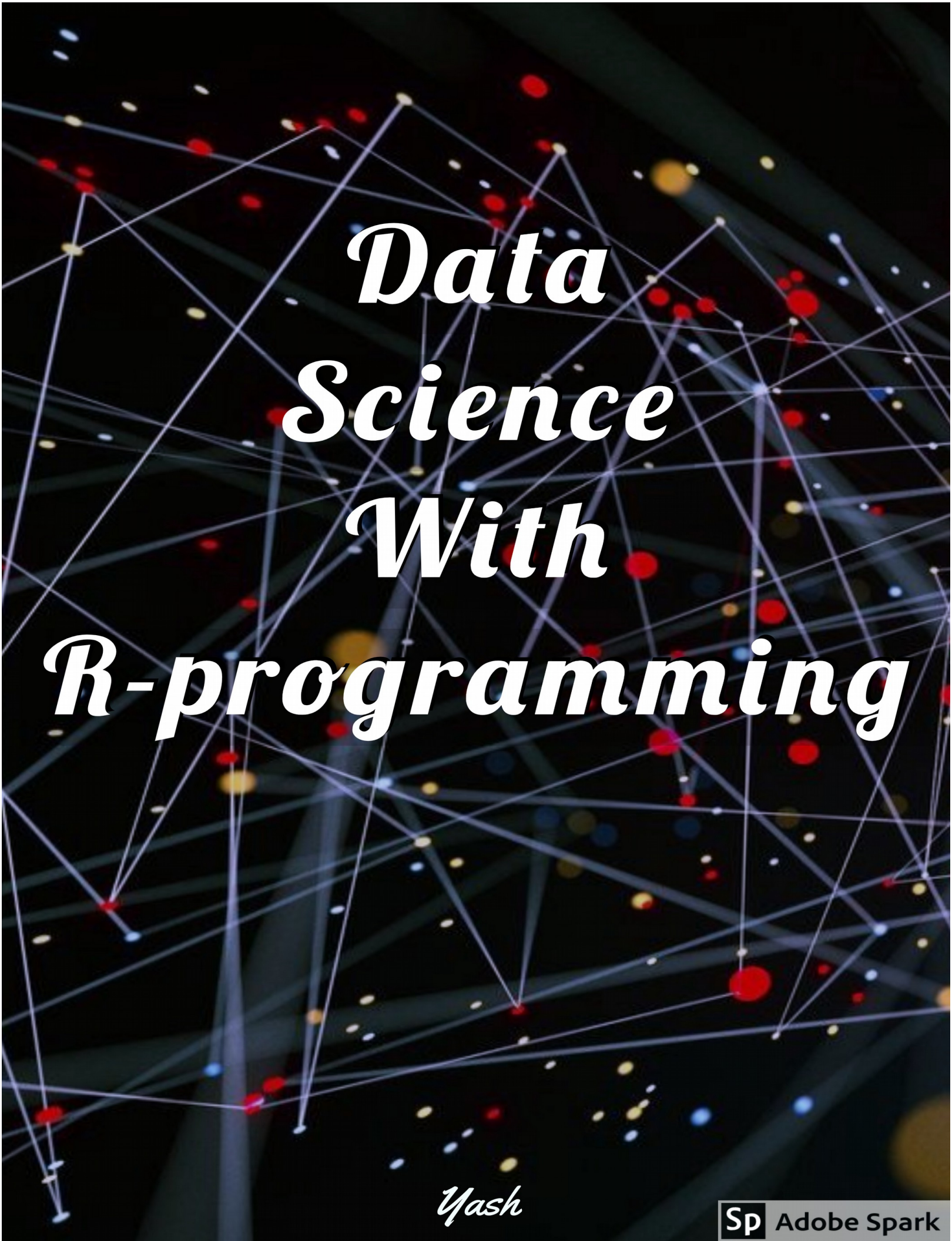

Data
Science
With
R-programming
Yash
Sp Adobe Spark

Dedication

This book is dedicated to every data science Teacher who loves to teach but hates all the crap that come with it…..

1. Data Science Life Cycle

1.1 concept of data driven science

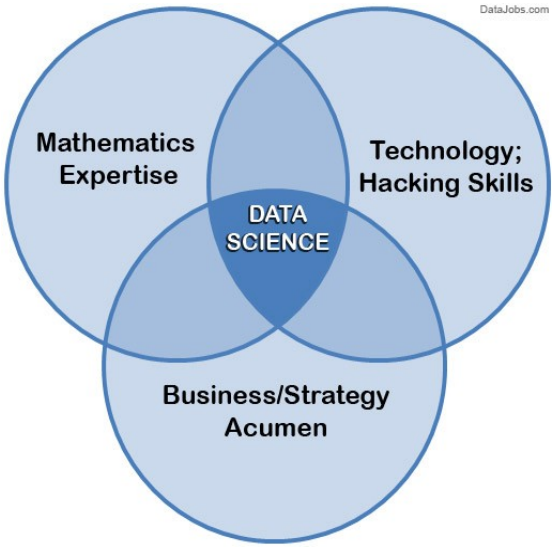
What is data science?

Data science is a technique or field that uses scientific method,processes,algorithms and systems to extract knowledge and essential information from data in various from, both structured and unstructured format

Data science is a concept to unify statistics,data analysis, machine learning, AI, deep learning and their related method

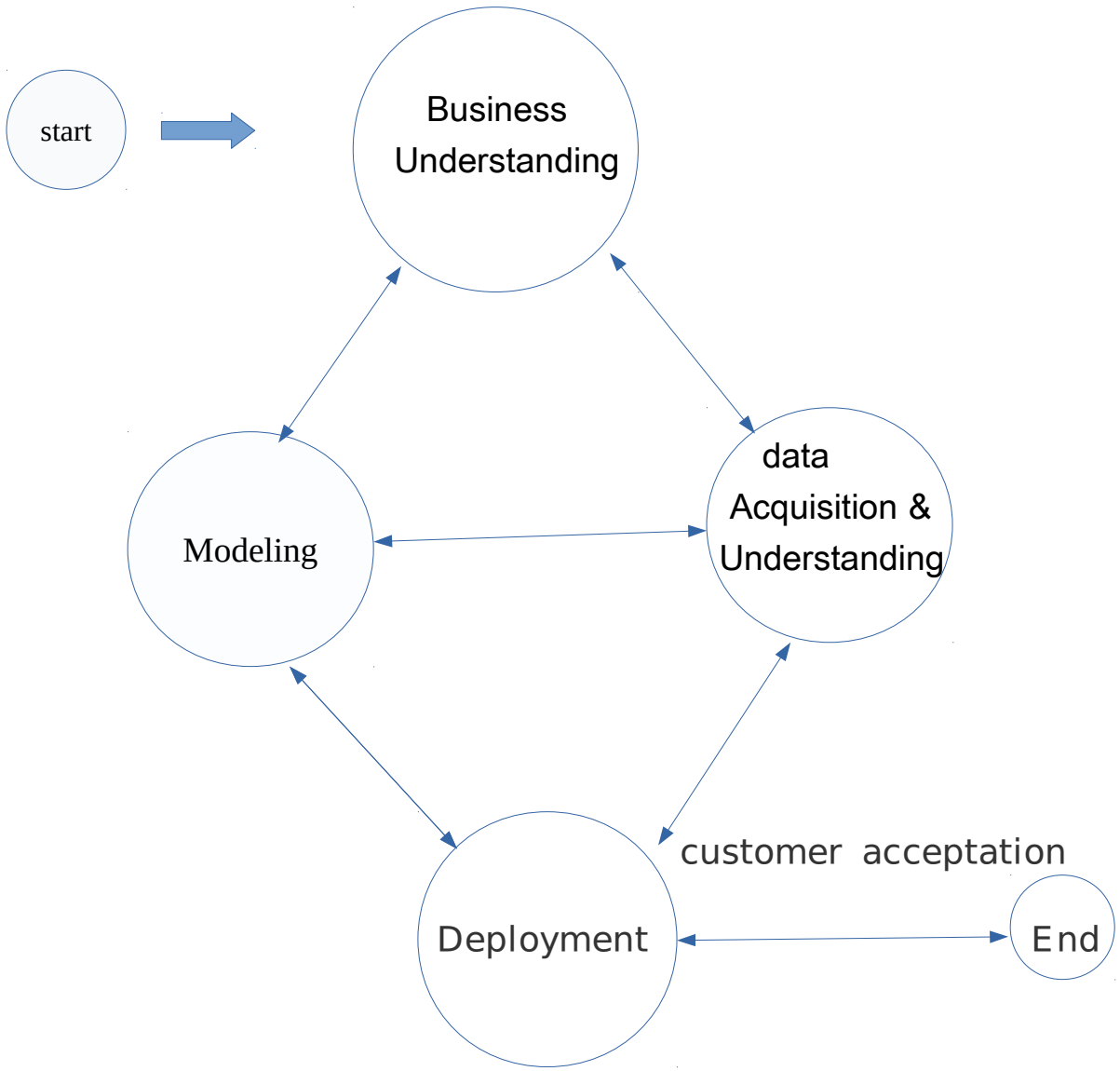
Data science is the field of study that combines domain expertise, programming skills, and knowledge of math and statistic to extract meaningful insights from data data science practitioners apply machine learning algorithm and deep learning algorithm to numbers, text, images, video, audio and more to produce artificial intelligence(AI) systems that perform tasks which ordinarily require human intelligence. In turn, these systems generate insights that analysts and business users translate into tangible business value

Data science is a blend of skills in three major area:



Data science contain mathematics, statistic and technology skill like python, R programming, big-data, data-warehousing, databases, data sets, data engineering and Business knowledge or strategy Acumen.

Data science Life cycle



Data Acquisition & Understanding :

1. Data Source → Databases, Data warehouses, Server, Cloud-databases etc.
2. Pipeline → Data science pipelines are sequences of processing and analysis steps applied to data for a specific purpose
3. Environment → On-premises vs Cloud Databases vs Data Lake vs small vs big-data
4. Wrangling, Exploration and Cleaning → structured vs unstructured, data data validation and Cleanup Visualization

Modeling :

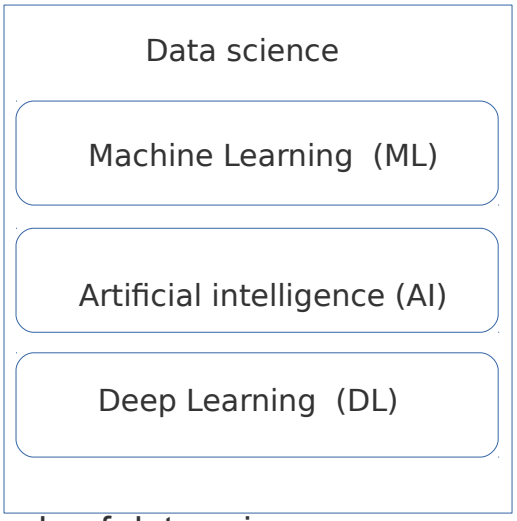
1. Feature Engineering→ Feature Engineering is the process of using Domain Knowledge of data to create features that make machine learning algorithm work properly on data

2. Model Training → Model Training is process of making model that Model help to solve a particular problems

3. Model Evaluation → Model Evaluation in this process basically we perform above task
    A) Testing (data load)
    B) Accuracy Testing (Check how much accurate our model)
    C) Efficiency Testing(Check how much efficient)
    D) Cross Validation
    E) Model Reporting

Deployment :

Scoring Performance and monitoring etc.



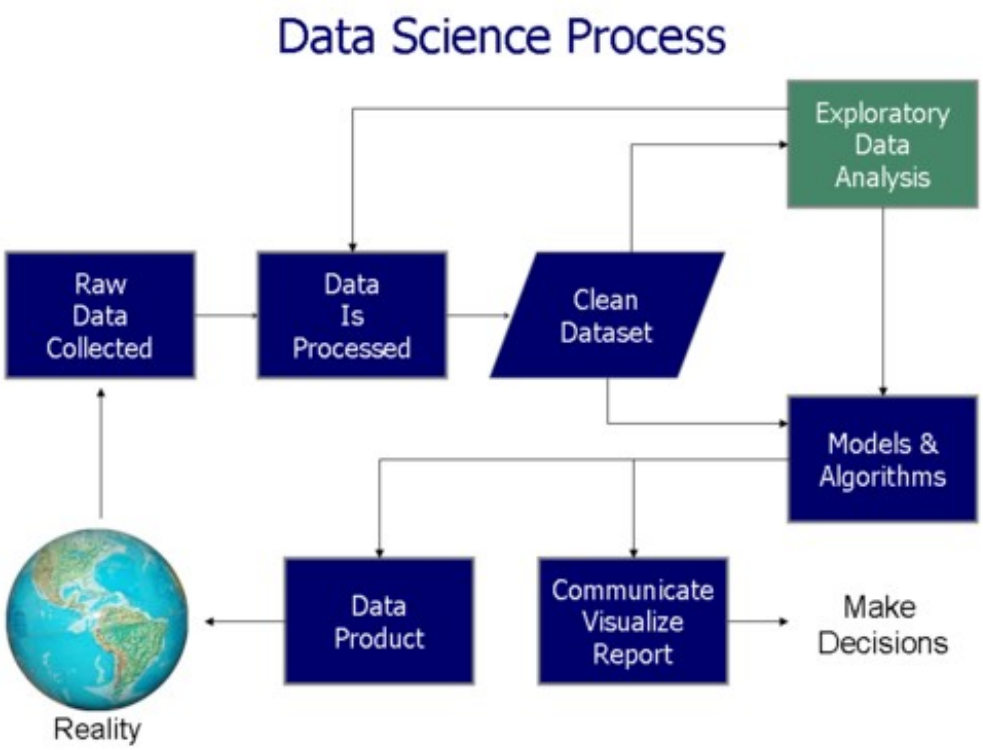
Three major pole of data science

1. machine learning → Machine learning focuses on the development of computer programs that can access data and use learn for themselves

2. Artificial intelligence → The theory and development of computer systems able to perform tasks normally requiring human intelligence. Such as visual perception, speech recognition, decision-making and translation between languages.

3. deep learning → **Deep learning** is a subset of **machine learning** in artificial intelligence (AI) that has networks capable of **learning** unsupervised from data that is unstructured or unlabeled. Also known as **deep** neural **learning** or **deep neural network**

1.2 Flow process for data science problems



Raw data : (sometimes called source **data** or atomic **data**) is **data** that has not been processed for use. A distinction is sometimes made between **data** and information to the effect that information is the end product of **data** processing. **Raw data** that has undergone processing is sometimes referred to as cooked **data**.

Data Cleaning :

Data cleansing or data cleaning is the process of detecting and correcting corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data.

exploratory data analysis :

In statistics, exploratory data analysis is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task

1.3Data Science problems and a solution framework

Is it a Data Science problem?

A true data science problem may:

- Categorize or group data
- Identify patterns
- Identify anomalies
- Show correlations
- Predict outcomes

A good data science problem should be **specific and conclusive**. For example:

As personal wealth increases, how do key health markers change?

Where in California do most people with heart disease live?

Conversely, a **vague and unmeasurable** problem may not be a good fit for a data science solution. For example:

What is the link between finances and health?

Are people in California healthier?

Solution:

Linear Regression

Logistic Regression.

Decision Tree

SVM

Naive Bayes.

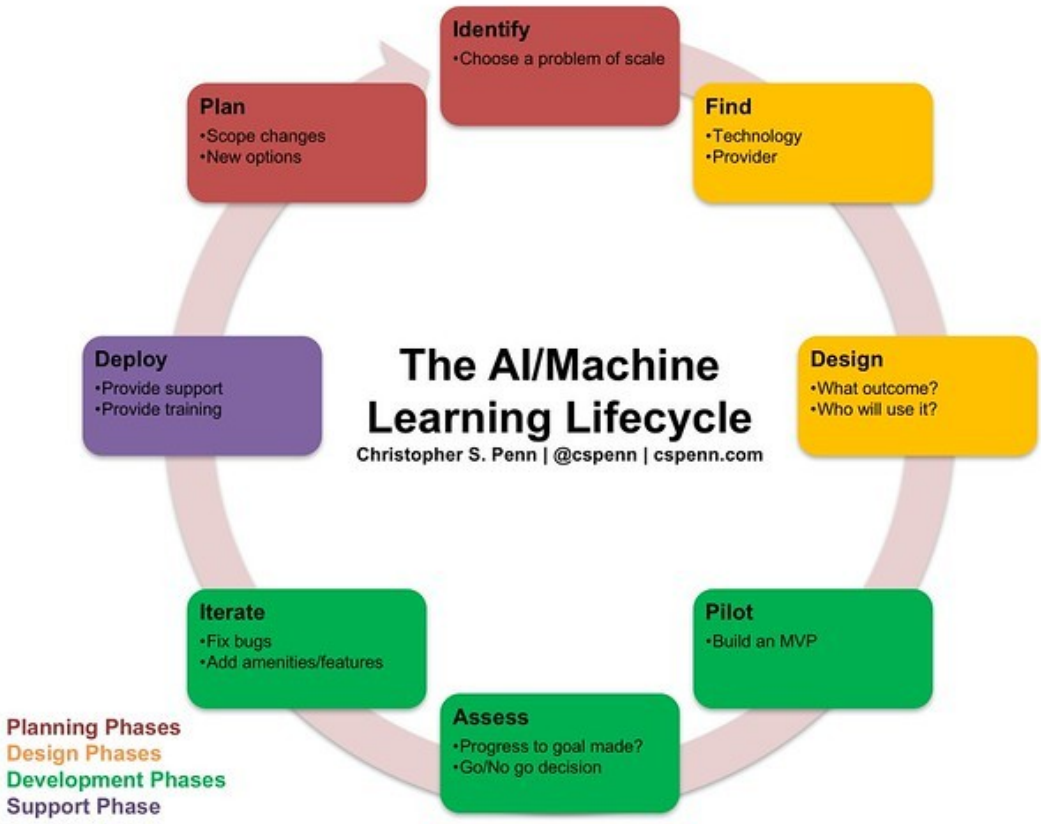
kNN.

K-Means.

Random Forest.

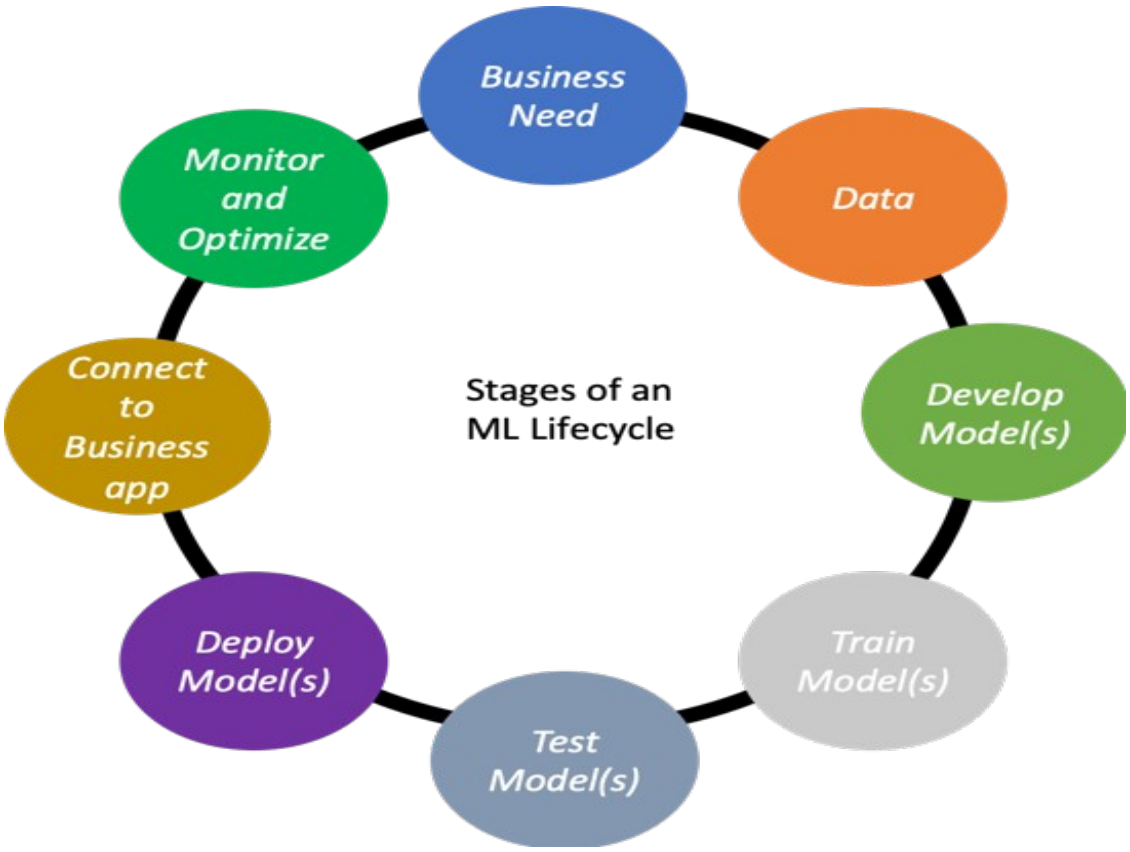
1.4 Artificial Intelligence(AI)-Life cycle & Applications

Artificial intelligence is a technology that is already impacting how users interact with, and are affected by the Internet. In the near future, its impact is likely to only continue to grow. AI has the potential to vastly change the way that humans interact, not only with the digital world, but also with each other, through their work and through other socioeconomic institutions – for better or for worse.



