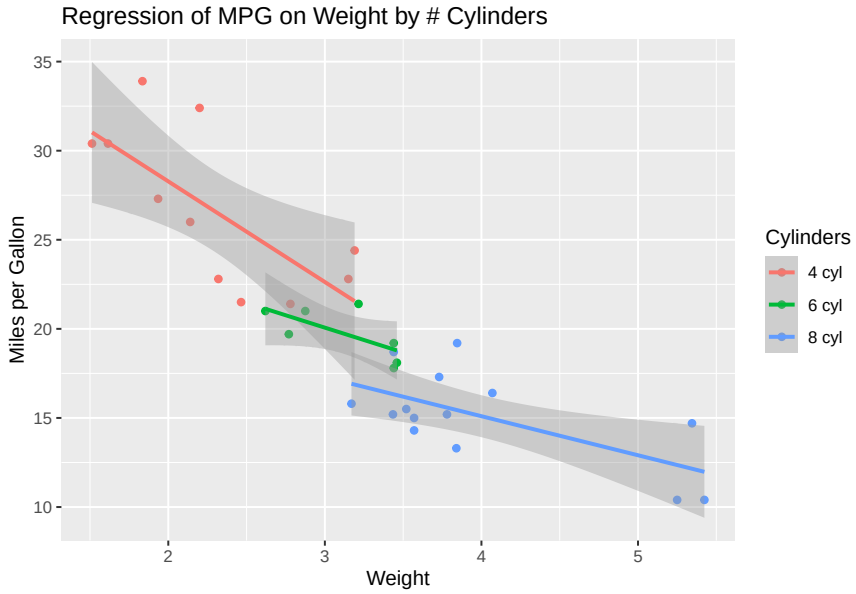
- Scatterplot by category (using ggplot)

```
library(ggplot2)
data("diamonds")
ggplot(diamonds) +
    geom_point(aes(x=carat, y=price, colour=cut)) +
    scale_y_log10()+
    facet_wrap(~cut)+
    ggtitle("Diamonds: Carat vs. Price by Cut")
```



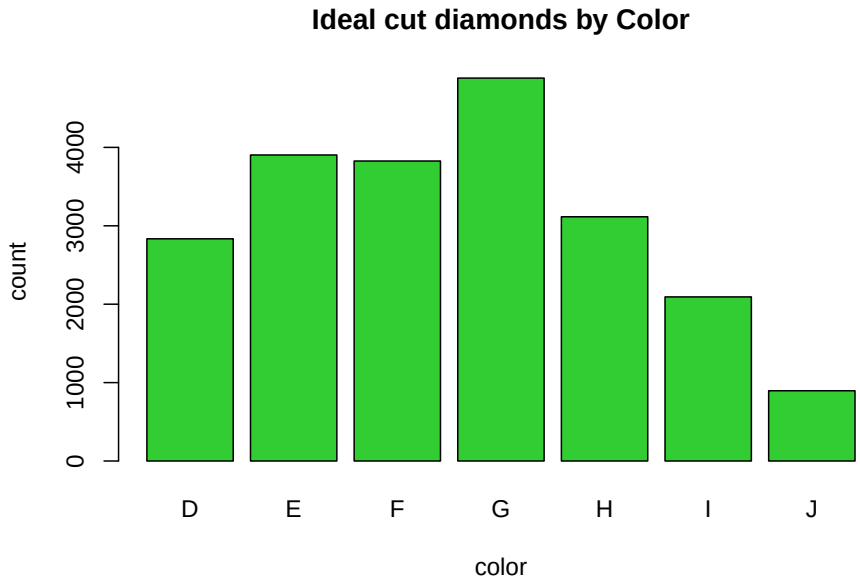
- Scatterplot by category with regression lines (using ggplot2)

```
library(ggplot2)
ggplot(mtcars, aes(wt, mpg, color=cyl))+
    geom_point()+
    geom_smooth(method="lm")+
    labs(title="Regression of MPG on Weight by # Cylinders",
         x = "Weight",
         y = "Miles per Gallon",
         color = "Cylinders")
## `geom_smooth()` using formula 'y ~ x'
```



- Bar Plot using Base Graphics

```
library(ggplot2) # for diamonds dataset
data("diamonds")
ideal_cut_colors = diamonds[diamonds$cut == "Ideal", "color"]
# using base graphics barplot() function
barplot(table(ideal_cut_colors),
        xlab = "color",
        ylab = "count",
        main = "Ideal cut diamonds by Color",
        col = "limegreen")
```

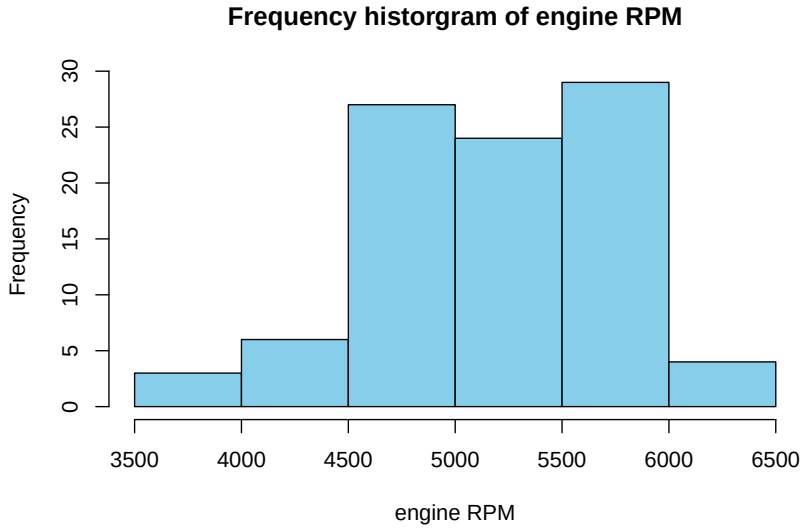


  - `limegreen` is one of many cool color choices supported by R. See http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf for a nice 8-page compendium!

- Frequency histogram using base graphics

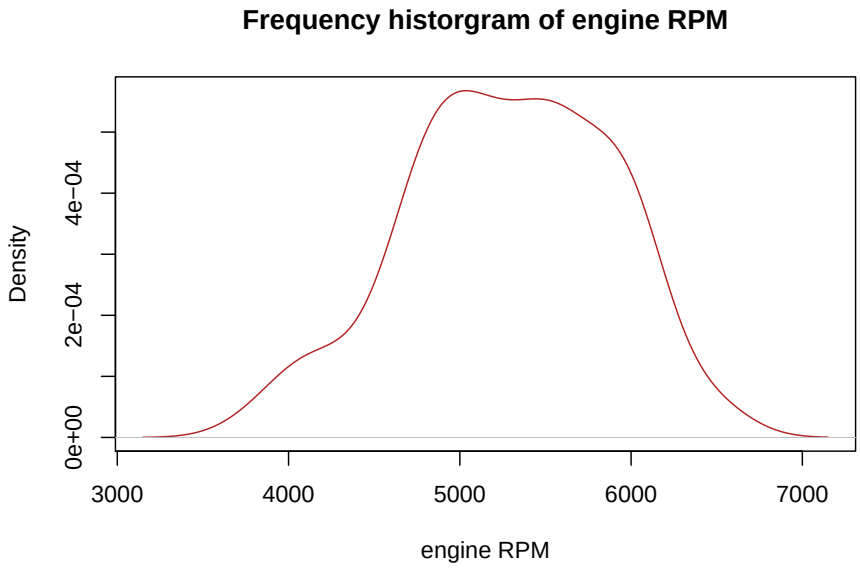
  - The `MASS` library provides datasets for Venables and Ripley's MASS textbook

```
library(MASS)
# using the base graphics hist() function
hist(Cars93$RPM,
     xlab = "engine RPM",
     main = "Frequency historgram of engine RPM",
     col = "skyblue")
```



- Density plot using base graphics

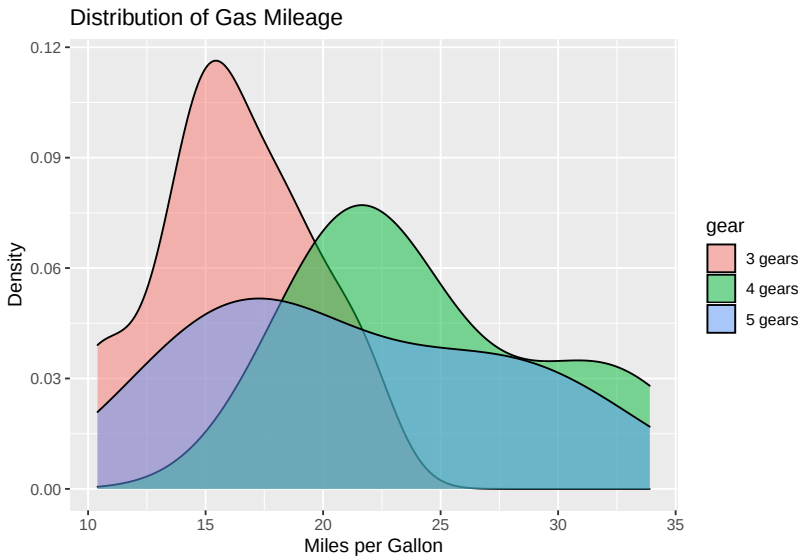
```
library(MASS)
plot(density(Cars93$RPM),
     xlab = "engine RPM",
     main = "Frequency historgram of engine RPM",
     col = "firebrick")
```



- Density plot using ggplot2
  - The `qplot()` function is a "quick plot" wrapper for `ggplot()`.

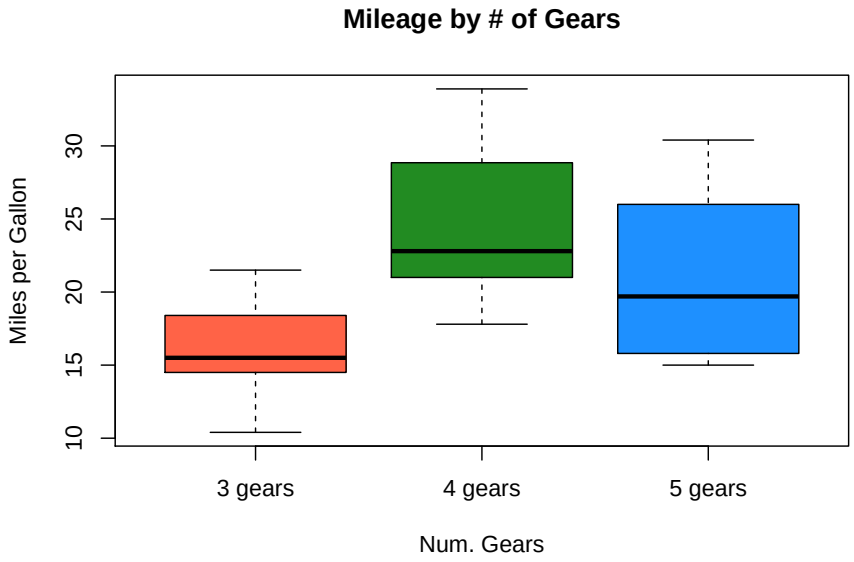
– It uses an interface that is similar to the `plot()` function of the base graphics package.

```
library(ggplot2)

qplot(mpg,
      data = mtcars,
      geom = "density",
      fill = gear, alpha=I(0.5),
      main = "Distribution of Gas Mileage",
      xlab = "Miles per Gallon",
      ylab = "Density")
```



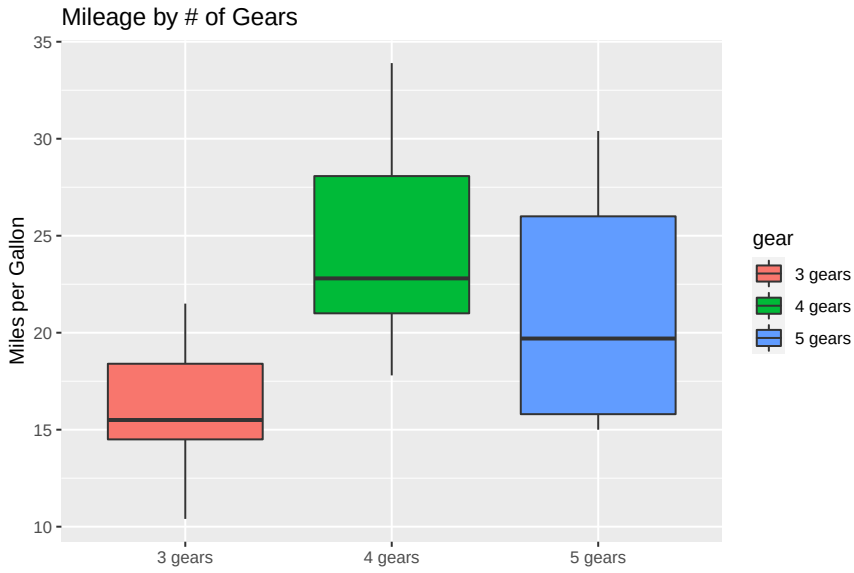
- Box-and-whisker plot using base graphics

```
boxplot(formula = mpg ~ gear,
        data = mtcars,
        main = "Mileage by # of Gears",
        xlab = "Num. Gears",
        ylab = "Miles per Gallon",
        col = c("tomato", "forestgreen", "dodgerblue"))
```



- Box-and-whisker plot using ggplot

```
library(ggplot2)
ggplot(data = mtcars,
       aes(gear, mpg, fill=gear))+
        geom_boxplot()+
        labs(title="Mileage by # of Gears",
             x = "",
             y = "Miles per Gallon")
```



2.6 Day 13: Basic visualization

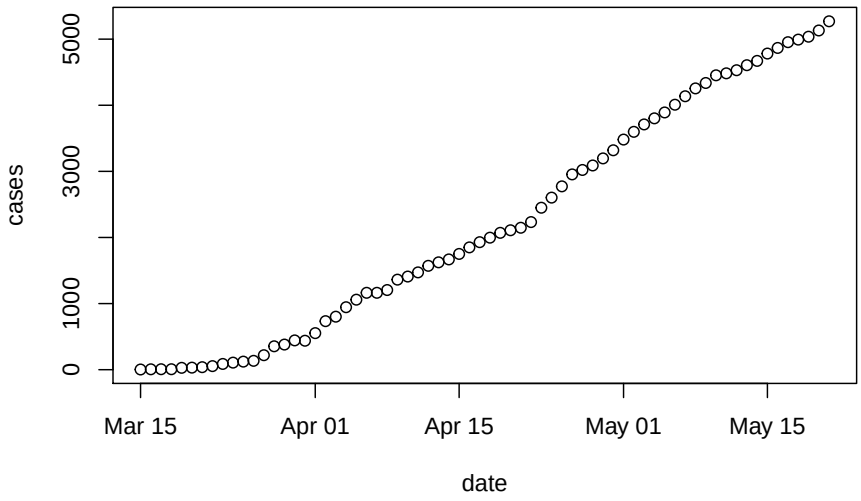
Let's get the current Erie county data, and create the `new_cases` column

```
## retrieve and clean the current data set
url <- "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv"
us <- read.csv(url, stringsAsFactors = FALSE)
us <- within(us, {
    date = as.Date(date, format = "%Y-%m-%d")
})

## get the Erie county subset
erie <- subset(us, (county == "Erie") & (state == "New York"))

## add the `new_cases` column
erie <- within(erie, {
    new_cases <- diff( c(0, cases) )
})
```

Simple visualization

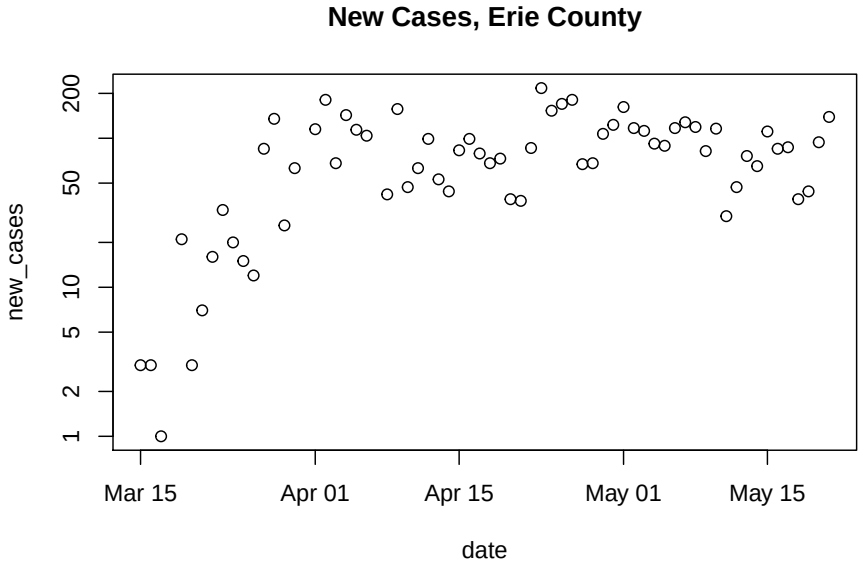
- We'll use the `plot()` function to create a visualization of the progression of COVID cases in Erie county.
- `plot()` can be used with a `formula`, similar to how we used `aggregate()`.
- The `formula` describes the independent (y-axis) variable as a function of the dependent (x-axis) variable
- For our case, the formula will be `cases ~ date`, i.e., plot the number of cases on the y-axis, and date on the x-axis.
- As with `aggregate()`, we need to provide, in the second argument, the `data.frame` where the variables to be plotted can be found.
- Ok, here we go...

```
plot( cases ~ date, erie)
```



- It might be maybe more informative to plot new cases (so that we can see more easily whether social distancing and other measures are having an effect on the spread of COVID cases. Using log-transformed new cases helps to convey the proportional increase

```
plot( new_cases ~ date, erie, log = "y", main = "New Cases, Erie County" )
## Warning in xy.coords(x, y, xlabel, ylabel, log): 3 y values <= 0 omitted from
## logarithmic plot
```



- See `?plot.formula` for some options available when using the formula interface to plot. Additional arguments are described on the help page `?help.default`.

2.7 Day 14: Functions

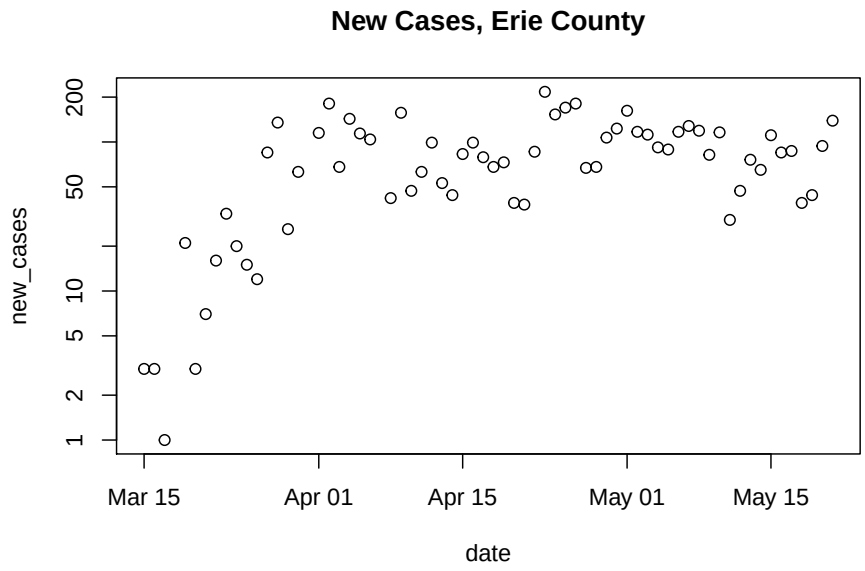
Yesterday we created a plot for Erie county. The steps to create this plot can be separated into two parts

1. Get the full data

```
url <- "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv"
us <- read.csv(url, stringsAsFactors = FALSE)
us <- within(us, {
    date = as.Date(date, format = "%Y-%m-%d")
})
```

2. Subset, update, and plot the data for county of interest

```
erie <- subset(us, (county == "Erie") & (state == "New York"))
erie <- within(erie, {
    new_cases <- diff( c(0, cases) )
})
plot( new_cases ~ date, erie, log = "y", main = "New Cases, Erie County" )
## Warning in xy.coords(x, y, xlabel, ylabel, log): 3 y values <= 0 omitted from
## logarithmic plot
```



What if we were interested in a different county? We could repeat (cut-and-paste) step 2, updating and generalizing a little

- Define a new variable to indicate the county we are interested in plotting

```
county_of_interest <- "Westchester"
```

- `paste()` concatenates its arguments together into a single character vector.

We use this to construct the title of the plot

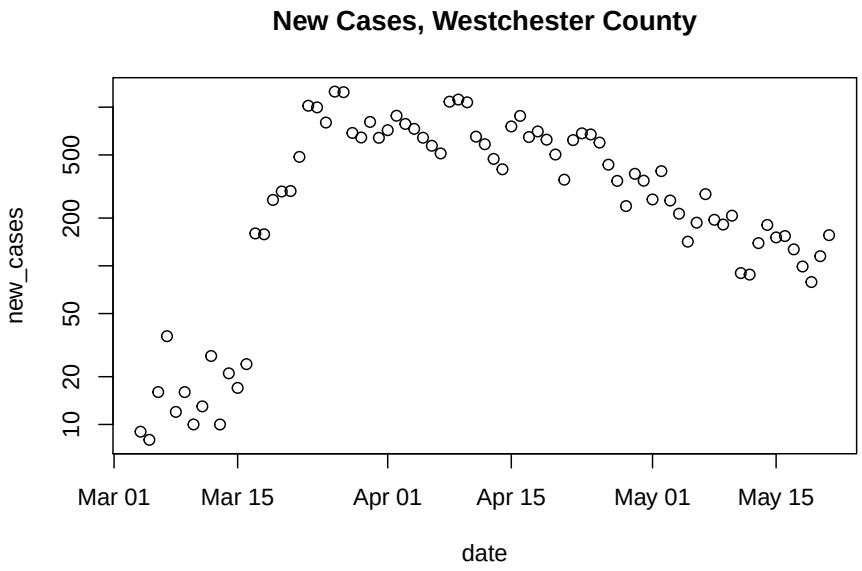
```
main_title <- paste("New Cases,", county_of_interest, "County")
```

- Now create and update a subset of the data for the county that we are interested in

```
    county_data <-
   subset(us, (county == county_of_interest) & (state == "New York"))
    county_data <- within(county_data, {
   new_cases <- diff( c(0, cases) )
    })
```

- … and finally plot the county data

```
plot( new_cases ~ date, county_data, log = "y", main = main_title)
```



- Here is the generalization

```
county_of_interest <- "Westchester"

main_title <- paste("New Cases,", county_of_interest, "County")
county_data <-
    subset(us_data, (county == county_of_interest) & (state == "New York"))
county_data <- within(county_data, {
    new_cases <- diff( c(0, cases) )
})
plot( new_cases ~ date, county_data, log = "y", main = main_title)
```

It would be tedious and error-prone to copy and paste this code for each county we were interested in.

A better approach is to write a `function` that takes as inputs the `us` data.frame, and the name of the county that we want to plot. Functions are easy to write

- Create a variable to contain the function, use the keyword `function` and then the arguments you want to pass in.

```
plot_county <-
    function(us_data, county_of_interest)
```

- … then provide the 'body' of the function between curly braces

```
{
    main_title <- paste("New Cases,", county_of_interest, "County")
    county_data <-
        subset(us_data, (county == county_of_interest) & (state == "New York"))
    county_data <- within(county_data, {
        new_cases <- diff( c(0, cases) )
    })

    plot( new_cases ~ date, county_data, log = "y", main = main_title)
}
```

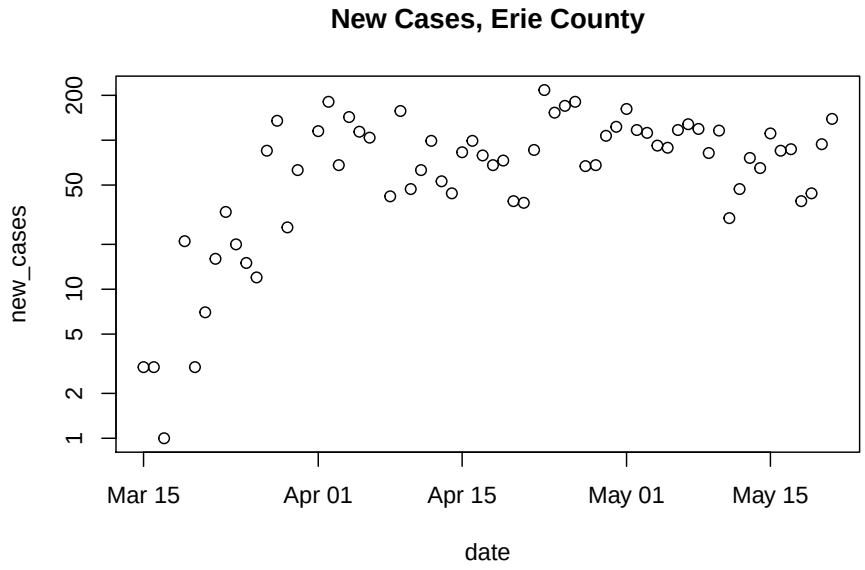
- Normally, the last evaluated line of the code (the `plot()` statement in our example) is returned from the function and can be captured by a variable. In our specific case, `plot()` creates the plot as a *side effect*, and the return value is actually the special symbol `NULL`.

- Here's the full definition

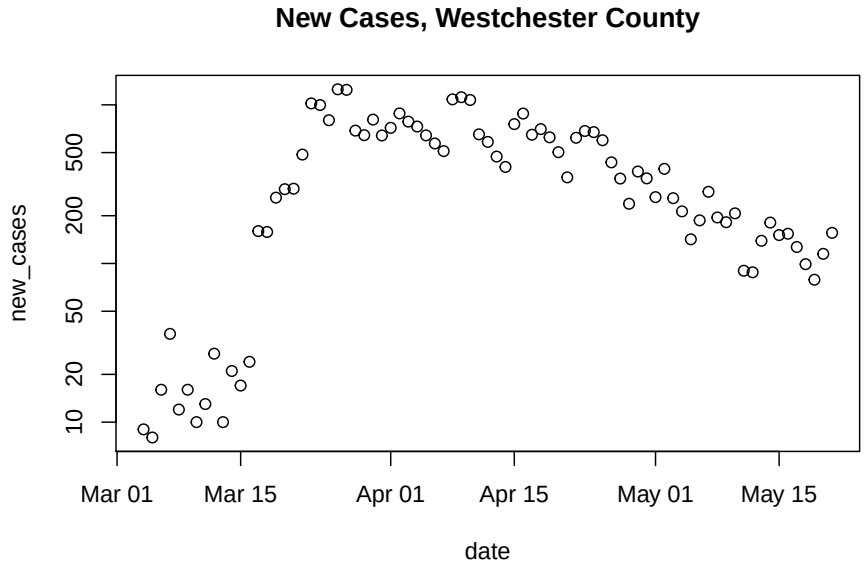
```
plot_county <-
    function(us_data, county_of_interest)
{
    main_title <- paste("New Cases,", county_of_interest, "County")
    county_data <-
        subset(us_data, (county == county_of_interest) & (state == "New York"))
    county_data <- within(county_data, {
        new_cases <- diff( c(0, cases) )
    })

    plot( new_cases ~ date, county_data, log = "y", main = main_title)
}
```

- Run the code defining the function in the *R* console, then use it to plot different counties:

```
plot_county(us, "Erie")
## Warning in xy.coords(x, y, xlabel, ylabel, log): 3 y values <= 0 omitted from
## logarithmic plot
```




```
plot_county(us, "Westchester")
```



Hmm, come to think of it, we might want to write a simple function to get and clean the US data.

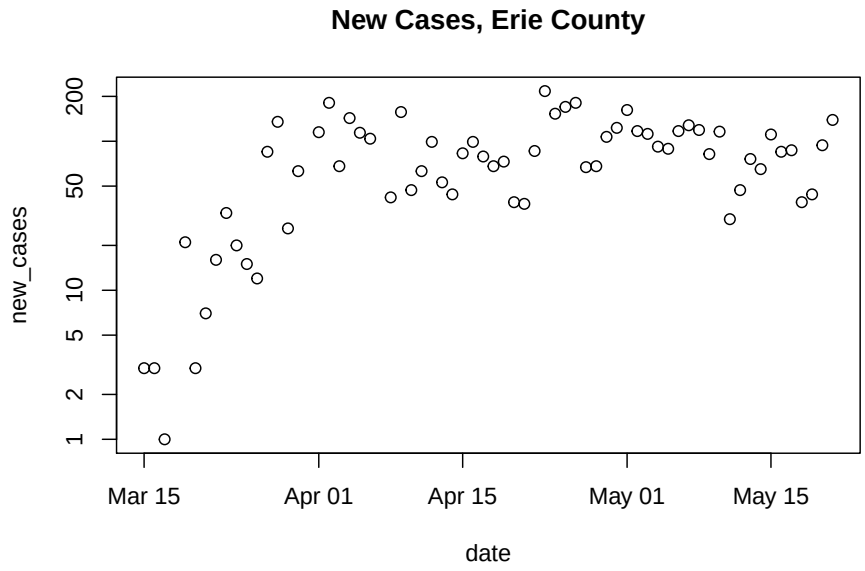
- Get and clean the US data; we don't need any arguments, and the return value (the last line of code evaluated) is the cleaned data

```
get_US_data <-
    function()
{
```

```
    url <- "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv"
    us <- read.csv(url, stringsAsFactors = FALSE)
    within(us, {
        date = as.Date(date, format = "%Y-%m-%d")
    })
}
```

- Verify that it is now just two lines to plot county-level data

```
us <- get_US_data()
plot_county(us, "Erie")
## Warning in xy.coords(x, y, xlabel, ylabel, log): 3 y values <= 0 omitted from
## logarithmic plot
```



- How could you generalize `plot_county()` to plot county-level data for a county in any state? Hint: add a `state =` argument, perhaps using default values

```
plot_county <-
    function(us_data, county = "Erie", state = "New York")
{
    ## your code here!
}
```

Chapter 3

Packages and the 'tidyverse'

3.1 Day 15 (Monday) Zoom check-in

Review and troubleshoot (15 minutes)

Over the weekend, I wrote two functions. The first retrieves and 'cleans' the US data set.

```
get_US_data <-
    function()
{
    ## retrieve data from the internet
    url <- "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv"
    us <- read.csv(url, stringsAsFactors = FALSE)

    ## update 'date' from character vector to 'Date'.  this is the
    ## last line of executed code in the function, so the return
    ## value (the updated 'us' object) is returned by the functino
    within(us, {
        date = as.Date(date, format = "%Y-%m-%d")
    })
}
```

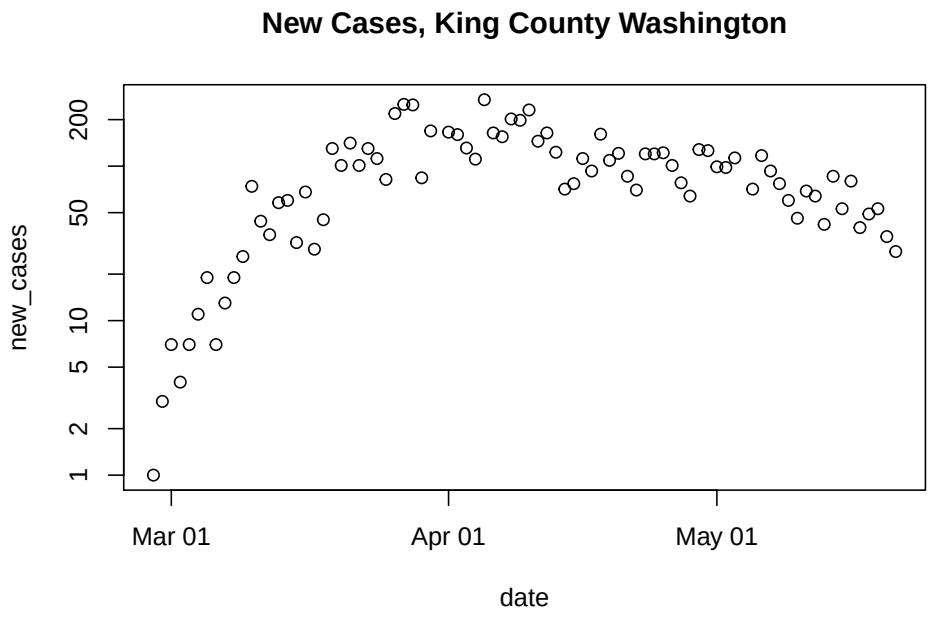
The second plots data for a particular county and state

```
plot_county <-
    function(us_data, county_of_interest = "Erie", state_of_interest = "New York")
{
    ## create the title for the plot
```

```
    main_title <- paste(
        "New Cases,", county_of_interest, "County", state_of_interest
    )

    ## subset the us data to just the county and state of interest
    county_data <- subset(
        us_data,
        (county == county_of_interest) & (state == state_of_interest)
    )

    ## calculate new cases for particular county and state
    county_data <- within(county_data, {
        new_cases <- diff( c(0, cases) )
    })

    ## plot
    plot( new_cases ~ date, county_data, log = "y", main = main_title)
}
```

I lived in Seattle (King County, Washington), for a while, and this is where the first serious outbreak occurred. Here's the relevant data:

```
us <- get_US_data()
plot_county(us, "King", "Washington")
## Warning in xy.coords(x, y, xlabel, ylabel, log): 2 y values <= 0 omitted from
## logarithmic plot
```



Packages (20 minutes)

Base *R*

- *R* consists of 'packages' that implement different functionality. Each package contains *functions* that we can use, and perhaps data sets (like the `mtcars`) data set from Friday's presentation) and other resources.
- *R* comes with several 'base' packages installed, and these are available in a new *R* session.
- Discover packages that are currently available using the `search()` function. This shows that the 'stats', 'graphics', 'grDevices', 'utils', 'datasets', 'methods', and 'base' packages, among others, are available in our current *R* session.

```
> search()
## [1] ".GlobalEnv"        "package:stats"     "package:graphics"
## [4] "package:grDevices" "package:utils"     "package:datasets"
## [7] "package:methods"   "Autoloads"         "package:base"
```

- When we create a variable like

```
x <- c(1, 2, 3)
```

  R creates a new *symbol* in the `.GlobalEnv` location on the search path.

- When we evaluate a function like `length(x)`…
  - *R* searches for the function `length()` along the `search()` path. It doesn't find `length()` in the `.GlobalEnv` (because we didn't define it there), or in the 'stats', 'graphics', … packages. Eventually, *R* finds the definition of `length` in the 'base' package.
  - *R* then looks for the definition of `x`, finds it in the `.GlobalEnv`.
  - Finally, *R* applies the definition of `length` found in the base package to the value of `x` found in the `.GlobalEnv`.

Contributed packages

- *R* would be pretty limited if it could only do things that are defined in the base packages.
- It is 'easy' to write a package, and to make the package available for others to use.
- A major repository of contributed packages is CRAN – the Comprehensive *R* Archive Network. There are more than 15,000 packages in CRAN.
- Many CRAN packages are arranged in task views that highlight the most useful packages.

Installing and attaching packages

- There are too many packages for all to be distributed with *R*, so it is necessary to *install* contributed packages that you might find interesting.

- once a package is installed (you only need to install a package once), it can be 'loaded' and 'attached' to the search path using `library()`.

