Lab 1 Analysis

Stephen Cook

August 27 2018

Analysis

All code in this document is enclosed in "code chunks". Anything in there may be executed by R if you press the little green "play" button at the right side of the code chunk. You may insert a new code chunk by going to "Code" in the menu ribbon, and clicking on "Insert Chunk". I like to use the quick-key ctr+alt+i, but do whatever feels best for you.

All labs will generally require the same three steps:

1. Import data
2. Perform some calculations
3. Plot the results (and perform statistics as necessary)

Import packages

This section just imports the packages which add extra functions to R that we will need for the analysis. Don't think too hard about this part - just include it. If you haven't installed the package yet, this will throw errors. Click on "Packages" in the lower right hand pane to install them if necessary.

```r
require(tidyverse)
require(stringr)
```

Import data

We need to get our data into R before we can actually do any work. The `read.csv()` function goes hunting for a `.csv` file with that name and assigns it to the variable `human`. The rest of the stuff inside the `()` just tells the `read.csv()` function that we have headers, and that our file is comma-separated.

```r
human <- read.csv(file = "human_data.csv", header = T, sep = ",")
```

Now that we have `human` imported, it will appear as a variable in the Enrivonment pane (upper right). If you click the blue drop-down, it will give you some information about the type of data stored in each column, and if you click directly on `human` it will open a tab showing you the data in table-format.

Calculations

Right now we only need to calculate how old each individual was when they died. The `mutate` function will create a new column based on previous columns in the data. The `%>%` is called a piping-function. Read it as "take the thing before the `%>%` and do the following to it". You can string a bunch of these together and it would read like "then do this... then do this... then do this". We assign all of this back to `human` with the `<-` symbol.

```r
human <- human %>%
  mutate(AGE = DEATH - BIRTH)
```

Click on `human` again. You should now have a new column named `AGE`.