1.5 Machine Learning(ML)-Life Cycle & Applications

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.



1. Virtual Personal Assistants

Siri, Alexa, Google Now are some of the popular examples of virtual personal assistants. As the name suggests, they assist in finding information, when asked over voice. All you need to do is activate them and ask "What is my schedule for today?", "What are the flights from Germany to London", or similar questions. For answering, your personal assistant looks out for the information, recalls your related queries, or send a command to other resources (like phone apps) to collect info. You can even instruct assistants for certain tasks like "Set an alarm for 6 AM next morning", "Remind me to visit Visa Office day after tomorrow".

2. Predictions while Commuting

Traffic Predictions: We all have been using GPS navigation services. While we do that, our current locations and velocities are being saved at a central server for managing traffic. This data is then used to build a map of current traffic. While this helps in preventing the traffic and does congestion analysis, the underlying problem is that there are less number of cars that are equipped with GPS. Machine learning in such scenarios helps to estimate the regions where congestion can be found on the basis of daily experiences.

3. Online Customer Support

A number of websites nowadays offer the option to chat with customer support representative while they are navigating within the site. However, not every website has a live executive to answer your queries. In most of the cases, you talk to a chatbot. These bots tend to extract information from the website and present it to the customers. Meanwhile, the chatbots advances with time. They tend to understand the user queries better and serve them with better answers, which is possible due to its machine learning algorithms.

4. Product Recommendations

You shopped for a product online few days back and then you keep receiving emails for shopping suggestions. If not this, then you might have noticed that the shopping website or the app recommends you some items that somehow matches with your taste. Certainly, this refines the shopping experience but did

you know that it's machine learning doing the magic for you? On the basis of your behaviour with the website/app, past purchases, items liked or added to cart, brand preferences etc., the product recommendations are made.

5. Online Fraud Detection

Machine learning is proving its potential to make cyberspace a secure place and tracking monetary frauds online is one of its examples. For example: Paypal is using ML for protection against money laundering. The company uses a set of tools that helps them to compare millions of transactions taking place and distinguish between legitimate or illegitimate transactions taking place between the buyers and sellers.

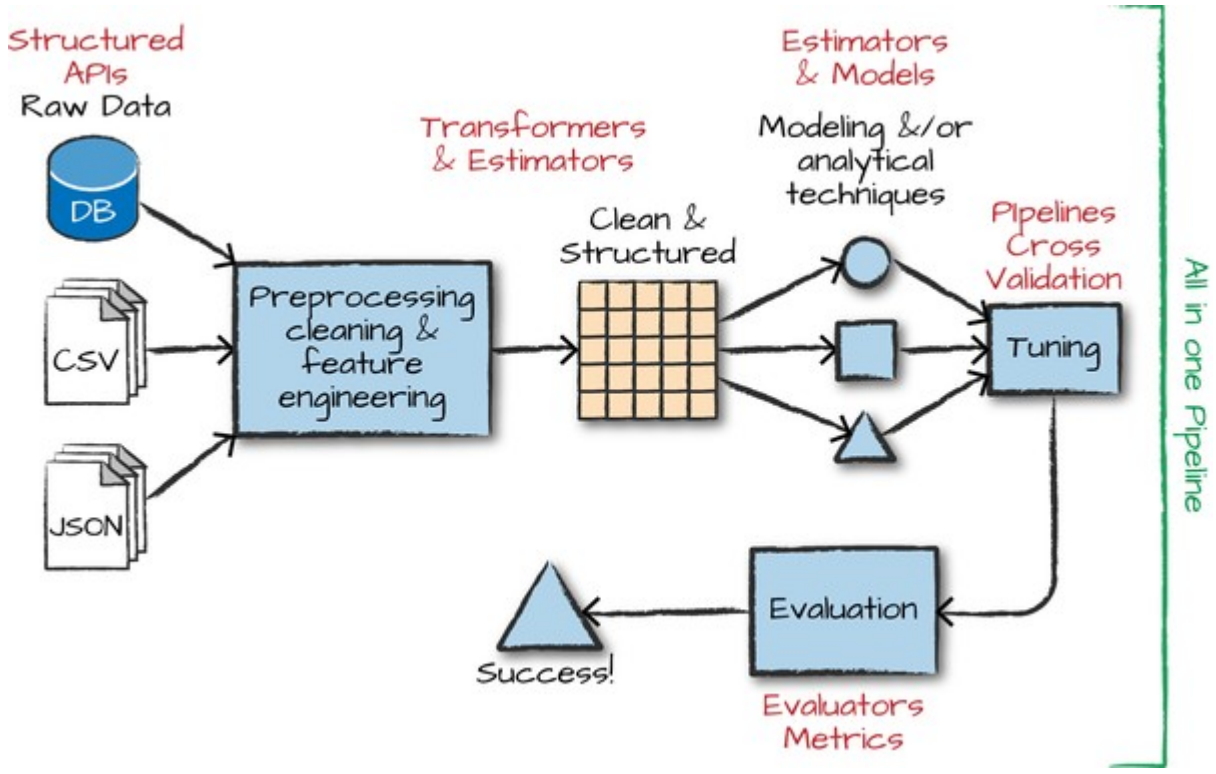
1.6 Deep Learning(DL) Life cycle Applications

Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network.

How Deep Learning Works

Deep learning has evolved hand-in-hand with the digital era, which has brought about an explosion of data in all forms and from every region of the world. This data, known simply as big data, is drawn from sources like social media, internet search engines, e-commerce platforms, and online cinemas, among others. This enormous amount of data is readily accessible and can be shared through fintech applications like cloud computing.

However, the data, which normally is unstructured, is so vast that it could take decades for humans to comprehend it and extract relevant information. Companies realize the incredible potential that can result from unraveling this wealth of information and are increasingly adapting to AI systems for automated support.



Deep Learning applications

1. Self-driving cars

Companies building these types of driver-assistance services, as well as full-blown self-driving cars like Google's, need to teach a computer how to take over key parts (or all) of driving using digital sensor systems instead of a human's senses. To do that companies generally start out by training algorithms using a large amount of data.

You can think of it how a child learns through constant experiences and replication. These new services could provide unexpected business models for companies.

2. Deep Learning in Healthcare

Breast or Skin-Cancer diagnostics? Mobile and Monitoring Apps? or prediction and personalised medicine on the basis of Biobank-data? AI is completely reshaping life sciences, medicine, and healthcare as an industry. Innovations in AI are advancing the future of precision medicine and population health management in unbelievable ways. Computer-aided detection, quantitative imaging, decision support tools and computer-aided diagnosis will play a big role in years to come.

3. Voice Search & Voice-Activated Assistants

One of the most popular usage areas of deep learning is voice search & voice-activated intelligent assistants. With the big tech giants have already made significant investments in this area, voice-activated assistants can be found on nearly every smartphone. Apple's Siri is on the market since October 2011. Google Now, the voice-activated assistant for Android, was launched less than a year after Siri. The newest of the voice-activated intelligent assistants is Microsoft Cortana.

4. Automatic Machine Translation

This is a task where given words, phrase or sentence in one language, automatically translate it into another language.

Automatic machine translation has been around for a long time, but deep learning is achieving top results in two specific areas:

- Automatic Translation of Text
- Automatic Translation of Images

Text translation can be performed without any pre-processing of the sequence, allowing the algorithm to learn the dependencies between words and their mapping to a new language.

5. Automatic Text Generation

This is an interesting task, where a corpus of text is learned and from this model new text is generated, word-by-word or character-by-character.

The model is capable of learning how to spell, punctuate, form sentences and even capture the style of the text in the corpus. Large recurrent neural networks are used to learn the relationship between items in the sequences of input strings and then generate text.

Installation guide for R and RStudio

Step 1 – Install R

1. Download the R installer from https://cran.r-project.org/

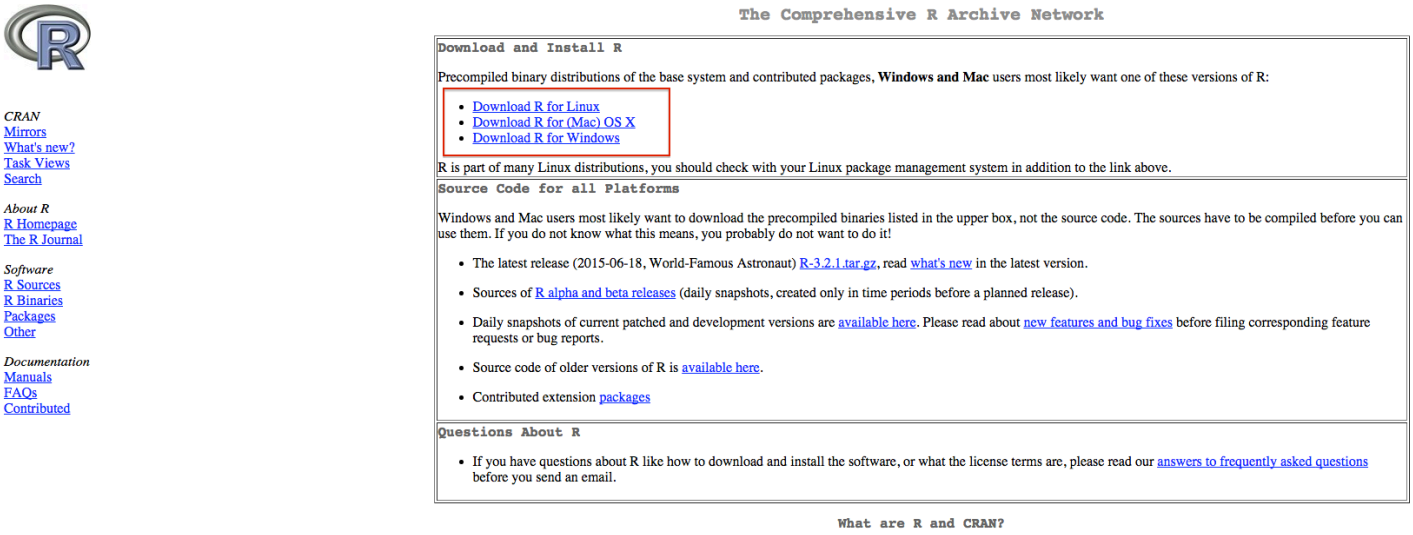


Figure 1. Screenshot of http://cran.csiro.au/

2. Run the installer. Default settings are fine. If you do not have admin rights on your laptop, then **ask you local IT support**. In that case, it is important that you also ask them to give you full permissions to the R directories. Without this, you will not be able to install additional packages later

Step 2 – Install RStudio

1. Download RStudio: https://www.rstudio.com/products/rstudio/download/

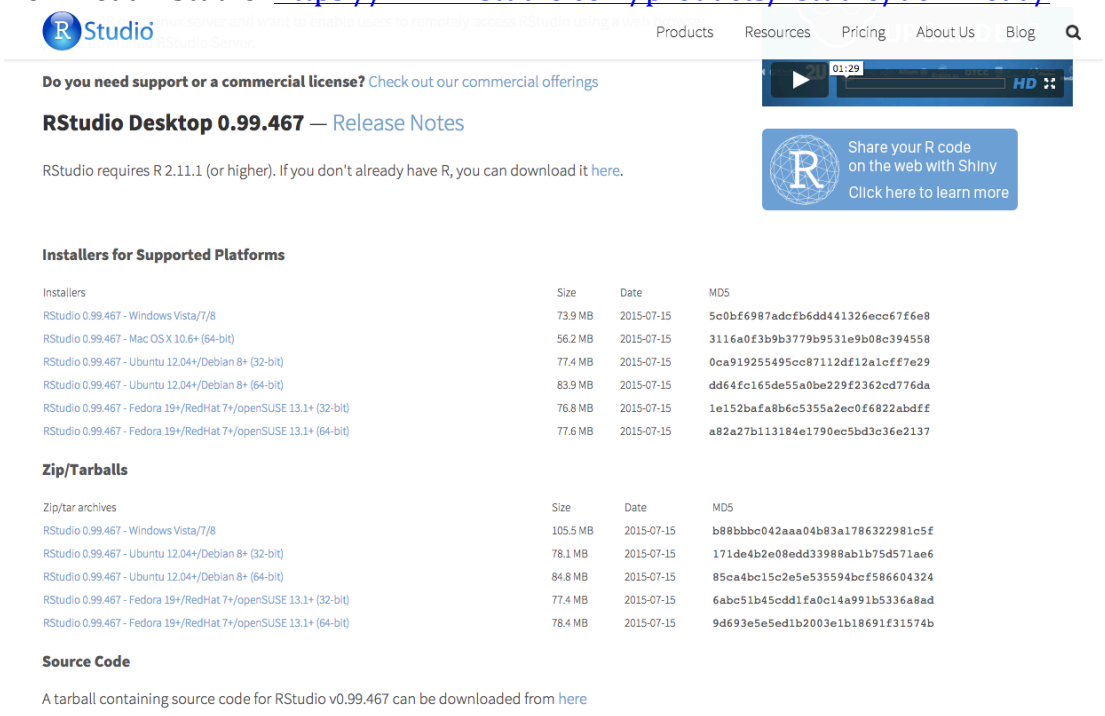


Figure 2. Download RStudio on https://www.rstudio.com/products/rstudio/download/

2. Once the installation of R has completed successfully (and not before), run the RStudio installer.

3. If you do not have administrative rights on your laptop, step 2 may fail. Ask your IT Support or download a pre-built zip archive of RStudio which doesn't need installing. The link for this is towards the bottom of the download page, highlighted in Image 2.

    a. Download the appropriate archive for your system (Windows/Linux only – the Mac version can be installed into your personal "Applications" folder without admin rights).

    b. Double clicking on the zip archive should automatically unpack it on most Windows machines.

Step 3 – Check that R and RStudio are working

1. Open RStudio. It should open a window that looks similar to image 3 below.

2. In the left hand window, by the '>'sign, type '4+5'(without the quotes) and hit enter. An output line reading '[1] 9' should appear. This means that R and RStudio are working.

3. If this is not successful, contact us or your local IT support for further advice

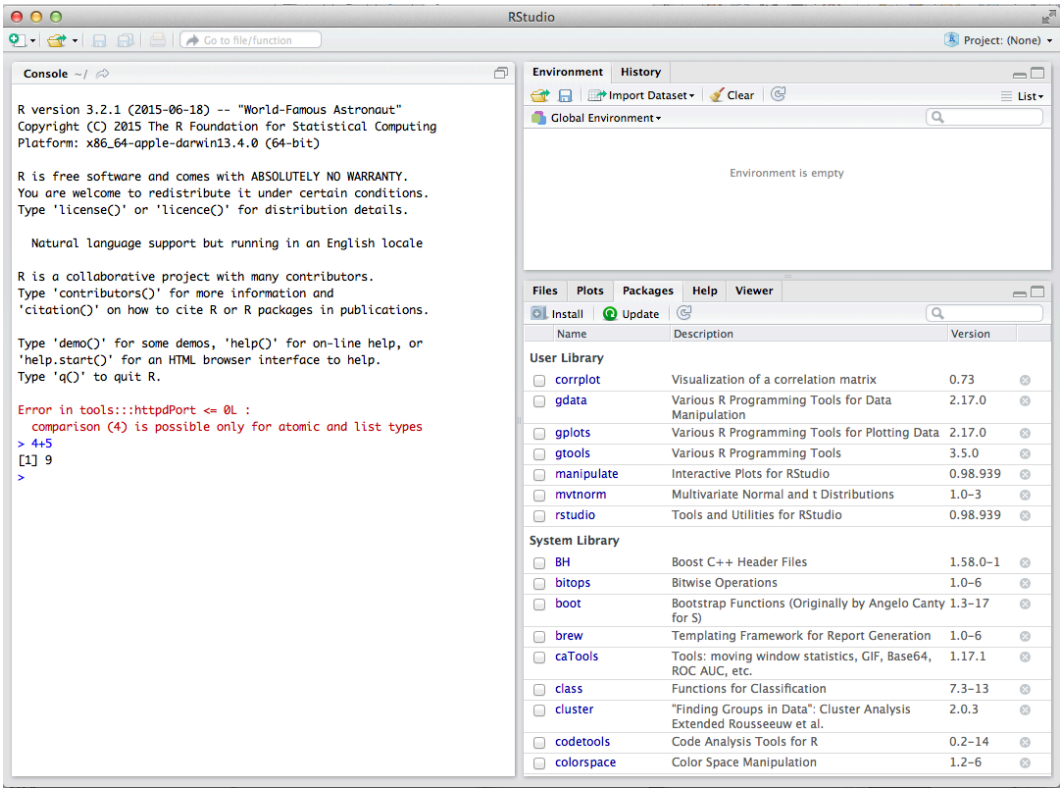


Figure 3. Running R with RStudio

Step 4 – Install R packages required for the workshop

1. Click on the tab ' Packages' then 'Install' as shown in Image 4. Or Tools -> Install packages.
2. **Install the following packages**: `mixOmics` **`version 6.1.0`**`, mvtnorm, RColorBrewer, corrplot, igraph` (see Image 4). **For apple mac users,** if you are unable to install the mixOmics imported library rgl, you will need to install the XQuartz software first https://www.xquartz.org/

3. Check that the packages are installed by typing '`library(mixOmics)`' (without the quotes) in the prompt and press enter (see Image 5).
4. Then type '`sessionInfo()`' and check that mixOmics version 6.1.0 has been installed (image 6).

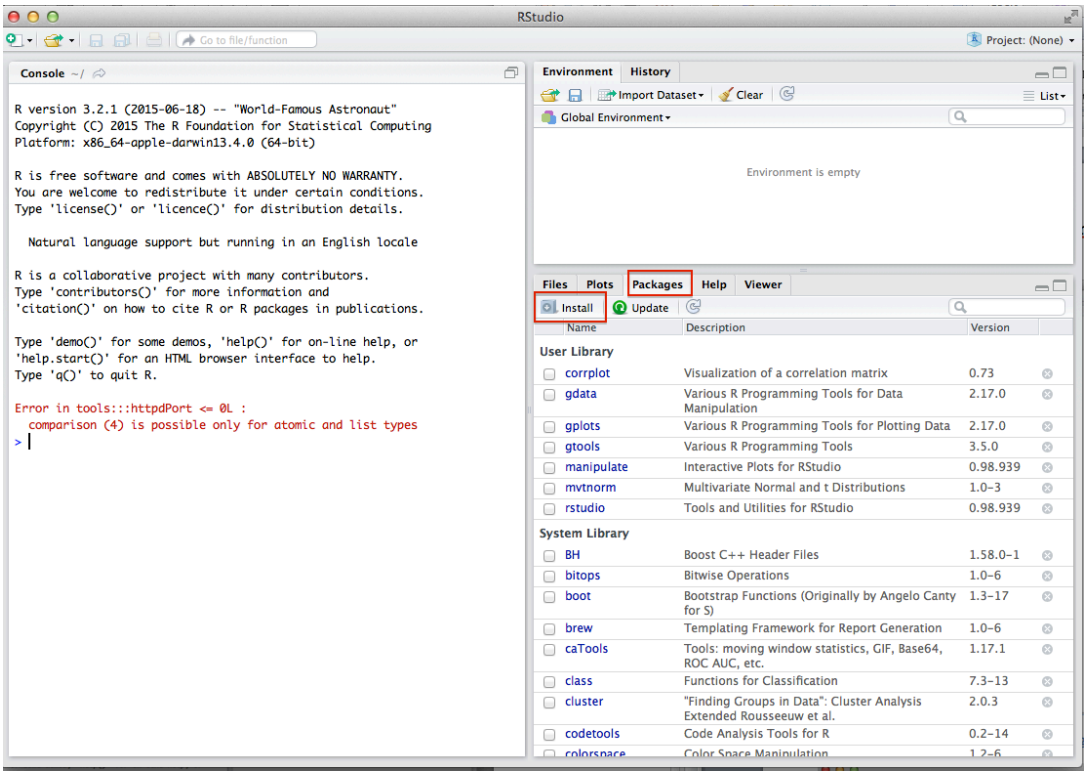


Figure 4. Click on Install to install R packages.

