Lab manual QuantRMA

Matthew J. C. Crump, Anjali Krishnan, Stephen Volz, and Alla Chavarga
Adapted for use at EUC by Thomas Hulst and Thanos Kostopoulos

Last Compiled 2020-12-16

Contents

Preface

Important notes

This is the lab manual for Quantitative Research Methods & Analysis at EUC. As with the textbook, this manual was adapted from "Answering questions with data" by Mattew J.C. Crump.

The original text is part of a larger OER (Open Educational Resource) course for teaching undergraduate statistics in psychology. As such, the text assumes you are a psychology student and many of the examples are drawn from the field of psychology. This does not mean that this course is only useful for you if you have an interest in psychology. The field of psychology will serve as a vehicle to teach you important concepts and skills in quantitative research methods and data analysis, but the concepts and skills taught are universal.

This manual provides the exercises we will work on during labs. We use open-data sets that are usually paired with a primary research article. The manual is a free and open resource. See below for more information about copying, making change, or contributing to the lab manual.

Attributions

The original lab manual was authored by Matt Crump with exercises adapted and expanded from Open Stats Labs.

CC BY-SA 4.0 license

- Adapt: remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

- Attribution: You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- ShareAlike: If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
- No additional restrictions: You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Getting started

In this course we will be using R as a tool to analyze data, and as a tool to help us gain a better understanding of what our analyses are doing. Throughout each lab we will show you how to use R to solve specific problems, and then you will use the examples to solve assignments. R is a very deep programming language, and in many ways we will only be skimming the surface of what R can do. Along the way, there will be many pointers to more advanced techniques that interested students can follow to become experts in using R for data-analysis, and computer programming in general.

R is primarily a computer programming language for statistical analysis. It is *free*, and *open-source* (many people contribute to developing it), and runs on most operating systems. It is a powerful language that can be used for all sorts of mathematical operations, data-processing, analysis, and graphical display of data. I even used R to write this lab manual. And, I use R all the time for my own research, because it makes data-analyis fast, efficient, transparent, reproducible, and exciting.

0.1 Why R?

There are lots of different options for using computers to analyze data, so why use R?[1] The options all have pros and cons, and can be used in different ways to solve a range of different problems. Some software allows you to load in data, and then analyze the data by clicking different options in a menu. This can sometimes be fast and convenient. For example, once the data is loaded, all you have to do is click a couple buttons to analyze the data!

[1] Other well-known statistical software packages are:

- SPSS
- SAS
- JASP
- Julia
- Matlab
- Python with SciPy packages

However, many aspects of data-analysis are not so easy. For example, particular analyses often require that the data be formatted in a particular way so that the program can analyze it properly. Often times when a researcher wants to ask a new question of an existing data set, they have to spend time re-formatting the data. If the data is large, then reformatting by hand is very slow, and can lead to errors. Another option, is to use a scripting language to instruct the computer how reformat the data. This is very fast and efficient. R provides the ability to do everything all in one place. You can load in data, reformat it any way you like, then analyze it anyway you like, and create beautiful graphs and tables (publication quality) to display your findings. Once you get the hang of R, it becomes very fast and efficient.

0.2 Installing R and R Studio

Download and install R onto your computer. The website for downloading R is: https://cloud.r-project.org. Make sure to select the version of R which is right for your operating system (i.e. Windows for Windows and Mac OS for Mac's).

After you have installed R on your computer, you should want to install another program called R Studio. This program provides a user-friendly interface for using R. You must already have installed R before you perform this step. The R Studio website is: http://www.rstudio.com.

Find the download link on the front-page, and then download R Studio desktop version for your computer. After you have installed R Studio you will be ready to start using R.

The website R Studio Cloud even allows you to run R scripts in the cloud (with some minor limitations), so you can also practice R from your web-browser!

0.3 R Studio notes and tips

0.3.1 Console

When you open up R studio you will see three or four main windows (the placement of each are configurable). In the above example, the bottom left window is the command line (terminal or console) for R. This is used to directly enter commands into R. Once you have entered a command here, press enter to execute the command. The console is useful for entering single lines of code and running them. Often times this occurs when you are learning how to correctly execute a line of code in R. Your first few attempts may be incorrect resulting in errors, but trying out different variations on your code in the command line can help you produce the correct code. Pressing the up arrow while in the console will scroll through the most recently executed lines of code.

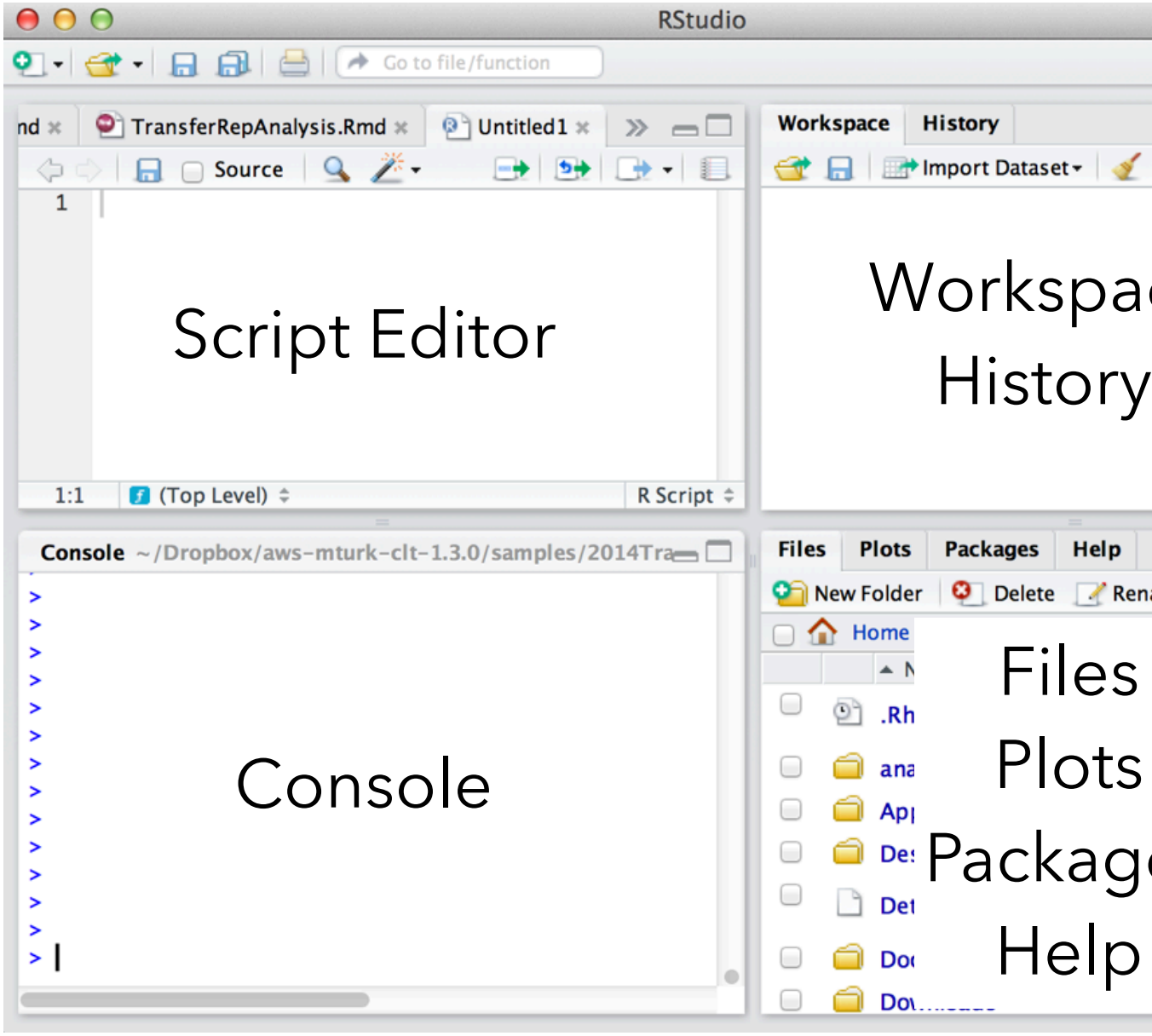


Figure 1: The R-studio workspace

0.3.2 Script Editor

The top left corner contains the script editor. This is a simple text editor for writing and saving R scripts with many lines. Several tabs can be opened at once, with each tab representing a different R script. R scripts can be saved from the editor (resulting in a .r file). Whole scripts can be run by copy and pasting them into the console and pressing enter. Alternatively, you can highlight portions of the script that you want to run (in the script editor) and pressing the button for running the current line/section: green arrow pointing right.

0.3.3 Workspace and History

The top right panel contains (at least) two tabs, one for the workspace and another for history. The workspace lists out all of the variables and functions that are currently loaded in R's memory. You can inspect each of the variables by clicking on them. This is generally only useful for variables that do not contain large amounts of information. The history tab provides a record of the recent commands executed in the console.

0.3.4 File, Plot, Packages, Help

The bottom-right window has four tabs for files, plots, packages, and help. The files tab allows browsing of the computers file directory. An important concept in R is the **current working directory**. This is file folder that R points to by default. Many functions in R will save things directly to this directory, or attempt to read files from this directory. The current working directory can be changed by navigating to the desired folder in the file menu, and then clicking on the more option to set that folder to the current working directory. This is especially important when reading in data to R. The current working directory should be set to the folder containing the data to be inputted into R. The plots tab will show recent plots and figures made in R. The packages tab lists the current R libraries loaded into memory, and provides the ability to download and enable new R packages (more about that later). The help menu is an invaluable tool. Here, you can search for individual R commands to see examples of how they are used. Sometimes the help files for individual commands are opaque and difficult to understand, so it is necessary to do a Google search to find better examples of using these commands.

0.4 How to complete the labs

Each of the labs focuses on particular data-analysis problems, from graphing data, computing descriptive statistics, to running inferential tests in R. The R

exercises usually come in three parts: a training part, a generalization part, and a writing part. The training part includes step-by-step examples of R code that solves particular problems. The generalization part gives short assignments to change parts of the provided code to solve a new problem. The writing part tasks you with answering questions about statistical concepts.

The way to complete each lab is to open a new R Markdown document in R Studio, and then document your progression through each of the parts. By doing this, you will become familiar with how R and R Studio works, and how to create documents that preserve both the code and your notes all in one place. You will upload the R Markdown document at the end of each lab to complete your work. There are a few tricks to getting started that are outlined below.

0.4.1 R projects

What is an R project? As you work inside R Studio you will be creating text documents and doing things like loading data and saving the results of your analyses. As your work grows and becomes more complex, you can often find yourself creating many different files. The R project folder is a very useful way of organizing your files all in one place so you can find them later. If you double-click an R project file, R-studio will automatically load and restore your last session. In the labs, you will be using your R project folder to:

1. save data files into this folder
2. save R Markdown files that you will use to write your R-code and lab notes
3. save the results of your analyses

We will provide an R Project which is setup for this course in particular below.

0.4.2 Installing libraries

When you install R and R Studio, you get what is called Base R. Base R contains many libraries that allow you to conduct statistical analyses. Because R is free and open-source, many other developers have created add-on libraries that extend the functionality of R. **We use some of these libraries, and you need to install them before you can do the labs**.

