Introduction to R and the Tidyverse

Welcome to SummeR of R at the Brandeis Library!

Who we are

Margarita Corral

mcorral@brandeis.edu

Make an appointment with Margarita!

Shannon Hagerty

shannonhagerty@brandeis.edu

Make an appointment with Shannon!

What is R and the tidyverse?

- R is an open source programming language that is very popular for data analysis/science
- You're using R in Rstudio and interacting with an Rmarkdown file.
- Because R is so popular for data analysis it's likely that someone has already written the code needed to do many tasks you want to do. This makes it much easier to jump right into R and get a lot of data analysis tasks done right away.
- Functions are actions you want done to something, there's code behind the function making that action reality. In R usually the function name indicates the action (but this totally depends on how good the creator was at naming). Whatever you want the action done on goes inside parentheses that immediately follow the function name. For example if I want to take the average of several numbers I can use a function called mean:

```
mean(c(99,95,93,44,87))
```

```
## [1] 83.6
```

I wanted to take the average of 5 test grades, I had to use another function c()...C for combine... that can combine multiple objects into one so that the mean function knows its taking the average of those 5 items.

- R comes with a bunch of functions built in (like mean() and c()) these are usually functions that are broadly useful. But you can bring in extra functions by downloading packages, packages are collections of functions.
- The tidyverse is an "opinionated collection of R packages designed for data science" - www.tidyverse.org
- Tidyverse packages are designed to make data wrangling, analysis, and graphing much simpler and more enjoyable.
- Tidyverse packages share a philosophy of data organization, i.e. they all expect tidy data

Tidy Data

Tidy data is set up so that each row is an observation and each column is a variable.

Consider these two tables where I guess which game of thrones characters will survive season 8.

Survives	Dies

Survives	Dies
Jon Snow	Daenerys
Arya	The Mountain
Sansa	Brienne
Samwell Tarly	
Gilly	

AS OPPOSED TO

Character	Prediction
Jon Snow	Survives
Daenerys	Dies
Arya	Survives
The Mountain	Dies
Sansa	Survives
Brienne	Dies
Samwell Tarly	Survives
Gilly	Survives

The second data set is tidy while the first table is not. The tidyverse has a package called **tidyr** that can help you convert your data set into a tidy format (e.g. you can convert from the first game of thrones table to the second with the **tidyr** function *gather()*).

A tidy data format makes it easier to use the rest of the tidyverse packages (and is also just easier to look at and understand).

Tidyverse workflow

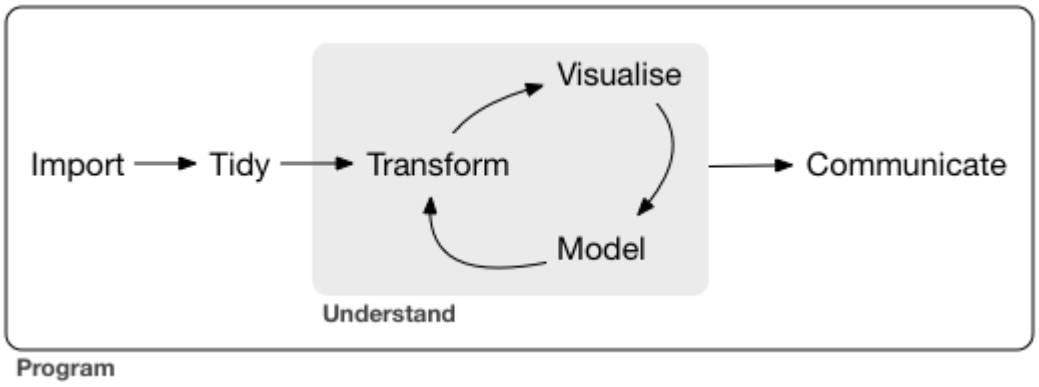


Image from *R for Data Science by Hadley Wickham and Garret Grolemund*

- Import the Data
- Get data into a tidy format
- Transform the data (may include grouping, averaging, calculating new variables etc.)
- Visualize
- Model
- Communicate

Install/load the Tidyverse

The very first time you want to use a package you first need to install it.

```
# this is a comment R, a hashtag lets R know this bit of text is just a note and is not meant to be run
install.packages('tidyverse')
```

You only need to install once. Then whenever you want to use the package you load it in to R with the library() function. This is analagous to how you download a program onto your computer but you still have to click to open it when you want to use it. The library() function tells R to open the package so you can use any functions inside it.

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------
## v ggplot2 3.1.1       v purrr   0.3.2
## v tibble  2.1.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.4.0
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Because tidyverse is actually a package of packages, the output of library() shows you which packages have been loaded and are now available for you to use (as well as their versions). If you load a single package there may be no output when you load the package, but you successfully loaded it as long as there is no error message.

It also lets you know conflicts. In this case the conflict means that we have loaded the dplyr packge and it has two functions that share a name with two functions from the stats package (i.e. filter and lag). So now if you call the function filter() it is going to use the dplyr function names filter NOT the stats package function named filter.

Does function masking matter? We can get info on what each of these functions do by putting a ? in front of the function to bring up the help documentation. We can specify which filter() function we are talking abput by using the notation 'package_name :: function_name()'

```
?stats::filter()
```

and compare the filter function in stats to dplyr, you can also use the help function

```
?dplyr::filter()
```

Sessions Goals

1. Get a general idea of the tidyverse and workflow
2. Read data into R
3. Create summaries from the data
4. Create a plot

1. Import Data

One of the packages we loaded in with the tidyverse was readr which has the function read_csv() this function brings csv files into R and we're going to us it now to load our data. We're going to be looking at data from the World Happiness Report