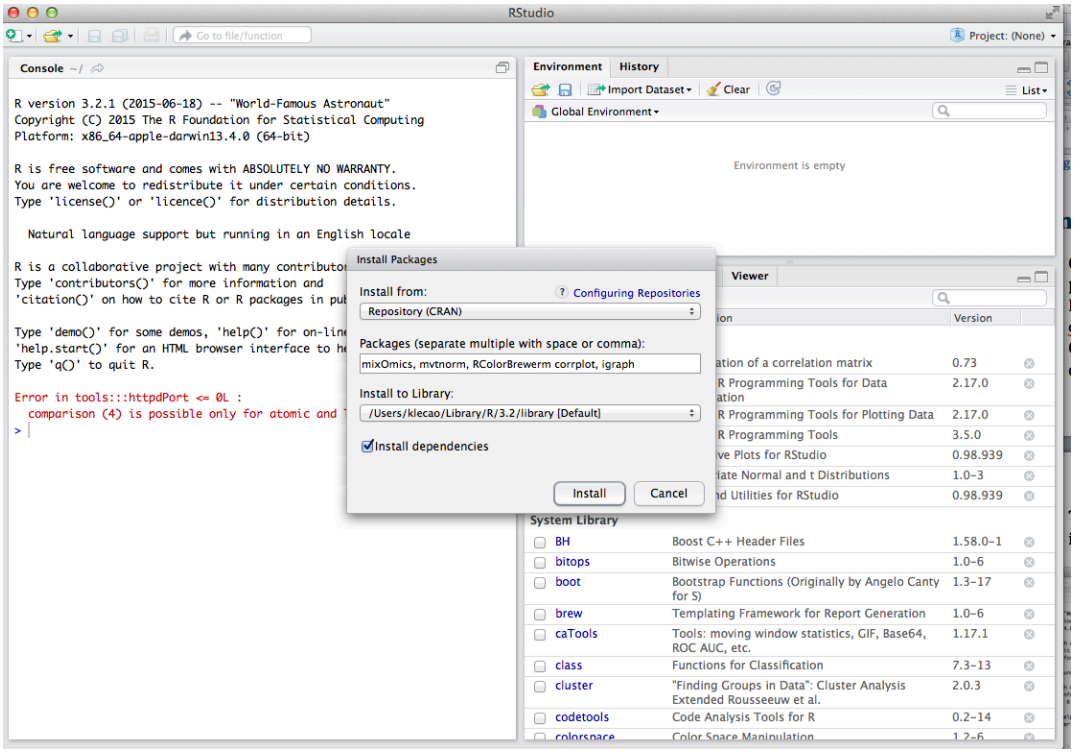


Figure 5. Specify the list of packages to be installed

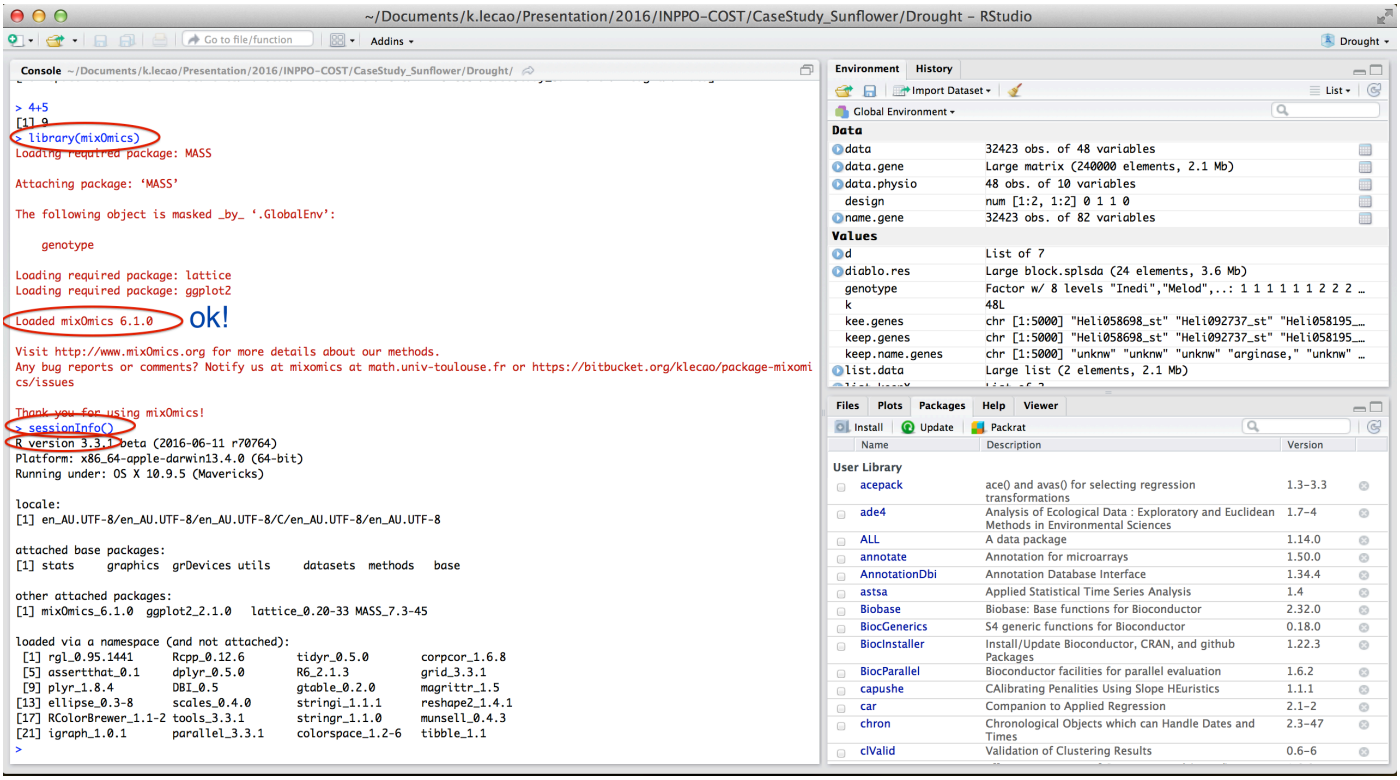


Figure 6. Check that the package mixOmics is installed and has the version 6.1.0.

That's it, let's get started!

2 Introduction to R

2.1 What is R?

R is a programming language developed by Ross Ihaka and Robert Gentleman in 1993
R is has an extensive catalog of statistical and graphical methods. t includes machine learning algorithm, linear regression, time series, statistical inference to name a few. Most of the R libraries are written in R, but for heavy computational task, C, C++ and Fortran codes are preferred.

R is not only entrusted by academic, but many large companies also use R programming language, including Uber, Google, Airbnb, Facebook and so on. Data analysis with R is done in a series of steps; programming, transforming, discovering, modeling and communicate the results

2.2 Why R for Data Science

R is simple to understand.
R has lot's of predefined libraries that help to make a data science model
R is provide rich graphical representation of data (**statistical** data that are **representation**)
R is interpreter language means it fast enough
R is provided console we can run program like a command

R is cross platform (Windows, Mac , Linux)

2.2 Variable and datatype in R

A variable provides us with named storage that our programs can manipulate. A variable in R can store an atomic vector, group of atomic vectors or a combination of many Robjects. A valid variable name consists of letters, numbers and the dot or underline characters. The variable name starts with a letter or the dot not followed by a number

Variable Name	Validity	Reason
var_name2	Valid	Has letters, numbers, dot and underscore
var_name%	Invalid	Has the character '%'. Only dot(.) and underscore allowed.
2var_name	Invalid	Starts with a number
.var_name, var.name	Valid	Can start with a dot(.) but the dot(.)should not be followed by a number.
.2var_name	Invalid	The starting dot is followed by a number making it invalid.

You can use the ← character to assign a variable, note how it kind of looks like an arrow pointing from the object to the variable name.

```r
# Use hashtags for comments
variable.name <- 100

# Let's see the variable!
print(variable.name)
```

Variable Assignment

The variables can be assigned values using leftward, rightward and equal to operator. The values of the variables can be printed using **print()** or **cat()** function. The **cat()** function combines multiple items into a continuous print output.

```r
# Assignment using equal operator.
var.1 = c(0,1,2,3)

# Assignment using leftward operator.
var.2 <- c("learn","R")

# Assignment using rightward operator.
c(TRUE,1) -> var.3

print(var.1)
cat ("var.1 is ", var.1 ,"\n")
cat ("var.2 is ", var.2 ,"\n")
cat ("var.3 is ", var.3 ,"\n")
```

Output

```
[1] 0 1 2 3
var.1 is  0 1 2 3
```

var.2 is learn R
var.3 is 1 1

2.4 Data Type of a Variable

In R, a variable itself is not declared of any data type, rather it gets the data type of the R - object assigned to it. So R is called a dynamically typed language, which means that we can change a variable's data type of the same variable again and again when using it in a program.

```
var_x <- "Hello"
cat("The class of var_x is ",class(var_x),"\n")

var_x <- 34.5
cat("  Now the class of var_x is ",class(var_x),"\n")

var_x <- 27L
cat("   Next the class of var_x becomes ",class(var_x),"\n")
```

OutPut

```
The class of var_x is  character
Now the class of var_x is  numeric
Next the class of var_x becomes  integer
```

Finding Variables

To know all the variables currently available in the workspace we use the **ls()** function. Also the ls() function can use patterns to match the variable names.

print(ls())

[1] "my var" "my_new_var" "my_var" "var.1"

Note – It is a sample output depending on what variables are declared in your environment. The ls() function can use patterns to match the variable names.

We can use variables together and work with them, for example:

```
bank.account <- 100

deposit <- 10

bank.account <- bank.account + deposit
```

2.5 Operators :

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. R language is rich in built-in operators and provides following types of operators.

Types of Operators in R

We have the following types of operators in R programming –

1) Arithmetic Operators

2) Relational Operators

3) Logical Operators

4) Assignment Operators

5) Miscellaneous Operators

Arithmetic Operators

Following table shows the arithmetic operators supported by R language. The operators act on each element of the vector.

Operator	Description	Example
+	Adds two vectors	v <- c(2 , 5.5 , 6) t <- c(8, 3, 4) print(v+t) Output [1] 10.0 8.5 10.0
-	Subtracts second vector from the first	v <- c(2 , 5.5 ,6) t <- c(8, 3, 4) print(v-t) output [1] -6.0 2.5 2.0
*	Multiplies both vectors	v <- c(2,5.5,6) t <- c(8, 3, 4) print(v*t) output [1] 16.0 16.5 24.0
/	Divide the first vector with the second	v <- c(2,5.5,6) t <- c(8, 3, 4) print(v/t)

		output [1] 0.250000 1.833333 1.500000
%%	Give the remainder of the first vector with the second	v <- c(2,5.5,6) t <- c(8, 3, 4) print(v%%t) output [1] 2.0 2.5 2.0
^	The first vector raised to the exponent of second vector	v <- c(2,5.5,6) t <- c(8, 3, 4) print(v^t) output [1] 256.000 166.375 1296.000

Relational Operators

Operator	Description	Example
>	Checks if each element of the first vector is greater than the corresponding element of the second vector.	v <- c(2, 5.5 , 6,9) t <- c(8, 2.5 , 14,9) print(v>t) output [1] FALSE TRUE FALSE FALSE
<	Checks if each element of the first vector is less than the corresponding element of the second vector.	v <- c(2,5.5,6,9) t <- c(8,2.5,14,9) print(v < t) output [1] TRUE FALSE TRUE FALSE
==	Checks if each element of	v <- c(2 , 5.5, 6, 9)

	the first vector is equal to the corresponding element of the second vector.	t <- c(8 , 2.5 , 14,9) print(v == t) output [1] FALSE FALSE FALSE TRUE
<=	Checks if each element of the first vector is less than or equal to the corresponding element of the second vector.	v <- c(2,5.5,6,9) t <- c(8,2.5,14,9) print(v<=t) output [1] TRUE FALSE TRUE TRUE
>=	Checks if each element of the first vector is greater than or equal to the corresponding element of the second vector.	v <- c(2 ,5.5 , 6 , 9) t <- c(8,2.5,14,9) print(v>=t) output [1] FALSE TRUE FALSE TRUE
!=	Checks if each element of the first vector is unequal to the corresponding element of the second vector	v <- c(2,5.5,6,9) t <- c(8,2.5,14,9) print(v!=t) [1] TRUE TRUE TRUE FALS

Assignment Operators:

Operator	Description	Example
<– or =	Called Left Assignment assign a value to variable	v1 <- c(3,1,TRUE,2+3) v2 <<- c(3,1,TRUE,2+3) v3 = c(3,1,TRUE,2+3) print(v1)

or <<–		print(v2) print(v3) output: [1] 3 1 1 5 [1] 3 1 1 5 [1] 3 1 1 5
-> or ->>	Called Right Assignment assign value to variable	c(3,1,TRUE,2+3i) -> v1 c(3,1,TRUE,2+3i) ->> v2 print(v1) print(v2) output: [1] 3+0i 1+0i 1+0i 2+3i [1] 3+0i 1+0i 1+0i 2+3i

Logical operator:

Operator	Description	Example
&	It is called Element-wise Logical AND operator. It combines each element of the first vector with the	v <- c(3,1,TRUE,FALSE) t <- c(4,1,FALSE,TRUE) print(v&t)

	corresponding element of the second vector and gives a output TRUE if both the elements are TRUE	output: [1] TRUE TRUE FALSE FALSE
|	It is called Element-wise Logical OR operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if one the elements is TRUE.	v <- c(12,0,TRUE,FALSE) t <- c(4,1,FALSE,FALSE) print(v|t) output: [1] TRUE TRUE TRUE FALSE
!	It is called Logical NOT operator. Takes each element of the vector and gives the opposite logical value.	v <- c(3,0,TRUE,2+2i) print(!v) output: [1] FALSE TRUE FALSE FALSE

The logical operator && and || considers only the first element of the vectors and give a vector of single element as output.

i

Operator	Description	Example
&&	Called Logical AND operator. Takes first element of both the vectors and gives the TRUE only if both are TRUE.	Z ← 3 (z > 2) && (z < 5) output: [1] TRUE

||	Called Logical OR operator. Takes first element of both the vectors and gives the TRUE if one of them is TRUE.	Z ← 1 (z > 2) & (z < 5) (z < 2) & (z > 5) output: [1] FALSE [2] TRUE

Miscellaneous Operators:

These operators are used to for specific purpose and not general mathematical or logical computation.

Operator	Description	Example
:	Colon operator. It creates the series of numbers in sequence for a vector.	v <- 2:8 print(v) output [1] 2 3 4 5 6 7 8
%in%	This operator is used to identify if an element belongs to a vector.	v1 <- 8 v2 <- 12 t <- 1:10 print(v1 %in% t) print(v2 %in% t) output: [1] TRUE [1] FALSE
%*%	This operator is used to multiply a matrix with its transpose.	M = matrix(c(2,6,5,1,10,4), nrow = 2,ncol = 3,byrow = TRUE) t = M %*% t(M)

		print(t) output: [,1] [,2] [1,] 65 82 [2,] 82 117

Variables

Let's go over how to assign variables in R.

You can use the **<-** character to assign a variable, note how it kind of looks like an arrow pointing from the object to the variable name.

A variable provides us with named storage that our programs can manipulate. A variable in R can store an atomic vector, group of atomic vectors or a combination of many Robjects. A valid variable name consists of letters, numbers and the dot or underline characters. The variable name starts with a letter or the dot not followed by a number

Some variable name valid or invalid

Variable Name	Validity	Reasone
var_name2	Valid	Has letters, numbers, dot and underscore
var_name%	Invalid	Has the character '%'. Only dot(.) and underscore allowed.
.var_name, var.name	Valid	Can start with a dot(.) but the dot(.)should not be followed by a number.
.2var_name	Invalid	The starting dot is followed by a number making it invalid.

Variable Assignment

The variables can be assigned values using leftward, rightward and equal to operator. The values of the variables can be printed using print() or cat() function. The cat() function combines multiple items into a continuous print output.

In [1]:

```
# Assignment using leftward operator.
variable.name <- 100
```

In [1]:

```
# Assignment using equal operator
var  =  100
```

In [1]:

```
# Assignment using rightward operator
var1  ->  100
```

In [2]:

```
# Let's see the output
print(variable.name)
print(var)
print(var1)
```

Out[2]:

```
100
100
100
```

Working with variables

We can use variables together and work with them, for example:

In [3]:

```
bank.account <- 100
```

In [5]:

```
deposit <- 10
```

In [6]:

```
bank.account <- bank.account + deposit
```

In [7]:

```
bank.account
```

Out[7]:

110

That's all for now, let's go learn about the basic data types!

R Data Types

Let's discuss some basic data types in R.

Numerics

Decimal (floating point values) are part of the numeric class in R

In [1]:

```
n <- 2.2
```

Integers

Natural (whole) numbers are known as integers and are also part of the numeric class

In [3]:

```
i <- 5
```

Logical

Boolean values (True and False) are part of the logical class. In R these are written in All Caps.

In [5]:

```
t <- TRUE
f <- FALSE
```

In [6]:

```
t
```

Out[6]:

TRUE

In [7]:

```
f
```

Out[7]:

FALSE

Characters

Text/string values are known as characters in R. You use quotation marks to create a text character string:

In [8]:

```
char <- "Hello World!"
```

In [15]:

```
char
```

Out[15]:

'Hello World!'

In [16]:

```
# Also single quotes
c <- 'Single Quote Char'
```

In [17]:

```
c
```

Out[17]:

'Single Quote Char'

Checking Data Type Classes

You can use the class() function to check the data type of a variable:

In [10]:

```
class(t)
```

Out[10]:

'logical'

In [11]:

```
class(f)
```

Out[11]:

'logical'

In [18]:

```
class(char)
```

Out[18]:

'character'

In [19]:

```
class(c)
```

Out[19]:

'character'

character

In [13]:

```
class(n)
```

Out[13]:

'numeric'

In [14]:

```
class(i)
```

Out[14]:

'numeric'

Those are some of the basic data types in R. Next we will learn about one of the main data building blocks of R, the vector!

Arithmetic with R

Addition

In [5]:

```
1+2
```

Out[5]:

3

Subtraction

In [7]:

```
5-3
```

Out[7]:

2

Division

In [8]:

```
1/2
```

Out[8]:

0.5

Exponents

In [9]:

```
2^3
```

Out[9]:

8

Modulo

In [11]:

```
5 %% 2
```

Out[11]:

1

Order of Operations

In [12]:

```
(100 * 2) + (50 / 2)
```

Out[12]:

225

That's it for the basics!

Comparison Operators

In R we can use comparison operators to compare variables and return logical values. Let's see some relatively self-explanatory examples:

Greater Than

In [3]:

```
5 > 6
```

Out[3]:

FALSE

In [7]:

```
6 > 5
```

Out[7]:

TRUE

We can also do element by element comparisons for two vectors:

In [5]:

```
v1 <- c(1,2,3)
v2 <- c(10,20,30)
```

In [6]:

```
v1 < v2
```

Out[6]:

TRUE TRUE TRUE

Greater Than or Equal to

In [8]:

```
6 >= 6
```

Out[8]:

TRUE

In [10]:

```
6 >= 5
```

Out[10]:

TRUE

In [11]:

```
6 >= 7
```

Out[11]:

FALSE

Less Than and Less than or Equal To

In [12]:

```
3 < 2
```

Out[12]:

FALSE

In [13]:

```
2 <= 2
```

Out[13]:

TRUE

Be very careful with comparison operators and negative numbers! Use spacing to keep things clear. An example of a dangerous situation:

In [14]:

```
var <- 1
```

In [15]:

```
var
```

Out[15]:

1

In [16]:

```
# Comparing var less than negative 2
var < -2
```

Out[16]:

FALSE

In [17]:

```
# Accidentally reassigning var!
var <- 2
```

In [18]:

```
var
```

Out[18]:

2

Keep syntax highlighting in mind to help you avoid this mistake!

Not Equal

In [19]:

```
5 != 2
```

Out[19]:

TRUE

In [26]:

```
5 != 5
```

Out[26]:

FALSE

Equal

In [27]:

```
5 == 5
```

Out[27]:

TRUE

In [28]:

```
2 == 3
```

Out[28]:

FALSE

Vector Comparisons

We can apply a comparison of a single number to an entire vector, for example:

In [29]:

```
v <- c(1,2,3,4,5)
```

In [30]:

```
v < 2
```

Out[30]:

TRUE FALSE FALSE FALSE FALSE

In [31]:

```
v == 3
```

Out[31]:

FALSE FALSE TRUE FALSE FALSE

Creating a Matrix

Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout. They contain elements of the same atomic types. Though we can create a matrix containing only characters or only logical values, they are not of much use. We use matrices containing numeric elements to be used in mathematical calculations. A Matrix is created using the matrix() function.