- As an exercise, try to attach the 'readr', 'dplyr', and 'ggplot2' packages

```
library(readr)
library(dplyr)
library(ggplot2)
```

- If any of these fails with a message like

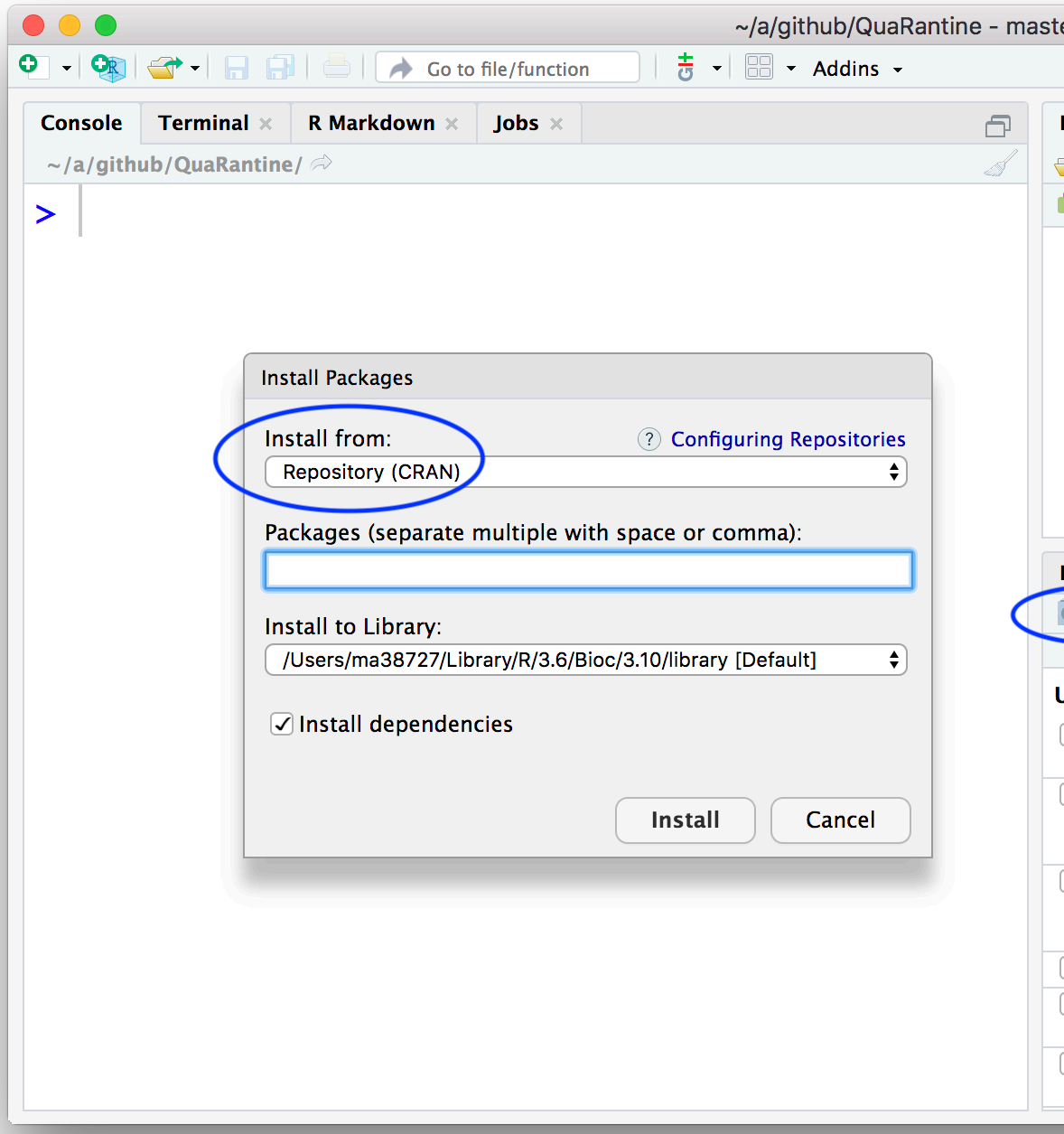
```
library("dplyr")
## Error in library("dplyr") : there is no package called 'dplyr'
```

  it means that the package has not been installed (or that you have a typo in the name of the library!)

- Install any package that failed when `library()` was called with

```
install.packages(c("readr", "dplyr"), repos = "https://cran.r-project.org")
```

  Alternatively, use the *RStudio* interface to select (in the lower right panel, by default) the 'Packages' tab, 'Install' button.


~/a/github/QuaRantine - maste
Go to file/function
Addins
Console
Terminal
R Markdown
Jobs
~/a/github/QuaRantine/
>
Install Packages
Install from:
Configuring Repositories
Repository (CRAN)
Packages (separate multiple with space or comma):
Install to Library:
/Users/ma38727/Library/R/3.6/Bioc/3.10/library [Default]
Install dependencies
Install
Cancel

- One package may use functions from one or more other packages, so when you install, for instance 'dplyr', you may actually install *several* packages.

The 'tidyverse' of packages (20 minutes)

The 'tidyverse' of packages provides a very powerful paradigm for working with data.

- Based on the idea that a first step in data analysis is to transform the data into a standard format. Subsequent steps can then be accomplished in a much more straight-forward way, using a small set of functions.
- Hadley Wickham's 'Tidy Data' paper provides a kind of manifesto for what constitutes tidy data:
  1. Each variable forms a column.
  2. Each observation forms a row.
  3. Each type of observational unit forms a table
- We'll look at the readr package for data input, and the dplyr package for essential data manipulation.

readr for fast data input

- Load (install if necessary!) and attach the readr package

```
library(readr)

## if it fails to load, try
##     install.packages("readr", repos = "https://cran.r-project.org")
```

- Example: US COVID data. N.B., `readr::read_csv()` rather than `read.csv()`

```
url <- "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties
us <- read_csv(url)
## Parsed with column specification:
## cols(
##   date = col_date(format = ""),
##   county = col_character(),
##   state = col_character(),
##   fips = col_character(),
##   cases = col_double(),
##   deaths = col_double()
## )
us
## # A tibble: 164,886 x 6
##    date       county      state      fips  cases deaths
##    <date>     <chr>       <chr>      <chr> <dbl>  <dbl>
##  1 2020-01-21 Snohomish   Washington 53061     1      0
```

```
##  2 2020-01-22 Snohomish   Washington 53061     1      0
##  3 2020-01-23 Snohomish   Washington 53061     1      0
##  4 2020-01-24 Cook        Illinois   17031     1      0
##  5 2020-01-24 Snohomish   Washington 53061     1      0
##  6 2020-01-25 Orange      California 06059     1      0
##  7 2020-01-25 Cook        Illinois   17031     1      0
##  8 2020-01-25 Snohomish   Washington 53061     1      0
##  9 2020-01-26 Maricopa    Arizona    04013     1      0
## 10 2020-01-26 Los Angeles California 06037     1      0
## # ... with 164,876 more rows
```

- The `us` data is now represented as a `tibble`: a nicer `data.frame`
- Note that
  - `date` has been deduced correctly
  - `read_csv()` does not coerce inputs to `factor` (no need to use `stringsAsFactors = FALSE`)
  - The tibble displays nicely (first ten lines, with an indication of total lines)

dplyr for data manipulation

- Load and attach the dplyr package.

```
library(dplyr)
```

- dplyr implements a small number of *verbs* for data transformation
  - A small set of functions that allow very rich data transformation
  - All have the same first argument – the `tibble` to be transformed
  - All allow 'non-standard' evaluation – use the variable name without quotes `"`.
- `filter()` rows that meet specific criteria

```
filter(us, state == "New York", county == "Erie")
## # A tibble: 68 x 6
##    date       county state    fips  cases deaths
##    <date>     <chr>  <chr>    <chr> <dbl>  <dbl>
##  1 2020-03-15 Erie   New York 36029     3      0
##  2 2020-03-16 Erie   New York 36029     6      0
##  3 2020-03-17 Erie   New York 36029     7      0
##  4 2020-03-18 Erie   New York 36029     7      0
##  5 2020-03-19 Erie   New York 36029    28      0
##  6 2020-03-20 Erie   New York 36029    31      0
##  7 2020-03-21 Erie   New York 36029    38      0
##  8 2020-03-22 Erie   New York 36029    54      0
##  9 2020-03-23 Erie   New York 36029    87      0
```

```
## 10 2020-03-24 Erie   New York 36029   107      0
## # ... with 58 more rows
```

- dplyr uses the 'pipe' %>% as a way to chain data and functions together

```
us %>%
    filter(state == "New York", county == "Erie")
## # A tibble: 68 x 6
##    date       county state    fips  cases deaths
##    <date>     <chr>  <chr>    <chr> <dbl>  <dbl>
##  1 2020-03-15 Erie   New York 36029     3      0
##  2 2020-03-16 Erie   New York 36029     6      0
##  3 2020-03-17 Erie   New York 36029     7      0
##  4 2020-03-18 Erie   New York 36029     7      0
##  5 2020-03-19 Erie   New York 36029    28      0
##  6 2020-03-20 Erie   New York 36029    31      0
##  7 2020-03-21 Erie   New York 36029    38      0
##  8 2020-03-22 Erie   New York 36029    54      0
##  9 2020-03-23 Erie   New York 36029    87      0
## 10 2020-03-24 Erie   New York 36029   107      0
## # ... with 58 more rows
```

  - The pipe works by transforming whatever is on the left-hand side of the %>% to the first argument of the function on the right-hand side.
  - Like `filter()`, most dplyr functions take as their first argument a tibble, and return a tibble. So the functions can be chained together, as in the following example.

- `select()` specific columns

```
us %>%
    filter(state == "New York", county == "Erie") %>%
    select(state, county, date, cases)
## # A tibble: 68 x 4
##    state    county date       cases
##    <chr>    <chr>  <date>     <dbl>
##  1 New York Erie   2020-03-15     3
##  2 New York Erie   2020-03-16     6
##  3 New York Erie   2020-03-17     7
##  4 New York Erie   2020-03-18     7
##  5 New York Erie   2020-03-19    28
##  6 New York Erie   2020-03-20    31
##  7 New York Erie   2020-03-21    38
##  8 New York Erie   2020-03-22    54
##  9 New York Erie   2020-03-23    87
## 10 New York Erie   2020-03-24   107
## # ... with 58 more rows
```

Other common verbs (see tomorrow's quarantine)

- `mutate()` (add or update) columns
- `summarize()` one or more columns
- `group_by()` one or more variables when performing computations. `ungroup()` removes the grouping.
- `arrange()` rows based on values in particular column(s); `desc()` in descending order.
- `count()` the number of times values occur

Other 'tidyverse' packages

- Packages adopting the 'tidy' approach to data representation and management are sometimes referred to as the tidyverse.
- ggplot2 implements high-quality data visualization in a way consistent with tidy data representations.
- The tidyr package implements functions that help to transform data to 'tidy' format; we'll use `pivot_longer()` later in the week.

3.2 Day 16 Key tidyverse packages: readr and dplyr

Start a script for today. In the script

- Load the libraries that we will use

```
library(readr)
library(dplyr)
```

- If *R* responds with (similarly for dplyr)

```
Error in library(readr) : there is no package called 'readr'
```

then you'll need to install (just once per *R* installation) the readr pacakge

```
install.packages("readr", repos = "https://cran.r-project.org")
```

Work through the following commands, adding appropriate lines to your script

- Read US COVID data. N.B., `readr::read_csv()` rather than `read.csv()`

```
url <- "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv"
us <- read_csv(url)
## Parsed with column specification:
## cols(
##   date = col_date(format = ""),
##   county = col_character(),
##   state = col_character(),
```

```
##   fips = col_character(),
##   cases = col_double(),
##   deaths = col_double()
## )
us
## # A tibble: 164,886 x 6
##    date       county      state      fips  cases deaths
##    <date>     <chr>       <chr>      <chr> <dbl>  <dbl>
##  1 2020-01-21 Snohomish   Washington 53061     1      0
##  2 2020-01-22 Snohomish   Washington 53061     1      0
##  3 2020-01-23 Snohomish   Washington 53061     1      0
##  4 2020-01-24 Cook        Illinois   17031     1      0
##  5 2020-01-24 Snohomish   Washington 53061     1      0
##  6 2020-01-25 Orange      California 06059     1      0
##  7 2020-01-25 Cook        Illinois   17031     1      0
##  8 2020-01-25 Snohomish   Washington 53061     1      0
##  9 2020-01-26 Maricopa    Arizona    04013     1      0
## 10 2020-01-26 Los Angeles California 06037     1      0
## # ... with 164,876 more rows
```

- `filter()` rows that meet specific criteria

```
us %>%
    filter(state == "New York", county == "Erie")
## # A tibble: 68 x 6
##    date       county state    fips  cases deaths
##    <date>     <chr>  <chr>    <chr> <dbl>  <dbl>
##  1 2020-03-15 Erie   New York 36029     3      0
##  2 2020-03-16 Erie   New York 36029     6      0
##  3 2020-03-17 Erie   New York 36029     7      0
##  4 2020-03-18 Erie   New York 36029     7      0
##  5 2020-03-19 Erie   New York 36029    28      0
##  6 2020-03-20 Erie   New York 36029    31      0
##  7 2020-03-21 Erie   New York 36029    38      0
##  8 2020-03-22 Erie   New York 36029    54      0
##  9 2020-03-23 Erie   New York 36029    87      0
## 10 2020-03-24 Erie   New York 36029   107      0
## # ... with 58 more rows
```

- `select()` specific columns

```
us %>%
    filter(state == "New York", county == "Erie") %>%
    select(state, county, date, cases)
## # A tibble: 68 x 4
##    state    county date       cases
##    <chr>    <chr>  <date>     <dbl>
```

```
##  1 New York Erie   2020-03-15     3
##  2 New York Erie   2020-03-16     6
##  3 New York Erie   2020-03-17     7
##  4 New York Erie   2020-03-18     7
##  5 New York Erie   2020-03-19    28
##  6 New York Erie   2020-03-20    31
##  7 New York Erie   2020-03-21    38
##  8 New York Erie   2020-03-22    54
##  9 New York Erie   2020-03-23    87
## 10 New York Erie   2020-03-24   107
## # ... with 58 more rows
```

- `mutate()` (add or update) columns

```
erie <-
    us %>%
    filter(state == "New York", county == "Erie")
erie %>%
    mutate(new_cases = diff(c(0, cases)))
## # A tibble: 68 x 7
##    date       county state    fips  cases deaths new_cases
##    <date>     <chr>  <chr>    <chr> <dbl>  <dbl>     <dbl>
##  1 2020-03-15 Erie   New York 36029     3      0         3
##  2 2020-03-16 Erie   New York 36029     6      0         3
##  3 2020-03-17 Erie   New York 36029     7      0         1
##  4 2020-03-18 Erie   New York 36029     7      0         0
##  5 2020-03-19 Erie   New York 36029    28      0        21
##  6 2020-03-20 Erie   New York 36029    31      0         3
##  7 2020-03-21 Erie   New York 36029    38      0         7
##  8 2020-03-22 Erie   New York 36029    54      0        16
##  9 2020-03-23 Erie   New York 36029    87      0        33
## 10 2020-03-24 Erie   New York 36029   107      0        20
## # ... with 58 more rows
```

- `summarize()` one or more columns

```
erie %>%
    mutate(new_cases = diff(c(0, cases))) %>%
    summarize(
        duration = n(),
        total_cases = max(cases),
        max_new_cases_per_day = max(new_cases),
        mean_new_cases_per_day = mean(new_cases),
        median_new_cases_per_day = median(new_cases)
    )
## # A tibble: 1 x 5
##   duration total_cases max_new_cases_per~ mean_new_cases_per~ median_new_cases_~
```

```
##      <int>       <dbl>              <dbl>              <dbl>              <dbl>
## 1       68        5270                217               77.5               77.
```

- `group_by()` one or more variables when performing computations

```
us_county_cases <-
    us %>%
    group_by(county, state) %>%
    summarize(total_cases = max(cases))

us_state_cases <-
    us_county_cases %>%
    group_by(state) %>%
    summarize(total_cases = sum(total_cases))
```

- `arrange()` based on a particular column; `desc()` in descending order.

```
us_county_cases %>%
    arrange(desc(total_cases))
## # A tibble: 2,975 x 3
## # Groups:   county [1,740]
##    county        state         total_cases
##    <chr>         <chr>               <dbl>
##  1 New York City New York           200507
##  2 Cook          Illinois            67551
##  3 Los Angeles   California          42037
##  4 Nassau        New York            39487
##  5 Suffolk       New York            38553
##  6 Westchester   New York            32672
##  7 Philadelphia  Pennsylvania        20700
##  8 Middlesex     Massachusetts       19930
##  9 Wayne         Michigan            19538
## 10 Hudson        New Jersey          17814
## # ... with 2,965 more rows

us_state_cases %>%
    arrange(desc(total_cases))
## # A tibble: 55 x 2
##    state         total_cases
##    <chr>               <dbl>
##  1 New York           363404
##  2 New Jersey         155312
##  3 Illinois           103397
##  4 Massachusetts       90761
##  5 California          88520
##  6 Pennsylvania        69260
##  7 Texas               53575
```

```
##  8 Michigan            53483
##  9 Florida             48675
## 10 Maryland            43661
## # ... with 45 more rows
```

- `count()` the number of times values occur (duration of the pandemic?)

```
us %>%
    count(county, state) %>%
    arrange(desc(n))
## # A tibble: 2,975 x 3
##    county        state             n
##    <chr>         <chr>         <int>
##  1 Snohomish     Washington      122
##  2 Cook          Illinois        119
##  3 Orange        California      118
##  4 Los Angeles   California      117
##  5 Maricopa      Arizona         117
##  6 Santa Clara   California      112
##  7 Suffolk       Massachusetts   111
##  8 San Francisco California      110
##  9 Dane          Wisconsin       107
## 10 San Diego     California      102
## # ... with 2,965 more rows
```

3.3 Day 17 Visualization with ggplot2

Setup

Load packages we'll use today

```
library(readr)
library(dplyr)
library(ggplot2)
library(tidyr)
```

Remember that packages need to be installed before loading; if you see…

```
> library(ggplot2)
## Error in library(ggplot2) : there is no package called 'ggplot2'
```

…then you'll need to install the package and try again

```
install.packages("ggplot2", repos = "https://cran.r-project.org")
library(ggplot2)
```

Input data using `readr::read_csv()`

```
url <- "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv"
us <- read_csv(url)
## Parsed with column specification:
## cols(
##   date = col_date(format = ""),
##   county = col_character(),
##   state = col_character(),
##   fips = col_character(),
##   cases = col_double(),
##   deaths = col_double()
## )
```

Create the Erie county subset, with columns `new_cases` and `new_deaths`

```
erie <-
    us %>%
    filter(county == "Erie", state == "New York") %>%
    mutate(
        new_cases = diff(c(0, cases)),
        new_deaths = diff(c(0, deaths))
    )
```

ggplot2 essentials

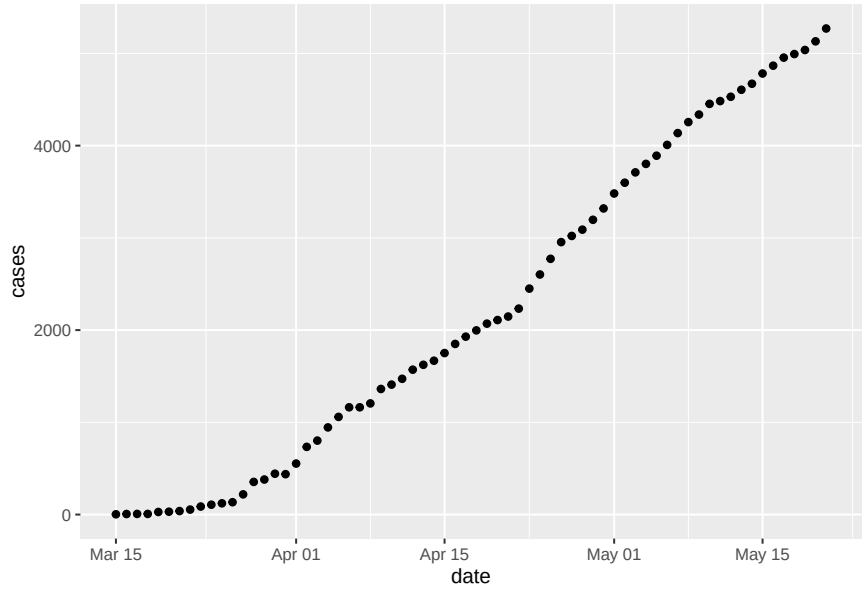
The 'gg' in ggplot2

- 'Grammar of Graphics' – a formal, scholarly system for describing and creating graphics.
- See the usage guide, and the data visualization chapter of R for Data Science.
- The reference section of the usage guide provides a good entry point

A first plot

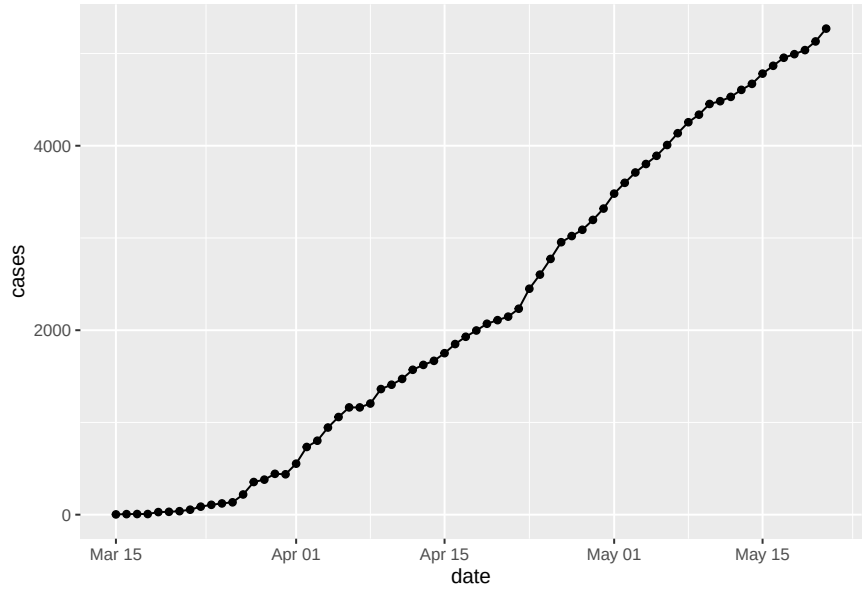
- Specify the data to use. Do this by (a) providing the tibble containing the data (`erie`) and (b) communicating the 'aesthetics' of the overall graph by specifying the `x` and `y` data columns – `ggplot(erie, aes(x = date, y = new_cases))`
- Add a `geom_` describing the geometric object used to represent the data, e.g., use `geom_point()` to represent the data as points.

```
ggplot(erie, aes(date, cases)) +
    geom_point()
```



- Note that the plot is assembled by adding elements using a simple `+`. Connect the points with `geom_line()`.

```
ggplot(erie, aes(date, cases)) +
    geom_point() +
    geom_line()
```

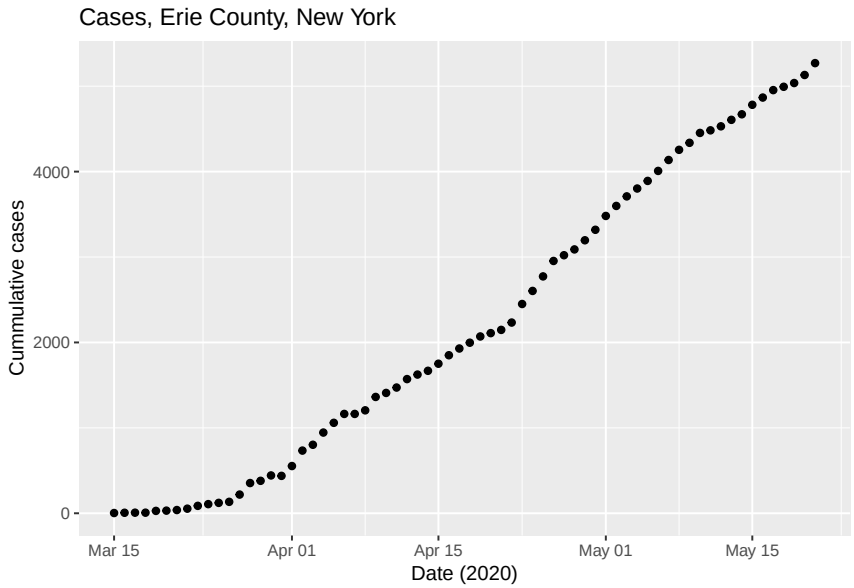


- Plots can actually be captured in a variable, e.g., `p`

```
p <- ggplot(erie, aes(date, cases)) +
    geom_point()
```

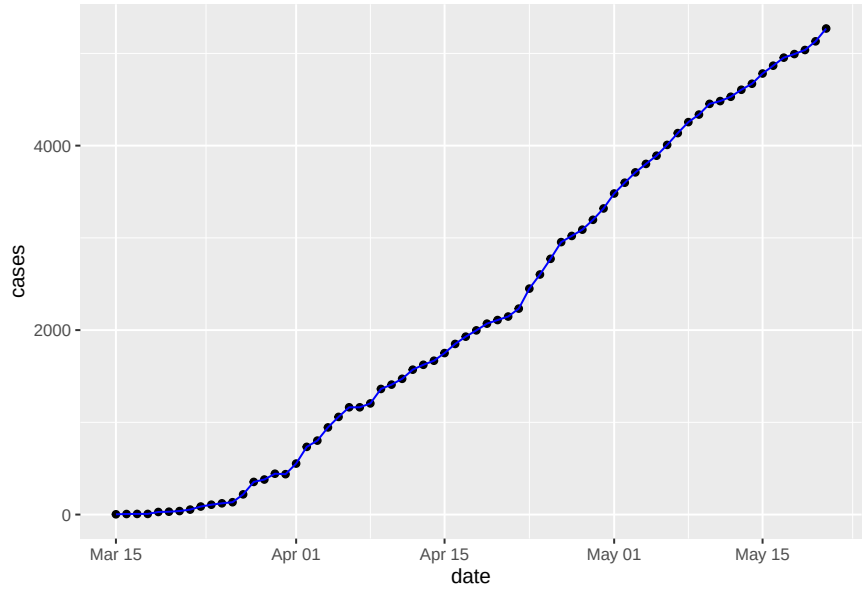
... and then updated and displayed

```
p +
    xlab("Date (2020)") +
    ylab("Cummulative cases") +
    ggtitle("Cases, Erie County, New York")
```



- Arguments to each `geom` influece how the geometry is displayed, e.g.,

```
p +
    geom_line(color = "blue")
```



COVID-19 in Erie county

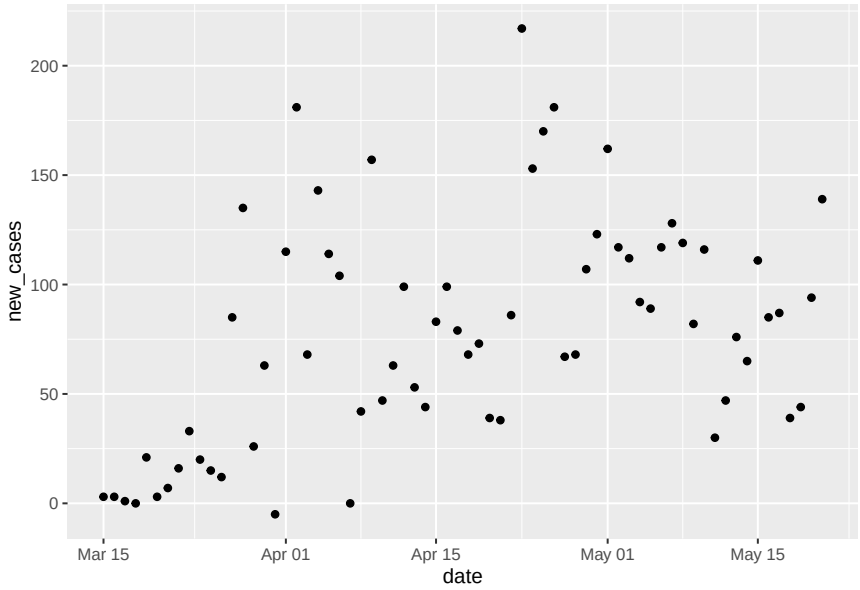
New cases

- Create a base plot using `new_cases)`

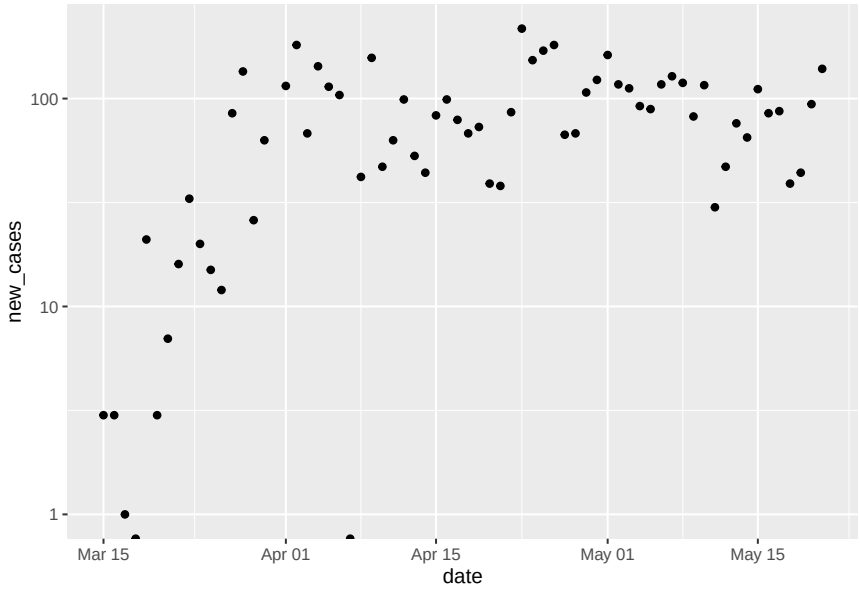
```
p <- ggplot(erie, aes(date, new_cases)) +
    geom_point()
```

- Visualize on a linear and a log-transformed y-axis

```
p # linear
```



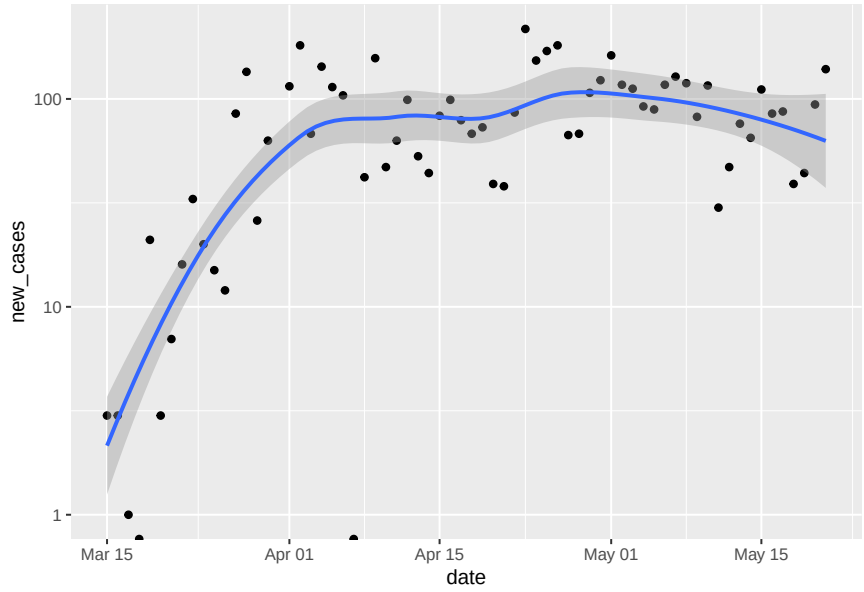

```
p + scale_y_log10()
## Warning in self$trans$transform(x): NaNs produced
## Warning: Transformation introduced infinite values in continuous y-axis
## Warning: Removed 1 rows containing missing values (geom_point).
```



Add a smoothed line to the plot. By default the smoothed line is a local regression appropriate for exploratory data analysis. Note the confidence bands displayed in the plot, and how they convey a measure of certainty

about the fit.

```
p +
    scale_y_log10() +
    geom_smooth()
## Warning in self$trans$transform(x): NaNs produced
## Warning: Transformation introduced infinite values in continuous y-axis
## Warning in self$trans$transform(x): NaNs produced
## Warning: Transformation introduced infinite values in continuous y-axis
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
## Warning: Removed 3 rows containing non-finite values (stat_smooth).
## Warning: Removed 1 rows containing missing values (geom_point).
```



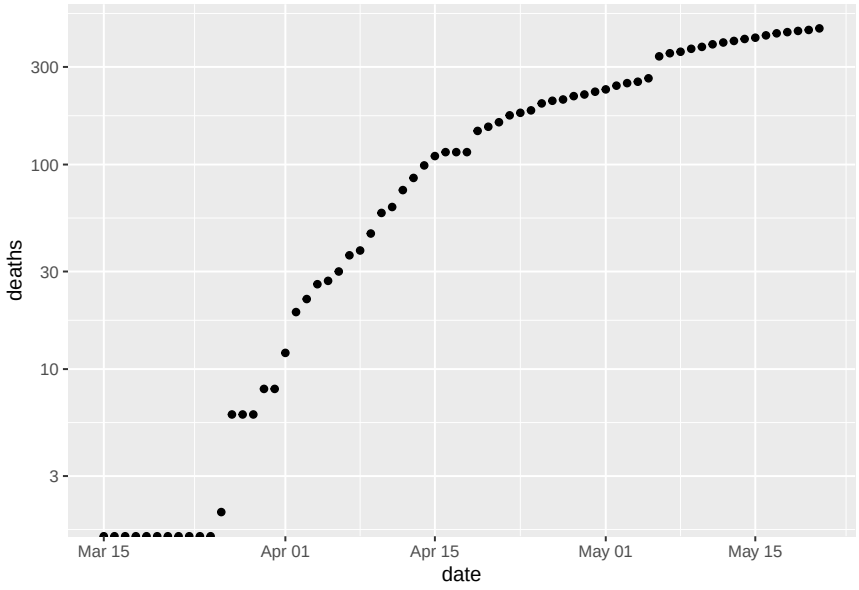
- Reflect on the presentation of data, especially how log-transformation and a clarifies our impression of the local progress of the pandemic.

- The local regression used by `geom_smooth()` can be replaced by a linear regressin with `geom_smooth(method = "lm")`. Create this plot and reflect on the assumptions and suitability of a linear model for this data.

New cases and mortality

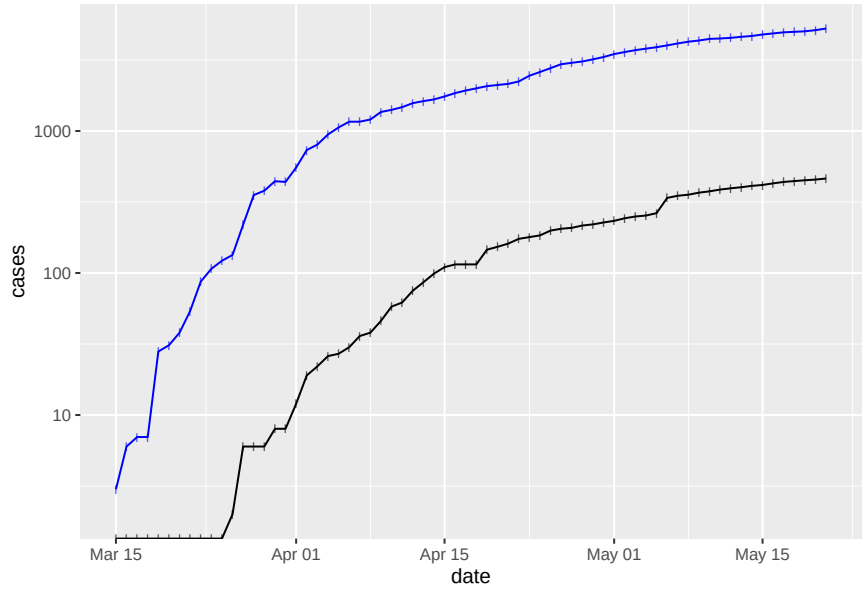
- It's easy to separately plot `deaths` by updating the aesthetic in `ggplot()`

```
ggplot(erie, aes(date, deaths)) +
    scale_y_log10() +
    geom_point()
## Warning: Transformation introduced infinite values in continuous y-axis
```



- What about ploting cases and deaths? Move the `aes()` argument to the individual geometries. Use different colors for each geometry

```
ggplot(erie) +
    scale_y_log10() +
    geom_point(aes(date, cases), shape = "|", color = "blue") +
    geom_line(aes(date, cases), color = "blue") +
    geom_point(aes(date, deaths), shape = "|") +
    geom_line(aes(date, deaths))
## Warning: Transformation introduced infinite values in continuous y-axis

## Warning: Transformation introduced infinite values in continuous y-axis
```



Deaths lag behind cases by a week or so.