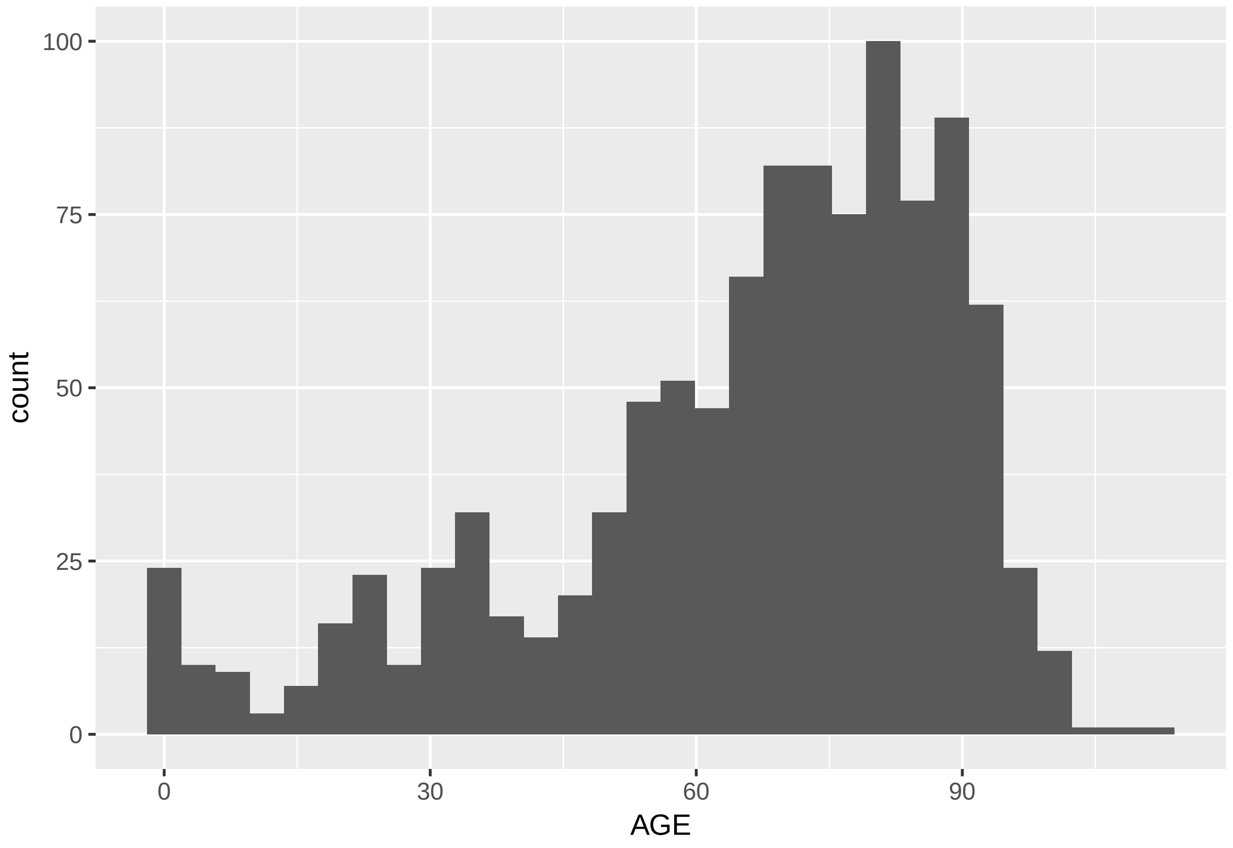
Plotting

Now the fun part! A histogram is the simplest way of visualizing what our data is doing. The goal is to get an object called `plot1`, so everything after `<-` is what goes into creating `plot1`. `ggplot()` initializes the plot, and we can then start throwing layers on the plot (ggplot is a very powerful plotting add-on that lets you build layers on your graph using `geoms`).

Every geom needs to know where to go to get the data (use the `data =` argument), as well as an aesthetics argument that needs an x and y value to graph. In the histogram's case, it only needs an x value.

