# Lab 0: Introduction to R in 100 Lines or Less

SDS358: Applied Regression Analysis

*Michael J. Mahometa, Ph.D.*

.

> "You can't go from knowing nothing to becoming an expert without going through a period of great frustration and great suckiness."
>
> *Hadley Wickham*

## Introduction

R is awesome (at least that's my opinion). We can run analyses, keep a record of what we're doing with syntax, and even make nice looking output (like the page you are reading now). This little intro is just that - little and an intro. We'll cover the basics of using R: getting data in, manipulating it, and analyzing it. By the way, I use R and RStudio synonymous.

Now, the GOOD news. I'll never ask you to generate R syntax **from scratch**. We'll use our Lab time to look at R code to answer a research question. You can then literally copy that code, change it a little, and use it to answer the question for your take home Lab Assessment.

## First thing's first

For this class, we'll need to use a "helper" package. But it's not up on CRAN. It's my own creation, and you can find it (and install it) here:

```
install.packages("devtools") #if needed...
devtools::install_github("MichaelJMahometa/SDSRegressionR")

library(SDSRegressionR) #"Activate" the package with the library() command
```

We'll also need some other helpers:

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

## Basic R idea

R is really just a big calculator. We can do simple things like addition, subtraction, root, logs, etc.:

```
7 + 2
```

```
## [1] 9
```

One of the greatest things about R is that we can assign these results as *objects* for further use. Here, I give the result of "7+2" an to an object to something called "chicken" - yes, the names of objects are completely arbitrary.

```
object <- 7+2
```

At any time, we can call the object out to see it:

```
object
```

```
## [1] 9
```

And, we can do mathematical operations on that object (as long as it's numeric….)

```
object * 2
```

```
## [1] 18
```

We can also create different **kinds** of objects. This object is created using one of the most typically ways - using a "concatenation" of ideas (numbers, letters, strings, etc.). This will create what's called a "**vector**" (simply a gathering of similar items).

```
rooster <- c(7,2)
rooster
```

```
## [1] 7 2
```

And again, we can preform operations on this object:

```
rooster * 2
```

```
## [1] 14  4
```

And we can perform "**functions**" on objects.

```
sum(rooster)
```

```
## [1] 9
```

# Getting data into R

## Manual Input

It's great that R is a big calculator, but that's not really how we'll be using it. We'll be using it as a software tool to do work on DATA. And to to work on data, we need to get it into R.

## Reading in a CSV file

Manual input is possible with R, but not easy - it's slow and tedious. So, if you happen to have an Excel file, you can save it as a CSV file, then bring in the CSV file to RStudio. And, there are *TWO* ways to do this:

**1. Using the GUI (point and click) method in the most recent version of RStudio:**

Go to the **Environment** window, and select "Import Dataset."

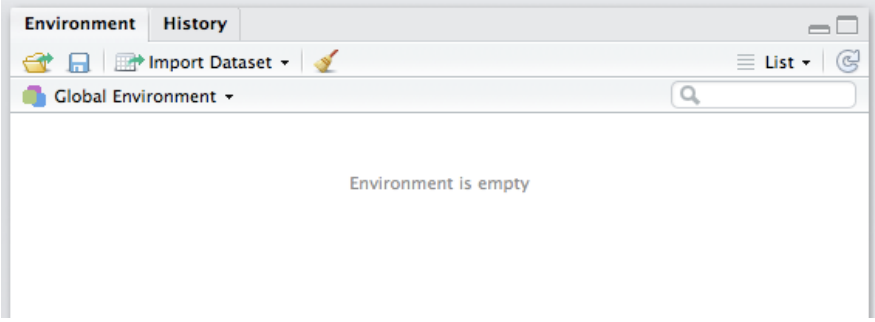



Figure 1: Environment Window

Then, click on the "From Text (readr)..." option:

