TDU

Introduction to R for Public Health Investigations

Workbook for day 2

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

Canada

Contents

Acknowledgements

Day 2 relies on materials developed in collaboration with subject matter expert Emma Cumming, Canadian Public Health Service. The Training and Development Unit is grateful for her significant contributions to this course.

Practical Exercise

Instructions

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 50 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

You've been sent on a mobilization to a TB outbreak, affecting members of a First Nations band both on and off-reserve. The affected reserve (population 2000) is in a remote northern area, with the nearest town being 25 km away (mixed settler and First Nations population, population 7500). TB cases off-reserve are all linked to a rooming house in the nearby town. You've been provided with data by the local health authorities, and need to clean it for analysis. The site wishes you to record all your code and steps in an R Markdown file, so they can repeat the analysis once you leave, if necessary. Your R Markdown report will be "rendered" or exported into a Word document.

There are three main ways to render static reports in r markdown: html, pdf, and word documents. For this exercise we will produce a word document report, the main benefits of which are the ability for collaborators to comment/edit after you have published the report, and because of most people's familiarity with Word. To render an r markdown file in Word, it is helpful to have a few packages installed that have been "developed to facilitate the production of word documents and PowerPoint presentations from and with R". These packages function well with ***tidyverse*** packages like ***tidyr, ggplot2***, and there are three main ways to render static reports in R Markdown: html, pdf, and word documents.

- **officer**[1]: helps generate word or powerpoint documents with R Markdown.

[1] https://ardata-fr.github.io/officeverse/

- **officedown**[2]: facilitates the formatting of Microsoft word documents produced by R Markdown documents, including paragraph formatting, sections, table formatting, and references/captions.
- **Flextable**[2]: helps easily create nice looking tables for reporting.

1. Set up your workspace

Start by setting up your workspace. If you haven’t done so already, create new folders on your computer to organize the files for Day 2 exercises as you did for Day 1:

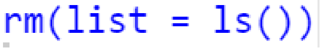
1. Within the IntroToR folder that you created on Day 1, create a subfolder for Day 2 called: "Exercise_Day2".
2. Create new folders within Exercise_Day2 called: "output"; "data"; "scripts".
3. Move the files for Day 2 from GitHub to their corresponding folders. (Remember, you may optionally create your own scripts, or work from the one’s we’ve provided in GitHub).

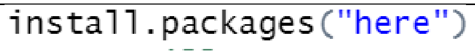
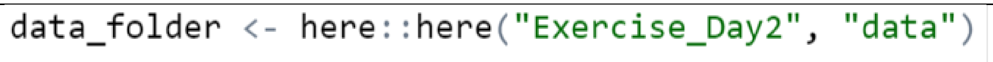
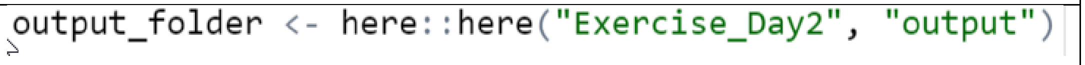
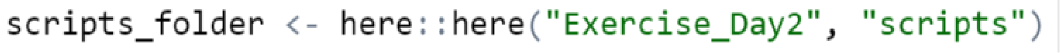
Note that structuring folders this way and creating new scripts as suggested in the tasks below will set learners up for success in the last stage of this practical exercise where an automated report will be created using R Markdown.

In RStudio:

One of the advantages of setting up an .RProj file is that it lets us pick up where we left off with the Day 1 exercises. This means that our working directory, history, and environment will already be set for us when we open the project file. *Neat*, right!?

Now that you have set up your project folders:

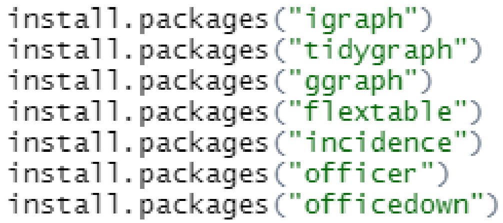
Task	Code
Open your *IntroToR.Rproj* (R-project) by either double-clicking the project file you created yesterday, or by opening RStudio and navigating to **File > Open Project** and selecting the project file you created in Day 1.	
If your project environment still contains objects left over from Day 1, let’s clear it so we can start fresh with Day 2 and avoid any confusion. Either select **Session > Clear Workspace** from the RStudio window, or (less	 `rm(list = ls())`

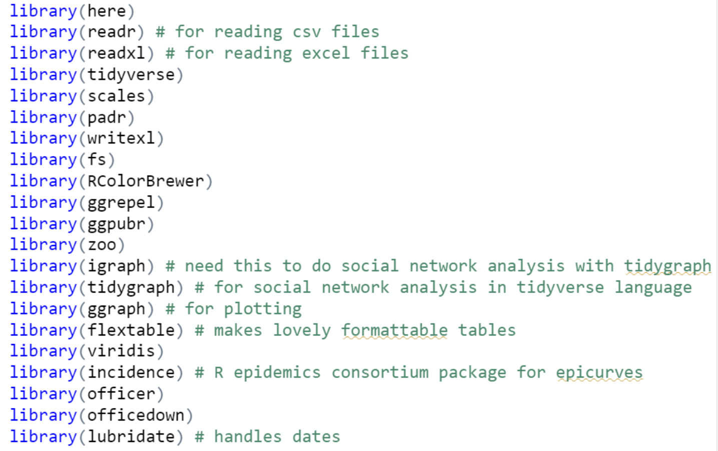
optimal) execute the following command in the console:	
Open a new script and save it as “01_1_define_paths.R” inside your scripts folder.	
Install the here package if you did not install it previously during the day 1 exercise. Installing packages only needs to be done once.	 `install.packages("here")`
Create an object that will direct R to the folder where your raw data are saved. Execute the statement.	 `data_folder <- here::here("Exercise_Day2", "data")`
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_Day2", "output")`
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_Day2", "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_1_define_paths.R”	

2. Load packages

Load the packages you will need for this project:

Task	**Code**	**Library info – for reference**
Open a new script and save it in your Exercise_Day2/scripts folder as “01_2_load_libraries.R”		

Install packages that aren’t already installed in RStudio on your computer. ***Note that this only needs to be done once, and isn’t necessary for any packages installed on Day 1.	
	`install.packages("igraph")`
	`install.packages("tidygraph")`
	`install.packages("ggraph")`
	`install.packages("flextable")`
	`install.packages("incidence")`
	`install.packages("officer")`
	`install.packages("officedown")`
		igraph: https://igraph.org/r/
		tidygraph: https://tidygraph.data-imaginist.com/
		ggraph: https://ggraph.data-imaginist.com/
		flextable: https://davidgohel.github.io/flextable/
		incidence: https://www.repidemicsconsortium.org/incidence/
		officer: https://davidgohel.github.io/officer/

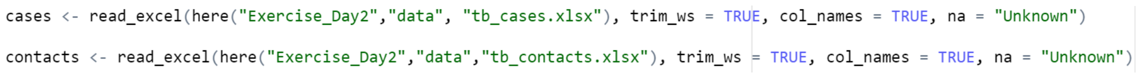
Load the libraries (installed packages) you will need for your project every time you will be using them.	 `library(here)`
	`library(readr) # for reading csv files`
	`library(readxl) # for reading excel files`
	`library(tidyverse)`
	`library(scales)`
	`library(padr)`
	`library(writexl)`
	`library(fs)`
	`library(RColorBrewer)`
	`library(ggrepel)`
	`library(ggpubr)`
	`library(zoo)`
	`library(igraph) # need this to do social network analysis with tidygraph`
	`library(tidygraph) # for social network analysis in tidyverse language`
	`library(ggraph) # for plotting`
	`library(flextable) # makes lovely formattable tables`
	`library(viridis)`
	`library(incidence) # R epidemics consortium package for epicurves`
	`library(officer)`
	`library(officedown)`
	`library(lubridate) # handles dates`
		officedown:
		https://davidgohel.github.io/officedown/

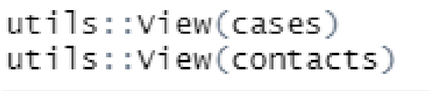
Ensure a heading is included in your script with details regarding the purpose, author, date, modifications, and other pertinent notes.		
Save your script as “01_2_load_libraries.R ”		

3. Load the data

Load the data you will need into RStudio:

Task	Code
Open a new script and save it in your Exercise_Day2/scripts folder as “01_3_load_data.R”	

Load the data required for this analysis. Explain in your own words what the code to the right is doing. What is the purpose of the trim_ws, col_names, and na arguments in the read_excel() function? Hint: If you aren’t sure, try looking them up under read_excel() in the Help files.	
	cases <- read_excel(here("Exercise_Day2","data", "tb_cases.xlsx"), trim_ws = TRUE, col_names = TRUE, na = "Unknown")
	contacts <- read_excel(here("Exercise_Day2","data","tb_contacts.xlsx"), trim_ws = TRUE, col_names = TRUE, na = "Unknown")

Run utils::View(cases) in the console. Note: View is with a capital V. What happens? How does this compare to the view() function? Which do you think is most useful to you in reviewing the data for cleaning?	
	utils::View(cases)
	utils::View(contacts)

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_3_load_data.R”	

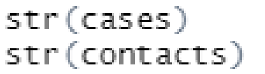
Resource: https://rveryday.wordpress.com/2016/11/29/examine-a-data-frame-in-r-with-7-basic-functions/

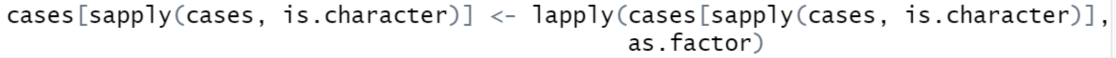
Note that these functions are not part of the tidyverse

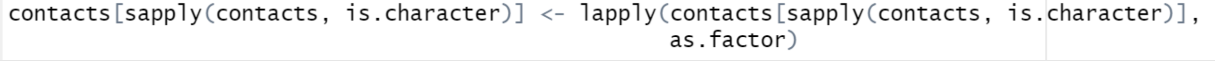
4. Clean and process the data

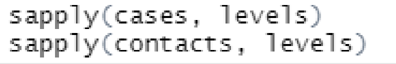
Now that the data are loaded into RStudio, you must clean and process it.