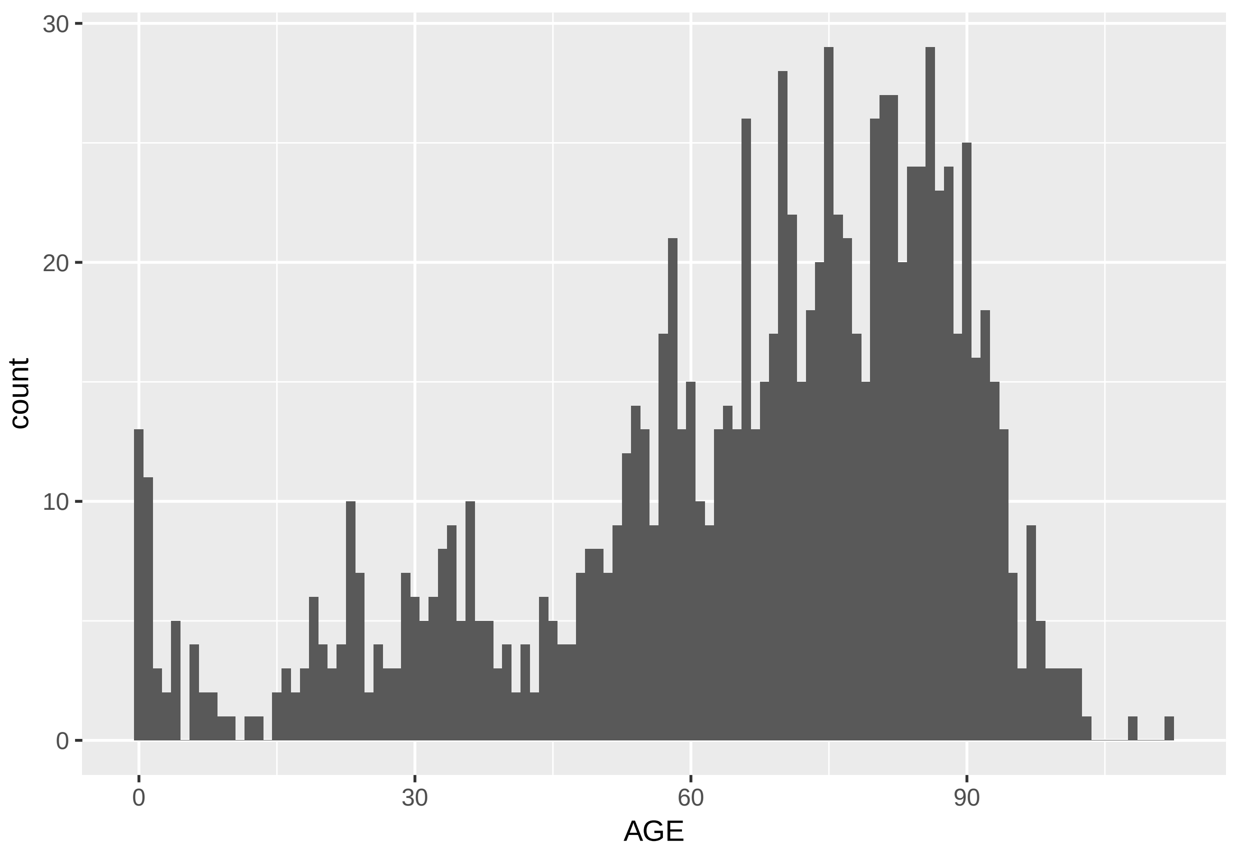
```
plot1 <- ggplot() +
  geom_histogram(data = human,
                 aes(x = AGE))

plot1
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



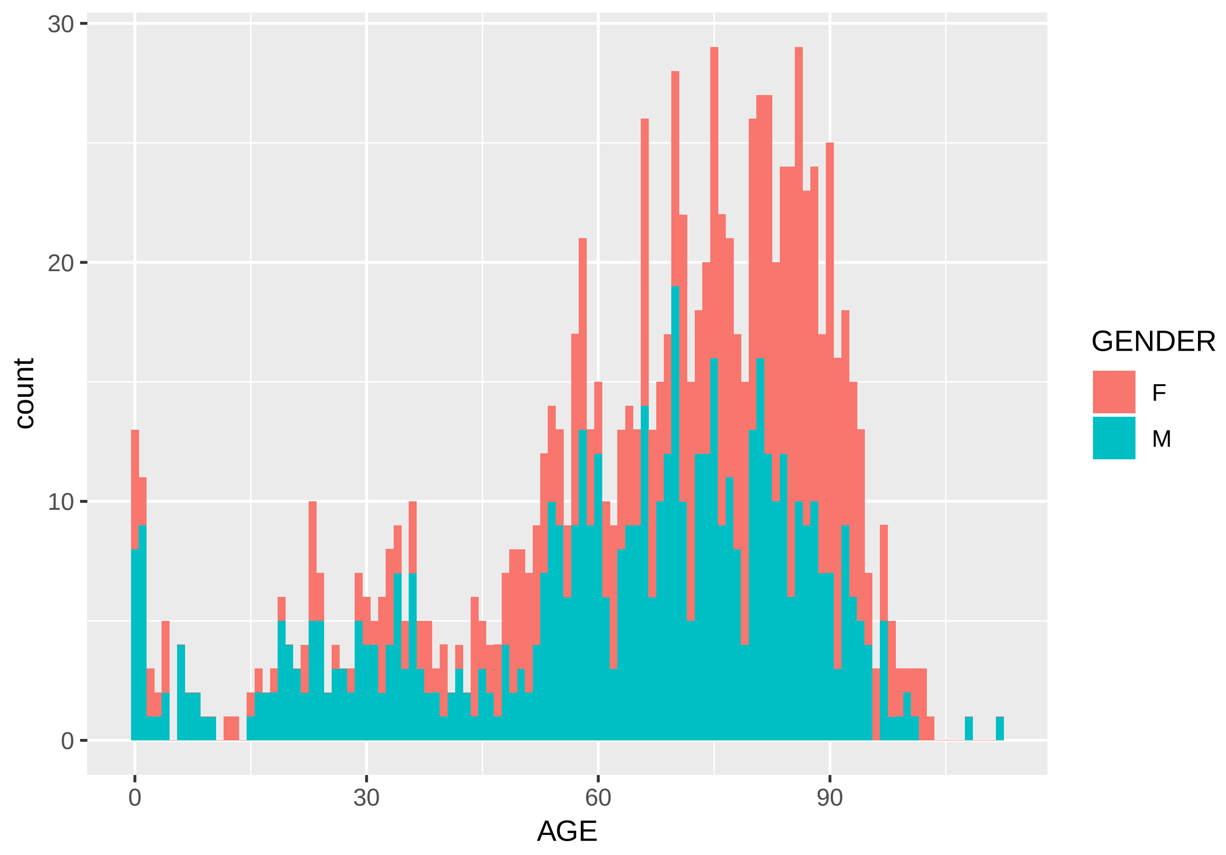
Let's change the `binwidth` to `1` to see every single year plotted out.

