Intro to Quarto and Tidyverse

Monica Alexander

10/01/24

Table of contents

By the end of this lab you should know the basics of

- RStudio Projects
- Quarto
- Main tidyverse functions
- `ggplot`

RStudio Projects

RStudio projects are associated with R working directories. They are good to use for several reasons:

- Each project has their own working directory, so make dealing with file paths easier
- Make it easy to divide your work into multiple contexts
- Can open multiple projects at one time in separate windows

To make a new project in RStudio, go to File –> New Project. If you've already set up a repo for this class, then select 'Existing Directory' and choose the folder that will contain all your class materials. This will open a new RStudio window, that will be called the name of your folder.

In future, when you want to do work for this class, go to your class folder and open the .Rproj file. This will open an RStudio window, with the correct working directory, and show the files you were last working on.

Quarto

This is a Quarto document. Quarto allows you to create nicely formatted documents (HTML, PDF, Word) that also include code and output of the code. This is good because it's reproducible, and also makes reports easier to update when new data comes in. Each of the grey chunks contains R code, just like a normal R script. You can choose to run each chunk separately, or knit the whole document using Knit the button above, which creates your document.

To start a new Quarto file in Rstudio, go to File –> New File –> Quarto, then select Document and whatever you want to compile the document as (I chose pdf, and that's generally what we'll be doing in this class). Notice that this and the other inputs (title, author) are used to create the 'yaml', the bit at the start of the document. You can edit this, like I have for example to include table of contents.

When you hit 'Render' a pdf will be created and saved in the same folder as your `qmd` file. There are various options for output code, results, etc. For example, if you don't want your final report to include the code (but just the output, e.g. graphs or tables) then you can specify `#| echo=FALSE` at the top of the chunk (note: this syntax is different R Markdown).

Quarto is a newer version of R Markdown. I'm fine if people would rather stick to R Markdown. A helpful intro is here: https://quarto.org/docs/get-started/hello/rstudio.html

Writing math

Writing equations is essentially the same as in LaTeX. You can write inline equations using the $ e.g. $y = ax + b$. You can write equations on a separate line with two $s e.g.

$$y = ax + b$$

In pdf documents you can have numbered equations using

$$y = ax + b \tag{1}$$

Getting greek letters, symbols, subscripts, bars etc is the same as LaTeX. A few examples are below

- $Y_{i,j}$
- $\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n}$
- $\alpha\beta\gamma$
- $X \rightarrow Y$
- $Y \sim N(\mu, \sigma^2)$

Tidyverse

Read in some packages that we'll be using:

```
#install.packages("tidyverse")
library(tidyverse)
```

On top of the base R functionality, there's lots of different packages that different people have made to improve the usability of the language. One of the most successful suite of packages is now called the 'tidyverse'. The tidyverse contains a range of functionality that help to manipulate and visualize data.

Read in mortality rates for Ontario. These data come from the Canadian Human Mortality Database.

```
dm <- read_table("https://www.prdh.umontreal.ca/BDLC/data/ont/Mx_1x1.txt", skip = 2, col_t
```

```
Warning: 494 parsing failures.
row    col               expected actual
108 Female no trailing characters      . 'https://www.prdh.umontreal.ca/BDLC/data/ont/Mx_1x1
109 Female no trailing characters      . 'https://www.prdh.umontreal.ca/BDLC/data/ont/Mx_1x1
```

```
110 Female no trailing characters      . 'https://www.prdh.umontreal.ca/BDLC/data/ont/Mx_1x1
110 Male   no trailing characters      . 'https://www.prdh.umontreal.ca/BDLC/data/ont/Mx_1x1
110 Total  no trailing characters      . 'https://www.prdh.umontreal.ca/BDLC/data/ont/Mx_1x1
... ...... ...................... ...... .............................................
See problems(...) for more details.
```

```
head(dm)
```

```
# A tibble: 6 x 5
   Year Age    Female    Male   Total
  <dbl> <chr>   <dbl>   <dbl>   <dbl>
1  1921 0     0.0978  0.129   0.114
2  1921 1     0.0129  0.0144  0.0137
3  1921 2     0.00521 0.00737 0.00631
4  1921 3     0.00471 0.00457 0.00464
5  1921 4     0.00461 0.00433 0.00447
6  1921 5     0.00372 0.00361 0.00367
```

The object `dm` is a data frame, or tibble. Every column can be a different data type (e.g. we have integers and characters).

