**Study Guide Unit 1 (Week 1) Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
DATA 110 – Saidi**

**Intro to DATA 110 – Data Communication and Visualization**

*“The simple graph has brought more information to the data analyst’s mind than any other device.”* John Tukey

* [Intro to Data Vis](#_Why_Data_Visualization?)
* [Size of Data](#_The_Size_of)
* [Telling Stories with Data](#_Telling_Stories_with)
* [Better Visualizations](#_Better_Visualizations_)
* [Shape of Data](#_What_shape_is)
* [Categorical vs Continuous](#_Types_of_data:)
* [Introducing R Studio and R Markdown](#_Introducing_R_and)
* [Piping Operator](#_What_is_Piping)
* [Bar Charts for Categorical Data](#_Bar_Charts_for)
* [Intro to Tidyverse](#_Introduction_to_Tidyverse)
* [Homework Week 1](#_Homework_Week_1)

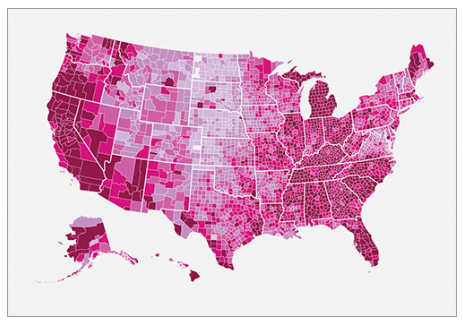
# **Why Data Visualization?**

Visualizations tell stories, find hidden patterns in data, and make compelling cases for making decisions. These stories can help you solve real-world problems, like decreasing crime, improving healthcare, and moving traffic on the freeway. They can shed light on social justice issues and disparities, and graphics can help politicians and companies make important decisions.

John Tukey

1. The greatest value of a picture is when it forces us to notice what we never expected to see.
2. The simple graph has brought more information to the data analyst’s mind than any other device.
3. Numerical quantities focus on **expected** values, graphical summaries on **unexpected** values.





Look at **Figures 1-1 and 1-2** Maps of U.S. unemployment for 2004 to 2009. Discuss these answers with your neighbor.

1. How is this map broken up?
2. What do the darker colors mean?
3. What regions had the most dark areas?
4. Do you see a pattern or trend from 2004 to 2009? Be specific.
5. Is there additional information you would have liked included on these maps?

# Data visualization is part art and part science. The challenge is to get the art right without getting the science wrong and vice versa. A data visualization first and foremost has to accurately convey the data. It must not mislead or distort. If one number is twice as large as another, but in the visualization they look to be about the same, then the visualization is wrong. At the same time, a data visualization should be aesthetically pleasing. Good visual presentations tend to enhance the message of the visualization. If a figure contains jarring colors, imbalanced visual elements, or other features that distract, then the viewer will find it harder to inspect the figure and interpret it correctly (Fundamentals of Data Visualization, Claus O. Wilke)

A shift in release of government data came in mid-2009 with the launch of [Data.gov](file:///C:\Users\rsaidi\Dropbox\Rachel\MontColl\DATA%20110\Notes\data.gov), a comprehensive catalog of data provided by federal agencies and represents transparency and accountability of groups and officials. Now all this information is in one place and better formatted for analysis and visualization.

# **The Size of Data (in case you didn’t know….)**

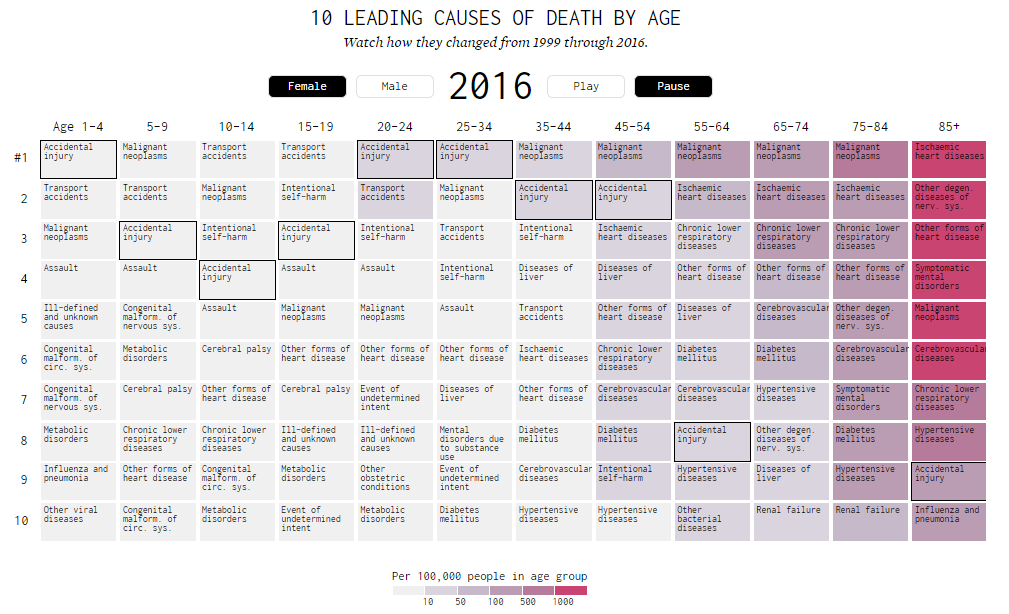
|  |  |  |
| --- | --- | --- |
| Potential database volumes in bytes for some typical applications (volumes estimated to the nearest order of magnitude). Strictly, bytes are counted in powers of 2 – 1 kilobyte is 1024 bytes, not 1000. | | |
| 1 megabyte | 1 000 000 | Single data set in a small project database |
| 1 gigabyte | 1 000 000 000 | Entire street network of a large city or small country |
| 1 terabyte | 1 000 000 000 000 | Elevation of entire Earth surface recorded at 30 m intervals |
| 1 petabyte | 1 000 000 000 000 000 | Satellite image of entire Earth surface at 1 m resolution |
| 1 exabyte | 1 000 000 000 000 000 000 | A possible 3-D representation of the entire Earth at 10 m resolution |
| 1 zettabyte | 1 000 000 000 000 000 000 000 | One-fifth of the capacity (in 2013) of U.S. National Security Agency Utah Data Center |

# **Telling Stories with Data**

## Some Reputable Journalism Sites with Visualizations

* Vox
* New York Times
* Wall Street Journal
* Bloomberg News
* Fivethirtyeight.com
* Washington Post
* Gapminder.org
* FlowingData (<https://flowingdata.com/>) by Nathan Yau or go to Data Underload: <https://flowingdata.com/category/projects/data-underload/>