For example, in any of the labs, whenever you see a line code that uses the word library like this `library("libraryname")`, this line of code telling R to load up that library so it can be used. The `libraryname` would be replaced with the actual name of the library. For example, you will see code like this in the labs:

```
library(data.table)
```

This line of code is saying that the `data.table` library needs to be loaded. You can check to see if any library is already loaded by clicking on the "packages" tab in the bottom right hand panel. You will see many packages listed in alphabetical order. Packages that are currently loaded and available have a checkmark. If you scroll down and find that you **do not** have `data.table` installed, then you need to install it. To install any package follow these steps:

1. Click on the packages tab
2. Find the "install" button in the top left hand corner of the packages tab.
3. Click the install button
4. Make sure "install from:" is set to CRAN repository
5. Make sure "dependencies" is clicked on (with a checkmark)
6. Type the name of the library into the search bar.
7. As you type, you should see the names of different packages you can install pop-up in a drop-down menu. **You must be connected to the internet to install packages**
8. Once you find the package (e.g., `data.table`), click it, or just make sure the full, correctly spelled name, is in the search bar
9. Press the install button

You should see some text appear in the console while R installs the package.

10. After you have installed the package, you should now see that it is listed in the packages tab.
11. You can turn the package on by clicking it in the package tab.
12. OR, you can turn the package on by running the command `library(data.table)` in the console. To do this type `library(data.table)` into the console, and press enter.

0.4.3 Quick install

That was a really longwinded way of installing libraries, but fortunately, we can also tell R to install all of the packages we need in one go. Copy the following lines of code into the console, and press enter. From now on, any text in the grey blocks is R code which can be run by copy/pasting it in the R Studio console and pressing enter. Note you can select all of the lines at once, then copy them, then paste all of them into the console, and press enter to run them all. After each of the packages are installed, you will then be able to load them using `library()`.

```
install.packages("ggplot2")
install.packages("dplyr")
install.packages("data.table")
install.packages("summarytools")
install.packages("gapminder")
install.packages("ggpubr")
install.packages("knitr")
install.packages("rmarkdown")
```

0.4.4 R Markdown

Once you have the necessary packages installed you can begin creating R Markdown documents for each lab. We admit that at the beginning, R Markdown documents might seem a little bit confusing, but you will find they are extremely useful and flexible. Basically, what R Markdown allows you to do is combine two kinds of writing, 1) writing R code to conduct analyses, and 2) writing normal text, with headers, sub-headers, and paragraphs. You can think of this like a lab journal, that contains both your writing about what you are doing (e.g., notes to self), and the code that you use for analysis. Additionally, when your code does something like make a graph, or run a statistical test, you can ask R Markdown to print the results.

The R Markdown website has an excellent tutorial that is well worth checking out in your own time: https://rmarkdown.rstudio.com/lesson-1.html

0.4.5 R Markdown lab templates

We have created a set of template documents for each lab that can be downloaded here: download lab templates.

When you unzip the file you should find the following:

1. A new folder titled "RMarkdownLabs"
2. Inside the folder you will see the "RMarkdownLabs" file
3. A data folder containing data files for the labs
4. A "LabTemplates" folder containing the R Markdown for each lab

Each template .Rmd file contains three sections for the R part of the lab. You will write your notes and answers to the question in the .Rmd file. At the end of each lab you will "knit" the file (knitting is a term for making your documents look prettier) and upload the result to Canvas.

Chapter 1

Week 1: Introduction

This first lab consists of two parts. In the first part we will work on several exercises related to the readings about research methods. In the second part you will learn how to graph data using R. This lab assumes you have read the required readings of Week 1 and completed the Getting started guide. You will be completing each lab by writing your notes and code in an R Markdown document. You should have setup your files as described in this part of the Getting started section. At the end of this lab you will upload the document with your answers to Canvas.

Let's get started!

Follow these steps:

1. Copy the template file for lab 1, "Lab01_Introduction_StudentNumber.Rmd", and place it into the "RMarkdownLabs" folder (i.e. copy it out of the template folder, and into the RMarkdownLabs folder).
2. Rename the file to add your own student number, eg., "Lab01_Introduction_311476"
3. Double-click the "RMarkdownLabs.Rproj" file
4. R Studio will now load up.
5. If you click the files tab, you will see all of the files and folders inside the "RMarkdownLabs" folder
6. Click the lab file and it will now load into the editor window.
7. You can keep your notes, copy/paste R code, and answer the questions of this lab in the document.

When we made this course, we assumed that most students would be unfamiliar with R and R Studio, and might even be frightened of it. Don't worry. It's going

to be way easier than you think. We know that it will seem challenging at first. But, we think that with lots of working examples, you will get the hang of it, and by the end of the course you will be able to do things you might never have dreamed you can do. It's really a fantastic skill to learn, even if you aren't planning on going on to do research. If anything during this lab is unclear, please do not hesitate to ask your tutor.

1.1 Part one: Research methods

Discuss the questions about research methods below in groups of 3 and register the answers in the .Rmd file for this lab.

Question 1

In the required readings of this week we called to process of clarifying abstract concepts and translating them into specific, observable measures **operationalization**. This involves both a **nominal** and an **operational** definition. Describe in your own words what these terms mean.

Question 2

Two different definitions of **emotional well-being** are provided by the Mental Health Foundation. For each of the following definitions, decide whether it constitutes a nominal or an operational definition:

a. "A positive sense of well-being which enables an individual to be able to function in society and meet the demands of everyday life."
b. "People in good mental health have the ability to recover effectively from illness, change or misfortune."

Question 3

Two different definitions of **financial literacy** can be found in literature. For each of the following definitions, decide whether it constitutes a nominal or an operational definition:

a. "The ability to read, analyze, manage and communicate about the personal financial conditions that affect material well-being."
b. "The ability to manage effectively personal savings, credits and borrowed money as well as personal investments."

Question 4

Suppose you want to study financial literacy, given the numerous benefits it brings to society, and given the documented lack of financial education. Would

you use the following operational definition of financial literacy: "The ability to correctly predict short term fluctuations in the stock market?"

Question 5

The graph below is a visual representation of the concepts of validity and reliability.

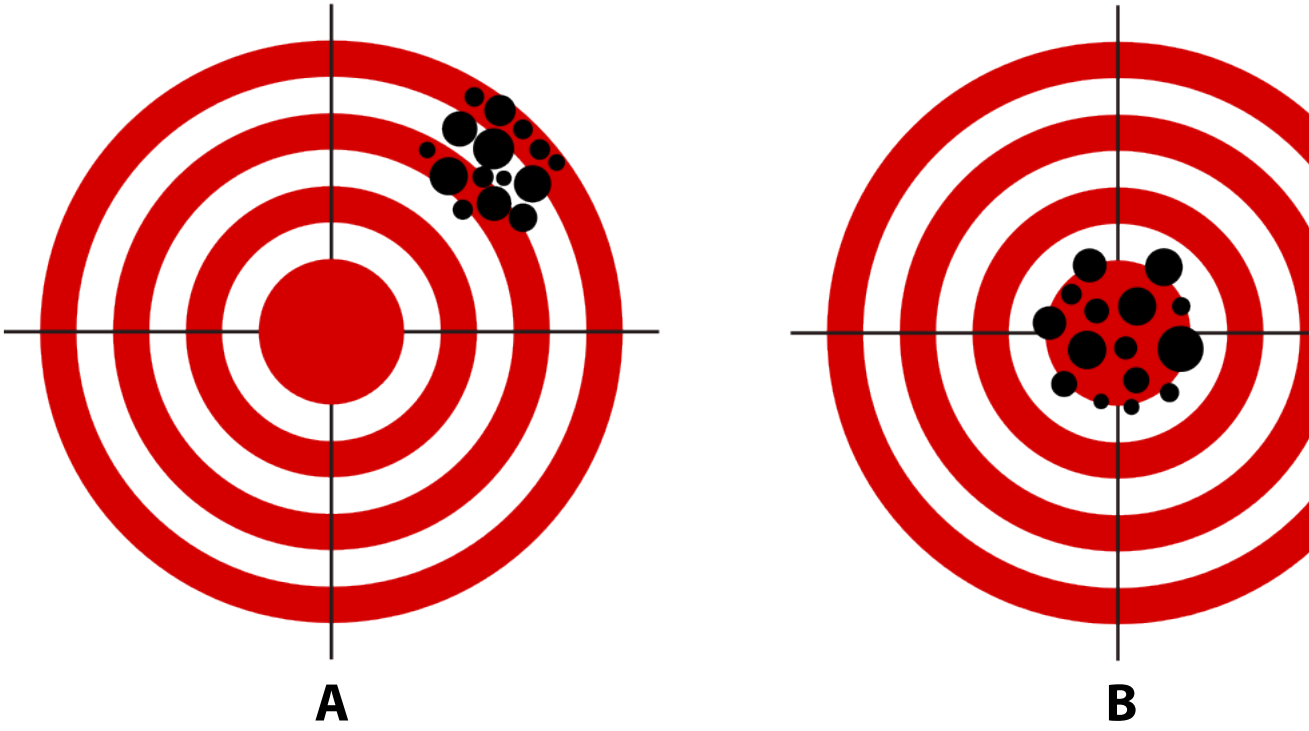


Figure 1.1: A visual representation of the concepts validity and reliability. Imagine the theoretical construct you want to measure is the bullseye of the dartboard and the dots represent an attempt at measurement. Illustration adapted from an illustration by Nevit Dilmen, Wikimedia Commons.

For each one of the three statements below, indicate whether it corresponds to dart board A, B, C or none.

- The measure of our concept is valid, but not reliable.
- The measure of our concept is reliable, but not valid.
- The measure of our concept is neither valid, not reliable.
- The measure of our concept is both valid and reliable.

Question 6

The National Health Care Institute of the Netherlands partners with local schools to provide a weekly physical exercise program for children ages 6-14. The sessions are designed to last throughout the whole academic year, and they will take place in afternoon hours. They also consist of both a theoretical and a practical part. In the theoretical part, volunteers strive to increase children's exercise habits by teaching them about the benefits of regular exercise, whereas in the practical part, they organize various age-appropriate sports activities for children to participate in. Changes in exercise habits are measured via a questionnaire at the end of the program. However, the program manager is concerned that the questionnaire is not producing high-quality observations, particularly for questions that ask children about their exercise habits before participating in the program. Assuming the problem is with measurement and not with the program design…

- What is the most likely measurement problem? Reliability or validity?
- What type of error is most likely producing this problem? Constant error, random error and/or correlated error?
- How might the program address this measurement problem?

Question 7

The Dutch Environmental Assessment Agency aims to identify sections of Dutch rivers for stream bank restoration. The goal of this work is to create stream bank conditions that can lead to eventual water quality improvements. Crews of national service volunteers implement remediation in accordance with the waterway management plan, including removal of trash and debris from stream banks, removal of invasive plants, reintroduction of native plants, and erosion abatement. Land managers from the Ministry of Infrastructure and Water Management inspect project sites within two weeks of project completion. The assessment instrument used by land managers contains checkbox items to indicate whether various remediation actions were taken but does not provide a way to assess the quality of these remediation actions with respect to environmental standards. This problem should be of high concern to the land managers, given the fact that high quality environmental standards are hard to meet, even when all the appropriate actions have been taken. Assuming the problem is with measurement and not with the program design…

- What is the most likely measurement problem? Reliability or validity?
- What type of error is most likely producing this problem? Constant error, random error and/or correlated error?
- How might the program address this measurement problem?