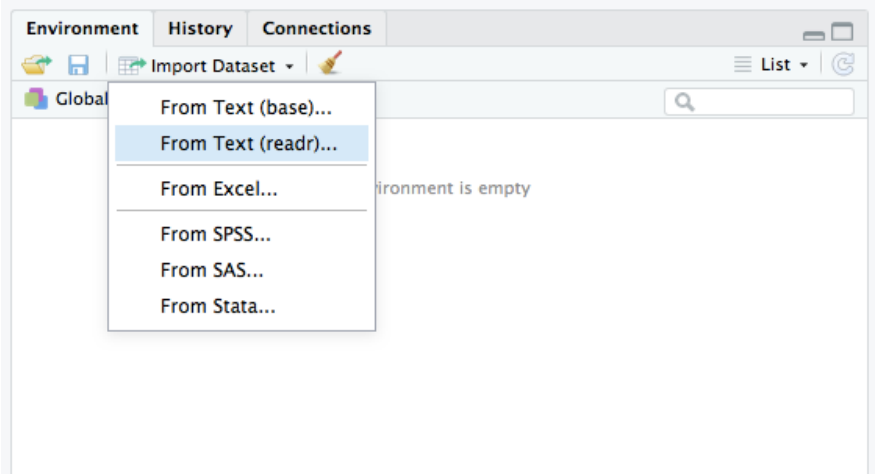



Figure 2: Importing

Navigate to the .csv file you want, and you'll see this window open up:

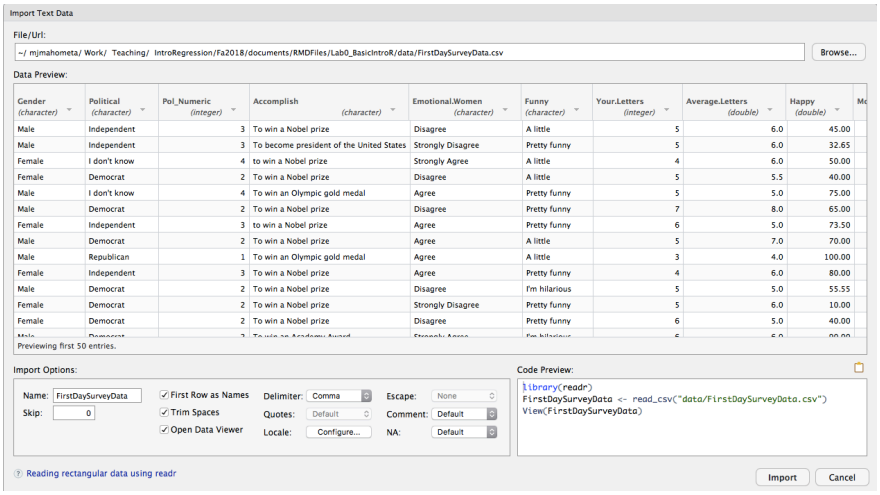



Figure 3: CSV Options Window

There are some things you'll need to pay attention to here:

- Make sure you *name* the data file what you want to use in RStudio (by default, it will be the name of the file). Here, I've named the data file "surv" for ease of typing in RStudio.
- Make sure the *"First Row as Names"* box it checked.

Now the data is in RStudio. (RStudio even shows you the data in the "Data View" window.)

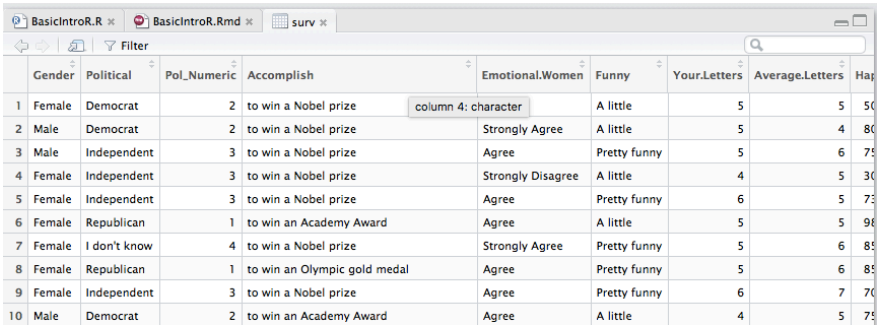



Figure 4: Data View

Another thing to pay attention to:

- Using this method leverages a new way to read in data (`read_csv()`). This creates a "*tibble*" instead of a dataframe. These *tibbles* are more useful in the long run, but *some* functions may not play nice with them (yet). Just something to be aware of...
- AND, as good practice, we'll want our data to have a *unique identifier*—to keep track of observations. If it doesn't have one, we'll need to add one in:

```
surv <- rownames_to_column(surv, var="UniqueID")
```

**2. Using syntax to get data in.**

The *other* way to get data into RStudio is with a function. This will require the use of the function in the syntax window, but it makes the **reproducability** of your work much higher. (Someone (me) needs only look at your syntax to see where you're inputting your data from.)