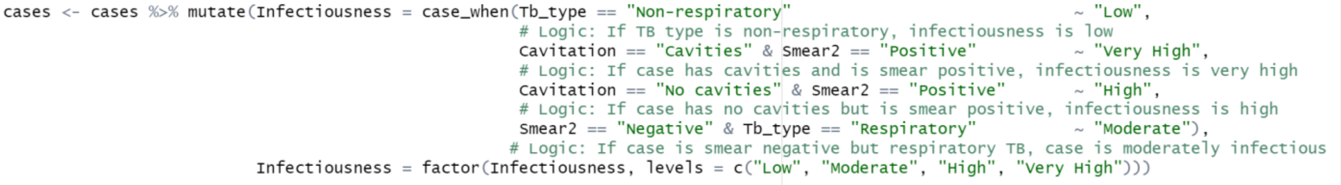
Task	Code
Open a new script and save it in your Exercise_Day2/scripts folder as “02_1_clean_data.R”	

First take a look at the variables in your cases and contacts data frames using the str function. The str function shows the structure of your data frame and includes information on the: number of rows and columns, column names, class of each column (type of data stored - i.e. character, numeric, etc.), and the first few observations of each variable. Look at the variable types in your cases and contacts datasets. Which variables need to be converted from string?	
	`str(cases)`
	`str(contacts)`

Convert string text variables in the cases dataset into factor variables. Factors can be used to represent categorical data (ordered or unordered).	
	`cases[sapply(cases, is.character)] <- lapply(cases[sapply(cases, is.character)],`
	`as.factor)`

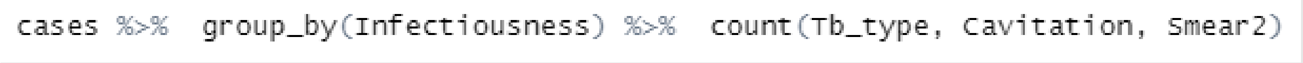
<table>
<tr><td>Conversion will aide in plotting activities in this exercise.</td><td></td></tr>
<tr><td>Convert string text variables in the contacts dataset into factor variables</td><td>

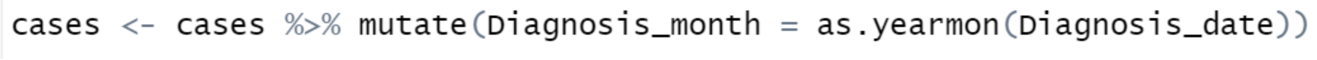
```
contacts[sapply(contacts, is.character)] <- lapply(contacts[sapply(contacts, is.character)],
                                                   as.factor)
```
</td></tr>
<tr><td>View the levels you have just created using the sapply function and specifying the levels option.
Note: Variables that are not classified as factors will have their level listed as "NULL”</td><td>

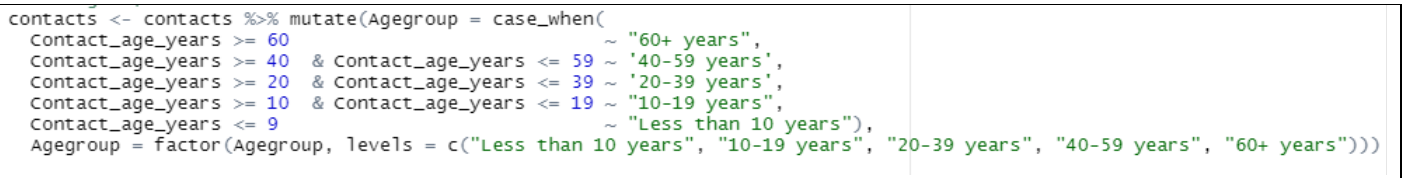
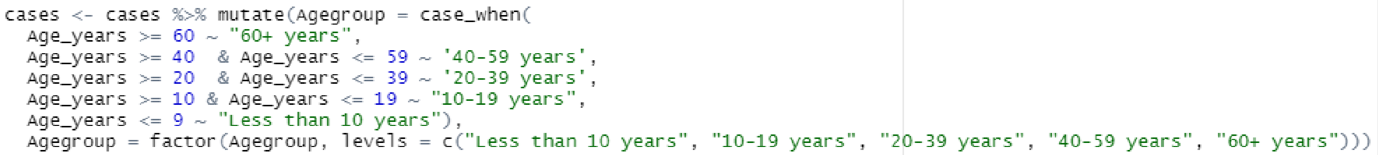
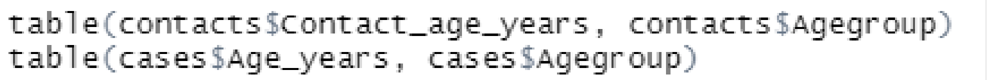
```
sapply(cases, levels)
sapply(contacts, levels)
```
</td></tr>
<tr><td>Create a new variable called Infectiousness indicating infectiousness of cases based on the variables: Tb_type, Cavitation, and Smear2. Use the following criteria to assign infectiousness as Low, Moderate, High and Very high[2]:
- If TB type is non-respiratory, infectiousness is low
- If case is smear negative but has respiratory TB, case is moderately infectious</td><td>

```
cases <- cases %>% mutate(Infectiousness = case_when(Tb_type == "Non-respiratory"                             ~ "Low",
                                                     # Logic: If TB type is non-respiratory, infectiousness is low
                                                     Cavitation == "Cavities" & Smear2 == "Positive"          ~ "Very High",
                                                     # Logic: If case has cavities and is smear positive, infectiousness is very high
                                                     Cavitation == "No cavities" & Smear2 == "Positive"       ~ "High",
                                                     # Logic: If case has no cavities but is smear positive, infectiousness is high
                                                     Smear2 == "Negative" & Tb_type == "Respiratory"          ~ "Moderate"),
                                                    # Logic: If case is smear negative but respiratory TB, case is moderately infectious
                          Infectiousness = factor(Infectiousness, levels = c("Low", "Moderate", "High", "Very High")))
```
</td></tr>
</table>

[2] For review see section 30. Conditional Operations on the R for Epidemiology site: https://www.r4epi.com/conditional-operations.html

- If case has no cavities but is smear positive, infectiousness is high - If case has cavities and is smear positive, infectiousness is very high Like an if statement, the arguments are evaluated in order, so you must proceed from the most specific to the most general. This variable creation uses dplyr's mutate and the case_when() function. The values you want your new variable to take follow the "~" symbol The factor function allows us to set the order of the variables levels. Can you think of any reasons why we would want to set the order of non-ordered values?	
Check to see if the levels were set up correctly. You can do this using the group_by function and creating a table. Group the data by the new	 `cases %>%  group_by(Infectiousness) %>%  count(Tb_type, Cavitation, Smear2)`

variable you just created (Infectiousness) and include counts of the variables you want to include in your check (Tb_type, Cavitation, and Smear2). Does it look like Infectiousness was categorized correctly?	
Create a new variable called Diagnosis_month displaying diagnosis date as a year and month. Note: In the console, try checking your work by creating a cross tabulation of the Diagnosis_month and Diagnosis_date variables by using the table function. Note: This is a base R function and not tidyverse so you need to add the dataframe name and a $ before every variable you include in your code (e.g., table(cases$Diagnosis_month, cases$Diagnosis_date)	 `cases <- cases %>% mutate(Diagnosis_month = as.yearmon(Diagnosis_date))`

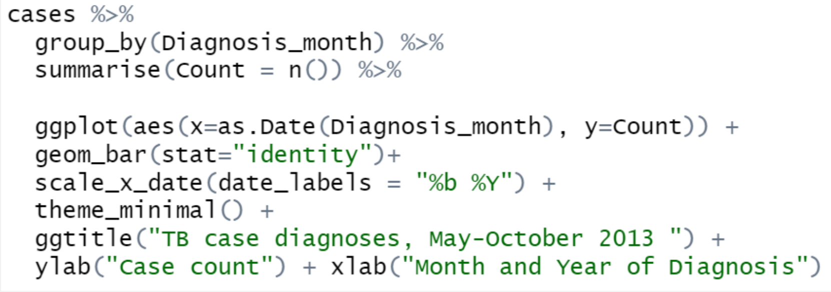
<table>
<tr>
<td>Using the mutate and case_when functions create a new variable called Agegroup containing the following age groups for contacts:
Less than 10 years
10-19 years
20-39 years
40-59 years
60+ years</td>
<td>
<pre>
contacts <- contacts %>% mutate(Agegroup = case_when(
  Contact_age_years >= 60   ~ "60+ years",
  Contact_age_years >= 40 & Contact_age_years <= 59 ~ '40-59 years',
  Contact_age_years >= 20 & Contact_age_years <= 39 ~ '20-39 years',
  Contact_age_years >= 10 & Contact_age_years <= 19 ~ "10-19 years",
  Contact_age_years <= 9   ~ "Less than 10 years"),
  Agegroup = factor(Agegroup, levels = c("Less than 10 years", "10-19 years", "20-39 years", "40-59 years", "60+ years")))
</pre></td>
</tr>
<tr>
<td>Create a new variable called Agegroup containing age groups for cases using the same age groups as you did for contacts.</td>
<td>
<pre>
cases <- cases %>% mutate(Agegroup = case_when(
  Age_years >= 60 ~ "60+ years",
  Age_years >= 40 & Age_years <= 59 ~ '40-59 years',
  Age_years >= 20 & Age_years <= 39 ~ '20-39 years',
  Age_years >= 10 & Age_years <= 19 ~ "10-19 years",
  Age_years <= 9 ~ "Less than 10 years"),
  Agegroup = factor(Agegroup, levels = c("Less than 10 years", "10-19 years", "20-39 years", "40-59 years", "60+ years")))
</pre></td>
</tr>
<tr>
<td>Check to see if you have correctly classified your age groups using the table function. Note: This is a base R function and not tidyverse so you need to add the dataframe name and a $ before every variable you include in your code (i.e. cases$Age_years)</td>
<td>
<pre>
table(contacts$Contact_age_years, contacts$Agegroup)
table(cases$Age_years, cases$Agegroup)
</pre></td>
</tr>
<tr>
<td>Ensure a heading is included in your script with details</td>
<td></td>
</tr>
</table>

regarding the purpose, author, date, modifications, and other pertinent notes.	
Save your script as “02_1_clean_data.R”	