1.2 Part two: Graphing data

> The commonality between science and art is in trying to see profoundly - to develop strategies of seeing and showing. —Edward Tufte

As we have found out from the textbook and lecture, when we measure things, we get lots of numbers. Too many numbers. Sometimes so many your head explodes just thinking about them. One of **the most helpful things** you can do to begin to make sense of these numbers, is to look at them in graphical form. Unfortunately, for sight-impaired individuals, graphical summary of data is much more well-developed than other forms of summarizing data for our human senses. Some researchers are developing auditory versions of visual graphs, a process called **sonification**, but we aren't prepared to demonstrate that here. Instead, we will make charts, and plots, and things to look at, rather than the numbers themselves, mainly because these are tools that are easiest to get our hands on, they are the most developed, and they work really well for visual summary. If time permits, at some point I would like to come back here and do the same things with sonification. I think that would be really, really cool!

During this part of the lab we'll do these things in R:

1. Load some data in R
2. Talk a little bit about how the data is structured
3. Make graphs of the data so we can look at it and make sense of it

1.2.1 Important info

1. Data for NYC film permits was obtained from the NYC open data website. The raw .csv file can be found here: Film_Permits.csv

2. Gapminder data from the gapminder project (copied from the R gapminder library) can be downloaded in raw .csv format here: gapminder.csv

1.2.2 Generate some data

In order to graph data, we need to have some data first…Actually, with R, that's not quite true. Run the following bit of code and see what happens. Note that from now on, any text in the grey blocks is R code which can be run by copy/pasting it in the R Studio console and pressing enter. Additionally, you can copy and paste the code below to your lab file so as to keep notes of your work.

```
hist(rnorm(100, mean=50, sd=25))
```

You just made R sample 100 numbers, and then plot the results in a histogram. Pretty neat. We'll be doing some of this later in the course, where get R to make fake data for us, and then we learn to think about how data behaves under different kinds of assumptions.

For now, let's do something that might be a little bit more interesting… let's figure out what movies are going to be filmed in NYC. It turns out that NYC makes a lot of data about a lot things open and free for anyone to download and look at. This is the NYC Open Data website: https://opendata.cityofnewyork.us. I searched through the data, and found a data file that lists the locations of film permits for shooting movies all throughout the Burroughs. There are multiple ways to load this data into R.

1. If you have downloaded the RMarkdownLabs.zip file, then you already have the data file in the data folder. Assuming you are working in your main directory (your .rmd file is saved in the main folder that contains both the data and template folders), then use the following commands to load the data.

```
library(data.table)
nyc_films <-fread("data/Film_Permits.csv")
```

2. If the above method doesn't work, you can try loading the data from the course website using:

```
library(data.table)
nyc_films <- fread("https://raw.githubusercontent.com/thomashulst/quantrma_lab/master/
```

If you are having issues getting the data loaded, talk to your tutor.

1.2.3 Look at the data

You will be downloading and analyzing all kinds of data files this quad. We will follow the very same steps every time. The steps are to load the data, then look at it. You want to see what you've got.

In R Studio, you will now see a variable called `nyc_films` in the top right-hand corner of the screen, in the environment tab. If you click this thing, it will show you the contents of the data in a new window. The data is stored in something we call a `data frame`. It's R lingo, for the thing that contains the data. Notice it is shaped like a rectangle, with rows going across, and columns going up and down. It looks kind of like an excel spreadsheet if you are familiar with Excel.

It's useful to know you can look at the data frame this way if you need to. But, this data frame is really big, it has 50,728 rows of data. That's a lot too much to look at.

1.2.3.1 summarytools

The summarytools package gives a quick way to summarize all of the data in a data frame. Here's how. When you run this code you will see the summary in the viewer on the bottom right hand side. There's a little browser button (arrow on top of little window) that you can click to expand and see the whole thing in a browser.

```
library(summarytools)
view(dfSummary(nyc_films))
```

That is super helpful, but it's still a lot to look at. Because there is so much data here, it's pretty much mind-boggling to start thinking about what to do with it.

1.2.4 Make plots to answer questions

Let's walk through a couple questions we might have about this data. We can see that there were 50,728 film permits made. We can also see that there are different columns telling us information about each of the film permits. For example, the `Borough` column lists the Borough for each request, whether it was made for: Manhattan, Brooklyn, Bronx, Queen's, or Staten Island. Now we can ask our first question, and learn how to do some plotting in R.

1.2.4.1 Where are the most film permits being requested?

Do you have any guesses? Is it Manhattan, or Brooklyn, of the Bronx? Or Queen's or Staten Island? We can find out by plotting the data using a bar plot. We just need to count how many film permits are made in each borough, and then make different bars represent the the counts.

First, we do the counting in R. Run the following code:

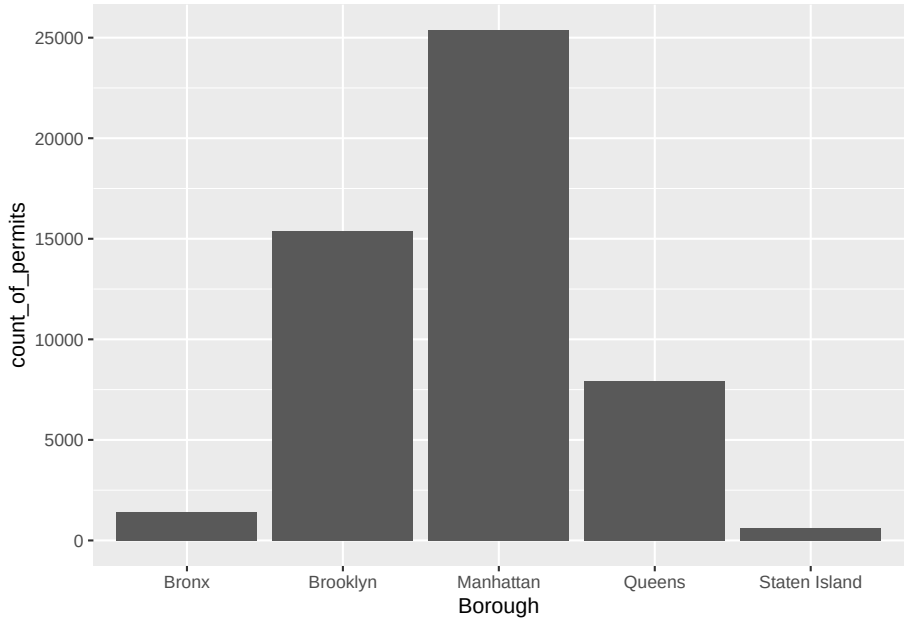
```
library(dplyr)

counts <- nyc_films %>%
          group_by(Borough) %>%
          summarize(count_of_permits = length(Borough))
```

The above grouped the data by each of the five Borough's, and then counted the number of times each Borough occurred (using the `length` function). The result is a new variable called `count`. I chose to name this variable `count`. You can see that it is now displayed in the top-right hand corned in the environment tab. If you gave `count` a different name, like `muppets`, then it would be named what you called it.

If you click on the `counts` variable, you will see the five boroughs listed, along with the counts for how many film permits were requested in each Borough. These are the numbers that we want to plot in a graph.

We do the plot using a fantastic package called `ggplot2`. It is very powerful once you get the hand of it, and when you do, you will be able to make all sorts of interesting graphs. Here's the code to make the plot:

```
library(ggplot2)

ggplot(counts, aes(x = Borough, y = count_of_permits )) +
  geom_bar(stat="identity")
```

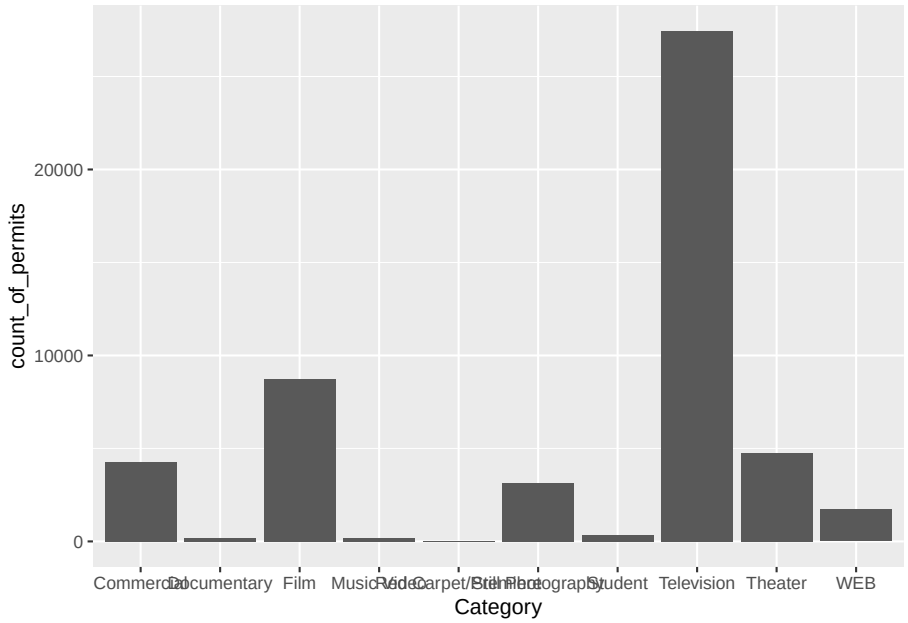


There it is, we're done here! We can easily look at this graph, and answer our question. Most of the film permits were requested in Manhattan, followed by Brooklyn, then Queen's, the Bronx, and finally Staten Island.

1.2.4.2 What kind of "films" are being made, what is the category?

We think you might be skeptical of what you are doing here, copying and pasting things. Soon you'll see just how fast you can do things by copying and pasting, and make a few little changes. Let's quickly ask another question about what kinds of films are being made. The column `Category`, gives us some information about that. Let's just copy paste the code we already made, and see what kinds of categories the films fall into. See if you can tell what I changed in the code to make this work, I'll do it all at once:

```
counts <- nyc_films %>%
          group_by(Category) %>%
          summarize(count_of_permits = length(Category))

ggplot(counts, aes(x = Category, y = count_of_permits )) +
  geom_bar(stat="identity")
```



OK, so this figure might look a bit weird because the labels on the bottom are running into each other. We'll fix that in a bit. First, let's notice the changes.