'Long' data and an alternative approach to plotting multiple curves.

- Let's simplify the data to just the columns of interest for this exercise

```
simple <-
    erie %>%
    select(date, cases, deaths)
simple
## # A tibble: 68 x 3
##    date       cases deaths
##    <date>     <dbl>  <dbl>
##  1 2020-03-15     3      0
##  2 2020-03-16     6      0
##  3 2020-03-17     7      0
##  4 2020-03-18     7      0
##  5 2020-03-19    28      0
##  6 2020-03-20    31      0
##  7 2020-03-21    38      0
##  8 2020-03-22    54      0
##  9 2020-03-23    87      0
## 10 2020-03-24   107      0
## # ... with 58 more rows
```

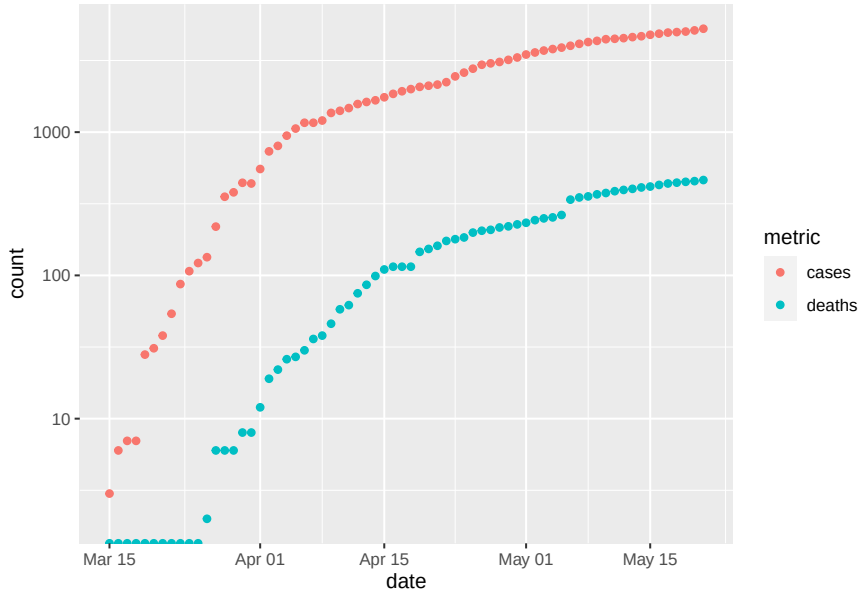
- Use `tidyr::pivot_longer()` to transform the two columns 'cases' and 'deaths' into a column that indicates 'name' and 'value'; 'name' is 'cases' when the corresponding 'value' came from the 'cases' column, and similarly for 'deaths'. See the help page `?tidyr::pivot_longer` and tomorrow's

exercises for more on `pivot_longer()`.

```
longer <-
    simple %>%
    pivot_longer(
        c("cases", "deaths"),
        names_to = "metric",
        values_to = "count"
    )
longer
## # A tibble: 136 x 3
##    date       metric count
##    <date>     <chr>  <dbl>
##  1 2020-03-15 cases      3
##  2 2020-03-15 deaths     0
##  3 2020-03-16 cases      6
##  4 2020-03-16 deaths     0
##  5 2020-03-17 cases      7
##  6 2020-03-17 deaths     0
##  7 2020-03-18 cases      7
##  8 2020-03-18 deaths     0
##  9 2020-03-19 cases     28
## 10 2020-03-19 deaths     0
## # ... with 126 more rows
```

- Plot `date` and `value, coloring points byname'`

```
ggplot(longer, aes(date, count, color = metric)) +
    scale_y_log10() +
    geom_point()
## Warning: Transformation introduced infinite values in continuous y-axis
```



COVID-19 in New York State

We'll explore 'facet' visualizations, which create a panel of related plots

Setup

- From the US data, extract Erie and Westchester counties and New York City. Use `coi` ('counties of interest') as a variable to hold this data

```
coi <-
    us %>%
    filter(
        county %in% c("Erie", "Westchester", "New York City"),
        state == "New York"
    ) %>%
    select(date, county, cases, deaths)
coi
## # A tibble: 229 x 4
##    date       county        cases deaths
##    <date>     <chr>         <dbl>  <dbl>
##  1 2020-03-01 New York City     1      0
##  2 2020-03-02 New York City     1      0
##  3 2020-03-03 New York City     2      0
##  4 2020-03-04 New York City     2      0
##  5 2020-03-04 Westchester       9      0
##  6 2020-03-05 New York City     4      0
##  7 2020-03-05 Westchester      17      0
##  8 2020-03-06 New York City     5      0
```

```
##  9 2020-03-06 Westchester      33      0
## 10 2020-03-07 New York City    12      0
## # ... with 219 more rows
```

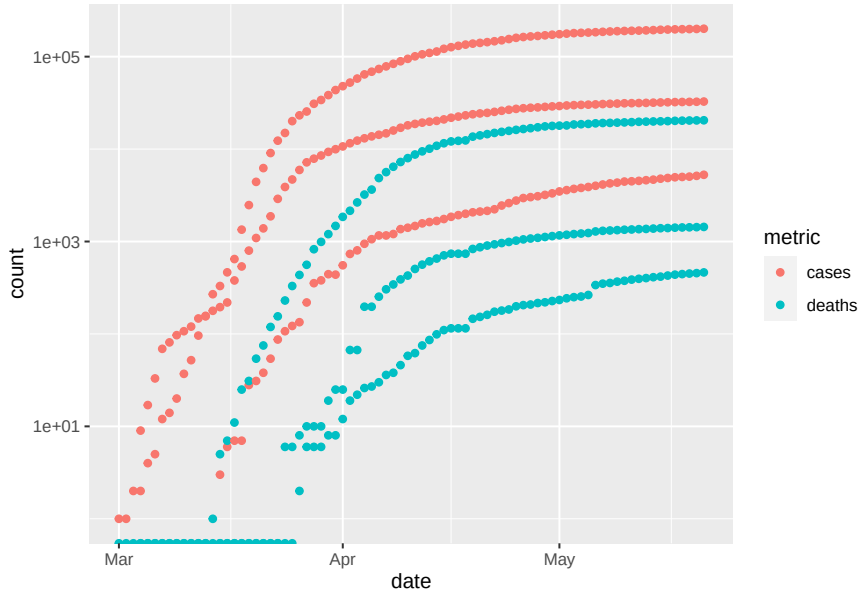
- Pivot `cases` and `deaths` into long form

```
coi_longer <-
    coi %>%
    pivot_longer(
        c("cases", "deaths"),
        names_to = "metric",
        values_to = "count"
    )
coi_longer
## # A tibble: 458 x 4
##    date       county        metric count
##    <date>     <chr>         <chr>  <dbl>
##  1 2020-03-01 New York City cases      1
##  2 2020-03-01 New York City deaths     0
##  3 2020-03-02 New York City cases      1
##  4 2020-03-02 New York City deaths     0
##  5 2020-03-03 New York City cases      2
##  6 2020-03-03 New York City deaths     0
##  7 2020-03-04 New York City cases      2
##  8 2020-03-04 New York City deaths     0
##  9 2020-03-04 Westchester   cases      9
## 10 2020-03-04 Westchester   deaths     0
## # ... with 448 more rows
```

Visualization

- We can plot cases and deaths of each county...

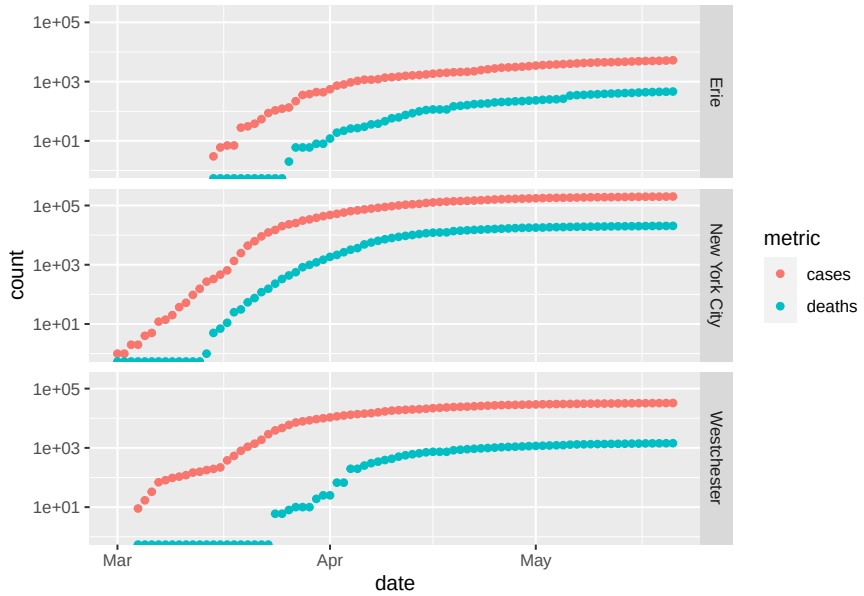
```
p <-
    ggplot(coi_longer, aes(date, count, color = metric)) +
    scale_y_log10() +
    geom_point()
p
## Warning: Transformation introduced infinite values in continuous y-axis
```



... but this is too confusing.

- Separate each county into a facet

```
p + facet_grid(rows=vars(county))
## Warning: Transformation introduced infinite values in continuous y-axis
```



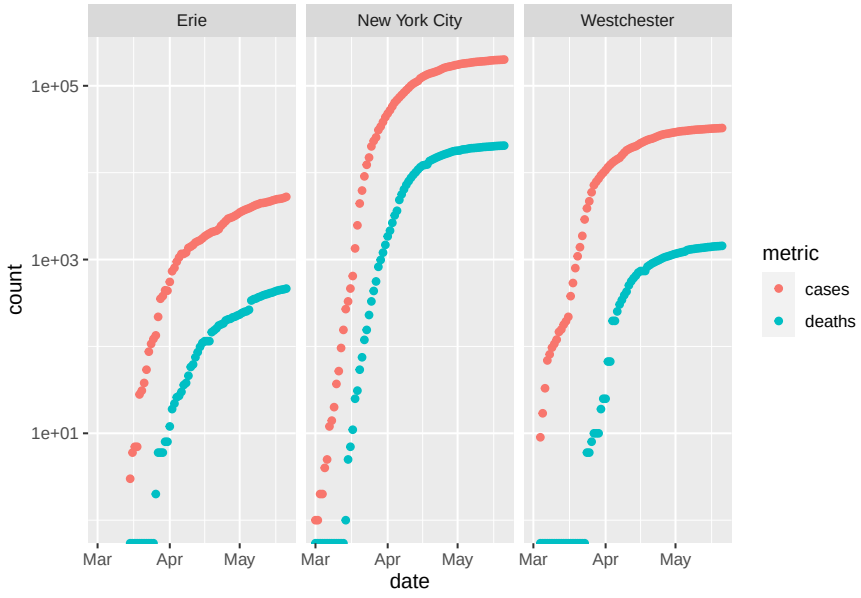
Note the common scales on the x and y axes.

- Plotting counties as 'rows' of the graph emphasize temporal comparisons

– e.g., the earlier onset of the pandemic in Westchester and New York City compared to Erie, and perhaps longer lag between new cases and deaths in Westchester.

- Plotting countes as 'columns' emphasizes comparison between number of cases and deaths – there are many more cases in New York City than in Erie County.

```
p + facet_grid(cols=vars(county))
## Warning: Transformation introduced infinite values in continuous y-axis
```



COVID-19 nationally

Setup

- Summarize the total (maximum) number of cases in each county and state

```
county_summary <-
    us %>%
    group_by(county, state) %>%
    summarize(
        cases = max(cases),
        deaths = max(deaths)
    )
county_summary
## # A tibble: 2,975 x 4
## # Groups:   county [1,740]
##    county    state          cases deaths
```

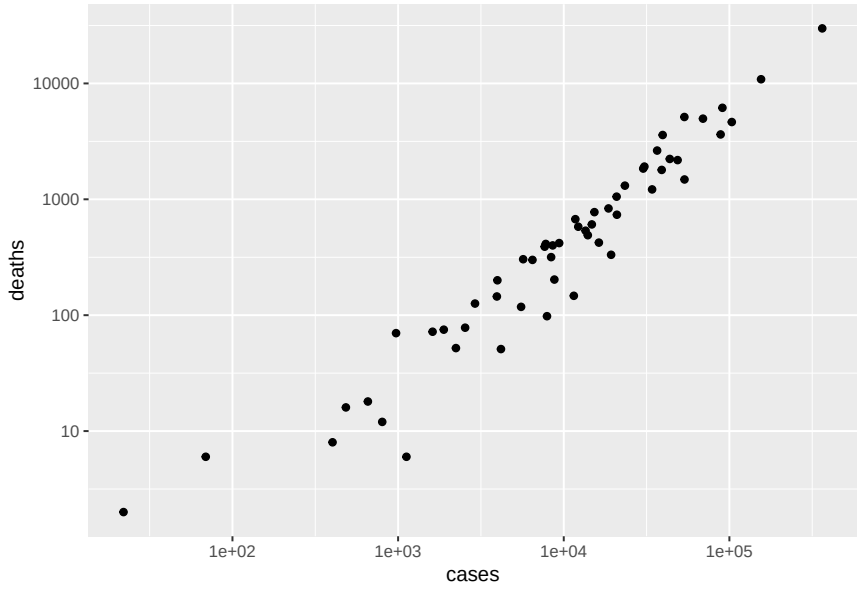
```
##    <chr>     <chr>          <dbl>  <dbl>
##  1 Abbeville South Carolina    36      0
##  2 Acadia    Louisiana        269     15
##  3 Accomack  Virginia         709     11
##  4 Ada       Idaho            792     23
##  5 Adair     Iowa               6      0
##  6 Adair     Kentucky          95     18
##  7 Adair     Missouri          34      0
##  8 Adair     Oklahoma          78      3
##  9 Adams     Colorado        2759    108
## 10 Adams     Idaho              3      0
## # ... with 2,965 more rows
```

- Now summarize the number of cases per state

```
state_summary <-
    county_summary %>%
    group_by(state) %>%
    summarize(
        cases = sum(cases),
        deaths = sum(deaths)
    ) %>%
    arrange(desc(cases))
state_summary
## # A tibble: 55 x 3
##    state          cases deaths
##    <chr>          <dbl>  <dbl>
##  1 New York      363404  29840
##  2 New Jersey    155312  10864
##  3 Illinois      103397   4643
##  4 Massachusetts  90761   6160
##  5 California     88520   3628
##  6 Pennsylvania   69260   4966
##  7 Texas          53575   1483
##  8 Michigan       53483   5131
##  9 Florida        48675   2179
## 10 Maryland       43661   2230
## # ... with 45 more rows
```

- Plot the relationship between cases and deaths as a scatter plot

```
ggplot(state_summary, aes(cases, deaths)) +
    scale_x_log10() +
    scale_y_log10() +
    geom_point()
```



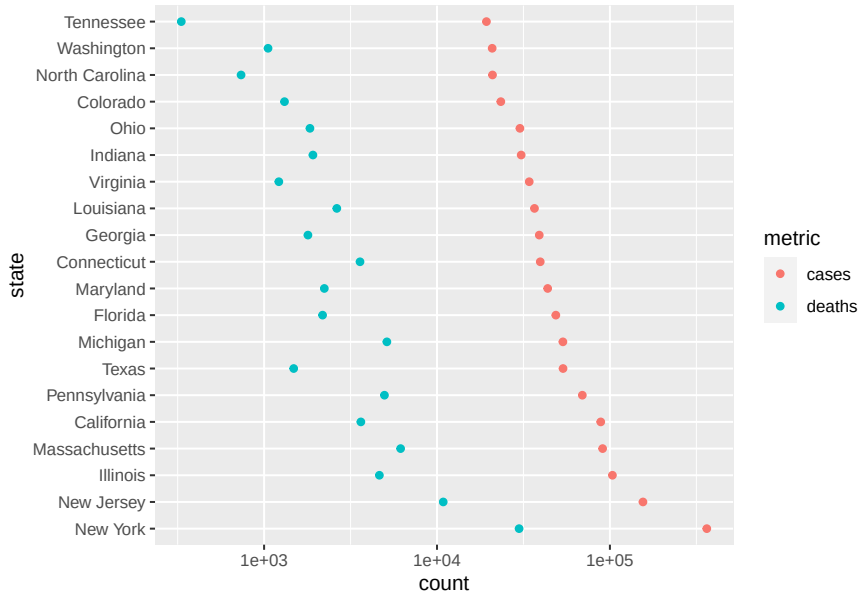
- Create a 'long' version of the state summary. The transformations include making 'state' a factor with the 'levels' ordered from most- to least-affected state. This is a 'trick' so that states are ordered, when displayed, from most to least affected. The transformations also choose only the 20 most-affected states using `head(20)`.

```
state_longer <-
    state_summary %>%
    mutate(
        ## this 'trick' causes 'state' to be ordered from most to
        ## least cases, rather than alphabetically
        state = factor(state, levels = state)
    ) %>%
    head(20) %>% # look at the 20 states with the most cases
    pivot_longer(
        c("cases", "deaths"),
        names_to = "metric",
        values_to = "count"
    )
state_longer
## # A tibble: 40 x 3
##    state         metric  count
##    <fct>         <chr>   <dbl>
##  1 New York      cases  363404
##  2 New York      deaths  29840
##  3 New Jersey    cases  155312
##  4 New Jersey    deaths  10864
```

```
##  5 Illinois      cases  103397
##  6 Illinois      deaths   4643
##  7 Massachusetts cases   90761
##  8 Massachusetts deaths   6160
##  9 California    cases   88520
## 10 California    deaths   3628
## # ... with 30 more rows
```

- Use a dot plot to provide an alternative representation that is more easy to associated statsistics with individual states

```
ggplot(state_longer, aes(x = count, y = state, color = metric)) +
    scale_x_log10() +
    geom_point()
```



3.4 Day 18 Worldwide COVID data

Setup

- Start a new script and load the packages we'll use

```
library(readr)
library(dplyr)
library(ggplot2)
library(tidyr)      # specialized functions for transforming tibbles
```

These packages should have been installed during previous quarantines.

Source

- CSSE at Johns Hopkins University, available on github

```
hopkins = "https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_c
csv <- read_csv(hopkins)
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   `Province/State` = col_character(),
##   `Country/Region` = col_character()
## )
## See spec(...) for full column specifications.
```

'Tidy' data

- The data has initial columns describing region, and then a column for each date of the pandemic. There are 266 rows, corresponding to the different regions covered by the database.
- We want instead to 'pivot' the data, so that each row represents cases in a particular region on a particular date, analogous to the way the US data we have been investigating earlier has been arranged.
- tidyr provides functions for manipulating a `tibble` into 'tidy' format.
- `tidyr::pivot_longer()` takes a 'wide' data frame like `csv`, and allows us to transform it to the 'long' format we are interested in.
  - I discovered how to work with `pivot_longer()` using its help page `?tidyr::pivot_longer`
  - The first argument represents columns to pivot or, as a convenience when these are negative values, columns we *do not* want to pivot. We *do not* want to pivot columns 1 through 4, so this argument will be `-(1:4)`.
  - The `names_to` argument is the column name we want to use to refer to the names of the columns that we *do* pivot. We'll pivot the columns that have a date in them, so it makes sense to use `names_to = "date"`.
  - The `values_to` argument is the column name we want to use for the pivoted values. Since the values in the main part of `csv` are the number of cases observed, we'll use `values_to = "cases"`
- Here's what we have after pivoting

```
csv %>%
    pivot_longer(-(1:4), names_to = "date", values_to = "cases")
## # A tibble: 32,186 x 6
##    `Province/State` `Country/Region`   Lat  Long date    cases
```

```
##    <chr>            <chr>            <dbl> <dbl> <chr>   <dbl>
##  1 <NA>             Afghanistan         33    65 1/22/20     0
##  2 <NA>             Afghanistan         33    65 1/23/20     0
##  3 <NA>             Afghanistan         33    65 1/24/20     0
##  4 <NA>             Afghanistan         33    65 1/25/20     0
##  5 <NA>             Afghanistan         33    65 1/26/20     0
##  6 <NA>             Afghanistan         33    65 1/27/20     0
##  7 <NA>             Afghanistan         33    65 1/28/20     0
##  8 <NA>             Afghanistan         33    65 1/29/20     0
##  9 <NA>             Afghanistan         33    65 1/30/20     0
## 10 <NA>             Afghanistan         33    65 1/31/20     0
## # ... with 32,176 more rows
```

- We'd like to further clean this up data
  - Format our newly created 'date' column (using `as.Date()`, but with a `format=` argument appropriate for the format of the dates in this data set)
  - Re-name, for convenience, the `County/Region` column as just `country`. This can be done with `rename(country = "Country/Region")`
  - Select only columns of interest – `country`, `date`, `cases`
  - Some countries have multiple rows, because the data is a provincial or state levels, so we would like to sum all cases, grouped by `country` and `date`

```
world <-
    csv %>%
    pivot_longer(-(1:4), names_to = "date", values_to = "cases") %>%
    mutate(date = as.Date(date, format = "%m/%d/%y")) %>%
    rename(country = "Country/Region") %>%
    group_by(country, date) %>%
    summarize(cases = sum(cases))
world
## # A tibble: 22,748 x 3
## # Groups:   country [188]
##    country     date       cases
##    <chr>       <date>     <dbl>
##  1 Afghanistan 2020-01-22     0
##  2 Afghanistan 2020-01-23     0
##  3 Afghanistan 2020-01-24     0
##  4 Afghanistan 2020-01-25     0
##  5 Afghanistan 2020-01-26     0
##  6 Afghanistan 2020-01-27     0
##  7 Afghanistan 2020-01-28     0
```

```
##  8 Afghanistan 2020-01-29      0
##  9 Afghanistan 2020-01-30      0
## 10 Afghanistan 2020-01-31      0
## # ... with 22,738 more rows
```

- Let's also calculate `new_cases` by country
  - Use `group_by()` to perform the `new_cases` computation for each country
  - Use `mutate()` to calculate the new variable
  - Use `ungroup()` to remove the grouping variable, so it doesn't unexpectedly influence other calculations
  - re-assign the updated `tibble` to the variable `world`

```
world <-
    world %>%
    group_by(country) %>%
    mutate(new_cases = diff(c(0, cases))) %>%
    ungroup()
```

Exploration

- Use `group_by()` and `summarize()` to find the maximum (total) number of cases, and `arrange()` `these in`desc()'ending order

```
world %>%
    group_by(country) %>%
    summarize(n = max(cases)) %>%
    arrange(desc(n))
## # A tibble: 188 x 2
##    country              n
##    <chr>            <dbl>
##  1 US             1577147
##  2 Russia          317554
##  3 Brazil          310087
##  4 United Kingdom  252246
##  5 Spain           233037
##  6 Italy           228006
##  7 France          181951
##  8 Germany         179021
##  9 Turkey          153548
## 10 Iran            129341
## # ... with 178 more rows
```

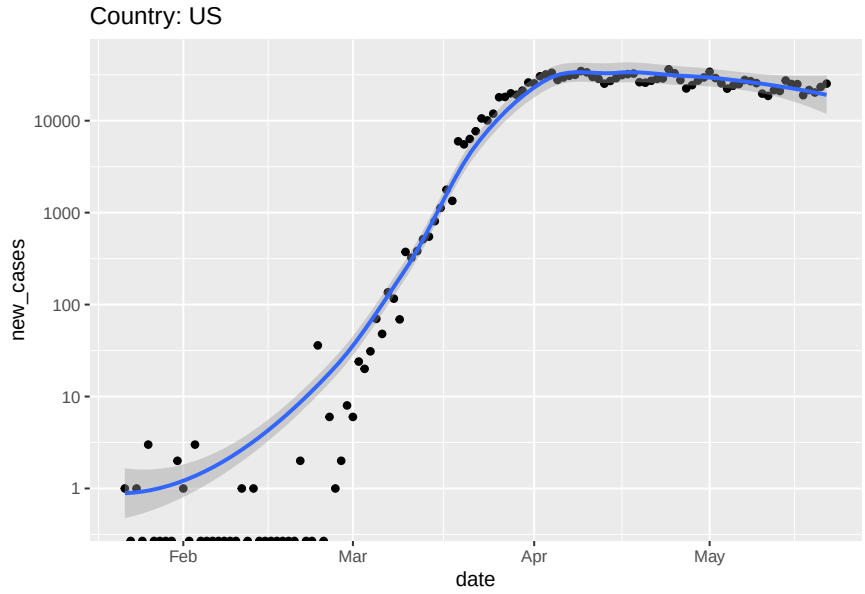
Visualization

- Start by creating a subset, e.g., the US

```
country <- "US"
us <-
    world %>%
    filter(country == "US")
```

- Use ggplot2 to visualize the progression of the pandemic

```
ggplot(us, aes(date, new_cases)) +
    scale_y_log10() +
    geom_point() +
    geom_smooth() +
    ggtitle(paste("Country:", country))
## Warning: Transformation introduced infinite values in continuous y-axis

## Warning: Transformation introduced infinite values in continuous y-axis
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
## Warning: Removed 25 rows containing non-finite values (stat_smooth).
```



It seems like it would be convenient to capture our data cleaning and visualization steps into separate functions that can be re-used, e.g., on different days or for different visualizations.

- write a function for data retrieval and cleaning

```
get_world_data <-
    function()
{
    ## read data from Hopkins' github repository
```

```
    hopkins = "https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/cs
    csv <- read_csv(hopkins)

    ## 'tidy' the data
    world <-
        csv %>%
        pivot_longer(-(1:4), names_to = "date", values_to = "cases") %>%
        mutate(date = as.Date(date, format = "%m/%d/%y")) %>%
        rename(country = "Country/Region")

    ## sum cases across regions within aa country
    world <-
        world %>%
        group_by(country, date) %>%
        summarize(cases = sum(cases))

    ## add `new_cases`, and return the result
    world %>%
        group_by(country) %>%
        mutate(new_cases = diff(c(0, cases))) %>%
        ungroup()
}
```

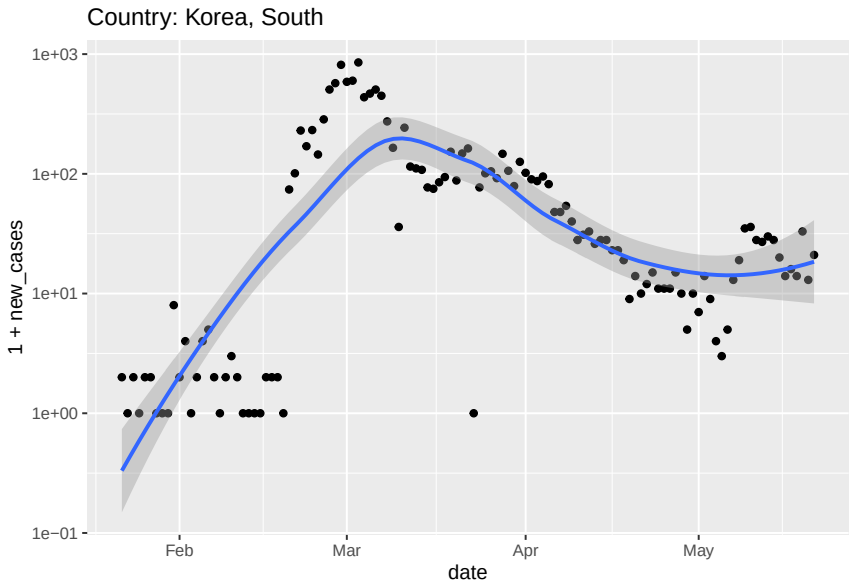
- ...and for plotting by country

```
plot_country <-
    function(tbl, view_country = "US")
{
    country_title <- paste("Country:", view_country)

    ## subset to just this country
    country_data <-
        tbl %>%
        filter(country == view_country)

    ## plot
    country_data %>%
        ggplot(aes(date, 1 + new_cases)) +
        scale_y_log10() +
        geom_point() +
        ## add method and formula to quieten message
        geom_smooth(method = "loess", formula = y ~ x) +
        ggtitle(country_title)
}
```

- Note that, because the first argument of `plot_country()` is a tibble, the

output of `get_world_data()` can be used as the input of `plot_country()`, and can be piped together, e.g.,

```
world <- get_world_data()
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   `Province/State` = col_character(),
##   `Country/Region` = col_character()
## )
## See spec(...) for full column specifications.
world %>% plot_country("Korea, South")
```



3.5 Day 19 (Friday) Zoom check-in

3.5.1 Logistics

- Stick around after class to ask any questions.
- Remember Microsoft Teams for questions during the week.

3.5.2 Review and trouble shoot (40 minutes)

Setup

```
library(readr)
library(dplyr)
library(tibble)
```

```
library(tidyr)

url <- "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv"
us <- read_csv(url)
```

Packages

- `install.packages()` versus `library()`
- Symbol resolution: `dplyr::filter()`

The `tibble`

- Compact, informative display
- Generally, no rownames

```
mtcars %>%
    as_tibble(rownames = "model")
## # A tibble: 32 x 12
##    model          mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  car
##    <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl
##  1 Mazda RX4     21       6  160    110  3.9   2.62  16.5     0     1     4
##  2 Mazda RX4 ~   21       6  160    110  3.9   2.88  17.0     0     1     4
##  3 Datsun 710    22.8     4  108     93  3.85  2.32  18.6     1     1     4
##  4 Hornet 4 D~   21.4     6  258    110  3.08  3.22  19.4     1     0     3
##  5 Hornet Spo~   18.7     8  360    175  3.15  3.44  17.0     0     0     3
##  6 Valiant       18.1     6  225    105  2.76  3.46  20.2     1     0     3
##  7 Duster 360    14.3     8  360    245  3.21  3.57  15.8     0     0     3
##  8 Merc 240D     24.4     4  147.    62  3.69  3.19  20       1     0     4
##  9 Merc 230      22.8     4  141.    95  3.92  3.15  22.9     1     0     4
## 10 Merc 280      19.2     6  168.   123  3.92  3.44  18.3     1     0     4
## # ... with 22 more rows
```

Verbs

- `filter()`: filter rows meeting specific criteria

```
erie <-
    us %>%
    filter(county == "Erie", state == "New York")
```

- `select()`: select column
- `summarize()`: summarize column(s) to a single value
  - `n()`: number of rows in the tibble
- `mutate()`: modify and create new columns
- `arrange()`: arrange rows so that specific columns are in order
  - `desc()`: arrange in descending order. Applies to individual columns

- `group_by()` / `ungroup()`: identify groups of data, e.g., for `summarize()` operations

```
us %>%
    ## group by county & state, summarize by MAX (total) cases,
    ## deaths across each date
    group_by(county, state) %>%
    summarize(cases = max(cases), deaths = max(deaths)) %>%
    ## group the _result_ by state, summarize by SUM of cases,
    ## deaths in each county
    group_by(state) %>%
    summarize(cases = sum(cases), deaths = sum(deaths)) %>%
    ## arrange from most to least affected states
    arrange(desc(cases))
## # A tibble: 55 x 3
##    state          cases deaths
##    <chr>          <dbl>  <dbl>
##  1 New York      363404  29840
##  2 New Jersey    155312  10864
##  3 Illinois      103397   4643
##  4 Massachusetts  90761   6160
##  5 California     88520   3628
##  6 Pennsylvania   69260   4966
##  7 Texas          53575   1483
##  8 Michigan       53483   5131
##  9 Florida        48675   2179
## 10 Maryland       43661   2230
## # ... with 45 more rows
```

  - `ungroup()` to remove grouping

- In scripts, it seems like the best strategy, for legibility, is to evaluate one verb per line, and to chain not too many verbs together into logical 'phrases'.

```
## worst?
state <- us %>% group_by(county, state) %>% summarize(cases = max(cases), deaths = max(death

## better?
state <-
    us %>%
    group_by(county, state) %>%
    summarize(cases = max(cases), deaths = max(deaths)) %>%
    group_by(state) %>%
    summarize(cases = sum(cases), deaths = sum(deaths)) %>%
    arrange(desc(cases))
```

```
## best?
county_state <-
    us %>%
    group_by(county, state) %>%
    summarize(cases = max(cases), deaths = max(deaths))

state <-
    county_state %>%
    group_by(state) %>%
    summarize(cases = sum(cases), deaths = sum(deaths)) %>%
    arrange(desc(cases))
```

Cleaning: `tidyr::pivot_longer()`

- `cases` and `deaths` are both 'counts', so could perhaps be represented in a single 'value' column with a corresponding 'key' (name) column telling us whether the count is a 'case' or 'death'

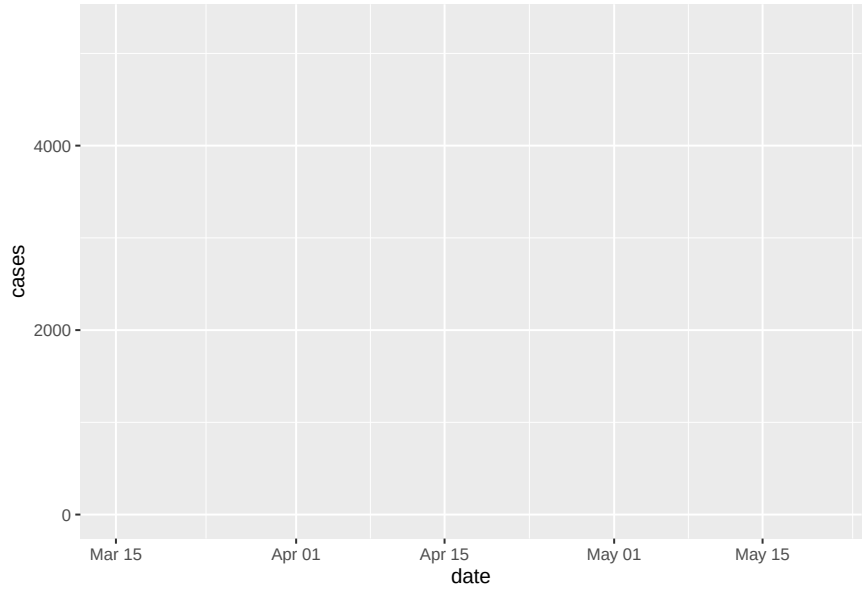
```
erie %>%
    pivot_longer(
        c("cases", "deaths"),
        names_to = "event",
        values_to = "count"
    )
## # A tibble: 136 x 6
##    date       county state    fips  event  count
##    <date>     <chr>  <chr>    <chr> <chr>  <dbl>
##  1 2020-03-15 Erie   New York 36029 cases      3
##  2 2020-03-15 Erie   New York 36029 deaths     0
##  3 2020-03-16 Erie   New York 36029 cases      6
##  4 2020-03-16 Erie   New York 36029 deaths     0
##  5 2020-03-17 Erie   New York 36029 cases      7
##  6 2020-03-17 Erie   New York 36029 deaths     0
##  7 2020-03-18 Erie   New York 36029 cases      7
##  8 2020-03-18 Erie   New York 36029 deaths     0
##  9 2020-03-19 Erie   New York 36029 cases     28
## 10 2020-03-19 Erie   New York 36029 deaths     0
## # ... with 126 more rows
```