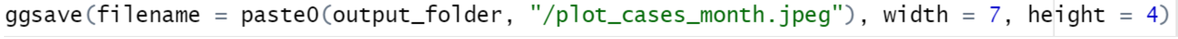
5. Visualize the data

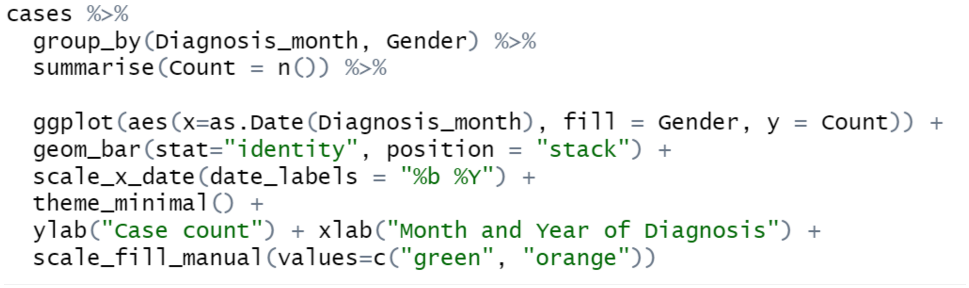
Plot the case data over time. Bonus (if you have extra time)! Try exploring different themes:

<table>
<tr><th>Task</th><th>Code</th></tr>
<tr><td>Open a new script and save it in your Exercise_Day2/scripts folder as “03_1_plot_case_time.R”</td><td></td></tr>
<tr><td>Plot the case data by diagnoses month.

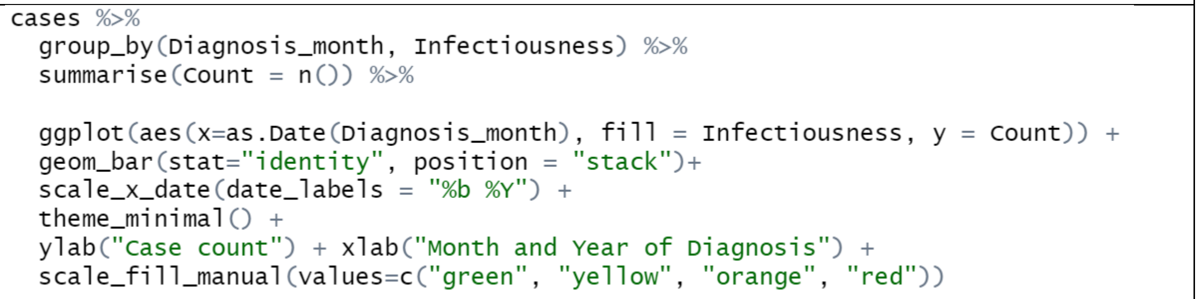
First Summarize the case data into a frequency table of case count per month of diagnosis and then pipe this frequency table into a GGPLOT command for a bar plot, with the x-axis as Diagnosis Month and the y-axis as Count.
Add in the following formats to your graph:
- Set the bar heights equal to the value in the data, rather than counts (using stat=”identity”)
- Use a month and year date format i.e. Aug 2020
- Use the minimal theme
- Add labels: x-axis - “Month and Year of Diagnosis”; Y-axis - “Case Count”</td><td>
<pre>
cases %>%
  group_by(Diagnosis_month) %>%
  summarise(Count = n()) %>%

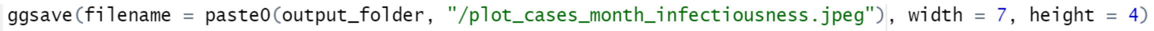
  ggplot(aes(x=as.Date(Diagnosis_month), y=Count)) +
  geom_bar(stat="identity")+
  scale_x_date(date_labels = "%b %Y") +
  theme_minimal() +
  ggtitle("TB case diagnoses, May-October 2013 ") +
  ylab("Case count") + xlab("Month and Year of Diagnosis")
</pre></td></tr>
</table>

Extra resource: For more help with formatting dates - https://www.r-bloggers.com/2013/08/date-formats-in-r/	
Save the resulting figure as a jpeg in your specified output folder and call it plot_cases_month Note: the paste0() function pastes together any text strings you provide, with no spaces between.	 `ggsave(filename = paste0(output_folder, "/plot_cases_month.jpeg"), width = 7, height = 4)`

Plot the case data by diagnosis month and gender
First summarize the case data into a frequency table of case counts per month of diagnosis and by gender, and then pipe this frequency table into a GGPLOT command for a bar plot, with the x-axis as Diagnosis Month and the y-axis as Count. In bar plots, you can colour-stratify by another variable (in the case gender) by specifying "fill= Gender" in the aes() call.
	
	`cases %>%`
	`  group_by(Diagnosis_month, Gender) %>%`
	`  summarise(Count = n()) %>%`
	
	`  ggplot(aes(x=as.Date(Diagnosis_month), fill = Gender, y = Count)) +`
	`  geom_bar(stat="identity", position = "stack") +`
	`  scale_x_date(date_labels = "%b %Y") +`
	`  theme_minimal() +`
	`  ylab("Case count") + xlab("Month and Year of Diagnosis") +`
	`  scale_fill_manual(values=c("green", "orange"))`

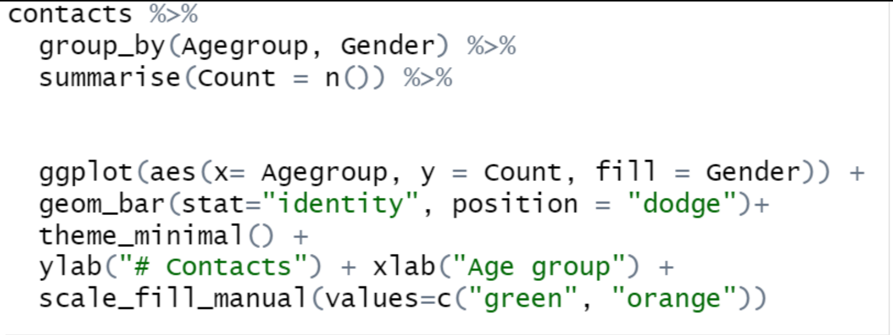
Use the same formatting as the previous graph except for the following changes: - Set your bar plot to be stacked (using Position = "stack") rather than a side-by-side one (i.e. position = "dodge") - Add labels: x-axis - “Month and Year of Diagnosis”; Y-axis - “Case Count” Specify the bar colours to be green for females and orange for males (using the scale_fill_manual option). The order you list these colours will match the order of any factor variable. If you do not know the order you can check with the levels() function using the code: levels(cases$Gender)	
Save the resulting figure as a jpeg in your specified output folder and call it plot_cases_month_gender	ggsave(filename = paste0(output_folder, "/plot_cases_month_gender.jpeg"), width = 7, height = 4)

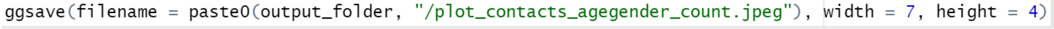
Plot the case data by diagnosis month and infectiousness.
 	
First, summarize the case data into a frequency table of case counts per month of diagnosis and by infectiousness, and then pipe this frequency table into a GGPLOT command for a bar plot, with the x-
	 cases %>%
	  group_by(Diagnosis_month, Infectiousness) %>%
	  summarise(Count = n()) %>%
	
	  ggplot(aes(x=as.Date(Diagnosis_month), fill = Infectiousness, y = Count)) +
	  geom_bar(stat="identity", position = "stack")+
	  scale_x_date(date_labels = "%b %Y") +
	  theme_minimal() +
	  ylab("Case count") + xlab("Month and Year of Diagnosis") +
	  scale_fill_manual(values=c("green", "yellow", "orange", "red"))

axis as Diagnosis Month, fill as Infectiousness and y-axis as Count. Specify the bar colours: green=low; yellow=moderate; orange=high and red=very high. To check the order of the Infectiousness levels you can use the code: levels(cases$Infectiousness) Format the graph using the same options as the previous graph	
Save the resulting figure as a jpeg in your specified output folder and call it plot_cases_month_infectiousness	 ggsave(filename = paste0(output_folder, "/plot_cases_month_infectiousness.jpeg"), width = 7, height = 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 “03_1_plot_case_time.R”	

Plot contact demographics. Bonus (if you have extra time)! Try exploring different themes. https://ggplot2.tidyverse.org/reference/ggtheme.html

Task	**Code**
Open a new script and save it in your Exercise_Day2/scripts folder as “03_2_plot_contact_demographics.R”	

Plot the contacts data by age group and gender.
First, make a frequency table of contact age group and gender counts and pipe the frequency table into a ggplot with the x-axis as Agegroup, fill as Gender and the y-axis as Count.
Add in the following formats to your graph:
- Set the bar heights equal to the value in the data, rather than counts (using stat=”identity”)
- Set the bar plot to sit side-by-side (using Position = "dodge")
- Use the minimal theme
- Add labels: x-axis - “Age group; Y-axis - “Contacts”
- Specify the bar colours to be green for females and orange for males (using the scale_fill_manual option).
To check order of the Gender levels use the code: levels(contacts$Gender)
	
	`contacts %>%`
	`  group_by(Agegroup, Gender) %>%`
	`  summarise(Count = n()) %>%`
	
	`  ggplot(aes(x= Agegroup, y = Count, fill = Gender)) +`
	`  geom_bar(stat="identity", position = "dodge")+`
	`  theme_minimal() +`
	`  ylab("# Contacts") + xlab("Age group") +`
	`  scale_fill_manual(values=c("green", "orange"))`

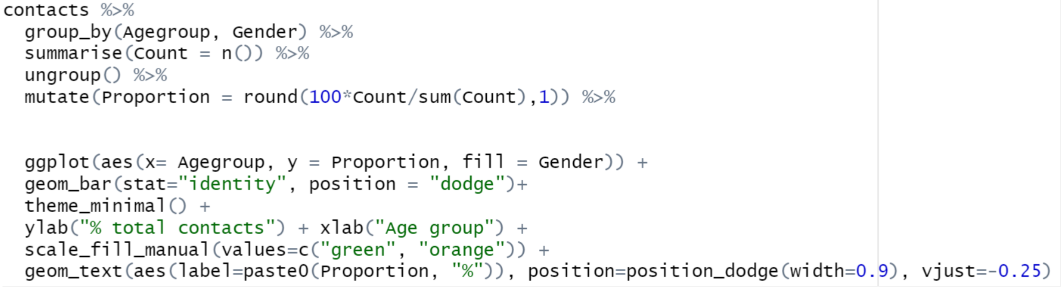
Save the resulting graph as an image (jpeg) in the outputs folder and call it plot_contacts_agegender_count	 `ggsave(filename = paste0(output_folder, "/plot_contacts_agegender_count.jpeg"), width = 7, height = 4)`

Plot the contacts data by age group and gender proportions.

