Intro to R Markdown and Tidyverse

Monica Alexander

8 January 2021

Contents

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

- RStudio Projects
- R Markdown
- Main tidyverse functions
- `ggplot`

2 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.

3 R Markdown

This is an R Markdown document. R Markdown 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 R Markdown file in Rstudio, go to File –> New File –> R Markdown, then select Document and whatever you want to compile the document as (I chose pdf). 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.

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 beginning of the chunk within the curly brackets (or set global options like I have done above).

3.1 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

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

4 Tidyverse

Read in some packages that we'll be using:

```r
library(tidyverse)
library(here)
```

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 Canada. These data come from the Human Mortality Database, which is a great source for studying mortality in high-income countries.

```r
dm <- read_table(here("data", "CAN_Mx_1x1.txt"), skip = 2)
head(dm)
```

```
## # A tibble: 6 x 5
##    Year Age   Female   Male     Total
##   <dbl> <chr> <chr>    <chr>    <chr>
## 1  1921 0     0.105821 0.138250 0.122259
## 2  1921 1     0.015593 0.017806 0.016710
## 3  1921 2     0.007409 0.008521 0.007970
## 4  1921 3     0.005442 0.006111 0.005779
## 5  1921 4     0.004563 0.004745 0.004655
## 6  1921 5     0.003433 0.003828 0.003633
```

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

4.1 Important tidyverse functions

You should feel comfortable using the following functions

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

4.2 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'.

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)
```

```
## # A tibble: 111 x 5
##     Year Age   Female   Male     Total
##    <dbl> <chr> <chr>    <chr>    <chr>
##  1  1935 0     0.065913 0.086446 0.076373
##  2  1935 1     0.010397 0.011375 0.010894
##  3  1935 2     0.004727 0.005106 0.004919
##  4  1935 3     0.003133 0.003910 0.003526
##  5  1935 4     0.002496 0.002812 0.002656
##  6  1935 5     0.001965 0.002325 0.002147
##  7  1935 6     0.001806 0.001818 0.001812
##  8  1935 7     0.001321 0.001887 0.001607
##  9  1935 8     0.001402 0.001638 0.001521
## 10  1935 9     0.001221 0.001534 0.001379
## # ... with 101 more rows
```

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

```
dm %>%
  filter(Year==1935, Age=="10")
```

```
## # A tibble: 1 x 5
##    Year Age   Female   Male     Total
##   <dbl> <chr> <chr>    <chr>    <chr>
## 1  1935 10    0.001214 0.001461 0.001339
```

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

```
dm %>%
  filter(Year==1935, Age=="10") %>%
  select(Year, Age, Female)
```

```
## # A tibble: 1 x 3
##    Year Age   Female
##   <dbl> <chr> <chr>
## 1  1935 10    0.001214
```

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

```r
dm %>%
  filter(Year==1935, Age=="10") %>%
  select(-Total)
```

```
## # A tibble: 1 x 4
##    Year Age   Female   Male
##   <dbl> <chr> <chr>    <chr>
## 1  1935 10    0.001214 0.001461
```

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

```r
dm %>%
  arrange(-Year)
```

```
## # A tibble: 10,101 x 5
##     Year Age   Female   Male     Total
##    <dbl> <chr> <chr>    <chr>    <chr>
##  1  2011 0     0.004296 0.005275 0.004798
##  2  2011 1     0.000232 0.000329 0.000282
##  3  2011 2     0.000134 0.000179 0.000157
##  4  2011 3     0.000123 0.000164 0.000144
##  5  2011 4     0.000093 0.000078 0.000085
##  6  2011 5     0.000084 0.000091 0.000087
##  7  2011 6     0.000056 0.000129 0.000094
##  8  2011 7     0.000091 0.000124 0.000108
##  9  2011 8     0.000109 0.000071 0.000089
## 10  2011 9     0.000063 0.000060 0.000061
## # ... with 10,091 more rows
```

4.3 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.

First, a bit of a clean up (make the ages and mortality rates numbers not characters)

```r
dm <-
  dm %>%
  mutate(Age = as.numeric(Age),
         Female = as.numeric(Female),
         Male = as.numeric(Male),
         Total = as.numeric(Male))
```

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