We're going to want to do plotting and analysis with the data so we need to not just read it in but tell R to remember it so we can do these things later on.You can do this by assigning a name for R to call the data frame and remember it by. Let's call it world_happiness.

```
world_happiness <- read_csv('world_happiness.csv')
```

```
## Parsed with column specification:
## cols(
##   Country = col_character(),
##   Year = col_double(),
##   Happiness_score = col_double(),
##   Log_GDP_per_cap = col_double(),
##   Social_support = col_double(),
##   Life_expectancy = col_double(),
##   Freedom = col_double(),
##   Generosity = col_double(),
##   Perception_corruption = col_double(),
##   Government_confidence = col_double(),
##   Democratic_quality = col_double()
## )
```

The assignment operator <- is used to save why usea thing with a variable name. (R will also let you use = to assign a variable name but its better to use the <- because = can do different things in different contexts while <- always creates a variable, so its good form to avoid = in variable creation)

We have now successfully loaded in the world happiness dataset. In the top right panel under the environment tab you should now see world_happiness listed. That panel is where R lists any variables you have asked it to remember. When you use read_csv it lets you know that the dataframe has been read in and it gives you the column names as well as the type of data stored in that column (col_character is treated as a string/word and col_double is a number).

```
#try creating any variable say x is 5 or if you can save a word or letter to a variable name by saying
```

You can use the View function to look at your dataframe.

```
View(world_happiness)
```

We can check out the dimensions of our dataframe with the dim() function:

```
dim(world_happiness)
```

```
## [1] 1704   11
```

There are 1704 rows and 11 columns.

We can also print the dataframe in the Rmd.

```
world_happiness
```

```
## # A tibble: 1,704 x 11
##    Country  Year Happiness_score Log_GDP_per_cap Social_support
##    <chr>   <dbl>           <dbl>           <dbl>          <dbl>
##  1 Afghan~  2008            3.72            7.17          0.451
##  2 Afghan~  2009            4.40            7.33          0.552
##  3 Afghan~  2010            4.76            7.39          0.539
##  4 Afghan~  2011            3.83            7.42          0.521
##  5 Afghan~  2012            3.78            7.52          0.521
##  6 Afghan~  2013            3.57            7.52          0.484
##  7 Afghan~  2014            3.13            7.52          0.526
##  8 Afghan~  2015            3.98            7.50          0.529
```

```
##  9 Afghan~  2016            4.22            7.50          0.559
## 10 Afghan~  2017            2.66            7.50          0.491
## # ... with 1,694 more rows, and 6 more variables: Life_expectancy <dbl>,
## #   Freedom <dbl>, Generosity <dbl>, Perception_corruption <dbl>,
## #   Government_confidence <dbl>, Democratic_quality <dbl>
```

We can get a general summary for the dataframe using summary()

```
summary(world_happiness)
```

```
##    Country               Year      Happiness_score Log_GDP_per_cap
##  Length:1704        Min.   :2005   Min.   :2.662   Min.   : 6.457
##  Class :character   1st Qu.:2009   1st Qu.:4.611   1st Qu.: 8.304
##  Mode  :character   Median :2012   Median :5.340   Median : 9.406
##                     Mean   :2012   Mean   :5.437   Mean   : 9.222
##                     3rd Qu.:2015   3rd Qu.:6.274   3rd Qu.:10.193
##                     Max.   :2018   Max.   :8.019   Max.   :11.770
##                                                    NA's   :28
##  Social_support   Life_expectancy    Freedom         Generosity
##  Min.   :0.2902   Min.   :32.30   Min.   :0.2575   Min.   :-0.33638
##  1st Qu.:0.7475   1st Qu.:58.30   1st Qu.:0.6384   1st Qu.:-0.11553
##  Median :0.8331   Median :65.00   Median :0.7527   Median :-0.02208
##  Mean   :0.8106   Mean   :63.11   Mean   :0.7338   Mean   : 0.00008
##  3rd Qu.:0.9044   3rd Qu.:68.30   3rd Qu.:0.8482   3rd Qu.: 0.09352
##  Max.   :0.9873   Max.   :76.80   Max.   :0.9852   Max.   : 0.67774
##  NA's   :13       NA's   :28      NA's   :29       NA's   :82
##  Perception_corruption Government_confidence Democratic_quality
##  Min.   :0.0352        Min.   :0.06877       Min.   :-2.4482
##  1st Qu.:0.6961        1st Qu.:0.33473       1st Qu.:-0.7905
##  Median :0.8058        Median :0.46411       Median :-0.2274
##  Mean   :0.7513        Mean   :0.48197       Mean   :-0.1361
##  3rd Qu.:0.8765        3rd Qu.:0.61486       3rd Qu.: 0.6505
##  Max.   :0.9833        Max.   :0.99360       Max.   : 1.5750
##  NA's   :96            NA's   :174           NA's   :146
```

With the exception of variables are numeric so R calculated basic stats for each column and lets us know where we have missing data (the numbers follwoing NA's

2. Filter the Data

The tidyverse has several packages that allow you to transform your data. * Stringr lets you manipulates strings * lubridate has functions at make working with dates and times easier * dplyr allows you to manipulate dataframes

We have 1704 observations and each observation is a given country's happiness score in a certain year. Let's start exploring the data but let's start small.

Let's just look at one country. We're going to use: filter() - this function lets you grab certain rows out of a data frame. So if I want to grab just the United States happiness data I could do this:

```
United_States<-filter(world_happiness,Country == 'United States' ) #filter the world happiness data fra
```

Inside filter, the first bit of information it needs is the dataframe its going to filter rows from then you can add the conditional statement you want R to filter by (i.e. rows where Country is equal to 'United States'). Then you tell it the Column name you want to filter on conditionally and you say what you want value of that column to be *exactly equal to* by using the == (double equal signs) and giving the desired value. Note:

we could not have filtered for USA, US, or United States of America because that was not how the country was entered in the dataframe. It also will not work if you do not put quotes around 'United States'

How do we know what to put inside a function? We can check out the documentation:

```
?dplyr::filter()
```

We can also refer to rstudio cheat sheet for dplyr package The cheat sheet also has helpful visuals showing how data is wrangled.

