TDU.

Introduction to R for Public Health Investigations

Workbook for Day 1

Public Health Agency of Canada   Agence de la santé publique du Canada

Canada

Contents

PRACTICAL EXERCISE

Instructions

Duration: 3 hours 10 minutes

Learners are provided with a scenario, questions, and tasks with associated code to perform each task. We recommend:

Novice users (Boatswains): Use this workbook and R script(s) provided on GitHub. Using this workbook as a guide, run the code we've provided piece by piece to understand what each chunk of code does, and what various functions are doing. At this point don't worry too much about being able to write or debug code.

Beginner/Intermediate users (First Mates): R code is provided as a screen capture image in this workbook. You should have sufficient understanding of coding to get a general sense of what the code is doing by reading it (with the assistance of the help documentation, a few Google searches as needed, and comments in the R scripts we've provided on GitHub). It is our intention to have you write the code out from the guide as you progress through the scenario. Cross reference to the R script(s) provided on GitHub if you encounter any tangly problems.

Advanced users (Master Mariners): We encourage you to try writing your own code where you like and contrast it with the code used for the exercise, and to help your peers as questions arise. Cross reference to the R script(s) provided on GitHub if you encounter any tangly problems.

Get as far as you can with this exercise within 2 hours and 10 mins (maximum). Don't worry if you need extra time. The learning curve for R is steep and learners will benefit most from dedicated time for practicing. Reach out to your course facilitators by Slack or by email if you require assistance with the course material.

Introduction

It is the beginning of December, 2020. You are placed in the Office of the Provincial Health Officer at the British Columbia Ministry of Health. You and your public health colleagues are busy responding to the COVID-19 pandemic and developing an understanding of the occurrence of COVID-19 in the province. One of the items requested by the Provincial Health Officer for her situational awareness is a regularly updated analysis of COVID-19 in other Canadian jurisdictions. This is an additional task added to your already heavy workload.

1. Would you consider automating this analysis? What factors would you use to make this decision?

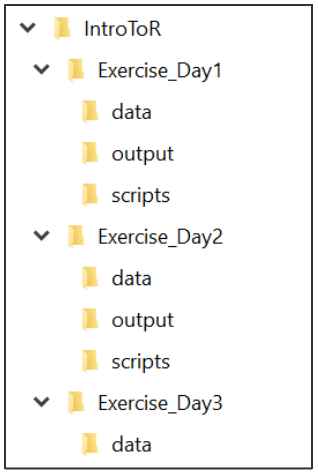
2. What pieces of information would you include in this update?

3. Where would you find the data you need to conduct this analysis? Hint: Think of sources that are already publicly available.

After considering your options you decide that it makes sense for you to automate this task. It has also been settled upon that your portion of this update will include an appendix of figures: epidemic curves for cases and deaths for each province. You've also decided that you will use the Government of Canada's open source COVID-19 line list along with population estimates from Statistics Canada. It is your job to produce the figures and share them with the team compiling information for the Provincial Health Officer.

Setting up your workspace

A critical step before planning or executing an analysis in R (or any software language) is setting up an efficient workspace. In this course, we'll ask you to follow along with the example folder structure that we've created in order to minimize conflicts in the code when accessing data files or sourcing scripts. In your own projects in the future, however, feel free to experiment with what works best for how you like to code!

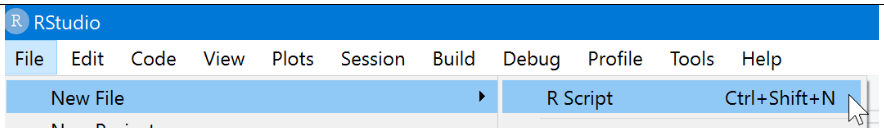


By the end of the three days in this course, your folders should look something like the image on the right. Note that we haven't explicitly stated *where* you should keep your parent folder, '**IntroToR**'. This is because we'll be using *.Rproj* ("R Project") files and the *here()* package to facilitate *relevant* file paths instead of *absolute* file paths. There are pros and cons to each of these approaches that we'll discuss during the course, but when sharing R code widely with users who may have very different computer set-ups than your own, setting up your code with relative file paths will typically save a lot of work down the line.

Start by setting up your workspace. Create a file structure on your computer that will organize all the files for each exercise (e.g., **scripts**, **data**, **output**).

1. Create a new folder on your computer called "IntroToR".
2. Within the IntroToR folder, create a subfolder for Day 1 of the course called: "Exercise_Day1".
3. Create new folders within Exercise_Day1 called: "output"; "data"; "scripts".
4. Move the contents of the data folder for Day 1 from GitHub to the new "IntroToR/Exercise_Day1/data" folder. (If you wish to, you may also copy over the files located in the "scripts" folder on GitHub into your script folder, however, we encourage you to first practice writing these scripts yourself, and retain the pre-written ones for reference).
5. Once your folder structure is set-up, open RStudio and go to File > New Project.
6. On the next window, select "Existing Directory" and click "Browse" to navigate to your **IntroToR** folder. With the IntrotoR folder selected, click "Create Project." A new file, "*IntroToR.Rproj*" should appear in the parent directory of your workspace.

Now that you have set up your project folders, you can get started on writing the scripts to be used in your analysis.