Important tidyverse functions

You should feel comfortable using the following functions

- The pipe `|>` or `%>%`
- `filter`
- `select`
- `arrange`
- `mutate`
- `group_by`
- `summarize`
- `pivot_longer` and `pivot_wider`

Piping, filtering, selecting, arranging

A central part of manipulating tibbles is using the `|>` function. This is a pipe, but should be read as saying 'and then'. Note that the `|>` syntax is the base version of the pipe (new last year). Previously the syntax was `%>%` through the `magrittr` package. They essentially function the same.

For example, say we just want to pull out mortality rates for 1935. We would take our tibble *and then* filter to only include 1935:

```
dm |>
  filter(Year==1935) # two equals signs logical
```

```
# A tibble: 111 x 5
    Year Age     Female    Male   Total
   <dbl> <chr>    <dbl>   <dbl>   <dbl>
 1  1935 0     0.0513   0.0652  0.0584
 2  1935 1     0.00607  0.00742 0.00676
 3  1935 2     0.00350  0.00321 0.00336
 4  1935 3     0.00187  0.00321 0.00255
 5  1935 4     0.0013   0.00238 0.00185
 6  1935 5     0.00152  0.00186 0.00169
 7  1935 6     0.00136  0.00174 0.00155
 8  1935 7     0.00120  0.00154 0.00137
 9  1935 8     0.000984 0.00130 0.00114
10  1935 9     0.000996 0.00140 0.00120
# i 101 more rows
```

```
# we could say
filter(dm, Year==1935)
```

```
# A tibble: 111 x 5
    Year Age     Female    Male   Total
   <dbl> <chr>    <dbl>   <dbl>   <dbl>
 1  1935 0     0.0513   0.0652  0.0584
 2  1935 1     0.00607  0.00742 0.00676
 3  1935 2     0.00350  0.00321 0.00336
 4  1935 3     0.00187  0.00321 0.00255
 5  1935 4     0.0013   0.00238 0.00185
 6  1935 5     0.00152  0.00186 0.00169
 7  1935 6     0.00136  0.00174 0.00155
 8  1935 7     0.00120  0.00154 0.00137
 9  1935 8     0.000984 0.00130 0.00114
10  1935 9     0.000996 0.00140 0.00120
# i 101 more rows
```

You can also filter by more than one condition; say we just wanted to look at 10 year olds in 1935:

```r
dm |>
  filter(Year==1935, Age==10)
```

```
# A tibble: 1 x 5
   Year Age      Female    Male   Total
  <dbl> <chr>     <dbl>   <dbl>   <dbl>
1  1935 10    0.000884 0.00143 0.00116
```

If we only wanted to look at 10 year olds in 1935 who were female, we could filter *and then* select the female column.

```r
dm |>
  filter(Year==1935, Age==10) |>
  select(Female)
```

```
# A tibble: 1 x 1
    Female
     <dbl>
1 0.000884
```

You can also remove columns by selecting the negative of that column name.

```r
colnames(dm)
```

```
[1] "Year"   "Age"    "Female" "Male"   "Total"
```

```r
dm |>
  select(-Total)
```

```
# A tibble: 10,989 x 4
    Year Age    Female    Male
   <dbl> <chr>   <dbl>   <dbl>
 1  1921 0     0.0978  0.129
 2  1921 1     0.0129  0.0144
 3  1921 2     0.00521 0.00737
 4  1921 3     0.00471 0.00457
 5  1921 4     0.00461 0.00433
 6  1921 5     0.00372 0.00361
 7  1921 6     0.00265 0.00393
 8  1921 7     0.00295 0.00351
 9  1921 8     0.00237 0.00285
10  1921 9     0.00198 0.00255
# i 10,979 more rows
```