Here's how to import the same .csv file, but now with a function:

```
surv <- read_csv("data/FirstDaySurveyData.csv")

# Take a quick peek to confirm...
surv
```

```
## # A tibble: 123 x 20
##    Gender Political  Pol_Numeric Accomplish        Emotional.Women  Funny
##    <chr>  <chr>            <int> <chr>             <chr>            <chr>
##  1 Male   I don't k~           4 To win an Olympi~ Agree            Prett~
##  2 Male   Democrat             2 To win a Nobel p~ Strongly Disagr~ Prett~
##  3 Male   Independe~           3 To win a Nobel p~ Agree            Prett~
##  4 Female Democrat             2 To win a Nobel p~ Disagree         Prett~
##  5 Female I don't k~           4 To win a Nobel p~ Disagree         A lit~
##  6 Male   Democrat             2 To win a Nobel p~ Disagree         A lit~
##  7 Male   I don't k~           2 To win a Nobel p~ Agree            A lit~
##  8 Male   Independe~           3 To become presid~ Agree            Prett~
##  9 Female Democrat             2 To win a Nobel p~ Agree            A lit~
## 10 Female Democrat             2 To win a Nobel p~ Agree            I'm h~
## # ... with 113 more rows, and 14 more variables: Your.Letters <int>,
## #   Average.Letters <dbl>, Happy <dbl>, Mother.Born <int>,
## #   Father.Born <int>, Sleep.Tues <dbl>, Sleep.Sat <dbl>, Powers <dbl>,
## #   Exclusive <int>, Height <dbl>, `Shoe Size` <chr>, feet <int>,
## #   inches <dbl>, height.inches <dbl>
```

**A Note on Functions**

Functions are tools. R comes with a ton of them to start. BUT, as R evolves, other users add to those tools—they create their own functions—and share them with the rest of us. To do so, we need to get one or more **packages**—that hold the functions. Think of a package as an add-on. It increases the usability of R.

We'll be using some choice packages throughout the course to help with our regression modeling. In fact, this course even has it's own package to make things a little easier when we first get started. And, I'll be keeping a running list of R packages that will be needed for our labs (if it's used in any lab) on our Canvas site, along with the most up-to-date version of our own class's package: **SDSRegressionR**.

To get a package (and a function) that you don't have on onto your machine, you can either use the GUI (like bringing in data). Go to the "Packages" tab and select "Install."

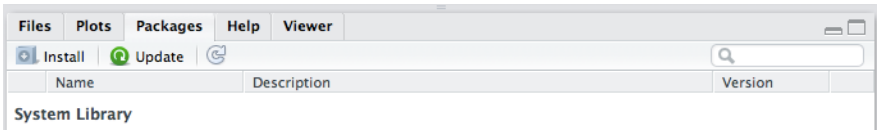



Figure 5: Packages Tab

Then, you'll see this window.

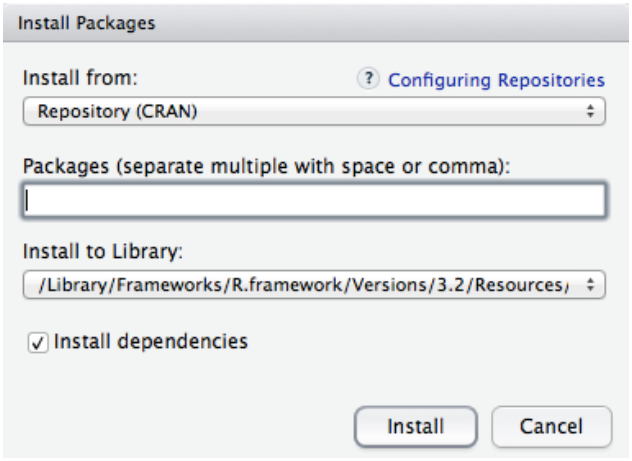



Figure 6: Package Selection Window

Just type in the package name, and you are good to go!

Now, we'll use the package with the `library()` function: (We only need to use the library function to load a new package once per R session.)