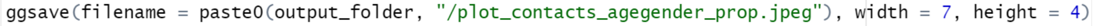
First, make a frequency table of contact age group and gender counts. Next, you will need to ungroup or proportions will be calculated within the first variable which in this case is Agegroup. Once ungrouped, create a variable called Proportion that calculates proportions using the overall total as the denominator. Pipe them into a ggplot with the x-axis as Agegroup, y-axis as Proportion and fill as Gender.

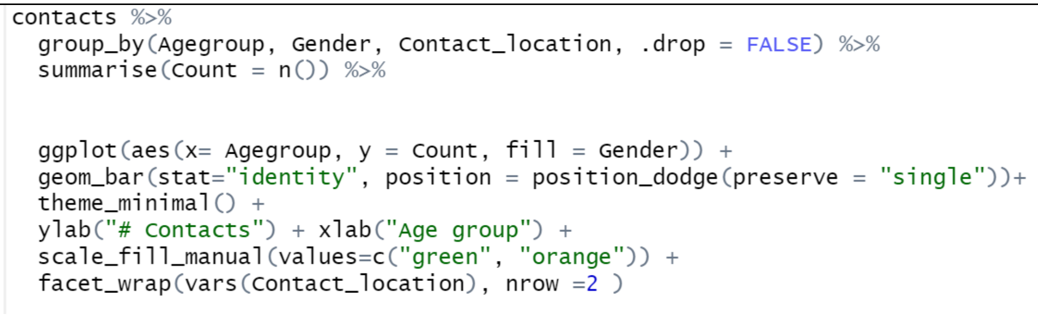
Use the same formatting as the previous graph except for the following changes:

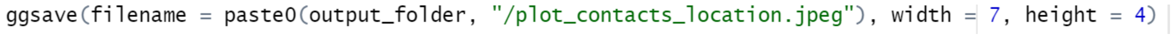
- Add labels: x-axis - “Age group”; Y-axis - “% total contacts”
- Add labels to each of the bars using the aes(label) option. Specify that you want to display the “Proportion” and a “%” sign. Also specify the positions of your labels using position_dodge (so that labels are above each bar on your graph) and specify the label font size.

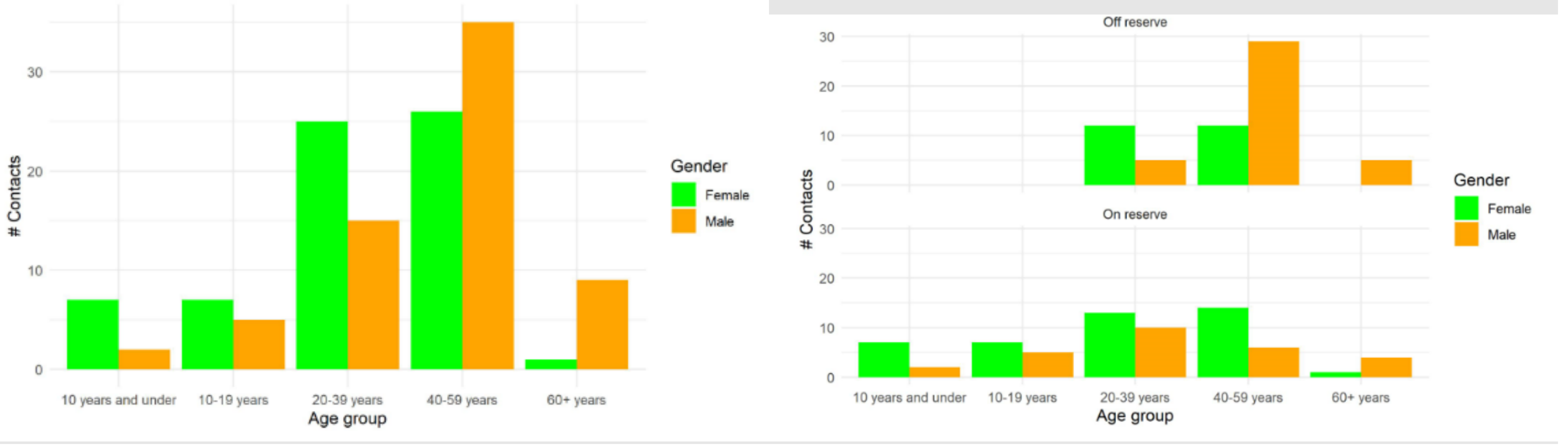


```
contacts %>%
  group_by(Agegroup, Gender) %>%
  summarise(Count = n()) %>%
  ungroup() %>%
  mutate(Proportion = round(100*Count/sum(Count),1)) %>%


  ggplot(aes(x= Agegroup, y = Proportion, fill = Gender)) +
  geom_bar(stat="identity", position = "dodge")+
  theme_minimal() +
  ylab("% total contacts") + xlab("Age group") +
  scale_fill_manual(values=c("green", "orange")) +
  geom_text(aes(label=paste0(Proportion, "%")), position=position_dodge(width=0.9), vjust=-0.25)
```

Save the resulting graph as an image in the outputs folder and call it plots_contacts_agegender_prop	 `ggsave(filename = paste0(output_folder, "/plot_contacts_agegender_prop.jpeg"), width = 7, height = 4)`
Bonus (If you have time)! Try excluding the following line from your code above: ungroup() %>% What happens to your graph?	

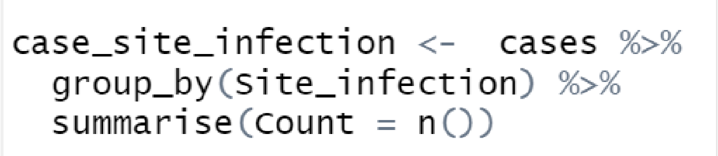
Create two graphs in one figure for contacts by age and gender on and off reserve.
 	
First, make a frequency table of contact age group and gender counts by location. Use the .drop = FALSE option which pads your summary table to include zeros in the tables and figures. Pipe the frequency table into a ggplot with the x-axis as "Agegroup", y-axis as "Count" and fill as "Gender".
 	
Use the same formatting as the previous graph except for the following changes:
- Set the bar plot to sit side-by-side and preserve the bar width using the following code: position = position_dodge(preserve = "single")
	
	`contacts %>%`
	`  group_by(Agegroup, Gender, Contact_location, .drop = FALSE) %>%`
	`  summarise(Count = n()) %>%`
	
	`  ggplot(aes(x= Agegroup, y = Count, fill = Gender)) +`
	`  geom_bar(stat="identity", position = position_dodge(preserve = "single"))+`
	`  theme_minimal() +`
	`  ylab("# Contacts") + xlab("Age group") +`
	`  scale_fill_manual(values=c("green", "orange")) +`
	`  facet_wrap(vars(Contact_location), nrow =2 )`

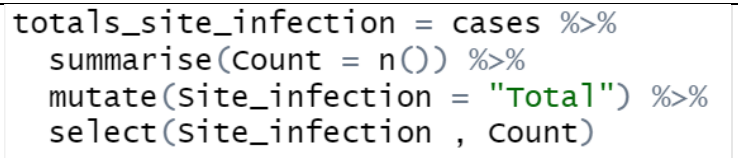
- Add labels: x-axis - “Age group”; Y-axis - “# contacts” - Use the facet_wrap option to create a panel based on the Contact_location. Specify that you would like 2 rows (nrow=2)	
Save the resulting graph as an image (jpeg) in the outputs folder and call it plots_contacts_location	 `ggsave(filename = paste0(output_folder, "/plot_contacts_location.jpeg"), width = 7, height = 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 03_2_plot_contact_demographics	



Create a nicely formatted frequency table for site of infection using Flextable:

Task	Code
Open a new script and save it in your Exercise_Day2/scripts folder as “03_3_tab_case_site.R”	

First, summarize case data into a frequency table by site of infection. Call the newly created data frame “case_site_infection” Because we want counts by site of infection, you will need to group_by site of infection.
To view your table either click on the new data frame you have created called case_site_infection or write the code: case_site_infection
	
	`case_site_infection <-  cases %>%`
	`  group_by(Site_infection) %>%`
	`  summarise(Count = n())`

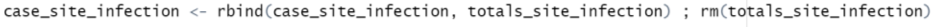
To add a totals row we create a one-row table (called “totals_site_infection”) that does not group by any variable, to get the total count.
Use the following functions to create your table:
- Summarise - gives total count of cases
- Mutate - allows you to create a new column in order to match with the frequency table you've created earlier, so you can append this totals row to it.
- Select - limits the columns to match the frequency table you created earlier.
To view your table either click on the new data frame you have created called totals_site_infection or write the code: totals_site_infection
	
	`totals_site_infection = cases %>%`
	`  summarise(Count = n()) %>%`
	`  mutate(Site_infection = "Total") %>%`
	`  select(Site_infection , Count)`

<table>
<tr><td></td><td></td></tr>
<tr><td>

Bind table with the totals row using the rbind function

Note: The rm function is used to remove unneeded objects. In this example, totals_site_infection was removed as it was no longer needed after we combined our tables.