Visualization

```
library(ggplot2)
```

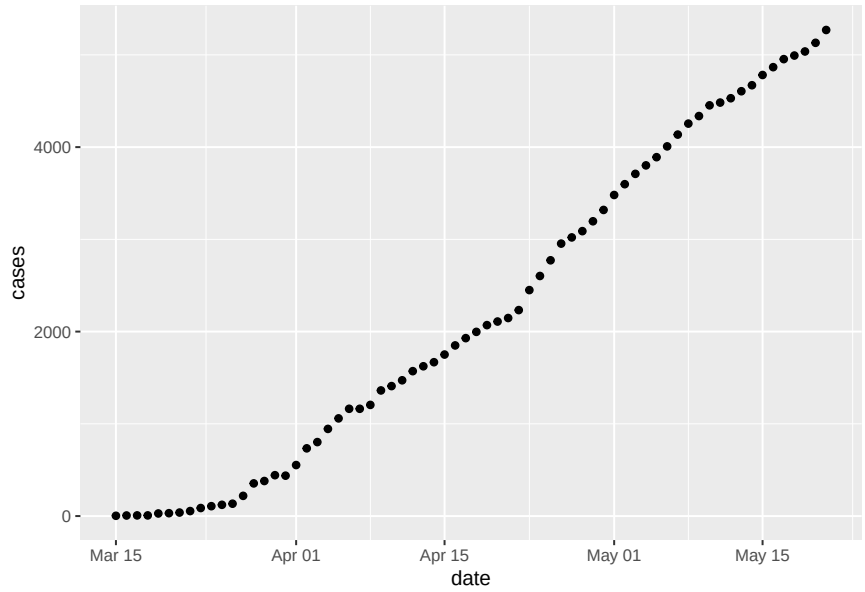
- `ggplot()`: where does the data come from?
- `aes()`: what parts of the data are we going to plot
  - a tibble and `aes()` are enough to know the overall layout of the plot

```
ggplot(erie, aes(date, cases))
```



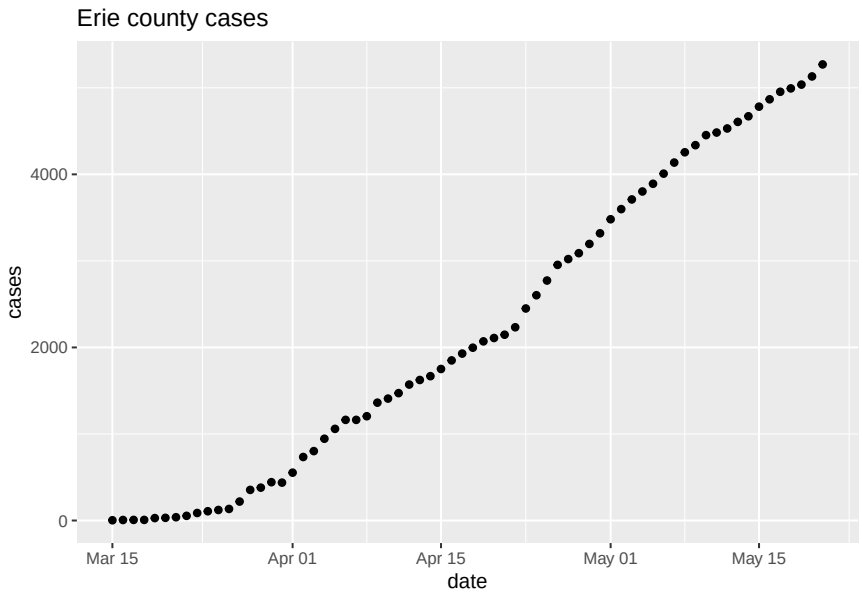
- `geom_*()`: how to plot the aesthetics
  - 'add' to other plot components

```
ggplot(erie, aes(date, cases)) +
    geom_point()
```



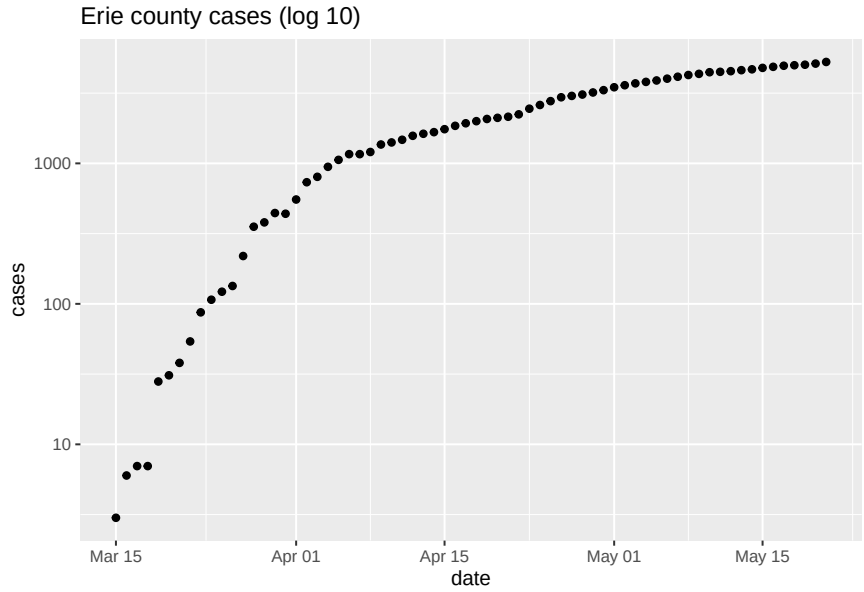
- Additional ways to decorate the data

```
ggplot(erie, aes(date, cases)) +
    geom_point() +
    ggtitle("Erie county cases")
```



- Plots can be captured as a variable, and subsequently modified

```
p <-
    ggplot(erie, aes(date, cases)) +
    geom_point() +
    ggtitle("Erie county cases")
p +
    scale_y_log10() +
    ggtitle("Erie county cases (log 10)")
```



Global data

- get_world_data()

```
get_world_data <-
    function()
{
    ## read data from Hopkins' github repository
    hopkins = "https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_1
    csv <- read_csv(hopkins)

    ## 'tidy' the data
    world <-
        csv %>%
        pivot_longer(-(1:4), names_to = "date", values_to = "cases") %>%
        mutate(date = as.Date(date, format = "%m/%d/%y")) %>%
        rename(country = "Country/Region")

    ## sum cases across regions within aa country
    world <-
        world %>%
        group_by(country, date) %>%
        summarize(cases = sum(cases))

    ## add `new_cases`, and return the result
    world %>%
        group_by(country) %>%
        mutate(new_cases = diff(c(0, cases))) %>%
```

```
        ungroup()
}
```

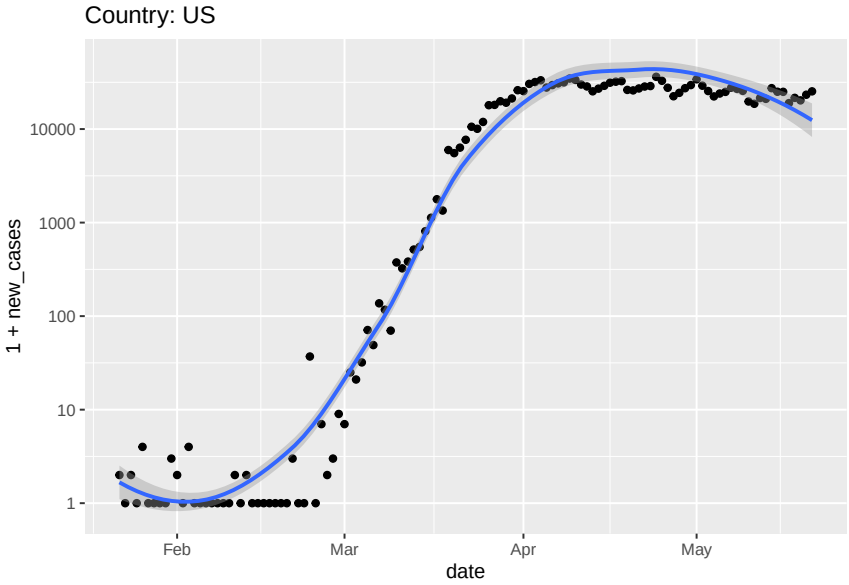
- plot_country()

```
plot_country <-
    function(tbl, view_country = "US")
{
    country_title <- paste("Country:", view_country)

    ## subset to just this country
    country_data <-
        tbl %>%
        filter(country == view_country)

    ## plot
    country_data %>%
        ggplot(aes(date, 1 + new_cases)) +
        scale_y_log10() +
        geom_point() +
        ## add method and formula to quieten message
        geom_smooth(method = "loess", formula = y ~ x) +
        ggtitle(country_title)
}
```

- *functions* to encapsulate common operations

- typically saved as *scripts* that can be easily `source()`'ed into *R*

```
world <- get_world_data()
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   `Province/State` = col_character(),
##   `Country/Region` = col_character()
## )
## See spec(...) for full column specifications.
world %>% plot_country("US")
```



3.5.3 Weekend activities (15 minutes)

- Explore global pandemic data
- Critically reflect on data resources and interpretation

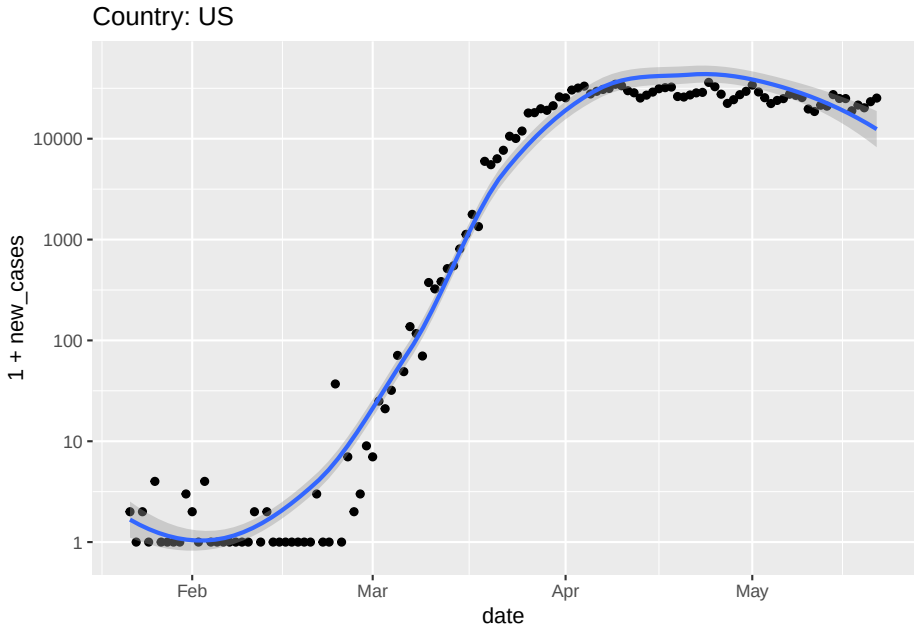
3.6 Day 20 Exploring the course of pandemic in different regions

Use the data and functions from quarantine day 18 to place the pandemic into quantitative perspective. Start by retrieving the current data

```
world <- get_world_data()
```

Start with the United States

```
world %>% plot_country("US")
```



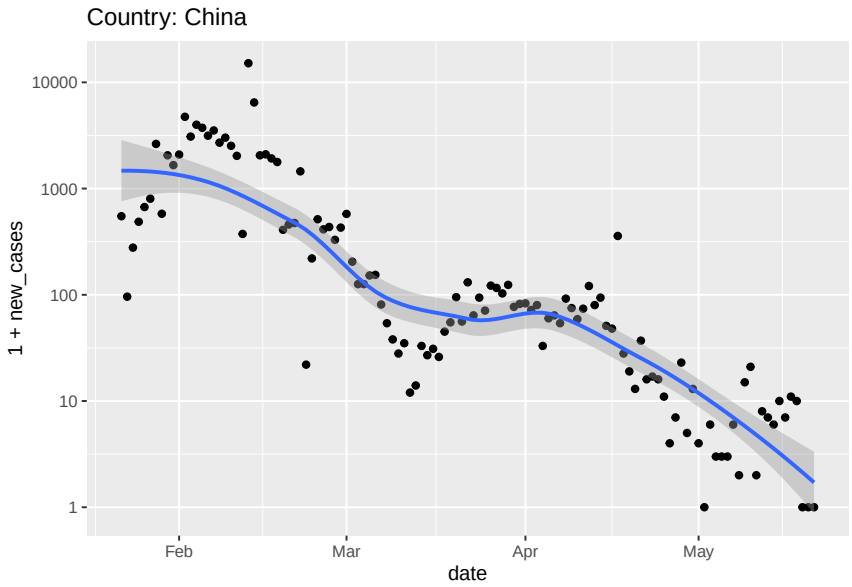
- When did 'stay at home' orders come into effect? Did they appear to be effective?

- When would the data suggest that the pandemic might be considered 'under control', and country-wide stay-at-home orders might be relaxed?

Explore other countries.

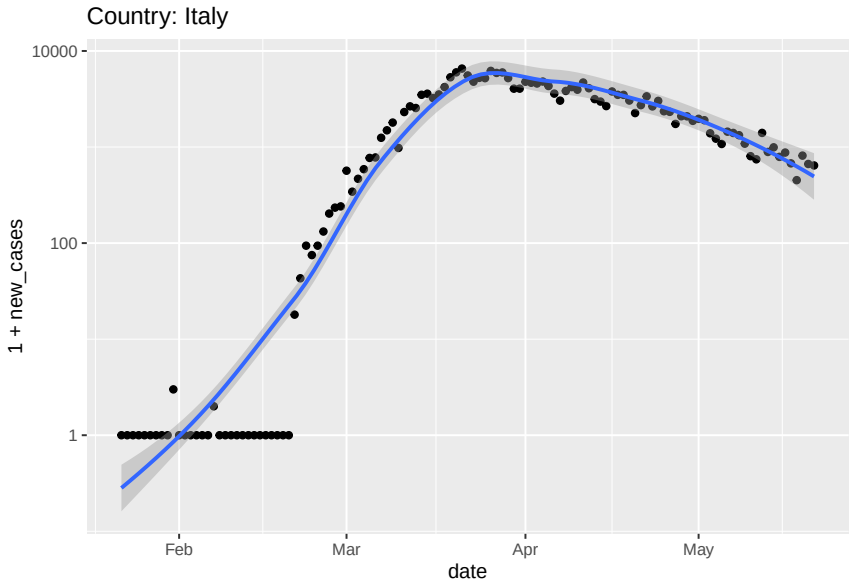
- The longest trajectory is probably displayed by China

```
world %>% plot_country("China")
```



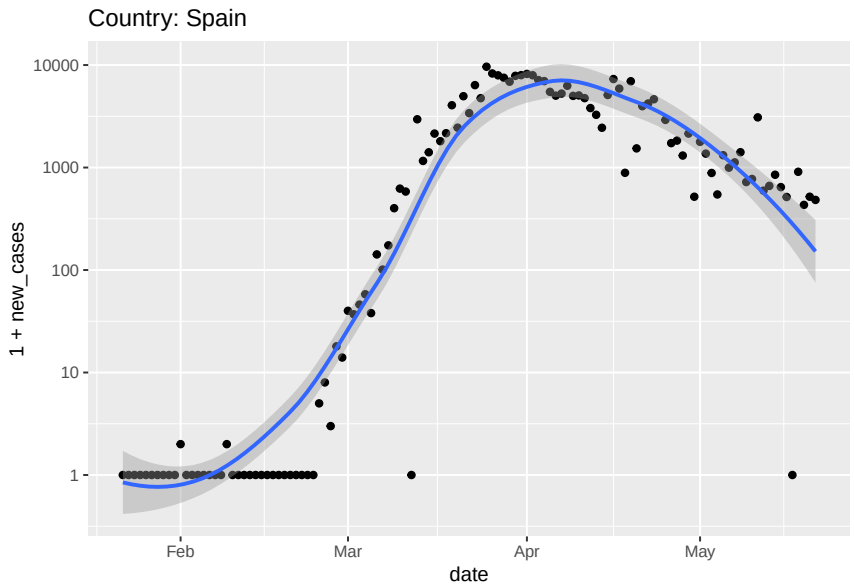
- Italy and Spain were hit very hard, and relatively early, by the pandemic

```
world %>% plot_country("Italy")
```



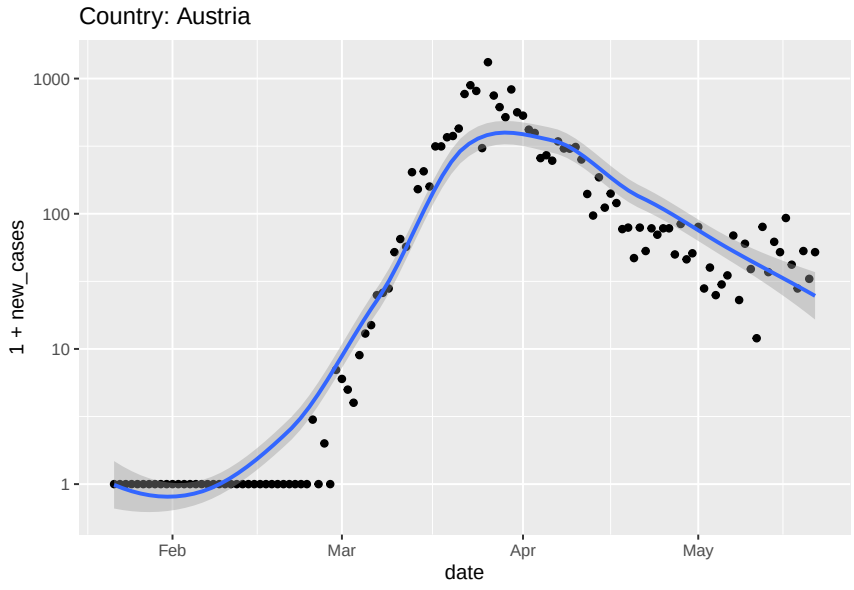

```
world %>% plot_country("Spain")
## Warning in self$trans$transform(x): NaNs produced
## Warning: Transformation introduced infinite values in continuous y-axis
## Warning in self$trans$transform(x): NaNs produced
## Warning: Transformation introduced infinite values in continuous y-axis
```

```
## Warning: Removed 1 rows containing non-finite values (stat_smooth).
## Warning: Removed 1 rows containing missing values (geom_point).
```



- Austria relaxed quarantine very early, in the middle of April; does that seem like a good idea?

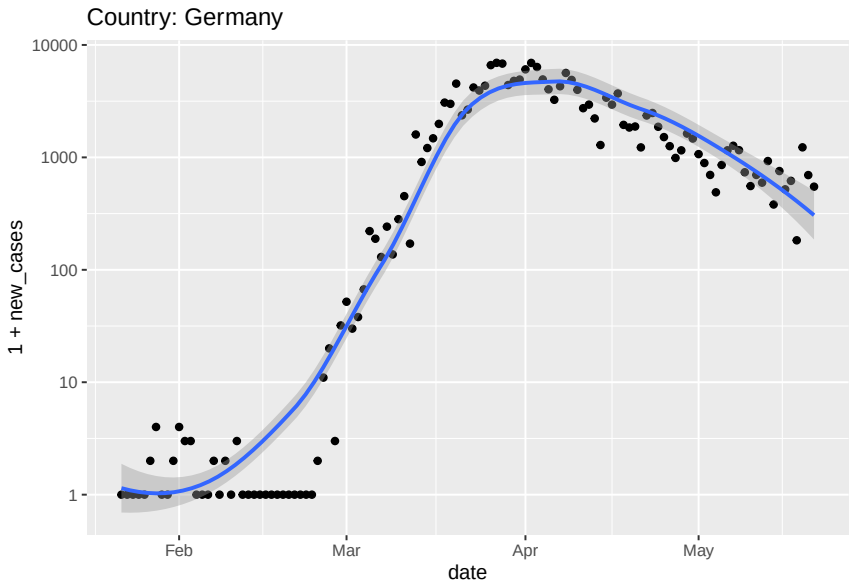
```
world %>% plot_country("Austria")
```



- Germany also had strong leadership (e.g., chancellor Angela Merkel pro-

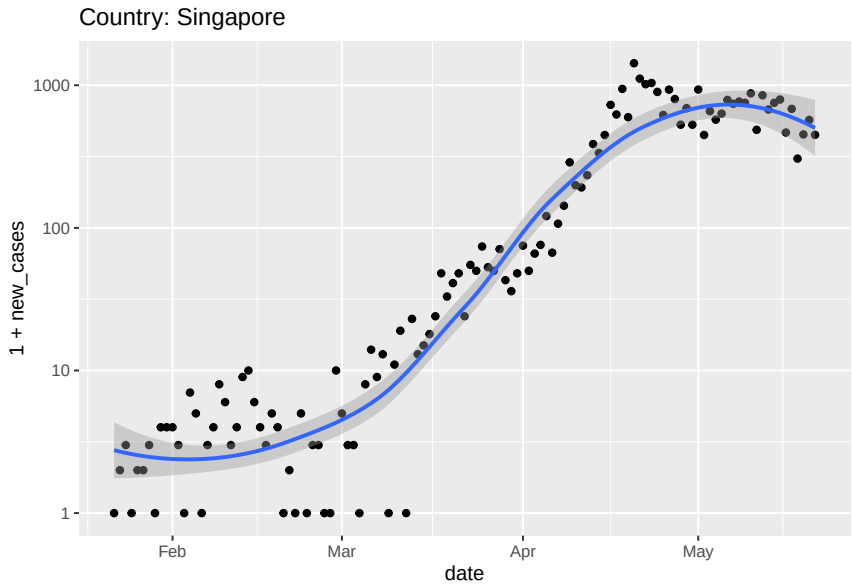
vided clear and unambiguous rules for Germans to follow, and then self-isolated when her doctor, whom she had recently visited, tested positive) and an effective screening campaign (e.g., to make effective use of limited testing resources, in some instances pools of samples were screened, and only if the pool indicated infection were the individuals in the pool screened.

```
world %>% plot_country("Germany")
```



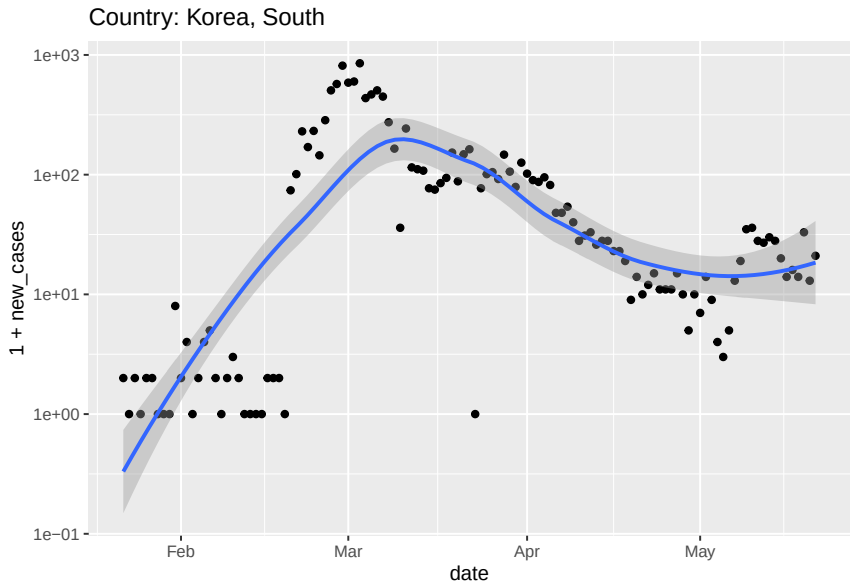
- At the start of the pandemic, Singapore had excellent surveillance (detecting individuals with symptoms) and contact tracing (identifying and placing in quarantine those individuals coming in contact with the infected individuals). New cases were initially very low, despite proximity to China, and Singapore managed the pandemic through only moderate social distancing (e.g., workers were encouraged to operate in shifts; stores and restaurants remained open). Unfortunately, Singaporeans returning from Europe (after travel restrictions were in place there) introduced new cases that appear to have overwhelmed the surveillance network. Later, the virus spread to large, densely populated migrant work housing. Singapore's initial success at containing the virus seems to have fallen apart in the face of this wider spread, and more severe restrictions on economic and social life were imposed.

```
world %>% plot_country("Singapore")
```



- South Korea had a very 'acute' spike in cases associated with a large church. The response was to deploy very extensive testing and use modern approaches to tracking (e.g., cell phone apps) coupled with transparent accounting. South Korea imposed relatively modest social and economic restrictions. It seems like this has effectively 'flattened the curve' without pausing the economy.

```
world %>% plot_country("Korea, South")
```



Where does your own exploration of the data take you?

3.7 Day 21 Critical evaluation

The work so far has focused on the mechanics of processing and visualizing the COVID-19 state and global data. Use today for reflecting on interpretation of the data. For instance

- To what extent is the exponential increase in COVID presence in early stages of the pandemic simply due to increased availability of testing?
- How does presentation of data influence interpretation? It might be interesting to visualize, e.g., US cases on linear as well as log scale, and to investigate different ways of 'smoothing' the data, e.g., the trailing 7-day average case number. For the latter, one could write a function

  ```
  seven_day_average <- function(x) {
      x <- stats::filter(x, rep(1/7, 7), sides = 1)
      as.vector(x)
  }
  ```

  and apply this to `new_cases` prior to visualization. Again this could be presented as log-transformed or on a linear scale.
- Can you collect additional data, e.g., on population numbers from the US census, to explore the relationship between COVID impact and socioeconomic factors?
- How has COVID response been influenced by social and political factors in different parts of the world? My (Martin) narratives were sketched out to some extend on Day 20. What are your narratives?
- What opportunities are there to intentionally or unintentionally misrepresent or mis-interpret the seemingly 'hard facts' of COVID infection?

Chapter 4

Machine learning

This week you will learn about machine learning for classification using *R*. Objectives are:

- To provide an overview of the underlying concepts of machine learning for classification.
- To provide an introduction to some popular classification algorithms.
- To explore these classification algorithms using various packages in *R*.
- To apply various classification algorithms to the statewide COVID-19 dataset.

4.1 Day 22 (Monday) Zoom check-in

Here is an overview of what we'll cover in today's Zoom session:

- Overview of the COVID-19 Machine Learning Dataset (10 minutes)
- Overview of Machine Learning for Classification (20 minutes)
- Introduction to the Random Forest algorithm (5 minutes)
- A Random Forest Example in *R* using COVID-19 Data (25 minutes)

4.1.1 A COVID-19 Dataset for Machine Learning

We'd like to predict the severity of COVID-19 in a given state using statewide *feature* data like population, urban density, number of hospital beds, date of stay at home order, etc. We've already seen that we can get the information about cases and deaths from the New York Times github page. However, gathering corresponding statewide feature data requires quite a bit of hunting through various public websites. Consequently, we're going to skip over the painstaking

process of marshalling the feature data and just provide you with a dataset that is already nice and prepped for machine learning.

You'll work with two `.csv` files - a *data* file that contains a veriety of statewide data, and a *metadata* file that describes the various columns of the data file. This combination of data and metadata files is a common way of sharing datasets.

To give you an idea of what was involved in assembling the data and metadata file, a summary of the data collection and processing steps is given below:

- First, a snapshot of the New York Times (NYT) COVID-19 data from April 27th was downloaded from the nytimes github repo and stored on local disk.

```
url <- "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties
file <- "us-counties_04_27_2020.csv"
destination <- file.path("workdir", file)
download.file(url, destination)
```

- The NYT COVID-19 data was processed using *R* :
  - The cases and deaths in `us-counties_04_27_2020.csv` were aggregated into statewide values.
  - The death rate was calculated and categorized according to an 8-point severity index.
  - Finally, the statewide data (augmented with death rate and severity index) was exported as a `.csv` file.

```
library(readr)
library(dplyr)
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## get data from file
covid_data_file <- file.path("./workdir", "us-counties_04_27_2020.csv")

## read the data as a tibble
us_data <- read_csv(covid_data_file)
## Parsed with column specification:
## cols(
##   date = col_date(format = ""),
##   county = col_character(),
```

```
##   state = col_character(),
##   fips = col_character(),
##   cases = col_double(),
##   deaths = col_double()
## )

## aggregate by county and state
county_state <-
    us_data %>%
    group_by(county, state) %>%
    summarize(cases = max(cases), deaths = max(deaths))

## aggregate by state
state <-
    county_state %>%
    group_by(state) %>%
    summarize(cases = sum(cases), deaths = sum(deaths))

## calculate death rate
state <-
    state %>%
    mutate(death_rate = 100.00 * deaths / cases)

# Assign the following severity index using cut:
##  PSI Death.Rate
##  1   < 0.1%
##  2   0.1% - 0.5%
##  3   0.5% - 1.0%
##  4   1.0% - 2.0%
##  5   2.0% - 4.0%
##  6   4.0% - 6.0%
##  7   6.0% - 8.0%
##  8   >8.0%
state <-
    state %>%
    mutate(
        severity_index =cut(
            death_rate,
            breaks = c(0.0, 0.1, 0.5, 1.0, 2.0, 4.0, 6.0, 8.0, 100.0),
            labels = 1:8
        )
    )

## write out as csv
my_out_file <- file.path("./workdir", "covid_data.csv")
```

```
write_csv(state, my_out_file)
```

- The resulting statewide COVID-19 *label* data (i.e. what we would like to predict) was augmented with 32 statewide *features*, including population, percent urban, number of hospital beds, etc. Feature data was collected from a veriety of sources, including the Center for Disease Control, the American Heart Association, the U.S. Census Bureau, etc. In some cases the feature data was available for direct download (e.g. as a `.csv` file) and in other cases the feature data was manually harvested (e.g. cut-and-paste from websites).
- The augmented (i.e. features + labels) `.csv` file was split into two `.csv` files that you will need to download:
  - statewide_covid_19_data_04_27_2020.csv: This file contains the final COVID-19 machine learning data set, but features and labels are coded so that feature columns are named `X01`, `X02`, `X03`, etc. and label colunms are named `Y01`, `Y02`, `Y03`, etc.
  - statewide_covid_19_metadata_04_27_2020.csv: This file maps the column names in the data file to more meaningful names and desciptions (including units) of the associated variables. For example `X01` is `Pct_Sun` and has a description of `Percent sunny days`. This is known as *metadata* - data that describes other data.

Click on the links above to download the data and metadata files that you'll need for the machine learning examples presented throughout the week.

4.1.2 A Machine Learning Primer

Machine Learning (ML) may be defined as using computers to make inferences about data.

- A mapping of inputs to outputs, Y = f(X , β)

ML for Classification refers to algorithms that map inputs to a discrete set of outputs (i.e. classes or categories)

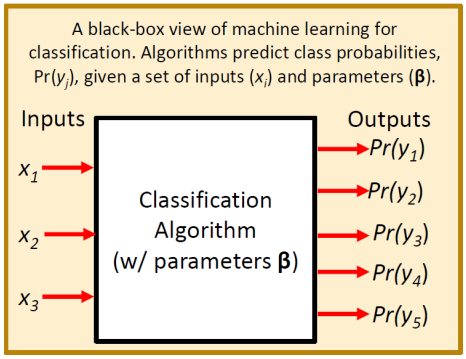
- For example, predicting health risk (mild, moderate, severe) based on patient data (height [m], weight [kg], age [years], smoker [yes/no], etc.)
- Or predicting pandemic severity index (PSI) of COVID-19 in a state based on statewide population data.

```
##  PSI Death.Rate    Example
##  1   < 0.1%        seasonal flu
##  2   0.1% - 0.5%   Asian flu
##  3   0.5% - 1.0%   n/a
##  4   1.0% - 2.0%   n/a
##  5   > 2.0%        Spanish flu
```

- Predictions are typically expressed as a vector of probabilities.
  - e.g. Pr(cancerous) vs. Pr(benign)
  - e.g. Pr(PSI=1), Pr(PSI=2), ..., Pr(PSI=5)

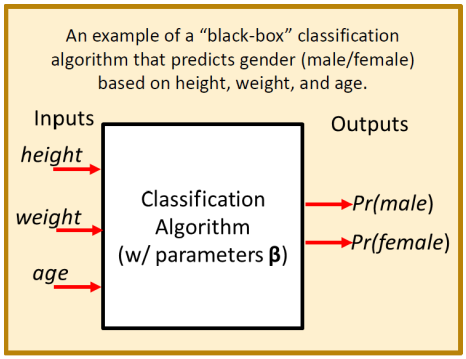
A "Black Box" View of Machine Learning

The diagram below illustrates the machine learning concept in terms of a "black box" model that converts inputs into predictions.



A "Black Box" Example

The diagram below illustrates a more concrete machine learning model that converts height, weight, and age into a prediction of whether or not the individual is male or female.



Some Important Machine Learning Terms

The above example highlights some important machine learning terminology:

- *Features* (X): Features are the inputs to the model. They are also known as descriptive attributes or explanatory variables. In terms of R, a single set of features corresponds to a tuple or row of data and a complete set of feature data maps nicely to a data frame or tibble.

- *Parameters* (β): Parameters are internal variables of the selected machine learning algorithm. They are also known as coefficients or training weights. These internal parameters need to be adjusted to minimize the deviation between predictions and observations. The machine learning algorithms and pacakages that we'll be using in R take care of this minimization process for us.

- *Labels* (Y): Labels are the outputs of the algorithm, corresponding to the categories you are attempting to predict.

- *Training Data*: Training data is a data set containing paired observations of inputs (i.e features) and outputs (i.e. labels). Training data is also known as measurement data, observation data or calibration data.

- *Training*: Training is the process of adjusting internal algorithm paramters (β) to obtain the best possible match between training data and corresponding model predictions. Training is also known as model calibration or parameter estimation.

An example set of training data that could be used in the gender prediction example is given below. The frist three columns contain feature data and the last column contains label data. We could use this data to train a model to predict a person's gender based on their height, weight and age.

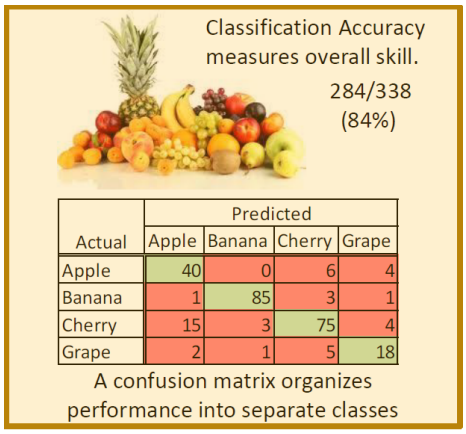
```
##  height_m weight_kg age_y gender
##  1.69      62       30    male
##  1.74      76       27    female
##  1.92      82       25    male
##  1.80     100       41    male
##  1.59      47       24    female
##  1.85      75       26    male
##  1.75      63       33    female
##  1.96      83       33    male
##  1.85      39       32    male
##  1.78      58       28    female
##  1.74      70       30    female
##  1.81      57       26    female
##  1.73      78       32    male
```

- *Test Data*: Like training data, test data is a data set containing paired observations of inputs (i.e features) and outputs (i.e. labels). However, test data is deliberately *not included* in the training process. Performance measures computed using test data help to quantify the expected performance of the model if/when it is applied to unlabeled data.

Upon completion of training, it is important to evaluate the quality of the model and its skill or ability at making correct predictions. Some terms related to this evaluation process are defined below:

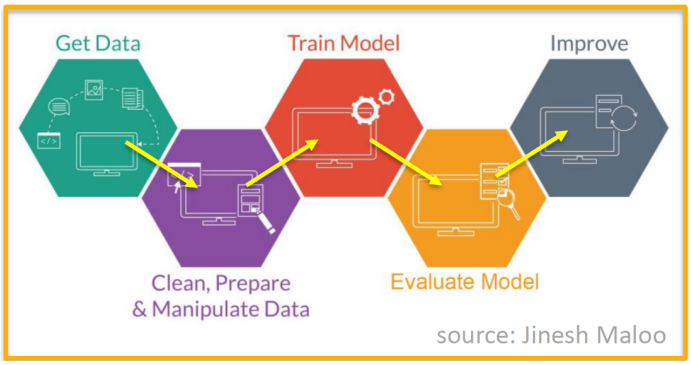
- *Classification Accuracy*: Classification accuracy is the ratio of correct predictions to the total predictions. Classification accuracy can be computed using the training data set or using the test data set.
- *Confusion Matrix*: The confusion matrix is a more detailed summary (relative to classification accuracy) of the performance of a classification algorithm. The diagonals of the confusion matrix count how often the algorithm yields the correct classification for each class. The off-diagonal entries count how often the algorithm confuses one class with another.