### Reading in other types of files

The most recent versions of RStudio can bring in lots of different versions of data (as we can see with the drop down menu: Excel data, SPSS, SAS, or Stata.

For the most part, I like importing a .csv file. They're easier and less dependent upon Excel or some other software. All of our data for labs will be in .csv format.

## Using R for Data Manuipulation

The main thing we'll be using R for is for regression analysis. And getting our data in is the first step in that process. But sometimes we need to work with our data before we can use it in the analysis. And to help, we'll be abiding by the rules and tools of the **tidyverse.**

Let's get the **tidyverse** package up and running:

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

First, let's take a look at the names of the variables in our dataset:

```
names(surv)
```

```
##  [1] "Gender"          "Political"       "Pol_Numeric"
##  [4] "Accomplish"      "Emotional.Women" "Funny"
##  [7] "Your.Letters"    "Average.Letters" "Happy"
## [10] "Mother.Born"     "Father.Born"     "Sleep.Tues"
## [13] "Sleep.Sat"       "Powers"          "Exclusive"
## [16] "Height"          "Shoe Size"       "feet"
## [19] "inches"          "height.inches"
```

Or, we can click the name of our data in the "Environment" window and we'll see our data in RStudio:

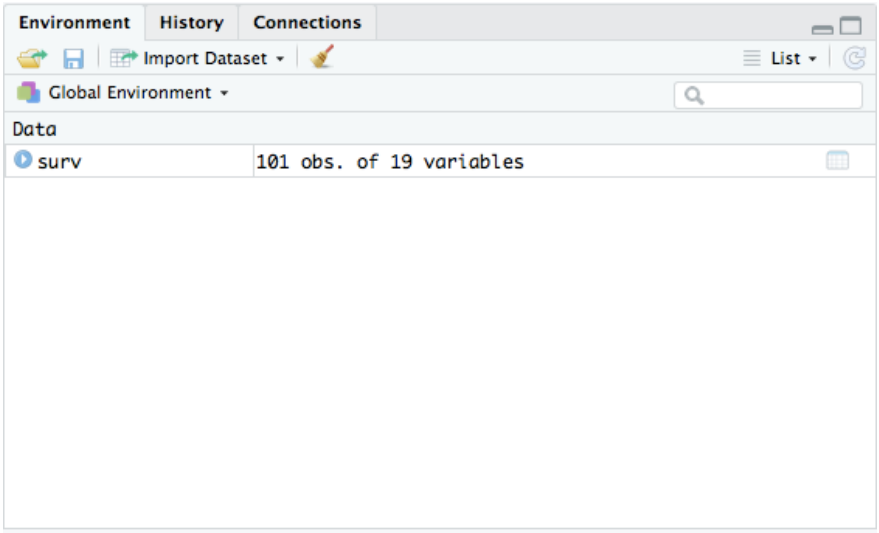



Figure 7: A New Dataset

Or, we can also use some syntax:

```
View(surv)
```

We can also look at variables within our "*Console*" window. Here's we can take a look at just one variable: Gender. There are TWO ways "call" a variable (as a vector) in using R and the **tidyverse**.

1. Using the `dataset$variable` method
2. Using the `pull()` function: (`pull(datatset, variable)`)

Both are valid. Sometime it's just easier to use one over the other. First, the `dataset$variable` method: We call the dataset (surv), and then the variable "Gender" with a **dollar sign** as the separator. Notice that R is case sensitive.

```
surv$Gender
```

The second method (`pull()`) is also valid. But, notice how the **tidyverse** functions don't usually require quotations.

```
pull(surv, Gender)
```

As we can also use variable numbers (the number of the variable in the dataset). Here, Gender is variable number 2 in the dataset, so the following also works:

```
pull(surv, 2)
```

We can easily call two variables (or more) at a time too using the `select()` function:

```
select(surv, Gender, Political)
```

We can even save our new selection of variables to a new dataset. This is often useful if you have a large number of variables and just want a simpler dataset to work with. Just don't forget the unique identifier variable:

```
small <- select(surv, UniqueID, Gender, Political)
```

## Subsetting Data

Sometime, we'll want a smaller portion of our data—not variables, but rows. We'll need to *filter* out observations that meet a certain criteria. So, we'll use the `filter()` function:

```
men <- filter(surv, Gender=="Male")
men_dem <- filter(surv, Gender=="Male" & Political=="Democrat")

sixsix <- filter(surv, height.inches >= 66)
sixsix_fem <- filter(surv, height.inches >= 66 & Gender=="Female")
```

## Creating new variables

### Recoding varaibles

The simplest "variable creation" is to make a new variable based on an existing one. For example, say we want to make a new variable that says "yes, this person was happy." We could look at our existing variable of "Happy" and decide on a cut-off.

