GY7702 Assignment

MSc Satellite Data Science at the University of Leicester

Elliot Greatrix

04/11/2020

Introduction

This document was created to meet the requirements of GY7702 R for Data Science at University of Leicester. It was designed and created in R Markdown, a markup language that allows users to create documents that can be formatted to embed code blocks, code outputs and hyperlinks. When the R Markdown file is compiled, the markup language is hidden and the document is displayed in plain text.

This content was created using R, Rstudio, RMarkdown and GitHub

Materials

The libraries used in this assignment:

```
library(tidyverse)
library(palmerpenguins)
library(dplyr)
library(knitr)
library(kableExtra)
library(readr)
library(tinytex)
library(lubridate)
library(gridExtra)
library(ggplot2)
```

For further information regarding the source code, data and libraries used in this assignment, please see the projects GitHub page.

References

Elliot would like to acknowledge that this document includes teaching materials from Dr Stefano De Sabbato for the module GY7702 R for Data Science. Dr Stefanos teaching materials can be found here

R for Data Science by Garrett Grolemund and Hadley Wickham, O'Reilly Media, 2016. See online book

Note to reader

Where a function is mentioned for the **first time**, a brief explanation will be given before the code block. Thereafter, this explanation will be dismissed.

Question 1

A vector of 25 numbers between 1 and 7 are listed below. These values represent answers to a survey question

NA 3 4 4 5 2 4 NA 6 3 5 4 0 5 7 5 NA 5 2 4 NA 3 3 5 NA

- 1 = Completely disagree
- 2 = Disagree
- 3 = Somehow disagree
- 4 = Neither agree or disagree
- 5 = Somehow agree
- 6 = Agree
- 7 = Completely agree
- NA = missing value

Question 1.1

Write the code necessary to check whether all participants to the survey either completely disagree or completely agree, once the missing values are excluded

```
# First, let's create the vector.
num <- c(NA, 3, 4, 4, 5, 2, 4, NA, 6, 3, 5, 4, 0
         , 5, 7, 5, NA, 5, 2, 4, NA, 3, 3, 5, NA)

# Then, remove missing values.
num <- na.omit(num)

# And finally check whether all students either completely agree (= 7).
students_agree_complete <- all(num == 7)
```

```
## [1] FALSE
```

```
# Or completely disagree (=1).
students_disagree_complete <- all(num == 1)
```

```
## [1] FALSE
```

Question 1.2

Write the code necessary to extract the indexes related to the participants in the survey who at least somehow agree or more.

In the vector `num`, `which` students selected values between 5 and 7.

```
students_agree <- which(num %in% c(5:7))
```

```
## [1]  4  7  9 12 13 14 15 20
```

Question 2

Question 2.2

Write the code necessary to create a table showing species, island, bill length and body mass of the 10 Gentoo penguins in the penguins table with the highest body mass

The library palmerpenguins was created by Dr. Kristen Gorman and holds 2 data sets. For this assignment, we will work with `penguins`, a table containing 8 measured variables from 344 penguins!

Some of the penguins in `penguins` hold NA values for certain variables. Due to the analysis that we are going to do, these need to be removed

```
penguin_na_omitted <- na.omit(penguins)
```

As this task requires multiple steps, it will be more efficient to start a pipe. Lets send the *na omitted* table, `pen` down the pipe.

It is important to note that the functions in the pipe operator do not require the first argument, `pen` because it is being 'fed down' the pipe. For more information on pipes, click here

```
gentoo_heaviest <- penguin_na_omitted %>%
  # First, let's select all the variables that the question asks for
  dplyr::select(
    species, island, bill_length_mm, bill_depth_mm, body_mass_g
    )%>%

  # and select all the 'Gentoo' penguins.
  dplyr::filter(
    species == "Gentoo"
    )%>%

  # Order body mass in ascending order
  dplyr::arrange(
    -body_mass_g
    )
```