Task	Code
With RStudio open, ensure that you are working within your new R-Project environment by either going to **File > Open Project** and navigating to and opening your IntroToR.Rproj file OR opening/selecting your IntroToR.proj file by clicking on the R project icon in the upper right corner of the RStudio interface and navigating through the dropdown menu that will appear.	
Open RStudio and start a new R script by navigating to File > New File > R Script or using the keyboard shortcut Ctrl+Shift+N. Save this script in your “Exercise_Day1/scripts' folder as “01_define_paths.R”.	 RStudio File Edit Code View Plots Session Build Debug Profile Tools Help New File R Script Ctrl+Shift+N
Install the here package	`install.packages("here")`
Write a statement to create an object that will direct R to the location of your folder where your raw data are saved. Execute the statement. Explain in your own words what the line of code to the right is doing.	`data_folder <- here::here("Exercise_Day1", "data")`
Write a statement to create an object that will direct R to the folder where any figures or data cuts you create will be saved. Execute the statement.	`output_folder <- here::here("Exercise_Day1", "output")`
Write a statement to create an object that will direct R to the folder where you will save all of your R scripts associated with this project. Execute the statement.	`scripts_folder <- here::here("Exercise_Day1", "scripts")`
Ensure a heading is included in your script with details regarding the purpose, author, date, modifications, and other pertinent notes.	

Save your script as “01_define_paths.R”	

4. Load the libraries you will need for this project:

Task	**Code**	**Library info – for reference**
Start a new R script by navigating to File > New File > R Script or using the keyboard shortcut Ctrl+Shift+N. Save this script as “02_load_libraries.R” in your scripts folder.		

(Optional) Install packages that aren’t already installed in RStudio on your computer. Note that this only needs to be done once, and if you have already installed these packages in the pre-course learning, you do not need to do this now.	`install.packages("here")`
	`install.packages("tidyverse")`
	`install.packages("lubridate")`
	`install.packages("scales")`
	`install.packages("padr")`
	`install.packages("writexl")`
	`install.packages("fs")`
	`install.packages("RColorBrewer")`
	`install.packages("ggrepel")`
	`install.packages("ggpubr")`
	`install.packages("zoo")`
	`install.packages("viridis")`
		here: https://here.r-lib.org/
		
		readr: https://readr.tidyverse.org/
		
		readxl: https://readxl.tidyverse.org/
		
		tidyverse: https://www.tidyverse.org/
		
		scales: https://scales.r-lib.org/

Load the libraries (installed packages) you will need for your project every time you will be using them. Packages that are loaded will be checked off in the bottom right window under the Packages tab.	`library(here)`
	`library(tidyverse)`
	`library(lubridate)`
	`library(scales)`
	`library(padr)`
	`library(writexl)`
	`library(fs)`
	`library(RColorBrewer)`
	`library(ggrepel)`
	`library(ggpubr)`
	`library(zoo)`
	`library(viridis)`
		padr: https://www.rdocumentation.org/packages/padr/versions/0.5.3
		
		writexl: https://cran.r-project.org/web/packages/writexl/index.html
		
		fs: https://fs.r-lib.org/
		
		RColorBrewer: https://www.rdocumentation.org/packages/RColorBrewer/versions/1.1-2/topics/RColorBrewer
		
		zoo: https://www.rdocumentation.org/packages/zoo/versions/1.8-8
		
		viridis: https://www.rdocumentation.org/packages/viridis/versions/0.3.4

Ensure a heading is included in your script with details regarding the purpose, author, date, modifications, and other pertinent notes.		
Save your script as 02_load_libraries.R		

5. You've accessed a simplified version of the open access PHAC COVID-19 line list and saved it into your project folder. Similarly, you've identified a source of denominator data you'd like to use. As you do not need to work with this dataset prior to loading it into R, you've decided to read it in directly from the Statistics Canada website.

Task	Code
Start a new R script by navigating to File > New File > R Script or using the keyboard shortcut Ctrl+Shift+N and save it in your scripts folder as 03_import_data.R	
Explain in your own words what the line of code to the right is doing. Read the COVID-19 line list .csv file into RStudio.	`phac_raw <- read_csv(here("Exercise_Day1", "data", "covid19.csv"))`

Create a new character vector X and assign to it the web location of the .csv file containing the Statistics Canada denominator using the link below in the place of "WEB ADDRESS" in the code to the right: https://www150.statcan.gc.ca/t1/tbl1/en/dtl!downloadDbLoadingData.action?pid=1710000901&latestN=0&startDate=20200101&endDate=20200101&csvLocale=en&selectedMembers=%5B%5B2%2C3%2C4%2C5%2C6%2C7%2C8%2C9%2C10%2C11%2C12%2C14%2C15%5D%5D&checkedLevels=
 	
 What happens when you click the above web address?
	`x <- "WEB ADDRESS"`
	
	 `canada_pop_denom_raw <- read_csv(url(x))`

Explain in your own words what the second line of code to the right is doing. Read the population data .csv file into RStudio.	
Ensure a heading is included in your script with details regarding the purpose, author, date, modifications, and other pertinent notes.	
Save your script again as 03_import_data.R	

6. Now that the data are loaded into RStudio, you must clean and process it. First, let's clean the denominator (population) data:

Task	**Code**
Start a new R script by navigating to File > New File > R Script or using the keyboard shortcut Ctrl+Shift+N and save it in your scripts folder as 04_1_clean_population_denominator.R	