Sort the tibble according to a particular column using `arrange`, for example, Year in descending order:

```
dm |>
  arrange(-Year)
```

```
# A tibble: 10,989 x 5
    Year Age     Female     Male    Total
   <dbl> <chr>    <dbl>    <dbl>    <dbl>
 1  2019 0     0.00423  0.00481  0.00453
 2  2019 1     0.000216 0.000177 0.000196
 3  2019 2     0.000157 0.000162 0.00016
 4  2019 3     0.00007  0.00016  0.000117
 5  2019 4     0.000111 0.000132 0.000122
 6  2019 5     0.000096 0.000052 0.000074
 7  2019 6     0.000081 0.000039 0.000059
 8  2019 7     0.000107 0.000128 0.000118
 9  2019 8     0.000066 0.000026 0.000046
10  2019 9     0.000052 0.000177 0.000116
# i 10,979 more rows
```

NOTE: none of the above operations are saving. To save, you need to assign the output to an object. You can call it something new or overwrite the original.

Grouping, summarizing, mutating

In addition to `filter` and `select`, two useful functions are `mutate`, which allows you to create new variables, and `summarize`, which allows you to produce summary statistics. These are particularly powerful when combined with `group_by()` which allows you to do any operation on a tibble by group.

For example, let's create a new variable that is the ratio of male to female mortality at each age and year:

```
  dm <- dm |>
  mutate(mf_ratio = Male/Female)
```

Now, let's calculate the mean female mortality rate by age over all the years. To do this, we need to `group_by` Age, and then use `summarize` to calculate the mean:

```
summary_mean <- dm |>
  group_by(Age) |>
  summarize(mean_mortality = mean(Female, na.rm = TRUE))

dim(summary_mean)
```

```
[1] 111   2
```

```
dim(dm)
```

```
[1] 10989     6
```

Mean of males and females by age

```
dm |>
  group_by(Age) |>
  summarize(mean_mortality_f = mean(Female, na.rm = TRUE),
            mean_mortality_m = mean(Male, na.rm = TRUE))
```

```
# A tibble: 111 x 3
   Age   mean_mortality_f mean_mortality_m
   <chr>            <dbl>            <dbl>
 1 0             0.0254           0.0322
 2 1             0.00262          0.00297
 3 10            0.000426         0.000590
 4 100           0.426            0.462
 5 101           0.448            0.493
 6 102           0.493            0.566
 7 103           0.533            0.647
 8 104           0.660            0.780
 9 105           0.805            0.904
10 106           0.796            0.720
# i 101 more rows
```

Alternatively using `across`

```r
dm |>
  group_by(Age) |>
  summarize(across(Male:Female, mean, na.rm = TRUE))
```

```
Warning: There was 1 warning in `summarize()`.
i In argument: `across(Male:Female, mean, na.rm = TRUE)`.
i In group 1: `Age = "0"`.
Caused by warning:
! The `...` argument of `across()` is deprecated as of dplyr 1.1.0.
Supply arguments directly to `.fns` through an anonymous function instead.

  # Previously
  across(a:b, mean, na.rm = TRUE)

  # Now
  across(a:b, \(x) mean(x, na.rm = TRUE))
```

```
# A tibble: 111 x 3
   Age       Male   Female
   <chr>    <dbl>    <dbl>
 1 0     0.0322   0.0254
 2 1     0.00297  0.00262
 3 10    0.000590 0.000426
 4 100   0.462    0.426
 5 101   0.493    0.448
 6 102   0.566    0.493
 7 103   0.647    0.533
 8 104   0.780    0.660
 9 105   0.904    0.805
10 106   0.720    0.796
# i 101 more rows
```