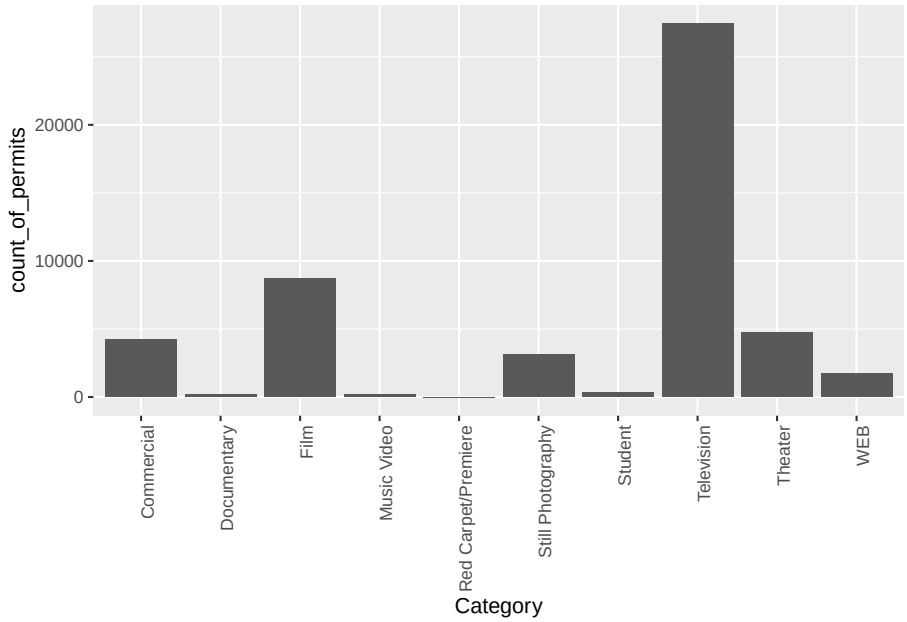
1. I changed `Borough` to `Category`. That was the main thing
2. I left out a bunch of things from before. None of the `library()` commands are used again, and I didn't re-run the very early code to get the data. R already has those things in it's memory, so we don't need to do that

first. If you ever clear the memory of R, then you will need to reload those things.

Fine, so how do we fix the graph? Good question. You probably have no idea how to do this, but googling your questions is a great way to learn R. It's what everybody does.

When you Google this question, you will likely find a lot of ways to fix your graph. The trick I used is to add the last line in the R code below. I just copy-pasted it from the solution I found on stack overflow (you will become friend's with stack overflow, there are many solutions there to all of your questions).

```
counts <- nyc_films %>%
          group_by(Category) %>%
          summarize(count_of_permits = length(Category))

ggplot(counts, aes(x = Category, y = count_of_permits )) +
  geom_bar(stat="identity")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



1.2.5 ggplot2 basics

Before we go further, I want to point out some basic properties of ggplot2, just to give you a sense of how it is working. This will make more sense in a few

weeks, so come back here to remind yourself. We'll do just a bit a basics, and then move on to making more graphs, by copying and pasting.

The ggplot function makes use of so-called layers. Layers you say? What are these layers? Well, it draws things from the bottom up. It lays down one layer of graphics, then you can keep adding on top, drawing more things. So the idea is something like: Layer 1 + Layer 2 + Layer 3, and so on. If you want Layer 3 to be Layer 2, then you just switch them in the code.

Here is a way of thinking about ggplot code

```
ggplot(name_of_data, aes(x = name_of_x_variable, y = name_of_y_variable)) +
    geom_layer()+
    geom_layer()+
    geom_layer()
```

What I want you to focus on in the above description is the + signs. What we are doing with the plus signs is adding layers to plot. The layers get added in the order that they are written. If you look back to our previous code, you will see we add a `geom_bar` layer, then we added another layer to change the rotation of the words on the x-axis. This is how it works.

BUT WAIT? How am I supposed to know and remember what to add? This is nuts! We know. You're not supposed to know just yet, how could you? We'll give you lots of examples where you can copy and paste, and they will work. That's how you'll learn for now. If you really want to read the help manual you can do that too. It's on the ggplot2 website. This will become useful after you already know what you are doing, but before that, it will probably just seem very confusing. However, it is pretty neat to look and see all of the different things you can do, it's very powerful.

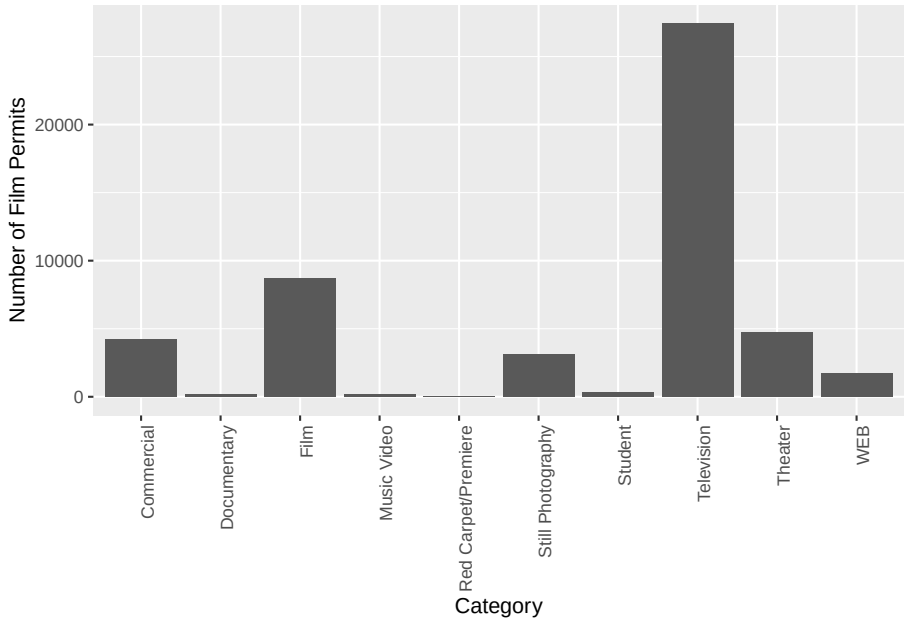
For now, let's the get the hang of adding things to the graph that let us change some stuff we might want to change. For example, how do you add a title? Or change the labels on the axes? Or add different colors, or change the font-size, or change the background? You can change all of these things by adding different lines to the existing code.

1.2.5.1 ylab() changes y label

The last graph had `count_of_permits` as the label on the y-axis. That doesn't look right. ggplot2 automatically took the label from the column, and made it be the name on the y-axis. We can change that by adding `ylab("what we want")`. We do this by adding a + to the last line, then adding `ylab()`

```
ggplot(counts, aes(x = Category, y = count_of_permits )) +
  geom_bar(stat="identity") +
```

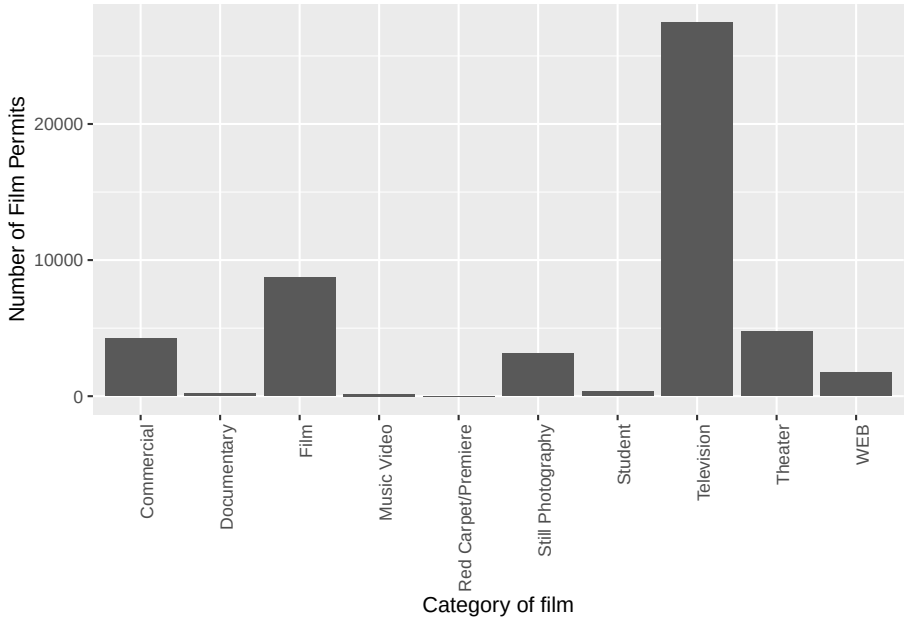
```
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits")
```



1.2.5.2 xlab() changes x label

Let's slightly modify the x label too:

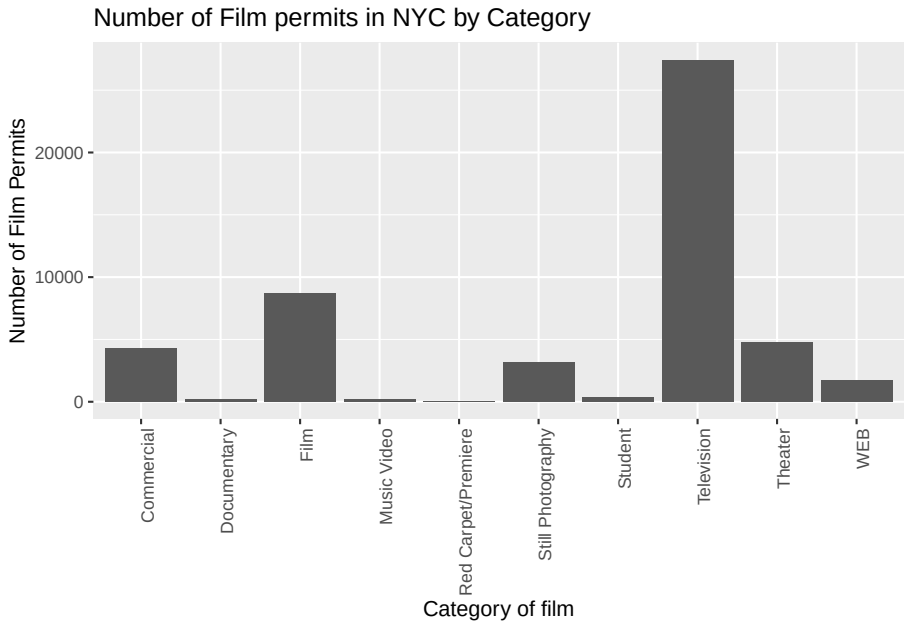
```
ggplot(counts, aes(x = Category, y = count_of_permits )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film")
```



1.2.5.3 ggtitle() adds title

Let's give our graph a title

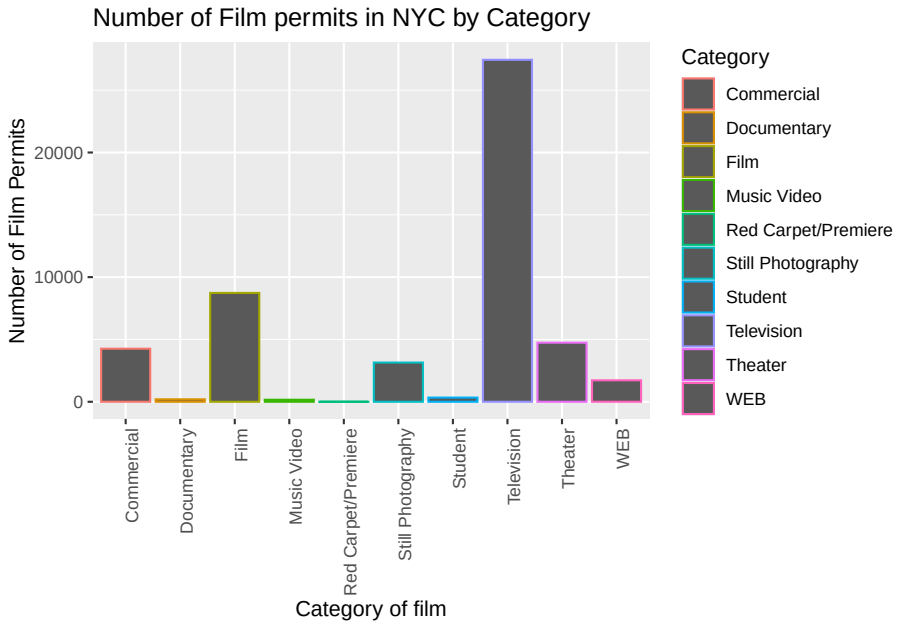
```
ggplot(counts, aes(x = Category, y = count_of_permits )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category")
```



1.2.5.4 color adds color

Let's make the bars different colors. To do this, we add new code to the inside of the `aes()` part:

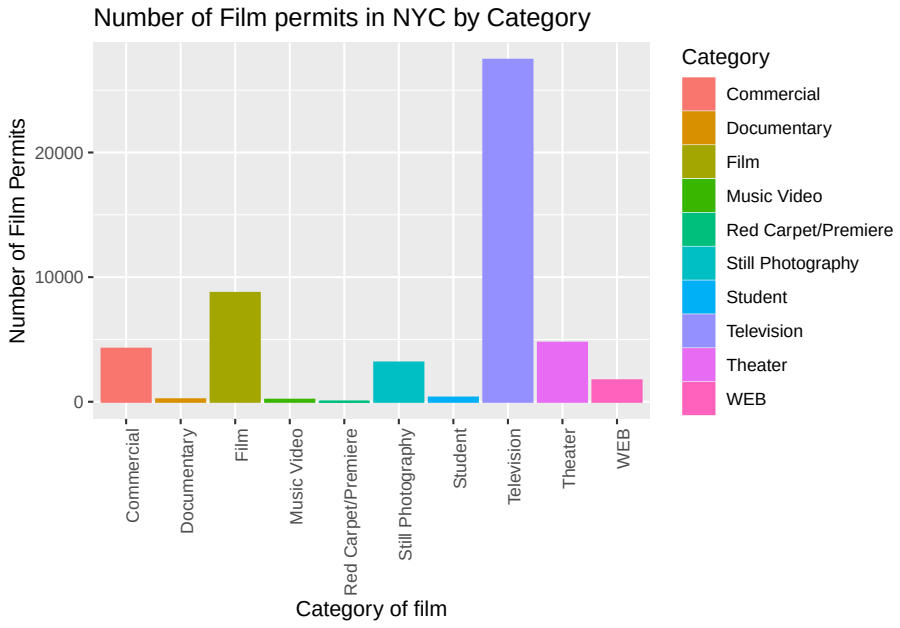
```
ggplot(counts, aes(x = Category, y = count_of_permits, color=Category )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category")
```



1.2.5.5 fill fills in color

Let's make the bars different colors. To do this, we add new code to the inside of the `aes()` part…Notice I've started using new lines to make the code more readable.