Let's confirm the new dataframe we created looks how we think it should

```
United_States
```

```
## # A tibble: 13 x 11
##    Country  Year Happiness_score Log_GDP_per_cap Social_support
##    <chr>   <dbl>           <dbl>           <dbl>          <dbl>
##  1 United~  2006            7.18            10.8          0.965
##  2 United~  2007            7.51            10.8         NA
##  3 United~  2008            7.28            10.8          0.953
##  4 United~  2009            7.16            10.8          0.912
##  5 United~  2010            7.16            10.8          0.926
##  6 United~  2011            7.12            10.8          0.922
##  7 United~  2012            7.03            10.8          0.903
##  8 United~  2013            7.25            10.8          0.925
##  9 United~  2014            7.15            10.9          0.902
## 10 United~  2015            6.86            10.9          0.904
## 11 United~  2016            6.80            10.9          0.897
## 12 United~  2017            6.99            10.9          0.921
## 13 United~  2018            6.88            10.9          0.904
## # ... with 6 more variables: Life_expectancy <dbl>, Freedom <dbl>,
## #   Generosity <dbl>, Perception_corruption <dbl>,
## #   Government_confidence <dbl>, Democratic_quality <dbl>
```

Looks good!

YOU TRY IT: Filter

Filter for another country, remember the country name must be in quotes and needs to match the way the country name is in the dataframe exactly. You'll want to give it a variable name that makes sense too.

```
#Change this code
United_States<-filter(world_happiness,Country == 'United States' )
```

Print your dataframe to make sure it looks how you think it should.

```
#Check out your dataframe
```

Visualize

Let's try plotting the United_States data, let's check out Happiness_score over time.

We'll use the main tidyverse visualization package: **ggplot2**

ggplot uses layers to build graphics. You have to start with the ggplot function and inside that you tell it what data set it is going to use:

```
ggplot(data = United_States)
```

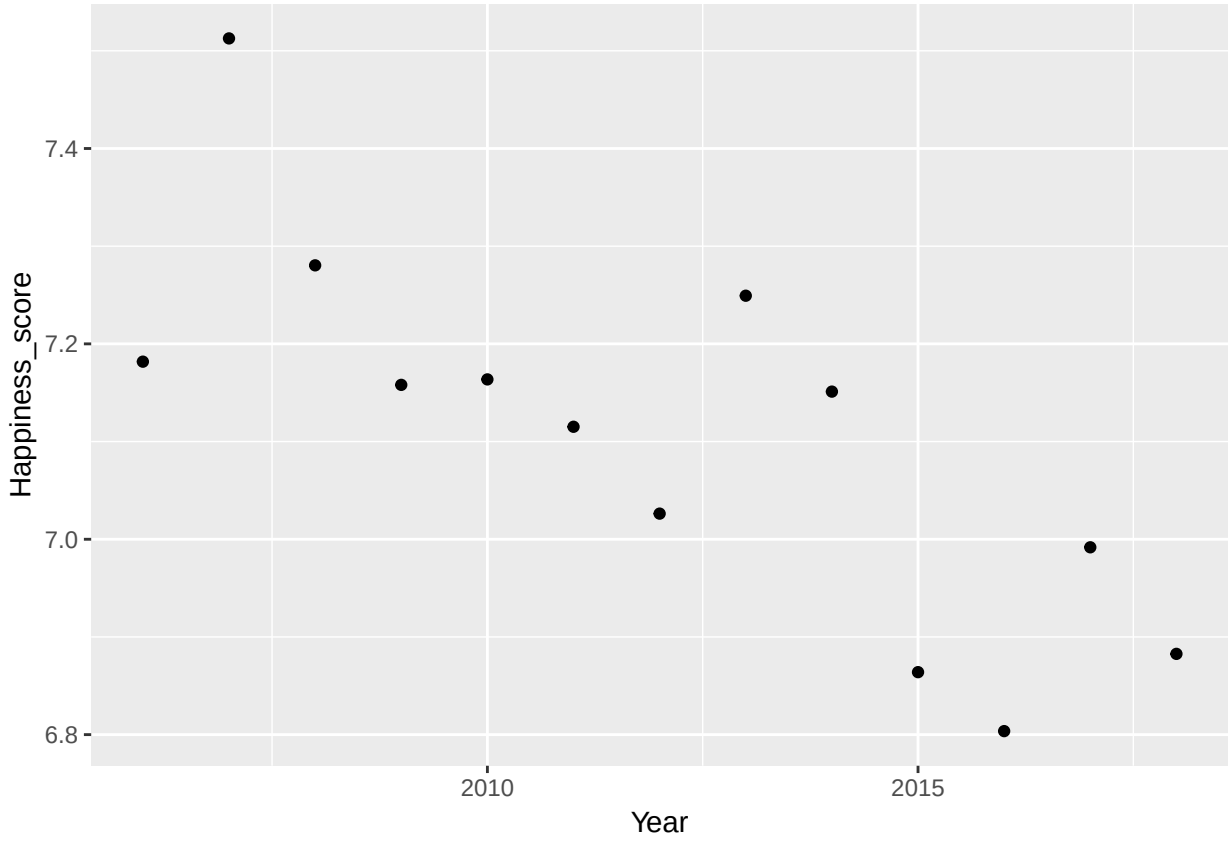
That would be our first layer, we want to build a scatterplot graph though where we have Year on the x axis and Happiness_score on the y axis. We'll add the next layer a geom layer, which sets the shape we want the data to take. We will use geom_point() to plot the points.