## Intersection of Data, Art and Entertainment

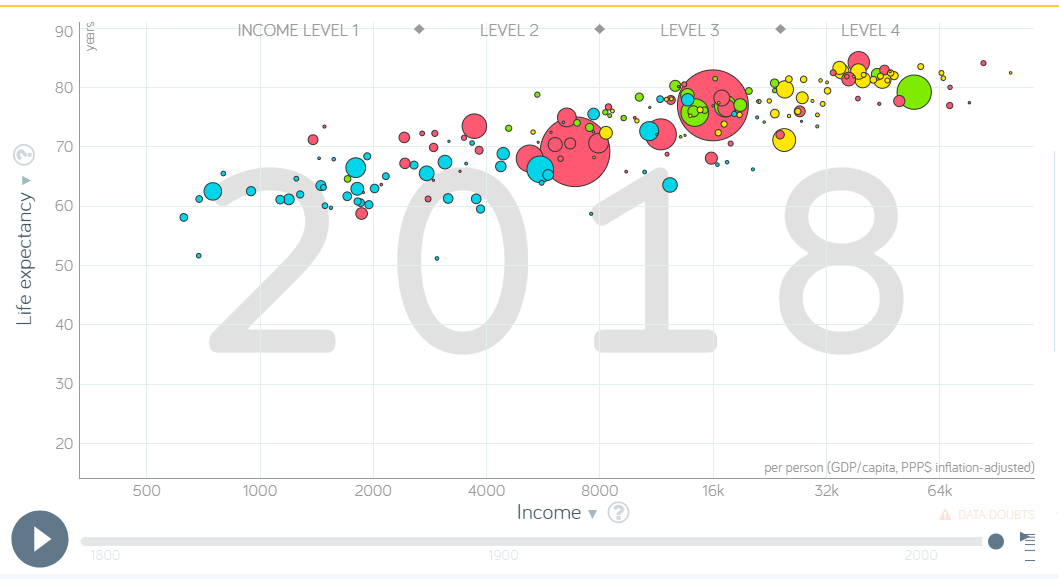
  
Data can be both informative and artful. Check out Nathan Yau’s many examples on <https://flowingdata.com/>. Look at the image of Nathan Yau’s “10 Leading Causes of Death by Age” <https://flowingdata.com/2018/10/02/shifting-death/> (see screen capture below). Discuss these answers with your neighbor.

1. Name all of the variables Yau considers.
2. What do the colors represent?
3. What is Yau’s intended effect in using animation?
4. Do you believe this is an effective data visualization? Why or why not?

## **Compelling Stories Through Data Exploration**

[200 Years, 200 Countries](https://www.youtube.com/watch?v=jbkSRLYSojo), by Hans Rosling

[Gapminder.org](https://www.gapminder.org/) (You should explore this entire site, but be sure to check out the Tools and Data links)



## **Better Visualizations** Check out [flowingdata.com](https://flowingdata.com/)

When creating visualizations, always consider the following:

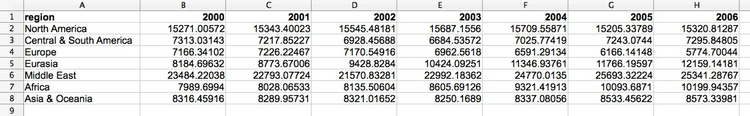
* Patterns
* Relationships
* Design
* Coding (legends and labels)
* Geometrical scaling
* Include source
* Consider your audience

## Principles of Analytic Graphics

* Show comparisons
* Show causality, mechanism, explanation
* Show multivariate data
* Integrate multiple modes of evidence
* Describe and document the evidence
* Content is king

# **What shape is your data?**

Particularly when data shows a time series for a single variable, it is often provided like this data on trends in international oil production by region, in “wide” format:



(Source: Peter Aldhous, from [U.S. Energy Information Administration](http://www.eia.gov/cfapps/ipdbproject/IEDIndex3.cfm?tid=5&pid=53&aid=1) data)

Here, all of the numbers represent the same variable, and there is a column for each year. This is good for people to read, but most software for data analysis and visualization does not work well with data in this format.

So if you receive **“wide” data,** you will usually need to convert it to **“long” format (**shown here) using some software, such as R or Open Refine:



(Source: Peter Aldhous, from [U.S. Energy Information Administration](http://www.eia.gov/cfapps/ipdbproject/IEDIndex3.cfm?tid=5&pid=53&aid=1) data)

Notice that now there is one column for each variable, which makes it easier for computers to understand.

# **Types of data: categorical vs. continuous (from Peter Aldhous)**

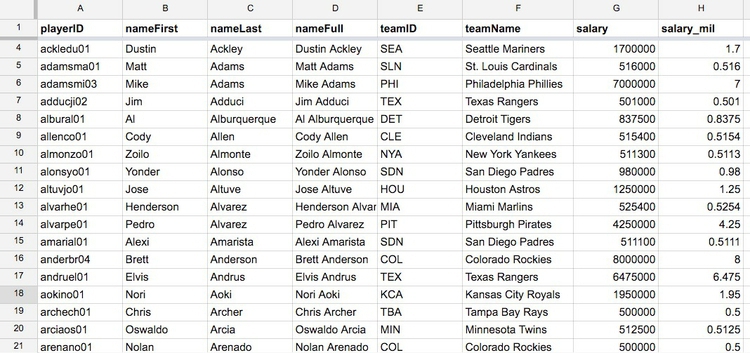
Before analyzing a dataset, or attempting to draw a graphic, it is important to consider what, exactly, you are working with.

Statisticians often use the term “variable.” This simply means any measure or attribute describing a particular item, or “record,” in a dataset. For example, school students might gather data about themselves for a class project, recording their gender and eye color, and height and weight. There is an important difference between gender and eye color, called “categorical” variables, and height and weight, termed “continuous.”

* **Categorical** variables are descriptive labels given to individual records, assigning them to different groups. The simplest categorical data is dichotomous, meaning that there are just two possible groups — in an election, for instance, people either voted, or they did not. More commonly, there are multiple categories. We analyze categorical data differently than continuous or discrete quantitative data.
* **Continuous** data is richer, consisting of numbers that can have a range of values on a sliding scale. Continuous variables might include temperature and amount of rainfall, age, height, weight, blood pressure, etc.

There is a third type of data we often need to consider: **date and time**. Perhaps the most common task in data journalism is to consider how a variable or variables have changed over time.

Datasets will usually contain a mixture of categorical and continuous variables. Here, for example, is a small part of a spreadsheet containing data on salaries for Major League Baseball players at the opening of the 2014 season:



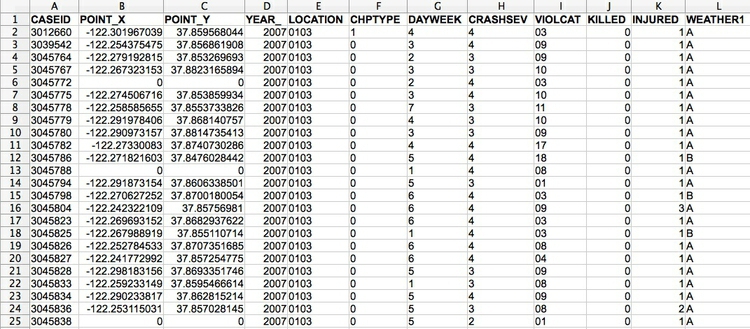
(Source: Peter Aldhous, data from [Lahman Baseball Database](http://www.seanlahman.com/baseball-archive/statistics/) data)

This is a typical data table layout

* the players --- form the rows
* variables --- arranged in columns

It is easy to recognize the categorical variables of teamID and teamName because they are each entered as text. The numbers for salary, expressed in full or in millions of dollars (salary\_mil), are continuous variables.