To do this, we'll leverage two major functions from the **tidyverse**: `mutate()` to create new variables, and `case_when()` to follow the rules of assignment:

```
surv <- mutate(surv,
               Hap_cat = case_when(Happy < 75 ~ 0,
                                   Happy >= 75 & Happy <= 100 ~ 1))
```

Let's check to see if that worked (this is ALWAYS a good idea) with the `table()` function:

```
table(select(surv, Happy, Hap_cat)) #Needs factoring!
```

We can also recode categorical variables, truncating groups down:

```
surv <- mutate(surv, Emot = case_when(Emotional.Women == "Strongly Disagree" |
                                Emotional.Women == "Disagree" ~ 0,
                                Emotional.Women == "Strongly Agree" |
                                Emotional.Women == "Agree" ~ 1))
#Checking
table(select(surv, Emotional.Women, Emot))
```

```
##                     Emot
## Emotional.Women      0  1
##   Agree              0 60
##   Disagree          40  0
##   Strongly Agree     0 10
##   Strongly Disagree 13  0
```

With categorical variables, we'll often need to tell R to put them in the right "order"—called factoring:

```
surv <- mutate(surv,
               Emotional.Women = factor(Emotional.Women,
                                        levels=c("Strongly Disagree", "Disagree",
                                                 "Agree", "Strongly Agree")))
#Checking
table(select(surv, Emotional.Women, Emot))
```

```
##                     Emot
## Emotional.Women      0  1
##   Strongly Disagree 13  0
##   Disagree          40  0
##   Agree              0 60
##   Strongly Agree     0 10
```

### Arithmetic

We can create new variable based on a arithmetic idea too, which is often useful when we have something called a composite score. Here, we'll use the `mutate()` function again:

```
surv <- mutate(surv, par_diff = Father.Born - Mother.Born)
```

We can also use a function called "rowwise()" to help (and the "pipe" to help):

```
surv <- surv %>%
  rowwise() %>%
  mutate(par_sum = sum(c(Father.Born, Mother.Born), na.rm=TRUE)) %>%
  ungroup()

#This is the same, just harder to read...
surv <- mutate(rowwise(surv), par_sum2 = sum(c(Father.Born, Mother.Born), na.rm=TRUE))
```

Or, we can even get the mean of several columns by changing the `sum()` function to `mean()`":

```
surv <- surv %>%
  rowwise() %>%
  mutate(par_mean = mean(c(Father.Born, Mother.Born), na.rm=TRUE)) %>%
  ungroup()
```

This means that we can use R's common arithmetic functions also (like `min()`, `max()` and the like):

```
surv <- surv %>%
  rowwise() %>%
  mutate(par_max = max(c(Father.Born, Mother.Born), na.rm=TRUE)) %>%
  ungroup()
```

# Summarizing Our Data

We will oftentimes need to summarize our data, and R is also good at doing that. We can summarize both **categorical** and **quantitative** data.

## Single Categorical Variable Summarization

The easiest way to summarize categorical data is with the use of the `table()`, `addmargins()`, and `prop.table()` functions:

```
#Using dplyr notation
t <- table(select(surv, Gender))
t
```

```
##
## Female   Male
##     57     66
```

```
#Get more information
addmargins(t)
```