Pivoting

We often need to switch between wide and long data format. The `dm` tibble is currently in wide format. To get it in long format we can use `pivot_longer`

```r
dm_long <- dm |>
  select(-mf_ratio) |>
```

```
    pivot_longer(Female:Total, names_to = "sex", values_to = "mortality")
  dm_long
```

```
# A tibble: 32,967 x 4
    Year Age   sex    mortality
   <dbl> <chr> <chr>      <dbl>
 1  1921 0     Female   0.0978
 2  1921 0     Male     0.129
 3  1921 0     Total    0.114
 4  1921 1     Female   0.0129
 5  1921 1     Male     0.0144
 6  1921 1     Total    0.0137
 7  1921 2     Female   0.00521
 8  1921 2     Male     0.00737
 9  1921 2     Total    0.00631
10  1921 3     Female   0.00471
# i 32,957 more rows
```

Revert this

```
  dm_long |>
    pivot_wider(names_from = "sex", values_from = "mortality")
```

```
# A tibble: 10,989 x 5
    Year Age    Female    Male   Total
   <dbl> <chr>   <dbl>   <dbl>   <dbl>
 1  1921 0     0.0978  0.129   0.114
 2  1921 1     0.0129  0.0144  0.0137
 3  1921 2     0.00521 0.00737 0.00631
 4  1921 3     0.00471 0.00457 0.00464
 5  1921 4     0.00461 0.00433 0.00447
 6  1921 5     0.00372 0.00361 0.00367
 7  1921 6     0.00265 0.00393 0.00330
 8  1921 7     0.00295 0.00351 0.00323
 9  1921 8     0.00237 0.00285 0.00262
10  1921 9     0.00198 0.00255 0.00227
# i 10,979 more rows
```

Using ggplot

You can plot things in R using the base `plot` function, but plots using `ggplot` are much prettier.

Say we wanted to plot the mortality rates for 30 year old males over time. In the function `ggplot`, we need to specify our data (in this case, a filtered version of dm), an x axis (Year) and y axis (Male). The axes are defined withing the `aes()` function, which stands for 'aesthetics'.

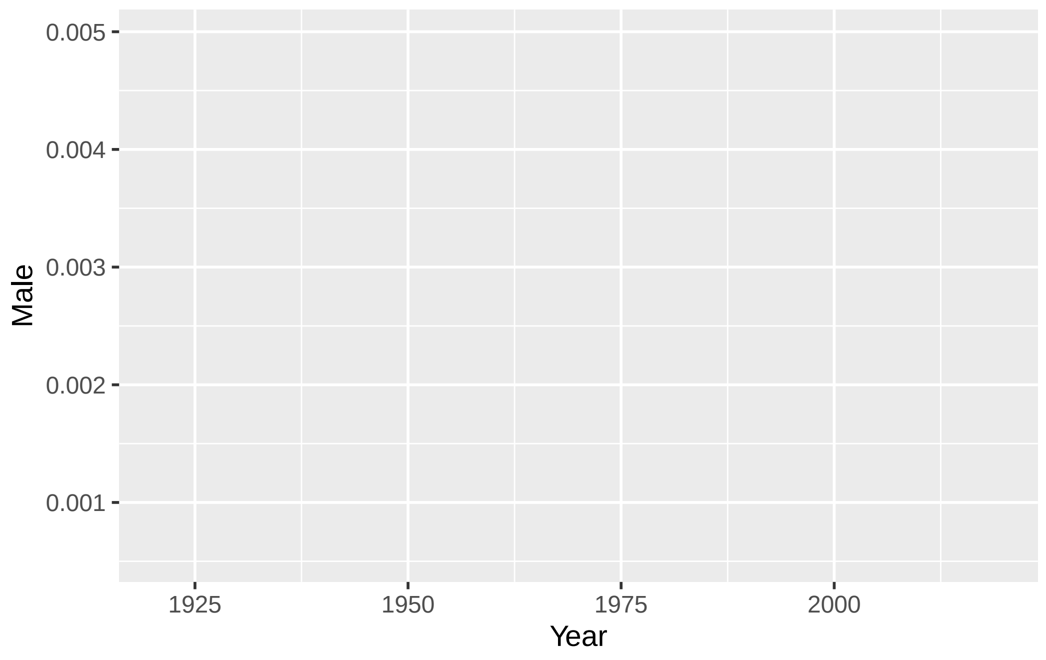
First let's get our data:

```
d_to_plot <- dm |>
  filter(Age==30) |>
  select(Year, Male)
d_to_plot
```

```
# A tibble: 99 x 2
    Year    Male
   <dbl>   <dbl>
 1  1921 0.00375
 2  1922 0.00462
 3  1923 0.00497
 4  1924 0.00412
 5  1925 0.00308
 6  1926 0.00308
 7  1927 0.00327
 8  1928 0.00356
 9  1929 0.00393
10  1930 0.00418
# i 89 more rows
```