Inside the geom layer we have to tell ggplot where to put the shape, these are called the mapping aesthetics. For this plot our mapping aesthetics are just x and y axis.

```
geom_point(mappings = aes(x=Year, y=Happiness_score))
```

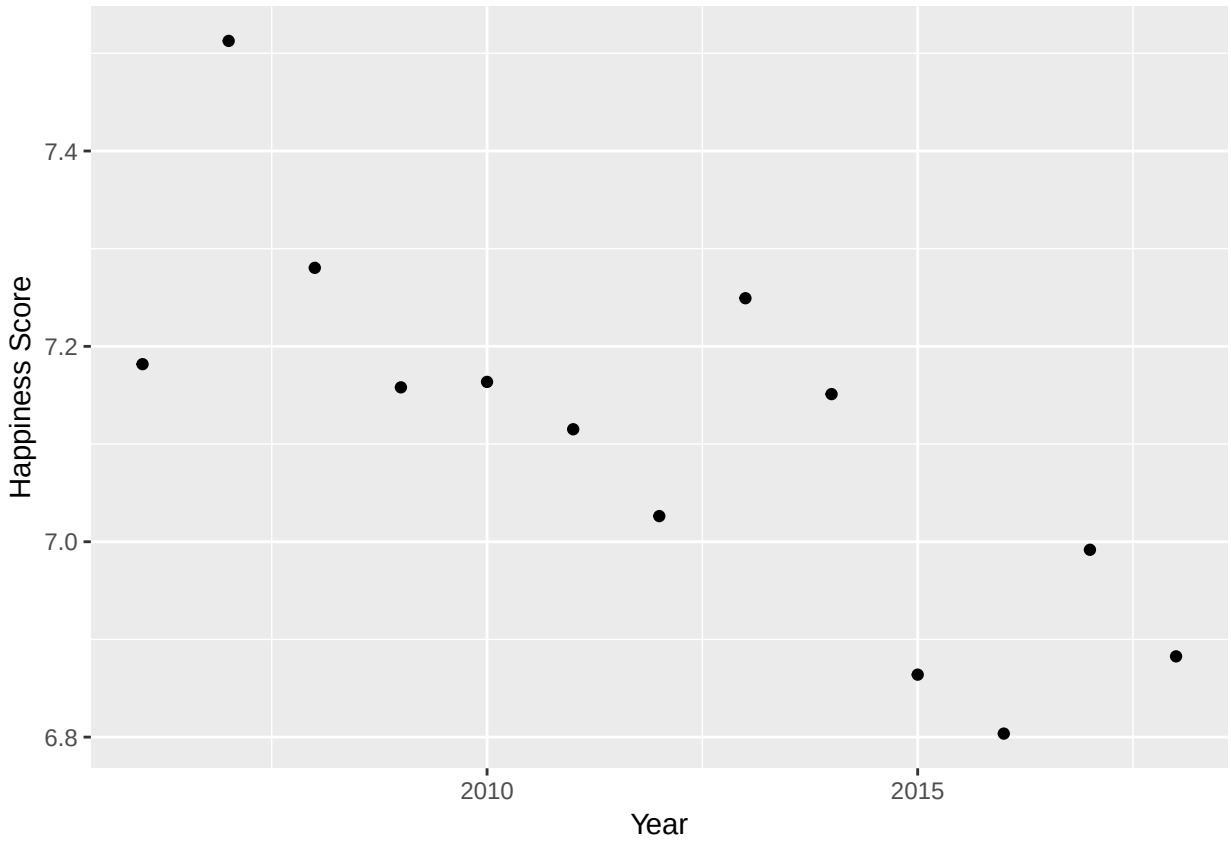
We put the layers together with a plus sign (NOTE ggplot() is always the first layer.)

```
ggplot(data=United_States)+geom_point(mapping=aes(x=Year, y=Happiness_score))
```



You can add more layers to build the graph you want, for example ggplot automatically takes the variable names for the x and y axis. This works out okay for Year but we will want to change the y axis label, you can do this by just adding another layer with ylab()

```
ggplot(data=United_States)+geom_point(mapping=aes(x=Year, y=Happiness_score))+ylab('Happiness Score')
```