The figure below illustrates the classification accuracy and confusion matrix for an example that attempts to classify images of fruits.



The Machine Learning Process

Now that we've defined some of the important machine learning terminology let's take a 30,000 foot view at the overall machine learning process. This is a general description of the process that you can follow each time you build and use a machine learning model. The process is illustrated in the figure below (with credit to Jinesh Maloo):



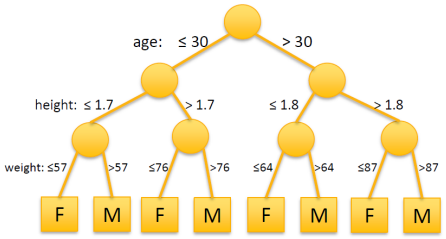
- Step 1: Prepare labeled data for training and validation.
- Step 2: Select a machine learning algorithm (i.e. model).
- Step 3: Train the model.
- Step 4: Evaluate model performance.
- If model is useful:
  - Step 5: Apply to unlabled data.
- Else (needs improvement):
  - Collect more data (go back to Step 1).
  - Revise model (go back to Step 2)

4.1.3 The Random Forest Algorithm

The random forest algorithm is a popular choice for machine learning.

- Over 20 *R* packages have an implementation of some form of the algorithm.
- We'll be using the `randomForest` pacakge.
- The algorithm is like `bagging` (boostrap aggregating) regression trees, but the rgression trees are de-correlated.

The figure below illustrates one possible `tree` in a `random forest` for a gender prediction model. In computer science terminology, each split in the figure is a `branch` of a graph `tree`. In simple terms, the split points are randomly generated and the resulting `trees` combine to form a `random forest`.

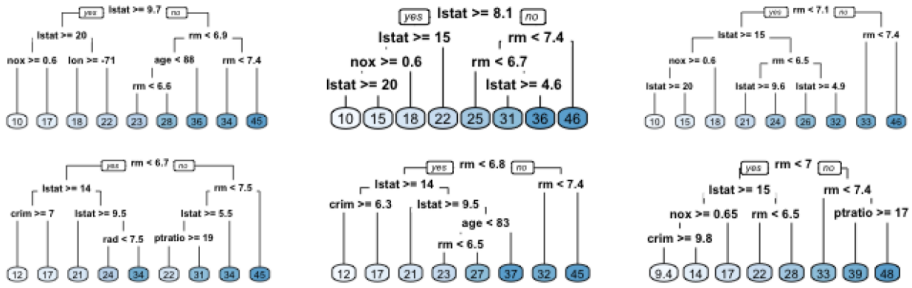


One possible classification tree for predicting gender. The bagging process develops multiple trees and makes a prediction by combining the results.

The figure below illustrates a set of 6 `trees` that make up a `random forest` for predicting housing prices. The image is courtesy Bradley Boehmke at the University of Cincinnati. Examine the figure closely and notice that:

- The split variables can differ across trees (not all variables are included in all trees).
- The split variables can differ within trees (not all paths consider the same set of variables).

- The order of splits can differ.
- The split values can differ.



The job of the random forest algorithm is to determine the optimal set of trees for your data set, including the splitting configuration of each tree (i.e. order, values, etc.).

Now you have a basic understanding of machine learning and the random forest algorithm. You're probably excited to get going with applying the algorithm! But first, we need to have a data set to work with. In the next section you'll learn about a data set that can be used for predicting the severity of COVID-19 in a state.

4.1.4 A Random Forest Example Using COVID-19 Data

Let's apply the random forest algorithm to the COVID-19 dataset. We'll build out the required *R* code in sections. To get started, open a new *R* script in *RStudio* and name it `covid_19_rf.R`. Enter the code below, but omit lines that begin with a double-hash (`##`) because these are the expected output:

```
#
# Part 1 - load the data
#
library(randomForest)
## randomForest 4.6-14
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:dplyr':
##
##     combine
library(readr)
library(dplyr)

data_file <- file.path("assets", "statewide_covid_19_data_04_27_2020.csv")
df <- read_csv(data_file)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   State = col_character(),
##   X31 = col_character(),
##   X32 = col_character()
## )
## See spec(...) for full column specifications.
# coerce severity to a factor (so RF algorithm uses classification)
df <-
    df %>%
    mutate(Y04 = as.factor(df$Y04))


metadata_file <- file.path("assets", "statewide_covid_19_metadata_04_27_2020.csv")
mdf <- read_csv(metadata_file)
## Parsed with column specification:
## cols(
##   ID = col_double(),
##   Code = col_character(),
##   Variable = col_character(),
##   Description = col_character()
## )
mdf
## # A tibble: 36 x 4
##       ID Code  Variable             Description
##    <dbl> <chr> <chr>                <chr>
##  1     1 X01   Pct_Sun              Percent sunny days
##  2     2 X02   Total_Hours_Sun      Total hours of sun
##  3     3 X03   Num_Clear_Days       Number of clear days
##  4     4 X04   Avg_RH               Average relative himidity
##  5     5 X05   Avg_Dew_Point        Average dew point
##  6     6 X06   Total_Population     Total population
##  7     7 X07   Senior_Pop_Thousands Population 65+ years in thousands
##  8     8 X08   Senior_Pop_Pct       Percentage of population 65+ year
##  9     9 X09   per_capita_income    per capita income
## 10    10 X10   Unemployment_Rate    Percent unemployment
## # ... with 26 more rows
```

Try to run the code. You may get an error about missing the `randomForest` package. You can install it from the RStudio console (see below) or using the installer in the RStudio `packages` pane.

```
install.packages("randomForest")
```

Now we've loaded the data and metadata file. Let's pick a subset of 5 of the

features and use them to try and predict the pandemic severity index (i.e. `Y04`). Add the `Part 2` code below to your RScript but omit lines that begin with a double-hash (`##`) :

```
#
# Part 2 - select features and label
#

# describe all possible features and labels
print(mdf, n = nrow(mdf))
## # A tibble: 36 x 4
##       ID Code  Variable           Description
##    <dbl> <chr> <chr>              <chr>
##  1     1 X01   Pct_Sun            Percent sunny days
##  2     2 X02   Total_Hours_Sun    Total hours of sun
##  3     3 X03   Num_Clear_Days     Number of clear days
##  4     4 X04   Avg_RH             Average relative himidity
##  5     5 X05   Avg_Dew_Point      Average dew point
##  6     6 X06   Total_Population   Total population
##  7     7 X07   Senior_Pop_Thousa~ Population 65+ years in thousands
##  8     8 X08   Senior_Pop_Pct     Percentage of population 65+ year
##  9     9 X09   per_capita_income  per capita income
## 10    10 X10   Unemployment_Rate  Percent unemployment
## 11    11 X11   Uninsured_Rate_Ch~ Percent uninsured children
## 12    12 X12   Uninsured_Rate_Ad~ Percent uninsured adults
## 13    13 X13   Heart_Disease_Rate Deaths per 100000  due to heart disease
## 14    14 X14   Heart_Disease_Dea~ Total deaths due to heart disease
## 15    15 X15   Tobacco_Use_Rate   Percentage of tobacco users
## 16    16 X16   Obesity_Prevalenc~ Percentage of population this is considered o~
## 17    17 X17   Num_Hospitals      Number of hospitals
## 18    18 X18   Num_Hosp_Staffed_~ Number of hospital beds
## 19    19 X19   Total_Hosp_Discha~ Total number of hospital discharges
## 20    20 X20   Hosp_Patient_Days  Total number of patient days in hospital
## 21    21 X21   Hosp_Gross_Patien~ Total hospital gross patient revenue
## 22    22 X22   Number_of_Farms    Total number of farms
## 23    23 X23   Urban_Population_~ Percentage of population living in urban areas
## 24    24 X24   Urban_Population   Total urban population
## 25    25 X25   Urban_Land_Area_S~ Amount of urban land area in square miles
## 26    26 X26   Urban_Density_Sq_~ Urban density in persons per square mile
## 27    27 X27   Urban_Land_Pct     Percentage of land use classified as urban
## 28    28 X28   Pct_Republican     Percentage of population registered republican
## 29    29 X29   Pct_Independent    Percentage of population not affiliated with ~
## 30    30 X30   Pct_Democrat       Percentage of population registered democrat
## 31    31 X31   Stay_at_Home_Star~ Date stay at home order issued
## 32    32 X32   Stay_at_Home_End_~ Date stay at home order scheduled to be lifted
## 33     1 Y01   cases              total number of covid-19 cases
```

```
## 34     2 Y02   deaths              total number of covid-19 deaths
## 35     3 Y03   death_rate          death rate as a percentage
## 36     4 Y04   severity_index      severity index based on death_rate: ## 1 = < ~

# select some features and the severity index label
my_x <- c("X01","X10","X12","X13","X23")
my_y <- "Y04"
my_xy <- c(my_x, my_y)

# get descriptions of the selected features and label
mdf %>% filter(Code %in% my_xy)
## # A tibble: 6 x 4
##      ID Code  Variable          Description
##   <dbl> <chr> <chr>             <chr>
## 1     1 X01   Pct_Sun           Percent sunny days
## 2    10 X10   Unemployment_Ra~ Percent unemployment
## 3    12 X12   Uninsured_Rate_~ Percent uninsured adults
## 4    13 X13   Heart_Disease_R~ Deaths per 100000  due to heart disease
## 5    23 X23   Urban_Populatio~ Percentage of population living in urban areas
## 6     4 Y04   severity_index   severity index based on death_rate: ## 1 = < 0.1~

# subset the dataframe
rf_df <- df %>% select(all_of(my_xy))
```

Now we'll add code to create and train a basic Random Forest model. Add the `Part 3` code below to your RScript but omit lines that begin with a double-hash (`##`):

```
#
# Part 3 - create and train the model
#
# split into train (75%) and test (25%) datasets
#
# note that `row_nubmer()` generates a vector of row numbers, and
#     > c(1, 2, 3, 4, 5, 6) %% 4 is
#     [1] 1 2 3 0 1 2

train <- rf_df %>% filter((row_number() %% 4) %in% 1:3)
test <- rf_df %>% filter(row_number() %% 4 == 0)

# create and train the RF model
model <- randomForest(Y04 ~ X01 + X10 + X12 + X13 + X23,
                      data = train)

# show results, includes confusion matrix for training data
model
```

```
##
## Call:
##  randomForest(formula = Y04 ~ X01 + X10 + X12 + X13 + X23, data = train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 65.79%
## Confusion matrix:
##   3 4 5 6 7 8 class.error
## 3 0 0 1 0 0 0   1.0000000
## 4 0 0 1 1 0 0   1.0000000
## 5 0 0 7 8 0 0   0.5333333
## 6 0 1 8 6 0 0   0.6000000
## 7 0 0 1 3 0 0   1.0000000
## 8 0 0 0 1 0 0   1.0000000

# measure of parameter importance
importance(model)
##     MeanDecreaseGini
## X01         3.765562
## X10         5.543642
## X12         5.701777
## X13         4.756719
## X23         5.086897
```

At this point the model is trained and the next step is to evaluate its usefulness at making predictions. Let's see how the model does at predicting the labels of the test dataset. Remember that the test data was *not* used during the training exericse. As such, the confusion matrix and classification accuracy associated with the test dataset provides a useful check of the skill of the model. Add the `Part 4` code below to your RScript but omit lines that begin with a double-hash (`##`) :

```
#
# Part 4 - evaluate the model using test data
#

# extract test predictions
preds <- predict(model, test)

# confusion matrix
actual <- factor(test$Y04)
predicted <- factor(preds)
common_levels <- sort(unique(c(levels(actual), levels(predicted))))
actual <- factor(actual, levels = common_levels)
```

```
predicted <- factor(predicted, levels = common_levels)
confusion <- table(actual,predicted)
print(confusion)
##       predicted
## actual 4 5 6 7
##      4 0 1 1 0
##      5 0 3 3 0
##      6 0 1 2 0
##      7 0 0 1 0

#classification accuracy
accuracy <- sum(diag(confusion)/nrow(test))
cat("Classification Accuracy = ", accuracy, "\n")
## Classification Accuracy =  0.4166667
```

Save your script and run it. Take a look at the results for the test dataset - the classifcation accuracy is well under 50%. Furthermore, there are systematic failures in the confusion matrix. It's not a very good model. The most likely culprit is that the set of features is inadequate for making the desired prediction. We should re-run the model using different or additional features.

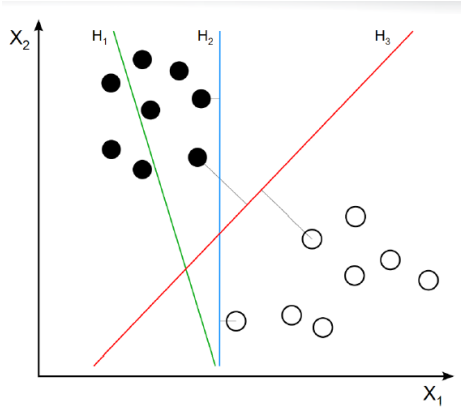
4.2 Day 23 - Support Vector Machines

For today's independent work you will learn about the Support Support Vector Machine (SVM) algorithm and apply it to the COVID-19 data that you worked with on Monday.

The SVM algorithm seeks to determine an optimal hyperplane that separates labeled observations.

- Hyperplanes can be linear or non-linear.
- Support vectors are data points lying closest to the optimal hyperplane.
- The `e1071` package in R provides an implementation of SVM.

The figure below illusrates a linear SVM. Line H_3 provides the optimal separation between the white and black data points. Points with perpendiculars to H_3 are the support vectors for the dataset. The SVM algorithm classifies unlabeled data points by examining their location with respect to the optimal hyperplane. Data points *above* line H_3 would be classified as "black" and datapoints *below* the line would be calssified as white.



4.2.1 An SVM Example using COVID-19

We've already laid a large part of the groundwork for machine learning with the random forest example. With a few modifications, the `covid_19_rf.R` script can be adapted to use an SVM algorithm instead of Random Forest.

- Open your `covid_19_rf.R` script in *RStudio*
- Click `File --> Save As ...` and name the file `covid_19_svm.R`
- In `Part 1` of `covid_19_svm.R`, replace `library(randomForest)` with `library(e1071)`. This will load the `svm` algorithm instead of the `randomForest` algorithm.
- In `Part 3` of the code, replace `randomForest()` with `svm()` and add the following argument to the `svm()` function: `probability = TRUE`. The `svm()` code should look something like this:

```
model <- svm(Y04 ~ X01 + X10 + X12 + X13 + X23,
             data = train,
             probability = TRUE)
```

- The `svm()` package does not provide useful implementations of `print(model)` or `importance(model)`. Comment out, or delete, those lines of `Part 3` and replace with `summary(model)`. When you are finished, the final section of the `Part 3` code should look something like this:

```
# show results, includes confusion matrix for training data
# print(model)
#
# measure of parameter importance
# importance(model)
#
```

```
# summarize the trained SVM
summary(model)
```

That's it! The rest of the code (i.e. `Part 4`) can be re-used. Save your `covid_19_svm.R` script and run it. How does the SVM algorithm perform in comparison with the Random Forest algorithm?

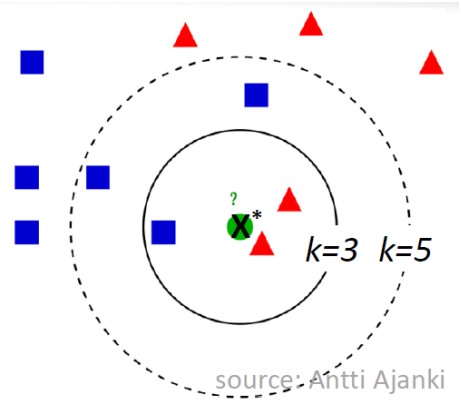
4.3 Day 24 - the k-Nearest Neighbors Algorithm

For today's independent work you will learn about the k-Nearest neighbors (KNN) algorithm and apply it to the COVID-19 data.

For a given unlabeled data point (X^*) the KNN algorithm identifies the nearest k labeled data points.

- Euclidean distance is typical
- *Normalization of data is necessary to prevent biased distances.*
- The label of X^* is predicted to be the most frequently occurring label among the k nearest neighbors.
- The `class` package in R provides an implementation of KNN.

The figure below illustrates the KNN approach and is courtesy of Antti Ajanki. For $k = 3$, the neighborhood contains 2 triangles and 1 square so we'd predict X^* is a triangle. For $k = 5$, the neighborhood contains 3 squares and 2 triangles so we'd predict X^* is a square.



With a few modifications, the `covid_19_svm.R` script can be adapted to use an KNN algorithm instead of SVM.

- Open your `covid_19_svm.R` script in *RStudio*
- Click `File --> Save As ...` and name the file `covid_19_knn.R`

- In `Part 1` of `covid_19_knn.R`, replace `library(e1071)` with `library(class)`. This will load the `knn` algorithm instead of the `svm` algorithm.

- The KNN algorithm requires the feature data to be normalized. Add the following line in `Part 1` of `covid_19_knn.R`. Add the line after the `df <- df %>% mutate(Y04 <- as.factor(Y04))` line, as shown below. If necessary, use `install.packages('BBmisc')` to install the `BBmisc` package and its `normalize()` function.

```
df <- df %>% mutate(Y04 <- as.factor(Y04))
df <- BBmisc::normalize(df, method = "range") # add this line
```

- In `Part 3` of the code, replace `svm()` with `knn()` and adjust the call to `knn()` so that it looks like this:

```
model <- knn(select(train, all_of(my_x)),
             select(test, all_of(my_x)),
             train$Y04,
             k = 7,
             prob = TRUE)
```

- The `knn()` package does not provide useful implementations `print(model)`, `importance(model)`, or `summary(model)`. Comment out, or delete, those lines of `Part 3`. When you are finished, the final section of the `Part 3` code should look something like this:

```
# show results, includes confusion matrix for training data
# print(model)
#
# measure of parameter importance
# importance(model)
#
# summarize the trained SVM
# summary(model)
```

- In `Part 4` of the script, replace:

```
preds <- predict(model, test)
```

with:

```
preds <- as.data.frame(model)[,1]
```

That's it! The rest of the code can be re-used. Save your `covid_19_knn.R` script and run it. How does the KNN algorithm perform in comparison with the Random Forest and SVM algorithms?

4.4 Day 25 - Exploring the KNN Algorithm

For today's independent work you will learn explore the algorithm settings of the KNN algorithm.

Open your `covid_19_knn.R` script and locate the line that creates the KNN model (`model = knn(...)`). Adjust the number of neighbors so that `k = 5` and re-run the script. Record the classification accuracy. Try again, but use `k = 3`.

Re-run each experiment several times (e.g. collect 10 trials of `k=5` and 10 trials of `k=3`, and so on). For a given value of `k`, do you get the same result each time?

Continue with these numerical experiments until you've filled out the table below:

```
## k_value  average_classification_accuracy
##    1                ???%
##    3                ???%
##    5                ???%
##    7                ???%
##    9                ???%
##   11                ???%
##   13                ???%
##   15                ???%
##   17                ???%
```

What do the results suggest about the most appropriate setting for `k`? Can you think of a better way to perform the numerical experiments and collect the results?

4.5 Day 26 (Friday) Zoom check-in

Today we'll check how you're doing with using machine learning in *R*. Then we'll get you prepared for weekend activities, where you'll continue to explore modeling using the COVID-19 dataset.

4.5.1 Review and trouble shoot (25 minutes)

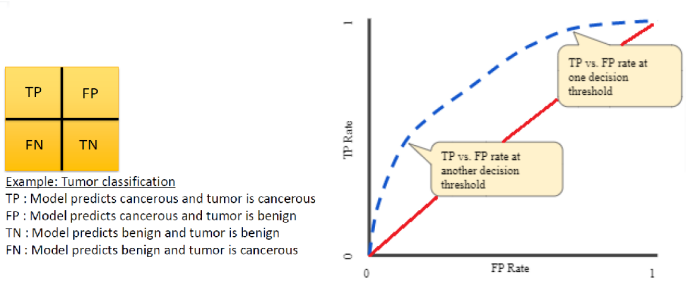
- Has everyone had a chance to try out at least one of the machine learning algorithms?
- Has anyone tried additional or alternative combinations of features?
- What is the best classification accuracy that you have been able to obtain?
- What parameters appear to be the most important?

4.5.2 This weekend (25 minutes)

ROC/AUC - Another Measure of Machine Learning Performance

We've already seen how classification accuracy and the confusion matrix give an indication of the performance of a trained machine learning algorithm. It's also good practice to examine the "ROC curve" and related "AUC" metrics.

- *ROC curve*: As illustrated in the figure below, the ROC (Receiver Operating Characteristic) curve plots the true positive (TP) vs. false positive (FP) rate at various probability thresholds. In the figure, the dashed blue line represents a hypothetical ROC curve for some machine learning model and the solid red line is the curve for a "non-informative" model (i.e. a model that makes a uniform random guess). As such, we'd like the blue curve to be as far above the red curve as possible.



- *AUC*: AUC stands for "area under curve" and is the area under the ROC curve. In the previous figure, the AUC would be the area under the dashed blue curve. Values of AUC quantify the degree to which an ROC curve lies above (or below) the "non-informative" curve. Some interesting AUC values:
  - AUC = 0.0: the model is always wrong (with respect to TP vs. FP)
  - AUC = 0.5: the model is no better than guessing (i.e. the model matches the red "non-informative" curve in the figure)
  - AUC = 1.0: the model is always right (with respect to TP vs. FP)

For a problem with multiple classes (as opposed to a binary True/False, Male/Female, or Yes/No problem) we can compute the ROC curve curve and AUC measures using a "one vs. all" approach:

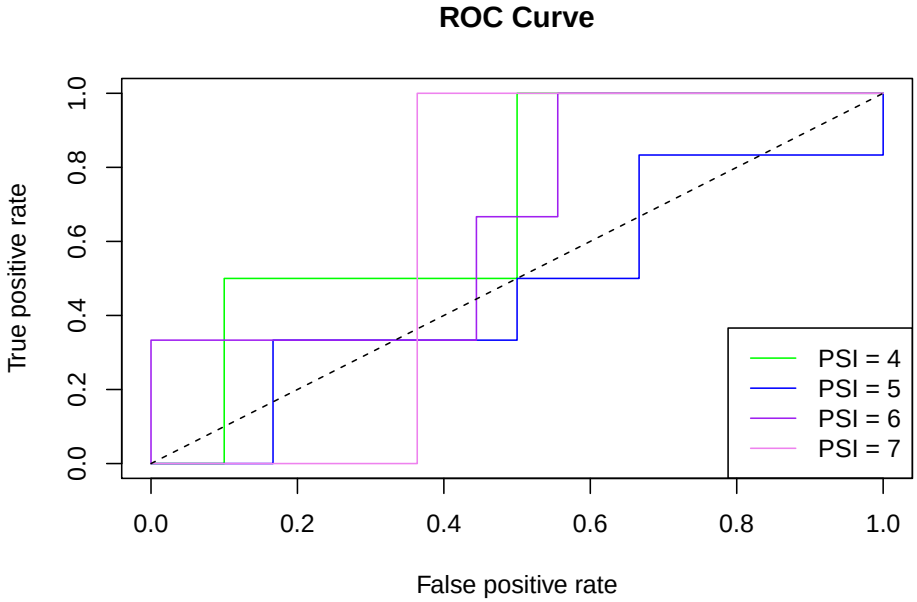
- First, extract predicted probabilities from the RF model (the scores).
- Next, extract actual classification for each category.
- Finally, leverage three commands of the `ROCR` module:
  - `prediction()`: retrieve scores

  - `performance()`: generates TPR, FPR, and AUC measures through two separate calls
  - `print()`: generates a TPR vs. FPR plot

The ROC/AUC calculation is fairly involved so we'll create a helper function for it and then incorporate the helper function into our machine learning scripts.

```
#
# Part 5 - ROC/AUC
#

# roc_one_vs_all()
#   A helper function to compute one vs. all ROC/AUC for a given level (i)
roc_one_vs_all <- function(i) {
    if(is.na(sum(probs))){
        return(NA)
    }

    actual <- as.numeric(y_test[[1]] == i)
    score <- probs[,i]

    pred <- ROCR::prediction(score, actual)
    perf <- ROCR::performance(pred, "tpr", "fpr")

    ROCR::plot(perf,
         main="ROC Curve",
         col=cols[as.numeric(i)],
         add = i != lvls[[1]])

    # calculate the AUC and print
    auc_i <- ROCR::performance(pred, measure="auc")
    as.numeric(auc_i@y.values)
}

# prepare data for computing ROC/AUC
x_test <- select(test, all_of(my_x))
y_test <- select(test, all_of(my_y))
# how we obtain probs depends on the algorithm
if (inherits(model, "randomForest"))
{
    probs <- predict(model, x_test, type='prob')
} else if (inherits(model, "svm"))
{
    probs <- predict(model, x_test, probability = TRUE)
    probs <- attributes(probs)
    probs <- as.data.frame(probs$probabilities)
```

```
    probs <- probs[,order(colnames(probs))]
} else # the KNN alg requires approximation of probabilities
{
  # highest predicted probabilities
  pnrst <- attributes(model)$prob
  # corresponding one-based categories
  y_one <- as.numeric(min(levels(df$Y04))) - 1
  vpreds <- as.integer(as.character(preds)) - y_one
  # map probabilities into a matrix of zeroes
  probs <- matrix(0.00,
                  nrow = length(pnrst),
                  ncol = length(levels(df$Y04)))
  probs[cbind(seq_along(vpreds), vpreds)] <- pnrst
  # coerce to data frame
  probs <- data.frame(probs)
  colnames(probs) <- levels(df$Y04)
}
lvls <- unique(as.character(y_test[[1]]))
cols <- c("red","orange","yellow","green",
          "blue","purple","violet","black")

# vapply helper function across levels
auc <- vapply(lvls, roc_one_vs_all, numeric(1))

# tack on legend and non-informative line
legend("bottomright",
       legend=paste("PSI =",lvls),
       col=cols[as.numeric(lvls)],
       lty=1,
       cex=1.0)
lines(x=c(0,1),
      y=c(0,1),
      lty=2)
```




```
# display AUC metrics
print(tibble(lvls, auc))
## # A tibble: 4 x 2
##   lvls    auc
##   <chr> <dbl>
## 1 4     0.7
## 2 5     0.472
## 3 6     0.667
## 4 7     0.636
```

Assuming you've been following along with the daily activities, you can add this `Part 5` code to any of your algorithm scripts (e.g. `covid_19_rf.R`, `covid_19_knn.R`, `covid_19_svm.R`, etc.). The code uses the `inherits()` function to adapt the output of each individual algorithm into a form that is suitable for the ROC/AUC calculation.

4.6 Day 27

Today you'll explore different combinations of features in your COVID-19 model.

- Select another 5 features and adjust `Part 2` and `Part 3` of your `covid_19_rf.R` script. In `Part 2` you'll need to adjust the `my_x` variable and in `Part 3` you'll need to adjust the model formula (e.g. `Y04 ~ X01 + ....`). Re-run the script and record the error rate, confusion matrix, and measures of parameter importance.

- Repeat the above step after adding an additional 5 features.
- How does the performance of the model change as more features are included?
- Does any particular parameter stand out in terms of importance?
- What does the most important parameter correspond to in termo of the metadata?

4.7 Day 28

Today you'll explore making some tweaks to the random forest model to see if you can improve its performance on the COVID-19 dataset. The algorithm parameters that you'll be adjusting are described below:

- `ntree` : The number of trees to grow. Default is 500.
- `mtry` : The number of variables randomly sampled as candidates at each split. Default is `sqrt(p)` where `p` is the number of features included in the model.

Let's perform some numerical experiments to explore how these algorithm parameters effect model performance:

- Setup a random forest model that has at least 16 features (see instructions from yesterday's activity).
- In `Part 3` of your `covid_19_rf.R` script, add the following arguments to the `randomForest()` function:
  - `ntree = 1000`
  - `mtry = 8`
- Re-run the script and record the error rate, confusion matrix, and measures of parameter importance.
- Repeat the above process using:
  - `ntree = 2000`
  - `mtry = 2`
- Does adjusting the parameters effect the model performance? If so, what observations can you make?
- Can you think of a "better" way to evaluate the influence of algorithm parameters?

Congratulations - you made it through a week of machine learning boot camp! You can download completed scripts (i.e. `Part 1` through `Part 5`) for each algorithm using the links below:

- Complete Random Forest Example
- Complete Support Vector Machine Example
- Complete k-Nearest Neighbors Example

Chapter 5

Bioinformatics with Bioconductor

5.1 Day 29 (Monday) Zoom check-in

5.1.1 Review and trouble shoot

5.1.2 Bioconductor, packages, and biological data

Bioconductor

- Repository of more than 1900 packages for the 'analysis and comprehension of high throughput genomic data'
- Bulk and single cell RNASeq; ChIP and other gene regulation, called variants, flow cytometery, proteomics, ...
- Package discovery using biocViews and workflows
- Landing pages & vignettes

Packages

- One-time installation of BiocManager

```
if (!requireNamespace("BiocManager"))
    install.packages("BiocManager", repos = "https://cran.r-project.org")
```

- Install *Bioconductor* or *CRAN* packages

```
## Biostrings and genbankr are Bioconductor packages, rentrez is from CRAN
BiocManager::install(c("Biostrings", "rentrez"))
```

- Validate installation

```
BiocManager::version() # current release is 3.11; needs R 4.0
## [1] '3.10'
BiocManager::valid()
## Warning: 10 packages out-of-date; 0 packages too new
##
## * sessionInfo()
##
## R version 3.6.3 Patched (2020-02-29 r77906)
## Platform: x86_64-apple-darwin17.7.0 (64-bit)
## Running under: macOS High Sierra 10.13.6
##
## Matrix products: default
## BLAS:   /Users/ma38727/bin/R-3-6-branch/lib/libRblas.dylib
## LAPACK: /Users/ma38727/bin/R-3-6-branch/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] BiocManager_1.30.10 compiler_3.6.3      magrittr_1.5
##  [4] bookdown_0.18       htmltools_0.4.0     tools_3.6.3
##  [7] yaml_2.2.1          Rcpp_1.0.4.6        stringi_1.4.6
## [10] rmarkdown_2.1       knitr_1.28          stringr_1.4.0
## [13] digest_0.6.25       xfun_0.13           rlang_0.4.6
## [16] evaluate_0.14
##
## Bioconductor version '3.10'
##
##   * 10 packages out-of-date
##   * 0 packages too new
##
## create a valid installation with
##
##   BiocManager::install(c(
##     "bookdown", "ellipsis", "modelr", "natserv", "RcppArmadillo", "sp",
##     "taxize", "tidyr", "tinytex", "xfun"
##   ), update = TRUE, ask = FALSE)
##
## more details: BiocManager::valid()$too_new, BiocManager::valid()$out_of_date
```

- Types of packages
  - Software (e.g., Biostrings, SingleCellExperiment)

  - Annotation (e.g., org.Hs.eg.db, TxDb.Hsapiens.UCSC.hg38.knownGene, GO.db)
  - Experiment data (e.g., airway, scRNAseq)

Key packages

- Biostrings for representing sequence data
- GenomicRanges for working with genomic coordinates
- SummarizedExperiment and SingleCellExperiment for representing experimental summaries, e.g., a gene x sample matrix of RNA-seq expression values.

'Class' and 'method'

- Classes represent data in ways that allow the specialized nature of the data to be exploited, e.g., it makes sense to calculate the `reverseComplement()` of a DNA sequence, but not of an arbitrary character vector.

```
library(Biostrings)
sequences <- c(
    chr1 = "CTGACT",
    chr2 = "CTGGGAACT"
)
dna <- DNAStringSet(sequences)
dna
##   A DNAStringSet instance of length 2
##     width seq                                                names
## [1]     6 CTGACT                                             chr1
## [2]     9 CTGGGAACT                                          chr2

## fails! `T` not in the RNA alphabet
result <- try( RNAStringSet(sequences) )
## Error in .Call2("new_XStringSet_from_CHARACTER", class(x0), elementType(x0),  :
##   key 84 (char 'T') not in lookup table
```

- Methods represent the implementation of common ('generic') operations, specialized to the specific classes that the method operates on.

```
length(dna)
## [1] 2
reverseComplement(dna)
##   A DNAStringSet instance of length 2
##     width seq                                                names
## [1]     6 AGTCAG                                             chr1
## [2]     9 AGTTCCCAG                                          chr2
translate(dna)
##   A AAStringSet instance of length 2
##     width seq                                                names
## [1]     2 MT                                                 chr1
```

```
## [2]     3 MGT                                                    chr2
```

  - E.g., one could `translate()` both a DNA sequence, and an RNA sequence, so there is both `translate,DNAStringSet-method` and `translate,RNAStringSet-method`

- Not all operations are implemented as methods on a class

  - E.g., `length()` is a concept shared by collections of DNA and RNA sequences, so the method is defined on a common 'parent' class

  - E.g., `dim()` of a two-dimensional object is sufficient to implement `nrow()` (as `dim()[1]`) and `ncol()` (as `dim()[2]`) so one could use a 'plain old' function `nrow <- function(x) dim(x)[1]` for `nrow()` / `ncol()`, relying on a `dim()` generic and methods to provide specialized behavior.

Getting help

- Vignettes, e.g., DESeq2 vignette! `browseVignettes(package = "DESeq2")`

- Help pages for classes, e.g., `?DNAStringSet`, `?GRanges`

- Discovering methods

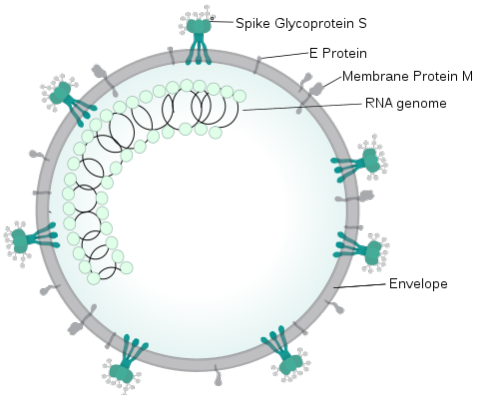
```
methods("reverse")
methods(class = "DNAStringSet")
```

- Looking at symbols made available by packages (after the package is attached to the `search()` path): `ls("package:GenomicRanges")`

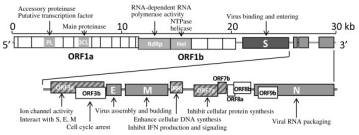
5.1.3 This week: SARS-CoV-2 sequence and human tissue-specific gene expression data

Biological background

- RNA virus (source, license: CC BY-SA 4.0)



- Key domains (source, license: CC BY 2)



Setup

- Some essential *Bioconductor* data structures

```
library(Biostrings)
library(GenomicRanges)
```

- Additional software

```
library(rentrez)
library(genbankr)
library(DECIPHER)
library(ggtree)
```

- Need to install software?
  - Install BiocManager

```
if (!requireNamespace("BiocManager"))
    install.packages("BiocManager", repos = "https://cran.r-project.org")
```

  - Install other *CRAN* or *Bioconductor* packges

```
pkgs <- c(
    ## packages needing installation, e.g.,
    "Biostrings", "GenomicRanges", "genbankr", "rentrez", "DECIPHER",
    "ggtree"
)
BiocManager::install(pkgs)
```

```
BiocManager::valid()
```

Biostrings and GenomicRanges

- Biostrings: representing DNA sequences

```
sequences <- c(
    chr1 = "CTGACT",
    chr2 = "CTGGGAACT"
)
dna <- DNAStringSet(sequences)
dna
##   A DNAStringSet instance of length 2
##     width seq                                                 names
## [1]     6 CTGACT                                              chr1
## [2]     9 CTGGGAACT                                           chr2
```

- GenomicRanges: annotations in genome space, e.g., create a variable representing 'regions of interest'

```
regions_of_interest <- c(
    "chr1:1-3",
    "chr1:3-6",
    "chr2:4-6"
)
roi <- GRanges(regions_of_interest)
roi
## GRanges object with 3 ranges and 0 metadata columns:
##       seqnames    ranges strand
##          <Rle> <IRanges>  <Rle>
##   [1]     chr1       1-3      *
##   [2]     chr1       3-6      *
##   [3]     chr2       4-6      *
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

- Many fun operations, e.g., extract sequences of regions of interest

```
getSeq(dna, roi)
##   A DNAStringSet instance of length 3
##     width seq
## [1]     3 CTG
## [2]     4 GACT
## [3]     3 GGA
```

Sequences and annotations

- Input FASTA files from New York State samples (it makes little biological sense to use just New York samples, but we'll do it anyway!) to

`DNAStringSet`

- Work the Genbank accession of the reference sequence
  - DNA sequence
  - Coding sequence annotations
  - Coding sequences

Alignment and visualization

- Multiple alignment using DECIPHER
- Visualization using ggtree

Gene expression

- Retrieving example data from ExperimentHub
- Annotating cell types

5.2 Day 30 DNA sequences and annotations

5.2.1 Setup

Set up a directory for working

```
workdir <- "workdir"
if (!dir.exists(workdir))
    dir.create(workdir)
```

Make sure the following packages are installed

```
library(Biostrings)
library(GenomicRanges)
library(rentrez)
library(genbankr)
```

- Install necessary packages, if needed

```
if (!requireNamespace("BiocManager"))
    install.packages("BiocManager", repos = "https://cran.r-project.org")

## for example...
pkgs <- c("Biostrings", "GenomicRanges", "rentrez", "genbankr")
BiocManager::install(pkgs)
```

5.2.2 Representing DNA sequences

Sample information

- Visit the NCBI SARS-CoV-2 site for orientation. We'll retrieve and analyse some of the samples sequenced in New York state

- Read a csv file summarizing records available from the US NCBI.