Before we talk about the Matrix, we should show a quick tip for quickly creating sequential numeric vectors, you can use the colon notation from slicing to create sequential vectors:

In [6]:

```
1:10
```

Out[6]:

```
 1  2  3  4  5  6  7  8  9  10
```

In [8]:

```
v <- 1:10
v
```

Out[8]:

```
 1  2  3  4  5  6  7  8  9  10
```

Great! Now, to create a matrix in R, you use the **matrix()** function. We can pass in a vector into the matrix:

In [10]:

```
matrix(v)
```

Out[10]:

1
2
3
4
5
6
7
8
9
10

Notice how the output is displayed. Here we have a two-dimensional matrix which is 10 rows by 1 column. Now what if we want to specify the number of rows? We can pass the parameter/argument into the matrix function called **nrow** which stands for number of rows:

In [13]:

```
matrix(v,nrow=2)
```

Out[13]:

1	3	5	7	9
2	4	6	8	10

Now we have a 2 by 5 matrix. Notice that the nrow argument allows this to happen. But how do we decide the fill order? We could have filed columns first (as we did above) or filled out the rows first insequential order. The **byrow** argument allows you to specify whether or not you want to fill out the matrix by rows or by columns. For example:

In [11]:

```
matrix(1:12,byrow = FALSE,nrow=4)
```

Out[11]:

1	5	9
2	6	10
3	7	11
4	8	12

In [14]:

```
matrix(1:12, byrow=TRUE, nrow=4)
```

Out[14]:

1	2	3
4	5	6
7	8	9
10	11	12

In [5]:

```
v
```

Out[5]:

```
 1  2  3  4  5  6  7  8  9  10  11  12
```

Creating Matrices from Vectors

We can combine vectors to later input them into a matrix. For example imagine the following vectors below of stock prices:

In [15]:

```
# not real prices
goog <- c(450,451,452,445,468)
msft <- c(230,231,232,236,228)
```

In [16]:

```
stocks <- c(goog,msft)
```

In [20]:

```
stock.matrix <- matrix(stocks,byrow=TRUE,nrow=2)
```

In [21]:

```
stock.matrix
```

Out[21]:

450	451	452	445	468

230	231	232	236	228

Naming Matrices

Now that we have our matrix, it would be nice to name the rows and columns for reference. We can do this similarly to the **names()** function for vectors, but in this case we define **colnames()** and **rownames()**. So let's name our stock matrix:

In [22]:

```
days <- c('Mon','Tue','Wed','Thu','Fri')
st.names <- c('GOOG','MSFT')
```

In [23]:

```
colnames(stock.matrix) <- days
rownames(stock.matrix) <- st.names
```

In [24]:

```
stock.matrix
```

Out[24]:

	Mon	Tue	Wed	Thu	Fri
GOOG	450	451	452	445	468
MSFT	230	231	232	236	228

Matrix Arithmetic

We can perform element by element mathematical operations on a matrix with a scalar (single number) just like we could with vectors. Let's see some quick examples:

In [3]:

```
mat <- matrix(1:50,byrow=TRUE,nrow=5)
mat
```

Out[3]:

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

In [2]:

```
# Multiplication
2*mat
```

Out[2]:

2	4	6	8	10	12	14	16	18	20
22	24	26	28	30	32	34	36	38	40
42	44	46	48	50	52	54	56	58	60
62	64	66	68	70	72	74	76	78	80
82	84	86	88	90	92	94	96	98	100

In [4]:

```
# Division (reciprocal)
1/mat
```

Out[4]:

1.0000000	0.5000000	0.3333333	0.2500000	0.2000000	0.1666667	0.1428571	0.1250000	0.1111111	0.1000000
0.09090909	0.08333333	0.07692308	0.07142857	0.06666667	0.06250000	0.05882353	0.05555556	0.05263158	0.05000000
0.04761905	0.04545455	0.04347826	0.04166667	0.04000000	0.03846154	0.03703704	0.03571429	0.03448276	0.03333333
0.03225806	0.03125000	0.03030303	0.02941176	0.02857143	0.02777778	0.02702703	0.02631579	0.02564103	0.02500000
0.02439024	0.02380952	0.02325581	0.02272727	0.02222222	0.02173913	0.02127660	0.02083333	0.02040816	0.02000000

In [5]:

```
# Division
mat/2
```

Out[5]:

0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
5.5	6.0	6.5	7.0	7.5	8.0	8.5	9.0	9.5	10.0
10.5	11.0	11.5	12.0	12.5	13.0	13.5	14.0	14.5	15.0
15.5	16.0	16.5	17.0	17.5	18.0	18.5	19.0	19.5	20.0

20.5	21.0	21.5	22.0	22.5	23.0	23.5	24.0	24.5	25.0

In [8]:

```
# Power
mat ^ 2
```

Out[8]:

1	4	9	16	25	36	49	64	81	100
121	144	169	196	225	256	289	324	361	400
441	484	529	576	625	676	729	784	841	900
961	1024	1089	1156	1225	1296	1369	1444	1521	1600
1681	1764	1849	1936	2025	2116	2209	2304	2401	2500

Comparison operators with matrices

We can similarly perform comparison operations across an entire matrix to return a matrix of logicals:

In [9]:

```
mat > 17
```

Out[9]:

FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE

Matrix Arithmetic with multiple matrices

We can use multiple matrices with arithmetic as well, for example:

In [13]:

```
mat + mat
```

Out[13]:

2	4	6	8	10	12	14	16	18	20
22	24	26	28	30	32	34	36	38	40
42	44	46	48	50	52	54	56	58	60
62	64	66	68	70	72	74	76	78	80
82	84	86	88	90	92	94	96	98	100

In [14]:

```
mat / mat
```

Out[14]:

1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

In [16]:

```
# Warning, big numbers!
mat ^ mat
```

Out[16]:

1	4	27	256	3125	46656	823543	16777216	38742
2.853117e+11	8.916100e+12	3.028751e+14	1.111201e+16	4.378939e+17	1.844674e+19	8.272403e+20	3.934641e+22	1.978
5.842587e+27	3.414279e+29	2.088047e+31	1.333736e+33	8.881784e+34	6.156120e+36	4.434265e+38	3.314552e+40	2.567
1.706917e+46	1.461502e+48	1.291100e+50	1.175664e+52	1.102507e+54	1.063874e+56	1.055513e+58	1.075912e+60	1.125
1.330878e+66	1.501309e+68	1.734377e+70	2.050774e+72	2.480636e+74	3.068035e+76	3.877924e+78	5.007021e+80	6.600

In [17]:

```
mat*mat
```

Out[17]:

1	4	9	16	25	36	49	64	81	100
121	144	169	196	225	256	289	324	361	400
441	484	529	576	625	676	729	784	841	900
961	1024	1089	1156	1225	1296	1369	1444	1521	1600
1681	1764	1849	1936	2025	2116	2209	2304	2401	2500

Matrix multiplication

A quick side note on matrix multiplications. You can perform arithmetic multiplication on an element by element basis using the * sign in R. You should note this is not the same as Matrix Multiplication. If you are familiar with the mathematics behind this topic and want to use R to perform true matrix multiplication, you can use the following:

In [18]:

```
mat2 <- matrix(1:9, nrow=3)
```

In [19]:

```
mat2
```

Out[19]:

1	4	7
2	5	8
3	6	9

In [20]:

```
# True Matrix Multiplication
mat2 %*% mat2
```

Out[20]:

30	66	102
36	81	126

Matrix Operations

Now that we've learned how to create a matrix, let's learn how to use functions and perform operations on it!

Run the following code to create the stock.matrix from earlier

In [18]:

```
# Prices
goog <- c(450,451,452,445,468)
msft <- c(230,231,232,236,228)

# Put vectors into matrix
stocks <- c(goog,msft)
stock.matrix <- matrix(stocks,byrow=TRUE,nrow=2)

# Name matrix
days <- c('Mon','Tue','Wed','Thu','Fri')
st.names <- c('GOOG','MSFT')
colnames(stock.matrix) <- days
rownames(stock.matrix) <- st.names

# Display
stock.matrix
```

Out[18]:

	Mon	Tue	Wed	Thu	Fri
GOOG	450	451	452	445	468
MSFT	230	231	232	236	228

We can perform functions across the columns and rows, such as colSum:

In [19]:

```
colSums(stock.matrix)
```

Out[19]:

Mon
680
Tue
682
Wed
684
Thu
681
Fri
696

In [20]:

```
# Doesn't really make sense for stocks, but just to show how it works
rowSums(stock.matrix)
```

Out[20]:

GOOG
2266
MSFT
1157

We can also do other mathematical operations, check this reference for all functions available.

In [21]:

```
rowMeans(stock.matrix)
```

Out[21]:

GOOG
453.2
MSFT
231.4

Binding columns and rows

Let's go ahead and see how we can add columns and rows to a matrix, we can use the **cbind()** to bind a new column, and **rbind()** to bind a new row. For example, let's bind a new row with Facebook stock:

In [22]:

```
FB <- c(111,112,113,120,145)
```

In [23]:

```
tech.stocks <- rbind(stock.matrix,FB)
```

In [24]:

```
tech.stocks
```

Out[24]:

	Mon	Tue	Wed	Thu	Fri
GOOG	450	451	452	445	468
MSFT	230	231	232	236	228
FB	111	112	113	120	145

Now let's add an average column to the matrix:

In [25]:

```
avg <- rowMeans(tech.stocks)
```

In [26]:

```
avg
```

Out[26]:

GOOG
453.2
MSFT
231.4
FB
120.2

In [27]:

```
tech.stocks <- cbind(tech.stocks,avg)
```

In [28]:

```
tech.stocks
```

Out[28]:

	Mon	Tue	Wed	Thu	Fri	avg
GOOG	450.0	451.0	452.0	445.0	468.0	453.2
MSFT	230.0	231.0	232.0	236.0	228.0	231.4
FB	111.0	112.0	113.0	120.0	145.0	120.2

Matrix Selection and Indexing

Just like with vectors, we use the square bracket notation to select elements from a matrix. Since we have two dimensions to work with, we'll use a comma to separate our indexing for each dimension.

So the syntax is then:

example.matrix[*rows*,*columns*]

where the index notation (e.g. 1:5) is put in place of the *rows* or *columns* . If either *rows* or *columns* is left blank, then we are selecting all the rows and columns.

Let's work through some examples:

In [1]:

```
mat <- matrix(1:50,byrow=TRUE,nrow=5)
```

In [2]:

```
mat
```

Out[2]:

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

In [3]:

```
# Grab first row
mat[1,]
```

Out[3]:

```
 1  2  3  4  5  6  7  8  9  10
```

In [4]:

```
#Grab first column
mat[,1]
```

Out[4]:

```
 1  11  21  31  41
```

In [6]:

```
# Grab first 3 rows
mat[1:3,]
```

Out[6]:

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

In [8]:

```
# Grab top left rectangle of:
# 1,2,3
# 11,12,13
#
mat[1:2,1:3]
```

Out[8]:

1	2	3
11	12	13

In [9]:

```
# Grab last two columns
mat[,9:10]
```

Out[9]:

9	10
19	20
29	30
39	40
49	50

In [14]:

```
# Grab a center square of:
# 15,16
# 25,26
mat[2:3,5:6]
```

Out[14]:

15	16
25	26

Vector Basics

Vectors are one of the key data structures in R which we wil be using. A vector is a 1 dimensional array that can hold character, numeric, or logical data elements.

We can create a vector by using the combine function **c()**. To use the function, we pass in the elements we want in the array, with each individual element separated by a comma.

Let's see some examples:

In [14]:

```
# Using c() to create a vector of numeric elements
nvec <- c(1,2,3,4,5)
```

In [15]:

```
class(nvec)
```

Out[15]:

'numeric'

In [16]:

```
# Vector of characters
cvec <- c('U','S','A')
```

In [17]:

```
class(cvec)
```

Out[17]:

'character'

In [18]:

```
lvec <- c(TRUE,FALSE)
```

In [19]:

```
lvec
```

Out[19]:

TRUE FALSE

In [20]:

```
class(lvec)
```

Out[20]:

'logical'

Note that we can't mix data types of the elements in an array, R will convert the other elements in the array to force everything to be of the same data type. Later on we will learn about the list data structure that can take on multiple data types!

Here's a quick example of what happens with arrays given different data types:

In [31]:

```
v <- c(FALSE,2)
```

In [32]:

```
v
```

Out[32]:

```
0  2
```

In [26]:

```
class(v)
```

Out[26]:

```
'numeric'
```

In [28]:

```
v <- c('A',1)
```

In [29]:

```
v
```

Out[29]:

```
'A'  '1'
```

In [30]:

```
class(v)
```

Out[30]:

```
'character'
```

Vector Names

We can use the **names()** function to assign names to each element in our vector. For example, imagine the folowing vector of a week of temperatures:

In [38]:

```
temps <- c(72,71,68,73,69,75,71)
```

In [39]:

```
temps
```

Out[39]:

```
72  71  68  73  69  75  71
```

We know we have 7 temperatures for 7 weekdays, but which temperature corresponds to which weekday? Does it start on Monday, Sunday, or another day of the week? This is where **names()** can be assigned in the following manner:

In [40]:

```
names(temps) <- c('Mon','Tue','Wed','Thu','Fri','Sat','Sun')
```

Now note what happens when we display the named vector:

In [41]:

```
temps
```

Out[41]:

Mon
72
Tue
71
Wed
68
Thu
73
Fri
69
Sat
75
Sun
71

We can now begin to see how the elements were assigned names! Depending on what IDE you are using, you may see the above displayed horizontally instead of vertically, that's totally fine!

We also don't have to rewrite the names vector over and over again, we can use simple use a variable name as a **names()** assignment, for example:

In [42]:

```
days <- c('Mon','Tue','Wed','Thu','Fri','Sat','Sun')
temps2 <- c(1,2,3,4,5,6,7)
names(temps2) <- days
```

In [43]:

```
temps2
```

Out[43]:

Mon
1
Tue
2
Wed
3
Thu
4
Fri
5
Sat
6
Sun
7

Vector Indexing and Slicing

You can use bracket notation to index and access individual elements from a vector:

In [11]:

```
v1 <- c(100,200,300)
v2 <- c('a','b','c')
```

In [12]:

```
v1
v2
```

Out[12]:

100 200 300

Out[12]:

'a' 'b' 'c'

Indexing works by using brackets and passing the index position of the element as a number. Keep in mind index starts at 1 (in some other programming languages indexing starts at 0).

In [13]:

```
# Grab second element
v1[2]
```

Out[13]:

200

In [14]:

```
v2[2]
```

Out[14]:

'b'

Multiple Indexing

We can grab multiple items from a vector by passing a vector of index positions inside the square brackets. For example:

In [17]:

```
v1[c(1,2)]
```

Out[17]:

100 200

In [19]:

```
v2[c(2,3)]
```

Out[19]:

'b' 'c'

In [20]:

```
v2[c(1,3)]
```

Out[20]:

'a' 'c'

Slicing

You can use a colon (:) to indicate a slice of a vector. The format is:

vector[start_index:stop_index]

and you will get that "slice" of the vector returned to you. For example:

In [21]:

```
v <- c(1,2,3,4,5,6,7,8,9,10)
```

In [22]:

```
v[2:4]
```

Out[22]:

2 3 4

In [23]:

```
v[7:10]
```

Out[23]:

7 8 9 10

Notice how the element st both the starting index and the stopping index are included.

Indexing with Names

We've previously seen how we can assign names to the elements in a vector, for example:

In [26]:

```
v <- c(1,2,3,4)
names(v) <- c('a','b','c','d')
```

We can use those names along with the indexing brackets to grab individual elements from the array!

In [27]:

```
v['a']
```

Out[27]:

a: 1

Or pass in a vector of names we want to grab:

In [28]:

```
# Notice how we can call out of order!
v[c('a','c','b')]
```

Out[28]:

a
1
c
3
b
2

Comparison Operators and Selection

As alluded to in the comparison operator lecture, we can use comparison operators to filter out elements from a vector. Sometimes this is referred to as boolean/logical masking, because you are creating a vector of logicals to filter out results you want. Let's see an example of this:

In [29]:

```
v
```

Out[29]:

a
1
b
2
c
3
d
4

In [30]:

```
v[v>2]
```

Out[30]:

c
3
d
4

Let's break this down to see how it works, we first get the vector v>2:

In [31]:

```
v>2
```

Out[31]:

a
FALSE
b
FALSE
c
TRUE
d
TRUE

Now we basically pass this vector of logicals through the brackets of the vector and only return true values at the matching index positions:

In [32]:

```
v[v>2]
```

Out[32]:

c
3
d
4

We could also assign names to these logical vectors and pass them as well, for example:

In [33]:

```
filter <- v>2
```

In [34]:

```
filter
```

Out[34]:

a
FALSE
b
FALSE
c
TRUE
d
TRUE

In [35]:

```
v[filter]
```

Out[35]:

c
3
d
4

Working with Vectors

We can perform basics arithmetic with vectors and operations will occur on an element by element basis, for example:

In [1]:

```
v1 <- c(1,2,3)
v2 <- c(5,6,7)
```

Adding Vectors

In [2]:

```
v1+v2
```

Out[2]:

```
6  8  10
```

Subtracting Vectors

In [3]:

```
v1-v1
```

Out[3]:

```
0  0  0
```

In [4]:

```
v1-v2
```

Out[4]:

```
-4  -4  -4
```

Multiplying Vectors

In [5]:

```
v1*v2
```

Out[5]:

```
5  12  21
```

Dividing Vectors

In [6]:

```
v1/v2
```

Out[6]:

```
0.2  0.333333333333333  0.428571428571429
```

Functions with Vectors

Functions with Vectors

Let's learn about some useful functions that we can use with vectors! A function will be in the form:

name_of_function(input)

later on we will learn about creating our own functions, but R comes with abunch of built-in functions that you will commonly use.

For example, if you want to sum all the elements in a numeric vector, you can use the **sum()** function. For example:

In [8]:

```
v1
```

Out[8]:

```
1 2 3
```

In [7]:

```
sum(v1)
```

Out[7]:

```
6
```

We can also check for things like the standard deviation, variance, maximum element, minimum element, product of elements:

In [13]:

```
v <- c(12,45,100,2)
```

In [14]:

```
# Standard Deviation
sd(v)
```

Out[14]:

```
44.1691823182937
```

In [18]:

```
#Variance
var(v)
```

Out[18]:

```
1950.91666666667
```

In [19]:

```
# Maximum Element
max(v)
```

Out[19]:

```
100
```

In [20]:

```
#Minimum Element
min(v)
```

Out[20]:

```
2
```

In [22]:

```
# Product of elements
prod(v1)
prod(v2)
```

Out[22]:

6

Out[22]:

210

if, else, else if Statements

Now it is time to finally start learning how we can program some sort of logic using R! Our first step in this learning journey for programming will be simple if, else, and else if statements.

Here is the syntax for an **if** statement in R:

```
if (condition){
    # Execute some code
}
```

So what does this actually mean if you've never seen an if statement before? It means that we can begin to apply some simple logic to our code. We say *if* some condition is *true* then execute the code inside of the curly brackets.

For example, let's say we have two variables, **hot** and **temp**. Imagine that **hot** starts off as FALSE and temp is some number in degrees. If the **temp** is greater than 80 than we want to assign **hot==TRUE**.

Let's see this in action:

In [3]:

```
hot <- FALSE
temp <- 60
```

In [5]:

```
if (temp > 80){
    hot <- TRUE

}
hot
```

Out[5]:

FALSE

In [10]:

```
# Reset temp
temp <- 100

if (temp > 80){
    hot <- TRUE

}

hot
```

Out[10]:

TRUE

Something to keep in mind is that you should format your code carefully so you can come back later on and easily read it! By convention we align the closing bracket with the **if** statement it refers to. However, because we use brackets we could be sloppy (not good!) and still have the code work out:

In [11]:

```
if( 1 == 1){        print('hi')}
```

```
[1] "hi"
```

In [13]:

```
if(1 ==   1)

{

    print('hi')

              }

# This works...but hard to read!
```

```
[1] "hi"
```

A good editor like RStudio will help you make sure everything is aligned well.

else

If we want to execute another block that occurs if the **if** statement is false, we can use an **else** statement to do this! It has the syntax:

```
if (condition) {
  # Code to execute if true
} else {
  # Code to execute if above was not true
}
```

Notice the alignment of the curly brackets and the use of the *else* keyword. Let's see it in action!

In [14]:

```
temp <- 30

if (temp > 90){
    print("Hot outside!")
} else{
    print("Its not too hot today!")
}
```

```
[1] "Its not too hot today!"
```

else if

What if we wanted more options to print out, rather than just two, the *if* and the *else*? This is where we can use the **else if** statement to add multiple condition checks, using **else** at the end to execute code if none of our conditions match up with and if or else if.