Create a vector consisting of province names and abbreviations for those names. What is the data type stored in this vector? What is it's length? What does the following command return: `provinces_short_names[5]` ?	`provinces_short_names <- c("Alberta" = "AB",`
	`"British Columbia" = "BC",`
	`"Manitoba" = "MB",`
	`"New Brunswick" = "NB",`
	`"Newfoundland and Labrador" = "NL",`
	`"Nova Scotia" = "NS",`
	`"Northwest Territories" = "NWT",`
	`"Ontario" = "ON",`
	`"Prince Edward Island" = "PEI",`
	`"Quebec" = "QC",`
	`"Saskatchewan" = "SK",`
	`"Yukon" = "YK",`
	`"Nunavut" = "NU")`

Create a new object by manipulating an existing object. Use the columns GEO and VALUE from Canada_pop_denom_raw to create a new dataset called pop_denom_canada_clean. Rename the variables so that province= GEO and PopTotal=Value. Then, using the mutate function, replace the full province names with the abbreviated names you previously created.	`pop_denom_canada_clean <- canada_pop_denom_raw %>%`
	`  select(province = GEO, PopTotal = VALUE) %>%`
	`  mutate(province = str_replace_all(province, provinces_short_names))`

Ensure a heading is included in your script with details regarding the purpose, author, date, modifications, and other pertinent notes.	
Save your script as 04_1_clean_population_denominator.R	

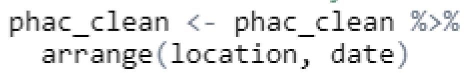
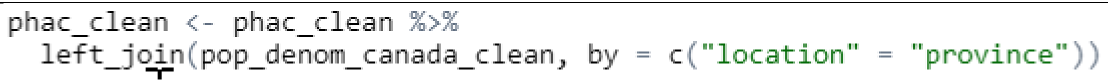
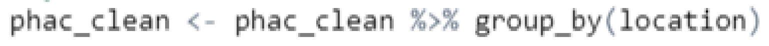
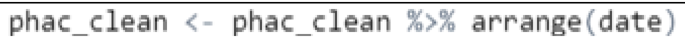
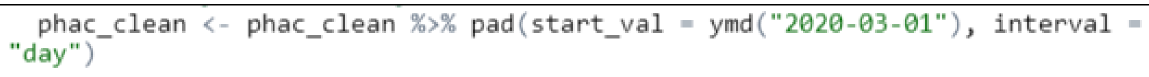
7. And now the numerator (line list) data:

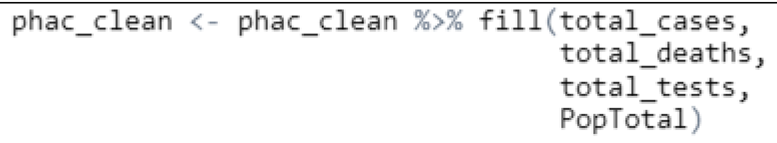
Task	**Code**
Start a new R script by navigating to File > New File > R Script or using the keyboard shortcut Ctrl+Shift+N and save it as “04_2_clean_phac_case_data.R”	

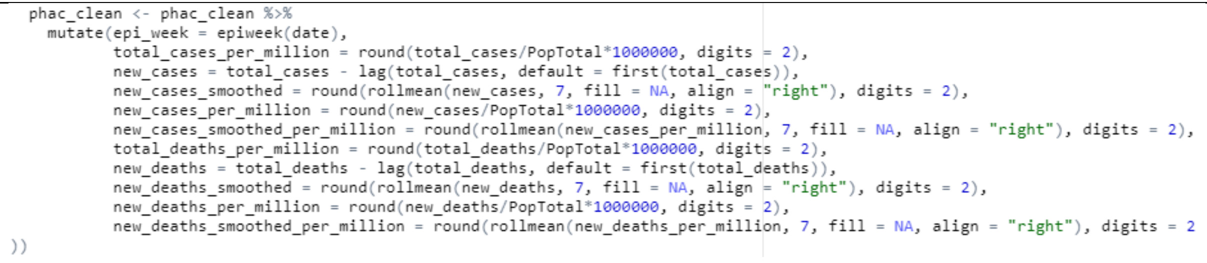
Create a new data frame based on raw line list data. This data frame will contain the variables prname, date, numconf, numdeaths, numtested. However, prname will be renamed as location, numconf as total_cases, numdeaths as total_deaths, and numtested as total_tests.	`phac_clean <- phac_raw %>%`
	`  select(location = prname,`
	`         date, total_cases = numconf,`
	`         total_deaths = numdeaths,`
	`         total_tests = numtested)`

Recode the character variable containing date information to a date variable and replace province names with abbreviations (as in previous step). (Hint: Use the mutate, dmy and the str_replace_all functions)	`phac_clean <- phac_clean %>%`
	`  mutate(date = dmy(date),`
	`       location = str_replace_all(location, provinces_short_names))`

Filter data to 1) keep records only with dates on or after March 1, 2020, and 2) exclude repatriated travellers. Hint: Use the operators “>=” "&", "!", and "%in%" to create a logic statement.	`phac_clean <- phac_clean %>%`
	`  filter(date >= ymd("2020-03-01") &`
	`         !location %in% c("Repatriated travellers"))`