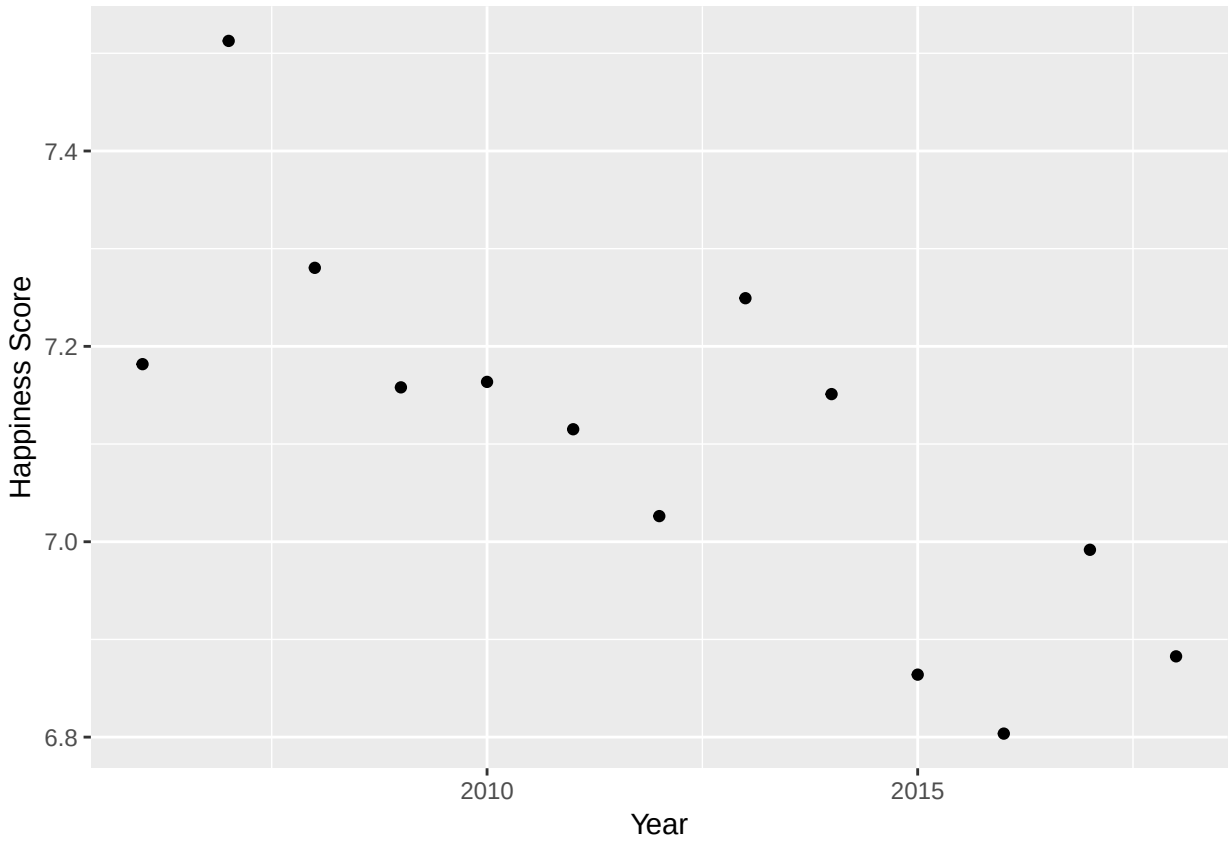


Looks like we're all getting less happy.

YOU TRY IT: ggplot

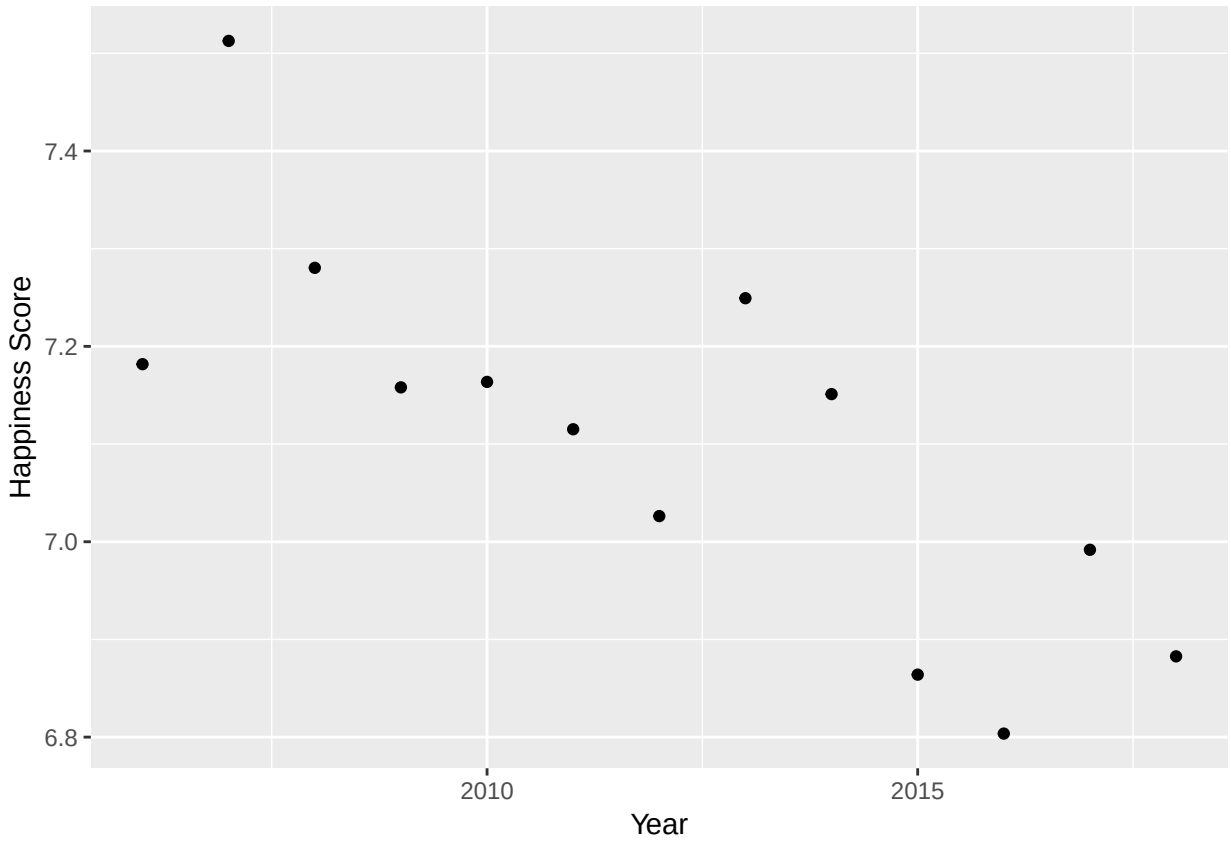
Try changing the x value to another variable from the United_States dataframe that you think might correlate to happiness score. Don't forget to change the y axis label too! And remember that the variable name must match the column name exactly!

```
ggplot(data=United_States)+geom_point(mapping=aes(x=Year, y=Happiness_score))+ylab('Happiness Score')
```



Okay now try adapting the code below to graph happiness score over time for the country you filtered for before in the You Try It: Filter steps.