We can also use `kable()` to produce well-formatted output tables

```
  # Return the 10 heaviest Gentoo penguins
kable(gentoo_heaviest[1:10,], format = "latex", booktabs = T) %>%
kable_styling(latex_options = c("striped", "scale_down"))
```

species	island	bill_length_mm	bill_depth_mm	body_mass_g
Gentoo	Biscoe	49.2	15.2	6300
Gentoo	Biscoe	59.6	17.0	6050
Gentoo	Biscoe	51.1	16.3	6000
Gentoo	Biscoe	48.8	16.2	6000
Gentoo	Biscoe	45.2	16.4	5950
Gentoo	Biscoe	49.8	15.9	5950
Gentoo	Biscoe	48.4	14.6	5850
Gentoo	Biscoe	49.3	15.7	5850
Gentoo	Biscoe	55.1	16.0	5850
Gentoo	Biscoe	49.5	16.2	5800

Question 2.3

Write the code necessary to create a table showing the average bill length per island, ordered by average bill length.

As this question states `per island`, we will use `group_by` to perform an average bill length for each island. We will also see `mean()` to determine average. This is a function built into R.

After the data set is grouped by island, average bill length will be calculated. The result will be appended to `pen` in a column called `avg_bill_length` using the `mutate()` function.

```r
penguin_avg_bill_length <- penguin_na_omitted %>%

  # Select the appropriate attributes
  dplyr::select(
    species, island, bill_length_mm, bill_depth_mm
    )%>%

  # Order by bill_length_mm. By default arrange sorts in ascending order.
  dplyr::arrange(
    bill_length_mm
    )%>%

  # Use group_by to execute the average for each island
  dplyr::group_by(
    island
    )%>%

  # Append the average bill_length_mm to the table pen
  dplyr::mutate(
    avg_bill_length_mm = mean(bill_length_mm)
    )%>%

  # Display 3 example average_bill_lengths for each island
  slice_head(n = 3)
```

species	island	bill_length_mm	bill_depth_mm	avg_bill_length_mm
Adelie	Biscoe	34.5	18.1	45.24847
Adelie	Biscoe	35.0	17.9	45.24847
Adelie	Biscoe	35.0	17.9	45.24847
Adelie	Dream	32.1	15.5	44.22195
Adelie	Dream	33.1	16.1	44.22195
Adelie	Dream	34.0	17.1	44.22195
Adelie	Torgersen	33.5	19.0	39.03830
Adelie	Torgersen	34.4	18.4	39.03830
Adelie	Torgersen	34.6	21.1	39.03830

Question 2.4

Write the code necessary to create a table showing the minimum, median and maximum proportion between bill length and bill depth by species.

The *proportion between bill length and bill depth* will be calculated as `bill_length_mm` / `bill_depth_mm`. This will return a unit-less measure which will be called `bl_bd`

R also contains inbuilt functions for obtaining the minimum, median and maximum values from a vector. So by example after `select()` and `group_by()`, lets take the minimum `bill_length_mm` and minimum `bill_depth_mm` and divide them to get the unit-less measurement `min_bl_bd`.

```
penguin_na_omitted %>%
# Select the necessary attributes
dplyr::select(
  species, island, bill_length_mm, bill_depth_mm
  )%>%

# Execute the proportions for each species
dplyr::group_by(
  species
  )%>%

# Append the proportions to the table pen, using mutate
dplyr::mutate(
  min_bl_bd = min(bill_length_mm) / min(bill_depth_mm)
  )%>%
```

Then repeat this for the median and maximum values

```
dplyr::mutate(
  med_bl_bd = median(bill_length_mm) / median(bill_depth_mm)
  )%>%

dplyr::mutate(
  max_bl_bd = max(bill_length_mm) / max(bill_depth_mm)
  )%>%
```

```
# Drop bill_length_mm and bill_depth_mm
dplyr::subset(
  select = -c(bill_length_mm, bill_depth_mm)
  )
```

This is a preview of 3 penguins from each island, demonstrating how the proportions are calculated *per island*.