**Never** assume that every **number** in a dataset represents a **continuous variable.** Text descriptions can make datasets unwieldy, so database managers often adopt simpler codes such as letting numbers store categorical data.   
  
You can see this in the following example, showing data on traffic accidents resulting in injury or death in Berkeley, downloaded from a database maintained by researchers on the Berkeley campus.



(Source: Peter Aldhous, from [Transportation Injury Mapping System](http://tims.berkeley.edu/) data)

Of the numbers seen here, only the YEAR, latitudes and longitudes (POINT\_Y and POINT\_X) and numbers of people KILLED or INJURED actually represent continuous variables. (Look carefully, and you will see that these **numbers are justified right within each cell**. **The other numbers are justified left**, like the text entries, because they were imported into the spreadsheet as text values.)

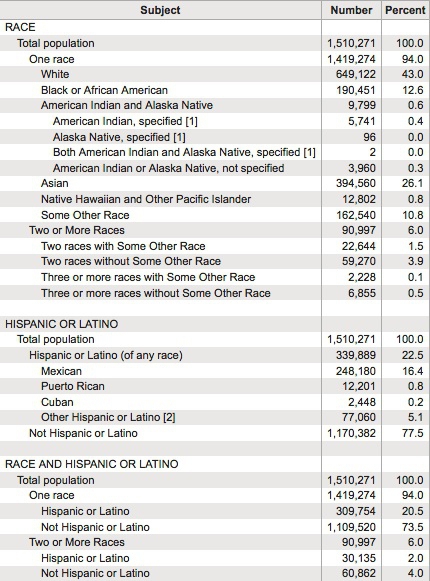
Like this example, many datasets are difficult to interpret without their supporting documentation. So each time you acquire a dataset, if necessary make sure you also obtain the “codebook” describing all of the variables/fields, and how they are coded. [Here is the codebook](http://paldhous.github.io/ucb/2016/dataviz/data/SWITRS_codebook.pdf) for the traffic accident data.

## Working with categorical data

You might imagine that there is little that you can do with categorical data alone, but it can be powerful, and can also be used to create new continuous variables.

The most basic operation with categorical data is to aggregate it by counting the number of records that fall into each category. This gives a table of **frequencies.**  These are often divided by the total number of records, then multiplied by 100 to show them as percentages of the total.

Here is an example, showing data on the racial and ethnic identities of residents of Alameda County, from the 2010 US Census:



(Source: [American FactFinder](http://factfinder2.census.gov/faces/nav/jsf/pages/index.xhtml), U.S. Census Bureau)

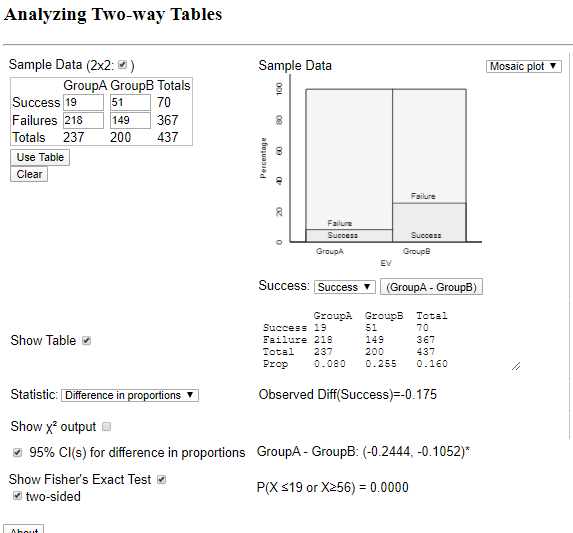
Creating frequency counts from categorical data creates a new continuous variable — what has changed is the level of analysis. In this example, the original data would consist of a huge table with a record for each person, noting their racial/ethnic identity as categorical variables; in creating the frequency table shown here, the level of analysis has shifted from the individual to the racial/ethnic group.

We can ask more interesting questions by considering two categorical variables together — as pioneering data journalist Philip Meyer showed when he collected and analyzed survey data to examine the causes of the 1967 Detroit Riot. In July of that year, one of the worst riots in U.S. history raged in the city for five days, following a police raid on an unlicensed after-hours bar. By the time calm was restored, 43 people were dead, 467 injured and more than 2,000 buildings were destroyed.

At the time, Detroit was regarded as a leader in race relations, so local racial discrimination was not initially seen as one of the main underlying causes of what happened. One popular theory at the time was that the riots were led by black residents who had moved to Detroit from the rural South. Meyer demolished this idea by examining data on whether or not the people surveyed had rioted, and whether they were brought up in the South or the North. He combined these results into a “contingency table” or “cross-tab”:

|  | **South** | **North** | **Total** |
| --- | --- | --- | --- |
| **Rioters** | 19 | 51 | 70 |
| **Non-rioters** | 218 | 149 | 367 |
| **Total** | 237 | 200 | 437 |

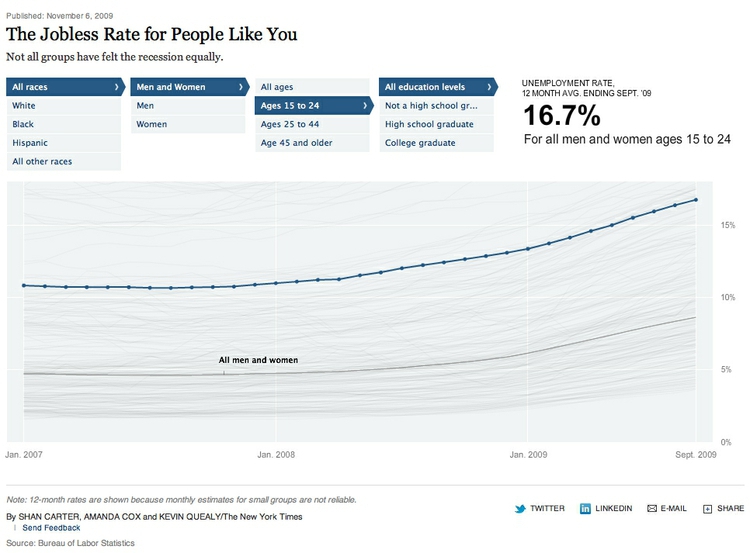
It certainly looks from these numbers as if Northerners were more likely to have participated in the riot.