</td><td>



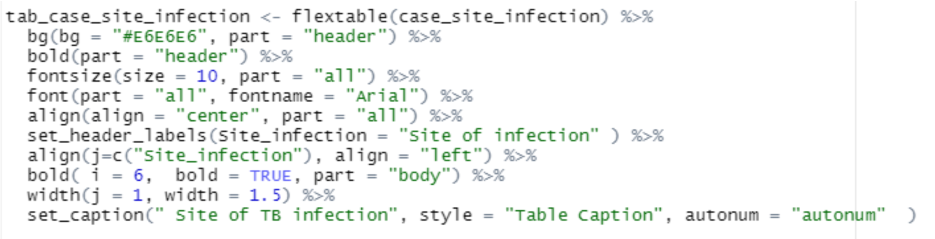
```r
case_site_infection <- rbind(case_site_infection, totals_site_infection) ; rm(totals_site_infection)
```

</td></tr>
<tr><td>

Create a flextable with the following options:

- Set the background colour of the header to grey (#E6E6E6) using the bg function
- Make the header bold using the bold function
- Make the font size 10 in all parts of the table using the fontsize function
- Change the font type to arial using the font function
- Make the text in the body centre-justified using the align function
- Change the column heading from “site_infection” to “Site of Infection” using the set_header_labels function
- Make the first column (Site_infection) left aligned using the align function
- Make the Total row at the bottom bold using the bold function (Hint: It’s the sixth row).
- Set the first column (Site_infection) width to 1.5 inches wide using the width function
- Add the title: “Site of TB infection” to the figure using the set_caption function. Use autonum so that the caption title for this figure will appear as a numbered caption in Word.

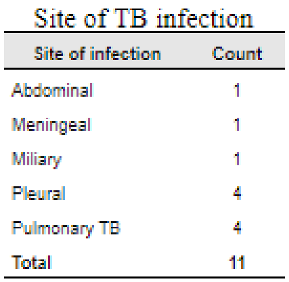
</td><td>



```r
tab_case_site_infection <- flextable(case_site_infection) %>%
  bg(bg = "#E6E6E6", part = "header") %>%
  bold(part = "header") %>%
  fontsize(size = 10, part = "all") %>%
  font(part = "all", fontname = "Arial") %>%
  align(align = "center", part = "all") %>%
  set_header_labels(Site_infection = "Site of infection" ) %>%
  align(j=c("Site_infection"), align = "left") %>%
  bold( i = 6,  bold = TRUE, part = "body") %>%
  width(j = 1, width = 1.5) %>%
  set_caption(" Site of TB infection", style = "Table Caption", autonum = "autonum"  )
```

</td></tr>
</table>

The following article provides a great overview of flextables and how to format them: https://davidgohel.github.io/flextable/articles/overview.html	
Print the table	`tab_case_site_infection`
Tidy your workspace by removing any objects no longer needed using the rm function. In this case remove case_site_infection	`rm(case_site_infection)`
Ensure a heading is included in your script with details regarding the purpose, author, date, modifications, and other pertinent notes.	
Save your script as “03_3_tab_case_site.R”	

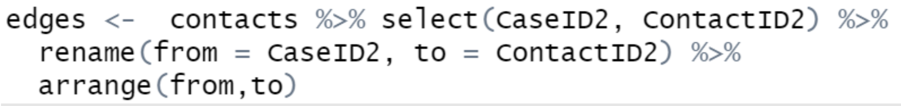


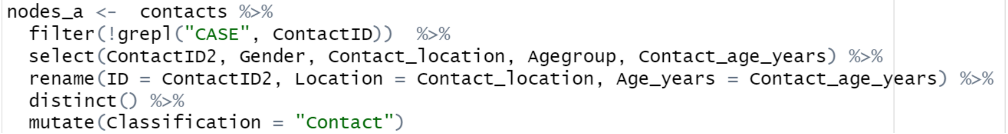
Site of TB infection

Site of infection	Count
Abdominal	1
Meningeal	1
Miliary	1
Pleural	4
Pulmonary TB	4
Total	11

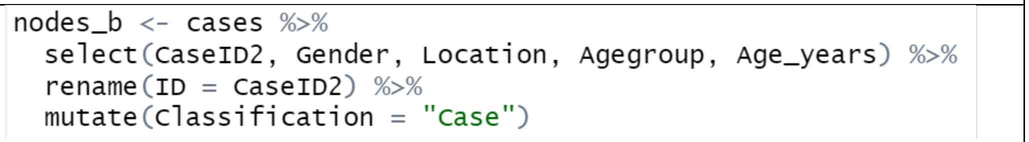
Draw a social network graph to illustrate the relationships between cases and contacts. First, we need to wrangle our cases and contacts data frames into node and edge data frames. This can be done in a thousand ways! The code below is just one way to do this.

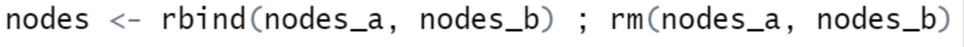
Task	**Code**
Open a new script and save it in your Exercise_Day2/scripts folders as “04_1_plot_sna.R”	

Create an edge data frame using the edge function. This depicts the relationship between the cases and the contacts. We only need 2 variables: the ids of cases (CaseID2) and the ids contacts (ContactID2).
Rename columns so that CaseID2= "from" and ContactID2= "to"
Reorder (using arrange) the columns so that “from” is first.
	
	`edges <-  contacts %>% select(CaseID2, ContactID2) %>%`
	`  rename(from = CaseID2, to = ContactID2) %>%`
	`  arrange(from,to)`

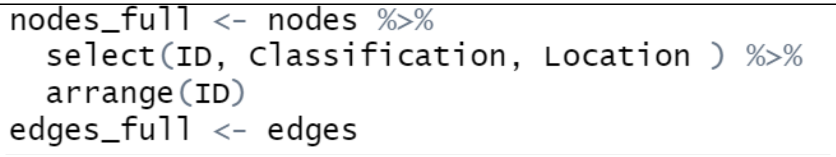
Create a node data frame from your contacts data called nodes_a.
Pare down the case-contact list to show unique contacts only, excluding cases who are named as contacts.
Exclude all rows that contain "CASE" text string in the ContactID (using filter).
	
	`nodes_a <-  contacts %>%`
	`  filter(!grepl("CASE", ContactID))  %>%`
	`  select(ContactID2, Gender, Contact_location, Agegroup, Contact_age_years) %>%`
	`  rename(ID = ContactID2, Location = Contact_location, Age_years = Contact_age_years) %>%`
	`  distinct() %>%`
	`  mutate(Classification = "Contact")`

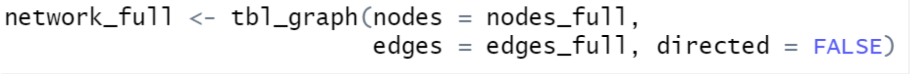
Select the variables we need (ContactID2, Gender, Contact_location, Agegroup, Contact_age_years) Rename variables as needed (ID = ContactID2, Location = Contact_location, Age_years = Contact_age_years). De-duplicate (using the distinct function). Select only unique rows. Create a new column (using mutate) and assign everyone the classification of 'Contact'.	

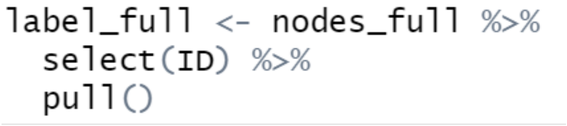
Create a node data frame from your cases data called nodes_b.
Select the variables we need (CaseID2, Gender, Location, Agegroup, Age_years).
Rename ID = CaseID2
Create a new column (using mutate) and assign all the classification of 'Case'.
	
	nodes_b <- cases %>%
	  select(CaseID2, Gender, Location, Agegroup, Age_years) %>%
	  rename(ID = CaseID2) %>%
	  mutate(Classification = "Case")

Bind the two matching data frames together (using rbind). To do this, columns must have the same names and be in the same order. Remove the nodes_a, nodes_b data frames as they are no longer needed (using rm).	 nodes <- rbind(nodes_a, nodes_b) ; rm(nodes_a, nodes_b)

Look at the nodes data frame you have created. Did it bind correctly?	

Convert our edge table into a tbl_graph object structure using the as_tbl_graph() function from tidygraph. It can take many different types of input data such as: data.frame, matrix, dendrogram, igraph, etc.
Rename edges and nodes to edges_full and nodes_full (just for clarity in coding).
Select the variable we need (ID, Classification, Location).
Sort by ID (using arrange).
	
	`nodes_full <- nodes %>%`
	`  select(ID, Classification, Location ) %>%`
	`  arrange(ID)`
	`edges_full <- edges`

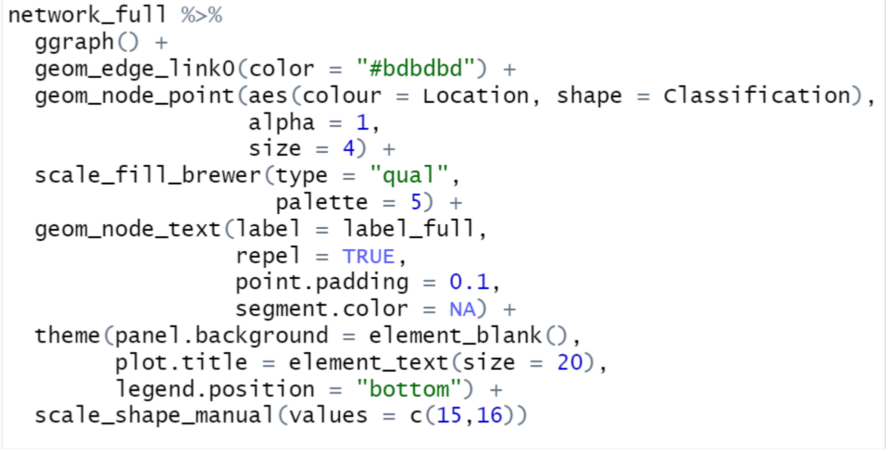
Create a tidygraph network object.
Tell tidygraph which data frame corresponds to the nodes & edges.
Note: The directed=FALSE option specifies that the relationships are non-directional in this case.
	
	`network_full <- tbl_graph(nodes = nodes_full,`
	`                          edges = edges_full, directed = FALSE)`

Define labels that will appear on the graph. In this case, ID is what we want to display as labels (using the select function).
Try running the code with and without the “%>% pull()” statement at the end. What do you think the pull() function is doing in this instance?
	