species	island	min_bl_bd	med_bl_bd	max_bl_bd
Adelie	Torgersen	2.070968	2.111413	2.139535
Adelie	Torgersen	2.070968	2.111413	2.139535
Adelie	Torgersen	2.070968	2.111413	2.139535
Chinstrap	Dream	2.493902	2.685637	2.788461
Chinstrap	Dream	2.493902	2.685637	2.788461
Chinstrap	Dream	2.493902	2.685637	2.788461
Gentoo	Biscoe	3.122137	3.160000	3.445087
Gentoo	Biscoe	3.122137	3.160000	3.445087
Gentoo	Biscoe	3.122137	3.160000	3.445087

Question 3

Question 3.1

Write the code necessary to load the data from *covid19_cases_20200301_20201017.csv* to a variable named covid_data

The library `readr` contains the function `read_csv()` which allows us to read .csv (comma-separated value) files into R.

```
# Reading the COVID-19 .csv from a local directory
covid_data <-
  readr::read_csv(
    "C:/.../data/covid19_cases_20200301_20201017.csv"
    )
```

Question 3.2

Write the code necessary to

- Create a complete table, containing a row for each day and area
- Replace NA values with the value available for the previous date
- Replace the remaining NA values with zero
- Subset only the area assigned to me (Dudley)

- Drop the `area_name` column
- Store the resulting table in a variable named `Dudley_complete_covid_data`

First, lets check how the data is delivered

specimen_date	area_name	newCasesBySpecimenDate	cumCasesBySpecimenDate
2020-03-01	Aberdeen City	0	0
2020-03-01	Aberdeenshire	0	0
2020-03-01	Angus	0	1
2020-03-01	Antrim and Newtownabbey	0	0
2020-03-01	Ards and North Down	0	0
2020-03-01	Argyll and Bute	0	0
2020-03-01	Armagh City, Banbridge and Craigavon	0	0
2020-03-01	Barking and Dagenham	1	1
2020-03-01	Barnet	0	1
2020-03-01	Belfast	0	0

Next we want all areas to have a date, even if that area does not have any cases on that date. We can do this by using `complete()` . This will lead to NA values. Using `fill()` , we can fill `Na` values with the next non `NA` value that is immediately above the `NA` value. This is called a `down` fill, and is the default of the `fill()` function.

We can `replace()` Na values with 0, filter for our area (Dudley) and finally drop the `area_name` column.

```
Dudley_complete_covid_data <-
  covid_data %>%

  # Create a complete table
  tidyr::complete(
    specimen_date, area_name
    )%>%
  dplyr::group_by(
    area_name
    )%>%
  dplyr::arrange(
    specimen_date
    )%>%

  # Fill newCases and cumCases with cases from the previous day
  tidyr::fill(
    newCasesBySpecimenDate, cumCasesBySpecimenDate
    )%>%

  # Set any other NA values to 0
  tidyr::replace_na(
    list(newCasesBySpecimenDate = 0)
    )%>%

  tidyr::replace_na(
    list(cumCasesBySpecimenDate = 0)
    )%>%

  # Filter out the area 'Dudley'
  filter(
```

```
    area_name == "Dudley"
    )%>%

  # Drop the area_name column
  subset(
    select = -c(area_name)
    )
```

specimen_date	newCasesBySpecimenDate	cumCasesBySpecimenDate
2020-03-01	0	0
2020-03-02	0	0
2020-03-03	0	0
2020-03-04	0	0
2020-03-05	0	0
2020-03-06	0	0
2020-03-07	1	1
2020-03-08	0	1
2020-03-09	2	3
2020-03-10	1	4

Question 3.3

Starting from the table Dudley_complete_covid_data created for Question 3.2:

- Create a copy of `Dudley_complete_covid_data` i.e., as another variable named `Dudley_day_before`
- Use the library lubridate to create a new column named `day_before` in the new table `Dudley_day_before` that reports the day before the day reported in the column `specimen_date`
- Drop the `specimen_date` and `cumCasesBySpecimenDate` columns from the `Dudley_day_before` table
- Rename the `newCasesBySpecimenDate` column of the the `Dudley_day_before table` to `newCases_day_before`
- Join [area]_day_before with [area]_complete_covid_data, where the column specimen_date of [area]_complete_covid_data is equal to the column `day_before` of `Dudley_day_before`
- Calculate a new column in the joined table, containing the number of new cases as a percentage of the number of new cases of the day before
- Store the resulting table in a variable named `Dudley_covid_development`

```
# Create a copy of '[area]_complete_covid_data', i.e., as another variable
# named '[area]_day_before'.
Dudley_day_before <-
  Dudley_complete_covid_data

# create a new column named day_to_match
Dudley_day_before$day_to_match <-
  Dudley_day_before$specimen_date

# Subtract 1 day from the day_to_match column
```

```
Dudley_day_before$day_to_match <-
  as.Date(
    Dudley_day_before$day_to_match
    )+1
```