```r
dm %>%
  mutate(m_f_ratio = Male/Female)
```

```
## # A tibble: 10,101 x 6
##     Year   Age  Female    Male   Total m_f_ratio
##    <dbl> <dbl>   <dbl>   <dbl>   <dbl>     <dbl>
##  1  1921     0 0.106   0.138   0.138        1.31
##  2  1921     1 0.0156  0.0178  0.0178       1.14
##  3  1921     2 0.00741 0.00852 0.00852      1.15
##  4  1921     3 0.00544 0.00611 0.00611      1.12
##  5  1921     4 0.00456 0.00474 0.00474      1.04
##  6  1921     5 0.00343 0.00383 0.00383      1.12
##  7  1921     6 0.00285 0.00363 0.00363      1.27
##  8  1921     7 0.00283 0.00318 0.00318      1.12
##  9  1921     8 0.00247 0.00262 0.00262      1.06
## 10  1921     9 0.00216 0.00245 0.00245      1.13
## # ... with 10,091 more rows
```

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:

```
dm %>%
  group_by(Age) %>%
  summarize(mean_female_mortality = mean(Female, na.rm = T))
```

```
## # A tibble: 111 x 2
##      Age mean_female_mortality
##    <dbl>                 <dbl>
##  1     0              0.0322
##  2     1              0.00389
##  3     2              0.00190
##  4     3              0.00138
##  5     4              0.00112
##  6     5              0.000910
##  7     6              0.000795
##  8     7              0.000701
##  9     8              0.000628
## 10     9              0.000576
## # ... with 101 more rows
```

4.4 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`

```
dm %>% pivot_longer(cols = Female:Total, names_to = "sex", values_to = "rate" )
```

```
## # A tibble: 30,303 x 4
##     Year   Age sex       rate
##    <dbl> <dbl> <chr>    <dbl>
##  1  1921     0 Female 0.106
##  2  1921     0 Male   0.138
##  3  1921     0 Total  0.138
##  4  1921     1 Female 0.0156
##  5  1921     1 Male   0.0178
##  6  1921     1 Total  0.0178
##  7  1921     2 Female 0.00741
##  8  1921     2 Male   0.00852
##  9  1921     2 Total  0.00852
## 10  1921     3 Female 0.00544
## # ... with 30,293 more rows
```

4.5 Using ggplot

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

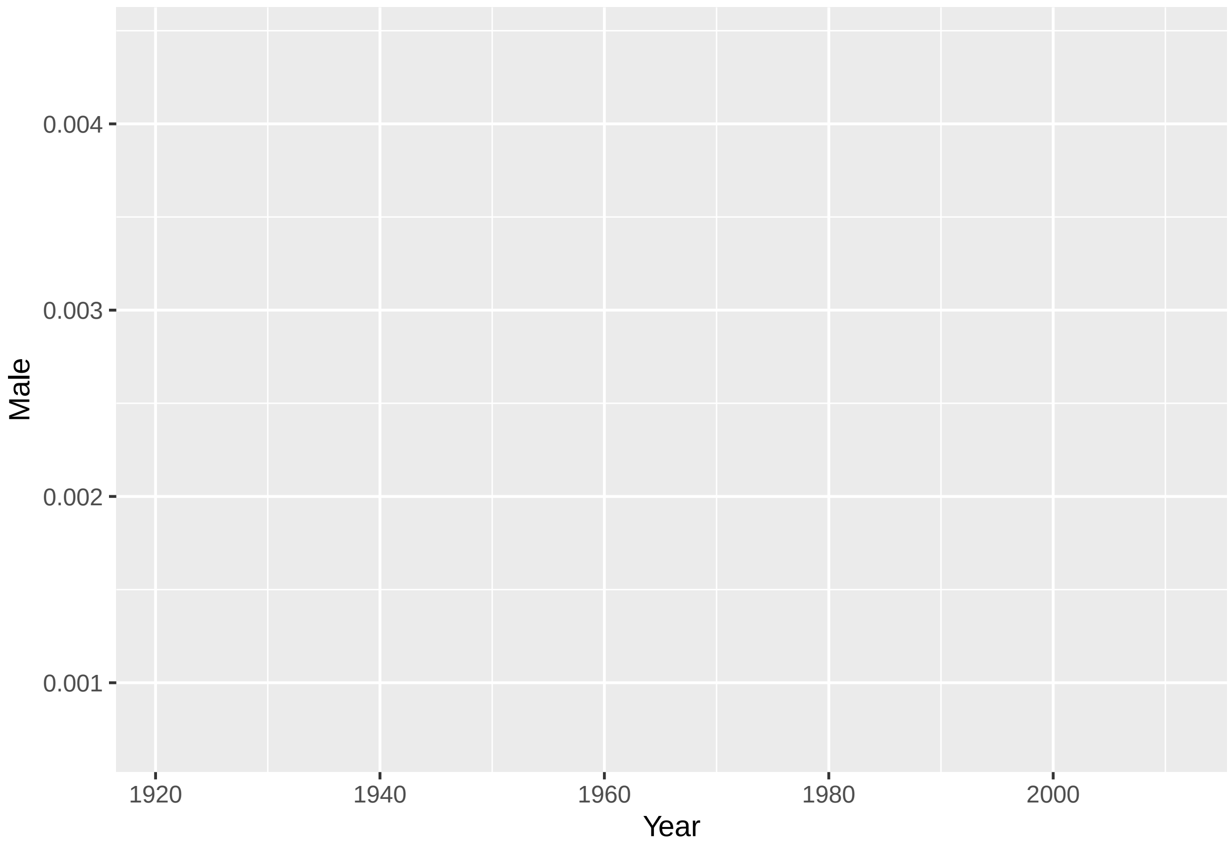
Say we wanted to plot the mortality rates for 30 year old males over time. In the function `ggplot`, we need to specify our data (in this case, a filtered version of dm), an x axis (Year) and y axis (Male). The axes are defined withing the `aes()` function, which stands for 'aesthetics'.

First let's get our data:

```
dm_to_plot <- dm %>%
  filter(Age==30)
```

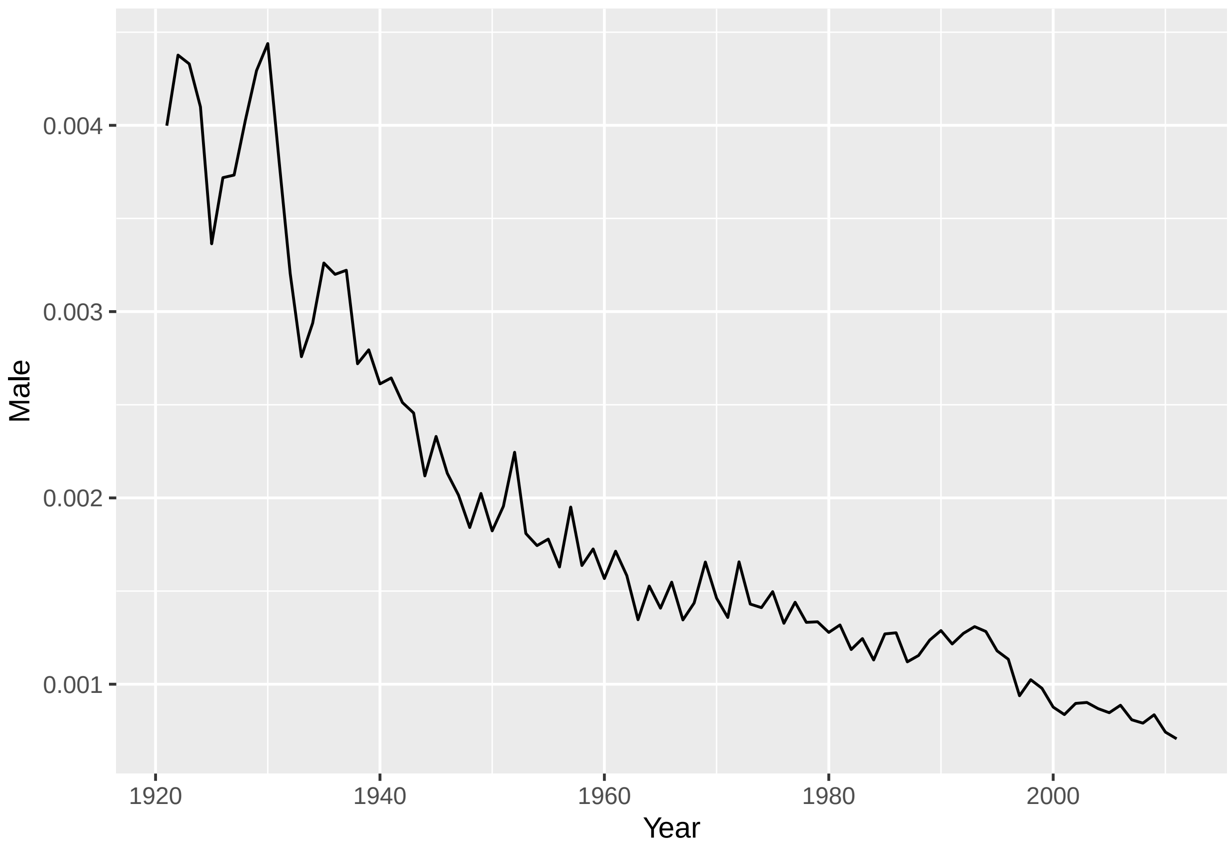
Now start the ggplot:

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