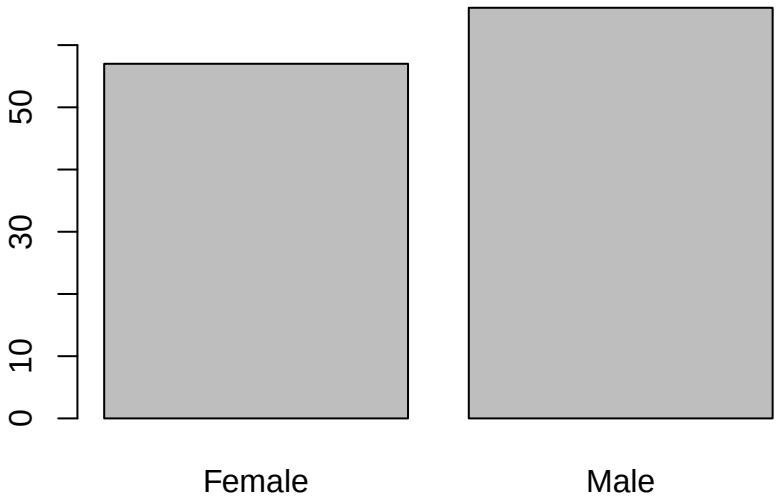
```
##
## Female   Male    Sum
##     57     66    123
```

```
prop.table(t)
```

```
##
##    Female      Male
## 0.4634146 0.5365854
```

It also helps to visualize the categorical summarization too using the "barplot()" function:

```
barplot(t)
```




```
barplot(prop.table(t))
```


## Two Categorical Variable Summarization

We can make a contingency table to help summarize two categorical variables:

```
t2 <- table(select(surv, Gender, Emotional.Women))
t2
```

```
##         Emotional.Women
## Gender   Strongly Disagree Disagree Agree Strongly Agree
##   Female                 8       16    28              5
##   Male                   5       24    32              5
```

```
addmargins(t2)
```

```
##         Emotional.Women
## Gender   Strongly Disagree Disagree Agree Strongly Agree Sum
##   Female                 8       16    28              5  57
##   Male                   5       24    32              5  66
##   Sum                   13       40    60             10 123
```

```r
prop.table(t2) #Table proportions
```

```
##          Emotional.Women
## Gender    Strongly Disagree   Disagree      Agree Strongly Agree
##   Female         0.06504065 0.13008130 0.22764228     0.04065041
##   Male           0.04065041 0.19512195 0.26016260     0.04065041
```

```r
prop.table(t2, 1) #Row proportions
```

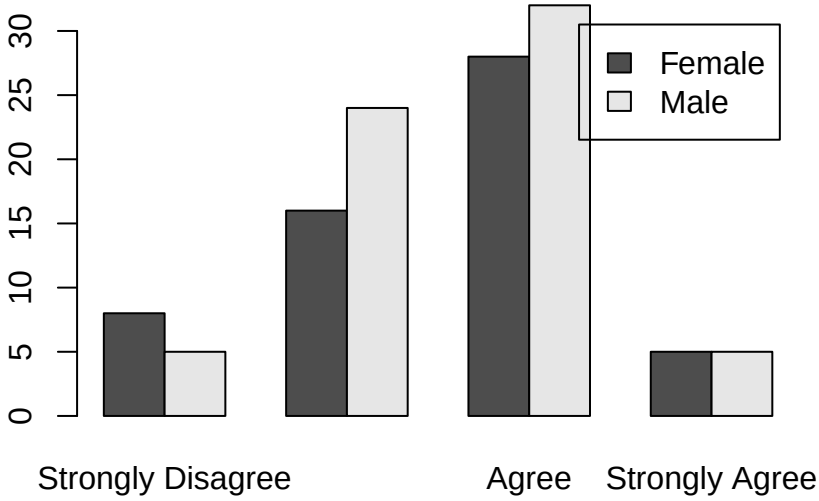
```
##          Emotional.Women
## Gender    Strongly Disagree   Disagree      Agree Strongly Agree
##   Female         0.14035088 0.28070175 0.49122807     0.08771930
##   Male           0.07575758 0.36363636 0.48484848     0.07575758
```

```r
prop.table(t2, 2) #Column proportions
```

```
##          Emotional.Women
## Gender    Strongly Disagree  Disagree     Agree Strongly Agree
##   Female          0.6153846 0.4000000 0.4666667      0.5000000
##   Male            0.3846154 0.6000000 0.5333333      0.5000000
```

And, we can visualize the results:

```r
barplot(t2, beside=TRUE, legend=TRUE)
```