```
plot1 <- ggplot() +
  geom_histogram(data = human,
                 aes(x = AGE),
                 binwidth = 1)

plot1
```



One of our hypotheses was about the difference between males and females. How do we visualize that? Just add another argument to the `aes()` part. You can use either `fill =` or `color =`, but `fill` is a little easier to see in this case.

```
plot1 <- ggplot() +
  geom_histogram(data = human,
                 aes(x = AGE, fill = GENDER),
                 binwidth = 1)

plot1
```



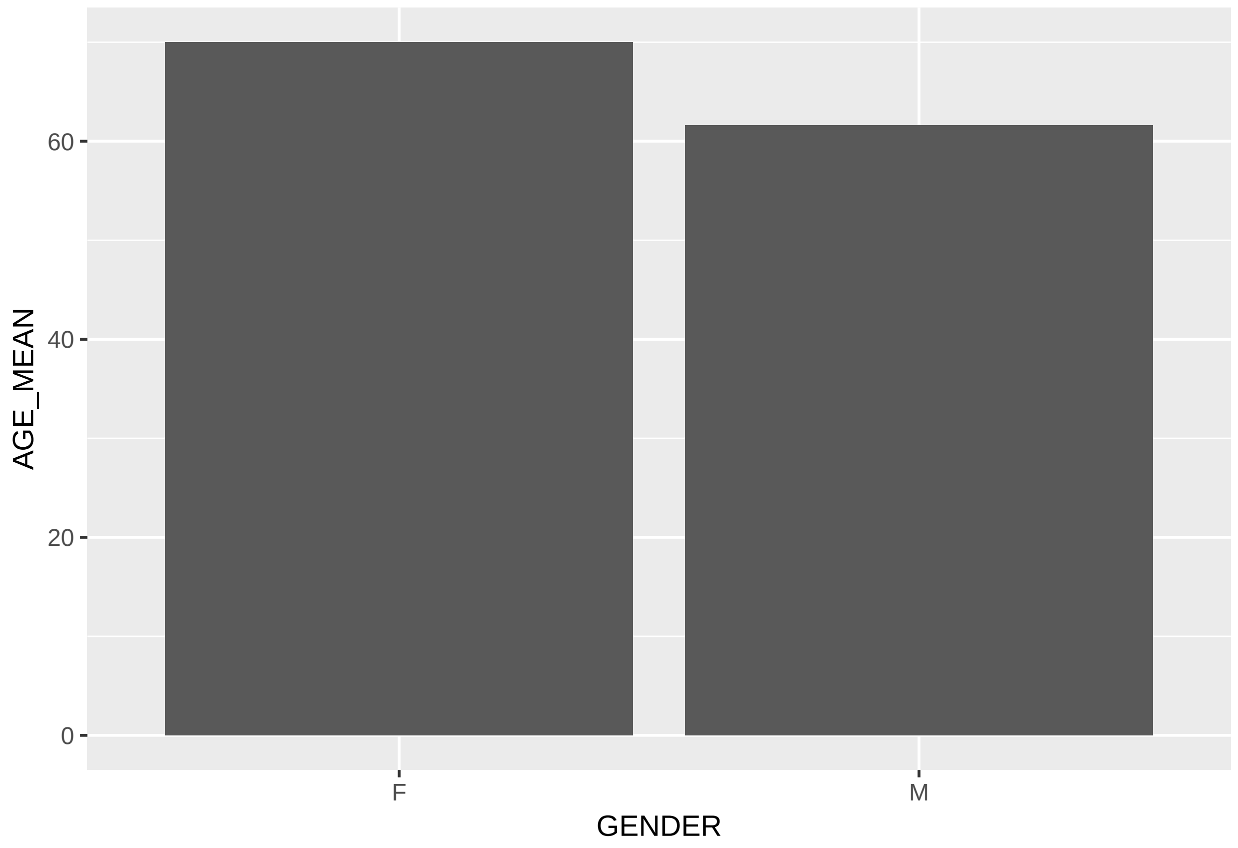
How would we plot the same data in a bar-graph? Well… we need to summarise the data a bit.