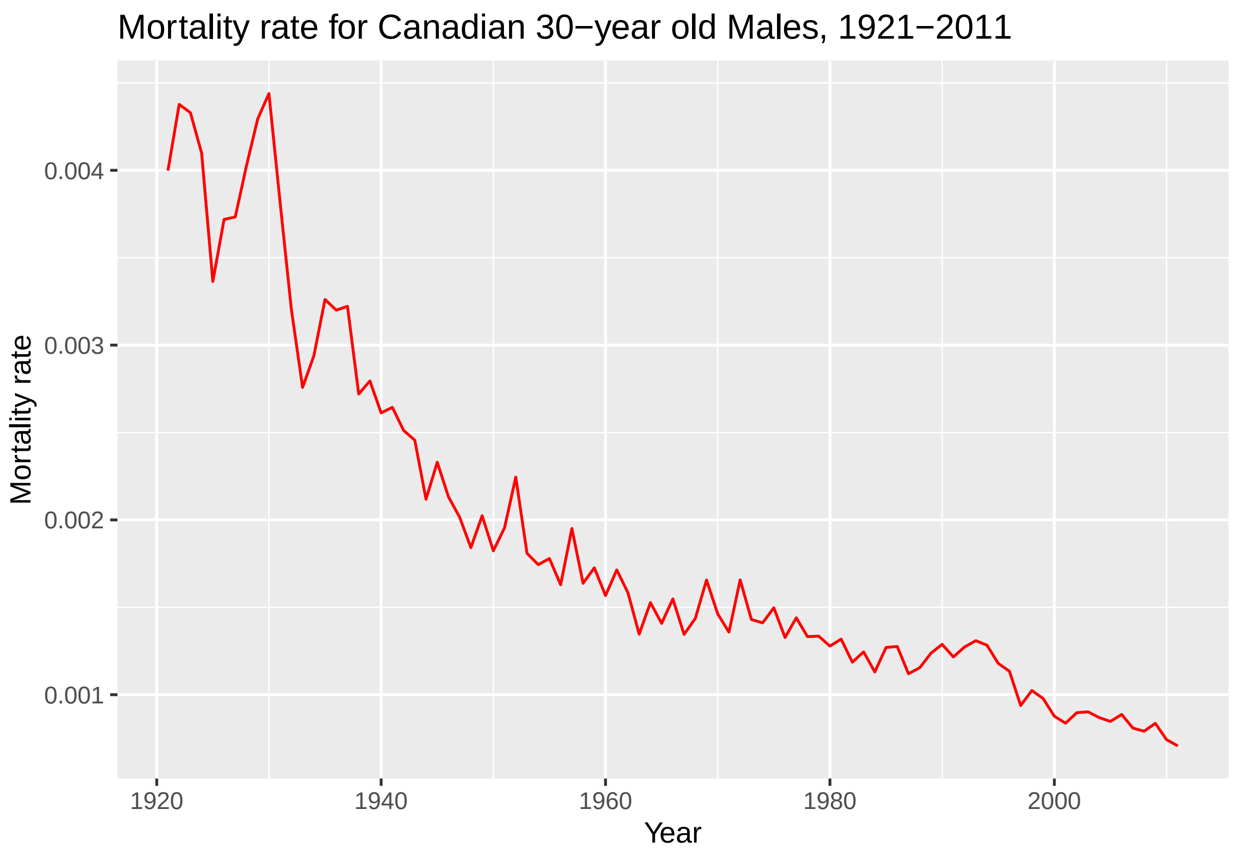
Notice the object `p` is just an empty box. The key to ggplot is layering: we now want to specify that we want a line plot using `geom_line()`:

```
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 = "red") +
  ylab("Mortality rate") +
  ggtitle("Mortality rate for Canadian 30-year old Males, 1921-2011")
```



4.5.1 More than one group

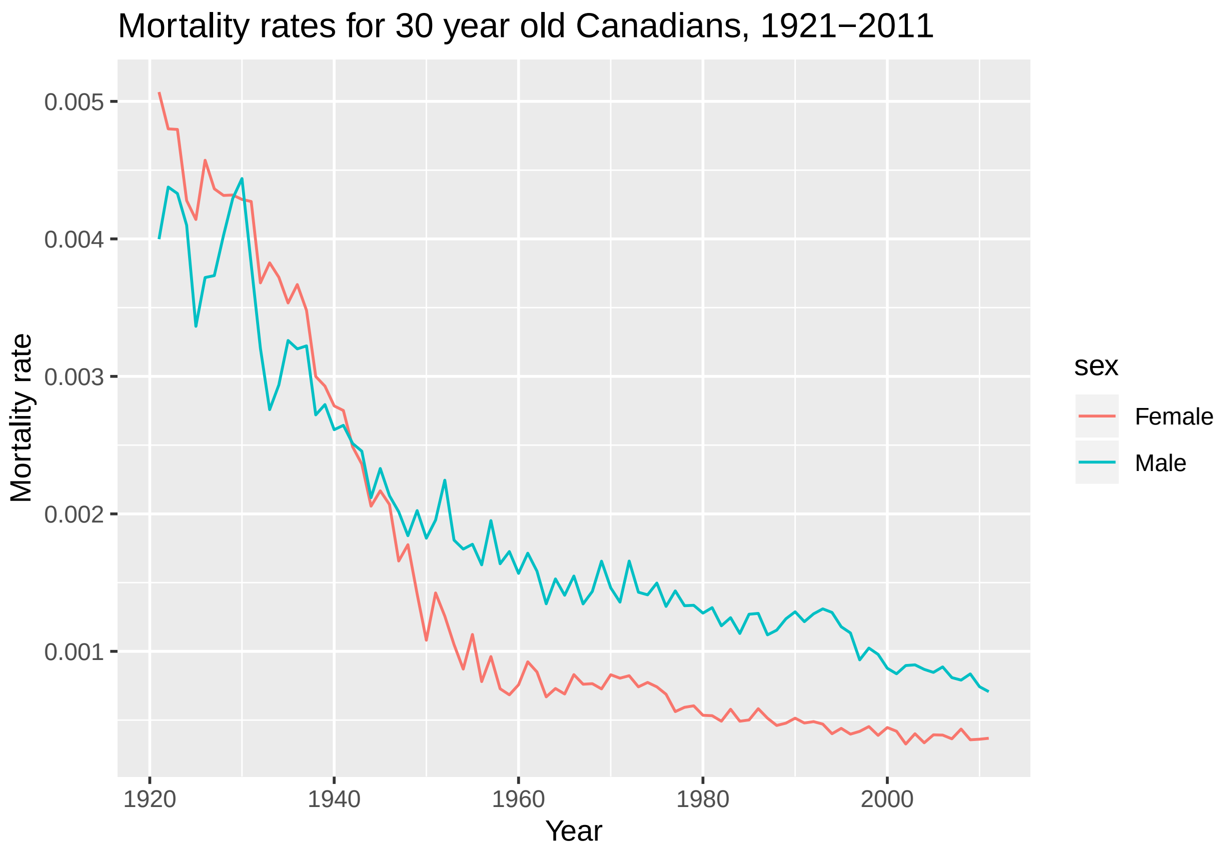
Now say we wanted to have trends for 30-year old males and females on the one plot. The easiest way to do this is to first reshape our data so it's in long format: so instead of having a column for each sex, we have one column indicating the sex, and another column indicating the Mx value