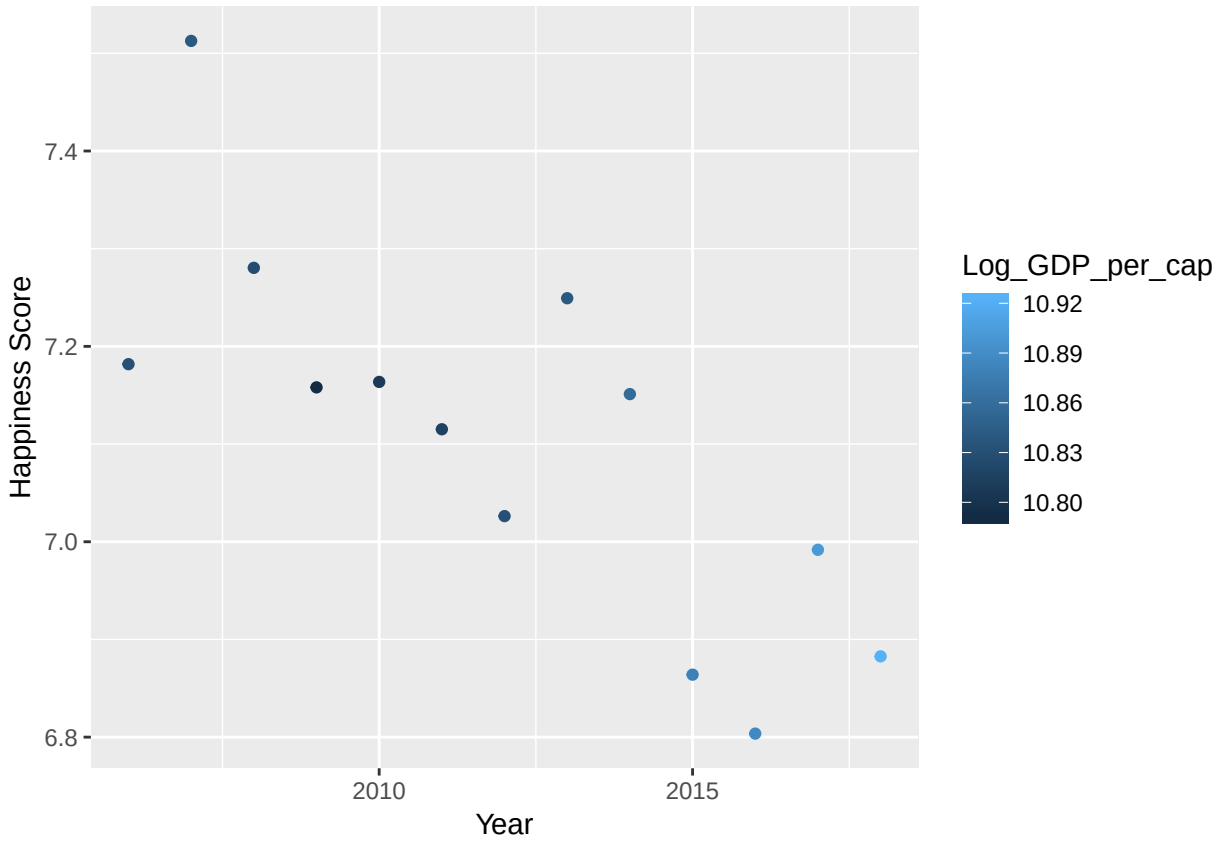
```
ggplot(data=United_States)+geom_point(mapping=aes(x=Year, y=Happiness_score))+ylab('Happiness Score')
```



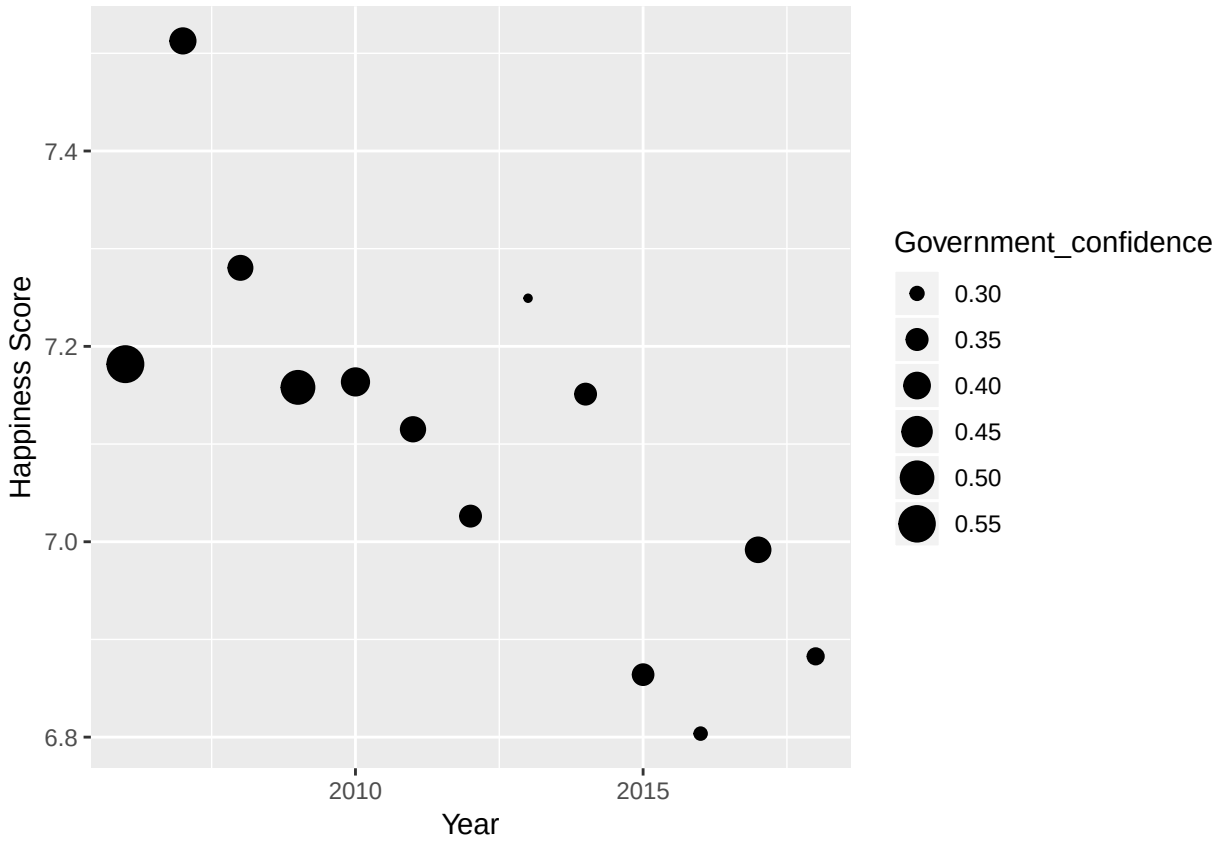
More ggplot options

Another way that we can see multiple variables at once is to have the color or size of points change as a function of other variables. This just goes inside the aesthetic mapping in geom_point, because that's the part of the code that tells R how to plot the points.

```
ggplot(data=United_States)+geom_point(mapping=aes(x=Year, y=Happiness_score, color=Log_GDP_per_cap))+yla
```




```
ggplot(data=United_States)+geom_point(mapping=aes(x=Year, y=Happiness_score, size=Government_confidence)
```