	`label_full <- nodes_full %>%`
	`  select(ID) %>%`
	`  pull()`

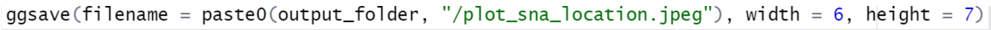
Create network plot (note how similar this is to calling a ggplot!)

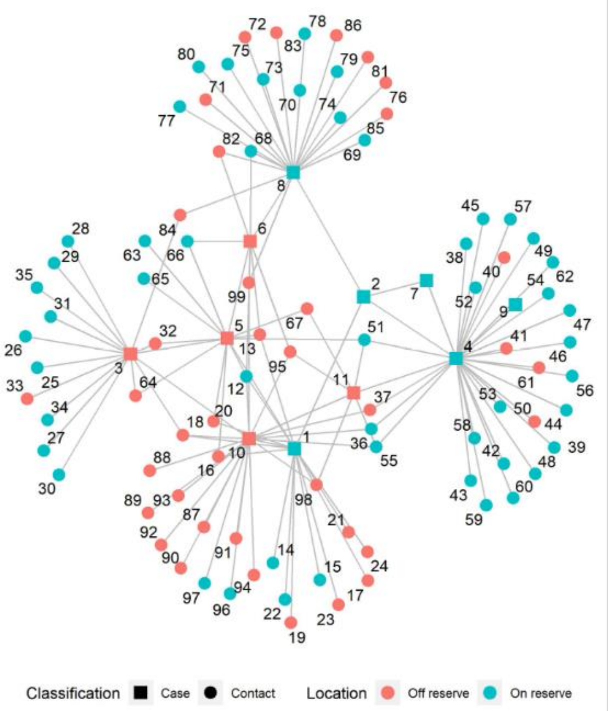
Add the following formats to your plot:

- Set the colour of the lines connecting edges to grey (bdbdbd) (using geom_edge_link0)
- Set the node colour to be dependent on Location and the shape on Classification (using aes)
- Set the transparency of the points (using alpha)
- Set the size of the points to 4 (using size)
- Select your colour palette (using scale_fille_brewer)
  Add the labels you defined a step earlier (called label_full) and set the labels so they do not overlap (using repel= TRUE)
- Adjust spacing of labels in relation to points by providing an option for the point.padding argument
- Remove lines attaching label to points by specifying NA in the segment.color argument
- Set the background to be blank (using panel.background= element(blank)),
- Change the font size of the title to 20 (using element_text(size=20))
- Move the legend position to the bottom (using legend.position= “bottom”)
- Set which shapes you want for the points using scale_shape_manual. Set cases as filled squares (option 15) and contacts as filled circles (option 16)



```
network_full %>%
  ggraph() +
  geom_edge_link0(color = "#bdbdbd") +
  geom_node_point(aes(colour = Location, shape = Classification),
                  alpha = 1,
                  size = 4) +
  scale_fill_brewer(type = "qual",
                    palette = 5) +
  geom_node_text(label = label_full,
                 repel = TRUE,
                 point.padding = 0.1,
                 segment.color = NA) +
  theme(panel.background = element_blank(),
        plot.title = element_text(size = 20),
        legend.position = "bottom") +
  scale_shape_manual(values = c(15,16))
```

- See link for more shape options: http://sape.inf.usi.ch/quick-reference/ggplot2/shape	
Save the resulting plot as a jpeg image in the outputs folder and call it plot_sna_location. Note: We want the image to be pretty large (6"x7").	 `ggsave(filename = paste0(output_folder, "/plot_sna_location.jpeg"), width = 6, height = 7)`
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_plot_sna	

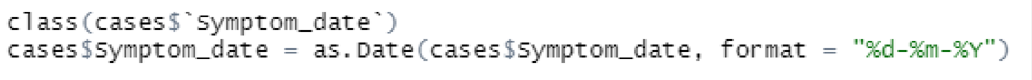

72
78
86
80
75
83
79
73
81
71
70
74
76
77
85
82
68
69
8
45
57
28
84
6
38
49
29
63
66
40
62
35
65
2
54
99
7
52
31
9
47
67
32
5
51
41
26
13
4
46
3
95
25
64
11
61
56
33
12
37
53
34
20
50
44
18
58
27
1
36
10
55
42
39
88
48
16
30
98
60
89
93
43
87
21
59
92
91
14
24
90
15
94
17
97
96
22
23
19
Classification
■ Case
● Contact
Location
● Off reserve
● On reserve

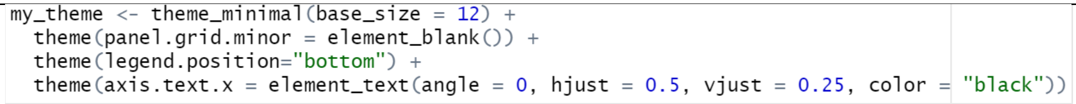
6. Bonus task 1

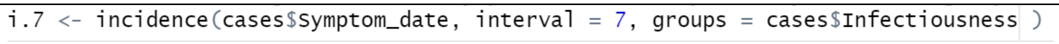
Plot an epi curve showing week of symptom onset, with infectiousness levels coded as different colours:

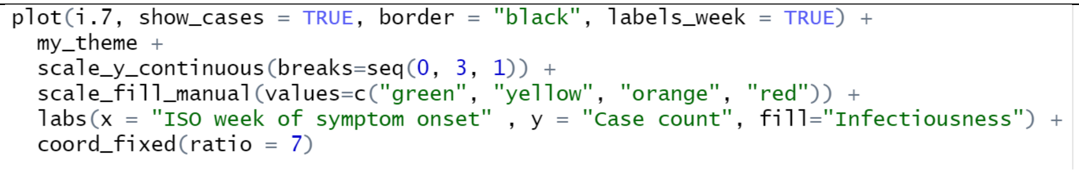
Note: Incidence package is not part of the tidyverse and so uses base R programming notation (e.g., to work with data you must name the data frame and the variables specifically separated by a '$', df$var1). It is however, an epidemiologist friendly package developed by the R Epidemics Consortium (RECON).

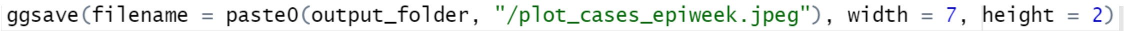
Task	Code
Open a new script and save it in your Exercise_Day2/scripts folder as "05_1_plot_epicurve.R"	

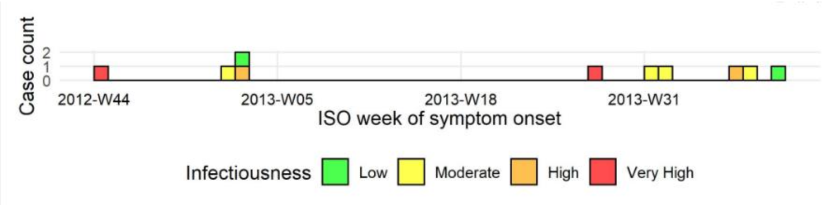
Check date format (using class).
This package likes dates formatted as Date, not POSIXct (type ?POSIXct in console to learn more).
Convert to "Date" format (%d-%m-%Y) if necessary (using as.Date).
	
	``class(cases$`Symptom_date`)``
	`cases$Symptom_date = as.Date(cases$Symptom_date, format = "%d-%m-%Y")`

Create a format theme for epicurve using ggplot2 language with the following formats:
- Set the theme to minimal (i.e. no background, annotations, etc.) using theme minimal
- Specify the base font size (all text elements in the plot) as 12 (base_size=12)
	
	`my_theme <- theme_minimal(base_size = 12) +`
	`  theme(panel.grid.minor = element_blank()) +`
	`  theme(legend.position="bottom") +`
	`  theme(axis.text.x = element_text(angle = 0, hjust = 0.5, vjust = 0.25, color = "black"))`

- Remove minor gridlines - Position the legend to the bottom (legend.position=bottom) - Adjust the height and angle of the x-axis text and make the font black.	
Create incidence object based on symptom onset date per 7 day interval, grouped by infectiousness.	 `i.7 <- incidence(cases$Symptom_date, interval = 7, groups = cases$Infectiousness )`

Plot the incidence object (to create the epi curve). Apply the theme you created above (my_theme) and add in the following additional formats:
- Display each case in the figure as a unique rectangle (show_cases = TRUE)
- Ensure x-axis value labels reflect breaks based on week, as opposed to something like the first of the month for example (labels_Week = TRUE)
- Add a black border around each of your cases (border=”black”)
- Specify your y-axis scale (from values of 0 to 3 by 1)
- Set colours for each level of infectiousness as follows: green=low; yellow=moderate; orange=high; red=very high (using scale_fill_manual).
- Change the title of the x-axis to “ISO week of symptom onset”
	
	`plot(i.7, show_cases = TRUE, border = "black", labels_week = TRUE) +`
	`  my_theme +`
	`  scale_y_continuous(breaks=seq(0, 3, 1)) +`
	`  scale_fill_manual(values=c("green", "yellow", "orange", "red")) +`
	`  labs(x = "ISO week of symptom onset" , y = "Case count", fill="Infectiousness") +`
	`  coord_fixed(ratio = 7)`

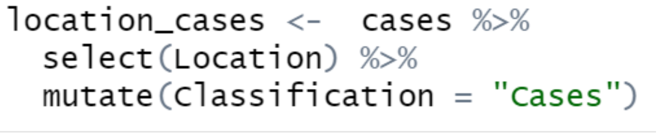
and the y-axis to “case count” (using labs function) - Label your legend “Infectiousness” under the label function (fill=’nfectiousness’) - Represent individual cases as squares, accounting for 7 days per square along x-axis (using coord_fixed(ratio = 7))	
Save the figure as a picture (.jpeg) in your output folder using ggsave. Name it: plot_cases_epiweek.	 `ggsave(filename = paste0(output_folder, "/plot_cases_epiweek.jpeg"), width = 7, height = 2)`
Save your script as 05_1_plot_epi_curve	

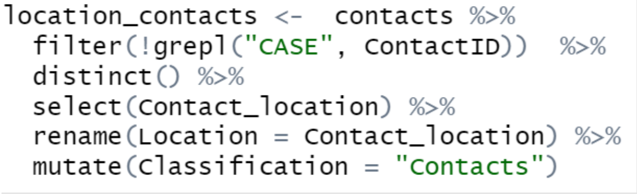


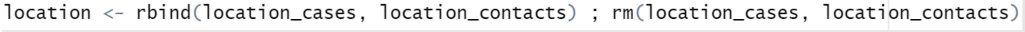
7. Bonus task 2

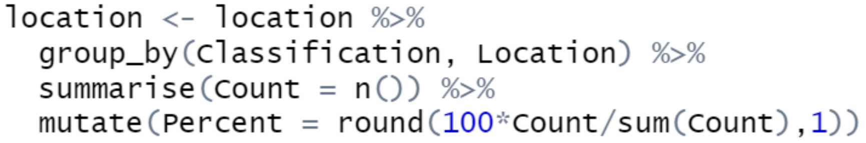
Create a table showing location of cases and contacts on and off reserve:

Task	Code
Open a new script and save it in your Exercise_Day2/scripts folder as “05_2_tab_location.R”	

You will need to wrangle the data first!
Create a new data frame called location_cases that contains two columns: Location (on or off reserve) and Classification (case, contact).
Extract the location column from the cases data frame (using select) and create a new column called "Classification" (using mutate).
	