```
dm_to_plot <- dm_to_plot %>%
  pivot_longer(Female:Total, "sex", values_to = "mx") %>%
  filter(sex!="Total")
dm_to_plot
```

```
## # A tibble: 182 x 4
##     Year   Age sex         mx
##    <dbl> <dbl> <chr>    <dbl>
##  1  1921    30 Female 0.00507
##  2  1921    30 Male   0.00400
##  3  1922    30 Female 0.00480
##  4  1922    30 Male   0.00438
##  5  1923    30 Female 0.00480
##  6  1923    30 Male   0.00433
##  7  1924    30 Female 0.00428
##  8  1924    30 Male   0.00410
##  9  1925    30 Female 0.00414
## 10  1925    30 Male   0.00336
## # ... with 172 more rows
```

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:

```
p2 <- ggplot(dm_to_plot, aes(Year, mx, color = sex)) +
  geom_line() +
  ylab("Mortality rate")+
  ggtitle("Mortality rates for 30 year old Canadians, 1921-2011")
p2
```



4.5.2 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:

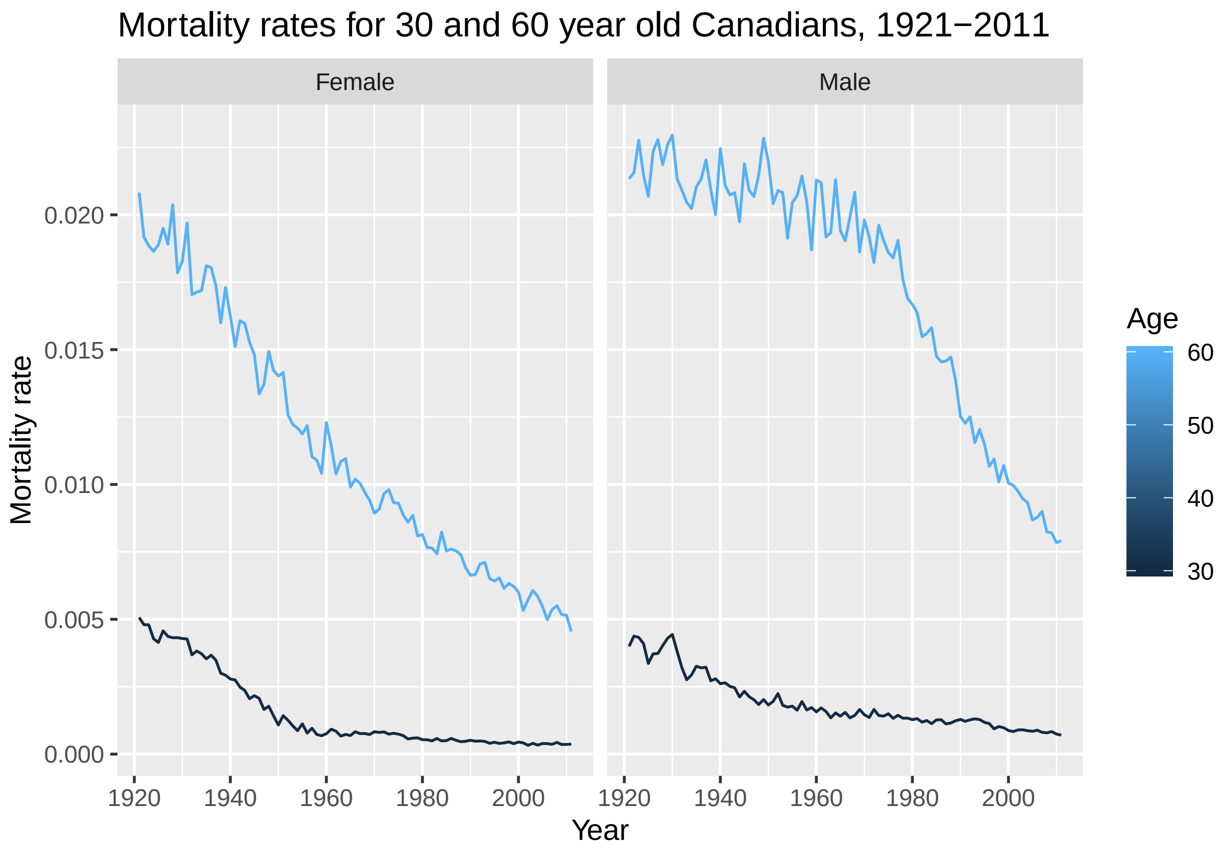
```
dm_to_plot <- dm %>%
  filter(Age==30|Age==60) %>%
  pivot_longer(Female:Total, "sex", values_to = "mx") %>%
  filter(sex!="Total")