If the above section contained one more step, we would have given serious thought of how to put it into a `pipe`. The next section however is a lot more complex, and the `pipe` makes it more readable and efficient. We'll carry out the operation first and then explain the output below.

```
Dudley_covid_development <-
  Dudley_day_before %>%

# Drop the `specimen_date` and `cumCasesBySpecimenDate` columns from the
# `[area]_day_before` table
subset(
  select = -c(specimen_date, cumCasesBySpecimenDate)
  ) %>%

# Rename the `newCasesBySpecimenDate` column of the the `[area]_day_before`
# table to `newCases_day_before`
dplyr::rename(
  newCases_day_before = newCasesBySpecimenDate
  ) %>%

# Join `[area]_day_before` with `[area]_complete_covid_data`
# where the column `specimen_date` of `[area]_complete_covid_data` is equal to
# the column `day_to_match` of `[area]_day_before
dplyr::left_join(
  Dudley_complete_covid_data,.,
  by= c("specimen_date" = "day_to_match")
  ) %>%

# number of new cases as a percentage of the number of new cases of the day
# before
dplyr::mutate(
  percentage_of_new_cases = (newCasesBySpecimenDate/newCases_day_before)*100
  )%>%

# Remove any trailing 0's from percentage_of_new_cases
dplyr::mutate(
  percentage_of_new_cases = round(
    percentage_of_new_cases, 0
    )
  ) %>%

# If there is any infinite values in the table, set these to NA
dplyr::mutate_if(
  is.numeric, list(~na_if(., Inf))
  )
```

specimen_date	newCasesBySpecimenDate	cumCasesBySpecimenDate	newCases_day_before	percentage_of_new_cases
2020-03-01	0	0	NA	NA
2020-03-02	0	0	0	NaN
2020-03-03	0	0	0	NaN
2020-03-04	0	0	0	NaN
2020-03-05	0	0	0	NaN
2020-03-06	0	0	0	NaN
2020-03-07	1	1	0	NA
2020-03-08	0	1	1	0
2020-03-09	2	3	0	NA
2020-03-10	1	4	2	50

`newCases_day_before` states the cases the day before `specimen_date` . `newCasesBySpecimenDate` states the cases on the `specimen_date`. This allows `newCases_day_before/newCasesBySpecimenDate` to efficiently calculate change in COVID cases over 2 days, across 1 row.

Some issues arose when the next day (`newcases_day_before`) contained 0 cases. Dividing by 0 is `inf` or undefined, so these results were set to `NA` . These were not converted to 0 because `inf != 0`.

Using `library(ggplot2)` and `library(gridExtra)` we can visualise the change in cases over `specimen_date` .The syntax here is quite self-explanatory, `geom_line` just specifies that it is a line plot, instead of a scatter (`geom_point`) for example.