Let's see this in action!

In [15]:

```
temp <- 30

if (temp > 80){
    print("Hot outside!")
} else if(temp<80 & temp>50){
    print('Nice outside!')
} else if(temp <50 & temp > 32){
    print("Its cooler outside!")
} else{
    print("Its really cold outside!")
}
```

```
[1] "Its really cold outside!"
```

In [17]:

```
temp <- 75

if (temp > 80){
    print("Hot outside!")
```

```
} else if(temp<80 & temp>50){
    print('Nice outside!')
} else if(temp <50 & temp > 32){
    print("Its cooler outside!")
} else{
    print("Its really cold outside!")
}
```

```
[1] "Nice outside!"
```

Final Example

Let's see a final more elaborate example of **if,else, and else if** to close out our discussion! Let's imagine that we have a store with two items for sale: *ham* and *cheese* and we want an automated report to go to HQ depending on how many we sell:

In [19]:

```
# Items sold that day
ham <- 10
cheese <- 10

# Report to HQ
report <- 'blank'


if(ham >= 10 & cheese >= 10){
    report <- "Strong sales of both items"

}else if(ham == 0 & cheese == 0){
    report <- "Nothing sold!"
}else{
    report <- 'We had some sales'
}
print(report)
```

```
[1] "Strong sales of both items"
```

Logical Operators

Logical Operators will allow us to combine multiple comparison operators. The logical operators we will learn about are:

- AND - &
- OR - |
- NOT - !

The best way to explain these is to see some examples.

In [1]:

```
# Imagine the variable x
x <- 10
```

Now we want to know if 10 is less than 20 AND greater than 5:

In [2]:

```
x < 20
```

Out[2]:

TRUE

In [3]:

```
x > 5
```

Out[3]:

TRUE

In [4]:

```
x < 20 & x > 5
```

Out[4]:

TRUE

We can also add parenthesis for readability and to make sure the order of comparisons is what we expect:

In [5]:

```
(x < 20) & (x>5)
```

Out[5]:

TRUE

In [6]:

```
(x < 20) & (x>5) & (x == 10)
```

Out[6]:

TRUE

We can basically think of this as a series of Logical Boolean values, TRUE & TRUE & TRUE. We return a single TRUE if they are all TRUE. Let's see an example of this not being the case:

In [7]:

```
x==2 & x > 1
```

Out[7]:

FALSE

Returned FALSE because while x > 1 is TRUE, we need BOTH to be TRUE, thus the AND statement &. What if we only want one of them to be true? That's when we use OR |. For example:

In [8]:

```
x==2 | x > 1
```

Out[8]:

TRUE

Here we only need one *or* the other to be true!

In [10]:

```
x==1 | x==12
```

Out[10]:

FALSE

NOT !

You can think about NOT as reversing any logical value in front of it, basically asking, "Is this NOT true?" For example:

In [13]:

```
(10==1)
```

Out[13]:

FALSE

In [14]:

```
!(10==1)
```

Out[14]:

TRUE

In [16]:

```
# We can stack them (pretty uncommon, but possible)
!!(10==1)
```

Out[16]:

FALSE

Use Case Example

Here's a quick example of a real use case for these operators. Imagine the following data frame:

In [32]:

```
df <- mtcars
```

In [33]:

```
df
```

Out[33]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.62	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.32	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.44	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.46	20.22	1	0	3	1
Duster 360	14.3	8	360	245	3.21	3.57	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.19	20	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.15	22.9	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.44	18.3	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.44	18.9	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.07	17.4	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.73	17.6	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.78	18	0	0	3	3
Cadillac Fleetwood	10.4	8	472	205	2.93	5.25	17.98	0	0	3	4
Lincoln Continental	10.4	8	460	215	3	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.2	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.9	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.7	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318	150	2.76	3.52	16.87	0	0	3	2
AMC Javelin	15.2	8	304	150	3.15	3.435	17.3	0	0	3	2
Camaro Z28	13.3	8	350	245	3.73	3.84	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79	66	4.08	1.935	18.9	1	1	4	1
Porsche 914-2	26	4	120.3	91	4.43	2.14	16.7	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9	1	1	5	2
Ford Pantera L	15.8	8	351	264	4.22	3.17	14.5	0	1	5	4
Ferrari Dino	19.7	6	145	175	3.62	2.77	15.5	0	1	5	6
Maserati Bora	15	8	301	335	3.54	3.57	14.6	0	1	5	8
Volvo 142E	21.4	4	121	109	4.11	2.78	18.6	1	1	4	2

This shows some data for various car models (its built in to R). Let's grab models with at least 20 mpg:

In [39]:

```
df[df['mpg'] >= 20,] # Notice the use of indexing with the comma
# subset(df,mpg>=20) # Could also use subset
```

Out[39]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.62	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.32	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Merc 240D	24.4	4	146.7	62	3.69	3.19	20	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.15	22.9	1	0	4	2
Fiat 128	32.4	4	78.7	66	4.08	2.2	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.9	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.7	2.465	20.01	1	0	3	1
Fiat X1-9	27.3	4	79	66	4.08	1.935	18.9	1	1	4	1
Porsche 914-2	26	4	120.3	91	4.43	2.14	16.7	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9	1	1	5	2
Volvo 142E	21.4	4	121	109	4.11	2.78	18.6	1	1	4	2

Great! Now let's combine filters with logical operators! Let's grab rows with cars of at least 20mpg and over 100 hp.

In [41]:

```
df[(df['mpg'] >= 20) & (df['hp'] > 100),]
```

Out[41]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.62	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4	4
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9	1	1	5	2
Volvo 142E	21.4	4	121	109	4.11	2.78	18.6	1	1	4	2

Hopefully you know see how useful these type of logical operators can be!

Logical Operators with Vectors

We have two options when use logical operators, a comparison of the entire vectors element by element, or just a comparison of the first elements in the vectors, to make sure the output is a single Logical, don't worry too much about this right now, we will cover it in more depth later on.

In [50]:

```
tf <- c(TRUE,FALSE)
tt <- c(TRUE,TRUE)
ft <- c(FALSE, TRUE)
```

In [51]:

```
tt & tf
```

Out[51]:

```
TRUE  FALSE
```

In [55]:

```
tt | tf
```

Out[55]:

TRUE TRUE

To compare first elements use && or ||

In [56]:

```
ft && tt
```

Out[56]:

FALSE

In [57]:

```
tt && tf
```

Out[57]:

TRUE

In [58]:

```
tt || tf
```

Out[58]:

TRUE

In [59]:

```
tt || ft
```

Out[59]:

TRUE

while loops

while loops are a while to have your program continuously run some block of code until a condition is met (made TRUE). The syntax is:

```
while (condition){
    # Code executed here
    # while condition is true
}
```

A major concern when working with a while loop is to make sure that at some point the condition should become true, otherwise the while loop will go forever! Remember you can use Crtl-C to kill a process in R Studio!

Here's a real quick side note on printing variables along with strings:

In [6]:

```
print('Just a string')
```

```
[1] "Just a string"
```

In [7]:

```
var <- 'a variable'
cat('My variable is: ',var)
```

```
My variable is:  a variable
```

In [9]:

```
var <- 25
cat('My number is:',var)
```

```
My number is: 25
```

In [12]:

```
# Could also use:
print(paste0("Variable is: ", var))
```

```
[1] "Variable is: 25"
```

We'll be using the **cat** function in the next example which shows how a while loop works:

In [14]:

```
x <- 0

while(x < 10){

    cat('x is currently: ',x)
    print(' x is still less than 10, adding 1 to x')

    # add one to x
    x <- x+1
}
```

```
x is currently:  0[1] " x is still less than 10, adding 1 to x"
x is currently:  1[1] " x is still less than 10, adding 1 to x"
x is currently:  2[1] " x is still less than 10, adding 1 to x"
x is currently:  3[1] " x is still less than 10, adding 1 to x"
x is currently:  4[1] " x is still less than 10, adding 1 to x"
x is currently:  5[1] " x is still less than 10, adding 1 to x"
```

```
x is currently:  6[1] " x is still less than 10, adding 1 to x"
x is currently:  7[1] " x is still less than 10, adding 1 to x"
x is currently:  8[1] " x is still less than 10, adding 1 to x"
x is currently:  9[1] " x is still less than 10, adding 1 to x"
```

Let's add an if statement to this logic!

In [15]:

```
x <- 0

while(x < 10){

    cat('x is currently: ',x)
    print(' x is still less than 10, adding 1 to x')

    # add one to x
    x <- x+1
    if(x==10){
        print("x is equal to 10! Terminating loop")
    }
}
```

```
x is currently:  0[1] " x is still less than 10, adding 1 to x"
x is currently:  1[1] " x is still less than 10, adding 1 to x"
x is currently:  2[1] " x is still less than 10, adding 1 to x"
x is currently:  3[1] " x is still less than 10, adding 1 to x"
x is currently:  4[1] " x is still less than 10, adding 1 to x"
x is currently:  5[1] " x is still less than 10, adding 1 to x"
x is currently:  6[1] " x is still less than 10, adding 1 to x"
x is currently:  7[1] " x is still less than 10, adding 1 to x"
x is currently:  8[1] " x is still less than 10, adding 1 to x"
x is currently:  9[1] " x is still less than 10, adding 1 to x"
[1] "x is equal to 10! Terminating loop"
```

break

You can use **break** to break out of a loop. Previously we showed an if statement checking for 10, but this wasn't actually stopping the loop, the while condition check on the next run stopped the loop, let's show an example where we could use **break** to terminate the loop:

In [17]:

```
x <- 0

while(x < 10){

    cat('x is currently: ',x)
    print(' x is still less than 10, adding 1 to x')

    # add one to x
    x <- x+1
    if(x==10){
        print("x is equal to 10!")
        print("I will also print, woohoo!")
    }
}
```

```
x is currently:  0[1] " x is still less than 10, adding 1 to x"
x is currently:  1[1] " x is still less than 10, adding 1 to x"
x is currently:  2[1] " x is still less than 10, adding 1 to x"
x is currently:  3[1] " x is still less than 10, adding 1 to x"
x is currently:  4[1] " x is still less than 10, adding 1 to x"
x is currently:  5[1] " x is still less than 10, adding 1 to x"
x is currently:  6[1] " x is still less than 10, adding 1 to x"
x is currently:  7[1] " x is still less than 10, adding 1 to x"
x is currently:  8[1] " x is still less than 10, adding 1 to x"
x is currently:  9[1] " x is still less than 10, adding 1 to x"
[1] "x is equal to 10!"
[1] "I will also print, woohoo!"
```

In [18]:

```
x <- 0

while(x < 10){

    cat('x is currently: ',x)
    print(' x is still less than 10, adding 1 to x')

    # add one to x
    x <- x+1
    if(x==10){
        print("x is equal to 10!")
        break
        print("I will also print, woohoo!")
    }
}
```

```
x is currently:  0[1] " x is still less than 10, adding 1 to x"
x is currently:  1[1] " x is still less than 10, adding 1 to x"
x is currently:  2[1] " x is still less than 10, adding 1 to x"
x is currently:  3[1] " x is still less than 10, adding 1 to x"
x is currently:  4[1] " x is still less than 10, adding 1 to x"
x is currently:  5[1] " x is still less than 10, adding 1 to x"
x is currently:  6[1] " x is still less than 10, adding 1 to x"
x is currently:  7[1] " x is still less than 10, adding 1 to x"
x is currently:  8[1] " x is still less than 10, adding 1 to x"
x is currently:  9[1] " x is still less than 10, adding 1 to x"
[1] "x is equal to 10!"
```

for loops

A **for loop** allows us to iterate over an object (such as a vector) and we can then perform and execute blocks of codes *for* every loop we go through. The syntax for a for loop is:

```
for (temporary_variable in object){
    # Execute some code at every loop
}
```

Let's start off by showing how to use a for loop with a vector:

For loop over a vector

We can think of looping through a vector in two different ways, the first way would be to create a temporary variable with the use of the **in** keyword:

In [1]:

```
vec <- c(1,2,3,4,5)
```

In [2]:

```
for (temp_var in vec){
    print(temp_var)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

The other way would be to loop a numbered amount of times and then use indexing to continually grab from the vector:

In [3]:

```
for (i in 1:length(vec)){
    print(vec[i])
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

For loop over a list

We can do the same thing with a list:

In [4]:

```
li <- list(1,2,3,4,5)
```

In [8]:

```
for (temp_var in li){
    print(temp_var)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

In [10]:

```
for (i in 1:length(li)){
    print(li[[i]]) # Remember to use double brackets!
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

For loop with a matrix

We can similarly loop through each individual element in a matrix:

In [14]:

```
mat <- matrix(1:25,nrow=5)
mat
```

Out[14]:

1	6	11	16	21
2	7	12	17	22
3	8	13	18	23
4	9	14	19	24
5	10	15	20	25

In [15]:

```
for (num in mat){
    print(num)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
[1] 11
[1] 12
[1] 13
[1] 14
[1] 15
[1] 16
[1] 17
[1] 18
[1] 19
[1] 20
[1] 21
[1] 22
[1] 23
[1] 24
[1] 25
```

Nested for loops

We can nest for loops inside one another,however be careful when doing this, as every additional for loop nested inside another may cause a significant amount of additional time for your code to finish executing. For example:

In [28]:

```
for (row in 1:nrow(mat)){
    for (col in 1:ncol(mat)){
        print(paste('The element at row:',row,'and col:',col,'is',mat[row,col]))
    }
}
```

```
[1] "The element at row: 1 and col: 1 is 1"
[1] "The element at row: 1 and col: 2 is 6"
[1] "The element at row: 1 and col: 3 is 11"
[1] "The element at row: 1 and col: 4 is 16"
[1] "The element at row: 1 and col: 5 is 21"
[1] "The element at row: 2 and col: 1 is 2"
[1] "The element at row: 2 and col: 2 is 7"
[1] "The element at row: 2 and col: 3 is 12"
[1] "The element at row: 2 and col: 4 is 17"
[1] "The element at row: 2 and col: 5 is 22"
[1] "The element at row: 3 and col: 1 is 3"
[1] "The element at row: 3 and col: 2 is 8"
[1] "The element at row: 3 and col: 3 is 13"
[1] "The element at row: 3 and col: 4 is 18"
[1] "The element at row: 3 and col: 5 is 23"
[1] "The element at row: 4 and col: 1 is 4"
[1] "The element at row: 4 and col: 2 is 9"
[1] "The element at row: 4 and col: 3 is 14"
[1] "The element at row: 4 and col: 4 is 19"
[1] "The element at row: 4 and col: 5 is 24"
[1] "The element at row: 5 and col: 1 is 5"
[1] "The element at row: 5 and col: 2 is 10"
[1] "The element at row: 5 and col: 3 is 15"
[1] "The element at row: 5 and col: 4 is 20"
[1] "The element at row: 5 and col: 5 is 25"
```

Introduction to Functions

This lecture will consist of explaining what a function is in R and how to create one. Functions will be one of our main building blocks when we construct larger and larger amounts of code to solve problems.

So what is a function?

Formally, a function is a useful device that groups together a set of statements so they can be run more than once. They can also let us specify parameters that can serve as inputs to the functions.

On a more fundamental level, functions allow us to not have to repeatedly write the same code again and again. If you remember back to the lessons on strings and lists, remember that we used a function length() to get the length of a string. Since checking the length of a sequence is a common task you would want to write a function that can do this repeatedly at command. Functions will be one of most basic levels of reusing code in R, and it will also allow us to start thinking about program design.

We already have seen built-in functions and we can use the **help** function to discover the *arguments* that the functions take in.

In [1]:

```
help(sum)
```

Out[1]:

sum {base}	R Documentation

Sum of Vector Elements

Description

`sum` returns the sum of all the values present in its arguments.

Usage

```
sum(..., na.rm = FALSE)
```

Arguments

`...`	numeric or complex or logical vectors.
`na.rm`	logical. Should missing values (including `NaN`) be removed?

Details

This is a generic function: methods can be defined for it directly or via the `Summary` group generic. For this to work properly, the arguments `...` should be unnamed, and dispatch is on the first argument.

If `na.rm` is `FALSE` an `NA` or `NaN` value in any of the arguments will cause a value of `NA` or `NaN` to be returned, otherwise `NA` and `NaN` values are ignored.

Logical true values are regarded as one, false values as zero. For historical reasons, `NULL` is accepted and treated as if it were `integer(0)`.

Loss of accuracy can occur when summing values of different signs: this can even occur for sufficiently long integer inputs if the partial sums would cause integer overflow. Where possible extended-precision accumulators are used, but this is platform-dependent.

Value

The sum. If all of `...` are of type integer or logical, then the sum is integer, and in that case the result will be `NA` (with a warning) if integer overflow occurs. Otherwise it is a length-one numeric or complex vector.

NB: the sum of an empty set is zero, by definition.

S4 methods

This is part of the S4 `Summary` group generic. Methods for it must use the signature `x, na.rm`

This is part of the S4 `Summary` group generic. Methods for it must use the signature `x, ..., na.rm`.

'plotmath' for the use of `sum` in plot annotation.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`colSums` for row and column sums.

Examples

```
## Pass a vector to sum, and it will add the elements together.
sum(1:5)

## Pass several numbers to sum, and it also adds the elements.
sum(1, 2, 3, 4, 5)

## In fact, you can pass vectors into several arguments, and everything gets added.
sum(1:2, 3:5)

## If there are missing values, the sum is unknown, i.e., also missing, ....
sum(1:5, NA)
## ... unless  we exclude missing values explicitly:
sum(1:5, NA, na.rm = TRUE)
```

[Package *base* version 3.2.2]

Notice how the format is:

```
name_of_function(input1,input2,....)
```

So how do we create this ourselves? Here is the syntax for writing your own function:

```
name_of_function <- function(arg1,arg2,...){
                # Code that gets executed when function is called
}
```

Let's see some examples.

Example 1

In [2]:

```
# Simple function, no inputs!
hello <- function(){
    print('hello!')
}
```

In [3]:

```
hello()
```

```
[1] "hello!"
```

Example 2

In [4]:

```
helloyou <- function(name){
    print(paste('hello ',name))
}
```

In [5]:

```
helloyou('Sammy')
```

```
[1] "hello  Sammy"
```

Example 3

In [6]:

```
add_num <- function(num1,num2){
    print(num1+num2)
}
```

In [7]:

```
add_num(5,10)
```

```
[1] 15
```

Default values

Notice that so far we've had to define every single argument in the function when using it, but we can also have default values by using an equals sign, for example:

In [8]:

```
hello_someone <- function(name='Frankie'){
    print(paste('Hello ',name))
}
```

In [9]:

```
# uses default
hello_someone()
```

```
[1] "Hello  Frankie"
```

In [10]:

```
# overwrite default
hello_someone('Sammy')
```

```
[1] "Hello  Sammy"
```

You'll see lots of built-in functions use default values for a variety of tasks, where the users will usually need a particular value.

Returning Values

So far we've only been printing out results, but what if we wanted to **return** the results so that we could assign them to a variable, we can use the *return* keyword for this task in the following manner:

In [12]:

```
formal <- function(name='Sam',title='Sir'){
    return(paste(title,' ',name))
}
```

In [13]:

```
formal()
```

Out[13]:

'Sir Sam'

In [14]:

```
formal('Issac Newton')
```

Out[14]:

'Sir Issac Newton'

Notice how we aren't printing, we are returning, meaning we can assign this to a variable:

In [15]:

```
var <- formal('Marie Curie','Ms.')
```

In [16]:

```
var
```

Out[16]:

'Ms. Marie Curie'

This is the sort of syntax you want to use for your functions when you want to pass arguments to them, and then get some sort of result in return.

Scope

Scope is the term we use to describe how objects and variable get defined within R. When discussing scope with functions, as a general rule we can say that if a variable is defined only inside a function than its scope is limited to that function. For example, consider the following function:

In [24]:

```
# Multiplies input by 5
times5 <- function(input) {
  result <- input ^ 2
  return(result)
}
```

In [27]:

```
pow_two(4)
result # Not defined outside the scope of the function
input # Not defined outside the scope of the function
```

Out[27]:

16

```
Error in eval(expr, envir, enclos): object 'result' not found
```

```
Error in eval(expr, envir, enclos): object 'input' not found
```

These error indicate that these variables are only defined inside the *scope* of the function. So variables defined inside of a function are only defined (or redefined) inside of that function. However, variables assigned outside of the function are *global* variables, and the function will have access to them due to their scope. For example:

In [35]:

```
v <- "I'm global v"
stuff <- "I'm global stuff"

fun <- function(stuff){
    print(v)
    stuff <- 'Reassign stuff inside func'
    print(stuff)
}
```

In [36]:

```
print(v) #print v
print(stuff) #print stuff
fun(stuff) # pass stuff to function
# reassignment only happens in scope of function
print(stuff)
```

```
[1] "I'm global v"
[1] "I'm global stuff"
[1] "I'm global v"
[1] "Reassign stuff inside func"
[1] "I'm global stuff"
```

So what is happening above? The following happens

print(v) will check for the global variable v, the outer scope

print(stuff) will also check for the global variable stuff

fun(stuff) will accept an argument stuff, print out v, and then reassign stuff (in the scope of the function) and print out stuff. Notice two things:

- The reassignment of stuff only effects the scope of the stuff variable inside the function
- The fun function first checks to see if **v** is defined at the function scope, if not (which was the case) it will then search the global scope for a variable names **v**, leading to it printing out "I'm global v".