dm_to_plot
```

```
## # A tibble: 364 x 4
##     Year   Age sex          mx
##    <dbl> <dbl> <chr>     <dbl>
##  1  1921    30 Female 0.00507
##  2  1921    30 Male   0.00400
##  3  1921    60 Female 0.0208
##  4  1921    60 Male   0.0213
##  5  1922    30 Female 0.00480
##  6  1922    30 Male   0.00438
##  7  1922    60 Female 0.0192
##  8  1922    60 Male   0.0216
##  9  1923    30 Female 0.00480
## 10  1923    30 Male   0.00433
## # ... with 354 more rows
```

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

```r
ggplot(dm_to_plot, aes(Year, mx, color = Age, group = Age)) +
  geom_line()+
  ylab("Mortality rate")+
  facet_grid(~sex)+
  ggtitle("Mortality rates for 30 and 60 year old Canadians, 1921-2011")
```



5 Lab Exercises

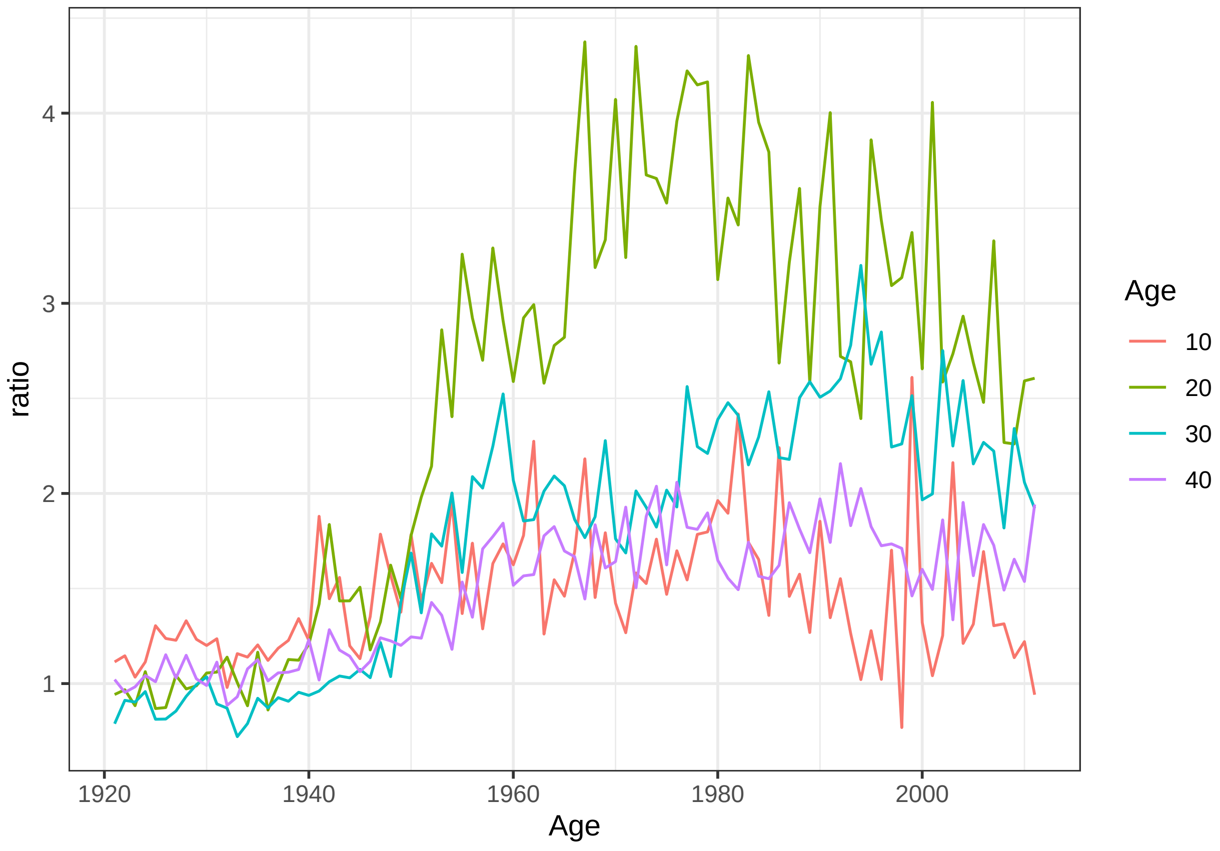
1. Plot the ratio of male to female mortality rates over time for ages 10,20,30 and 40 (different color for each age) and change the theme (e.g. `theme_bw()`)