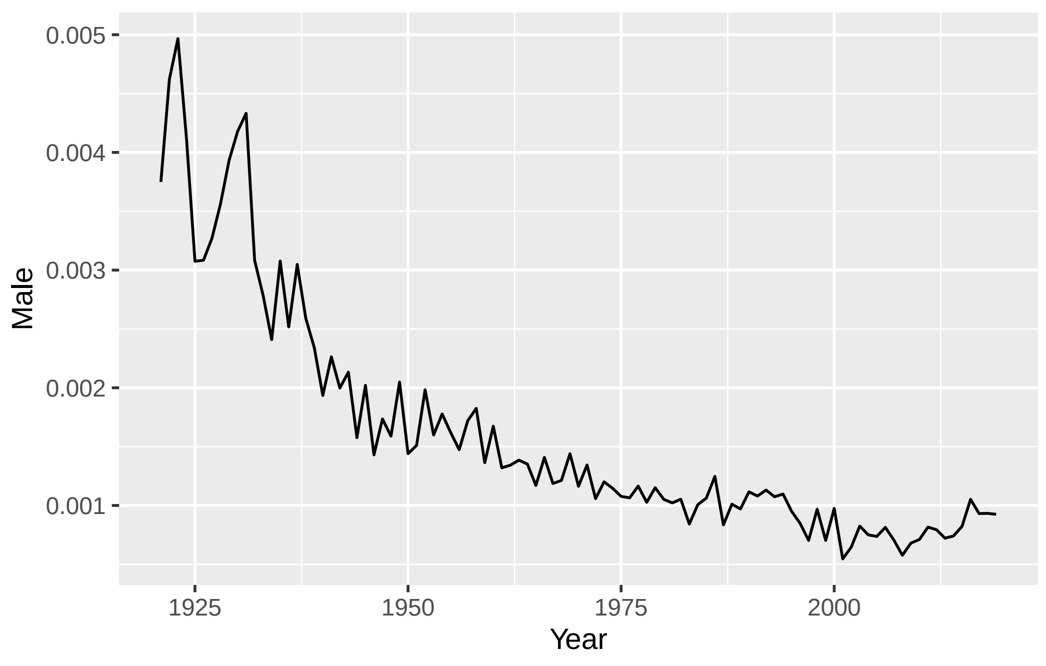
Now start the ggplot:

```
p <- ggplot(data = d_to_plot, aes(x = Year, y = Male))
p
```