Sort the data frame by location and date (Hint: Use the arrange function)	 `phac_clean <- phac_clean %>%` `arrange(location, date)`
Merge the numerator and denominator data by province (Hint: Use the left-join function)	 `phac_clean <- phac_clean %>%` `left_join(pop_denom_canada_clean, by = c("location" = "province"))`
Not every province has a row for every date. In order to add a row for every date and every province, you need to first group your data by province (hint: Use the group_by function)	 `phac_clean <- phac_clean %>% group_by(location)`
Now that your data is grouped by province you can sort the data by date. (Hint: Use the arrange function)	 `phac_clean <- phac_clean %>% arrange(date)`
Autocomplete the missing dates starting on March 1, 2020. Can you explain precisely what the pad function did to the data?	 `phac_clean <- phac_clean %>% pad(start_val = ymd("2020-03-01"), interval = "day")`

The newly inserted days are missing values for the other columns. Repeat the count values (i.e. for total_cases, total_deaths, total_tests and PopTotal) on newly inserted days (Hint: Use the fill function) Can you explain precisely what the fill function did to the dataset?	
	`phac_clean <- phac_clean %>% fill(total_cases,`
	`                                  total_deaths,`
	`                                  total_tests,`
	`                                  PopTotal)`

Calculate:
 	
1) Epiweek
2) New cases
3) Seven day rolling average of new cases
4) Rate of new cases per day
5) Seven day rolling average of rate of new cases
6) New deaths
7) Seven day rolling average of new deaths
8) Rate of new deaths per day
9) Seven day rolling average of rate of new deaths
	
	`phac_clean <- phac_clean %>%`
	`  mutate(epi_week = epiweek(date),`
	`         total_cases_per_million = round(total_cases/PopTotal*1000000, digits = 2),`
	`         new_cases = total_cases - lag(total_cases, default = first(total_cases)),`
	`         new_cases_smoothed = round(rollmean(new_cases, 7, fill = NA, align = "right"), digits = 2),`
	`         new_cases_per_million = round(new_cases/PopTotal*1000000, digits = 2),`
	`         new_cases_smoothed_per_million = round(rollmean(new_cases_per_million, 7, fill = NA, align = "right"), digits = 2),`
	`         total_deaths_per_million = round(total_deaths/PopTotal*1000000, digits = 2),`
	`         new_deaths = total_deaths - lag(total_deaths, default = first(total_deaths)),`
	`         new_deaths_smoothed = round(rollmean(new_deaths, 7, fill = NA, align = "right"), digits = 2),`
	`         new_deaths_per_million = round(new_deaths/PopTotal*1000000, digits = 2),`
	`         new_deaths_smoothed_per_million = round(rollmean(new_deaths_per_million, 7, fill = NA, align = "right"), digits = 2`
	`))`

(Hint: use the mutate, epiweek, lag, round, and rollmean functions)	
Ungroup the data	`phac_clean <- ungroup(phac_clean)`

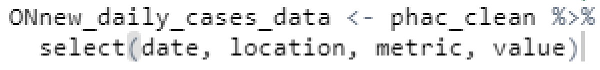
Transpose the data: describe what the pivot_longer() function has done to the data.
 	
Why would we want to transpose the data like this?
	`phac_clean <- phac_clean %>%`
	`  pivot_longer(3:17, names_to = "metric", values_to = "value")`

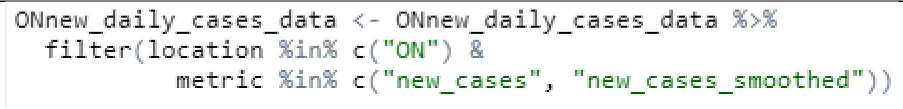
Create three new variables:
 	
1) Cases_deaths to flag whether a numeric value is cases or deaths
2) Count_rate to flag whether a numeric value is a count or a rate
3) Raw_smoothed to flag whether a numeric value is crude (raw) or smoothed
	`phac_clean <- phac_clean %>%`
	`  mutate(cases_deaths = case_when(str_detect(metric, "cases") ~ "cases",`
	`                                  str_detect(metric, "deaths") ~ "deaths",`
	`                                  TRUE ~ "other"),`
	`         count_rate = case_when(str_detect(metric, "million|thousand") ~ "rate",`
	`                                str_detect(metric, "population|positive|positivity") ~ NA_character_,`
	`                                TRUE ~ "count"),`
	`         raw_smoothed = case_when(str_detect(metric, "smoothed") ~ "smoothed",`
	`                                  TRUE ~ "raw"))`

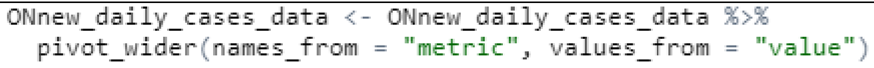
Ensure a heading is included in your script with details regarding the purpose, author, date,	