Filter for multiple things

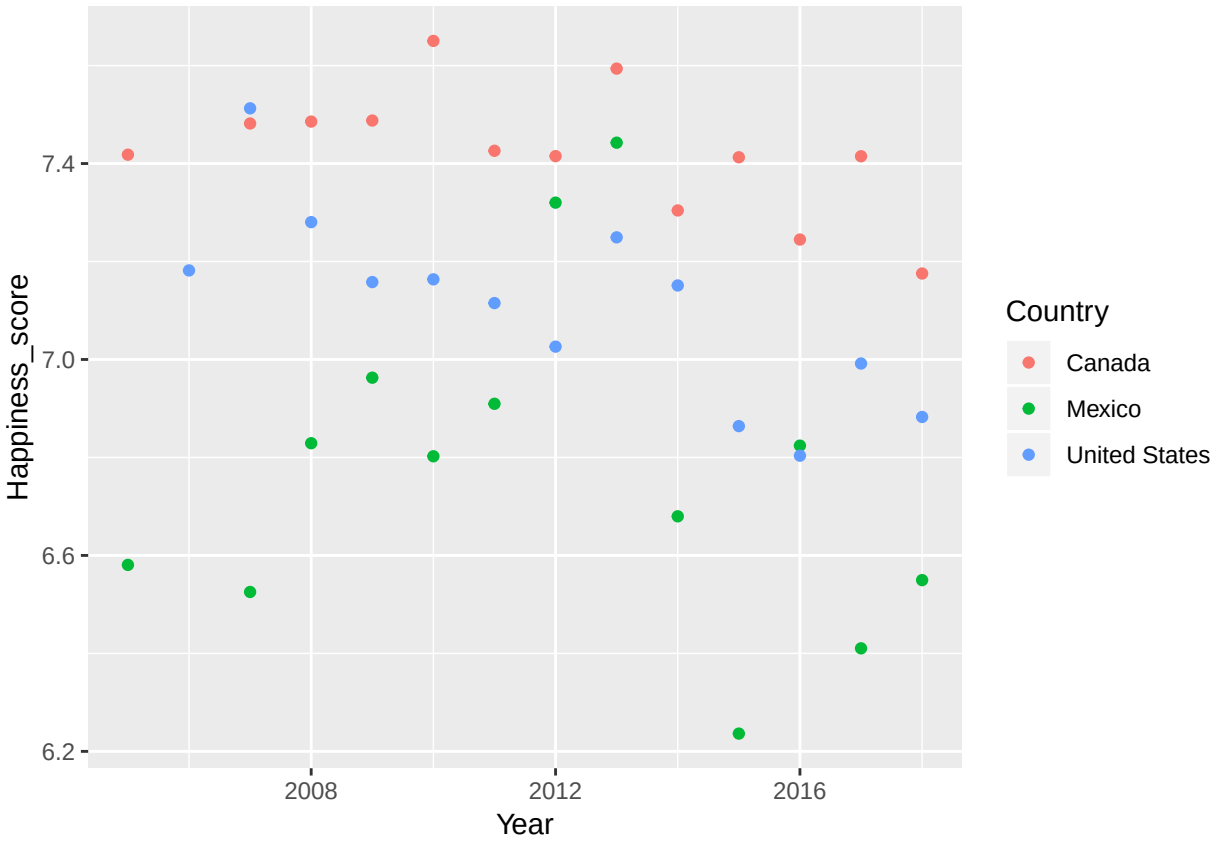
Another thing we may want to try is to filter for more than one country.Maybe we want to compare the happiness of Mexico, United States, and Canada. We need to filter the world_happiness dataframe for rows that have any of the three Countries in the country column. We can use %in% for this instead of == then put the names of the countries inside c()

```
US_and_neighbors<- filter(world_happiness, Country %in% c("United States", "Mexico", "Canada"))
```

This code tells R to filter world_happiness for rows where Country is one of the values in the list we created with c()

Now we can see how Mexico and Canada's happiness compares to US over time.Let's make a scatter plot where x is Year and Y is Happiness_score and points are colored by country.

```
ggplot(US_and_neighbors)+geom_point(aes(x=Year, y=Happiness_score, color=Country))
```



Multiple column filter, select, chaining

If you look closely at the graph we can see we have all three countries happiness scores after 2007, we only have two points for 2006 and one for 2007. Let's filter those years out, we can do this by adding to our original filter Year > 2007. Because Year is numeric we can use > or < or <= or >= to filter as well as the ==.

```
US_and_neighbors<- filter(world_happiness, Country %in% c("United States", "Mexico", "Canada"), Year >2
```

Since we're only insterested in the Happiness_score, Year, and Country we may want to just pull out these columns. We can use the select() function to do this. Inside the parentheses of select we have to first name the dataframe we want to select from, then name the columns we want.

```
US_and_neighbors <- select(US_and_neighbors, Country, Year, Happiness_score)
```