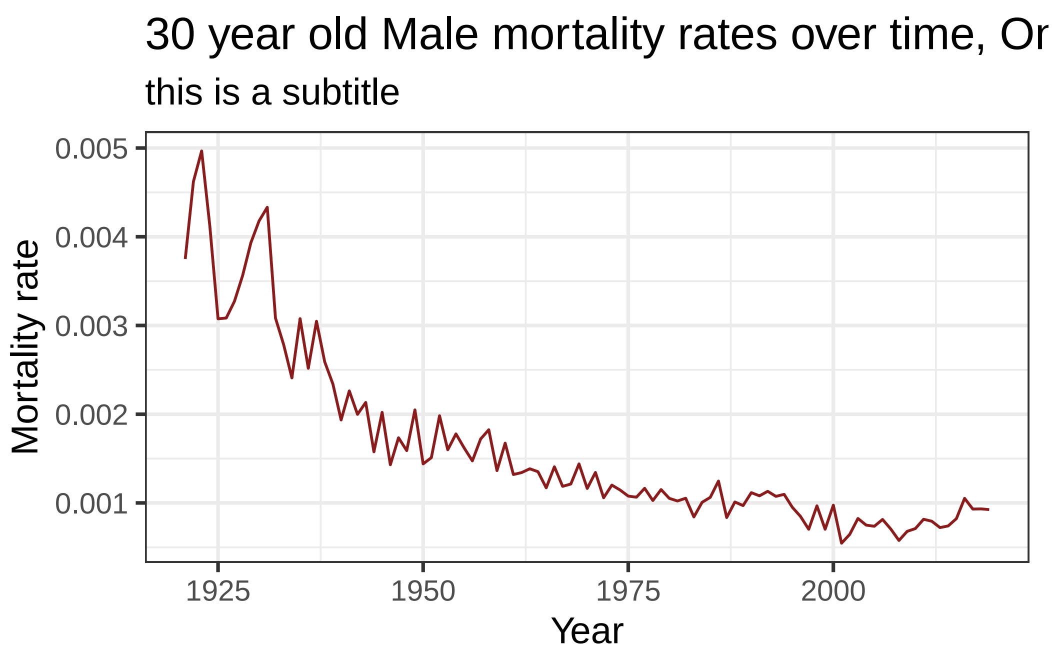
Notice the object `p` is just an empty box. The key to ggplot is layering: we now want to specify that we want a line plot using `geom_line()`:

```
p +
  geom_line()
```



Let's change the color of the line, and the y-axis label, and give the plot a title:

```
p +
  geom_line(color = "firebrick4")+
  labs(title = "30 year old Male mortality rates over time, Ontario",
       subtitle = "this is a subtitle",
       y = "Mortality rate") +
  theme_bw(base_size = 14)
```