	`location_cases <-  cases %>%`
	`  select(Location) %>%`
	`  mutate(Classification = "Cases")`

Create a contact data frame called location_contacts with the same columns you created for cases above (Classification and Location)
Exclude all rows that contain "CASE" text string in the ContactID (using filter and !grepl). Do you remember what these functions do?
	
	`location_contacts <-  contacts %>%`
	`  filter(!grepl("CASE", ContactID))  %>%`
	`  distinct() %>%`
	`  select(Contact_location) %>%`
	`  rename(Location = Contact_location) %>%`
	`  mutate(Classification = "Contacts")`

Select only unique rows (using distinct) and extract only the location column (using select). Rename contact_location to location (using rename) so that it matches the variable name you created for the location_cases data. Recreate a new column called "Classification" and have it read "Contacts" for all rows (using mutate).	
Combine the two data frames you have just created using the bind function so that you have case and contact locations in one column. Reminder: You can use the rm function to remove the no longer needed data frames (location_cases, location_contacts) to clean up your workspace.	 `location <- rbind(location_cases, location_contacts) ; rm(location_cases, location_contacts)`

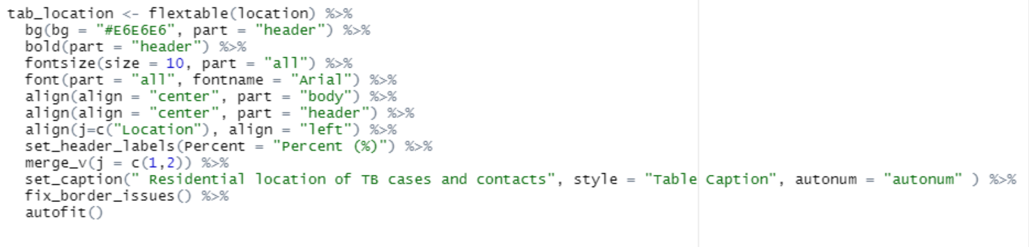
Summarize new location list into a frequency table with count and percent of cases and contacts on and off reserve.
Use the group_by and summarize functions to group and total the data by classification and location. Create a column with the % location (using mutate).
	
	`location <- location %>%`
	`  group_by(Classification, Location) %>%`
	`  summarise(Count = n()) %>%`
	`  mutate(Percent = round(100*Count/sum(Count),1))`

<table>
<tr>
<td>

Display the frequency table (calling it tab_location) in a nicely formatted flextable with the following options:

- Set the background colour of the header to grey (#E6E6E6) (using bg)
- Make the header bold (using bold)
- Make the font size 10 in all parts of the table (using fontsize)
- Change the font type to arial (using font)
- Make all the text (in the body and header) centre-justified (using align)
- Make the first column (Location) left aligned (using align)
- Change the “Percent” column heading to “Percent (%)” (using set_header_labels)
- Vertically merge duplicate rows (which are columns 1 and 2), so they don't repeat (using merge_v)
- Add the title: “Residential location of TB cases and contacts” to the figure (using set_caption). Set the style to “Table Caption” and use autonum so that the caption title for this figure will appear as a numbered caption in Word.
- Merging can sometimes remove outer border lines. Use the fix_border_issues to correct.
- Create nicely spaced column widths (using autofit).

</td>
<td>

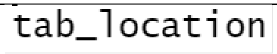


```
tab_location <- flextable(location) %>%
  bg(bg = "#E6E6E6", part = "header") %>%
  bold(part = "header") %>%
  fontsize(size = 10, part = "all") %>%
  font(part = "all", fontname = "Arial") %>%
  align(align = "center", part = "body") %>%
  align(align = "center", part = "header") %>%
  align(j=c("Location"), align = "left") %>%
  set_header_labels(Percent = "Percent (%)") %>%
  merge_v(j = c(1,2)) %>%
  set_caption(" Residential location of TB cases and contacts", style = "Table Caption", autonum = "autonum" ) %>%
  fix_border_issues() %>%
  autofit()
```

</td>
</tr>
<tr>
<td>

Print the table

</td>
<td>



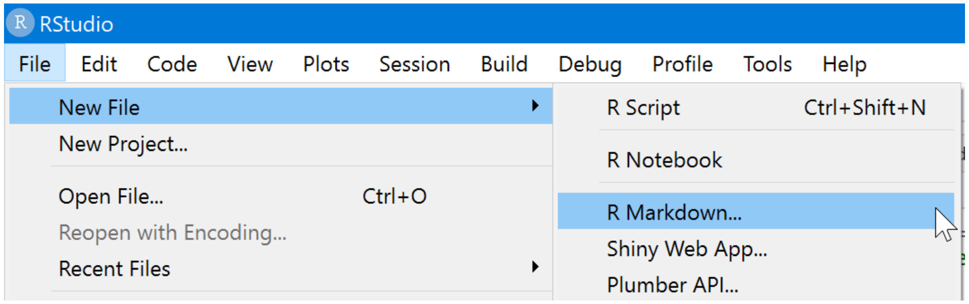
```
tab_location
```

</td>
</tr>
</table>

Tidy your work environment (using rm) to remove unneeded objects.	`rm(location)`
Ensure a heading is included in your script with details regarding the purpose, author, date, modifications, and other pertinent notes.	
Save your script as “05_2_tab_location.R”	

8. Create an automated report

Create a new R Markdown file by navigating to
File > New File > R Markdown



Select the default Output Format: **Word**
Click _> OK
Save this file as "**Day2_final.Rmd**" in your scripts folder.

Note: Locate and review the R Markdown cheat sheet and handout provided to you in the course materials. Keep on hand for use as a reference in this section.

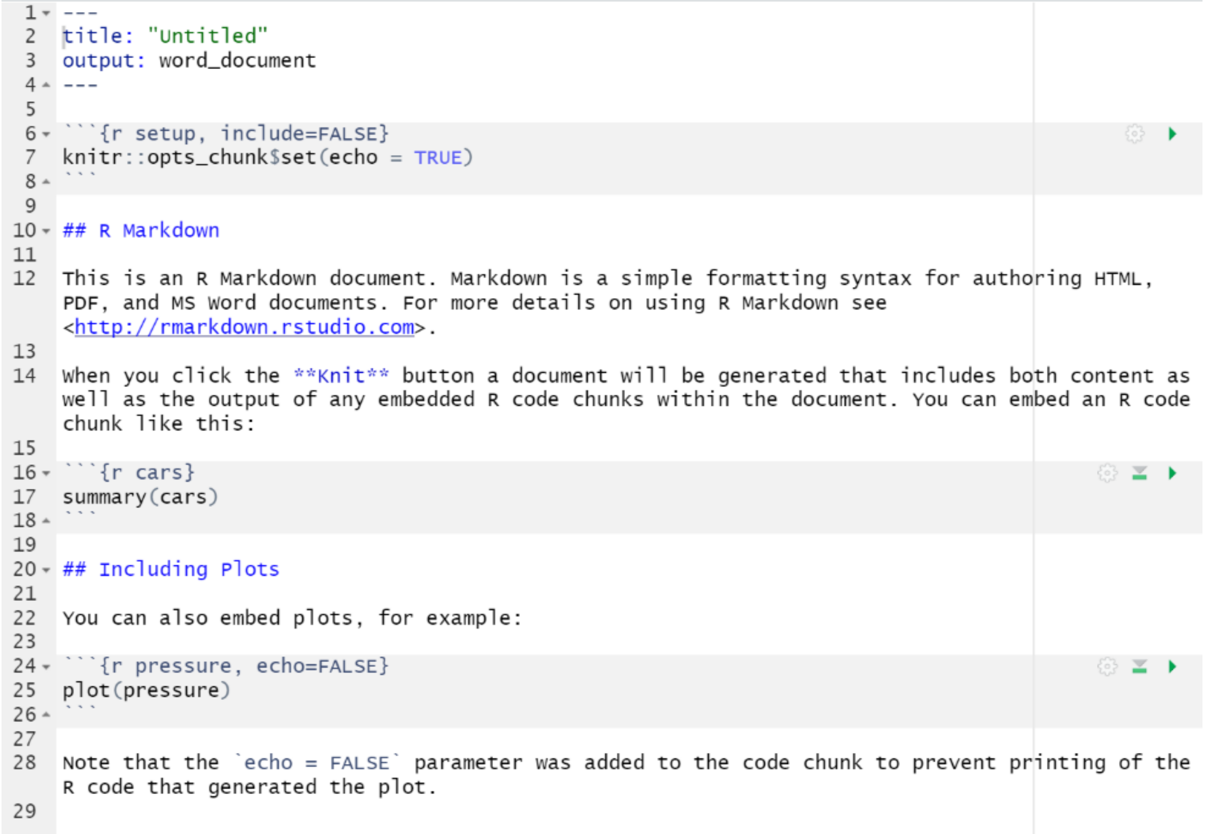
<table>
<tr><th>Task</th><th>Code</th></tr>
<tr>
<td>

Review the newly created R markdown template file:

- What type of document will be created from this code?
- What does the ```{r} at the beginning of the code chunk signify?
- What does the ``` at the end of the code chunk signify?
- What happens when you press the knit button Knit and save the resulting document?
- How does the text from this R markdown template now appear in the resulting file? How are titles added? Hyperlinks? How is text bolded?