```
ggplot(counts, aes(x = Category, y = count_of_permits,
                   color=Category,
                   fill= Category )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category")
```

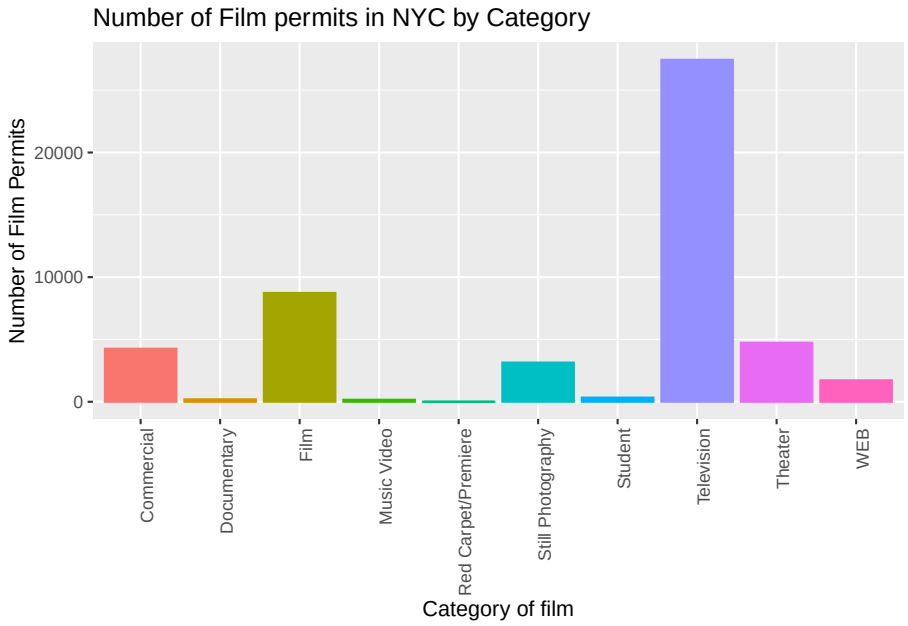


1.2.5.6 get rid of the legend

Sometimes you just don't want the legend on the side, to remove it add

```
theme(legend.position="none")
```

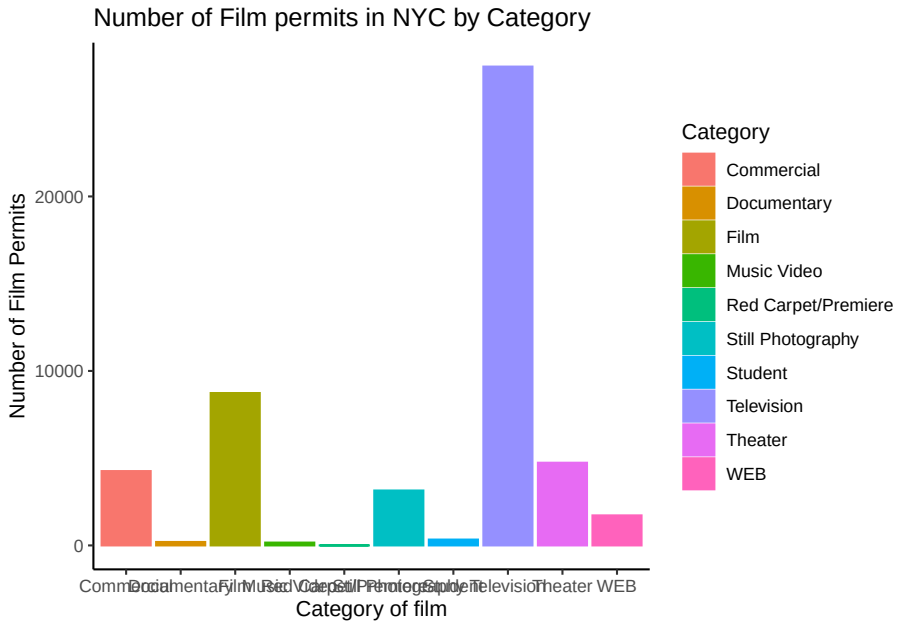
```
ggplot(counts, aes(x = Category, y = count_of_permits,
                   color=Category,
                   fill= Category )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category") +
  theme(legend.position="none")
```



1.2.5.7 theme_classic() makes white background

The rest is often just visual preference. For example, the graph above has this grey grid behind the bars. For a clean classic no nonsense look, use `theme_classic()` to take away the grid.

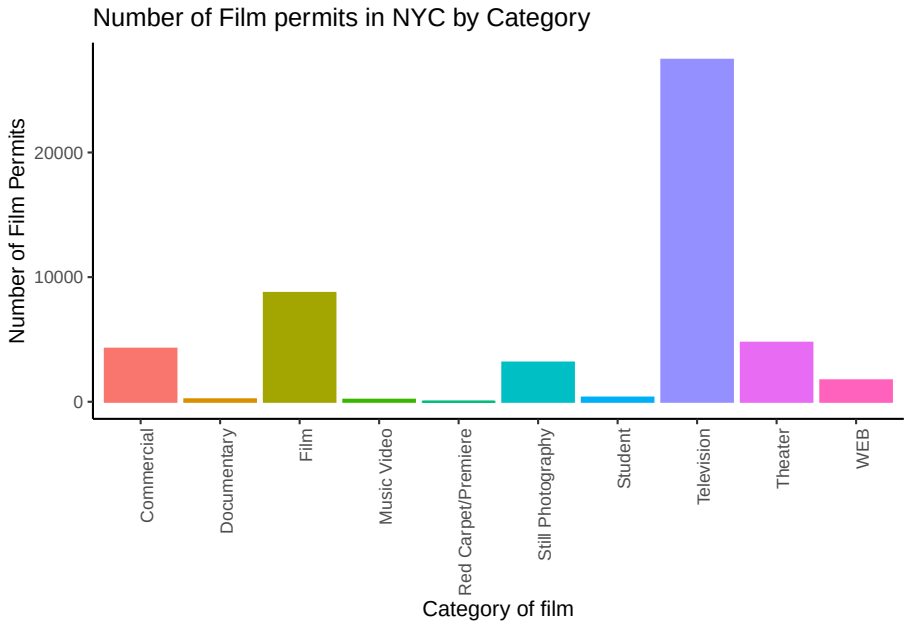
```
ggplot(counts, aes(x = Category, y = count_of_permits,
                   color=Category,
                   fill= Category )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category") +
  theme(legend.position="none") +
  theme_classic()
```



1.2.5.8 Sometimes layer order matters

Interesting, `theme_classic()` is misbehaving a little bit and incorrectly renders the axis labels and reintroduces the legend. It looks like we have some of our layers out of order, let's re-order. I just moved `theme_classic()` to just underneath the `geom_bar()` line. Now everything get's drawn properly.

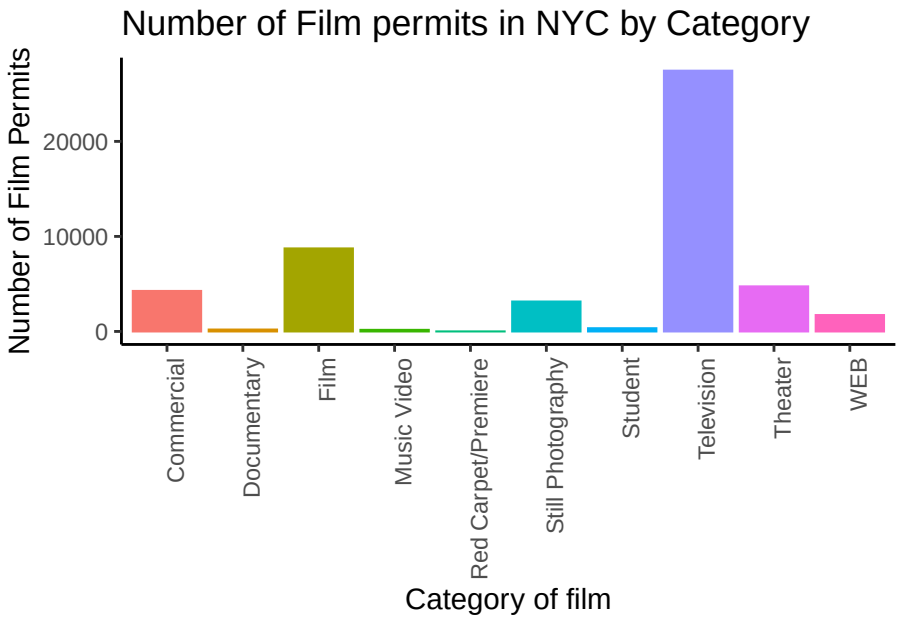
```
ggplot(counts, aes(x = Category, y = count_of_permits,
                   color=Category,
                   fill= Category )) +
  geom_bar(stat="identity") +
  theme_classic() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category") +
  theme(legend.position="none")
```



1.2.5.9 Font-size

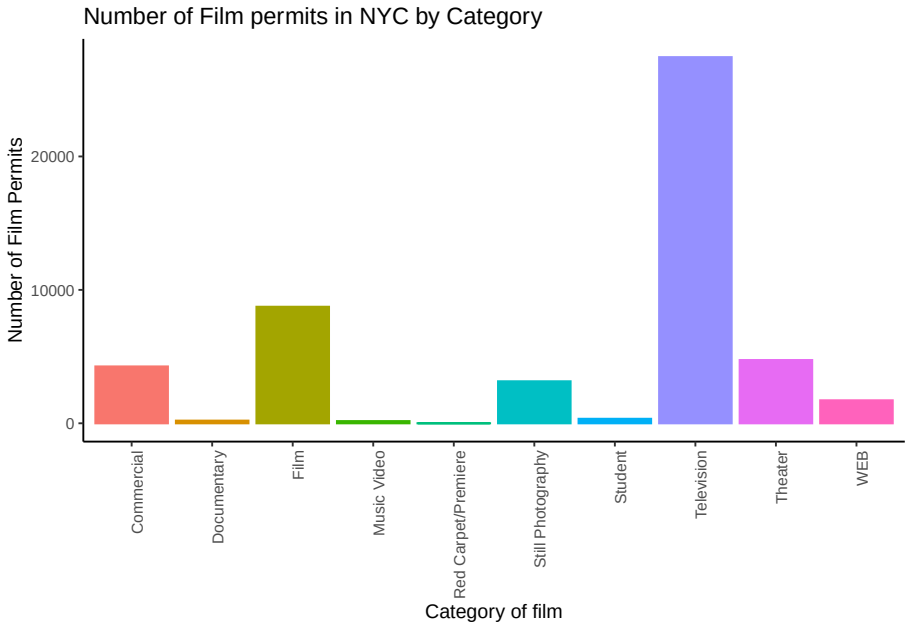
Changing font-size is often something you want to do. ggplot2 can do this in different ways. I suggest using the `base_size` option inside `theme_classic()`. You set one number for the largest font size in the graph, and everything else gets scaled to fit with that that first number. It's really convenient. Look for the inside of `theme_classic()`