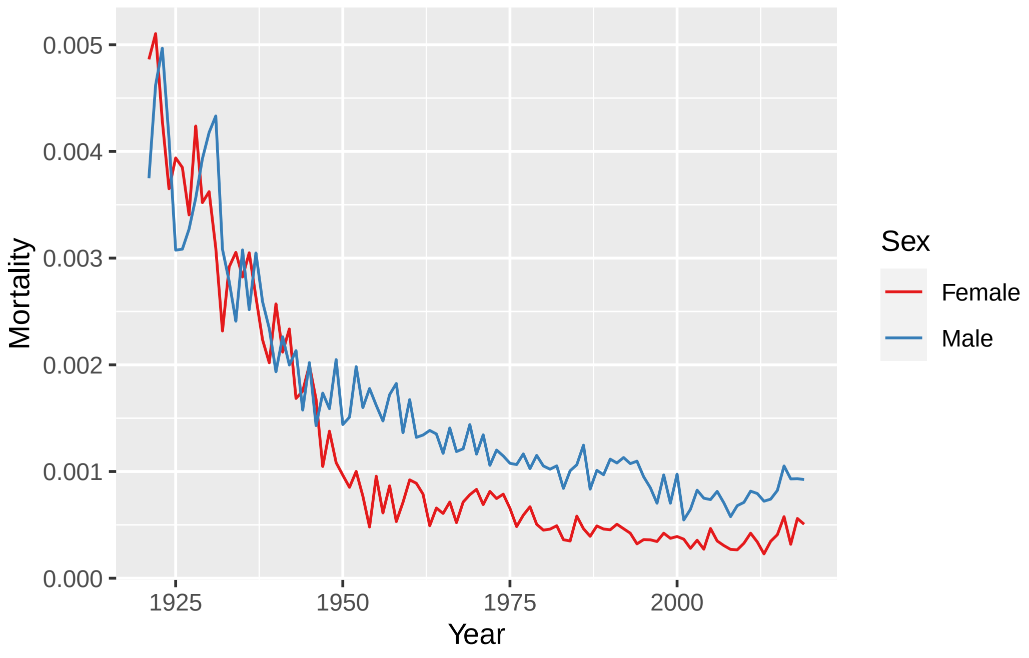
More than one group

Now say we wanted to have trends for 30-year old males and females on the one plot. The easiest way to do this is to first reshape our data so it's in long format: so instead of having a column for each sex, we have one column indicating the sex, and another column indicating the Mx value

```
dp <- dm |>
  filter(Age==30) |>
  select(Year:Male) |>
  pivot_longer(Female:Male, names_to = "Sex", values_to = "Mortality")
```

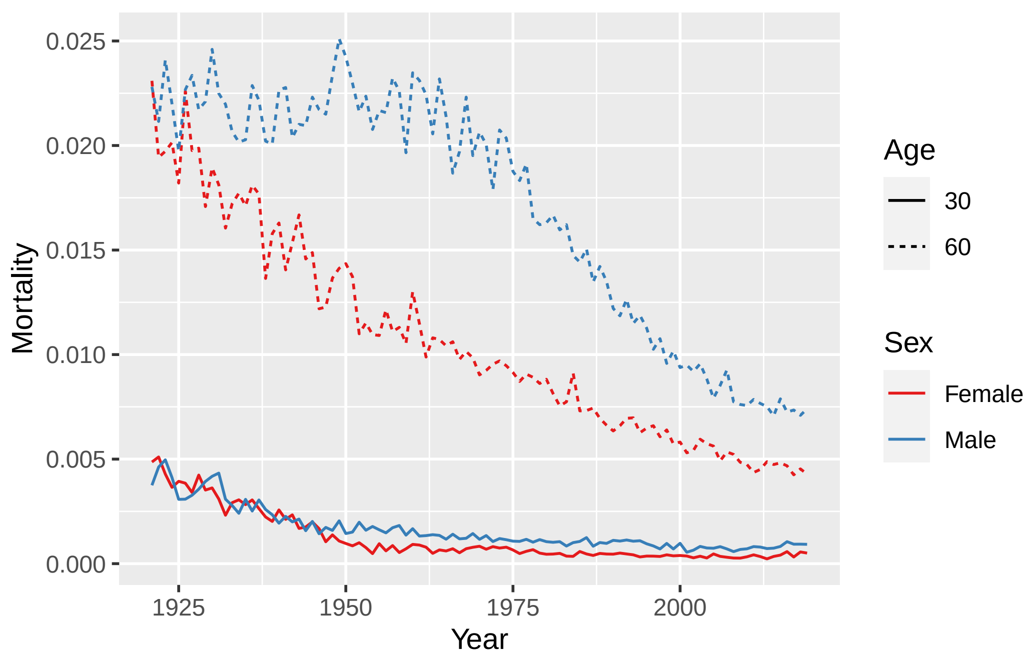
Now we can do a similar plot to before but we now have an added component in the `aes()` function: color, which is determined by sex:

```
dp |>
  ggplot(aes(x = Year, y = Mortality, color = Sex)) +
  geom_line() +
  scale_color_brewer(palette = "Set1")
```



```r
dp <- dm |>
  filter(Age==30|Age==60) |>
  select(Year:Male) |>
  pivot_longer(Female:Male, names_to = "Sex", values_to = "Mortality")

dp |>
  ggplot(aes(x = Year, y = Mortality, color = Sex, linetype = Age)) +
  geom_line() +
  scale_color_brewer(palette = "Set1")
```