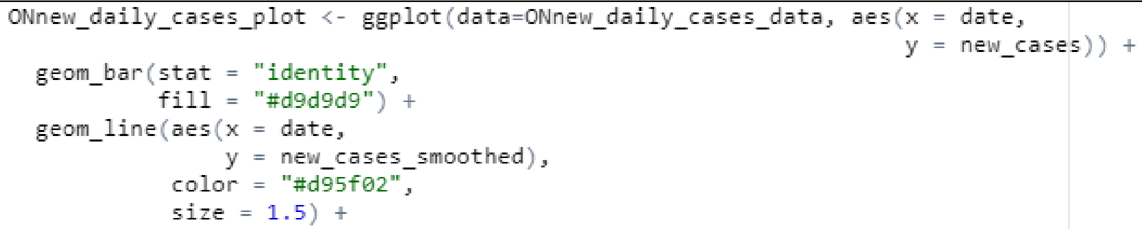
modifications, and other pertinent notes.	
Save your script again as 04_2_ clean_phac_case_data.R	

8. Now that the data are prepared, let's create an epicurve of new ON cases by day and include a seven-day moving average as our first figure:

Task	**Code**
Start a new R script by navigating to File > New File > R Script or using the keyboard shortcut Ctrl+Shift+N and save it in your scripts folder as "05_1_plot_epi_curve.R"	
Create a new dataframe that contains only Ontario data to be graphed and include the following variables: date, location, metric, and value (Hint: Use the select function)	 `ONnew_daily_cases_data <- phac_clean %>%` `  select(date, location, metric, value)`

Filter the data by location and desired metrics to	 `ONnew_daily_cases_data <- ONnew_daily_cases_data %>%`
	`  filter(location %in% c("ON") &`
	`          metric %in% c("new_cases", "new_cases_smoothed"))`

graph (i.e. new_ cases and new_cases smooth)	
Reshape the data to be plotted: What does the pivot_wider() function do to the data?	 `ONnew_daily_cases_data <- ONnew_daily_cases_data %>%` `  pivot_wider(names_from = "metric", values_from = "value")`

Create a new object for graphing the data. What is the purpose of the geom_bar() and geom_line() function?
 	
Helpful tip: If you want to plot your graph and see what the code is doing just leave out the last "+ and include a line at the end with: ONnew_daily_cases_plot. This allows you to see how each section of code is changing your graph.
	 `ONnew_daily_cases_plot <- ggplot(data=ONnew_daily_cases_data, aes(x = date,`
	`                                                                    y = new_cases)) +`
	`  geom_bar(stat = "identity",`
	`           fill = "#d9d9d9") +`
	`  geom_line(aes(x = date,`
	`                y = new_cases_smoothed),`
	`            color = "#d95f02",`
	`            size = 1.5) +`

What is the purpose of the geom_point() function?	`geom_point(data = phac_clean %>%`
	`select(date, location, metric, value) %>%`
	`filter(location %in% c("ON") &`
	`metric %in% c("new_cases_smoothed")) %>%`
	`pivot_wider(names_from = "metric", values_from = "value") %>%`
	`filter(date == max(date)),`
	`aes(x = date, y = new_cases_smoothed),`
	`size = 3,`
	`color = "#d95f02") +`

Set the theme, in this case minimal (i.e., no background annotations)	`theme_minimal() +`

Modify the theme by specifying text size and location, and removing background lines	`theme(axis.text.x = element_text(angle = 0, hjust = 0.5, size = 10),`
	`axis.text.y = element_text(size = 10),`
	`plot.title = element_text(size = 14),`
	`axis.title.x = element_text(size = 12, margin = margin(t = 20, r = 0, b = 0, l = 0)),`
	`axis.title.y = element_text(size = 12, margin = margin(t = 0, r = 20, b = 0, l = 0)),`
	`plot.caption = element_text(size = 12, margin = margin(t = 20, r = 0, b = 0, l = 0)),`
	`legend.position = "none",`
	`panel.grid.minor = element_blank(),`
	`panel.grid.minor.x = element_blank(),`
	`panel.grid.major.x = element_blank()) +`