```
human_avg <- human %>%
  group_by(GENDER) %>%
  summarise(AGE_MEAN = mean(AGE),
            AGE_SE = sd(AGE)/sqrt(n()))
```

Walk through that line-by-line. We are making something new called `human_avg`, which starts with `human` and walks through some `%>%` pipes. Try and read it out loud (it helps). We want to `group_by()` our variable of interest, then `summarise()`some things about the data. We use functions to help us calculate the mean age (`AGE_MEAN`) and the stadard error (`AGE_SE`). Click on the new data frame and take a look around.

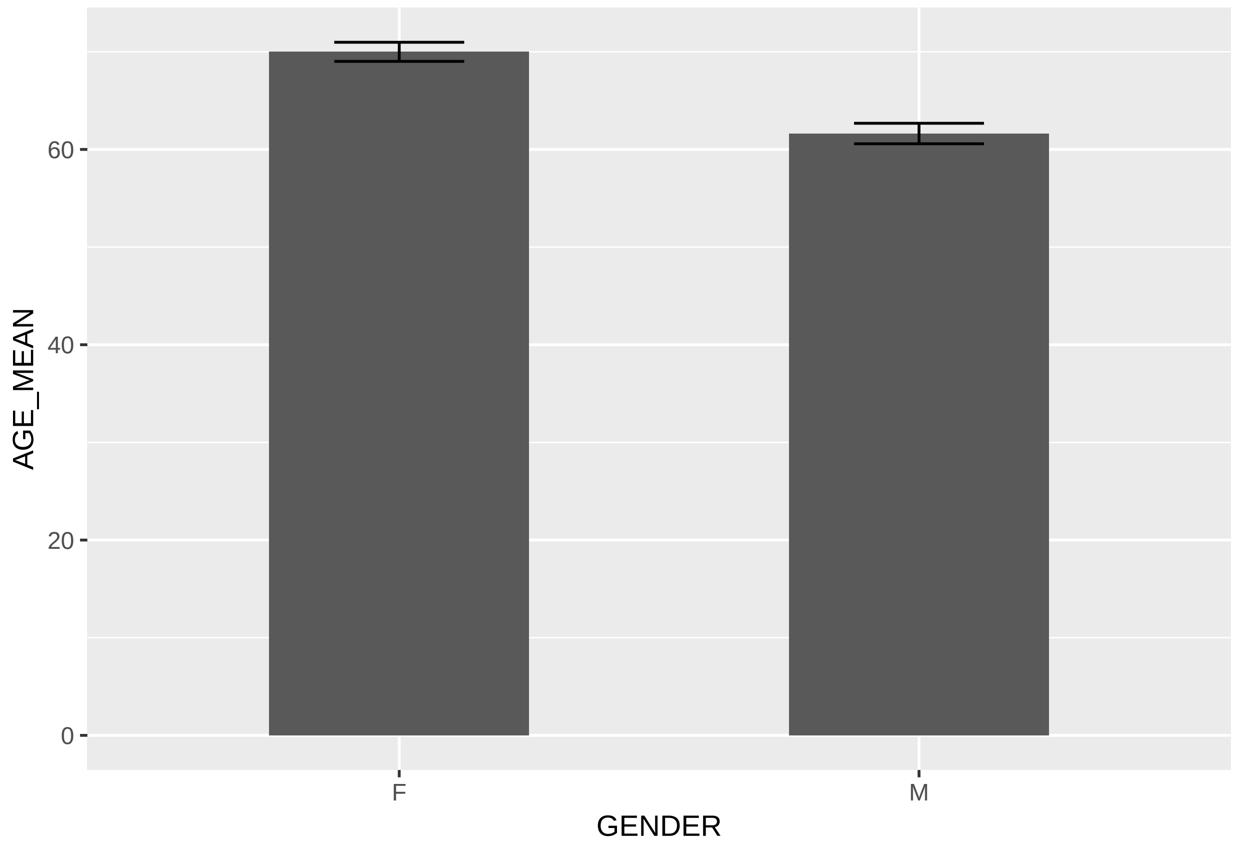
After doing the calculations the plotting is simple. Just change `geom_histogam()` to `geom_col()`, and direct it to look for the data in your newly created data frame.