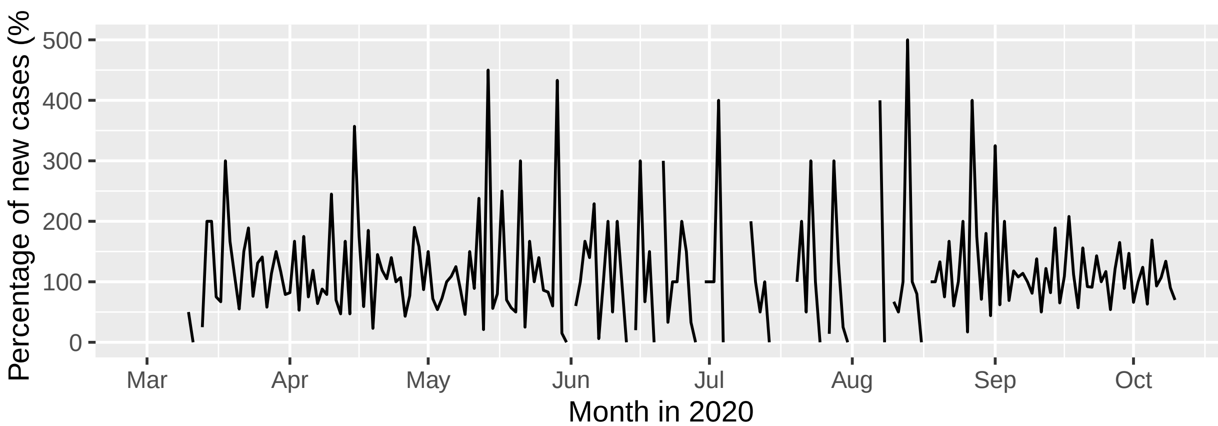
For transparency, I will comment the `ggplot()` code below once

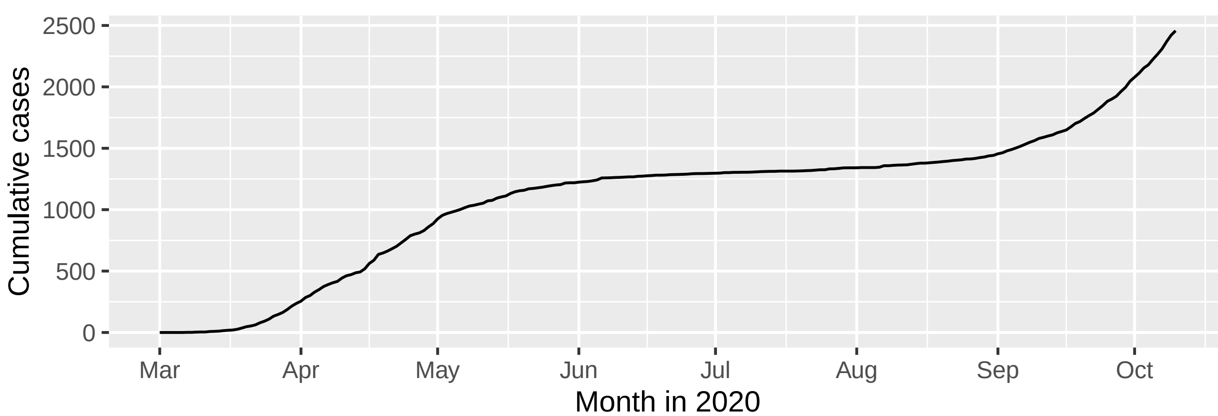
```r
percentage_change <- ggplot2::ggplot(

  # Specify the data
  data = Dudley_covid_development, mapping = aes(

    # Specify x axis and x axis format
    x = specimen_date,
    format = "%Y-%m-%d",

    # Specify y axis
    y = percentage_of_new_cases)) +

  # Specify the connection of observations along the x axis
  geom_path() +

  # Scale the x axis monnthly
    scale_x_date(
    date_breaks ="1 month",

    # Specify Month names, not month numbers (change to m for numbers)
    date_labels = "%b") +

  # Label x and y axis appropriately
  labs(
    x = "Month in 2020",
    y = "Percentage of new cases (%)")


# Repeat for cumulative graph
cumulative <- ggplot2::ggplot(
  data = Dudley_covid_development, mapping = aes(
```

```
    x = specimen_date,
    format = "%Y-%m-%d",
    y = cumCasesBySpecimenDate)) +
  geom_path() +
    scale_x_date(
    date_breaks ="1 month",
    date_labels = "%b") +
  labs(
    x = "Month in 2020",
    y = "Cumulative cases")
```

To display the 2 plots above each other we can use `gridExtra()`

```
gridExtra::grid.arrange(
  percentage_change,cumulative
  )
```





The first cases for Dudley were reported on 07/03/2020. The percentage of new cases fluctuated 50-200%, when compared to the previous day. There are outliers in this period on 18/03/2020 and 15/04/2020 where cases increased 300% and 360% respectively. The >100% daily rise in cases is highlighted in the cumulative cases which show a steady rise during the same period. This pattern continues throughout May, with a significant rise on 14/05/2020 to a 450% increase. The data in June and July presents a very different pattern. The percentage of new cases decreased below 100% more frequently. This caused a 'flattening' of the cumulative cases. Cases begin to rise again in August. The largest increase occurred on 13/08/2020 when cases increased 500%. This leads to a gradual rise in cumulative cases again throughout September, to a level similar to that of April.