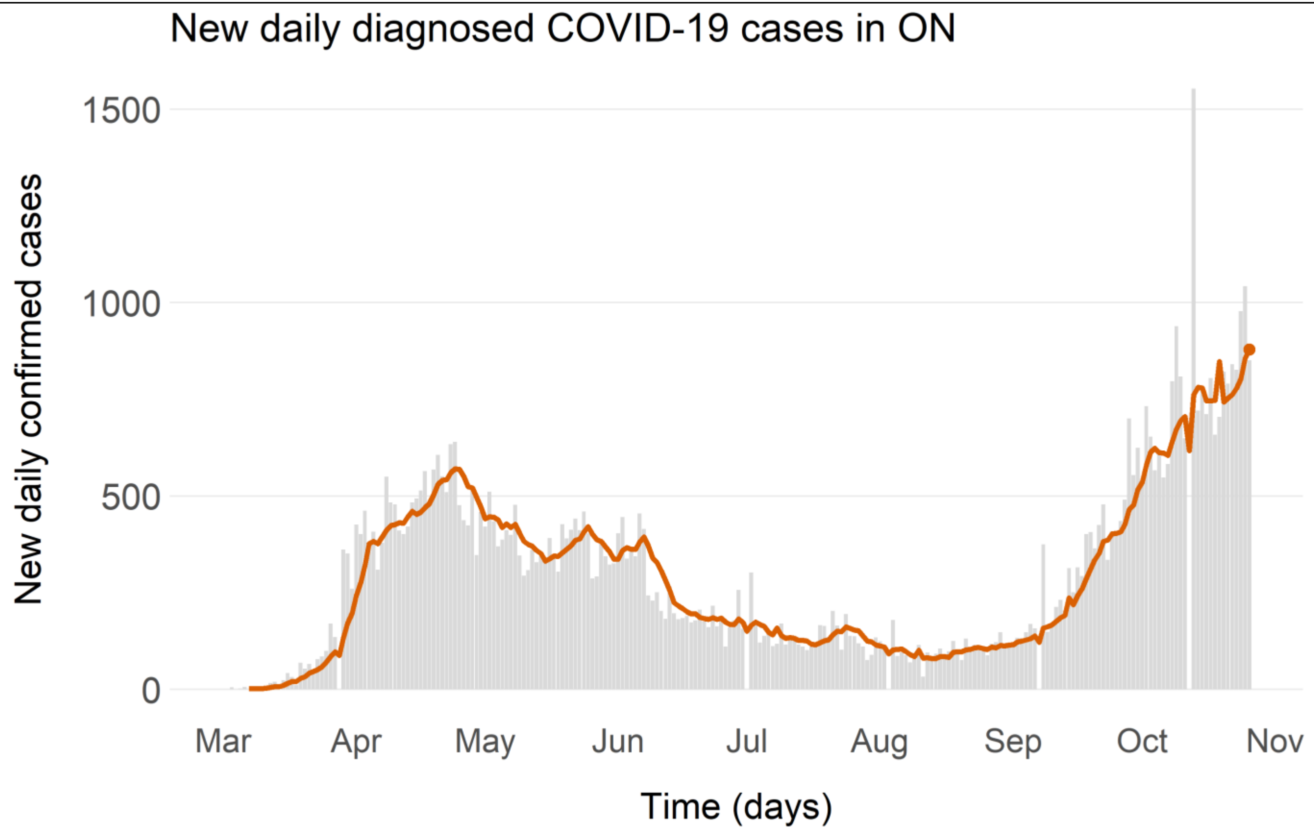
Set the scale of the figure to one month and add abbreviated month labels.	`scale_x_date(date_breaks = "1 months",`
	`date_labels = "%b") +`

Set titles
Graph title: New daily diagnosed COVID-19 cases in ON
X-axis: Time (days)
Y-axis: New daily confirmed cases
	`ggtitle("New daily diagnosed COVID-19 cases in ON") +`
	`xlab("Time (days)") +`
	`ylab("New daily confirmed cases")`

Plot the figure	`ONnew_daily_cases_plot`

Save the figure as a picture (png file) to your output folder using “ggsave”. Name it “ONnew_daily_cases_plot and automatically add todays date to the name.	`ggsave(paste0(output_folder, "/ONnew_daily_cases_plot_", today, ".png"),`
	`       plot = ONnew_daily_cases_plot,`
	`       width = 11,`
	`       height = 7,`
	`       units = "in")`

Oh no! There’s been a mistake! Can you review your code and the error and identify what the problem is?	*Hint: We have not told R what the object today is. Consider looking at the Sys.Date function in the help documentation and entering it in the console. What information does this function return? Try creating an object called today and assigning it to sys.date and then re-running your ggsave code:*
	
	`ggsave(paste0(output_folder, "/ONnew_daily_cases_plot_", today, ".png"),`
	`       plot = ONnew_daily_cases_plot,`
	`       width = 11,`
	`       height = 7,`
	`       units = "in")`

You've created your first figure for this project! Hooray! Time for some cake. Note: Your epi curve will look slightly different from this as the data is taken directly from the Government of Canada's open source COVID-19 line list and is regularly updated.	
Ensure a heading is included in your script with details regarding the purpose, author, date, modifications, and other pertinent notes.	

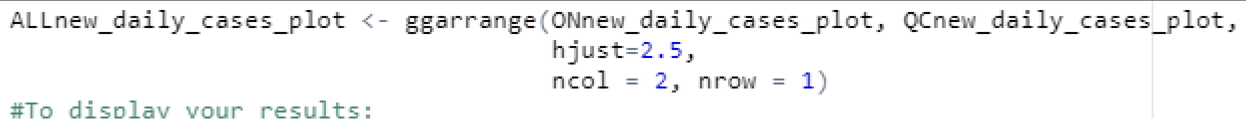
Save your script again as 05_1_ plot_epi_curve.R	

9. Review your code from steps 9. Can you identify all areas where the jurisdiction are specified? Recycle your code to recreate a similar figure for Quebec. Play around with some of the aesthetics (i.e. make your bars a different colour, change the size of your axis and label fonts, etc.). You can find different Hexcodes here: https://htmlcolorcodes.com/or simply type in English the colour you want (List of colours available: http://sape.inf.usi.ch/quick-reference/ggplot2/colour).