```
plot2 <- ggplot() +
  geom_col(data = human_avg,
           aes(x = GENDER, y = AGE_MEAN))

plot2
```



While accurate, that is pretty ugly. Let's try and add some bells and whistles, as well as some error-bars using `geom_errorbar()`. We can add new `geoms` using a `+` symbol.

```
plot2 <- ggplot() +
  geom_col(data = human_avg,
           aes(x = GENDER, y = AGE_MEAN),
           width = 0.5) +
  geom_errorbar(data = human_avg,
                aes(x = GENDER, ymin = AGE_MEAN - AGE_SE, ymax = AGE_MEAN + AGE_SE),
                width = 0.25)

plot2
```



Getting better… let's change those eye-sores of axis-labels.

```
plot3 <- ggplot() +
  geom_col(data = human_avg,
           aes(x = GENDER, y = AGE_MEAN),
           width = 0.5) +
  geom_errorbar(data = human_avg,
                aes(x = GENDER, ymin = AGE_MEAN - AGE_SE, ymax = AGE_MEAN + AGE_SE),
                width = 0.25) +
  xlab("Gender") +
  ylab("Age at death (mean)")

plot3
```

There ya go. You now have code to generate a bar-graph with error bars. Only thing you'll need to change for Lab Report 2 is the data and therefore the variable names.

Statistical analysis

So now that we have visualized the data… does gender *really* make a difference in how long a human individual survives? Lets split the dataset up so that we can run a t-test.

```
male <- human %>%
  filter(GENDER == "M") # filters only males out to assign to 'male'