But Meyer’s team only interviewed a sample of people from the affected neighborhoods, not everyone who lived there. If they had taken another sample, might they have obtained different results? This is one example where some more sophisticated statistical analysis can help. For contingency tables, a method known as the **chi-squared test** asks the relevant question: if Southerners and Northerners were in fact equally likely to have rioted, what is the likelihood of obtaining a sample as biased as this by chance alone? In this case, the chi-squared test told Meyer that the probability was less than one in a thousand. So Meyer felt confident writing in the newspaper that Northerners were more likely to have rioted. His work won a Pulitzer Prize for the Detroit Free Press and shifted the focus of political debate about the riot to racial discrimination in policing and housing in Detroit.

Try calculating the Chi-Squared Test by using the Rossman-Chance Applet for the Two-Way Table:

<http://www.rossmanchance.com/applets/ChisqShuffle.htm?FET=1>

## Categorical Data (continued)



Often we do not want to summarize a variable in a single number. But that doesn’t mean we have to show the entire distribution. Frequently data journalists divide the data into groups or “bins,” to reveal how those groups differ from one another. A good example is [this interactive graphic](http://www.nytimes.com/interactive/2009/11/06/business/economy/unemployment-lines.html) on the unemployment rate for different groups of Americans, published by The New York Times in November 2009:

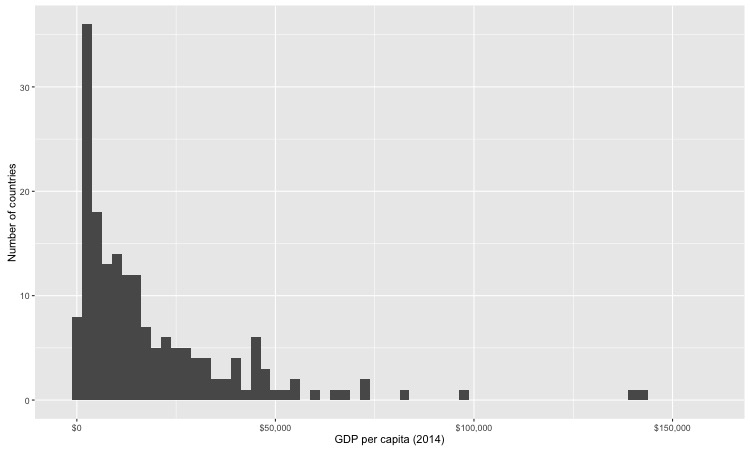
(Source: [The New York Times](http://www.nytimes.com/interactive/2009/11/06/business/economy/unemployment-lines.html))

In its base state, the graphic shows the overall jobless rate, and how this has changed over time. The buttons along the top allow you to filter the data to examine the rate for different groups. Most of the filtering is on categorical variables, but notice that the continuous variable of age is collapsed into a categorical variable dividing people into three groups: 15-24 years old, 24-44 years old, and 45 years or older.

To produce informative graphics that tell a clear story, data journalists often need to turn a continuous variable into a categorical variable by dividing it into bins.

Selecting the range of the bins depends on the story you are telling. In the jobless rate example, the bins divided the population into groups of young, mid-career and older workers, revealing how young workers in particular were bearing the brunt of the Great Recession. When binning data, it is again a good idea to look at the distribution, and experiment with different possibilities.

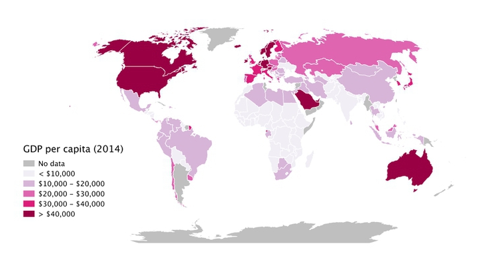
For example, the wealth of nations, measured in terms of gross domestic product (GDP) per capita in 2014, has a skewed distribution, similar to the baseball salaries. If we look at the distribution, drawn in increments of $2,500, we will see that it is highly skewed, rather like the baseball salaries:



(Source: Peter Aldhous, from [World Bank](http://data.worldbank.org/indicator/NY.GDP.PCAP.PP.CD) data)

Just a few countries had a GDP per capita of more than 50,000.

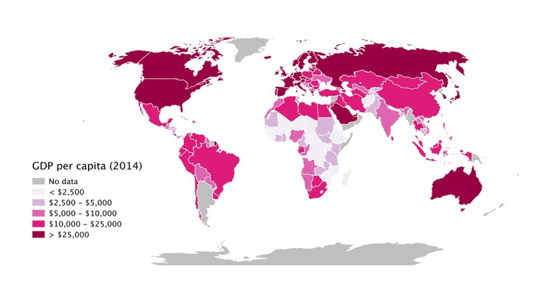
The maps below reveal how setting different ranges for the bins changes the story told by the data. For the first map, set the lower value for the top bin at $40,000, and then gave the bins equal ranges:



(Source: Peter Aldhous, from [World Bank](http://data.worldbank.org/indicator/NY.GDP.PCAP.PP.CD) data)

This might be useful for telling a story about how high per capita wealth is still concentrated into a small number of nations, but it does a fairly poor job of distinguishing between the per capita wealth of developing countries. For poorer people, small differences in wealth make a big difference to living conditions.

So for the second map, set the boundaries so that roughly equal numbers of countries fell into each of the five bins. Now Japan, most of Western Europe and Russia join the wealthiest bin, middle-income countries like Brazil, China, and Mexico are grouped in another bin, and there are more fine-grained distinctions between the per capita wealth of different developing countries:



(Source: Peter Aldhous, from [World Bank](http://data.worldbank.org/indicator/NY.GDP.PCAP.PP.CD) data)

Some visualization and mapping software gives you the option of putting equal numbers of records into each bin — usually called “quantiles” (the quartiles we encountered on the box plots are one example). Note that calculated quantiles won’t usually give you nice round numbers for the boundaries between bins. So you may want to adjust the values, as was done in the second map.

You may also want to examine histograms for obvious “valleys” in the data, which may be good places for the breaks between bins.