## Single Quantitative Variable Summarization

We can use some other functions like `mean()`, `sd()`, `fivenum()`, and `hist()` to help us out with our summarization:

```
#Using base vector notation:
mean(surv$Happy, na.rm = TRUE)
```

```
## [1] 68.66164
```

```
sd(surv$Happy, na.rm = TRUE)
```

```
## [1] 21.81974
```

```
fivenum(surv$Happy)
```

```
## [1]  10  50  75  85 100
```

```
hist(surv$Happy)
```


## Single Quantitative Variable Summarization by Group

Often, it will be useful to see the summarization of a quantitative variable across multiple groups. To do this, we'll use **dplyr**.

```
surv %>%
  group_by(Gender) %>%
  summarise(mn = mean(Happy, na.rm=TRUE))
```

```
## # A tibble: 2 x 2
##   Gender    mn
##   <chr>  <dbl>
## 1 Female  70.7
## 2 Male    66.9
```

Or, for even *more* information, we can use the **skimr** package:

```
install.packages("skimr")
library(skimr)
```

```
surv %>%
  group_by(Gender) %>%
  skim(Happy)
```

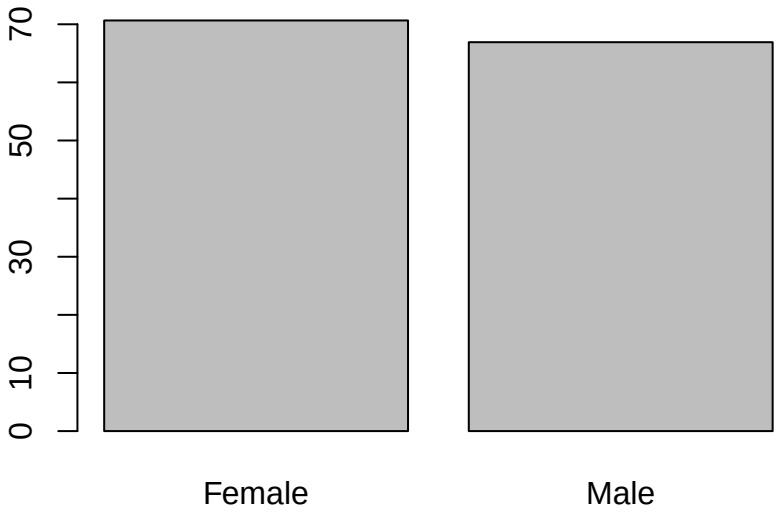
Skim summary statistics
n obs: 123
n variables: 27

Variable type: numeric

| Gender | variable | missing | complete | n | mean | sd | p0 | p25 | p50 | p75 | p100 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Female | Happy | 0 | 57 | 57 | 70.66 | 20.65 | 10 | 60 | 80 | 85 | 98 |
| Male | Happy | 1 | 65 | 66 | 66.91 | 22.81 | 11.11 | 50 | 75 | 85 | 100 |

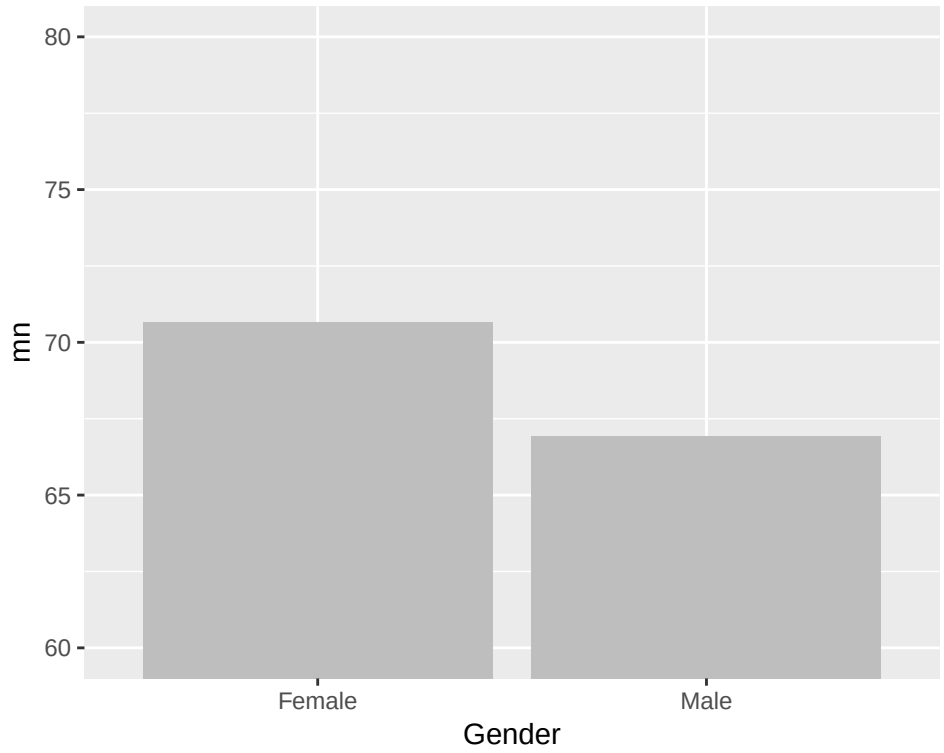
Again, the visualization helps also:

```
s <- surv %>%
  group_by(Gender) %>%
  summarise(mn = mean(Happy, na.rm=TRUE))
barplot(s$mn, names.arg=s$Gender)
```