Let's check out what the final dataframe looks like

```
US_and_neighbors
```

```
## # A tibble: 33 x 3
##    Country  Year Happiness_score
##    <chr>   <dbl>           <dbl>
##  1 Canada   2008            7.49
##  2 Canada   2009            7.49
##  3 Canada   2010            7.65
##  4 Canada   2011            7.43
##  5 Canada   2012            7.42
##  6 Canada   2013            7.59
##  7 Canada   2014            7.30
##  8 Canada   2015            7.41
```

```
##  9 Canada   2016            7.24
## 10 Canada   2017            7.41
## # ... with 23 more rows
```

That looks like what we want. Let's just recap the steps we took to get there. We had the world_happiness dataframe and then we filtered for the rows and created US_and_neighbors dataframe and then we selected from US_and_neighbors the columns we wanted and we *overwrote* the old US_and_neighbors dataframe with the one that only had the columns we wanted. It looked like this:

```
US_and_neighbors<- filter(world_happiness, Country %in% c("United States", "Mexico", "Canada"), Year >20
US_and_neighbors <- select(US_and_neighbors, Country, Year, Happiness_score)
```

A faster way to write this (and I think also a way that is more readable) is to use chaining. The tidyverse has a pipe operator %>% which is like saying the words 'and then'. Recap of above: We had the world_happiness dataframe **and then** we filtered for the rows and created US_and_neighbors dataframe **and then** we selected from US_and_neighbors the columns we wanted.

```
US_and_neighbors<- world_happiness%>% filter(Country %in% c("United States", "Mexico", "Canada"), Year >
```

Note that we no longer have to put the dataframe name inside filter() or select() because when the pipe operator Print the dataframe and confirm this produced what we think it should.

```
US_and_neighbors
```

```
## # A tibble: 33 x 3
##    Country  Year Happiness_score
##    <chr>   <dbl>           <dbl>
##  1 Canada   2008            7.49
##  2 Canada   2009            7.49
##  3 Canada   2010            7.65
##  4 Canada   2011            7.43
##  5 Canada   2012            7.42
##  6 Canada   2013            7.59
##  7 Canada   2014            7.30
##  8 Canada   2015            7.41
##  9 Canada   2016            7.24
## 10 Canada   2017            7.41
## # ... with 23 more rows
```

Group By and Summarise

We have ten years of data for three countries. We may want to calculate the decadal average happiness for each country.

```
US_and_neighbors %>% group_by(Country)%>% summarise(avg_happiness=mean(Happiness_score))
```

```
## # A tibble: 3 x 2
##   Country       avg_happiness
##   <chr>                 <dbl>
## 1 Canada                 7.42
## 2 Mexico                 6.82
## 3 United States          7.06
```

What to do during hacky hour?

- Work on your own project!

- TidyTuesday: *This week is wine data you can visit the github page for more info. The Tidy Tuesday moderator provides you with the line of code to read in the data from the github page.* `wine_ratings <- readr::read_csv("https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2019/201` You can create some cool graphs/ summaries from what we did here today. Some ideas: A graph showing the relationship between price of wine and points by wine raters, calculate the average score or price for each variety of wine, or each year(then graph it!).
  - Check out the ggplot cheat sheet
- Go over example below:

Work thru this example

It seems to me that government confidence and perception of corruption should be related. A higher score (closer to 1) indicates more confidence / perception of corruption. Seemingly these should be negatively correlated. I want to see this relationship and compare it between the UK, the US, and Germany.

1. First things first we want to filter the data for the three countries we're interested in.

```
#Create a variable that is the filtered world happiness data where Country is %in% the list c('United K
```

2. Let's make the plot! We want x axis to be perception of corruption and the y axis to be government confidence

```
#ggplot(data = WHATEVER YOU CALLED THE VARIABLE IN PREVIOUS STEP) + geom_point(mappings = aes(x=COLUMN_
```

Interesting it looks like the US loses confidence in the government more quickly when people perceive corruption. Next let's clean the graph up a little. Improve the x label and y label.

```
#Same code as above but add +xlab('YOUR LABEL HERE') and +ylab('YOUR LABEL HERE')
```

3. Let's check the trends of government confidence and perception of corruption for the the three countries over time. First government confidence

```
#make a plot where x is
```

And now perception of corruption over time

```
#code here
```

4. Can we get the mean values of corruption and government confidence for each of the three countries?

```
#YOUR DATAFRAME %>% group_by(Country) %>% summarise(avg_corruption_perception= mean(Perception_corrupti
```

5. What if we wanted to see how the trends were changing over time for corruption perception and goverment confidence, and we wanted the average of the three countries for each measurement for each year. Can you adapt the code you wrote above to make this happen?

```
#your code here
```

Try playing around with the dataset a bit more if you have time. You could get the average happiness for the whole world_happiness dataframe for each year to see if the world's happiness is trending positively or negatively. Or try filtering and comparing other countries results.

Bonus preview for next week

ggplots can be customized so much! One quick way they can change is adding a theme layer which alters the look of the plot

```
#some themes come standard with ggplot
ggplot(US_and_neighbors)+geom_point(aes(x=Year, y=Happiness_score, color=Country))+ylab('Happiness Score
```

```r
ggplot(US_and_neighbors)+geom_point(aes(x=Year, y=Happiness_score, color=Country))+ylab('Happiness Score

#I like the ggthemes package!
#uncomment below if you need to install
#install.packages('ggthemes')

library(ggthemes)
ggplot(US_and_neighbors)+geom_point(aes(x=Year, y=Happiness_score, color=Country))+ylab('Happiness Score
ggplot(US_and_neighbors)+geom_point(aes(x=Year, y=Happiness_score, color=Country))+ylab('Happiness Score
ggplot(US_and_neighbors)+geom_point(aes(x=Year, y=Happiness_score, color=Country))+ylab('Happiness Score
ggplot(US_and_neighbors)+geom_point(aes(x=Year, y=Happiness_score, color=Country))+ylab('Happiness Score
ggplot(US_and_neighbors)+geom_point(aes(x=Year, y=Happiness_score, color=Country))+ylab('Happiness Score
```