# **Introducing R and R Studio**

For homework, you will install R and R Studio using provided documents. Here are some shortcuts you will be able to use one you start to code:

|  |  |
| --- | --- |
| Operation | Shortcut |
| Insert Chunk | Ctrl Alt I |
| <- (assignment operator) | Alt hyphen |
| %>% (piping operator) | Ctrl Shift M |
| Comment out code blocks | Ctrl Shift C |
| Get info on function | Put cursor on function; F1 |
| Find and replace | Ctrl F |
| Clear | Ctrl L |

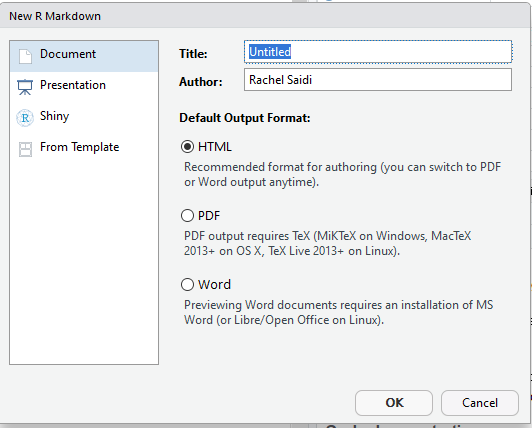
For more information on keyboard shortcuts, click on the **Tools tab** in **R Studio** and select **Keyboard Shortcut Help**

This is a really useful document with [R Markdown Reference Material](https://rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf)

**R Markdown to Explore Relationships**

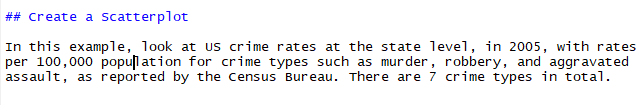
We will learn to code within the context of using R Markdown. Click here for more information on [R Markdown](https://rmarkdown.rstudio.com/articles_intro.html).

Start by opening a new R Markdown file. Select output format to be HTML or Word or PDF.

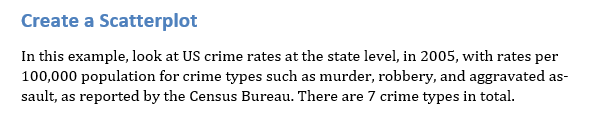


In the white space on the Markdown page, ## will give you subtitles – you know you have done it correctly because they are in blue

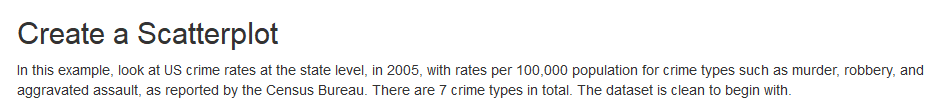
When you do not include hashtags, this gives text (in black). You can use #, ##, ###, etc. for different levels of subtitles, or no hashtags at all for plain text.



Here is how it looks when it is knitted in Word:



Here is how it looks when knitted to HTML



# **Create Chunks**

You can create chunks of code with the shortcut control+ alt+ I. Here is a chunk:



Once you have code typed in, to run just that chunk, click on the right green arrow.

Be sure to leave a line or two of space after a chunk before you add your next subtitle

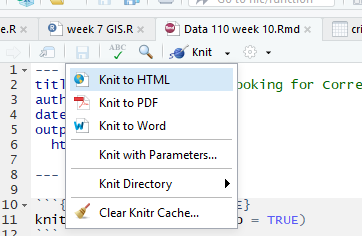
## Subtitle

If you don’t leave a space, it may not knit properly.

Once you have written all your subtitles, notes, and chunks, run the code to be sure it works.

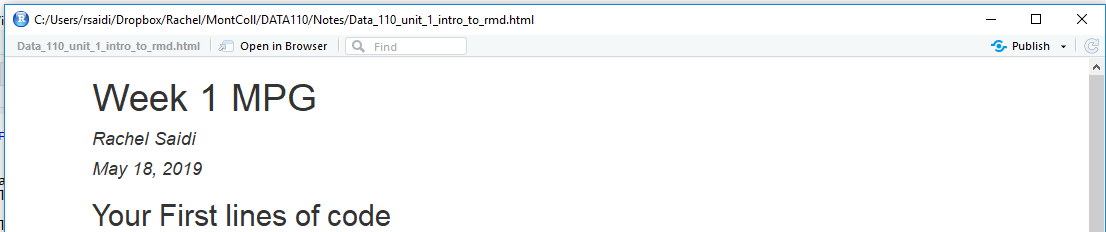
## Knitting

You can knit at any time to test how it looks. You can switch from knitting to Word, HTML, pdf at any time using the knit button at the top of your Markdown code.



## **Finally, Publish to Rpubs**

When you are done with your Markdown code, or when you just want to see what it looks like knitted to HTML, click the   
Rpubs “Publish” symbol:



You will need to set up a free Rpubs account, but once you do so, you will be able to publish your Markdown as an HTML page and share the link. Alternatively, you can publish your work to Github. We will learn about that later in the class.

## What is Piping Anyway?

## The Pipe Operator in R: Introduction (From Data Camp)

To understand what the pipe operator in R is and what you can do with it, it's necessary to learn the history behind it. Where does this weird combination of symbols come from? And why was it made like this?   
  
Now, you can look at the history from three perspectives: from a mathematical point of view, from a holistic point of view of programming languages, and from the point of view of the R language itself. You'll cover all three in what follows!

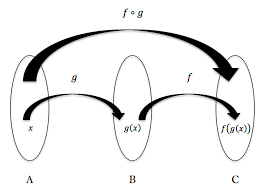
### History of the Pipe Operator in R

#### Mathematical History

If you have two functions, f: B→C and  g: A→B you can chain these functions together by taking the output of one function and inserting it into the next. In short, "chaining" means that you pass an intermediate result onto the next function.

For example, you can say, f(g(x)): g(x) serves as an input for f(), while x serves as input to g(). This is often called a composite function, where f is ***composed*** of g.

If you would want to note this down, you will use the notation f◦g, which reads as "f follows g", or “f of g”. Alternatively, you can visually represent this as:



#### Pipes in R

The history of this operator in R starts in 2012 when [Hadley Wickham](http://hadley.nz/) started the dplyr package on GitHub, which is based off of F# (pronounced F Sharp, as in Visual F# Programming Language, which is an open source, cross platform compiler, which can generate JavaScript and graphics processing unit (GPU) code. The question started:

*How can you implement F#'s forward pipe operator in R? The operator makes it possible to easily chain a sequence of calculations. For example, when you have an input data and want to call functions foo and bar in sequence, you can write data |> foo |> bar?*

"%>%" <- function(x,f) do.call(f,list(x))  
pi %>% sin  
[1] 1.224606e-16  
pi %>% sin %>% cos  
[1] 1  
cos(sin(pi))  
[1] 1

#### What is a Pipe?

In R, the pipe operator is, as you have already seen, %>%. You can think of this operator as being similar to the + in a ggplot2 statement. It takes the output of one statement and makes it the input of the next statement. When describing it, you can think of it as a "THEN".

This is one of the most powerful things about the Tidyverse. In fact, having a standardized chain of processing actions is called "a pipeline". Making pipelines for a data format is great, because you can apply that pipeline to incoming data that has the same formatting and have it output in a ggplot2 friendly format, for example.

### Why Use It?

R is a functional language, which means that your code often contains a lot of parenthesis, ( and ). When you have complex code, this often will mean that you will have to nest those parentheses together. This makes your R code hard to read and understand. Here's where %>% comes in to the rescue!