```
url <- "https://raw.githubusercontent.com/mtmorgan/QuaRantine/master/assets/05-ger
csv_file <- file.path(workdir, basename(url))
if ( !file.exists(csv_file) )
    download.file(url, csv_file)

records <- readr::read_csv(csv_file)
## Parsed with column specification:
## cols(
##   accession = col_character(),
##   link = col_character(),
##   date = col_date(format = ""),
##   country = col_character(),
##   region = col_character()
## )
records
## # A tibble: 2,434 x 5
##    accession link                                    date       country   regi
##    <chr>     <chr>                                   <date>     <chr>     <chr
##  1 NC_045512 https://www.ncbi.nlm.nih.gov/nuccore/N~ NA         China     <NA>
##  2 MT447189  https://www.ncbi.nlm.nih.gov/nuccore/M~ 2020-03-02 Uzbekist~ <NA>
##  3 MT447188  https://www.ncbi.nlm.nih.gov/nuccore/M~ 2020-03-02 Uzbekist~ <NA>
##  4 MT447177  https://www.ncbi.nlm.nih.gov/nuccore/M~ 2020-03-26 Iran      <NA>
##  5 MT447176  https://www.ncbi.nlm.nih.gov/nuccore/M~ 2020-03-20 Thailand  <NA>
##  6 MT447175  https://www.ncbi.nlm.nih.gov/nuccore/M~ 2020-03-19 Thailand  <NA>
##  7 MT447174  https://www.ncbi.nlm.nih.gov/nuccore/M~ 2020-03-01 Thailand  <NA>
##  8 MT447173  https://www.ncbi.nlm.nih.gov/nuccore/M~ 2020-03-18 Thailand  <NA>
##  9 MT447172  https://www.ncbi.nlm.nih.gov/nuccore/M~ 2020-03-18 Thailand  <NA>
## 10 MT447171  https://www.ncbi.nlm.nih.gov/nuccore/M~ 2020-03-16 Thailand  <NA>
## # ... with 2,424 more rows
```

FASTA sequence files

- Download a FASTA file containing DNA sequences of all samples sequenced in New York State (this doesn't make much sense biologically!)

```
url <- "https://raw.githubusercontent.com/mtmorgan/QuaRantine/master/assets/05-SAR
fasta_file <- file.path(workdir, basename(url))
if ( !file.exists(fasta_file) )
    download.file(url, fasta_file)
```

- This is a plain text file with a simple format. Each DNA sequence is on a line starting with an identifier, `>MT...`, and then lines representing the DNA (or RNA or amino acid) sequence.

```
readLines(fasta_file, 5)
## [1] ">MT434817"
```

```
## [2] "ATTAAAGGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAA"
## [3] "AATCTGTGTGGCTGTCACTCGGCTGCATGCTTAGTGCACTCACGCAGTATAATTAATAACTAATTACTGTCGTTGACAGG"
## [4] "ACACGAGTAACTCGTCTATCTTCTGCAGGCTGCTTACGGTTTCGTCCGTGTTGCAGCCGATCATCAGCACATCTAGGTTT"
## [5] "TGTCCGGGTGTGACCGAAAGGTAAGATGGAGAGCCTTGTCCCTGGTTTCAACGAGAAAACACACGTCCAACTCAGTTTGC"
```

- Load the Biostrings package

```
library(Biostrings)
options(Biostrings.coloring = FALSE)
```

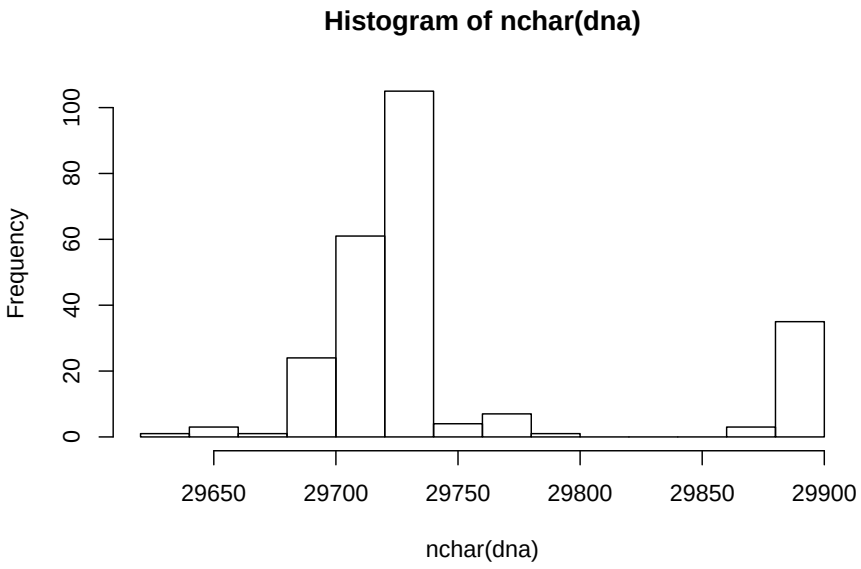
- Read the DNA sequences and explore basic properties

```
dna <- readDNAStringSet(fasta_file)
dna
##   A DNAStringSet instance of length 245
##       width seq                                                 names
##   [1] 29873 ATTAAAGGTTTATACCTTCCCA...GGAGAATGACAAAAAAAAAAAA MT434817
##   [2] 29879 ATTAAAGGTTTATACCTTCCCA...GGAGAATGACAAAAAAAAAAAA MT434816
##   [3] 29873 ATTAAAGGTTTATACCTTCCCA...GGAGAATGACAAAAAAAAAAAA MT434815
##   [4] 29882 ATTAAAGGTTTATACCTTCCCA...GGAGAATGACAAAAAAAAAAAA MT434814
##   [5] 29882 ATTAAAGGTTTATACCTTCCCA...GGAGAATGACAAAAAAAAAAAA MT434813
##   ...   ... ...
## [241] 29724 GATCTGTTCTCTAAACGAACTT...TGTACAGTGAACAATGCTAGGG MT370833
## [242] 29717 TCTAAACGAACTTTAAAATCTG...TACAGTGAACAATGCTAGGGAG MT370832
## [243] 29716 TCTAAACGAACTTTAAAATCTG...GTACAGTGAACAATGCTAGGGA MT370831
## [244] 29882 ATTAAAGGTTTATACCTTCCCA...GGAGAATGACAAAAAAAAAAAA MT325627
## [245] 29882 ATTAAAGGTTTATACCTTCCCA...GGAGAATGACAAAAAAAAAAAA MT304486
```

- Perform simple manipulations

```
## how many records?
length(dna)
## [1] 245

## subset
dna[1:3]
##   A DNAStringSet instance of length 3
##     width seq                                                 names
## [1] 29873 ATTAAAGGTTTATACCTTCCCAG...AGGAGAATGACAAAAAAAAAAAA MT434817
## [2] 29879 ATTAAAGGTTTATACCTTCCCAG...AGGAGAATGACAAAAAAAAAAAA MT434816
## [3] 29873 ATTAAAGGTTTATACCTTCCCAG...AGGAGAATGACAAAAAAAAAAAA MT434815
dna[c('MT434817', 'MT434815')]
##   A DNAStringSet instance of length 2
##     width seq                                                 names
## [1] 29873 ATTAAAGGTTTATACCTTCCCAG...AGGAGAATGACAAAAAAAAAAAA MT434817
## [2] 29873 ATTAAAGGTTTATACCTTCCCAG...AGGAGAATGACAAAAAAAAAAAA MT434815
```

```
## extract
dna[[1]]
##   29873-letter "DNAString" instance
## seq: ATTAAAGGTTTATACCTTCCCAGGTAACAAACCAAC...TTAATAGCTTCTTAGGAGAATGACAAAAAAAAAAA

## number of characters in each record
head(nchar(dna))
## [1] 29873 29879 29873 29882 29882 29882
hist(nchar(dna)) # histogram of sequence lengths
```



5.2.3 Working with `GenBankRecord` objects

Data is often represented in complicated formats that do not fit into the tidy 'table' format of a CSV file. An example is the information displayed on the SARS-CoV-2 reference genome web page.

Often 'someone' has written software to manipulate this data in a more convient way.

- Attach the rentrez package, which provides a way to query the NCBI's 'Entrez' collection of data bases. Use `BiocManager::install('rentrez')` if you need to install the package

```
library(rentrez)

## who's responsible?
maintainer("rentrez")
## [1] "David Winter <david.winter@gmail.com>"
```

```
## how to cite?
citation("rentrez")
##
## To cite rentrez in publications use:
##
##   Winter, D. J. (2017) rentrez: an R package for the NCBI eUtils API
##   The R Journal 9(2):520-526
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {{rentrez}: an R package for the NCBI eUtils API},
##     author = {David J. Winter},
##     journal = {The R Journal},
##     year = {2017},
##     volume = {9},
##     issue = {2},
##     pages = {520--526},
##   }
```

- The accession number for the SARS-CoV-2 reference sequence is `NC_045512`. Use `entrez_search()` to search the NCBI 'nuccore' database to discover information about this accession.

```
accession <- "NC_045512"
nuccore_record <- entrez_search("nuccore", accession)
id <- nuccore_record$ids
id
## [1] "1798174254"
```

- The `id` is an identifier that can be used to actually retrieve the genbank record

```
gb_record <-
    entrez_fetch("nuccore", id, rettype = "gbwithparts", retmode = "text")
```

- The record contains all the text in the record; if you like visualize it with

```
cat(gb_record)
```

Represent the data as `GenBankR` object

- Rather than try to 'munge' (e.g., copy and paste) relevant information from the GenBank record into R, use the genbankr package. Again, install with `BiocManager::install("genbankr")` if necessary

```
library(genbankr)
```

```
maintainer("genbankr")
## [1] "Gabriel Becker <becker.gabriel@gene.com>"

citation("genbankr")
##
## To cite package 'genbankr' in publications use:
##
##   Gabriel Becker and Michael Lawrence (2019). genbankr: Parsing GenBank
##   files into semantically useful objects. R package version 1.14.0.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {genbankr: Parsing GenBank files into semantically useful objects},
##     author = {Gabriel Becker and Michael Lawrence},
##     year = {2019},
##     note = {R package version 1.14.0},
##   }
```

- Use the function `readGenBank()` to parse the complicated text of `gb_record`

```
gb <- readGenBank(text = gb_record)
gb
## GenBank Annotations
## Severe acute respiratory syndrome coronavirus 2 isolate Wuhan-Hu-1, complete ge
## Accession: NC_045512
## 1 Sequence(s) with total length length: 29903
## 11 genes
## 12 transcripts
## 13 exons/cds elements
## 0 variations
## 34 other features
```

- Extract the reference sequence from the object

```
ref <- getSeq(gb)
ref
##   A DNAStringSet instance of length 1
##     width seq                                                names
## [1] 29903 ATTAAAGGTTTATACCTTCCCAG...AAAAAAAAAAAAAAAAAAAAAA Severe acute resp..
```

- Extract the coding sequences from the reference sequence; the `cds()` command is explained a little later.

```
cds <- getSeq(ref, cds(gb))
cds
```

```
##   A DNAStringSet instance of length 13
##      width seq
##  [1] 13203 ATGGAGAGCCTTGTCCCTGGTTTCAACGAGAAA...CTTCAGTCAGCTGATGCACAATCGTTTTTAAAC
##  [2]  8088 CGGGTTTGCGGTGTAAGTGCAGCCCGTCTTACA...GTTATTTCTAGTGATGTTCTTGTTAACAACTAA
##  [3] 13218 ATGGAGAGCCTTGTCCCTGGTTTCAACGAGAAA...GCACAATCGTTTTTAAACGGGTTTGCGGTGTAA
##  [4]  3822 ATGTTTGTTTTTCTTGTTTTATTGCCACTAGTC...GTGCTCAAAGGAGTCAAATTACATTACACATAA
##  [5]   828 ATGGATTTGTTTATGAGAATCTTCACAATTGGA...GAACCGACGACGACTACTAGCGTGCCTTTGTAA
##  ...   ... ...
##  [9]   366 ATGAAAATTATTCTTTTCTTGGCACTGATAACA...CTTTGCTTCACACTCAAAAGAAAGACAGAATGA
## [10]   132 ATGATTGAACTTTCATTAATTGACTTCTATTTG...CTGCAAGATCATAATGAAACTTGTCACGCCTAA
## [11]   366 ATGAAATTTCTTGTTTTCTTAGGAATCATCACA...CATGACGTTCGTGTTGTTTTAGATTTCATCTAA
## [12]  1260 ATGTCTGATAATGGACCCCAAAATCAGCGAAAT...TCCATGAGCAGTGCTGACTCAACTCAGGCCTAA
## [13]   117 ATGGGCTATATAAACGTTTTCGCTTTTCCGTTT...CAAGTAGATGTAGTTAACTTTAATCTCACATAG
```

- There are 13 coding sequences in the genome. Translate these to their amino acid sequences

```
translate(cds)
##   A AAStringSet instance of length 13
##      width seq
##  [1]  4401 MESLVPGFNEKTHVQLSLPVLQVRDVLVRGFGD...VCTVCGMWKGYGCSCDQLREPMLQSADAQSFLN
##  [2]  2696 RVCGVSAARLTPCGTGTSTDVVYRAFDIYNDKV...INDMILSLLSKGRLIIRENNRVVISSDVLVNN*
##  [3]  4406 MESLVPGFNEKTHVQLSLPVLQVRDVLVRGFGD...GMWKGYGCSCDQLREPMLQSADAQSFLNGFAV*
##  [4]  1274 MFVFLVLLPLVSSQCVNLTTRTQLPPAYTNSFT...SCLKGCCSCGSCCKFDEDDSEPVLKGVKLHYT*
##  [5]   276 MDLFMRIFTIGTVTLKQGEIKDATPSDFVRATA...VQIHTIDGSSGVVNPVMEPIYDEPTTTTSVPL*
##  ...   ... ...
##  [9]   122 MKIILFLALITLATCELYHYQECVRGTTVLLKE...QEEVQELYSPIFLIVAAIVFITLCFTLKRKTE*
## [10]    44 MIELSLIDFYLCFLAFLLFLVLIMLIIFWFSLELQDHNETCHA*
## [11]   122 MKFLVFLGIITTVAAFHQECSLQSCTQHQPYVV...CQEPKLGSLVVRCSFYEDFLEYHDVRVVLDFI*
## [12]   420 MSDNGPQNQRNAPRITFGGPSDSTGSNQNGERS...KQQTVTLLPAADLDDFSKQLQQSMSSADSTQA*
## [13]    39 MGYINVFAFPFTIYSLLLCRMNSRNYIAQVDVVNFNLT*
```

5.2.4 Genomic ranges and DNA sequences

Genomic ranges

- Attach the GenomicRanges package. If necessary, install with `BiocManager::install("GenomicRanges")`

```
library(GenomicRanges)
```

- Extract the coordinates of the coding sequences contained in the GenBank accession

```
cds <- cds(gb)
```

- Look at the genomic ranges defined in `cds`

```
granges(cds)
## GRanges object with 13 ranges and 0 metadata columns:
##                                                seqnames      ranges strand
##                                                   <Rle>   <IRanges>  <Rle>
##    [1] Severe acute respiratory syndrome coronavirus2   266-13468      +
##    [2] Severe acute respiratory syndrome coronavirus2 13468-21555      +
##    [3] Severe acute respiratory syndrome coronavirus2   266-13483      +
##    [4] Severe acute respiratory syndrome coronavirus2 21563-25384      +
##    [5] Severe acute respiratory syndrome coronavirus2 25393-26220      +
##    ...                                           ...         ...    ...
##    [9] Severe acute respiratory syndrome coronavirus2 27394-27759      +
##   [10] Severe acute respiratory syndrome coronavirus2 27756-27887      +
##   [11] Severe acute respiratory syndrome coronavirus2 27894-28259      +
##   [12] Severe acute respiratory syndrome coronavirus2 28274-29533      +
##   [13] Severe acute respiratory syndrome coronavirus2 29558-29674      +
##   -------
##   seqinfo: 1 sequence from NC_045512.2 genome
```

- There are 13 coding sequences in the reference genome. The sequence is named 'Severe acute respiratory syndrome coronavirus2'. The first region starts at position 266 of the reference genome, and goes to position 13468. The coding sequence is on the '+' strand.

- The full `cds` object has additional detail about each coding sequence. Display the full object (the display 'wraps' around several lines

```
cds
## GRanges object with 13 ranges and 14 metadata columns:
##                                                seqnames      ranges strand |
##                                                   <Rle>   <IRanges>  <Rle> |
##    [1] Severe acute respiratory syndrome coronavirus2   266-13468      + |
##    [2] Severe acute respiratory syndrome coronavirus2 13468-21555      + |
##    [3] Severe acute respiratory syndrome coronavirus2   266-13483      + |
##    [4] Severe acute respiratory syndrome coronavirus2 21563-25384      + |
##    [5] Severe acute respiratory syndrome coronavirus2 25393-26220      + |
##    ...                                           ...         ...    ... .
##    [9] Severe acute respiratory syndrome coronavirus2 27394-27759      + |
##   [10] Severe acute respiratory syndrome coronavirus2 27756-27887      + |
##   [11] Severe acute respiratory syndrome coronavirus2 27894-28259      + |
##   [12] Severe acute respiratory syndrome coronavirus2 28274-29533      + |
##   [13] Severe acute respiratory syndrome coronavirus2 29558-29674      + |
##                type        gene   locus_tag ribosomal_slippage
##         <character> <character> <character>          <logical>
##    [1]         CDS      ORF1ab  GU280_gp01               TRUE
##    [2]         CDS      ORF1ab  GU280_gp01               TRUE
##    [3]         CDS      ORF1ab  GU280_gp01              FALSE
```

```
##    [4]         CDS           S  GU280_gp02             FALSE
##    [5]         CDS       ORF3a  GU280_gp03             FALSE
##    ...         ...         ...         ...               ...
##    [9]         CDS       ORF7a  GU280_gp07             FALSE
##   [10]         CDS       ORF7b  GU280_gp08             FALSE
##   [11]         CDS        ORF8  GU280_gp09             FALSE
##   [12]         CDS           N  GU280_gp10             FALSE
##   [13]         CDS       ORF10  GU280_gp11             FALSE
##                                                  note codon_start
##                                           <character>   <numeric>
##    [1] pp1ab; translated by -1 ribosomal frameshift           1
##    [2] pp1ab; translated by -1 ribosomal frameshift           1
##    [3]                                          pp1a          1
##    [4]            structural protein; spike protein           1
##    [5]                                          <NA>          1
##    ...                                           ...        ...
##    [9]                                          <NA>          1
##   [10]                                          <NA>          1
##   [11]                                          <NA>          1
##   [12]                   ORF9; structural protein           1
##   [13]                                          <NA>          1
##                           product     protein_id         db_xref   translation
##                       <character>    <character> <CharacterList> <AAStringSet>
##    [1]          ORF1ab polyprotein YP_009724389.1 GeneID:43740578     MES...VNN
##    [2]          ORF1ab polyprotein YP_009724389.1 GeneID:43740578     MES...VNN
##    [3]           ORF1a polyprotein YP_009725295.1 GeneID:43740578     MES...FAV
##    [4]        surface glycoprotein YP_009724390.1 GeneID:43740568     MFV...HYT
##    [5]               ORF3a protein YP_009724391.1 GeneID:43740569     MDL...VPL
##    ...                         ...           ...               ...           ...
##    [9]               ORF7a protein YP_009724395.1 GeneID:43740573     MKI...KTE
##   [10]                      ORF7b YP_009725318.1 GeneID:43740574     MIE...CHA
##   [11]                ORF8 protein YP_009724396.1 GeneID:43740577     MKF...DFI
##   [12] nucleocapsid phosphoprotein YP_009724397.2 GeneID:43740575     MSD...TQA
##   [13]               ORF10 protein YP_009725255.1 GeneID:43740576     MGY...NLT
##           loctype       gene_synonym     gene_id transcript_id
##       <character>    <CharacterList> <character>   <character>
##    [1]      normal                NA  GU280_gp01  GU280_gp01.1
##    [2]      normal                NA  GU280_gp01  GU280_gp01.1
##    [3]      normal                NA  GU280_gp01  GU280_gp01.2
##    [4]      normal spike glycoprotein  GU280_gp02  GU280_gp02.1
##    [5]      normal                NA  GU280_gp03  GU280_gp03.1
##    ...         ...               ...         ...           ...
##    [9]      normal                NA  GU280_gp07  GU280_gp07.1
##   [10]      normal                NA  GU280_gp08  GU280_gp08.1
##   [11]      normal                NA  GU280_gp09  GU280_gp09.1
```

```
##   [12]      normal                 NA  GU280_gp10  GU280_gp10.1
##   [13]      normal                 NA  GU280_gp11  GU280_gp11.1
##   -------
##   seqinfo: 1 sequence from NC_045512.2 genome
```

or coerce to a tibble and explore

```
library(tibble)
library(dplyr)
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:GenomicRanges':
##
##     intersect, setdiff, union
## The following object is masked from 'package:GenomeInfoDb':
##
##     intersect
## The following objects are masked from 'package:Biostrings':
##
##     collapse, intersect, setdiff, setequal, union
## The following object is masked from 'package:XVector':
##
##     slice
## The following objects are masked from 'package:IRanges':
##
##     collapse, desc, intersect, setdiff, slice, union
## The following objects are masked from 'package:S4Vectors':
##
##     first, intersect, rename, setdiff, setequal, union
## The following objects are masked from 'package:BiocGenerics':
##
##     combine, intersect, setdiff, union
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
as_tibble(cds)
## # A tibble: 13 x 19
##    seqnames start   end width strand type  gene  locus_tag ribosomal_slipp~
##    <fct>    <int> <int> <int> <fct>  <chr> <chr> <chr>     <lgl>
##  1 Severe ~   266 13468 13203 +      CDS   ORF1~ GU280_gp~ TRUE
##  2 Severe ~ 13468 21555  8088 +      CDS   ORF1~ GU280_gp~ TRUE
##  3 Severe ~   266 13483 13218 +      CDS   ORF1~ GU280_gp~ FALSE
##  4 Severe ~ 21563 25384  3822 +      CDS   S     GU280_gp~ FALSE
```

```
##  5 Severe ~ 25393 26220   828 +      CDS   ORF3a GU280_gp~ FALSE
##  6 Severe ~ 26245 26472   228 +      CDS   E     GU280_gp~ FALSE
##  7 Severe ~ 26523 27191   669 +      CDS   M     GU280_gp~ FALSE
##  8 Severe ~ 27202 27387   186 +      CDS   ORF6  GU280_gp~ FALSE
##  9 Severe ~ 27394 27759   366 +      CDS   ORF7a GU280_gp~ FALSE
## 10 Severe ~ 27756 27887   132 +      CDS   ORF7b GU280_gp~ FALSE
## 11 Severe ~ 27894 28259   366 +      CDS   ORF8  GU280_gp~ FALSE
## 12 Severe ~ 28274 29533  1260 +      CDS   N     GU280_gp~ FALSE
## 13 Severe ~ 29558 29674   117 +      CDS   ORF10 GU280_gp~ FALSE
## # ... with 10 more variables: note <chr>, codon_start <dbl>, product <chr>,
## #   protein_id <chr>, db_xref <I<list>>, translation <chr>, loctype <chr>,
## #   gene_synonym <I<list>>, gene_id <chr>, transcript_id <chr>
```

For instance, we can find the gene associated with each coding sequence, using the following equivalent commands

```
cds$gene
##  [1] "ORF1ab" "ORF1ab" "ORF1ab" "S"      "ORF3a"  "E"      "M"      "ORF6"
##  [9] "ORF7a"  "ORF7b"  "ORF8"   "N"      "ORF10"

subset(cds, , gene)
## GRanges object with 13 ranges and 1 metadata column:
##                                                seqnames      ranges strand |
##                                                   <Rle>   <IRanges>  <Rle> |
##    [1] Severe acute respiratory syndrome coronavirus2   266-13468      + |
##    [2] Severe acute respiratory syndrome coronavirus2 13468-21555      + |
##    [3] Severe acute respiratory syndrome coronavirus2   266-13483      + |
##    [4] Severe acute respiratory syndrome coronavirus2 21563-25384      + |
##    [5] Severe acute respiratory syndrome coronavirus2 25393-26220      + |
##    ...                                              ...         ...    ... .
##    [9] Severe acute respiratory syndrome coronavirus2 27394-27759      + |
##   [10] Severe acute respiratory syndrome coronavirus2 27756-27887      + |
##   [11] Severe acute respiratory syndrome coronavirus2 27894-28259      + |
##   [12] Severe acute respiratory syndrome coronavirus2 28274-29533      + |
##   [13] Severe acute respiratory syndrome coronavirus2 29558-29674      + |
##                gene
##         <character>
##    [1]      ORF1ab
##    [2]      ORF1ab
##    [3]      ORF1ab
##    [4]           S
##    [5]       ORF3a
##    ...         ...
##    [9]       ORF7a
##   [10]       ORF7b
##   [11]        ORF8
```

```
##   [12]           N
##   [13]       ORF10
##   -------
##   seqinfo: 1 sequence from NC_045512.2 genome

as_tibble(cds) %>%
    ## info on gene and protein product
    dplyr::select(gene, product)
## # A tibble: 13 x 2
##    gene   product
##    <chr>  <chr>
##  1 ORF1ab ORF1ab polyprotein
##  2 ORF1ab ORF1ab polyprotein
##  3 ORF1ab ORF1a polyprotein
##  4 S      surface glycoprotein
##  5 ORF3a  ORF3a protein
##  6 E      envelope protein
##  7 M      membrane glycoprotein
##  8 ORF6   ORF6 protein
##  9 ORF7a  ORF7a protein
## 10 ORF7b  ORF7b
## 11 ORF8   ORF8 protein
## 12 N      nucleocapsid phosphoprotein
## 13 ORF10  ORF10 protein
```

Using genomic ranges to extract DNA sequences

- Recall the sequence of the entire genome

```
ref <- getSeq(gb)
ref
##   A DNAStringSet instance of length 1
##     width seq                                                names
## [1] 29903 ATTAAAGGTTTATACCTTCCCAG...AAAAAAAAAAAAAAAAAAAAAA Severe acute resp...
```

- Use `getSeq()` to extract the `DNAStringSet()` corresponding to the genomic ranges specified by `cds`

```
dna <- getSeq(ref, cds)
dna
##   A DNAStringSet instance of length 13
##      width seq
##  [1] 13203 ATGGAGAGCCTTGTCCCTGGTTTCAACGAGAAA...CTTCAGTCAGCTGATGCACAATCGTTTTTAAA
##  [2]  8088 CGGGTTTGCGGTGTAAGTGCAGCCCGTCTTACA...GTTATTTCTAGTGATGTTCTTGTTAACAACTA
##  [3] 13218 ATGGAGAGCCTTGTCCCTGGTTTCAACGAGAAA...GCACAATCGTTTTTAAACGGGTTTGCGGTGTA
##  [4]  3822 ATGTTTGTTTTTCTTGTTTTATTGCCACTAGTC...GTGCTCAAAGGAGTCAAATTACATTACACATA
##  [5]   828 ATGGATTTGTTTATGAGAATCTTCACAATTGGA...GAACCGACGACGACTACTAGCGTGCCTTTGTA
```

```
##  ...   ... ...
##  [9]   366 ATGAAAATTATTCTTTTCTTGGCACTGATAACA...CTTTGCTTCACACTCAAAAGAAAGACAGAATGA
## [10]   132 ATGATTGAACTTTCATTAATTGACTTCTATTTG...CTGCAAGATCATAATGAAACTTGTCACGCCTAA
## [11]   366 ATGAAATTTCTTGTTTTCTTAGGAATCATCACA...CATGACGTTCGTGTTGTTTTAGATTTCATCTAA
## [12]  1260 ATGTCTGATAATGGACCCCAAAATCAGCGAAAT...TCCATGAGCAGTGCTGACTCAACTCAGGCCTAA
## [13]   117 ATGGGCTATATAAACGTTTTCGCTTTTCCGTTT...CAAGTAGATGTAGTTAACTTTAATCTCACATAG
```

- It might be useful to coordinate information about each coding sequence with the sequence itself

```
cds$DNA <- dna
cds
## GRanges object with 13 ranges and 15 metadata columns:
##                                                seqnames      ranges strand |
##                                                   <Rle>   <IRanges>  <Rle> |
##    [1] Severe acute respiratory syndrome coronavirus2   266-13468      + |
##    [2] Severe acute respiratory syndrome coronavirus2 13468-21555      + |
##    [3] Severe acute respiratory syndrome coronavirus2   266-13483      + |
##    [4] Severe acute respiratory syndrome coronavirus2 21563-25384      + |
##    [5] Severe acute respiratory syndrome coronavirus2 25393-26220      + |
##    ...                                                       ...         ...    ... .
##    [9] Severe acute respiratory syndrome coronavirus2 27394-27759      + |
##   [10] Severe acute respiratory syndrome coronavirus2 27756-27887      + |
##   [11] Severe acute respiratory syndrome coronavirus2 27894-28259      + |
##   [12] Severe acute respiratory syndrome coronavirus2 28274-29533      + |
##   [13] Severe acute respiratory syndrome coronavirus2 29558-29674      + |
##              type        gene   locus_tag ribosomal_slippage
##       <character> <character> <character>          <logical>
##    [1]         CDS      ORF1ab  GU280_gp01               TRUE
##    [2]         CDS      ORF1ab  GU280_gp01               TRUE
##    [3]         CDS      ORF1ab  GU280_gp01              FALSE
##    [4]         CDS           S  GU280_gp02              FALSE
##    [5]         CDS       ORF3a  GU280_gp03              FALSE
##    ...         ...         ...         ...                ...
##    [9]         CDS       ORF7a  GU280_gp07              FALSE
##   [10]         CDS       ORF7b  GU280_gp08              FALSE
##   [11]         CDS        ORF8  GU280_gp09              FALSE
##   [12]         CDS           N  GU280_gp10              FALSE
##   [13]         CDS       ORF10  GU280_gp11              FALSE
##                                                note codon_start
##                                           <character>   <numeric>
##    [1] pp1ab; translated by -1 ribosomal frameshift           1
##    [2] pp1ab; translated by -1 ribosomal frameshift           1
##    [3]                                         pp1a           1
##    [4]            structural protein; spike protein           1
##    [5]                                         <NA>           1
```

```
##    ...                                              ...          ...
##    [9]                                             <NA>            1
##   [10]                                             <NA>            1
##   [11]                                             <NA>            1
##   [12]                      ORF9; structural protein               1
##   [13]                                             <NA>            1
##                             product     protein_id         db_xref   translatio
##                         <character>    <character> <CharacterList> <AAStringSet>
##    [1]          ORF1ab polyprotein YP_009724389.1 GeneID:43740578     MES...VN
##    [2]          ORF1ab polyprotein YP_009724389.1 GeneID:43740578     MES...VN
##    [3]           ORF1a polyprotein YP_009725295.1 GeneID:43740578     MES...FA
##    [4]        surface glycoprotein YP_009724390.1 GeneID:43740568     MFV...HY
##    [5]               ORF3a protein YP_009724391.1 GeneID:43740569     MDL...VP
##    ...                         ...            ...              ...          ...
##    [9]               ORF7a protein YP_009724395.1 GeneID:43740573     MKI...KT
##   [10]                      ORF7b YP_009725318.1 GeneID:43740574     MIE...CH
##   [11]                ORF8 protein YP_009724396.1 GeneID:43740577     MKF...DF
##   [12] nucleocapsid phosphoprotein YP_009724397.2 GeneID:43740575     MSD...TQ
##   [13]               ORF10 protein YP_009725255.1 GeneID:43740576     MGY...NL
##            loctype       gene_synonym     gene_id transcript_id            DNA
##        <character>    <CharacterList> <character>   <character> <DNAStringSet>
##    [1]      normal                  NA  GU280_gp01  GU280_gp01.1      ATG...AAC
##    [2]      normal                  NA  GU280_gp01  GU280_gp01.1      CGG...TAA
##    [3]      normal                  NA  GU280_gp01  GU280_gp01.2      ATG...TAA
##    [4]      normal spike glycoprotein  GU280_gp02  GU280_gp02.1      ATG...TAA
##    [5]      normal                  NA  GU280_gp03  GU280_gp03.1      ATG...TAA
##    ...         ...                 ...         ...           ...          ...
##    [9]      normal                  NA  GU280_gp07  GU280_gp07.1      ATG...TGA
##   [10]      normal                  NA  GU280_gp08  GU280_gp08.1      ATG...TAA
##   [11]      normal                  NA  GU280_gp09  GU280_gp09.1      ATG...TAA
##   [12]      normal                  NA  GU280_gp10  GU280_gp10.1      ATG...TAA
##   [13]      normal                  NA  GU280_gp11  GU280_gp11.1      ATG...TAG
##   -------
##   seqinfo: 1 sequence from NC_045512.2 genome
```

5.3 Day 31 Sequence alignment and visualization

5.3.1 'S' gene exploration

Read the 'S' gene sequence extracted from all genbank records into a `DNAStringSet`

```
url <- "https://raw.githubusercontent.com/mtmorgan/QuaRantine/master/assets/05-SARS-CoV2-S.fasta"
fasta_file <- file.path(workdir, basename(url))
if ( !file.exists(fasta_file) )
    download.file(url, fasta_file)
dna <- readDNAStringSet(fasta_file)
dna
##   A DNAStringSet instance of length 2289
##        width seq                                               names
##    [1]  3822 ATGTTTGTTTTTCTTGTTTTAT...GTCAAATTACATTACACATAA LC528232
##    [2]  3822 ATGTTTGTTTTTCTTGTTTTAT...GTCAAATTACATTACACATAA LC528233
##    [3]  3822 ATGTTTGTTTTTCTTGTTTTAT...GTCAAATTACATTACACATAA LC529905
##    [4]  3822 ATGTTTGTTTTTCTTGTTTTAT...GTCAAATTACATTACACATAA LC534418
##    [5]  3822 ATGTTTGTTTTTCTTGTTTTAT...GTCAAATTACATTACACATAA LC534419
##    ...   ... ...
## [2285]  3822 ATGTTTGTTTTTCTTGTTTTAT...GTCAAATTACATTACACATAA MT447174
## [2286]  3822 ATGTTTGTTTTTCTTGTTTTAT...GTCAAATTACATTACACATAA MT447175
## [2287]  3822 ATGTTTGTTTTTCTTGTTTTAT...GTCAAATTACATTACACATAA MT447176
## [2288]  3822 ATGTTTGTTTTTCTTGTTTTAT...GTCAAATTACATTACACATAA MT447177
## [2289]  3822 ATGTTTGTTTTTCTTGTTTTAT...GTCAAATTACATTACACATAA NC_045512
```

Explore the data. How many sequences? How many unique sequences?