See handouts: R Markdown Reference Guide and R Markdown Cheat Sheet.

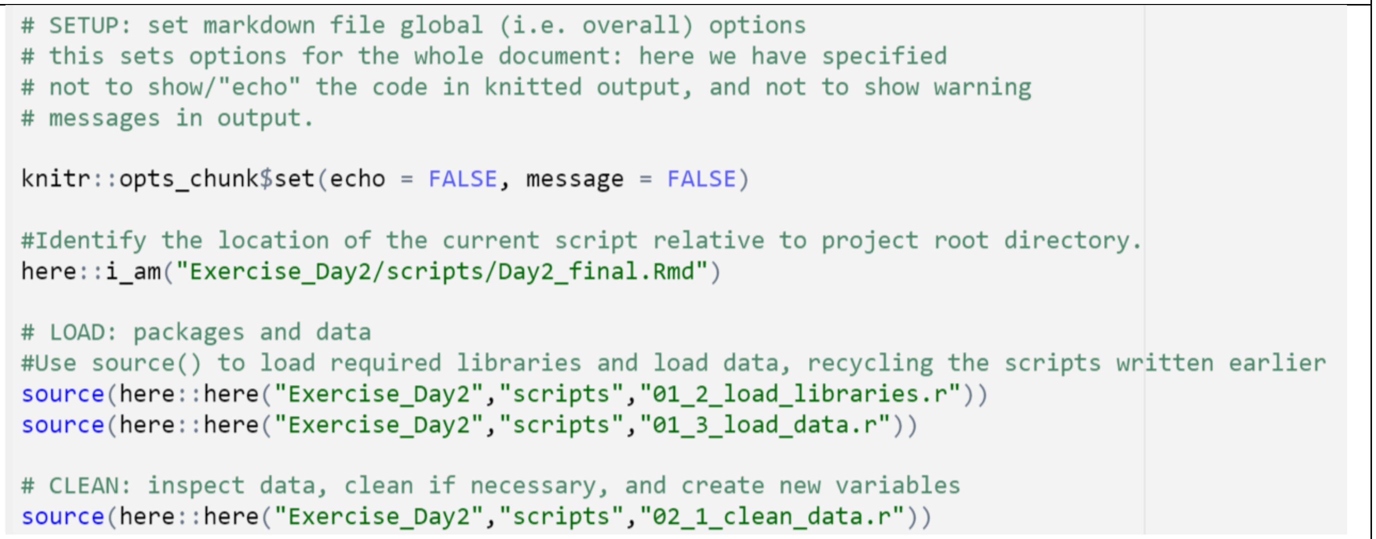
</td>
<td>



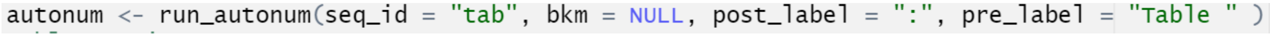
~~~
---
title: "Untitled"
output: word_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```{r cars}
summary(cars)
```

## Including Plots

You can also embed plots, for example:

```{r pressure, echo=FALSE}
plot(pressure)
```

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

~~~

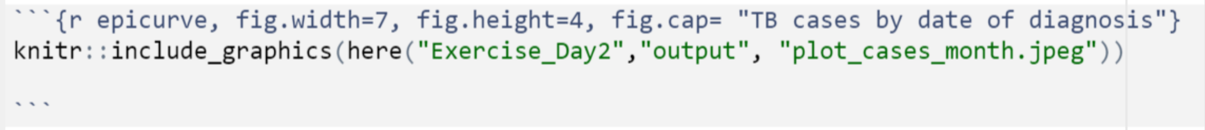
</td>
</tr>
</table>

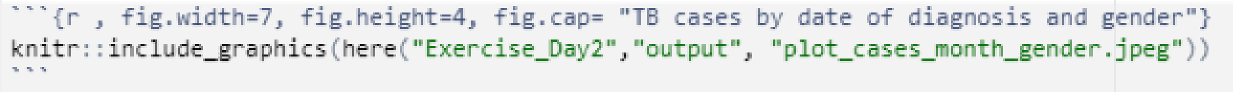
Close the new markdown file. Open and review Day2.Rmd What is the purpose of the source() function[3]?	
Add two extra lines to the title block following date: 1. Modified by: [add your name] 2. Date modified: [add the date]	

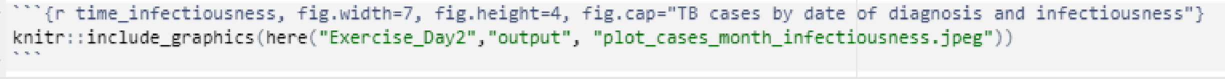
Review and use the code in the {r setup} chunk to the right as an example to:
a) set the location for the here() package
b) load the libraries and data using source() to access the scripts you've already written
c) clean the data by using source() to access your data-cleaning script.
	
	# SETUP: set markdown file global (i.e. overall) options
	# this sets options for the whole document: here we have specified
	# not to show/"echo" the code in knitted output, and not to show warning
	# messages in output.
	
	knitr::opts_chunk$set(echo = FALSE, message = FALSE)
	
	#Identify the location of the current script relative to project root directory.
	here::i_am("Exercise_Day2/scripts/Day2_final.Rmd")
	
	# LOAD: packages and data
	#Use source() to load required libraries and load data, recycling the scripts written earlier
	source(here::here("Exercise_Day2","scripts","01_2_load_libraries.r"))
	source(here::here("Exercise_Day2","scripts","01_3_load_data.r"))
	
	# CLEAN: inspect data, clean if necessary, and create new variables
	source(here::here("Exercise_Day2","scripts","02_1_clean_data.r"))

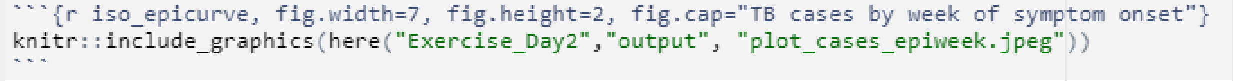
[3] For more information on how to source scripts in R: https://www.earthdatascience.org/courses/earth-analytics/multispectral-remote-sensing-data/source-function-in-R/

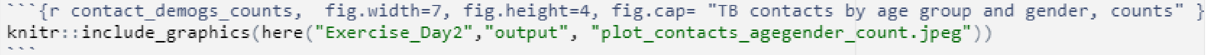
Explain in your own words what the code to the right is doing.	 autonum <- run_autonum(seq_id = "tab", bkm = NULL, post_label = ":", pre_label = "Table ")

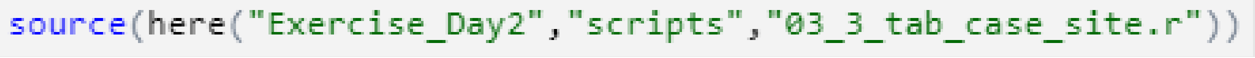
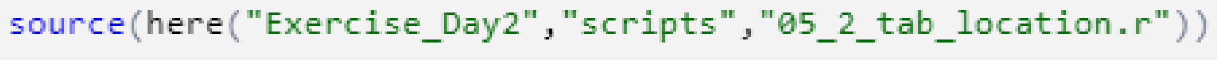
If necessary, change the code to the right to reflect the figure you created in plotting cases by month.	
	```
	{r epicurve, fig.width=7, fig.height=4, fig.cap= "TB cases by date of diagnosis"}
	knitr::include_graphics(here("Exercise_Day2","output", "plot_cases_month.jpeg"))
	```

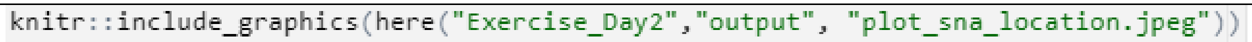
If necessary, change the code to the right to reflect the figure you created in plotting cases by month and gender.	
	```
	{r , fig.width=7, fig.height=4, fig.cap= "TB cases by date of diagnosis and gender"}
	knitr::include_graphics(here("Exercise_Day2","output", "plot_cases_month_gender.jpeg"))
	```

If necessary, change the code to the right to reflect the figure you created in plotting cases by month and infectiousness.	
	```
	{r time_infectiousness, fig.width=7, fig.height=4, fig.cap="TB cases by date of diagnosis and infectiousness"}
	knitr::include_graphics(here("Exercise_Day2","output", "plot_cases_month_infectiousness.jpeg"))
	```

If you created the bonus epicurve, use the code to the right to reflect the figure you created, otherwise delete this line.	
	```
	{r iso_epicurve, fig.width=7, fig.height=2, fig.cap="TB cases by week of symptom onset"}
	knitr::include_graphics(here("Exercise_Day2","output", "plot_cases_epiweek.jpeg"))
	```

If necessary, change the code to the right to reflect the figure you created in plotting contacts by age.	
	```
	{r contact_demogs_counts,  fig.width=7, fig.height=4, fig.cap= "TB contacts by age group and gender, counts" }
	knitr::include_graphics(here("Exercise_Day2","output", "plot_contacts_agegender_count.jpeg"))
	```

If necessary, change the code to the right to reflect the script you created to analyse sites.	 source(here("Exercise_Day2","scripts","03_3_tab_case_site.r"))
If you created the bonus table for location, use the	 source(here("Exercise_Day2","scripts","05_2_tab_location.r"))

code to the right to reflect the script you created, otherwise delete this line.	
If necessary, change the code to the right to reflect the figure you created in plotting the social network diagram.	 `knitr::include_graphics(here("Exercise_Day2","output", "plot_sna_location.jpeg"))`
Bonus! Edit the text for the word document in the R markdown file to reflect your observations.	
Rename and save the R markdown document. Include pertinent details in a heading.	
Knit the word document	

What edits would you make to this new word document based on how you coded figures and analyses?

Would you prefer to code your analyses all together in a single markdown file, or use source scripts like we did for the exercises in Day 1? Why would you choose your preferred approach over the other option?