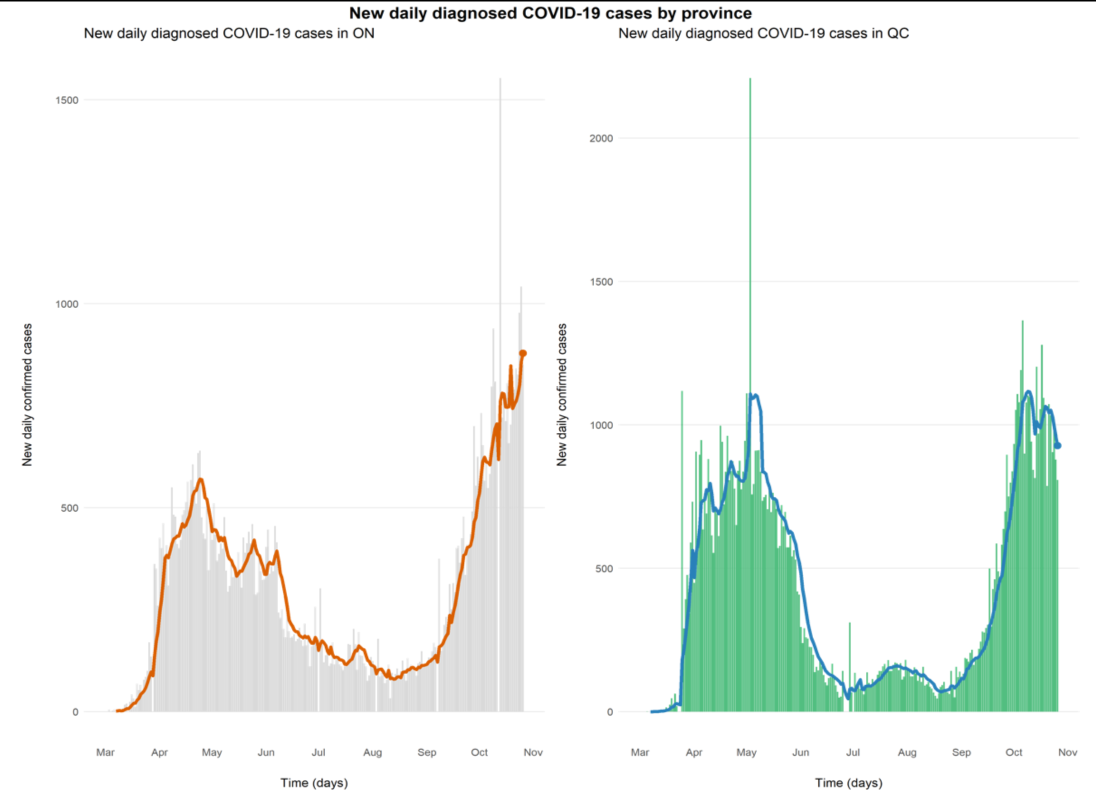
10. Great! You have code that works, and you're now going back to review and revise your code. To create a figure for each province will involve a lot of cutting and pasting, editing, and several pages of code. How would you manage this?

11. Bonus! In review and revision of your code, you've decided to edit your figures so that they can be laid out together in a single multipanel figure. Review your code in step 9. What changes would you consider making?

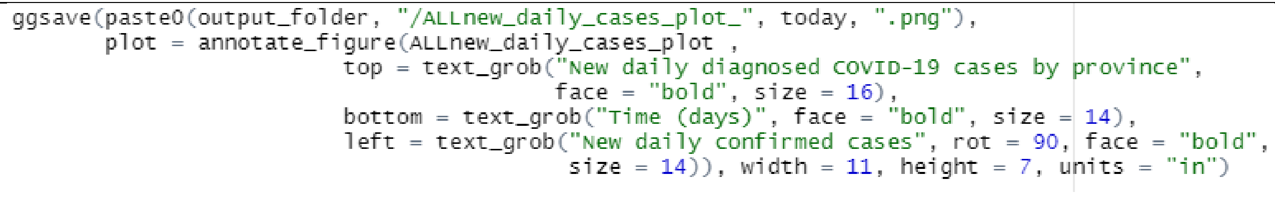
Task	**Code**

Create a multipanel figure. Edit the code to the right to include only your figures from Ontario and Quebec.	 `ALLnew_daily_cases_plot <- ggarrange(ONnew_daily_cases_plot, QCnew_daily_cases_plot,`
	`hjust=2.5,`
	`ncol = 2, nrow = 1)`
	