female <- human %>%
```

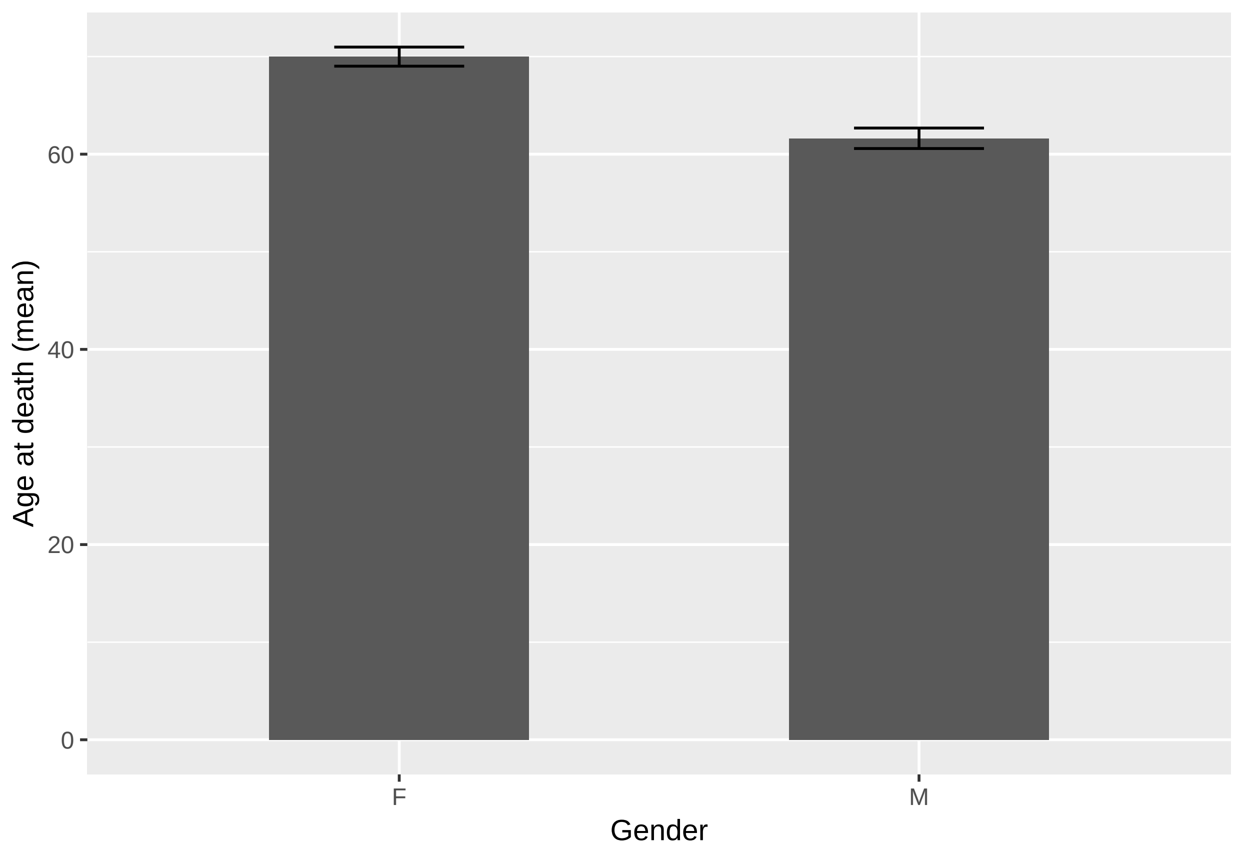


Figure 1: This is a really nice way to make figure captions without worrying about formatting in Microsoft Word

```r
  filter(GENDER == "F") # filters only females out to assign to 'female'
```

The t-test function in R is creatively named `t.test()`. It requires two vectors of numbers as imput, which is what the `$` is for below.

```r
human_ttest <- t.test(male$AGE, female$AGE)