```r
main_data <- read_table(here("data", "CAN_Mx_1x1.txt"), skip = 2)

Q1 <- main_data %>% mutate(Age = as.numeric(Age),
                           Female = as.numeric(Female),
                           Male = as.numeric(Male),
                           Total = as.numeric(Total))
# Q1
plot_1 <- Q1 %>%
  filter(Age == 10 | Age == 20 | Age == 30 | Age == 40) %>%
  mutate(Age = as.factor(Age)) %>%
  mutate(ratio = Male/Female) %>%
  select(Year, Age, ratio)

p <- ggplot(data = plot_1, aes(x = Year, y = ratio, color = Age)) +
  geom_line() +
  theme_bw() +
  xlab("Age") +
  ylab("ratio")

p
```



2. Find the age that has the highest female mortality rate each year (hint: `slice` might be useful here)

```r
main_data %>%
  group_by(Year) %>%
  slice(which.max(Female)) %>%
  select(Year, Age, Female)
```

```
## # A tibble: 91 x 3
## # Groups:   Year [91]
##     Year Age   Female
##    <dbl> <chr> <chr>
##  1  1921 107   4.171429
##  2  1922 106   0.774566
##  3  1923 107   1.978723
##  4  1924 107   2.419355
##  5  1925 106   0.613277
##  6  1926 108   4.186047
##  7  1927 107   4.171429
##  8  1928 107   6.000000
##  9  1929 106   1.471698
## 10  1930 107   6.000000
## # ... with 81 more rows
```

3. Use the `summarize_at()` function to calculate the standard deviation of mortality rates by age for the Male, Female and Total populations.

```
main_data %>%
  group_by(Age) %>%
  summarise_at(vars(Female:Total), sd, na.rm = TRUE)
```

```
## # A tibble: 111 x 4
##    Age     Female     Male    Total
##    <chr>    <dbl>    <dbl>    <dbl>
##  1 0     0.0307   0.0402   0.0355
##  2 1     0.00490  0.00562  0.00526
##  3 10    0.000546 0.000632 0.000587
##  4 100   0.0681   0.0800   0.0540
##  5 101   0.0725   0.0937   0.0602
##  6 102   0.0874   0.108    0.0701
##  7 103   0.116    0.145    0.0938
##  8 104   0.134    0.250    0.137
##  9 105   0.225    0.349    0.219
## 10 106   0.441    0.921    0.399
## # ... with 101 more rows
```