	`#To display your results:`

View your new multipanel plot	`ALLnew_daily_cases_plot`

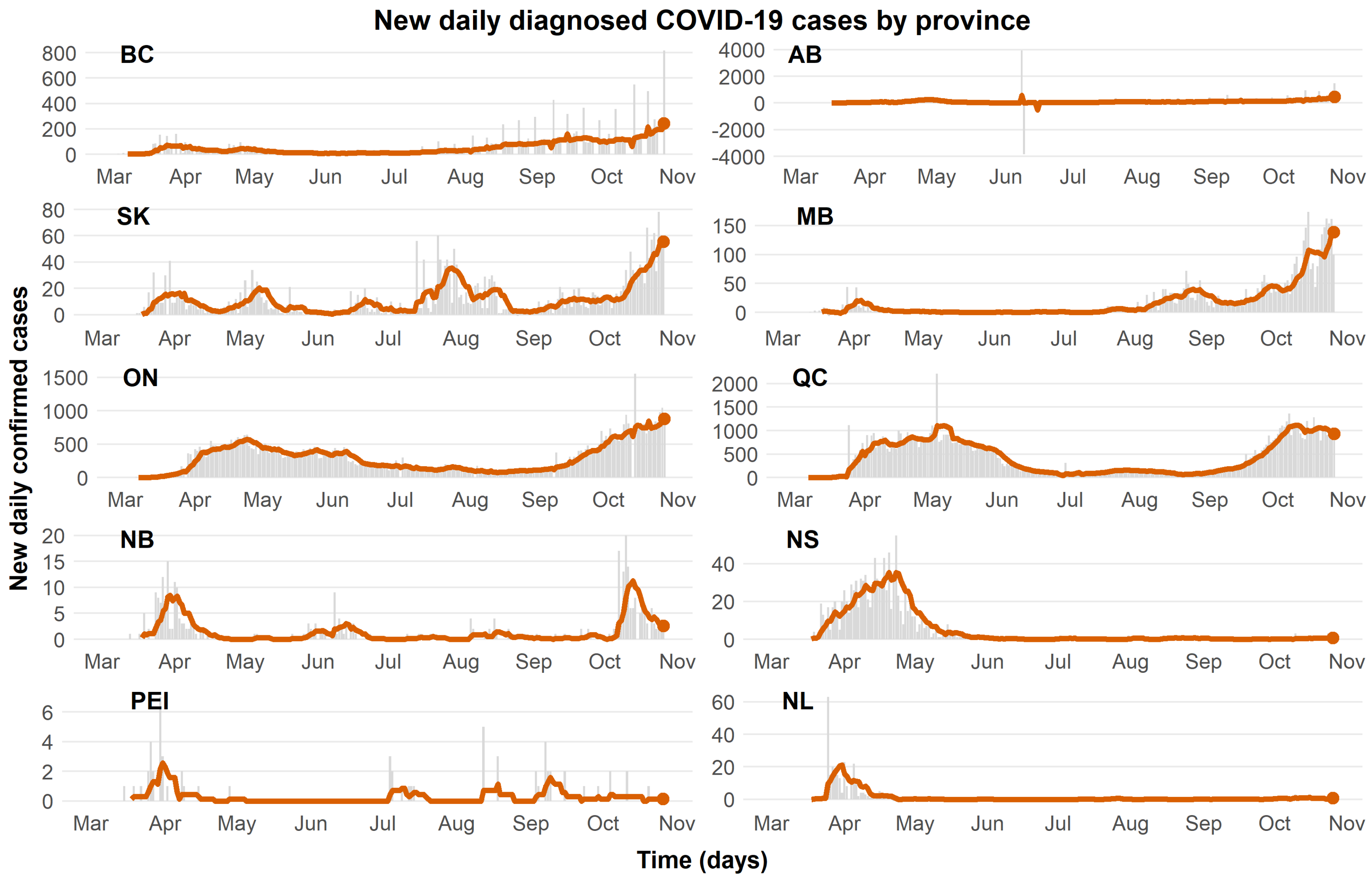
What do you notice about your plots? Is there anything you would go back and change to the original ON and QC epicurve to make it more visually appealing in a mulitpanel plot? Helpful tip: Look at the code for your original epi curve plot. Would you leave titles in as is?	
Rework your ON and QC epicurves until they format nicely into the multipanel plot.	Optional!

Hint: Try modifying the options in the **theme** function	

Export the figure with annotations (i.e., title and axis labels)	 `ggsave(paste0(output_folder, "/ALLnew_daily_cases_plot_", today, ".png"),`
	`       plot = annotate_figure(ALLnew_daily_cases_plot ,`
	`                              top = text_grob("New daily diagnosed COVID-19 cases by province",`
	`                                              face = "bold", size = 16),`
	`                              bottom = text_grob("Time (days)", face = "bold", size = 14),`
	`                              left = text_grob("New daily confirmed cases", rot = 90, face = "bold",`
	`                                               size = 14)), width = 11, height = 7, units = "in")`

Ensure a heading is included in your script with details regarding the purpose, author, date, modifications, and other pertinent notes.	
Save your 05_1_ plot_epi_curve.R	

12. Bonus! You've repurposed your code to make a multi panel plot for all provinces (See figure next page. Note: this code is not included in your scripts folder – but why don't you try recreating the figure on your own!). In taking a closer look at your figures, you notice something strange. What is going on with Alberta on June 8 and 9, 2020? Review data from those dates in the line list and all data created through processing and cleaning, and consider the methodology used to obtain counts for new daily cases. At what point was this error introduced? Would you choose to leave this figure the way it is? Why or why not? If not, how would you propose to fix this issue?


New daily diagnosed COVID-19 cases by province
New daily confirmed cases
Time (days)
BC
AB
SK
MB
ON
QC
NB
NS
PEI
NL
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
800
600
400
200
0
4000
2000
-2000
-4000
80
60
40
20
150
100
50
1500
1000
500
2000
15
10
5
6
4
2

SUPERSTAR FUNCTIONS

We've provided a list of the main functions used in this exercise for your reference. Functions written in bold font are the biggest stars of the day.

Superstar Functions	
here()	left_join()
install.packages()	**dmy()**
library()	str_replace_all()
read_csv()	Sys.Date()
paste0()	**ggplot()**
url()	**geom_bar()**
c()	geom_line()
select()	geom_point()
mutate()	theme()
filter()	element_text()
group_by()	element_blank()
ungroup()	scale_x_date()
arrange()	xlab()
pad()	ylab()
fill()	ggtitle()
pivot_longer()	aes()
pivot_wider()	**ggsave()**
str_detect()	ggarrange()
epiweek()	annotate_figure()