```
length(dna)
## [1] 2289

dna %>% unique() %>% length()
## [1] 427
```

Note that the DNA alphabet contains 'ambiguity' letters, corresponding to base sequences that were uncertain, e.g., 'A' or 'T'. Check out the help page for `?translate` to see options for translating ambiguity codes. How many unique protein sequences?

```
IUPAC_CODE_MAP
##      A      C      G      T      M      R      W      S      Y      K      V
##    "A"    "C"    "G"    "T"   "AC"   "AG"   "AT"   "CG"   "CT"   "GT"  "ACG"
##      H      D      B      N
##  "ACT"  "AGT"  "CGT" "ACGT"

dna %>% alphabetFrequency() %>% colSums()
##       A       C       G       T       M       R       W       S       Y       K
## 2563642 1649084 1604491 2898077       2       9       6       0      17       6
##       V       H       D       B       N       -       +       .
##       1       0       1       1   33212       0       0       0

aa <- dna %>% translate(if.fuzzy.codon = "solve") %>% unique()
```

```
aa
##   A AAStringSet instance of length 343
##       width seq                                                 names
##   [1]  1274 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* LC528232
##   [2]  1274 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* MT007544
##   [3]  1273 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* MT012098
##   [4]  1274 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* MT020781
##   [5]  1274 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* MT039890
##   ...   ... ...
## [339]  1274 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* MT446361
## [340]  1274 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* MT447160
## [341]  1274 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* MT447163
## [342]  1274 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* MT447168
## [343]  1274 MFVFLVLLPLVSSQCVNLTTRI...CCKFDEDDSEPVLKGVKLHYT* MT447177
```

The `table()` function counts the number of occurrences of each element, so

```
nchar(aa) %>% table() %>% as_tibble()
## # A tibble: 2 x 2
##   .         n
##   <chr> <int>
## 1 1273      3
## 2 1274    340
```

shows us that most sequences have 1274 amino acids.

The `names(aa)` are misleading – these are the identifiers of a representative accession, but several accessions may map to the same identifier. The following code creates a table that indicates which cluster each original identifier belongs to; the names of `aa` are updated accordingly.

```
all <- dna %>% translate(if.fuzzy.codon = "solve")
## match(x, y) finds the index of each element of x in y.
## match(c("apple", "pear"), c("pear", "apple", "banana")) finds
## that 'apple' in x matches the second element of y, and 'pear'
## matches the first element of y. So the result is c(2, 1).
id <- match(all, aa)
clusters <- tibble(
    accession = names(all),
    label = as.character(id)
)
names(aa) <- match(aa, aa)
```

With this information, it's easy to count the number of times each sequence was represented. The reference sequence belongs to cluster 1.

```
clusters %>% filter(accession == "NC_045512")
## # A tibble: 1 x 2
##   accession label
##   <chr>     <chr>
## 1 NC_045512 1

clusters %>%
    count(label) %>%
    arrange(desc(n))
## # A tibble: 343 x 2
##    label     n
##    <chr> <int>
##  1 11     1117
##  2 1       723
##  3 126      21
##  4 34       10
##  5 341      10
##  6 125       6
##  7 167       6
##  8 47        6
##  9 59        6
## 10 170       5
## # ... with 333 more rows
```

5.3.2 Sequence alignment

Setup

- Create a working directory, if necessary. E.g.,

```
workdir <- "workdir"
if (!dir.exists(workdir))
    dir.create(workdir)
```

- Install (if necessary, using `BiocManager::install("DECIPHER")`) and attach the DECIPHER package

```
library(DECIPHER)

maintainer("DECIPHER")
## [1] "Erik Wright <eswright@pitt.edu>"

citation("DECIPHER")
##
## Wright ES (2016). "Using DECIPHER v2.0 to Analyze Big Biological
## Sequence Data in R." _The R Journal_, *8*(1), 352-359.
##
```

```
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {Using DECIPHER v2.0 to Analyze Big Biological Sequence Data in R},
##     author = {Erik S. Wright},
##     journal = {The R Journal},
##     year = {2016},
##     volume = {8},
##     number = {1},
##     pages = {352-359},
##   }
```

- This package contains tools for multiple sequence alignment. Explore the introductory and other vignettes in this package.

```
browseVignettes(package = "DECIPHER")
```

- Check out one of the help pages, e.g.,

```
?AlignSeqs
```

  Use `example(AlignSeqs)` to run the examples on the help page.

Perform multiple alignment

- Perform multiple alignment on the unique amino acid sequences using DECIPHER; this has minimal effect, with single insertions in the few sequences with unusual `nchar()`.

```
aln <- AlignSeqs(aa, verbose = FALSE)
aln
##   A AAStringSet instance of length 343
##       width seq                                                 names
##   [1]  1274 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* 1
##   [2]  1274 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* 2
##   [3]  1274 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* 3
##   [4]  1274 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* 4
##   [5]  1274 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* 5
##   ...   ... ...
## [339]  1274 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* 339
## [340]  1274 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* 340
## [341]  1274 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* 341
## [342]  1274 MFVFLVLLPLVSSQCVNLTTRT...CCKFDEDDSEPVLKGVKLHYT* 342
## [343]  1274 MFVFLVLLPLVSSQCVNLTTRI...CCKFDEDDSEPVLKGVKLHYT* 343

nchar(aln) %>% table() %>% as_tibble()
## # A tibble: 1 x 2
##   .         n
```

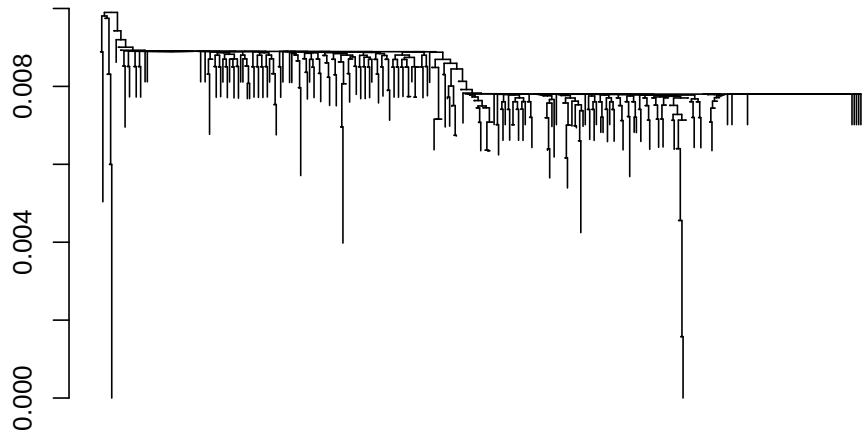
```
##   <chr> <int>
## 1 1274    343
```

- The 'distance matrix' measures the proximity (Hamming distance) of each sequence to other sequences in sequence space. The result is a large matrix of pairwise distances

```
dist <- DistanceMatrix(aln, verbose = FALSE)
class(dist)
## [1] "matrix"
dim(dist)
## [1] 343 343
dist[1:5, 1:5]
##              1           2           3           4           5
## 1 0.0000000000 0.0007849294 0.001569859 0.0007849294 0.0007849294
## 2 0.0007849294 0.0000000000 0.002354788 0.0015698587 0.0015698587
## 3 0.0015698587 0.0023547881 0.000000000 0.0023547881 0.0023547881
## 4 0.0007849294 0.0015698587 0.002354788 0.0000000000 0.0015698587
## 5 0.0007849294 0.0015698587 0.002354788 0.0015698587 0.0000000000
```

- Use the distance matrix to cluster sequences using the 'Neighbor Joining' algorithm; visualize the result (see `?plot.dendrogram` for plotting options when the first argument is of class `dendrogram`)

```
dendrogram <- IdClusters(dist, method = "NJ", type = "dendrogram")
## ================================================================================
##
## Time difference of 0.6 secs
dendrogram
## 'dendrogram' with 2 branches and 343 members total, at height 0.009898669
plot(dendrogram, leaflab = "none")
```



Save the tree to the working directory using the 'Newick' format

```
dendrogram_file <- file.path(workdir, "SARS-CoV2-S.newick")
WriteDendrogram(dendrogram, dendrogram_file, quoteLabels = FALSE)
```

5.3.3 Phylogenetic tree visualization

Setup

- Install (if necessary, using `BiocManager::install("ggtree")`) and attach the ggtree package

```
library(ggtree)

maintainer("ggtree")
## [1] "Guangchuang Yu <guangchuangyu@gmail.com>"

citation("ggtree")
##
## To cite ggtree in publications use:
##
##   Guangchuang Yu, David Smith, Huachen Zhu, Yi Guan, Tommy Tsan-Yuk
##   Lam. ggtree: an R package for visualization and annotation of
##   phylogenetic trees with their covariates and other associated data.
##   Methods in Ecology and Evolution 2017, 8(1):28-36
##
##   Guangchuang Yu, Tommy Tsan-Yuk Lam, Huachen Zhu, Yi Guan. Two methods
##   for mapping and visualizing associated data on phylogeny using
##   ggtree. Molecular Biology and Evolution 2018, 35(2):3041-3043. doi:
##   10.1093/molbev/msy194
##
## To see these entries in BibTeX format, use 'print(<citation>,
## bibtex=TRUE)', 'toBibtex(.)', or set
## 'options(citation.bibtex.max=999)'.
```

- The visualizations this package provides are not well-suited to *R*'s help pages; check out the online ggtree book to orient yourself.

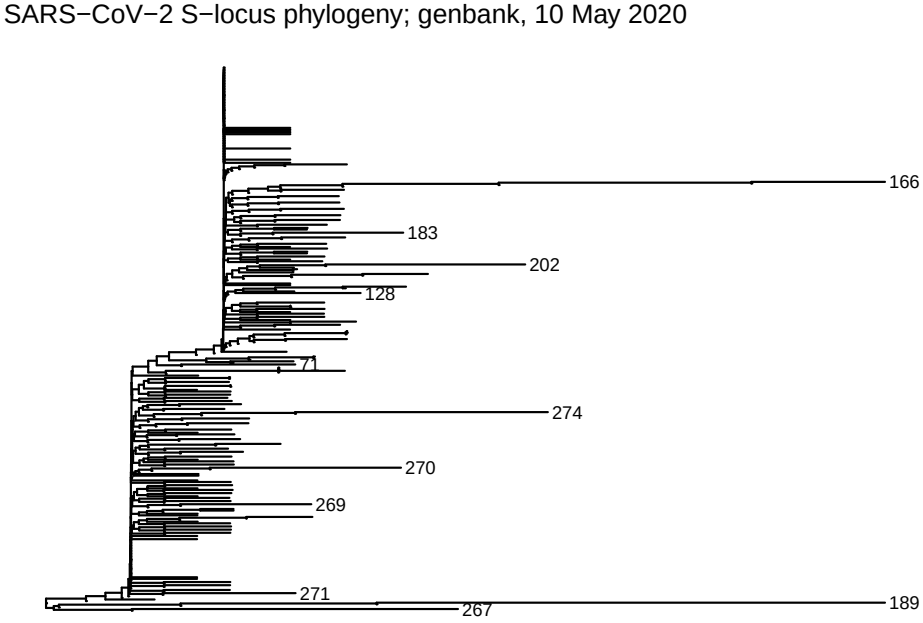
Read the data from the Newick-format file created by DECIPHER

```
dendrogram_file <- file.path(workdir, "SARS-CoV2-S.newick")
tree <- treeio::read.newick(dendrogram_file)
tree
##
## Phylogenetic tree with 343 tips and 288 internal nodes.
##
## Tip labels:
##  228, 267, 298, 294, 77, 189, ...
##
```

```
## Rooted; includes branch lengths.
tibble::as_tibble(tree)
## # A tibble: 631 x 4
##    parent  node branch.length label
##     <int> <int>         <dbl> <chr>
##  1    346     1       0       228
##  2    346     2       0.00385 267
##  3    347     3       0       298
##  4    348     4       0       294
##  5    349     5       0       77
##  6    349     6       0.00600 189
##  7    350     7       0       295
##  8    351     8       0.000580 280
##  9    352     9       0       297
## 10    353    10       0       87
## # ... with 621 more rows
```

Plot the tree, adding tip labels to the longest branches

```
ggtree(tree) +
    geom_tiplab(
        aes(
            subset = branch.length > .001,
            label = label
        ),
        size = 3
    ) +
    ggplot2::ggtitle("SARS-CoV-2 S-locus phylogeny; genbank, 10 May 2020")
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```



Color sequences by region

- Find nodes representing just a single sequence – 'singletons'
- Use GenBank country to identify which country these were sequenced from
- Color lineages according to singleton country of origin

5.4 Day 32 Single-cell expression data

This is a very preliminary foray into COVID-19 single-cell RNA-seq gene expression data. It is derived from a preprint by Muus et al, including a 'Terra' (NHGRI) cloud computing workspace.

RNA-seq is a technology for measuring RNA expression levels as a proxy for gene expression. Single-cell RNA seq measures RNA expression levels at the single-cell resolution, sometimes in conjunction with imaging or other spatial information. Often, the overall pattern of expression of a single cell can be used to classify the cell to a particular type, e.g., epithelial.

Many interesting questions present themselves in the context of COVID-19. It would be interesting to investigate patterns of expression of human genes known to interact with the SARS-CoV-2 virus, especially in tissues likely to be involved in the introduction of the virus to the human body. One could imagine further studies identifying, e.g., genetic variants that influence susceptibility or response to infection.

Our goal is to do some preliminary visualization of scRNA seq at human genes whose product is known to interact with virus proteins. A much more compre-

hensive guide is available as Orchestrating Single Cell Analysis with *Bioconductor*.

5.4.1 Setup

This module requires a recent version of *R* and *Bioconductor*. It also requires some advanced packages that may be difficult to install. Definitely it is necessary that

```
as.package_version(R.version) >= "3.6"
## [1] TRUE
BiocManager::version() >= "3.10"
## [1] TRUE
```

If those conditions are met, load the following libraries (use `BiocManager::install(<pkgs>)` if necessary)

```
library(SingleCellExperiment)
library(Matrix)
library(rhdf5)
library(dplyr)
library(hexbin)
```

Create a working directory for today.

```
workdir <- "workdir"
if (!dir.exists(workdir))
    dir.create(workdir)
```

A couple of scripts provide helper functions for working with the data. Download the following file to the working directory

```
url <- "https://raw.githubusercontent.com/mtmorgan/QuaRantine/master/assets/05-scRNAseq-import.R"
destination <- file.path(workdir, basename(url))
if (!file.exists(destination))
    download.file(url, destination)
```

Take a quick look at the downloaded file (e.g., in RStudio File -> open) and verify that it is an *R* script. Source the script in to your *R* session so that you have access to the functions in the file.

```
source(destination)
```

Download our sample data set

```
url <- "https://github.com/mtmorgan/QuaRantine/raw/master/assets/05-internal_nonsmokerslung.h5ad"
destination <- file.path(workdir, basename(url))
if (!file.exists(destination))
    download.file(url, destination)
```

The file is an 'hdf5' file format, used to store relatively large data sets in a

binary format that allows for faster and more robust input than a CSV file. *R* and other software packages can read hdf5 files, but Excel, etc.

5.4.2 Data input and exploration

There are two main components to the data. The first describes each observation (cell).

```
obs <- read_obs(destination)
obs
## # A tibble: 240,511 x 7
##      Age `Cell Type` Count Gender Patient_ID Tissue                cell_subset
##    <dbl> <fct>       <int> <fct>  <fct>      <fct>                 <fct>
##  1    57 Basal           1 F      HU28       internal_nonsmoker~ Epithelial (ba~
##  2    57 Basal           1 F      HU28       internal_nonsmoker~ Epithelial (ba~
##  3    57 Basal           1 F      HU28       internal_nonsmoker~ Epithelial (ba~
##  4    57 Basal           1 F      HU28       internal_nonsmoker~ Epithelial (ba~
##  5    57 Secretory       1 F      HU28       internal_nonsmoker~ Secretory
##  6    57 Basal           1 F      HU28       internal_nonsmoker~ Epithelial (ba~
##  7    57 Basal           1 F      HU28       internal_nonsmoker~ Epithelial (ba~
##  8    57 Basal           1 F      HU28       internal_nonsmoker~ Epithelial (ba~
##  9    57 Basal           1 F      HU28       internal_nonsmoker~ Epithelial (ba~
## 10    57 Secretory       1 F      HU28       internal_nonsmoker~ Secretory
## # ... with 240,501 more rows
```

If *R* complains that it can't find `read_obs`, then remember to `source()` the script downloaded earlier.

This experiment measured gene expression in 240511 cells. Use standard dplyr commands to explore information about the cells

- How many patients are represented? (we need to use `dplyr::count()` because another package attached to our search path also has a `count()` function)

```
obs %>% dplyr::count(Patient_ID)
## # A tibble: 7 x 2
##   Patient_ID     n
##   <fct>      <int>
## 1 HU28       12301
## 2 HU30       33971
## 3 HU37       66594
## 4 HU39       29908
## 5 HU49       56575
## 6 HU52       17768
## 7 HU62       23394
```

- What is the age and gender of each patient?

```
obs %>% distinct(Patient_ID, Age, Gender)
## # A tibble: 7 x 3
##   Patient_ID   Age Gender
##   <fct>      <dbl> <fct>
## 1 HU28        57   F
## 2 HU39         0.3 M
## 3 HU62        46   M
## 4 HU30        59   F
## 5 HU49        18   F
## 6 HU37        42   F
## 7 HU52        66   F
```

- Which cell types are represented (the column contains a space in its name, which causes problems for *R* unless the column is quoted)

```
obs %>% dplyr::count(`Cell Type`) %>% arrange(desc(n))
## # A tibble: 17 x 2
##    `Cell Type`        n
##    <fct>          <int>
##  1 Basal          67554
##  2 AT2            28326
##  3 Fibroblast     27574
##  4 T.NK.cells     26709
##  5 Myeloid        19730
##  6 Secretory      18643
##  7 AT1            11722
##  8 Endothelial     9611
##  9 Ciliated        9445
## 10 SmoothMuscle    9392
## 11 Pericytes       3385
## 12 B.cells         3232
## 13 Mesothelium     2179
## 14 Mast            1991
## 15 Tuft.like        677
## 16 Neuroendocrine   179
## 17 Ionocytes        162
```

- Each cell was classified to cell type based on overall gene expression. How many cells of each type were observed?

```
obs %>% dplyr::count(cell_subset)
## # A tibble: 17 x 2
##    cell_subset                          n
##    <fct>                            <int>
##  1 Endothelial cell                  9611
##  2 Epithelial (basal)               67554
##  3 Epithelial (ciliated)             9445
```

```
##  4 Epithelial cell (alveolar type I)  11722
##  5 Epithelial cell (alveolar type II) 28326
##  6 Fibroblast                         27574
##  7 Immune (B cell)                     3232
##  8 Immune (NKT cell)                  26709
##  9 Immune (mast cell)                  1991
## 10 Immune (myeloid)                   19730
## 11 Ionocyte                             162
## 12 Mesothelium                         2179
## 13 Neuroendocrine                       179
## 14 Pericyte                            3385
## 15 Secretory                          18643
## 16 Smooth muscle cell                  9392
## 17 Tuft                                 677
```

Single-cell RNA assays expression across all genes, but the current data set has been restricted to 22 genes relevant to virus interaction. Read in the expression levels of these 22 genes across all cells.

```
x <- read_expression(destination)
dim(x)
## [1]     22 240511
class(x)
## [1] "dgCMatrix"
## attr(,"package")
## [1] "Matrix"

x[, 1:3]
## 22 x 3 sparse Matrix of class "dgCMatrix"
##         1B_cell57_HU28 1B_cell418_HU28 1B_cell611_HU28
## ACE2         .               .              .
## TMPRSS2      0.6002854       0.7688452      .
## IL6          .               .              .
## IL6R         .               .              .
## IL6ST        .               0.7688452      .
## PCSK1        .               .              .
## PCSK2        .               .              .
## FURIN        .               .              .
## PCSK4        .               .              .
## PCSK5        .               .              .
## PCSK6        .               .              .
## PCSK7        .               .              0.7382636
## C1R          .               .              .
## C2           .               .              .
## C3           .               .              .
## C5           .               .              .
```

```
## CFI          .            .            .
## CTSS         0.9727762    0.7688452    .
## CTSL         .            .            .
## CTSB         1.2435541    1.1983210    .
## CTSC         .            0.7688452    .
## CTSE         .            .            .
```

This is a matrix of 22 x 240511 values, one entry for each gene and cell. However, single-cell assays are 'sparse', with a large number of zeros. To take advantage of this sparse data, *R* has represented the matrix as a special *sparse* matrix. The `.` in the display above represent zeros; the object `x` only contains information about the non-zero values.

How sparse is the data, i.e., what is the fraction of zeros in the data?

- Total number of elements

```
nrow(x) * ncol(x)
## [1] 5291242
```

- Total number of zero elements (`x == 0` returns a logical vector of `TRUE` and `FALSE` values; `sum()` treats `TRUE` as 1, and `FALSE` as 0, so `sum(x == 0)` is the number of zero elements)

```
sum(x == 0)
## [1] 4321307
```

- Fraction of zero elements

```
sum(x == 0) / (nrow(x) * ncol(x))
## [1] 0.8166905
```

- More than 80% of the values are zeros! This primarily represents limitations of the technology, where relatively few 'reads' are available from each cell, so only a fraction of the genes being expressed are detected.
- Use the function `rowMeans()` to calculate average expression of each gene; present this as a tibble

```
tibble(
    gene = rownames(x),
    average_expression = rowMeans(x)
)
## # A tibble: 22 x 2
##    gene    average_expression
##    <chr>                <dbl>
##  1 ACE2               0.00539
##  2 TMPRSS2            0.0877
##  3 IL6                0.144
##  4 IL6R               0.0923
```

```
##  5 IL6ST              0.386
##  6 PCSK1              0.00302
##  7 PCSK2              0.0117
##  8 FURIN              0.0908
##  9 PCSK4              0.00728
## 10 PCSK5              0.0740
## # ... with 12 more rows
```

5.4.3 Coordinating cell and expression data

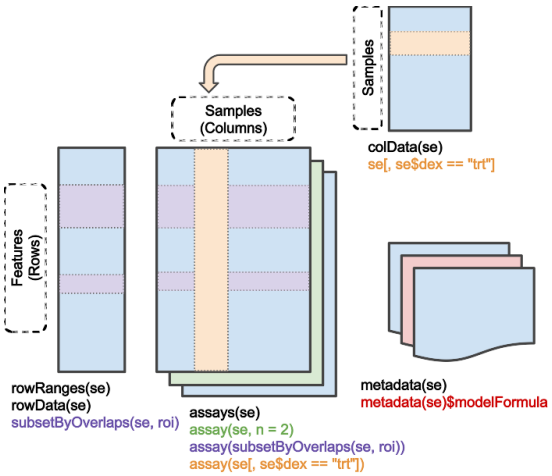
Notice that we have two separate data structures – a large matrix of expression values, and a tibble describing each cell. It seems error-prone to manage these data separately, e.g., what if we remove some rows from the cell annotations without removing the corresponding columns from the expression matrix?

A better strategy is to introduce an object that can manage both parts of the data in a coordinated way. This container is provided by the SingleCellExperiment package

```
library(SingleCellExperiment)
```

Use the *constructor* `SingleCellExperiment()` to create this object. The basic metaphor is that a `SingleCellExperiment` is like a matrix, but with column (and row) annotations. The column annotations are called `colData`.

```
knitr::include_graphics('images/SummarizedExperiment.png')
```



Create and view a `SingleCellExperiment`

```
sce <- SingleCellExperiment(x, colData = obs)
sce
## class: SingleCellExperiment
```

```
## dim: 22 240511
## metadata(0):
## assays(1): ''
## rownames(22): ACE2 TMPRSS2 ... CTSC CTSE
## rowData names(0):
## colnames(240511): 1B_cell57_HU28 1B_cell418_HU28 ...
##   RP2_neg_cell6794209_HU52 RP2_neg_cell6794225_HU52
## colData names(7): Age Cell Type ... Tissue cell_subset
## reducedDimNames(0):
## spikeNames(0):
## altExpNames(0):

dim(sce)
## [1]     22 240511
```

Use `subset()` to select particular rows and columns. The subset operates in a coordinated fashion on both the cell annotations and expression data, so there is no risk of 'book-keeping' error.

```
my_sce <-
    sce %>%
    subset(
        rownames(.) %in% "ACE2",
        `Cell Type` %in% c("Fibroblast", "SmoothMuscle")
    )

my_sce
## class: SingleCellExperiment
## dim: 1 36966
## metadata(0):
## assays(1): ''
## rownames(1): ACE2
## rowData names(0):
## colnames(36966): 1B_cell31801_HU28 1B_cell48220_HU28 ...
##   RP2_neg_cell6778193_HU52 RP2_neg_cell6785080_HU52
## colData names(7): Age Cell Type ... Tissue cell_subset
## reducedDimNames(0):
## spikeNames(0):
## altExpNames(0):

dim(my_sce)
## [1]     1 36966
```

For the exercises below, focus on the ACE2 gene. ACE2 is a gene on the outer surface of the cell membrane in lung and other tissue.

```
ace2 <- subset(sce, rownames(sce) %in% "ACE2")
ace2
## class: SingleCellExperiment
## dim: 1 240511
## metadata(0):
## assays(1): ''
## rownames(1): ACE2
## rowData names(0):
## colnames(240511): 1B_cell57_HU28 1B_cell418_HU28 ...
##   RP2_neg_cell6794209_HU52 RP2_neg_cell6794225_HU52
## colData names(7): Age Cell Type ... Tissue cell_subset
## reducedDimNames(0):
## spikeNames(0):
## altExpNames(0):
```

We will perform some simple visualization, as we start to explore the data. Create a tibble containing the `cell_subset` annotation and the expression values of the ACE2 gene. We make use of `$` to access columns of the cell annotation (`colData`) and `assay()` to access the expression data. `as.vector()` changes the one-dimensional array returned by `assay()` into a numeric vector.

```
tbl <- tibble(
    cell_subset = ace2$cell_subset,
    expression = as.vector(assay(ace2))
)
tbl
## # A tibble: 240,511 x 2
##    cell_subset        expression
##    <fct>                   <dbl>
##  1 Epithelial (basal)          0
##  2 Epithelial (basal)          0
##  3 Epithelial (basal)          0
##  4 Epithelial (basal)          0
##  5 Secretory                   0
##  6 Epithelial (basal)          0
##  7 Epithelial (basal)          0
##  8 Epithelial (basal)          0
##  9 Epithelial (basal)          0
## 10 Secretory                   0
## # ... with 240,501 more rows
```

Summarize the number of cells, non-zero expression values, and average expression of ACE2 for each cell subset.

```
tbl %>%
    group_by(cell_subset) %>%
```

```
    summarize(
        n = n(),
        n_non_zero = sum(expression != 0),
        mean_expression = mean(expression)
    )
```

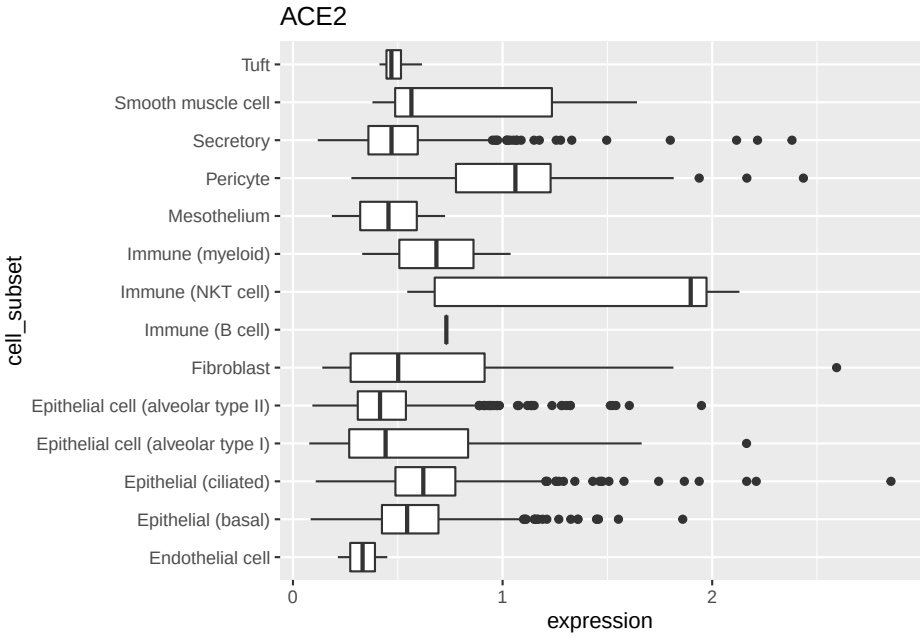
```
## # A tibble: 17 x 4
##    cell_subset                          n n_non_zero mean_expression
##    <fct>                            <int>      <int>           <dbl>
##  1 Endothelial cell                  9611          2       0.0000691
##  2 Epithelial (basal)               67554        854       0.00726
##  3 Epithelial (ciliated)             9445        229       0.0167
##  4 Epithelial cell (alveolar type I) 11722         67       0.00341
##  5 Epithelial cell (alveolar type II) 28326        594       0.00967
##  6 Fibroblast                       27574         25       0.000648
##  7 Immune (B cell)                   3232          1       0.000226
##  8 Immune (NKT cell)                26709          5       0.000270
##  9 Immune (mast cell)                1991          0       0
## 10 Immune (myeloid)                 19730          2       0.0000694
## 11 Ionocyte                           162          0       0
## 12 Mesothelium                       2179          2       0.000418
## 13 Neuroendocrine                     179          0       0
## 14 Pericyte                          3385         23       0.00765
## 15 Secretory                        18643        533       0.0146
## 16 Smooth muscle cell                9392          7       0.000642
## 17 Tuft                               677          4       0.00291
```

Pictures speak louder than words. Use the ggplot2 package

```
library(ggplot2)
```

to visualize as a horizontal boxplot the ACE2 expression values of each cell subtype. I had initially used `geom_boxplot()` without `coord_flip()`, but the cell subtype labels overlapped. I experimented with rotating the labels, but that just made them difficult to read. So `coord_flip()` makes the boxplot horizontal rather than vertical, the labels are easy to read, and the figure easy to interpret.

```
tbl %>%
    filter(expression > 0) %>%
    ggplot(aes(cell_subset, expression)) +
        geom_boxplot() +
        coord_flip() +
        ggtitle("ACE2")
```



I want to explore co-expression of two genes, ACE2 and TMPRSS2. I used `subset()` to select the relevant expression and cell annotation values, then created a tibble with gene, cell, and expression columns.

- The `rep()` function replicates each element of its first argument the number of times indicated by it's second argument; explore it's behavior with simple examples, e.g.,

```
1:3
## [1] 1 2 3

rep(1:3, 2)
## [1] 1 2 3 1 2 3

rep(1:3, each = 2)
## [1] 1 1 2 2 3 3
```

- When a two-dimensional array is coerced to a vector, it is 'flattened' by appending columns

```
m <- matrix(1:12, nrow = 4)
m
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
as.vector(m)
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

- Putting this together

```
two_genes <- subset(sce, rownames(sce) %in% c("ACE2", "TMPRSS2"))
tbl <- tibble(
    gene = rep(rownames(two_genes), ncol(two_genes)),
    cell = rep(colnames(two_genes), each = nrow(two_genes)),
    expression = as.vector(assay(two_genes))
)
tbl
## # A tibble: 481,022 x 3
##    gene    cell             expression
##    <chr>   <chr>                 <dbl>
##  1 ACE2    1B_cell57_HU28        0
##  2 TMPRSS2 1B_cell57_HU28        0.600
##  3 ACE2    1B_cell418_HU28       0
##  4 TMPRSS2 1B_cell418_HU28       0.769
##  5 ACE2    1B_cell611_HU28       0
##  6 TMPRSS2 1B_cell611_HU28       0
##  7 ACE2    1B_cell617_HU28       0
##  8 TMPRSS2 1B_cell617_HU28       0
##  9 ACE2    1B_cell885_HU28       0
## 10 TMPRSS2 1B_cell885_HU28       1.09
## # ... with 481,012 more rows
```

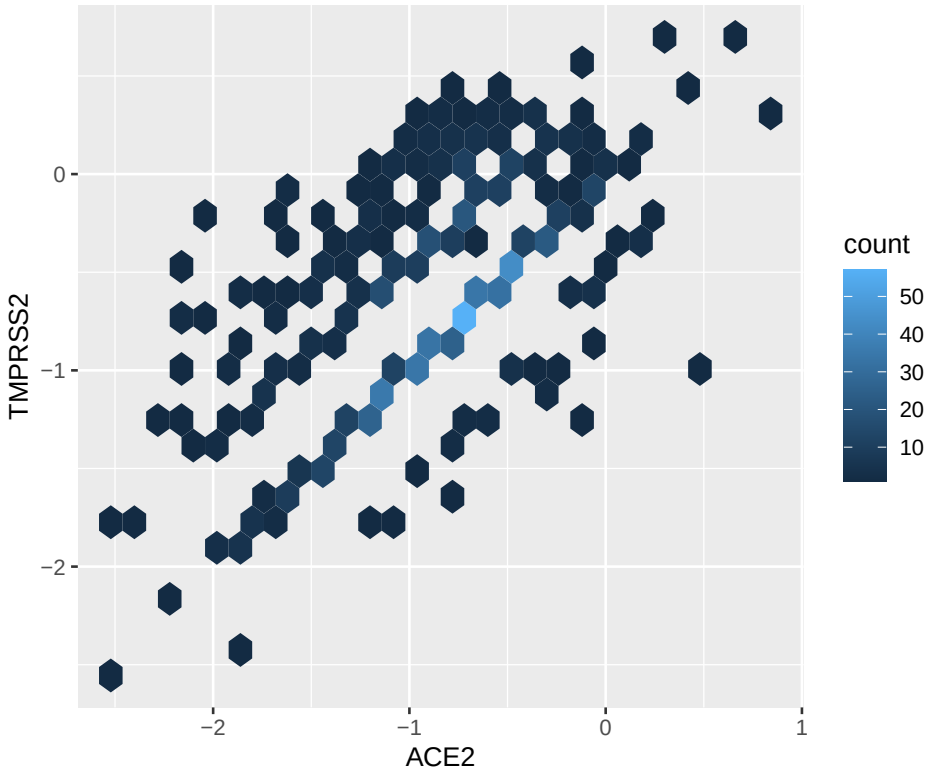
I'd like to plot the expression of one gene against the expression of the other. It is convenient to transform the tidy tibble we just created to have a separate column for each gene. Use `tidyr::pivot_wider()` for this operation

```
wider_tbl <-
    tbl %>%
    mutate(expression = log(expression)) %>%
    tidyr::pivot_wider(names_from=gene, values_from=expression)

wider_tbl
## # A tibble: 240,511 x 3
##    cell              ACE2   TMPRSS2
##    <chr>            <dbl>     <dbl>
##  1 1B_cell57_HU28    -Inf   -0.510
##  2 1B_cell418_HU28   -Inf   -0.263
##  3 1B_cell611_HU28   -Inf -Inf
##  4 1B_cell617_HU28   -Inf -Inf
##  5 1B_cell885_HU28   -Inf    0.0879
##  6 1B_cell968_HU28   -Inf -Inf
```

```
##  7 1B_cell1008_HU28  -Inf -Inf
##  8 1B_cell1060_HU28  -Inf   -0.300
##  9 1B_cell1156_HU28  -Inf   -0.675
## 10 1B_cell1370_HU28  -Inf -Inf
## # ... with 240,501 more rows
```

We're ready to plot expression values of TMPRSS2 against ACE2. If we were to use `geom_point()`, many points would over-plot and we would have a difficult time seeing overall pattern. So instead we use `geom_hex()` to divide the plot area into hexagonal regions, and to fill each region with a color that indicates the number of points in the region. `coord_fix()` ensures that the x- and y- axes have the same scale, and the plot is printed so that the 'aspect ratio' (height / width) is 1.