Question 4

Analyse the COVID and population data as you see fit

First, lets read in the population data

```
# Read in the population data from a local directory
pop <- readr::read_csv("C:/.../data/lad19_population.csv")
```

For a suitable comparison, we need to join `pop` and `covid_data`. They hold 2 common columns that state the name of the area. However in `covid_data` this column is called `area_name` and in `pop` it's called `lad19_area_name`. Lets rename `lad19_area_name` to `area_name` for an easier join.

Note: Generally, renaming variables is bad coding practice. However, because this is a relatively small data set and this is a fairly heuristic study a `rename()` serves it's purpose

specimen_date	area_name	newCasesBySpecimenDate	cumCasesBySpecimenDate	lad19_area_code	area_population
2020-03-01	Aberdeen City	0	0	S12000033	250000
2020-03-01	Aberdeenshire	0	0	S12000034	272728
2020-03-01	Angus	0	1	S12000041	117648
2020-03-01	Antrim and Newtownabbey	0	0	N09000001	143565
2020-03-01	Ards and North Down	0	0	N09000011	166667
2020-03-01	Argyll and Bute	0	0	S12000035	85898
2020-03-01	Armagh City, Banbridge and Craigavon	0	0	N09000002	217392
2020-03-01	Barking and Dagenham	1	1	E09000002	213115
2020-03-01	Barnet	0	1	E09000003	396277
2020-03-01	Belfast	0	0	N09000003	344538

This is looking good! Covid data for an area, alongside its associated population data. Next, lets extract some areas from this dataset `cov_pop` and called it `places`.

```
places <- c("Dudley","Plymouth")
```

Using a `pipe` , we can extract the data for the areas in `places` , normalize cumulative cases by the population of an area and drop any columns we don't need.

```
Dudley_Plymouth_covid_data <-
  cov_pop %>%

  # Filtering rows out of cov_pop that match the values in places
  dplyr::filter(
    area_name %in% places
    ) %>%

  # Remove area_name and newcasesbyspeciendate
  subset(
    select = -c(lad19_area_code, newCasesBySpecimenDate)
    )%>%

  # Show cumulative cases as a percentage of the population of an area
  dplyr::mutate(
    CumCases_percentage_of_pop = (cumCasesBySpecimenDate / area_population)*100
    )%>%

  # Remove the area_population
  subset(
```