```
ggplot(counts, aes(x = Category, y = count_of_permits,
                   color=Category,
                   fill= Category )) +
  geom_bar(stat="identity") +
  theme_classic(base_size = 15) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category") +
  theme(legend.position="none")
```



or make things small... just to see what happens

```
ggplot(counts, aes(x = Category, y = count_of_permits,
                   color=Category,
                   fill= Category )) +
  geom_bar(stat="identity") +
  theme_classic(base_size = 10) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category") +
  theme(legend.position="none")
```



1.2.5.10 ggplot2 summary

That's enough of the ggplot2 basics for now. You will discover that many things are possible with ggplot2. It is amazing. We are going to get back to answering some questions about the data with graphs. But, now that we have built the code to make the graphs, all we need to do is copy-paste, and make a few small changes, and boom, we have our graph.

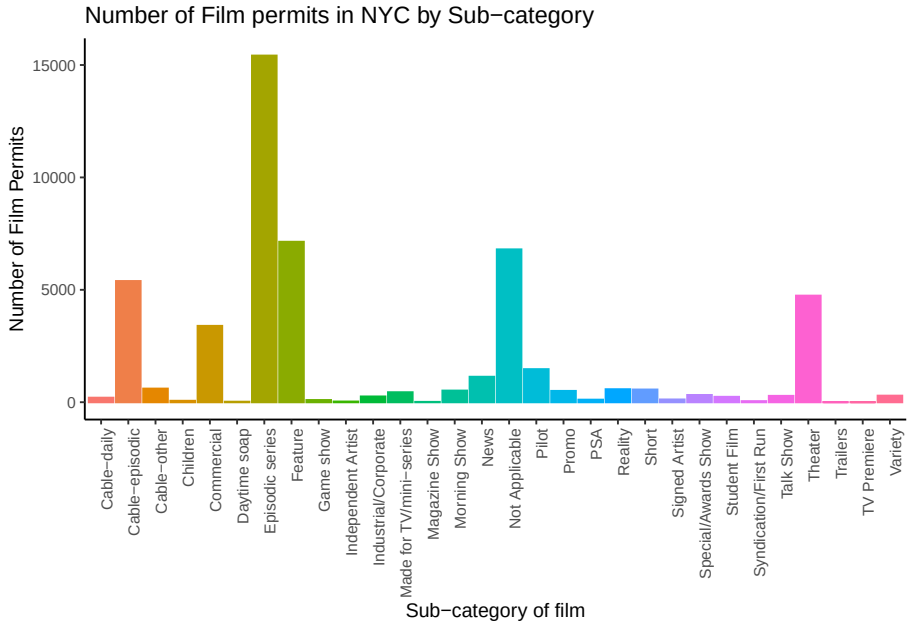
1.2.6 More questions about NYC films

1.2.6.1 What are the sub-categories of films?

Notice the `nyc_films` data frame also has a column for `SubCategoryName`. Let's see what's going on there with a quick plot.

```
# get the counts (this is a comment)
# comments are really useful: R will ignore them, but they can explain to a reader
# what is going on in the code

counts <- nyc_films %>%
          group_by(SubCategoryName) %>%
          summarize(count_of_permits = length(SubCategoryName))
```

```
# make the plot

ggplot(counts, aes(x = SubCategoryName, y = count_of_permits,
                   color=SubCategoryName,
                   fill= SubCategoryName )) +
  geom_bar(stat="identity") +
  theme_classic(base_size = 10) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Sub-category of film") +
  ggtitle("Number of Film permits in NYC by Sub-category") +
  theme(legend.position="none")
```



I guess "episodic series" are the most common. Using a graph like this gave us our answer super fast.

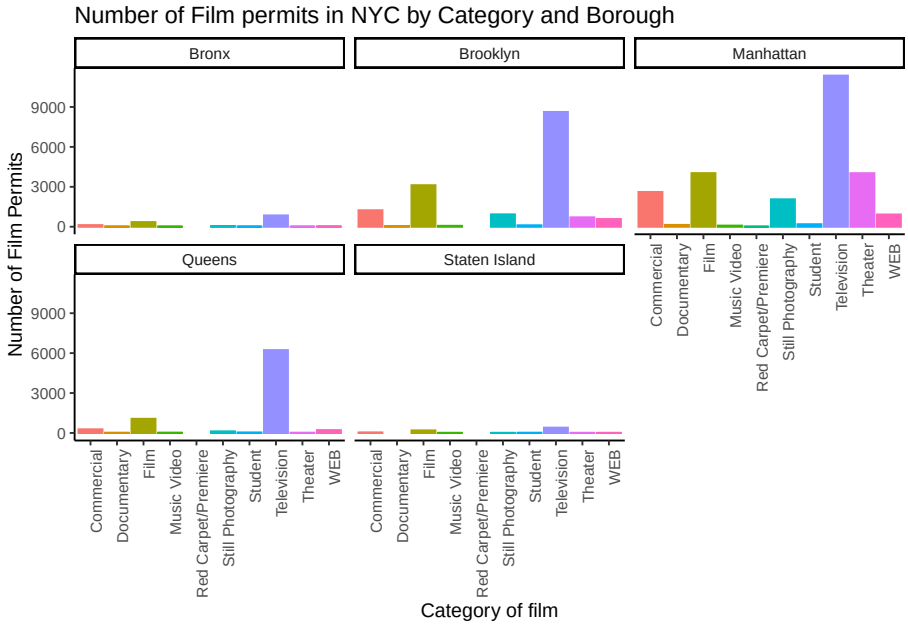
1.2.6.2 Categories by different Boroughs

Let's see one more really useful thing about ggplot2. It's called `facet_wrap()`. It's an ugly word, but you will see that it is very cool, and you can do next-level-super-hero graph styles with `facet_wrap` that other people can't do very easily.

Here's our question. We know that some films are made in different Boroughs, and that same films are made in different categories, but do different Boroughs

have different patterns for the kinds of categories of films they request permits for? Are their more TV shows in Brooklyn? How do we find out? Watch, just like this:

```
# get the counts

counts <- nyc_films %>%
          group_by(Borough,Category) %>%
          summarize(count_of_permits = length(Category))

# make the plot

ggplot(counts, aes(x = Category, y = count_of_permits,
                   color=Category,
                   fill= Category )) +
  geom_bar(stat="identity") +
  theme_classic(base_size = 10) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category and Borough") +
  theme(legend.position="none") +
  facet_wrap(~Borough, ncol=3)
```

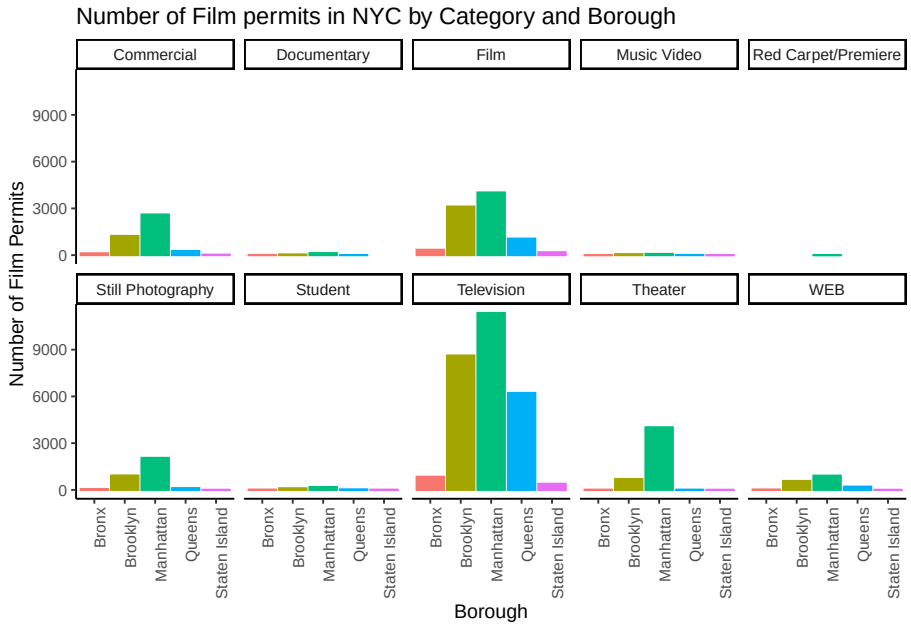


We did two important things. First we added `Borough` and `Category` into

the `group_by()` function. This automatically gives separate counts for each category of film, for each Borough. Then we added `facet_wrap(~Borough, ncol=3)` to the end of the plot, and it automatically drew us 5 different bar graphs, one for each Borough! That was fast. Imagine doing that by hand.

The nice thing about this is we can switch things around if we want. For example, we could do it this way by switching the `Category` with `Borough`, and facet-wrapping by Category instead of Borough like we did above. Do what works for you.