Take a look at the following example, which is a typical example of nested code:

# Initialize `x`  
x <- c(0.109, 0.359, 0.63, 0.996, 0.515, 0.142, 0.017, 0.829, 0.907)  
#Compute the logarithm of `x`, return suitably lagged and iterated differences, compute the exponential function and round the result  
round(exp(diff(log(x))), 1)

With the help of %>%, you can rewrite the above code as follows:

# Import `magrittr`  
library(magrittr)  
# Perform the same computations on `x` as above  
x %>% log() %>%  
 diff() %>%  
 exp() %>%  
 round(1)

In short, here are four reasons why you should be using pipes in R:

* You'll structure the sequence of your data operations from left to right, as opposed to from inside and out;
* You'll avoid nested function calls;
* You'll minimize the need for local variables and function definitions; And
* You'll make it easy to add steps anywhere in the sequence of operations.

### Additional Pipes

Even though %>% is the (main) pipe operator of the magrittr package, there are other operators that you should know and that are part of the same package:

## The compound assignment operator %<>%;

# Initialize `x`   
x <- rnorm(100)  
# Update value of `x` and assign it to `x`  
x %<>% abs %>% sort

\* Warning: take care in using the compound assignment operator, since it goes back AND forth. You should get comfortable with the one-directional pipe operator first.

## How to Use Pipes in R

Now you know how the %>% operator originated, what it actually is and why you should use it. It is time for you to discover how you can actually use it to your advantage. You will see that there are quite some ways in which you can use it!

### Basic Piping

Before you go into the more advanced usages of the operator, it's good to first take a look at the most basic examples that use the operator. In essence, you'll see that there are 3 rules that you can follow when you're first starting out:

* f(x) can be rewritten as x %>% f

In short, this means that functions that take one argument, function(argument), can be rewritten as follows: argument %>% function(). Take a look at the following, more practical example to understand how these two are equivalent:

# Compute the logarithm of `x`   
x %>% log()

### Compound Assignment Pipe Operations

There are situations where you want to overwrite the value of the left-hand side, just like in the example right below. Intuitively, you will use the assignment operator <- to do this.  
  
# Load in the Iris data

iris <- read.csv(url("http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"), header = FALSE)

# Add column names to the Iris data

names(iris) <- c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width", "Species")

# Compute the square root of `iris$Sepal.Length` and assign it to the variable

iris$Sepal.Length <-   
 iris$Sepal.Length %>%  
 sqrt()

However, there is a compound assignment pipe operator, which allows you to use a shorthand notation to assign the result of your pipeline immediately to the left-hand side:

# Compute the square root of `iris$Sepal.Length` and assign it to the variable  
iris$Sepal.Length %<>% sqrt

# Return `Sepal.Length`  
iris$Sepal.Length

**Note** that the compound assignment operator %<>% needs to be the first pipe operator in the chain for this to work. This is completely in line with what you just read about the operator being a shorthand notation for a longer notation with repetition, where you use the regular <- assignment operator.  
As a result, this operator will assign a result of a pipeline rather than returning it.

# **Bar Charts for Categorical Data (from Nicole Radziwill)**

To create a bar chart/bar plot in R with the barplot function using categorical data, which is a collection of numbers that represent frequencies (or counts) of events or outcomes that fall into different groups or categories. [Note: If you are trying to display distributions of quantitative data, choose a histogram instead. Bar charts are for categorical data only. BAR CHARTS ARE NOT THE SAME AS HISTOGRAMS!]

The lengths of the bars are proportional to the values they represent, and the bars can be oriented vertically or horizontally.

* Good bar charts are labeled nicely, with a clear description of the categories that are being counted on the horizontal (x) axis, and a label on the vertical (y) axis that indicates whether frequencies or counts are displayed.
* Membership into each category should be mutually exclusive. That is, you don’t want an observation to appear in multiple bars.
* Pie charts should be avoided. A bar chart is a better way to display your data. If you are trying to illustrate a collection of items that naturally add up to 100%, a pie chart may be appropriate. However, if there are multiple categories where it may be difficult to distinguish which slice is bigger (such as one observation of 28% and another observation of 29%) a bar chart may be more appropriate.
* If you want to display your data in terms of TWO categorical variables, choose a segmented bar chart (described in a separate chapter). Even More Caution: There is a BIG DIFFERENCE between a bar chart and a histogram! Even though a bar chart looks really similar to a histogram at first glance, take a close look at what kind of data is on the horizontal (x) axis. In a bar chart, the horizontal axis lists categories. In a histogram, the horizontal axis will contain ranges of numbers that represent a continuum (e.g. 0-10, 10-20, 20-30 and so forth). Also, in a bar chart, there will be some space between the bars indicating that the categories are separate from one another - whereas with a histogram, there will be no space between the bars! The bars will be very cozy in a histogram, mashed up against one another like they're at a crowded party, whereas the bars in a bar chart need a little more breathing room, and thus are distanced from one another.

[Click here](https://rpubs.com/rsaidi/605913) for your first link to Rpubs version of the following notes.  
  
Week 1 Intro to R and Markdown and Tidyverse

Rachel Saidi

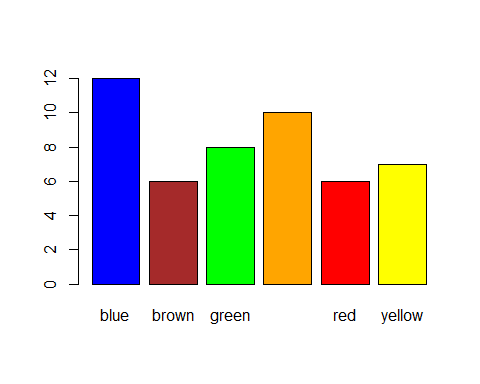
April 27, 2020

## Bar Charts (from Nicole Radziwill)

For small data sets, you may have already tallied up your observations, and you don’t need to load a whole file in to create your bar chart. Here is an example of data generated by opening one package of regular M&Ms to look at the distribution of colors, working with your data as a vector. The counts are: 12 blue ones, 6 brown ones, 8 green ones, etc.

We set a vector for the counts, a vector for the respective names of the values, and a vector of the palatte of colors to be used (these match the “names”). Then we call a barplot of the counts using the vector of colors.

mm.counts <- c(12,6,8,10,6,7)  
names(mm.counts) <- c("blue","brown","green","orange",  
"red","yellow")  
mm.colors <- c("blue","brown","green","orange","red","yellow") # creates a vector of the palette of colors to be used in the bar chart (that match the m & m colors)  
barplot(mm.counts,col=mm.colors)



# **Introduction to Tidyverse Using Continuous Data**

### Load Data from Three Different Sources

In the following notes, you will load data from three different sources: \* directly from a URL \* directly from pre-build datasets in R \* from a file you save in your own folder.

1. Load Data from a URL

You can load data from a folder or you can load data directly from a URL. The next example loads the dataset, “Test Scores”, directly from the URL where it resides.