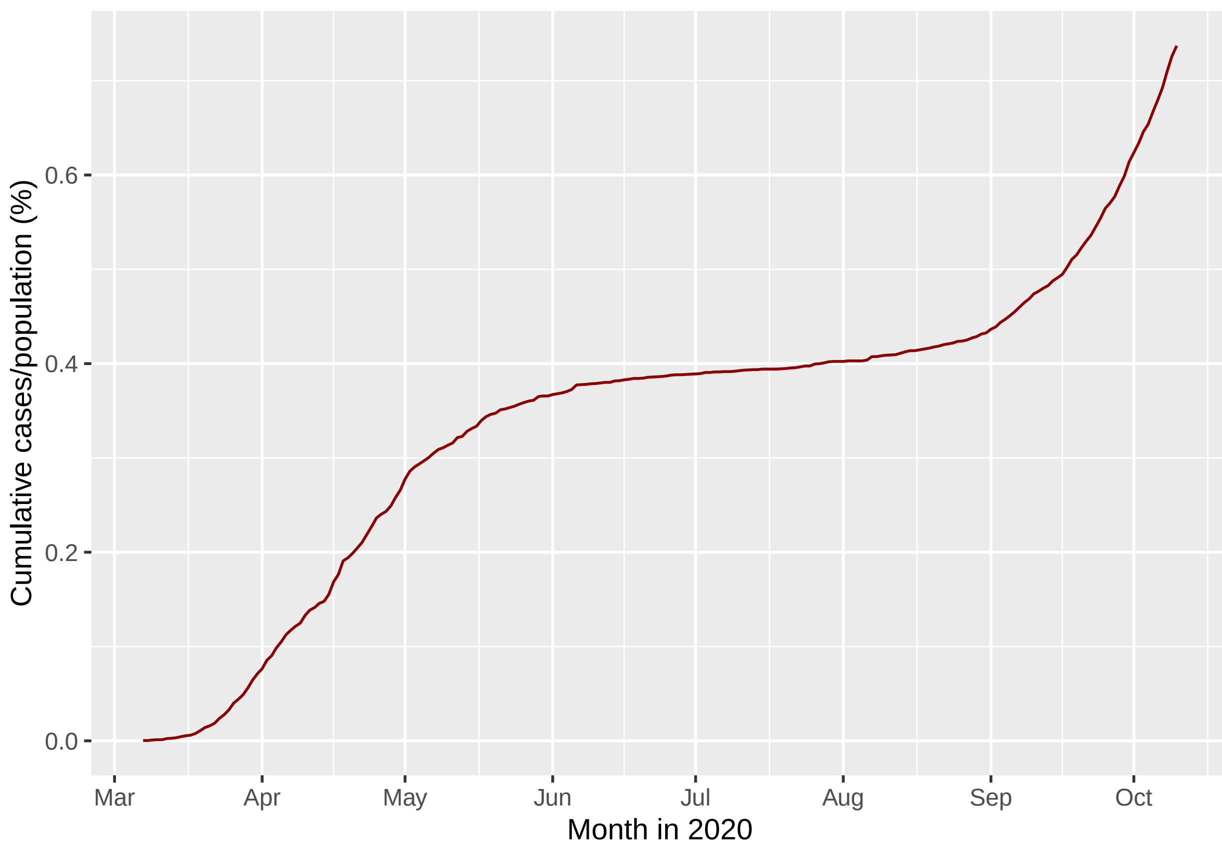
```
    select = -c(area_population, cumCasesBySpecimenDate)
    )
```

We can then show the cumulative cases for Dudley as a proportion of the population.

To show just Dudley, we need to `pivot_wider` to get `Dudley` in it's own column. Always good to have some practise with `pivot` as it can seem a bit unintuitive

`ggplot` then takes `Dudley_Plymouth_wide` as the data and the new column `Dudley` from the pivot to form the figure seen below.