Faceting

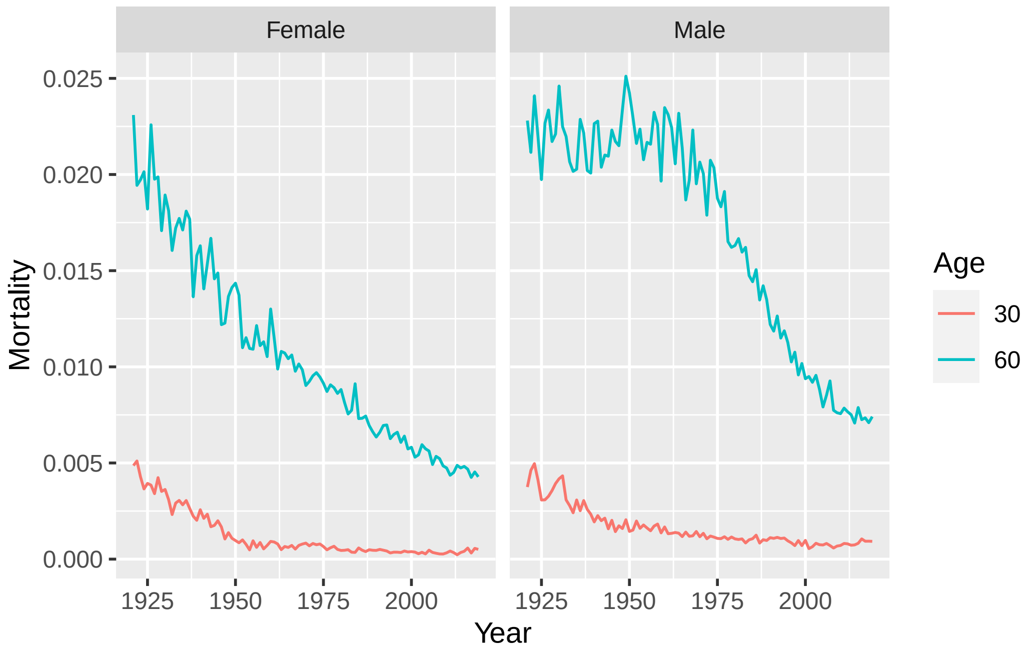
A neat thing about ggplot is that it's relatively easy to create 'facets' or smaller graphs divided by groups. Say we wanted to look at trends for 30 year olds and 60 year olds for both males and females. Let's get the data ready to plot:

```
dp
```

```
# A tibble: 396 x 4
    Year Age   Sex    Mortality
   <dbl> <chr> <chr>      <dbl>
 1  1921 30    Female   0.00486
 2  1921 30    Male     0.00375
 3  1921 60    Female   0.0231
 4  1921 60    Male     0.0228
 5  1922 30    Female   0.00510
 6  1922 30    Male     0.00462
 7  1922 60    Female   0.0194
 8  1922 60    Male     0.0212
 9  1923 30    Female   0.00429
10  1923 30    Male     0.00497
# i 386 more rows
```

Now let's plot, with a separate facet for each sex:

```
dp |>
  ggplot(aes(x = Year, y = Mortality, color = Age)) +
  geom_line()+
  facet_grid(~Sex)
```



Lab Exercises

Make a new Quarto or R Markdown file to answer these questions, and push to your repository on Github (both the .qmd and pdf file) by Monday 9am. The file should be appropriately named, and in a folder in your repo called 'labs' or something similar.

1. Plot the ratio of female to male mortality rates over time for ages 10,20,30 and 40 (different color for each age) and change the theme
2. Find the age that has the lowest female mortality rate each year
3. Use the `summarize(across())` syntax to calculate the standard deviation of mortality rates by age for the Male, Female and Total populations.
4. The Canadian HMD also provides population sizes over time (https://www.prdh.umontreal.ca/BDLC/data Use these to calculate the population weighted average mortality rate separately for males and females, for every year. Make a nice line plot showing the result (with

meaningful labels/titles) and briefly comment on what you see (1 sentence). Hint: `left_join` will probably be useful here.

5. Write down using appropriate notation, and run a simple linear regression with logged mortality rates as the outcome and age (as a continuous variable) as the covariate, using data for females aged less than 106 for the year 2000. Interpret the coefficient on age.