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

## -- Attaching packages -------------------------------------------------- tidyverse 1.2.1 --

## v ggplot2 3.2.1 v purrr 0.3.3  
## v tibble 2.1.3 v dplyr 0.8.3  
## v tidyr 1.0.0 v stringr 1.4.0  
## v readr 1.3.1 v forcats 0.4.0

## -- Conflicts ----------------------------------------------------- tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

allscores <- readr::read\_csv("https://goo.gl/MJyzNs")

## Parsed with column specification:  
## cols(  
## group = col\_double(),  
## pre = col\_double(),  
## post = col\_double(),  
## diff = col\_double()  
## )

dim(allscores)

## [1] 22 4

Notice R interprets the variable “group” as continuous values (col\_double). We will fix this later. The command “dim” provides the dimensions of the data, which are 22 observations (rows) by 4 variables (columns).

## Introducing ggplot2 and the grammar of graphics

### ggplot2 is a package that will load when you load tidyverse, or you can simply load it on its own

The “gg” in ggplot2 stands for “grammar of graphics,” an approach to drawing charts devised by the statistician Leland Wilkinson. Rather than thinking in terms of finished charts like a scatter plot or a column chart, it starts by defining the coordinate system (usually the X and Y axes of a cartesian system), maps data onto those coordinates, and then adds layers such as points, bars and so on. This is the logic behind ggplot2 code.

Some key things to understand about ggplot2:

. ggplot This is the master function that creates a ggplot2 chart.

. aes This function, named for “aesthetic mapping,” is used whenever data values are mapped onto a chart. So it is used when you define which variables are plotted onto the X and Y axes, and also if you want to change the size or color of parts of the chart according to values for a variable.

. geom All of the functions that add layers to a chart start with geom, followed by an underscore, for example geom\_point() or geom\_bar(). The code in the brackets for any geom layer styles the items in that layer, and can include aes mappings of values from data.

. theme This function modifies the appearance of elements of a plot, used, for example, to set size and font face for text, the position of a legend, and so on.

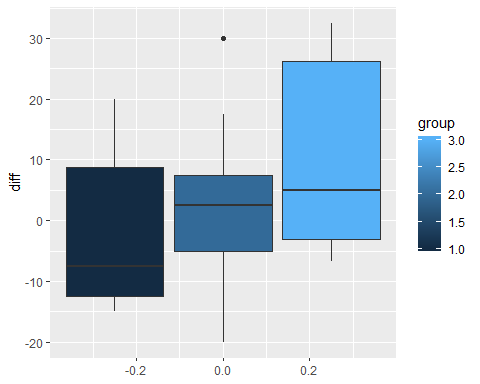
. scale Functions that begin with scale, followed by an underscore, are used to modify the way an aes mapping of data appears on a chart. They can change the axis range, for example, or specify a color palette to be used to encode values in the data.

. + is used each time you add a layer, a scale, a theme, or elements like axis labels and a title After a + you can continue on the same line of code or move the next line. I usually write a new line after each +, which makes the code easier to follow.

## Use Side-by-Side Boxplots

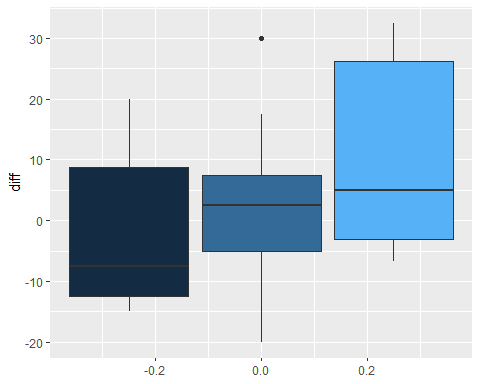
Here is some easy code to create 3 groups of boxplots with some easy-to-access data, filled by group. Since the groups are discrete, you can get rid of the shading.

boxpl <- allscores %>%   
 ggplot() +   
 geom\_boxplot(aes(y=diff, group=group,fill=group))  
boxpl



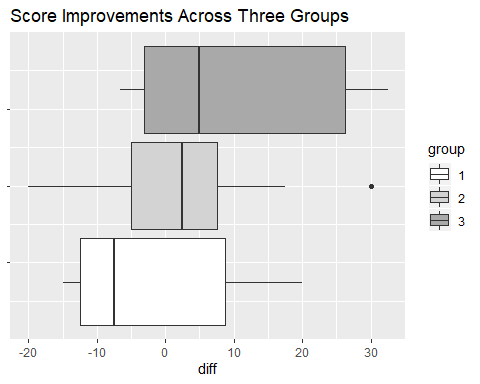
Notice that the legend give a continuous range of values for the scores, even though the scores are only 1, 2, or 3. The code guides(fill = FALSE) will get rid of the legend.

boxpl2 <- boxpl + guides(fill=FALSE)  
boxpl2



Ensure that the groups are considered as factors, rather than numbers. Then manually fill with the 3 colors: white, light gray, and dark gray. Make the boxplots orient horizontally.

allscores %>%  
 mutate(group=factor(group, levels=c("1","2","3"), ordered=TRUE)) %>%  
 ggplot() + geom\_boxplot(aes(y=diff, group=group, fill=group)) +  
 scale\_fill\_manual(values=c("white","lightgray","darkgray")) +  
 theme(axis.text.y=element\_blank()) +  
 ggtitle("Score Improvements Across Three Groups") +  
 coord\_flip()



## Load Built in Data from R

Some data frames are built in to R, such as mpg. Load the data, then use str and head to look at the data.

{r mpg} loads the data. Alternatively, you can use the command: load(“mpg”)

You will look at the data using the command “str” (gives the structure of the data), “head” (lists the first 6 rows of observations in the dataset), and “describe” from the “psych” package (gives quite detailed summary statistics on the continuous variables).

# {r mpg, warning = FALSE}   
# install.packages("tidyverse")  
# install.packages("psych")  
library(tidyverse)  
library(psych) # used for the "describe" command below

##   
## Attaching package: 'psych'

## The following objects are masked from 'package:ggplot2':  
##   
## %+%, alpha

str(mpg)

## Classes 'tbl\_df', 'tbl' and 'data.frame': 234 obs. of 11 variables:  
## $ manufacturer: chr "audi" "audi" "audi" "audi" ...  
## $ model : chr "a4" "a4" "a4" "a4" ...  
## $ displ : num 1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...  
## $ year : int 1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...  
## $ cyl : int 4 4 4 4 6 6 6 4 4 4 ...  
## $ trans : chr "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...  
## $ drv : chr "f" "f" "f" "f" ...  
## $ cty : int 18 21 20 21 16 18 18 18 16 20 ...  
## $ hwy : int 29 29 31 30 26 26 27 26 25 28 ...  
## $ fl : chr "p" "p" "p" "p" ...  
## $ class : chr "compact" "compact" "compact" "compact" ...

head(mpg)