```
wider_tbl %>%
    ggplot(aes(ACE2, TMPRSS2)) +
    geom_hex() +
    coord_fixed()
## Warning: Removed 239728 rows containing non-finite values (stat_binhex).
```



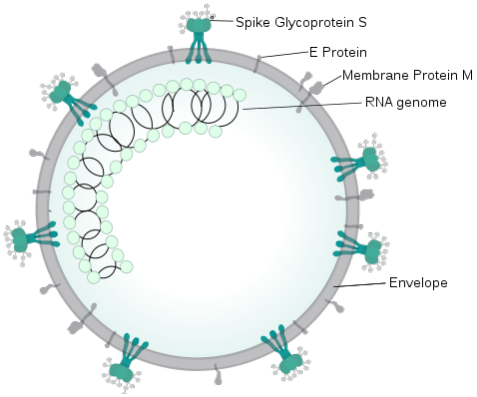
The plot shows a positive relation between expression of the two genes.

A major caveat to any conclusion of co-expression from the figure is that *covariates* may introduce spurious correlations. For instance, sequencing of one cell can be more efficient than sequencing of another, so *all* genes in one cell might appear to have higher expression than in another cell, but for completely uninteresting reasons (i.e., a technological artifact of sequencing efficiency). We would need to do a more sophisticated analysis to reach robust conclusions.

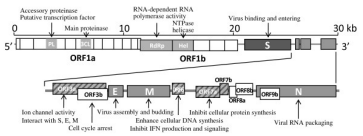
5.5 Day 33 (Friday) Zoom check-in

Biological background

- RNA virus (source, license: CC BY-SA 4.0)



- Key domains (source, license: CC BY 2)



5.5.1 Monday troubles (10 minutes)

Where we ran into trouble on Monday

- Load specific packages

```
library(Biostrings)
options(Biostrings.coloring = FALSE)
library(GenomicRanges)
```

- DNA sequences and genomic ranges

```
sequences <- c(chr1 = "CTGACT", chr2 = "CTGGGAACT")
dna <- DNAStringSet(sequences)
```

```
regions_of_interest <- c("chr1:1-3", "chr1:3-6", "chr2:4-6")
roi <- GRanges(regions_of_interest)
```

- So far so good, but…

```
> getSeq(dna, roi)
Error in (function (classes, fdef, mtable)  :
  unable to find an inherited method for function 'getSeq' for signature '"DNAStri
```

- What went wrong? The `getSeq()` *generic* is defined in Biostrings, but the `getSeq()` *method* `getSeq,DNAStringSet-method` is not! Hence 'unable to find an inherited method … for signature …'.
- Why did it work in the QuaRantine document? I had also loaded another package

```
library(genbankr)
```

  - genbankr depends on the package BSgenome
  - BSgenome defines a method `getSeq,DNAStringSet-method`.
  - The generic in Biostrings now finds the method in BSgenome!

```
getSeq(dna, roi)
##   A DNAStringSet instance of length 3
##     width seq
## [1]     3 CTG
## [2]     4 GACT
## [3]     3 GGA
```

5.5.2 Review and trouble shoot (40 minutes)

Bioconductor

- Useful resource: statistical analysis and comprehension of high-throughput genomic data.
- `BiocManager::install()`, `BiocManager::version()`, `BiocManager::valid()`
- Key concepts: package, vignette, class, method.

SARS-CoV-2 sequence data

- Querying Entrez for NC_045512 and parsing GenBank records

```
library(genbankr)
gb <- readGenBank(GBAccession("NC_045512"))
## No exons read from genbank file. Assuming sections of CDS are full exons
## No transcript features (mRNA) found, using spans of CDSs
gb
```

```
## GenBank Annotations
## Severe acute respiratory syndrome coronavirus 2 isolate Wuhan-Hu-1, complete genome.
## Accession: NC_045512
## 1 Sequence(s) with total length length: 29903
## 11 genes
## 12 transcripts
## 13 exons/cds elements
## 0 variations
## 34 other features
```

- Extract whole-genome sequence and annotation

```
ref <- getSeq(gb)
ref
##   A DNAStringSet instance of length 1
##     width seq                                                 names
## [1] 29903 ATTAAAGGTTTATACCTTCCCAG...AAAAAAAAAAAAAAAAAAAAAA Severe acute resp...

cds <- cds(gb)
getSeq(ref, subset(cds, gene == "S"))
##   A DNAStringSet instance of length 1
##     width seq
## [1]  3822 ATGTTTGTTTTTCTTGTTTTATTGCCACTAGTCT...GTGCTCAAAGGAGTCAAATTACATTACACATAA
```

- Multiple alignment and phylogenetic tree construction

```
library(DECIPHER)
url <- "https://raw.githubusercontent.com/mtmorgan/QuaRantine/master/assets/05-SARS-CoV2-S.f
fasta_file <- file.path(workdir, basename(url))
if ( !file.exists(fasta_file) )
    download.file(url, fasta_file)
dna <- readDNAStringSet(fasta_file)
aa <- dna %>% translate(if.fuzzy.codon = "solve") %>% unique()
aln <- AlignSeqs(aa, verbose = FALSE)
dist <- DistanceMatrix(aln, verbose = FALSE)
dendrogram <-
    IdClusters(dist, method = "NJ", type = "dendrogram", verbose = FALSE)
```

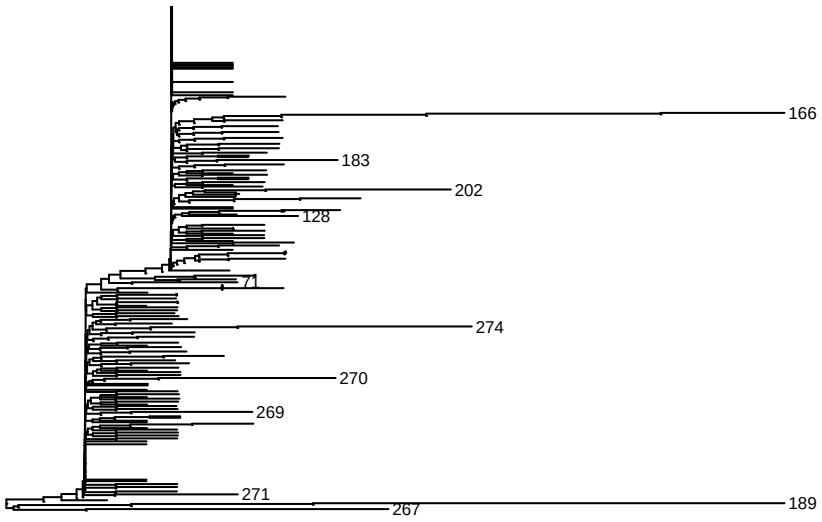
- Save intermediate result and take a deep breath

```
dendrogram_file <- file.path(workdir, "SARS-CoV2-S.newick")
WriteDendrogram(dendrogram, dendrogram_file, quoteLabels = FALSE)
```

- Visualization

```
library(ggtree)
tree <- treeio::read.newick(dendrogram_file)
ggtree(tree) +
```

```
    geom_tiplab(
        aes(subset = branch.length > .001, label = label),
        size = 3
    ) +
    ggplot2::ggtitle("SARS-CoV-2 S-locus phylogeny; genbank, 10 May 2020")
```



Single cell expression

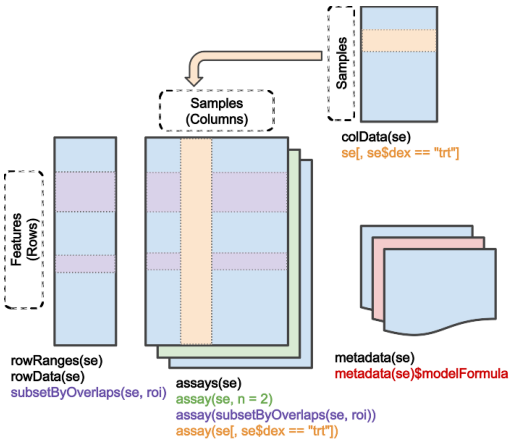
- Retrieve script and file to a local working directory

```
workdir <- "workdir"
if (!dir.exists(workdir))
    dir.create(workdir)

url <- "https://raw.githubusercontent.com/mtmorgan/QuaRantine/master/assets/05-scR
destination <- file.path(workdir, basename(url))
if (!file.exists(destination))
    download.file(url, destination)

url <- "https://github.com/mtmorgan/QuaRantine/raw/master/assets/05-internal_nonsm
h5_file <- file.path(workdir, basename(url))
if (!file.exists(h5_file))
    download.file(url, h5_file)
```

- Input and explore cell annotations and expression values

```
obs <- read_obs(h5_file)       # tibble
x <- read_expression(h5_file) # sparse matrix
```

- Coordinate cell annotation and expression values as `SingleCellExperiment`

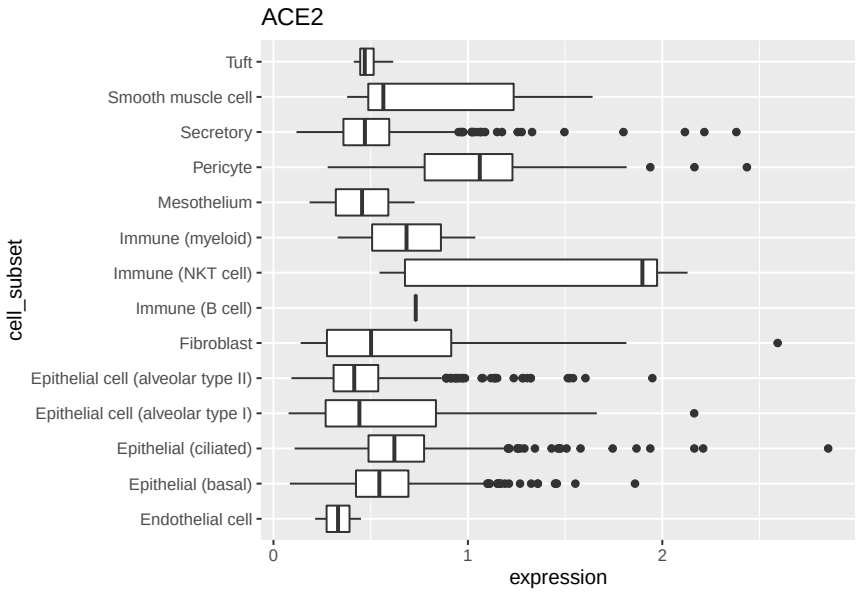



```
sce <- SingleCellExperiment(x, colData = obs)
sce
## class: SingleCellExperiment
## dim: 22 240511
## metadata(0):
## assays(1): ''
## rownames(22): ACE2 TMPRSS2 ... CTSC CTSE
## rowData names(0):
## colnames(240511): 1B_cell57_HU28 1B_cell418_HU28 ...
##   RP2_neg_cell6794209_HU52 RP2_neg_cell6794225_HU52
## colData names(7): Age Cell Type ... Tissue cell_subset
## reducedDimNames(0):
## spikeNames(0):
## altExpNames(0):
```

- Etc., e.g., cell subtype expression

```
library(ggplot2)

ace2 <- subset(sce, rownames(sce) %in% "ACE2")
tbl <- tibble(
    cell_subset = ace2$cell_subset,
    expression = as.vector(assay(ace2))
)

tbl %>%
    filter(expression > 0) %>%
    ggplot(aes(cell_subset, expression)) +
        geom_boxplot() +
        coord_flip() +
```

```
        ggtitle("ACE2")
```



- Many more single cell resources described in Orchestrating Single Cell Analysis with *Bioconductor*.

5.5.3 This weekend (10 minutes)

Saturday

- Tree visualization: color lineages based on observed attributes

Sunday

- Package exploration: identifying solutions for your research challenges.

5.6 Day 34 Revising tree visualization

The goal for today is to return to the Wednesday challenge: to color the S-locus gene phylogeny based on the country the lineage was sequenced in. The steps are as follows:

- Find nodes representing just a single sequence – 'singletons'
- Use GenBank country to identify which country these were sequenced from
- Color lineages according to singleton country of origin

5.7 Day 35 Exploring Bioconductor

The exciting part of science is that we are always on the edge – on our own edge of new discovery, where no one has gone before. We've barely scratched the surface of *Bioconductor* this week. Explore available packages for solutions that might be appropriate for **your particular problem**.

Chapter 6

Collaboration

In this final week of QuaRantine, we focus on communicating reproducible analyses with our peers. We will

- Write 'markdown' vignettes to describe and share our analyses
- Create and document functions that encapsulate common tasks or steps in a work flow
- Combine vignettes and documented functions into an *R* package that can be easily shared with others

Instead of counting upward from the begining of our quarantine, we count down to the end.

6.1 5 Days (Monday) Zoom check-in

6.1.1 Weekend review (5 minutes)

6.1.2 Vignettes (25 minutes; Shawn)

Vignette preparation

- Create a directory `Week-06` in the working directory

```
workdir = "workdir/Week-06"
if (!dir.exists(workdir)) {
    dir.create(workdir, recursive = TRUE)
}
```

Create an *R* markdown document

- `File -> New file -> R markdown...`
- Enter a `Title` and your name as `Author`

- Use `HTML` as default output
- *RStudio* creates a template
- Save it (e.g., click the floppy disk icon) in the `Week-06` folder. Use a title such as `mtcars_regression.Rmd`.

Preview (knit) the markdown template

- Click the `knit` button
- Install the `knitr` package if button is missing

Customize the template

- Let's add a demonstration of `ggplot2` using the `mtcars` data set:

Preview your edits

- Save your changes
- Click the `knit` button

After a bit of processing, the `Rmarkdown` is rendered as an `.html` page. Notice how the *R* code block has been evaluated and resulting console output and plot are included in the `.html`. This is great because we can be sure that the vignette is actually working (e.g. no syntax errors or other coding problems).

There's a related package called `bookdown` that allows you to assemble documents and publish them directly to the web for all to see!

6.1.3 Writing and documenting R functions (25 minutes)

Writing our own functions can be useful in several situations

- Capturing a common operation, such as retieving data from an internet resource.
- Representing a transformation that can be applied to different parts of the data, e.g., calculuating the difference in new cases, applied to different counties or states
- Summarizing overall steps in a work flow, e.g., translating, aligning, and clustering a DNA sequence.

As an example, the following function takes as input a vector representing the cummulative number of observations (e.g., new cases, deaths) over successive days. It calculates the difference in cases, and then uses `stats::filter()` to return the trailing average of the difference

```
trailing_difference <- function(x, n_days = 7) {
    diff <- diff(c(0, x))
    average_weights <- rep(1 / n_days, n_days)
    lag <- stats::filter(diff, average_weights, sides = 1)
```

```
    as.numeric(lag)
}
```

Here's a vector and the result of applying the function

```
obs <- c(1, 2, 4, 6, 7, 12, 14, 18, 19, 20, 20, 21, 22, 24)
trailing_difference(obs, n_days = 4)
##  [1]   NA   NA   NA 1.50 1.50 2.50 2.50 3.00 3.00 2.00 1.50 0.75 0.75 1.00
```

This function could be used, for instance, to calculate the seven-day average number of new cases in each county of the US.

Let's formalize this function, including documentation, in a separate file.

- Use RStudio File -> New Script to create a file `workdir/Week-06/trailing_difference.R`
- Document the function using `roxygen` formatting. This involves placing special comment characters `#'` at the start of lines, and using 'tags' that describe different parts of the function.
  - `@title`: a one-line description of the help page
  - `@description`: a short (paragraph-length?) summary of what capabilities the help page documents
  - `@param`: one for each argument, describing the value (e.g., '`numeric()`' and meaning (e.g., 'vector of observations over successive days')
  - `@return`: the value returned by the function
  - `@examples`: valid *R* code illustrating how the function works.

We use the special tags `@importFrom` to tell *R* that we want to use the `filter` from the stats package, and `@export` to indicate that the function is meant for the 'end user' (i.e., us!).

```
```
#' @title Trailing difference of a vector
#'
#' @description Calculate the difference of successive elements of a
#'   vector, and then the running average of the difference. The
#'   width of the difference can be specied as an argument.
#'
#' @param x numeric() vector of observations.
#'
#' @param n_days scalar (length 1) numeric() number of days used to
#'   calculate the trailing average. The length of `x` should be
#'   greater than `n_days`.
#'
#' @return numeric() vector with the same length of x, representing
```
```

```
#'     the n_day average difference in x. Initial values are `NA`.
#'
#' @examples
#' obs <- c(1, 2, 4, 6, 7, 12, 14, 18, 19, 20, 20, 21, 22, 24)
#' trailing_difference(obs, n_day = 4
#'
#' @importFrom stats filter
#'
#' @export
trailing_difference <- function(x, n_days = 7) {
    diff <- diff(c(0, x))
    average_weights <- rep(1 / n_days, n_days)
    lag <- stats::filter(diff, average_weights, sides = 1)
    as.vector(lag)
}
```

6.1.4 Create an *R* pacakge!

The vignette and documents function are great for our own use, but we'd really like to share these with our colleagues so that they too can benefit from our work. This is very easy to do.

- In RStudio, choose File -> New project -> New directory -> R package
- Enter a package name, e.g., `MyQuarantine`, and use the `add` button to select the vignette `mtcars_regression.Rmd` and *R* `trailing_difference.R` files.
- Choose a location for your package, select the 'Open in a new session' button, and click 'Create project'.

The end result is a directory structure that actually represents an *R* package that you can build and share with your colleagues. The directory contains

- An R/ folder, containing your R source code
- A vignettes/ folder, containing the vignette you wrote.

Later in the week we'll see how to

- Edit the `DESCRIPTION` file to describe your package
- Generate help pages in the `man/` folder from the roxygen comments in the `.R` files.
- Create the knit vignette from the source .Rmd file.
- Build the package for distribution and sharing with others.

6.2 4 Days Write a vignette!

Choose one week from the quarantine, and write a vignette summarizing the material. Start with an outline using using level 1 `#` and level 2 `##` headings as well as bulleted lists / short paragraphs. One could think of this as structured more-or-less along the lines of classic scientific paper, with introductino, methods, results (use case), and discussion

````
```
# Introduction to tidy data

# Tools for working with tidy data

## Key packages

- dplyr
- ggplot2

## 5 Essential functions

- `mutate()`
- `filter()`
- `select()`
- `group_by()` / `ungroup()`

# Use case

E.g., summarizing and visualizing cell subtypes in Week 5.

- narrative text describing what steps are being taken
- include code chunks for reproducible analysis
- include figures and / or summary tables to help communicate your results

# Discussion

A paragraph on strengths and limitations of the tidy approach

Narrative on use case / insights

Future directions

## Session information

- include a code chunk that has the command `sessionInfo()`; this
  documents  the specfici versions of packages you used.
```
````

6.3 3 Days Create documented, reusable functions!

Create functions that represents key operations (e.g., data retrieval), data transformations (e.g., trailing difference in new cases), or that integrate several related steps in an analysis (e.g., `translate()`, `unique()`, align, and cluster DNA sequences).

Place the function(s) in separate files (one or several functions per file). Document them using the notation introduced on Monday.

Make sure the functions work by writing simple examples.

6.4 2 Days Share your work as a package!

Use the steps outlined on Monday to create an *R* package from Tuesday's vignette and Wednesday's functions.

6.4.1 DESCRIPTION

Edit the DESCRIPTION file to include a Title and Description. Update the Author infromation to include your name. Add as maintainer your name and email address, using the format `Ima Maintainer <my@email.com>`.

6.4.2 Documentation

Make sure that `getwd()` returns the path to the package. Run the command (it may be necessary to install additional packages.

```
getwd() # in the directory of the project; use `setwd()` to change
devtools::document()
```

There may be problems with the roxygen that you wrote; investigate how to fix these.

This creates file(s) in the `man/` directory that transform the 'roxygen' comments (lines with `#'`) in the R/ file. Open one of the man files and use the 'preview' button to see the help page.

6.4.3 Vignettes

Add the following lines to the end of the DESCRIPTION file:

```
Suggests: knitr
VignetteBuilder: knitr
```

Update the 'yaml' at the top of the vignette

```
---
title: "Multiple regression plots"
author: "Shawn Matott"
date: "5/13/2020"
output: html_document
vignette: |
    %\VignetteIndexEntry{ Multiple Regression Plots }
    %\VignetteEngine{knitr::rmarkdown}
    %\VignetteEncoding{UTF-8}
---
```

Use the following command to build the vignette

```
getwd() # in the directory of the project
devtools::build_vignettes()
```

6.4.4 Build and share

With all the pieces now in place, choose Build -> Build Source Package. This creates a single file with a name like `MyQuarantine_0.1.0.tar.gz` that you can share with colleagues – use

```
install.packages("path/to/MyQuarantine_0.1.0.tar.gz", repos = NULL)
```

to install your pacakge!

6.5 Today! (Friday) Zoom check-in

6.5.1 Review and troubleshoot

Vignettes

- Any vignettes to share?
- Shruti's QuaRantine Learnings

Documented functions

Packages

6.5.2 Course review

Basic *R*

- Numeric, character, logical vectors
- subsetting, applying functions
- The data.frame

'Tidy' *R* and visualization

- The tibble and pipe (`%>%`)

- readr: `read_csv()`
- dplyr: `mutate()`, `filter()`, `select()`, `group_by()`, `summarize()`
- ggplot2
- Visualizing the pandemic locally and globally

Machine learning

- Underlying concepts
- Support vector machines, k-nearest neighbors, KNN
- Accuracy and confusion matrix; ROC curves and AUC

Bioinformatics

- *Bioconductor* packages
- Classes, generics, and methods for representing sequences and ranges
- Virus phylogeny
- Host gene expression

Reproducible communication

- Vignettes
- Documented functions
- Packages

6.5.3 Feedback & next steps

Acknowledgments

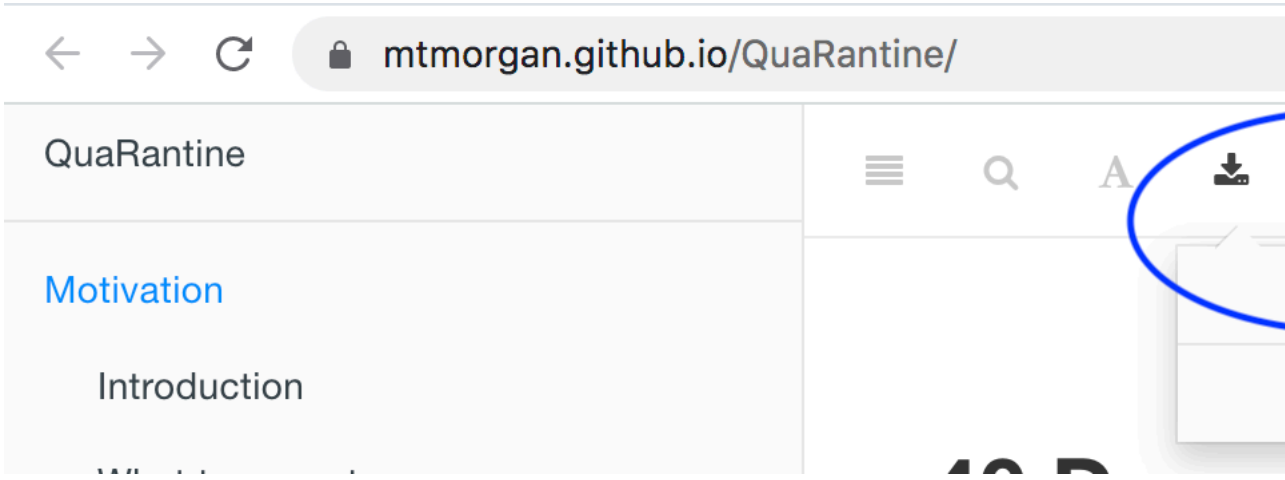
Research reported in this publication was supported by the Human Genome Research Institute and the National Cancer Institute Information Technology for Cancer Research of the National Institutes of Health under award numbers U41HG004059 and U24CA180996. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

Frequently asked questions

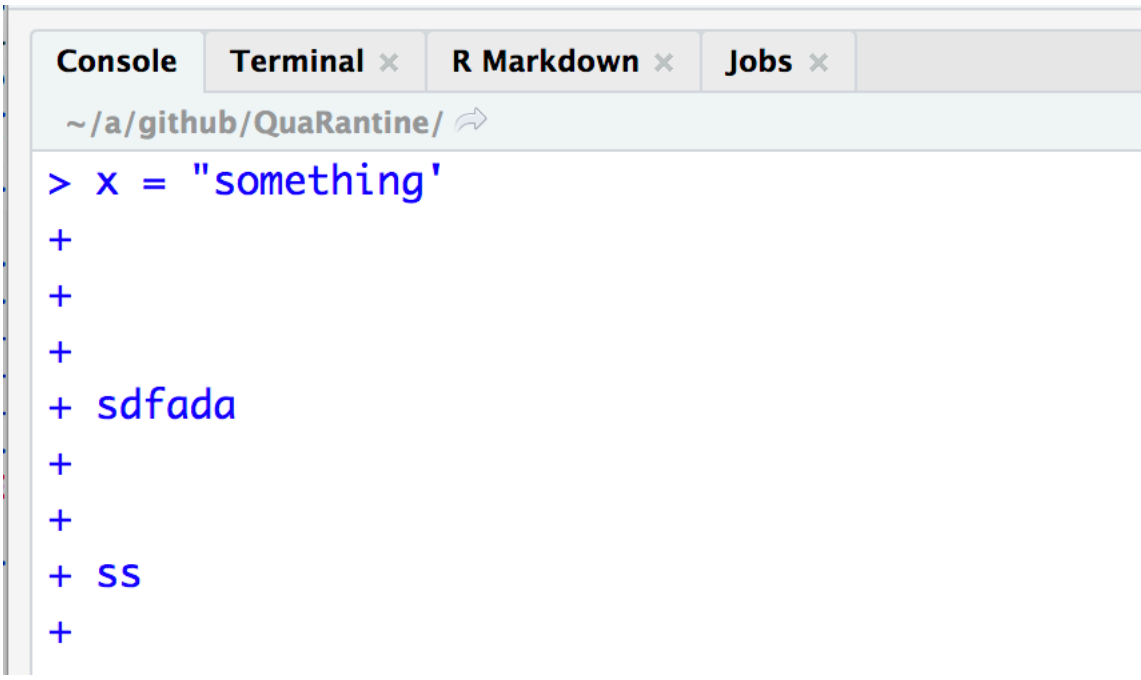
1. Is the course material available in PDF?

   Yes, click the 'Download' icon and PDF format in the title bar of the main document, as illustrated in the figure.

   Remember that the course material is a 'work in progress', so the PDF will need to be updated frequently throughout the course. Also, the book is not pretty; that's a task for a separate quarantine!

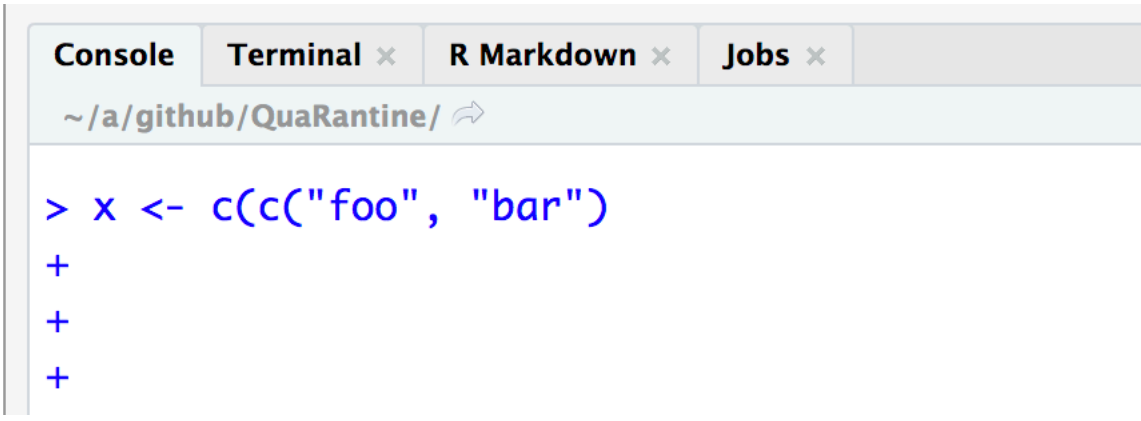


2. Whenever I press the 'enter' key, the RStudio console keeps saying + and doesn't evaluate my expression! See the figure below.



Notice that you've started a character string with a double `"`, and tried to terminate it with a single quote `'`. Because the quotes do not match, *R* thinks you're still trying to complete the entry of the variable, and it's letting you know that it is expecting more with the `+` prompt at the begining of the line.

A common variant of this is to open more parentheses than you close, as shown in



The solution is either to complete your entry (by entering a `"` or balancing the parentheses with `)`) or abandon your attempt by pressing `control-C`

or the escape key (usually in the top left corner of the keyboard)

3. Should I save scripts, individual objects (`saveRDS()`) or multiple objects / the entire workspace (`save()`, `save.image()`, `quit(save = "yes")`)?

   Reproducible research requires that one knows *exactly* how data was transformed, so writing and saving a script should be considered an essential 'best practice'.

   A typical script starts with some data generated by some third-party process, e.g., by entry into a spreadsheet or generated by an experiment. Often it makes sense to transform this through a series of steps to a natural 'way-point'. As a final step in the script, it might make sense to save the transformed object (e.g., a data.frame) using `saveRDS()`, but making sure that the file name is unambiguous, e.g., matching the name of the object in the script, and with a creation date stamp.

   I can't really imagine a situation when it would be good to use `save.image()` or `quit(save = "yes")` – I'll just end up with a bunch of objects whose content and provenance are completely forgotten in the mists of time (e.g., since yesterday!).

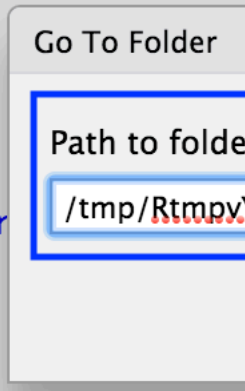
4. Where does RStudio create temporary files?

   The screenshot below shows that the *R* session seems to have created a temporary file path, and it seems like it's possible to `write.csv()` / `read.csv()` to that file (no errors in the blue square box at the bottom left!) but the file doesn't show up in the file widget (circle in `Files` tab on the right.

```
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> tempfile()
[1] "/tmp/RtmpvYph7S/filef625012737"
> activity <- c("check e-mail", "breakfast", "conference ca
k")
> minutes <- c(20, 30, 60, 60, 60)
> is_work <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
>
> levels <- c("connect", "exercise", "consult", "hobby", "e
> classification <- factor(
+     c("connect", "essential", "connect", "consult", "exer
+     levels = levels
+ )
>
> dates <- rep("04-14-2020", length(activity))
> date <- as.Date(dates, format = "%m-%d-%Y")
> activities <- data.frame(
+     activity, minutes, is_work, classification, date,
+     stringsAsFactors = FALSE
+ )
> temporary_file_path <- tempfile(fileext = ".csv")
> temporary_file_path
[1] "/tmp/RtmpvYph7S/filef6636094e.csv"
> write.csv(activities, temporary_file_path, row.names = FA
> imported_activities = read.csv(temporary_file_path, strin
```

The file widget is pointing to a particular directory; what you'd like to do is navigate to the directory where the temporary file is created. Do this by clicking on the three dots `...` (blue circle) in the `Files` tab, and enter the directory part of the the temporary file path (blue squares).

```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> tempfile()
[1] "/tmp/RtmpvYph7S/filef625012737"
> activity <- c("check e-mail", "breakfast", "conferer
k")
> minutes <- c(20, 30, 60, 60, 60)
> is_work <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
>
> levels <- c("connect", "exercise", "consult", "hobby", "essentia
> classification <- factor(
+     c("connect", "essential", "connect", "consult", "exercise"),
+     levels = levels
+ )
>
> dates <- rep("04-14-2020", length(activity))
> date <- as.Date(dates, format = "%m-%d-%Y")
> activities <- data.frame(
+     activity, minutes, is_work, classification, date,
+     stringsAsFactors = FALSE
+ )
> temporary_file_path <- tempfile(fileext = ".csv")
> temporary_file_path
[1] "/tmp/RtmpvYph7S/filef6636094e.csv"
> write.csv(activities, temporary_file_path, row.names = FALSE)
> imported_activities = read.csv(temporary_file_path, stringsAsFac
>
```



Once the file widget is pointed to the correct location, the file (last part of the `temporary_file_path`) appears…

```
Type 'demo()' for some demos, 'help()' for on-line help,
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> tempfile()
[1] "/tmp/RtmpvYph7S/filef625012737"
> activity <- c("check e-mail", "breakfast", "conference
k")
> minutes <- c(20, 30, 60, 60, 60)
> is_work <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
>
> levels <- c("connect", "exercise", "consult", "hobby",
> classification <- factor(
+     c("connect", "essential", "connect", "consult", "ex
+     levels = levels
+ )
>
> dates <- rep("04-14-2020", length(activity))
> date <- as.Date(dates, format = "%m-%d-%Y")
> activities <- data.frame(
+     activity, minutes, is_work, classification, date,
+     stringsAsFactors = FALSE
+ )
> temporary_file_path <- tempfile(fileext = ".csv")
> temporary_file_path
[1] "/tmp/RtmpvYph7S/filef6636094e.csv"
> write.csv(activities, temporary_file_path, row.names =
> imported_activities = read.csv(temporary_file_path, str
>
```

Navigate back to the original directory by clicking on the three dots `...` in the `Files` tab and enter `/cloud/project`.

Remember that the temporary file path is, well, temporary, and when you

start a new *R* session (or maybe restart your cloud session) the temporary path and anything saved there may no longer be available

5. When I try to install a package, *R* says a binary version exists but a newer version exists 'from source'. Do I want to install the binary version or the source version?

   Usually, the answer is to install the older binary version.

   The newer source version likely represents a bug fix or implements new features in the package, so at first blush it seems preferable. However, installing source versions of packages often require additional software that is not necessarily easy to install and maintain. You might spend a considerable amount of time configuring your system and working through arcane error messages to get the 'latest' source version.

   Almost always, the updated binary version becomes available in a day or so, and the best strategy is to install the old binary version now, and re-install the package the next time you remember, in a week or so.

6. What is the difference between quotes `"` (double quote), `'` (single quote), `` ` `` (backtick)

   `"` and `'` can be used interchangeably (but always matching open and closing quotes) to represent and element of a character vector: `"DNA"` and `'DNA'` are the same. It can be convenient to use one style over another in particular situations: `"3' DNA"`, `'Tolstoy says: "All happy families are alike; each unhappy family is unhappy in its own way."'`, where the choice of quotes means that the 'inner' quote does not have to be 'escaped', e.g., using single quotes in the first example: `'3\' DNA'`.

   The backtick `` ` `` is a special quote, and says to treat the enclosed expression as a symbol *name*. We came across one use case in the world-wide COVID-19 data, where the column names was `Region/Country`. If we wanted to create a variable with that name, we could NOT write `Region/Country = c("US", "Canada")` because *R* would interpret `Region` and `Country` as variables that were to be divided. Likewise, we couldn't use `"Region/Country" = c("US", "Canada")` because *R* would say 'hey, you're trying to assign a character vector to a character vector not to a symbol!'.

```
`Region/Country` = c("US", "Canada")
```

   says 'create a symbol `Region/Country` and assign it the value `c("US", "Canada")`'. Any time we want to use this variable, we have to quote it

```
length(`Region/Country`)
## [1] 2
```