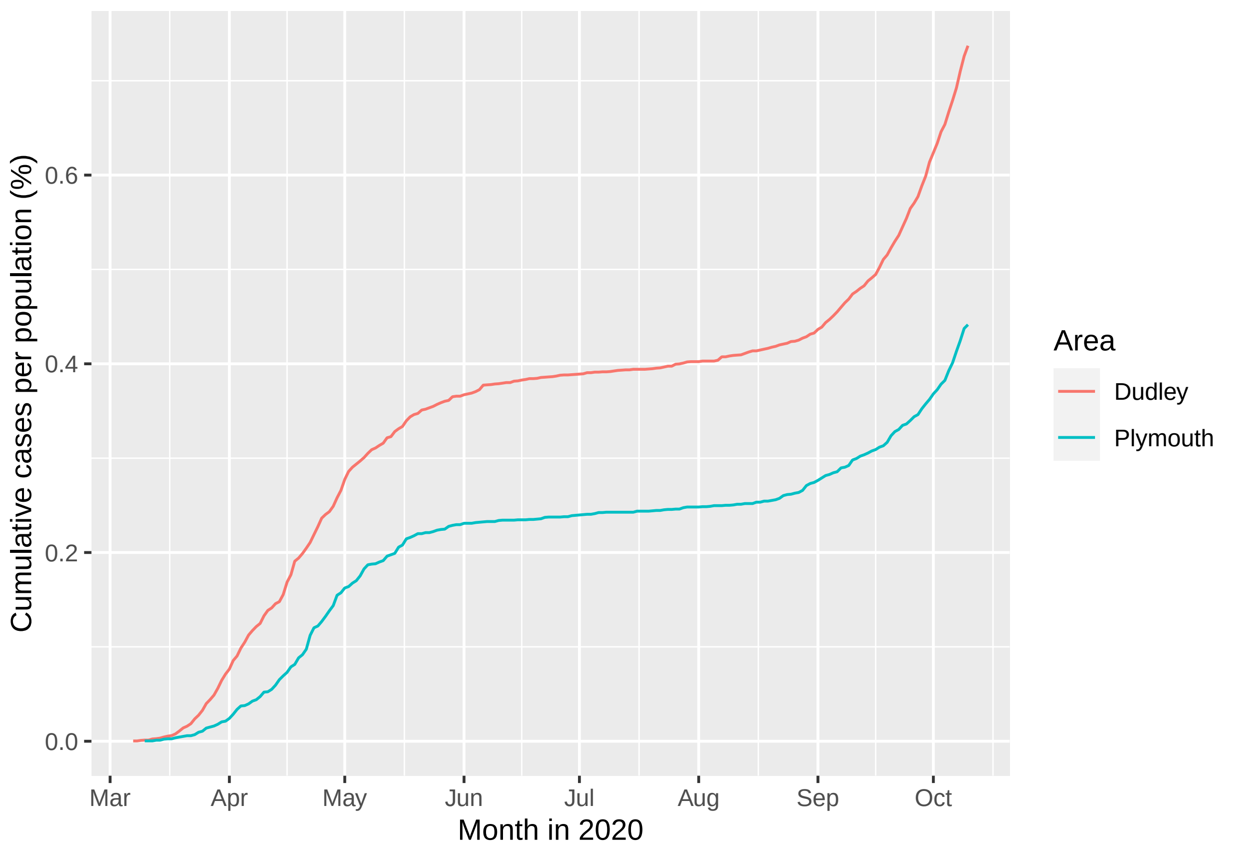
```
Dudley_Plymouth_wide <-
  tidyr::pivot_wider(
    Dudley_Plymouth_covid_data,

    # Form columns from the names in area_name
    names_from = area_name,

    # For those new columns, add in the values from CumCases_percentage_of_pop
    values_from = c(CumCases_percentage_of_pop)
    )
```



Interestingly, the original data set `Dudley_Plymouth_covid_data` was formatted perfectly for `ggplot` to visualise multiple areas on the same plot. This works because `ggplot` takes all the data and colours it by the `area_name` . `ggplot` is recognising 2 unique area names, and so they are coloured by 2 unique colours and displayed in an inbuilt legend.

Hence, we'll `pivot_long` to get back to our original data set

```
Dudley_Plymouth_long <-
  tidyr::pivot_longer(
    Dudley_Plymouth_wide,

    # Collapse all columns except specimen_date
    cols = -specimen_date,

    # Put the values that were in those columns, into a new column called
    # CumCases_percentage_of_pop
    names_to = c("CumCases_percentage_of_pop")
    )
```



"‘ The analysis above displays cumulative cases for Dudley and Plymouth as a percentage of the areas population. This is important as Dudley has a population of 333,334 while Plymouth has a population of 272,728. This was performed for a more honest assessment between the 2 areas. Dudley and Plymouth display cases in March. Dudley's first case is 07/03/10, compared to Plymouths later date of 10/03/2020. From this point, both areas display a very similar pattern in the rise of cases. That is, a sharp increase until the middle of May, followed by a flattened curve throughout June, July and August until cases increase again in September. A sharper rise, similar to March occur in October.

Though the pattern for both areas are the same, the proportion of cases when compared to the populations are very different. After the initial rise in April, 0.3% of Dudleys population has had a case. This is compared to 0.15% for Plymouth. Additionally, when the curve is flattened in the Summer months, ~0.4% of Dudley has has a case, compared to Plymouths ~0.25%. The second ‘spike’ in October only serves to widen this gap. The last data from both areas was 10/10/2020 where there was a ~0.3% difference between cumulative cases in Plymouth and Dudley.