Check out the function below and make sure you understand it:

In [39]:

```
double <- function(a) {
  a <- 2*a
  a
}
var <- 5
double(var)
var
```

Out[39]:

10

Out[39]:

5

2 Variable Plotting with ggplot2

In this lecture we will briefly show some examples of how you can compare two variables from a dataset. For these examples, you'll need the full control of ggplot instead of using qplot(), but we will show a quick example of what qplot is capable of. We'll use the built-in ggplot2 movies dataset:

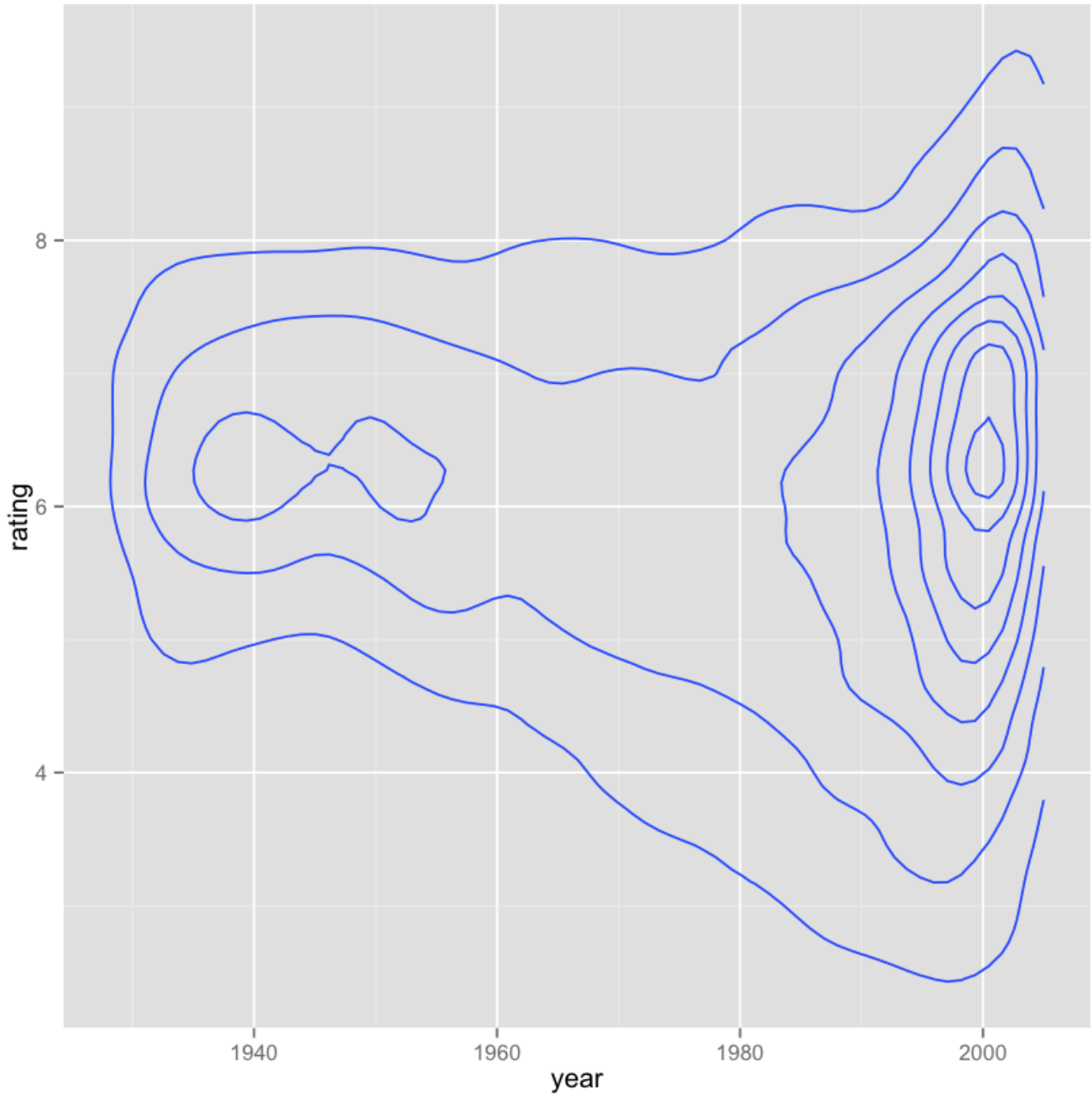
In [ ]:

```
library(ggplot2)
df <- movies
```

qplot()

In [28]:

```
qplot(x=year, y=rating, data = df, geom = "density2d")
```



In [ ]:

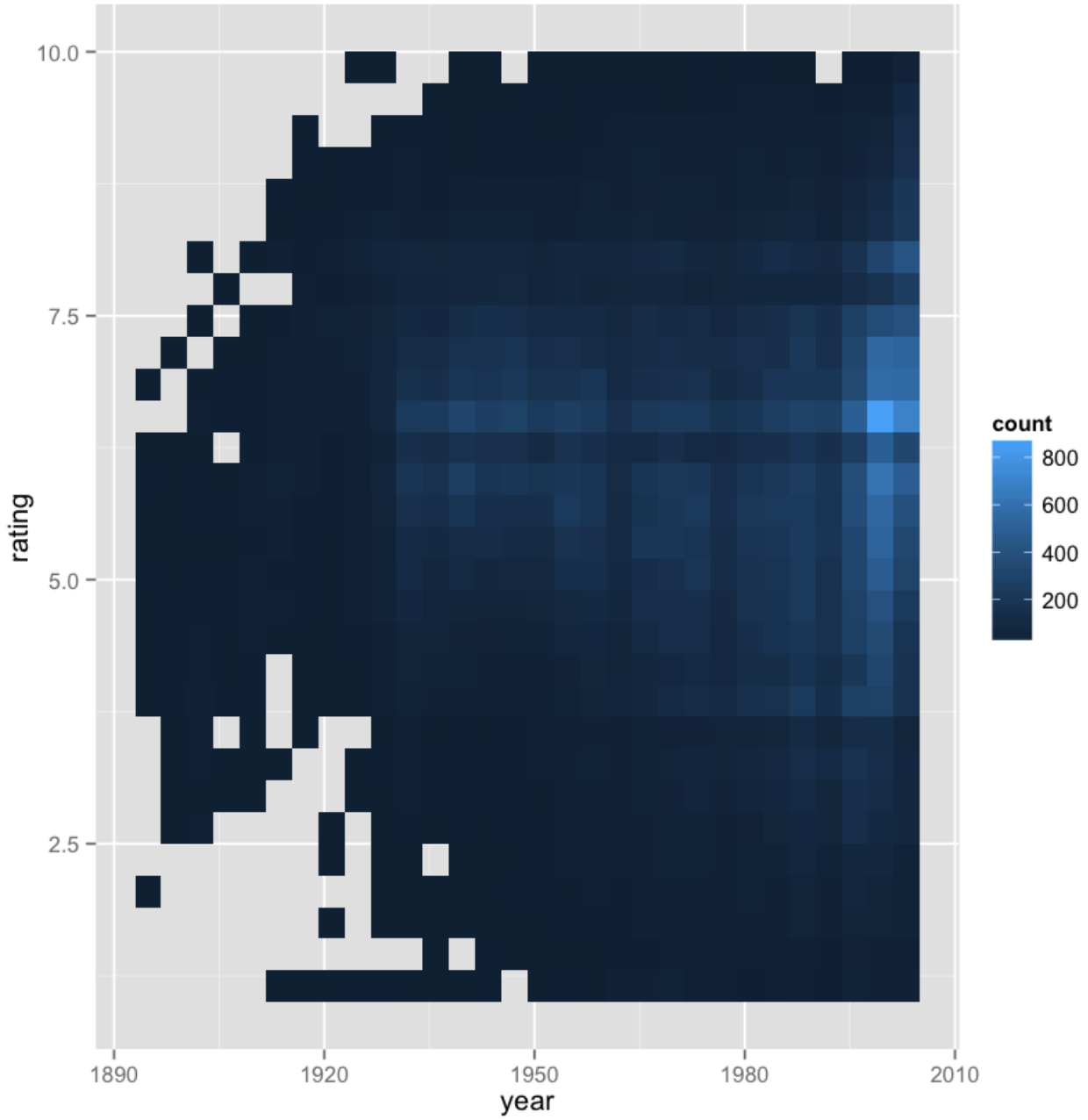
In [6]:

```
pl <- ggplot(movies,aes(x = year,y=rating))
```

2d Bin Chart

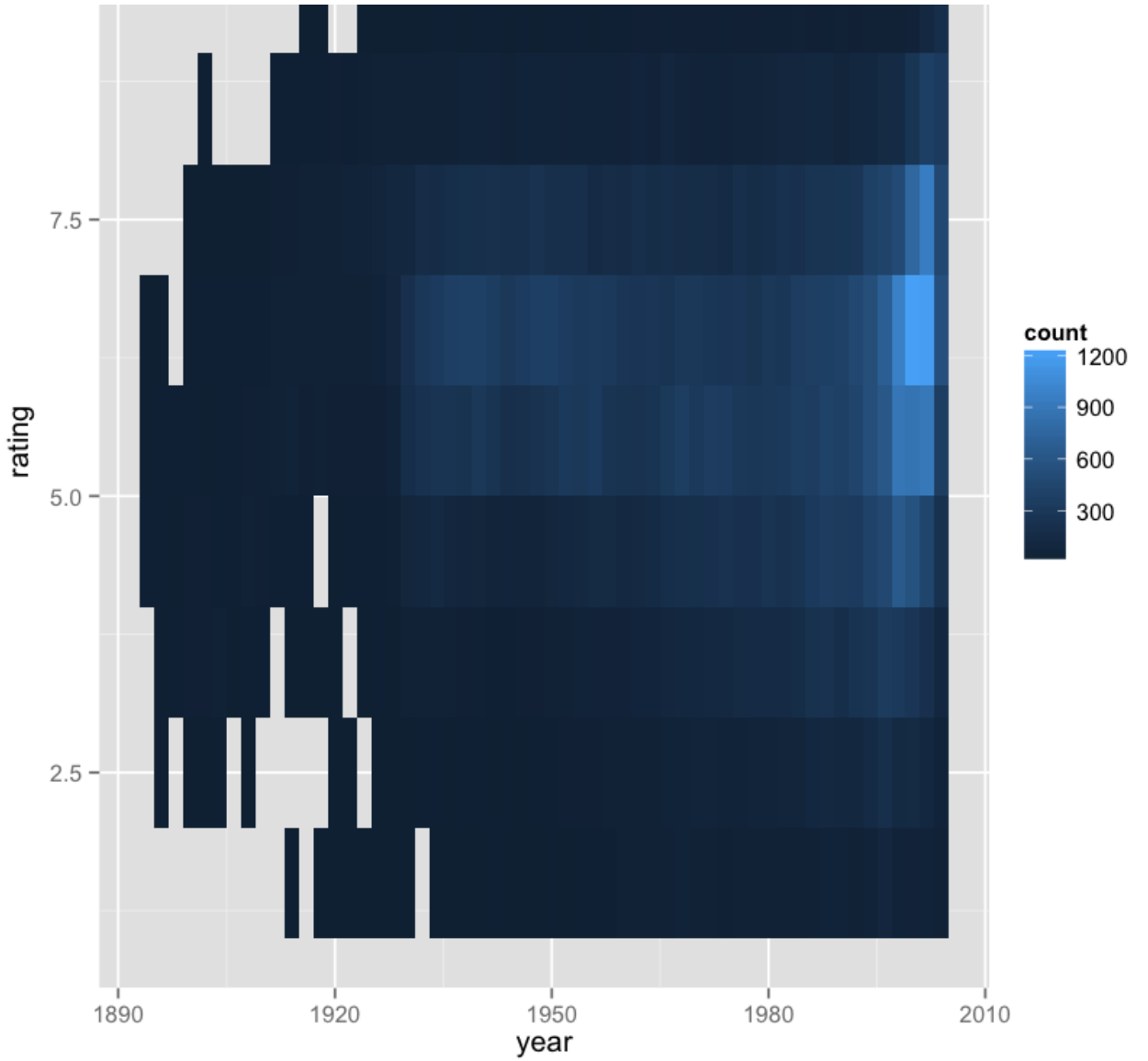
In [7]:

```
pl + geom_bin2d()
```



In [13]:

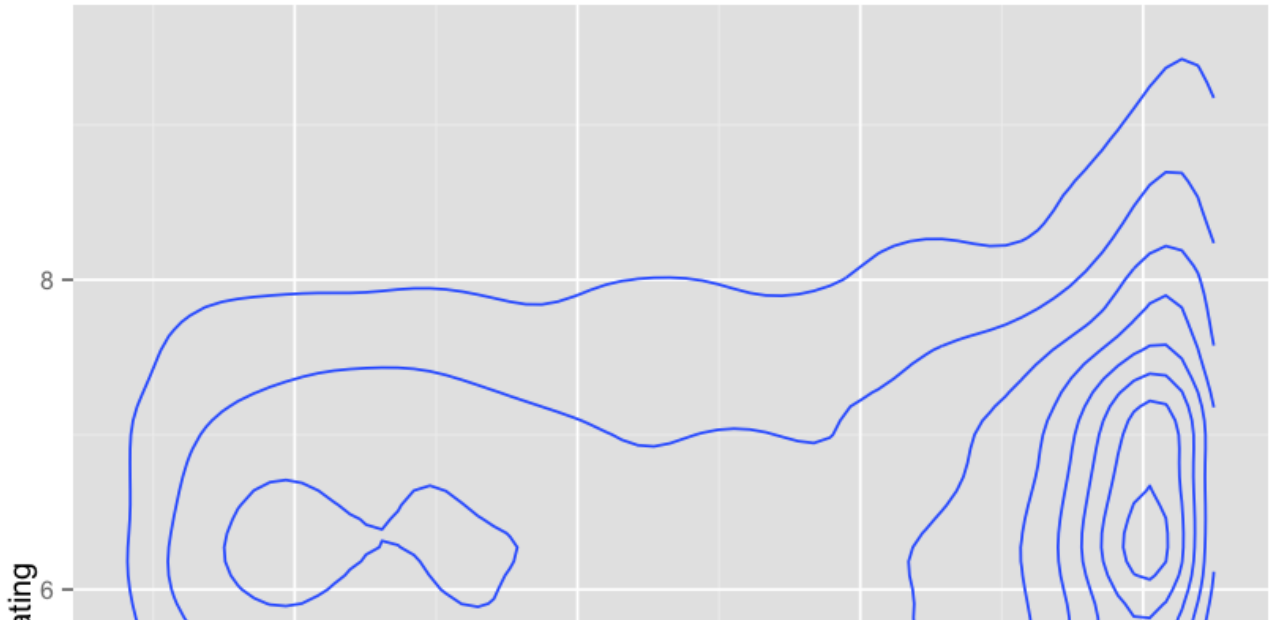
```
# Control bin sizes
pl + geom_bin2d(binwidth=c(2,1))
```

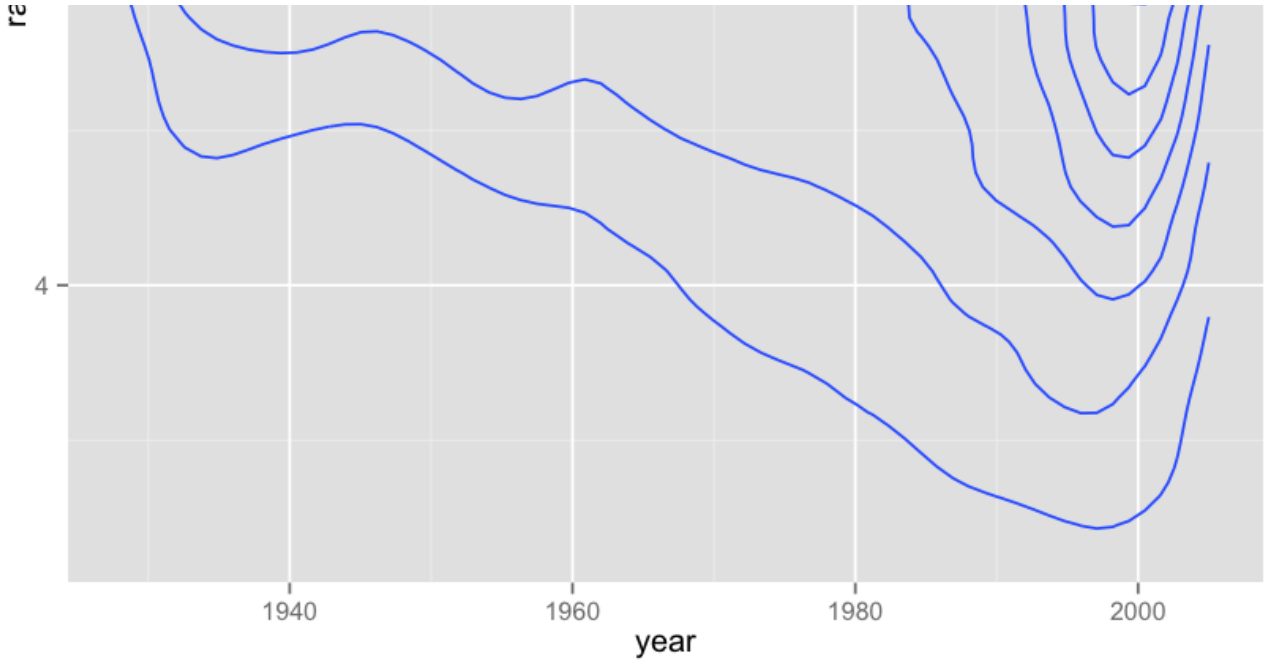


2d Density Plot

In [29]:

```
pl + geom_density2d()
```

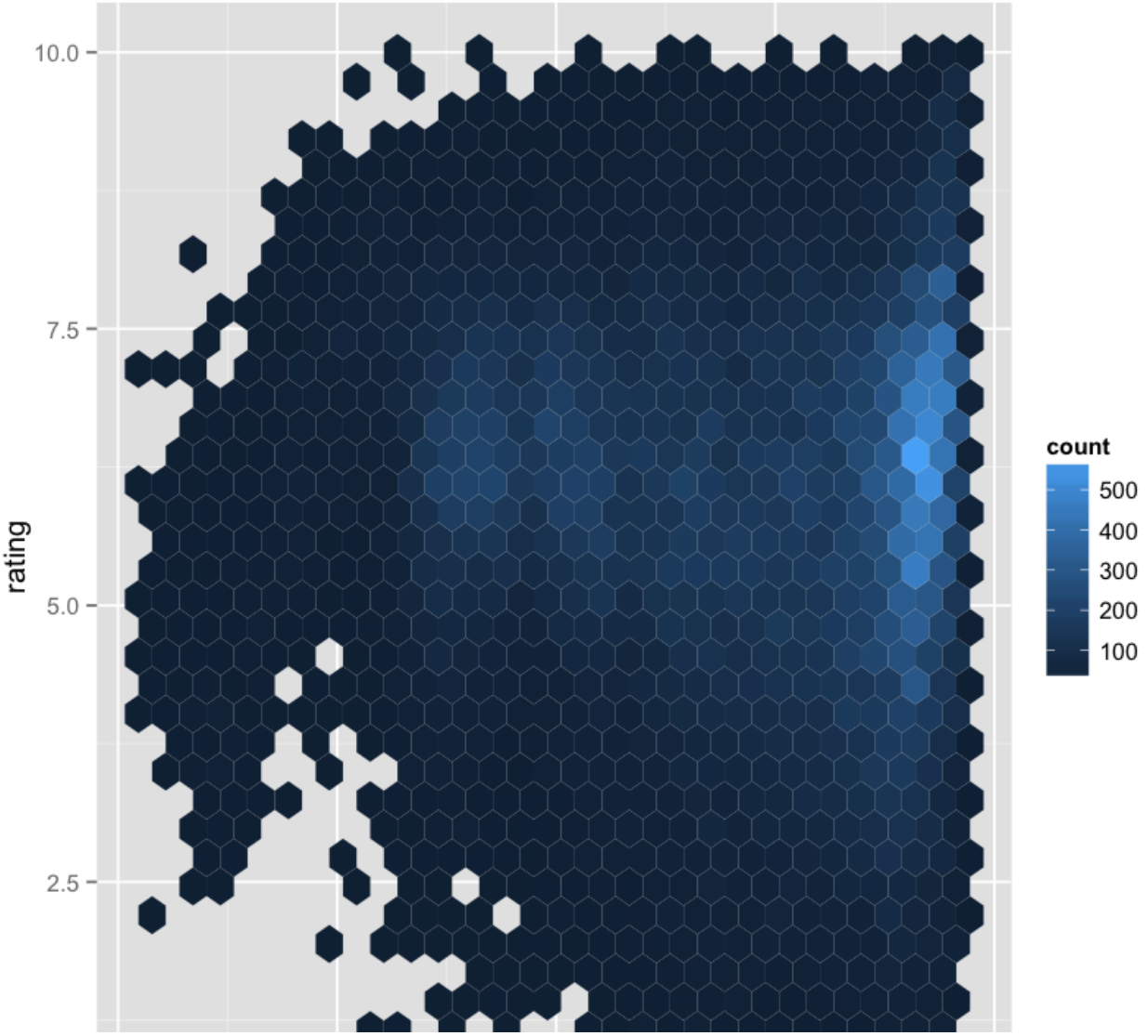


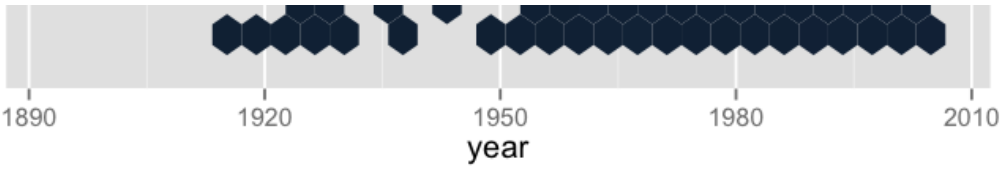


2d Hex Plot

In [30]:

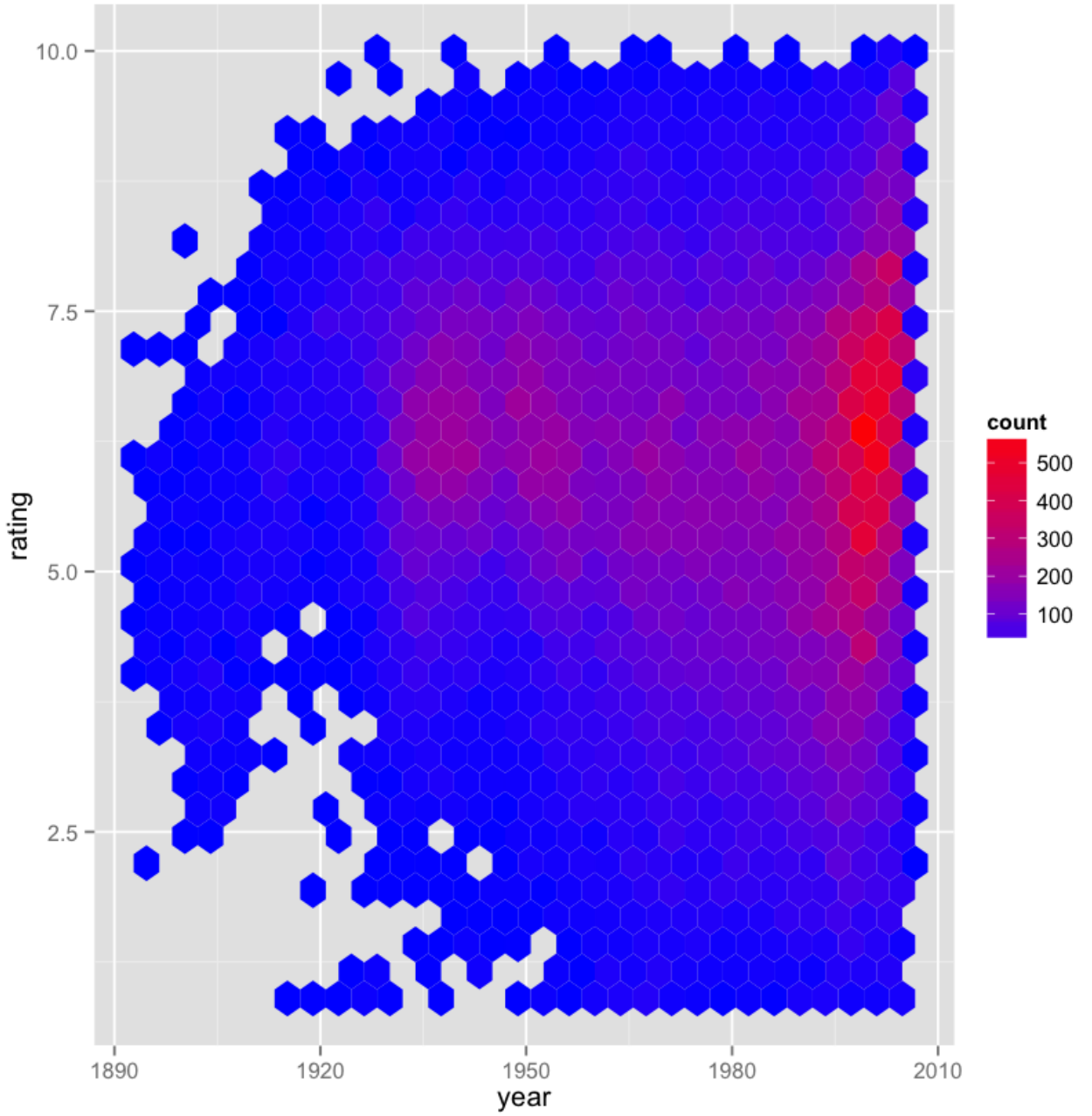
```
pl + geom_hex()
```





In [32]:

```
pl + geom_hex() + scale_fill_gradient(high='red',low='blue')
```



Barplots with ggplot2

Barplots are a useful way of displaying occurence counts when a histogram isn't quite what you're looking for! In ggplot2, there are two types of bar charts, determined by what is mapped to bar height. By default, geom_bar uses stat="count" which makes the height of the bar proportion to the number of cases in each group (or if the weight aethetic is supplied, the sum of the weights). If you want the heights of the bars to represent values in the data, use stat="identity" and map a variable to the y aesthetic.

There isn't a really simple,nice way to do this with qplot, so we'll skip straigh to using ggplot.

Let's see how we can create them using the built-in mpg dataset.

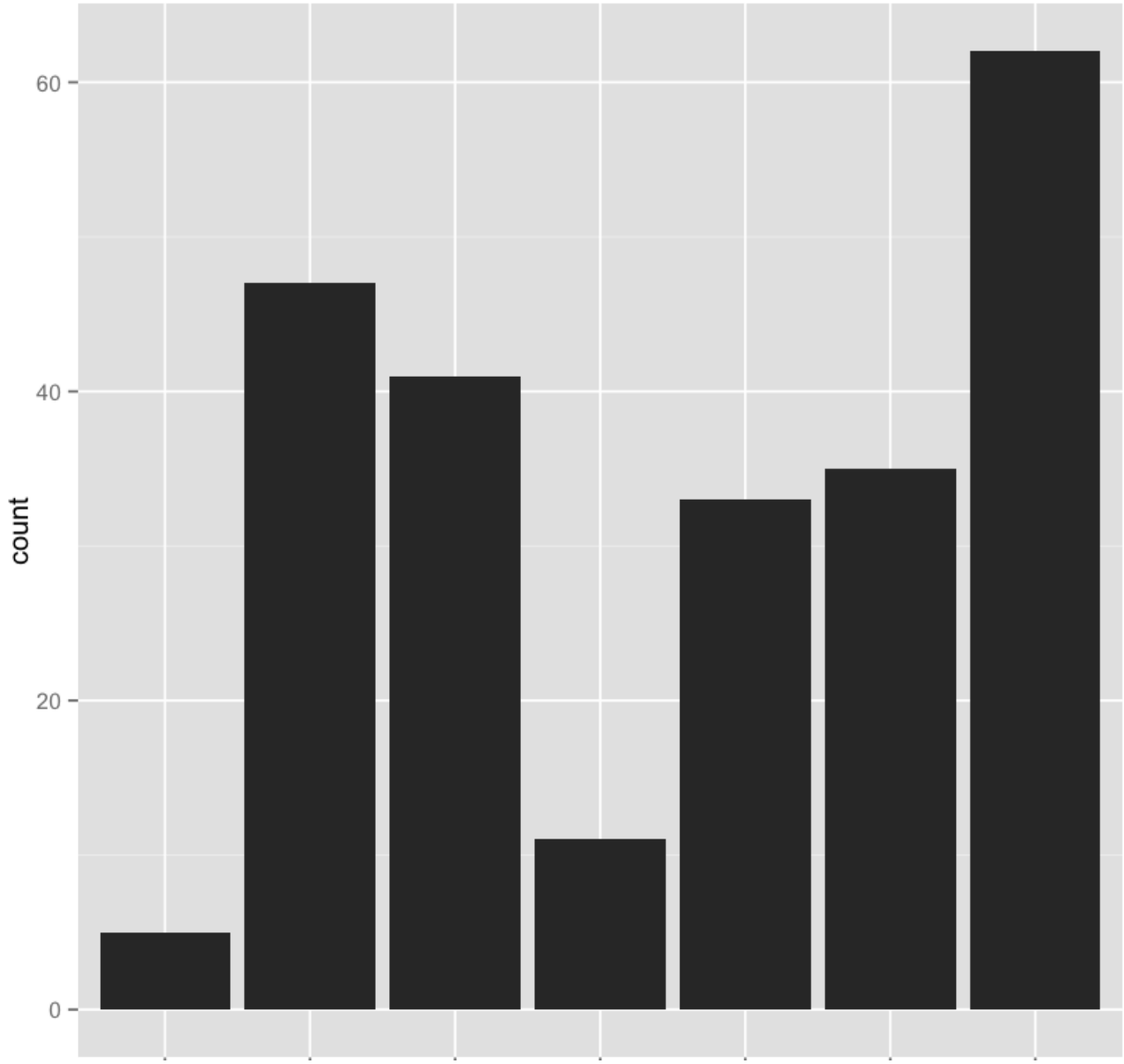
ggplot()

In [4]:

```
library(ggplot2)
```

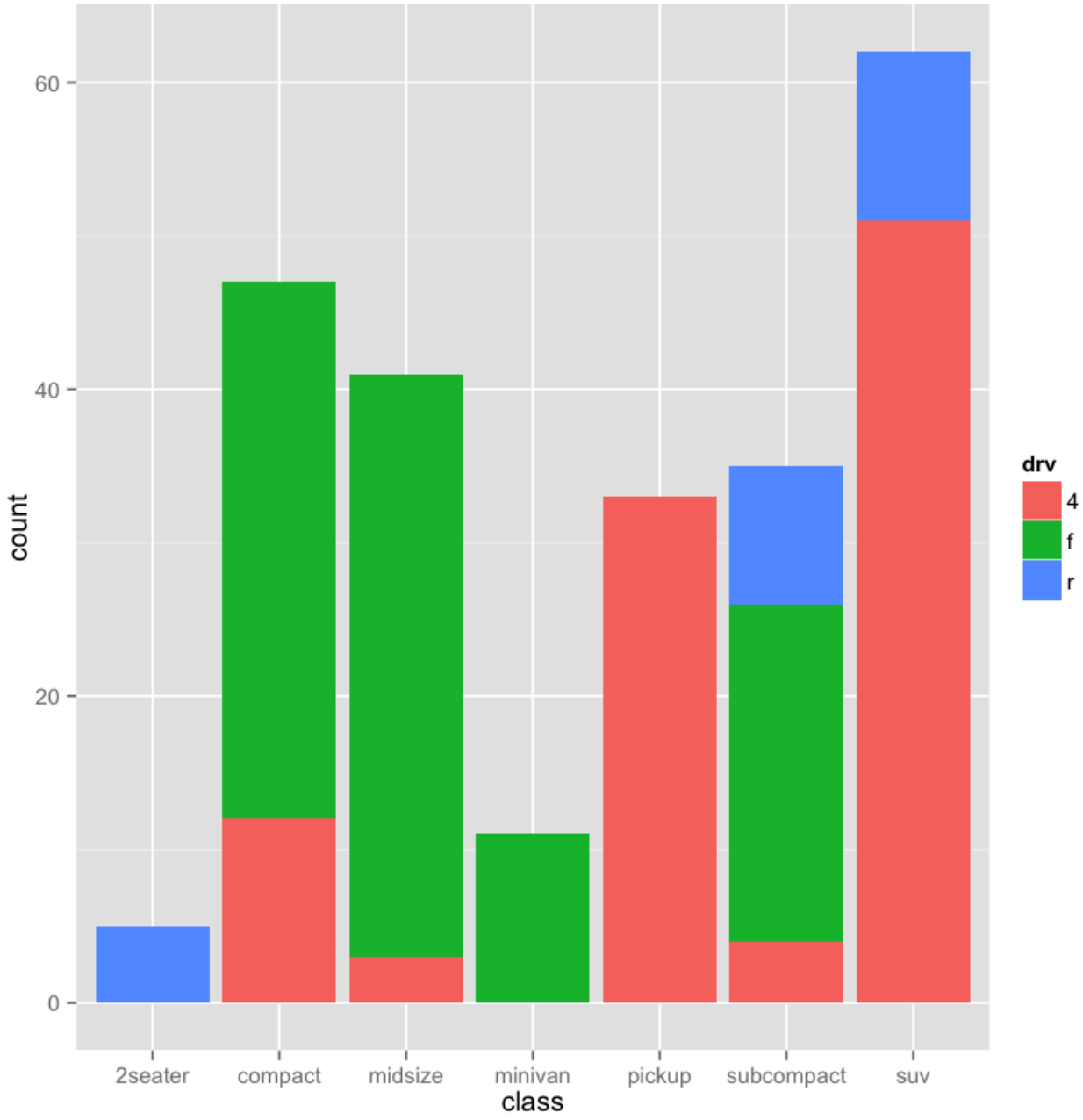
In [5]:

```
# counts (or sums of weights)
g <- ggplot(mpg, aes(class))
# Number of cars in each class:
g + geom_bar()
```



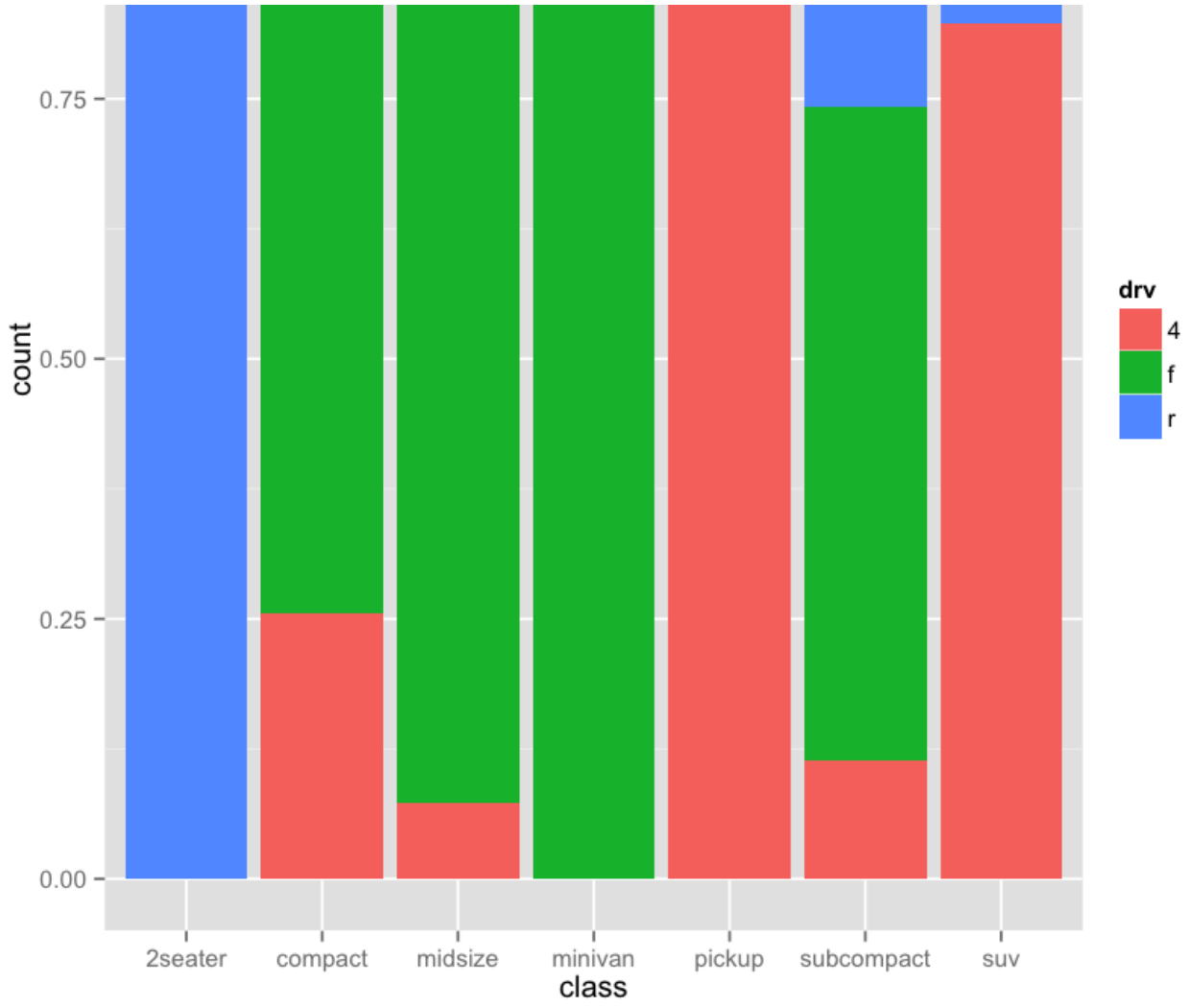
In [6]:

```
# Bar charts are automatically stacked when multiple bars are placed
# at the same location
g + geom_bar(aes(fill = drv))
```



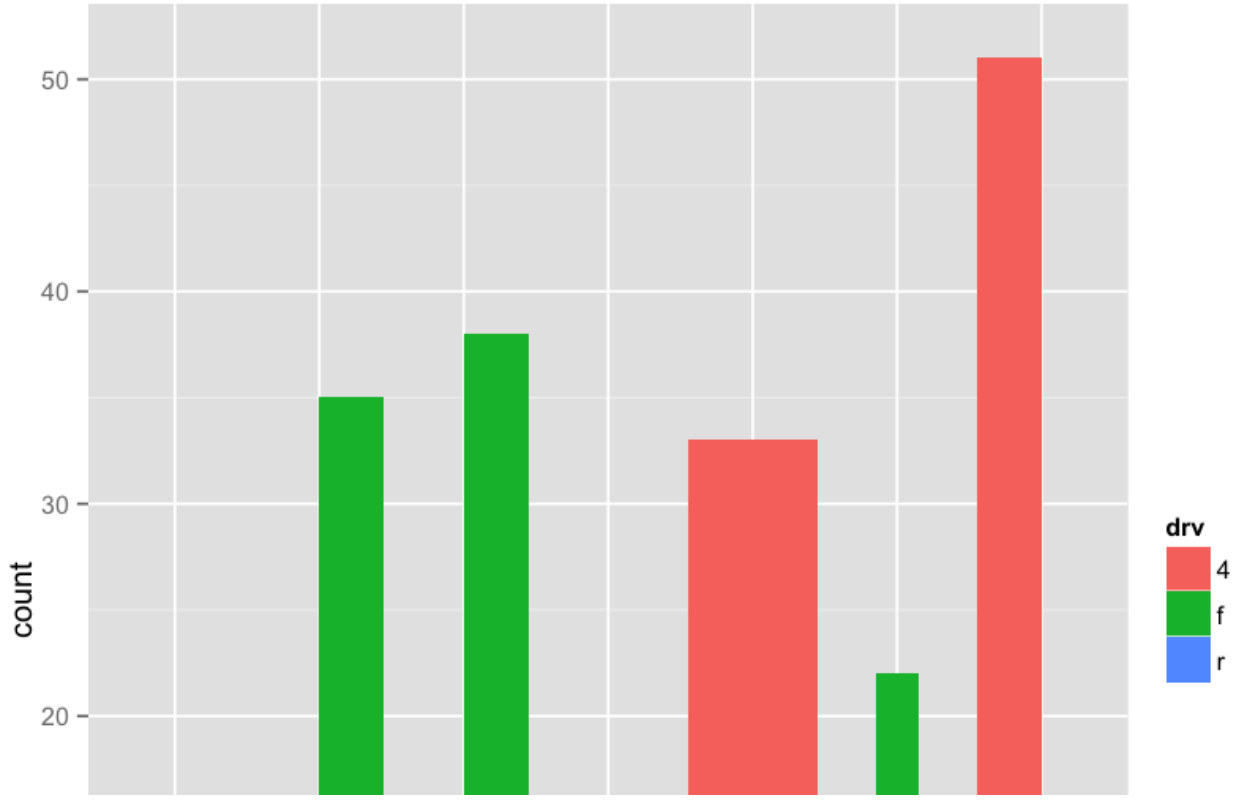
In [7]:

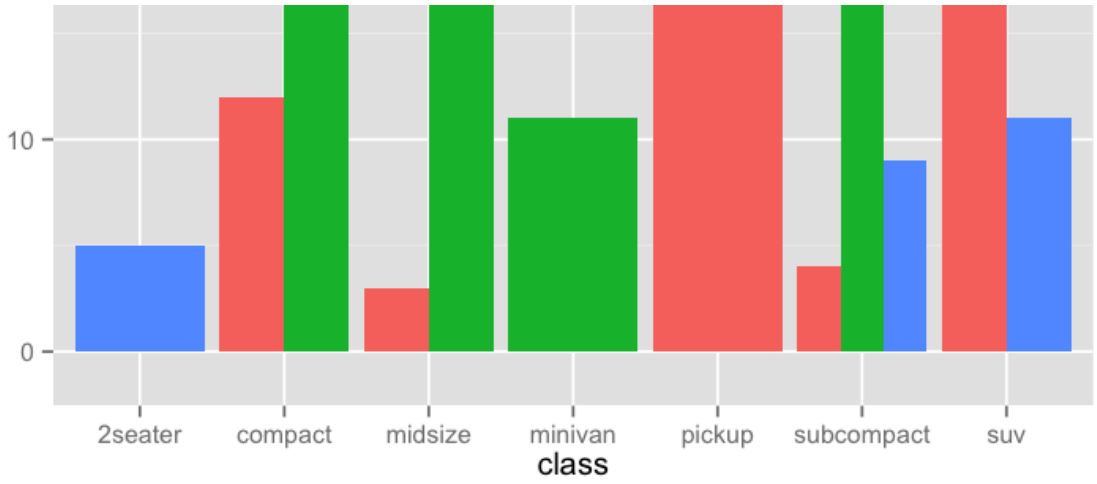
```
g + geom_bar(aes(fill = drv), position = "fill")
```



In [8]:

```
# You can instead dodge, or fill them
g + geom_bar(aes(fill = drv), position = "dodge")
```





That's it for the basics of barplots!