human_ttest
```

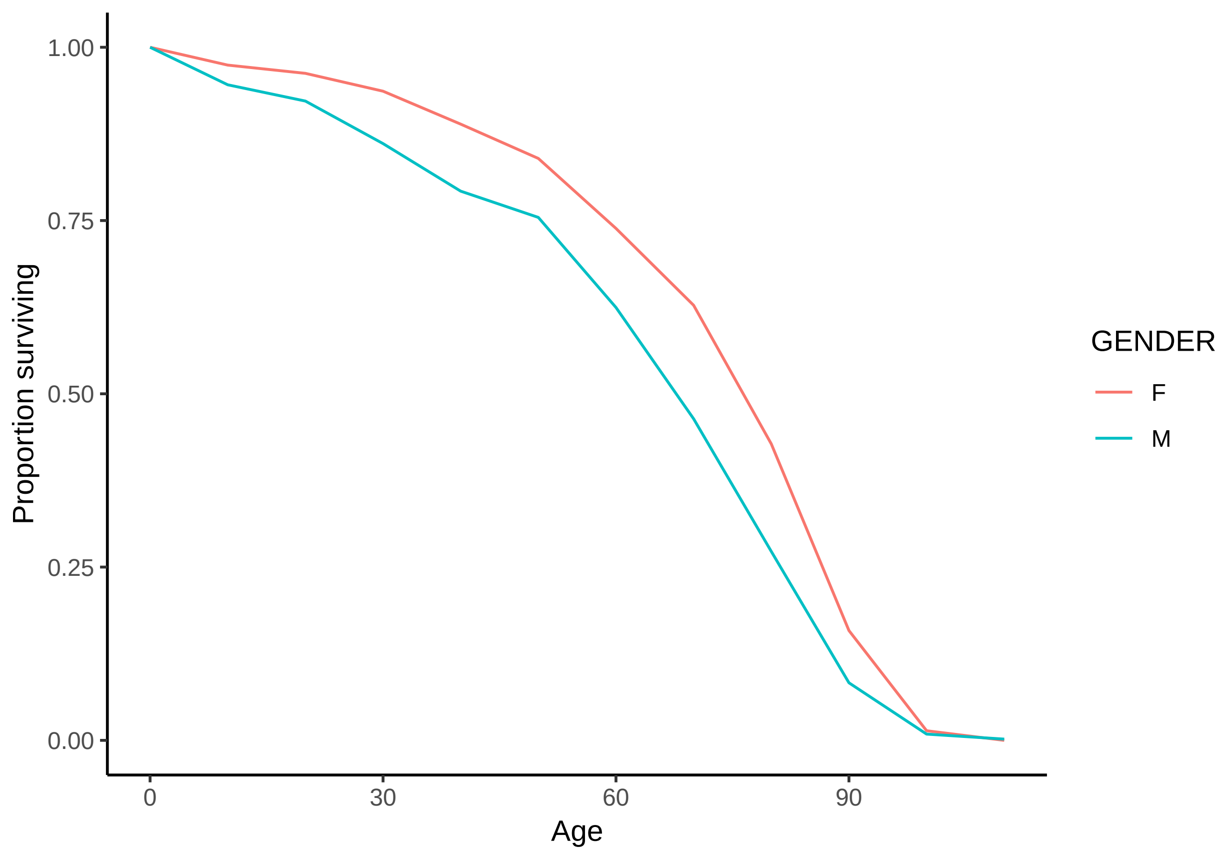
```
##
##  Welch Two Sample t-test
##
## data:  male$AGE and female$AGE
## t = -5.8325, df = 1056.4, p-value = 7.252e-09
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -11.18065  -5.55150
## sample estimates:
## mean of x mean of y
##  61.62996  69.99604
```

Survivorship curve

So how would we program a way to calculate the “classic” way of displaying survivorship information? Sure! I’m not going to walk you through this, but try and walk through and figure out what each line is doing.

```r
human_survivorship <- human %>%
  select(GENDER, AGE) %>%
  group_by(GENDER) %>%
  summarise(greater_0 = sum(AGE>=0)/n(),
            greater_10 = sum(AGE >=10)/n(),
            greater_20 = sum(AGE >=20)/n(),
            greater_30 = sum(AGE >=30)/n(),
            greater_40 = sum(AGE >= 40)/n(),
            greater_50 = sum(AGE >= 50)/n(),
            greater_60 = sum(AGE >= 60)/n(),
            greater_70 = sum(AGE >=70)/n(),
            greater_80 = sum(AGE >= 80)/n(),
            greater_90 = sum(AGE >= 90)/n(),
            greater_100 = sum(AGE >= 100)/n(),
            greater_110 = sum(AGE >= 110)/n()) %>%
  group_by(GENDER) %>%
  gather("AGE_RANGE", "COUNT", 2:ncol(.)) %>%
  mutate(AGE_RANGE = as.numeric(str_extract(AGE_RANGE, "[[:digit:]]+")))
```

```r
plot4 <- ggplot() +
  geom_line(data = human_survivorship,
            aes(x = AGE_RANGE, y = COUNT, group = GENDER, colour = GENDER)) +
  theme_classic() +
  xlab("Age") +
  ylab("Proportion surviving")

plot4
```


Proportion surviving
1.00
0.75
0.50
0.25
0.00
0
30
60
90
Age
GENDER
F
M