```
ggplot(counts, aes(x = Borough, y = count_of_permits,
                   color=Borough,
                   fill= Borough )) +
  geom_bar(stat="identity") +
  theme_classic(base_size = 10) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Borough") +
  ggtitle("Number of Film permits in NYC by Category and Borough") +
  theme(legend.position="none") +
  facet_wrap(~Category, ncol=5)
```



1.2.7 Gapminder Data

Gapminder is an organization that collects some really interesting worldwide

data. They also make cool visualization tools for looking at the data. There are many neat examples, and they have visualization tools built right into their website that you can play around with https://www.gapminder.org/tools/. That's fun to check out.

There is also an R package called `gapminder`. When you install this package, it loads in some of the data from gapminder, so we can play with it in R.

If you don't have the gapminder package installed, you can install it by running this code

```
install.packages("gapminder")
```

Once the package is installed, you need to load the new library, like this. Then, you can put the `gapminder` data into a data frame, like we do here: `gapminder_df`.

```
library(gapminder)
gapminder_df<-gapminder
```

1.2.7.1 Look at the data frame

You can look at the data frame to see what is in it, and you can use `summarytools` again to view a summary of the data.

```
view(dfSummary(gapminder_df))
```

There are 1704 rows of data, and we see some columns for country, continent, year, life expectancy, population, and GDP per capita.

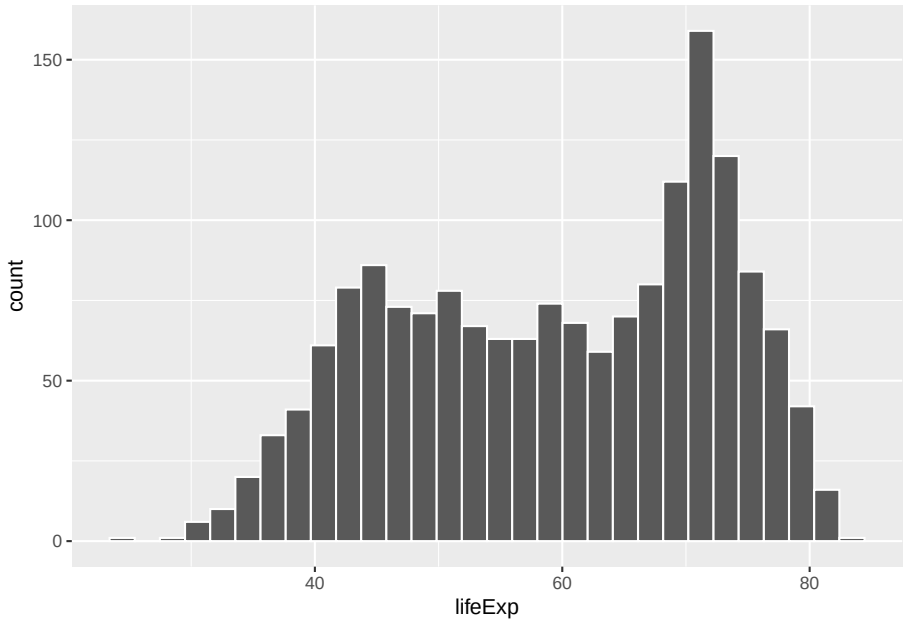
1.2.8 Asking questions about the gapminder data

We will show you how to graph some the data to answer a few different kinds of questions. Then you will form your own questions, and see if you can answer them with ggplot2 yourself. All you will need to do is copy and paste the following examples, and change them up a little bit

1.2.8.1 Life expectancy histogram

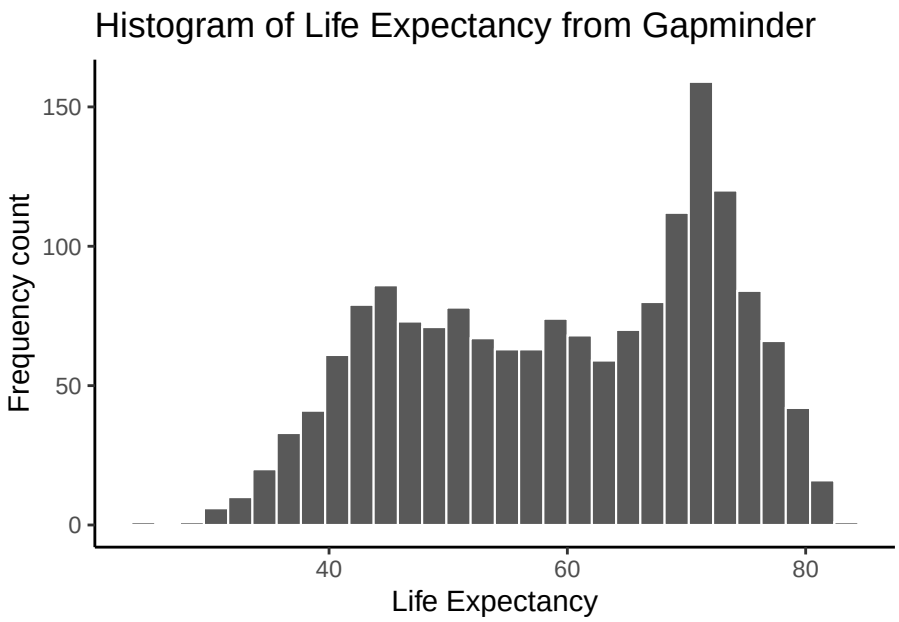
How long are people living all around the world according to this data set? There are many ways we could plot the data to find out. The first way is a histogram. We have many numbers for life expectancy in the column `lifeExp`. This is a big sample, full of numbers for 142 countries across many years. It's easy to make a histogram in ggplot to view the distribution:

```
ggplot(gapminder_df, aes(x=lifeExp))+
  geom_histogram(color="white")
```



See, that was easy. Next, is a code block that adds more layers and settings if you wanted to modify parts of the graph:

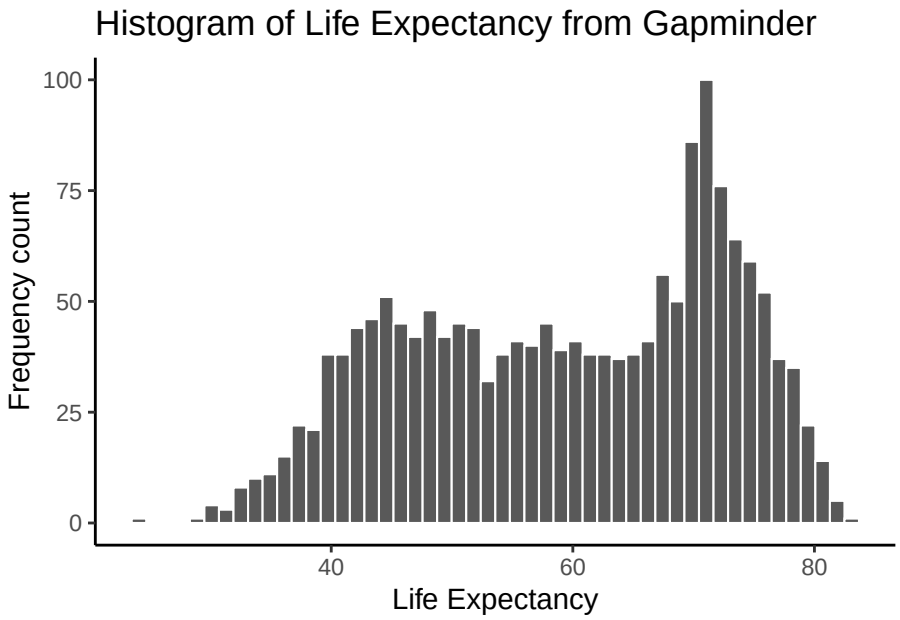
```
ggplot(gapminder_df, aes(x = lifeExp)) +
  geom_histogram(color="white")+
  theme_classic(base_size = 15) +
  ylab("Frequency count") +
  xlab("Life Expectancy") +
  ggtitle("Histogram of Life Expectancy from Gapminder")
```



The histogram shows a wide range of life expectancies, from below 40 to just over 80. Histograms are useful, they can show you what kinds of values happen more often than others.

One final thing about histograms in ggplot. You may want to change the bin size. That controls how wide or narrow, or the number of bars (how they split across the range), in the histogram. You need to set the `bins=` option in `geom_histogram()`.

```
ggplot(gapminder_df, aes(x = lifeExp)) +
  geom_histogram(color="white", bins=50)+
  theme_classic(base_size = 15) +
  ylab("Frequency count") +
  xlab("Life Expectancy") +
  ggtitle("Histogram of Life Expectancy from Gapminder")
```



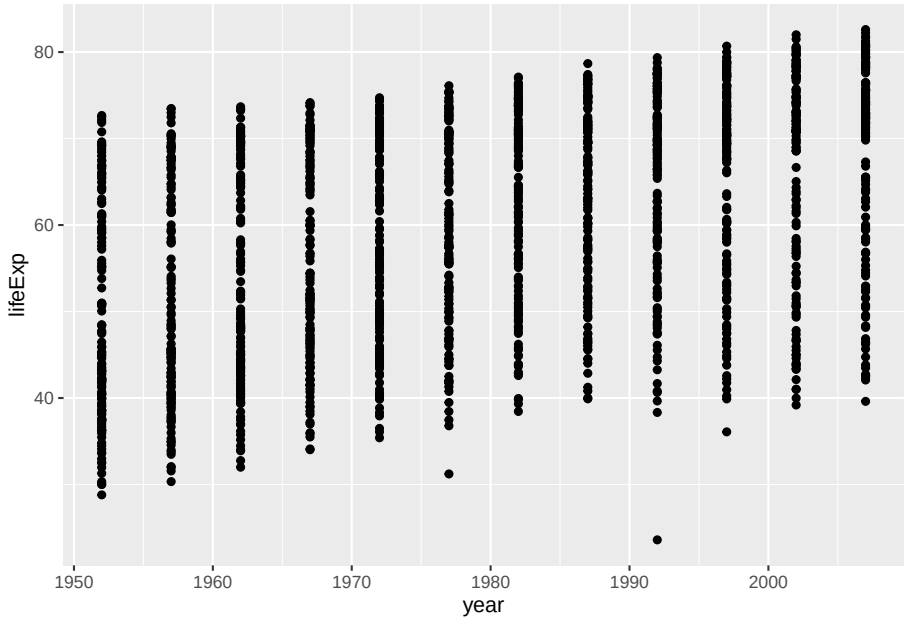
See, same basic patter, but now breaking up the range into 50 little equal sized bins, rather than 30, which is the default. You get to choose what you want to do.

1.2.8.2 Life expectancy scatterplot

We can see we have data for life expectancy and different years. So, does worldwide life expectancy change across the years in the data set? As we go into the future, are people living longer?

Let's look at this using a scatter plot. We can set the x-axis to be year, and the y-axis to be life expectancy. Then we can use `geom_point()` to display a whole bunch of dots, and then look at them. Here's the simple code:

```
ggplot(gapminder_df, aes(y= lifeExp, x= year))+
  geom_point()
```

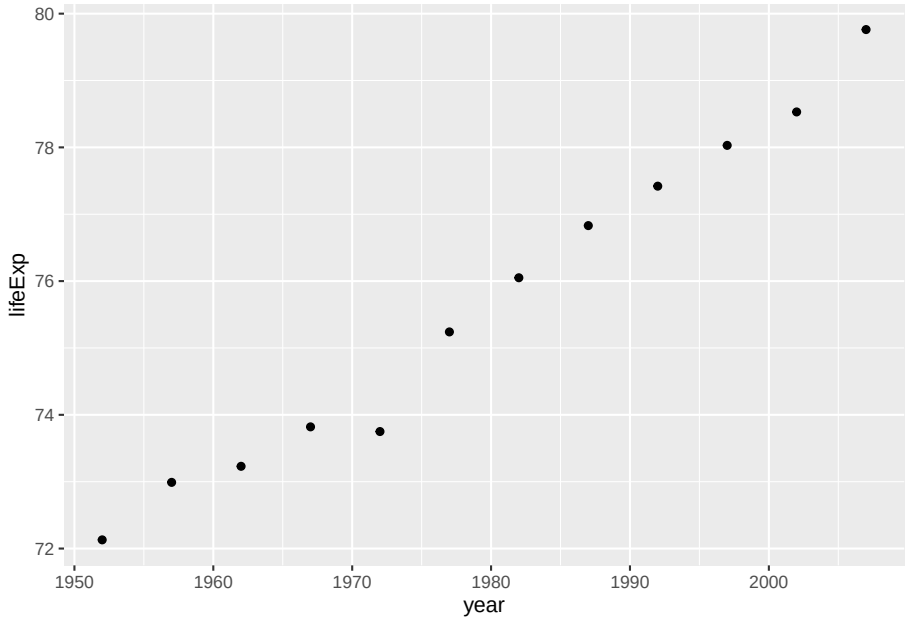


Whoa, that's a lot of dots! Remember that each country is measured each year. So, the bands of dots you see, show the life expectancies for the whole range of countries within each year of the database. There is a big spread inside each year. But, on the whole it looks like groups of dots slowly go up over years.

1.2.8.3 One country, life expectancy by year

I'm (Thomas) from The Netherlands, so maybe I want to know if life expectancy for Dutch people is going up over the years. To find out the answer for one country, we first need to split the full data set, into another smaller data set that only contains data for The Netherlands. In other words, we want only the rows where the word "Netherlands" is found in the `country` column. We will use the `filter` function from `dplyr` for this:

```
# filter rows to contain Canada

smaller_df <- gapminder_df %>%
                filter(country == "Netherlands")

# plot the new data contained in smaller_df

ggplot(smaller_df, aes(y= lifeExp, x= year))+
  geom_point()
```

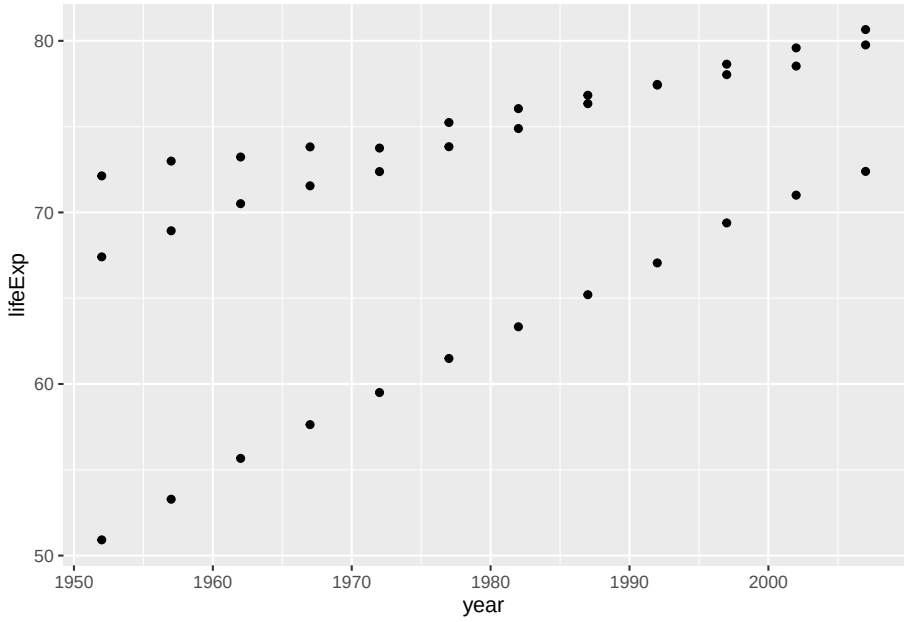


I would say things are looking good for Dutch people, their life expectancy is going up over the years!

1.2.8.4 Multiple countries scatterplot

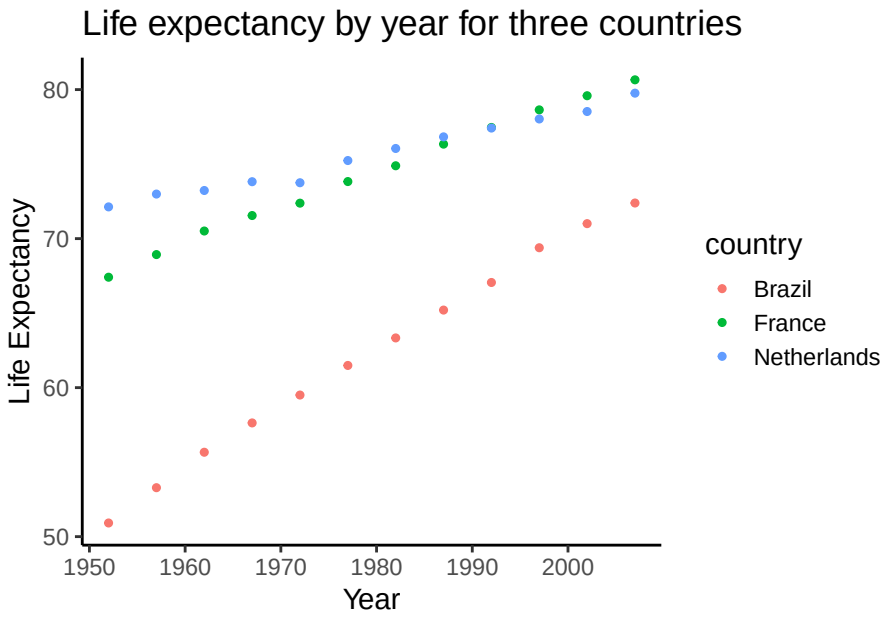
What if we want to look at a few countries altogether. We can do this too. We just change how we filter the data so more than one country is allowed, then we plot the data. We will also add some nicer color options and make the plot look pretty. First, the simple code:

```
# filter rows to contain countries of choice

smaller_df <- gapminder_df %>%
                filter(country %in% c("Netherlands","France","Brazil") == TRUE)

# plot the new data contained in smaller_df

ggplot(smaller_df, aes(y= lifeExp, x= year, group= country))+
  geom_point()
```



Nice, we can now see three sets of dots, but which are countries do they represent? Let's add a legend, and make the graph better looking.

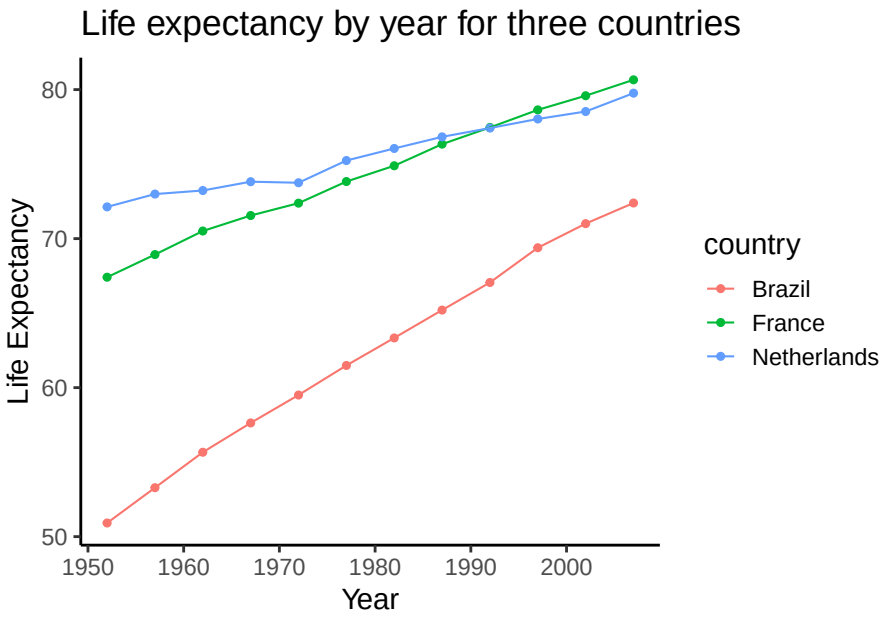
```
ggplot(smaller_df,aes(y= lifeExp, x= year,
                      group= country, color = country)) +
  geom_point()+
  theme_classic(base_size = 15) +
  ylab("Life Expectancy") +
  xlab("Year") +
  ggtitle("Life expectancy by year for three countries")
```



1.2.8.5 geom_line() connecting the dots

We might also want to connect the dots with a line, to make it easier to see the connection! Remember, ggplot2 draws layers on top of layers. So, we add in a new `geom_line()` layer.

```
ggplot(smaller_df,aes(y= lifeExp, x= year,
                      group= country, color = country)) +
  geom_point()+
  geom_line()+
  theme_classic(base_size = 15) +
  ylab("Life Expectancy") +
  xlab("Year") +
  ggtitle("Life expectancy by year for three countries")
```



1.2.9 Generalization Exercise

The following generalization exercises and writing assignment can be answered in the R Markdown document for this lab. Complete your work in that document and "knit" the file. Hand in your final product (knitting should generate an .html file) to Canvas. If you are unable to knit, submit the .Rmd file instead.

Use the code from above to attempt to solve the extra things we ask you do for this assignment. Your generalization exercises are as follows:

1. Make a graph plotting Life Expectancy by year for the five continents, using the `continent` factor. Make sure you change the title as well.
2. Make a graph plotting GDP per capita by year for the USA, Canada, and Mexico. Use the `gdpPercap` column for the GDP per capita data.
3. Make a new graph plotting anything you are interested in using the gapminder dataset. It just needs to be a plot that we have not given an example for.

1.2.10 Writing assignment

Describe what histograms are, how to interpret them, and what they are useful for. You should answer each of these questions:

a. What do the bars on a histogram represent?
b. How many bars can a histogram have?
c. What do the heights of the bars tell you?
d. What is on the x-axis and y-axis of a histogram?
e. What does the tallest bar on a histogram tell you?
f. What does the shortest bar on a histogram tell you?
g. What are some uses for histograms, why would you want to look at a histogram of some numbers that you collected?
h. Imagine you had two histograms, one was very wide and spread out, the other was very narrow with a very tall peak. Which histogram would you expect to contain more consistent numbers (numbers that are close to each other) and explain why.