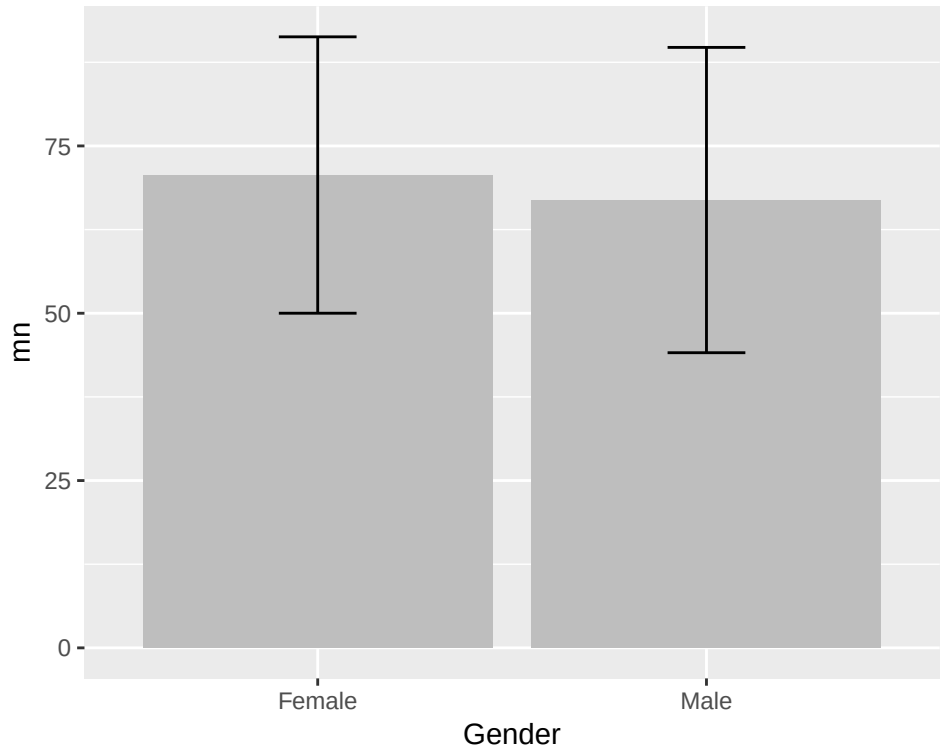

But, again, we can make this better with just a *little* more work:

```
library(ggplot2)
ggplot(s, aes(x=Gender, y=mn)) +
  geom_bar(stat="identity", fill="grey") +
  coord_cartesian(ylim=c(60,80))
```




And even better (all thanks to Hadley Wickham)...

```
s2 <- surv %>%
  group_by(Gender) %>%
  summarise(mn = mean(Happy, na.rm=TRUE),
            sd = sd(Happy, na.rm=TRUE))
ggplot(s2, aes(x=Gender, y=mn)) +
  geom_bar(stat="identity", fill="grey") +
  geom_errorbar(aes(ymin=mn-sd, ymax=mn+sd), width=.2)
```




## Last bit

And that's it. An introduction to R in 100 lines of code or less. Remember, I'll NEVER ask you to do this from scratch. I'll always give you the code that you can copy and alter to run your analysis.