## # A tibble: 6 x 11  
## manufacturer model displ year cyl trans drv cty hwy fl class   
## <chr> <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>   
## 1 audi a4 1.8 1999 4 auto(l5) f 18 29 p compa~  
## 2 audi a4 1.8 1999 4 manual(m5) f 21 29 p compa~  
## 3 audi a4 2 2008 4 manual(m6) f 20 31 p compa~  
## 4 audi a4 2 2008 4 auto(av) f 21 30 p compa~  
## 5 audi a4 2.8 1999 6 auto(l5) f 16 26 p compa~  
## 6 audi a4 2.8 1999 6 manual(m5) f 18 26 p compa~

describe(mpg)

## vars n mean sd median trimmed mad min max range skew  
## manufacturer\* 1 234 NaN NA NA NaN NA Inf -Inf -Inf NA  
## model\* 2 234 NaN NA NA NaN NA Inf -Inf -Inf NA  
## displ 3 234 3.47 1.29 3.3 3.39 1.33 1.6 7 5.4 0.44  
## year 4 234 2003.50 4.51 2003.5 2003.50 6.67 1999.0 2008 9.0 0.00  
## cyl 5 234 5.89 1.61 6.0 5.86 2.97 4.0 8 4.0 0.11  
## trans\* 6 234 NaN NA NA NaN NA Inf -Inf -Inf NA  
## drv\* 7 234 4.00 0.00 4.0 4.00 0.00 4.0 4 0.0 NaN  
## cty 8 234 16.86 4.26 17.0 16.61 4.45 9.0 35 26.0 0.79  
## hwy 9 234 23.44 5.95 24.0 23.23 7.41 12.0 44 32.0 0.36  
## fl\* 10 234 NaN NA NA NaN NA Inf -Inf -Inf NA  
## class\* 11 234 NaN NA NA NaN NA Inf -Inf -Inf NA  
## kurtosis se  
## manufacturer\* NA NA  
## model\* NA NA  
## displ -0.91 0.08  
## year -2.01 0.29  
## cyl -1.46 0.11  
## trans\* NA NA  
## drv\* NaN 0.00  
## cty 1.43 0.28  
## hwy 0.14 0.39  
## fl\* NA NA  
## class\* NA NA

It is essential to recognize that variables may be: int (integer), num (numeric), or double vs char (character) and factor (for categories)

Typically, chr or factor are used for discrete variables and int, dbl, or num for continuous variables.

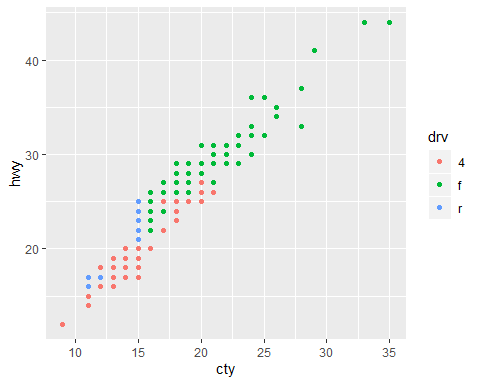
## Now make a scatterplot using ggplot2

Make a scatterplot of city vs highway miles per gallon, but sort/color points by either 4-wheel, front-wheel, or rear-wheel drive

Here is how we will code:

1. name the plot: “plot1” <-
2. call back the name of the dataset “mpg” and “pipe it” (more on that later) to create the frame for your plot
3. call “ggplot” to make a set of axes, with the aesthetics (aes) for city and highway mpg, but color points by the factors for drv
4. add geom\_point to see the points
5. call plot1 to see the entire plot

plot1 <- mpg %>%   
 ggplot(aes(cty, hwy, color = drv))+   
 geom\_point()  
plot1

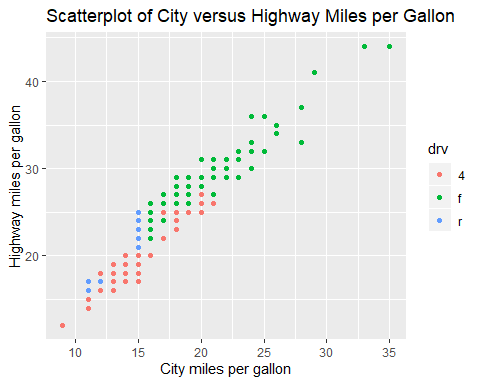


Notice that the blue points for rear-wheel drive are only at the lower left side of the plot (i.e., not great mpg). Red points for 4-wheel drive have a wider spread of points, but they are also mainly at the lower left corner of the plot. The green points for front-wheel drive are mostly at the upper right, for the higher mpg.

## Add a title and labels

Although there are already axes labels, we can do better. We should also add a title

plot1 <- mpg %>%   
 ggplot(aes(cty, hwy, color = drv))+   
 geom\_point()+  
 xlab("City miles per gallon") +  
 ylab("Highway miles per gallon") +  
 ggtitle("Scatterplot of City versus Highway Miles per Gallon")  
plot1



# **Homework Week 1**

1. (Ungraded) Install R and R-Studio on your computer using the documents and videos provided. Sign up for a free Rpubs account. Start exploring R Studio by copying the code from the Markdown page into your own Markdown document.
2. (Ungraded) Reread these notes and try copying, pasting, and running the code provided above.
3. (Worth up to 10 points) Listen to the podcast: [Stats and Stories](https://statsandstories.net/media1/2018/11/15/understanding-data-in-the-digital-age-stats-and-stories-episode-70) with Statistician and Journalist Mark Hanson   
   ([here is the abstract](https://bb-montgomerycollege.blackboard.com/bbcswebdav/pid-4278450-dt-content-rid-35530298_1/xid-35530298_1)). Write a one-paragraph reaction to this podcast. Be sure to edit for grammar/spelling/sentence structure.
4. (Worth up to 10 points)  Explore the “reputable sites” listed in the notes above (Vox, NY Times, FiveThiryEight, etc.).
   1. Select one site and then select one relatively current visualization (you certainly may be interested in COVID-19 related visualizations).
   2. On a Word document, paste the image (must have color).
   3. On this same document, write a summary about this image.
   4. Describe what has been done well in this visualization.
   5. Describe what could be improved; it might show selection bias or any other type of distortion.
   6. Be sure to include a link/url to your data visualization (and be sure to put your name at the top)

Submit the two graded assignments via the course Week 1 Assignment Dropboxes by **11:59 pm on Wednesday, June 10th .** We will present/discuss your submissions during the Thursday June 11th zoom class.

# Due Next Week – Get a Head Start

1. (Worth 10 points) Follow the [Week 2 Homework Tutorial](http://rpubs.com/rsaidi/518422)  (<http://rpubs.com/rsaidi/518422>). In your own new Markdown file, copy the code to create all **five plots**. (Plot 1, Plot 2, Plot 3, Plot 4, and Plot 5). Knit the markdown and publish it in Rpubs, then post the Rpubs link in the Assignment Dropbox.

This assignment will be due **by 11:59 pm on Wednesday, June 17th.** We will present your submissions in class on Thursday, June 18th.