Scatterplots with ggplot2

Scatter plots allow us to place points that let us see possible correlations between two features of a data set. Let's see how we can create them with ggplot!

We'll use the built-in mtcars dataset:

In [10]:

```
library('ggplot2')
df <- mtcars
```

In [13]:

```
head(df)
```

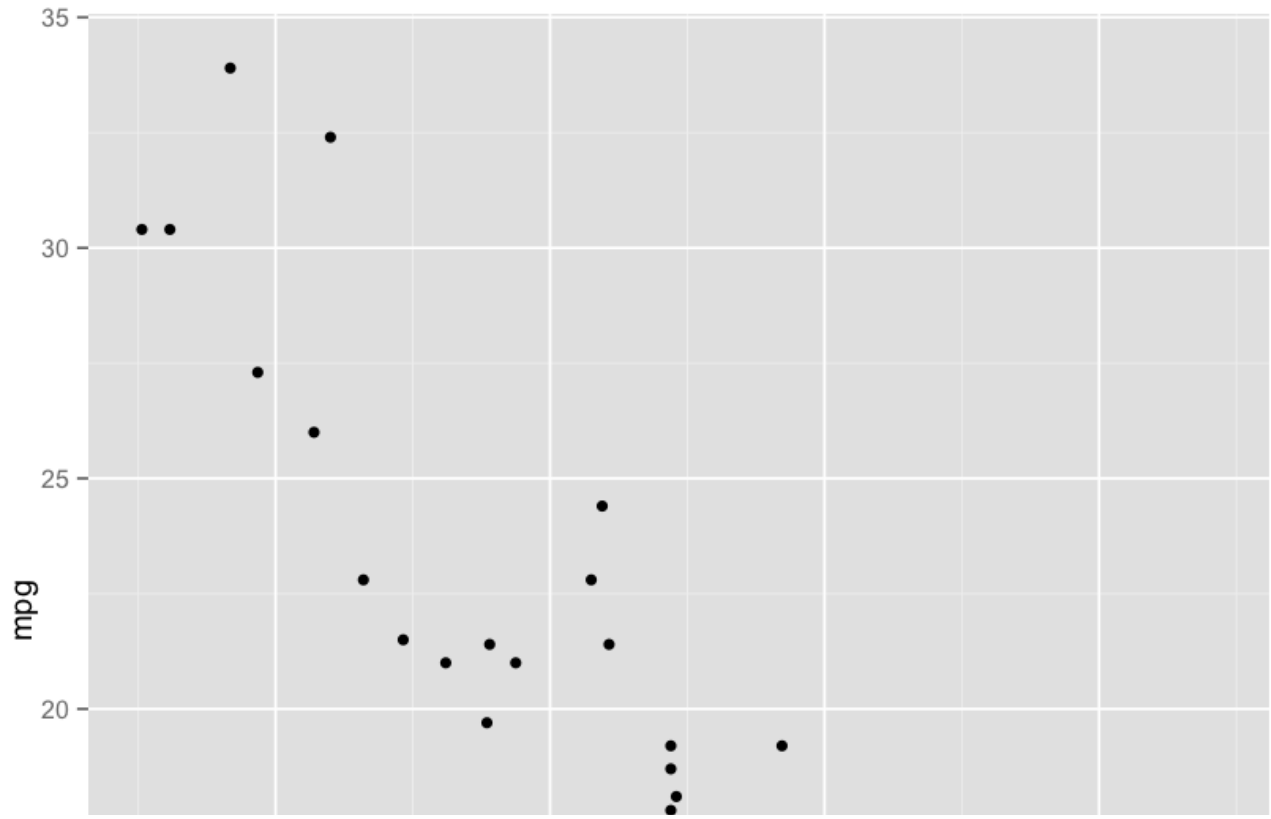
Out[13]:

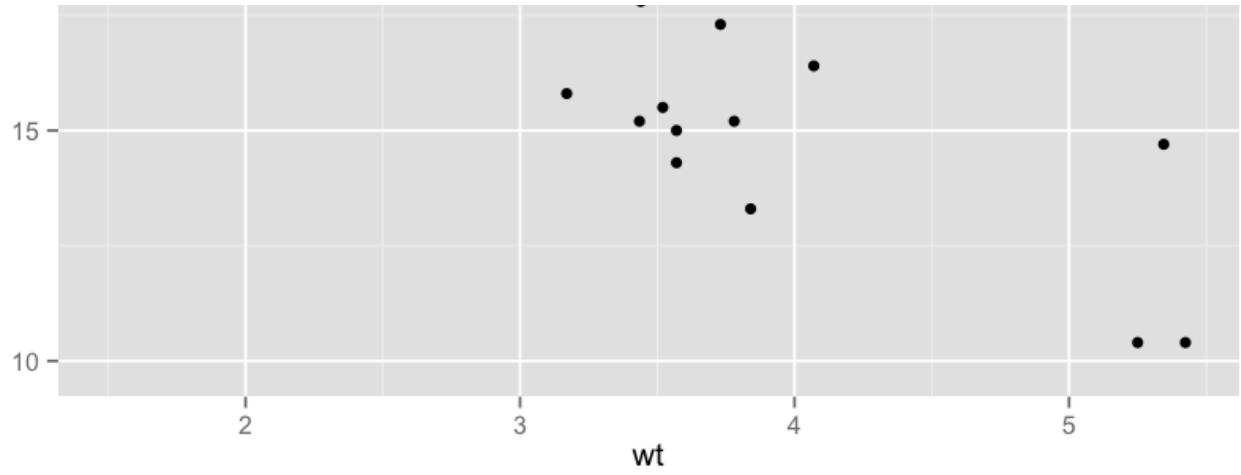
	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.62	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.32	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.44	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.46	20.22	1	0	3	1

qplot()

In [14]:

```
qplot(wt,mpg,data=df)
```



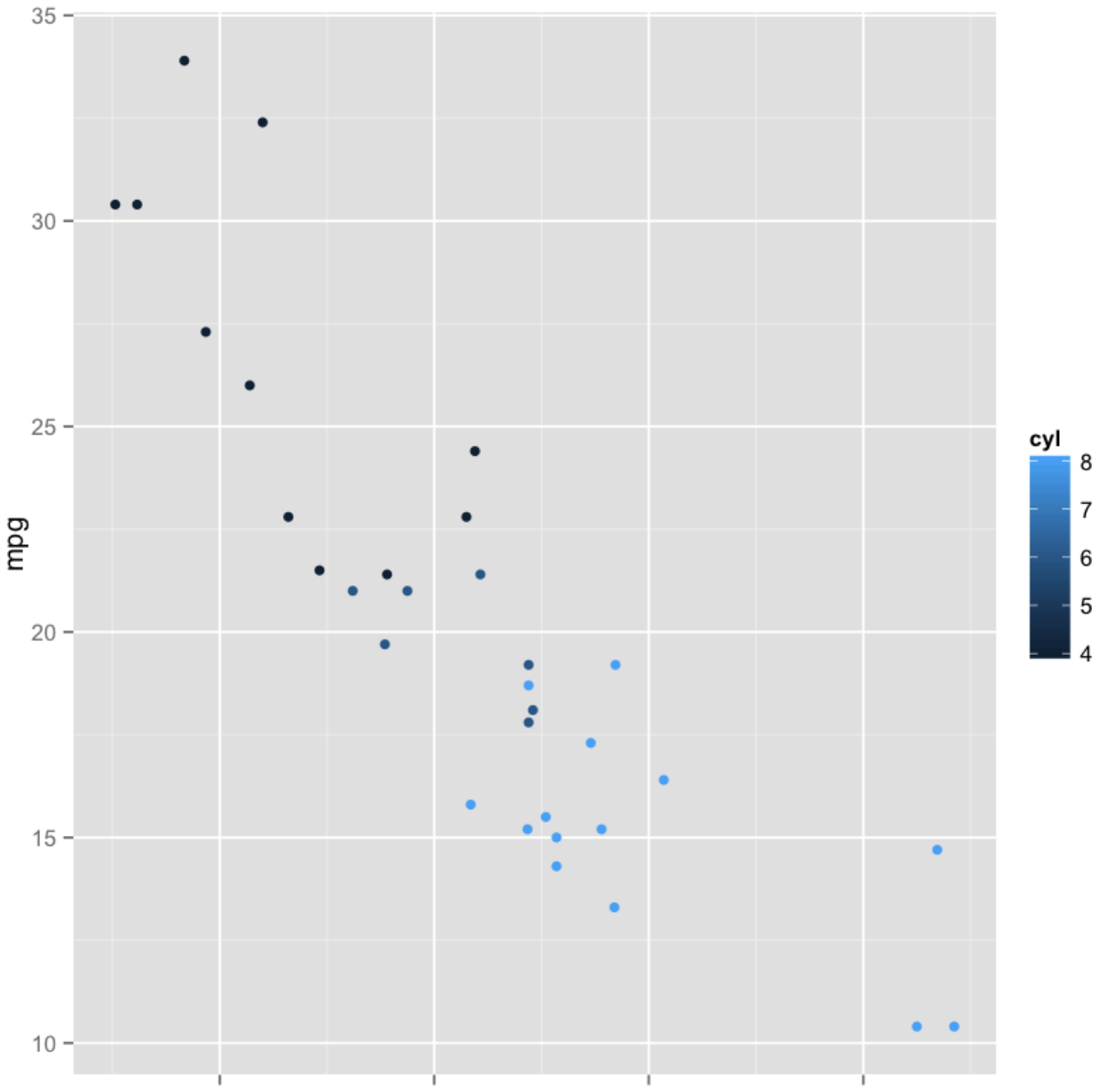


Adding a 3rd feature

We can add a third feature by adding a color gradient on each point, or by resizing each point based on their value of this 3rd feature. For example:

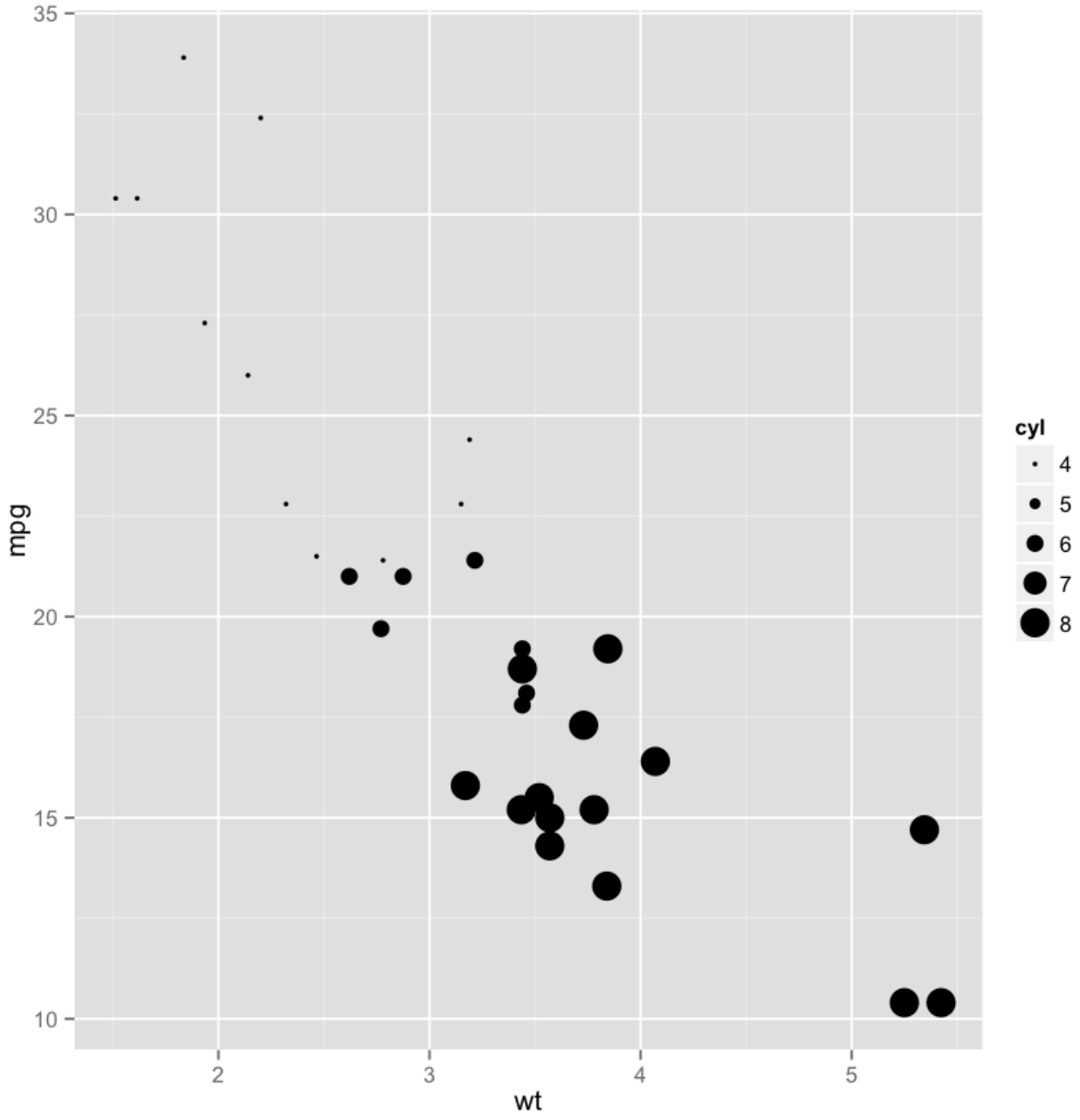
In [15]:

```
qplot(wt,mpg,data=df,color=cyl)
```



In [17]:

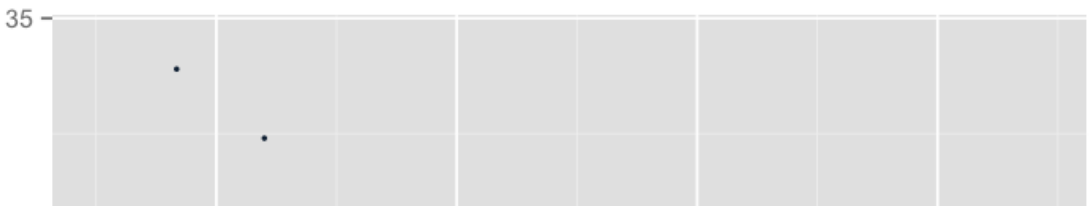
```
qplot(wt,mpg,data=df,size=cyl)
```

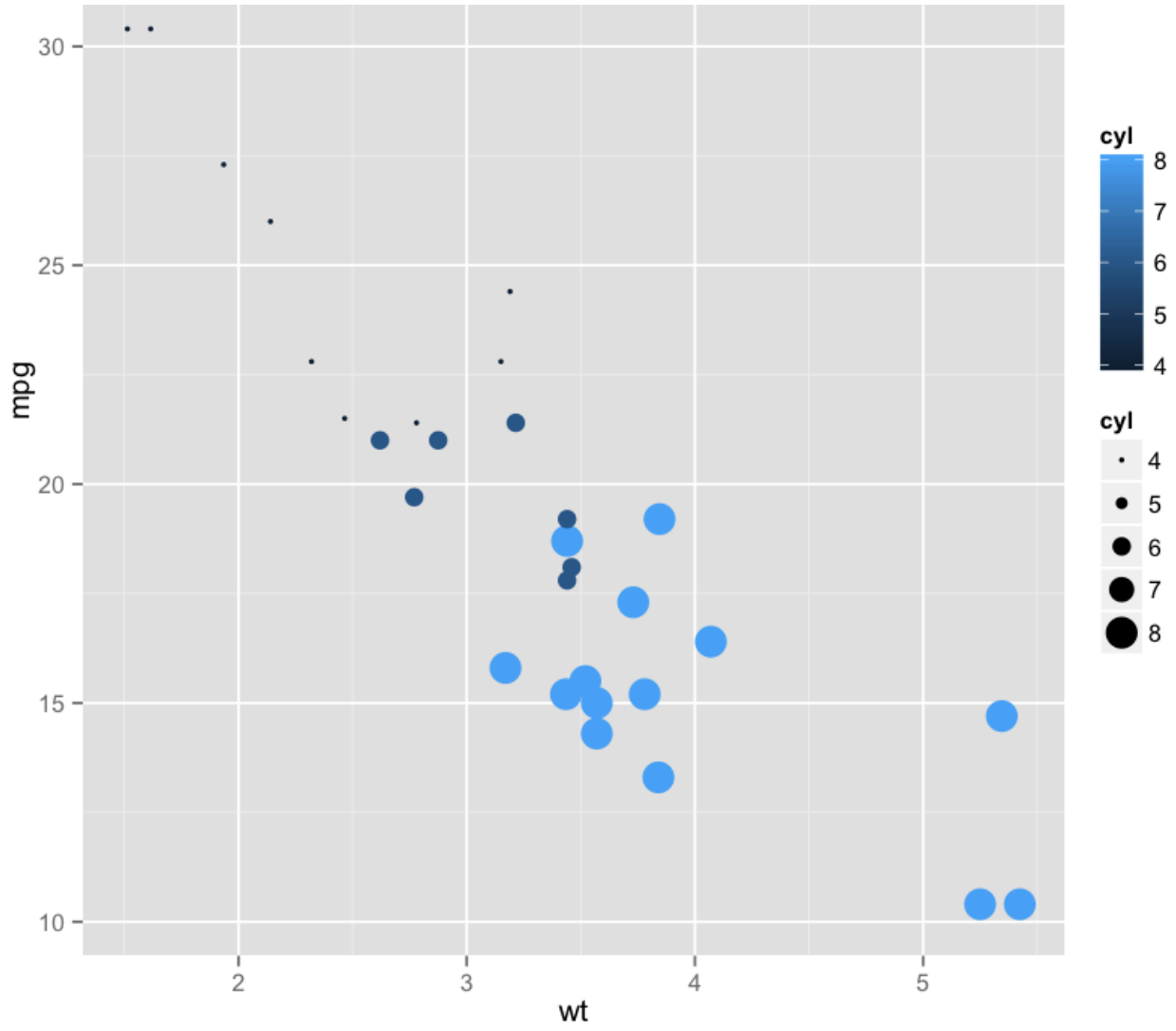


Or both

In [18]:

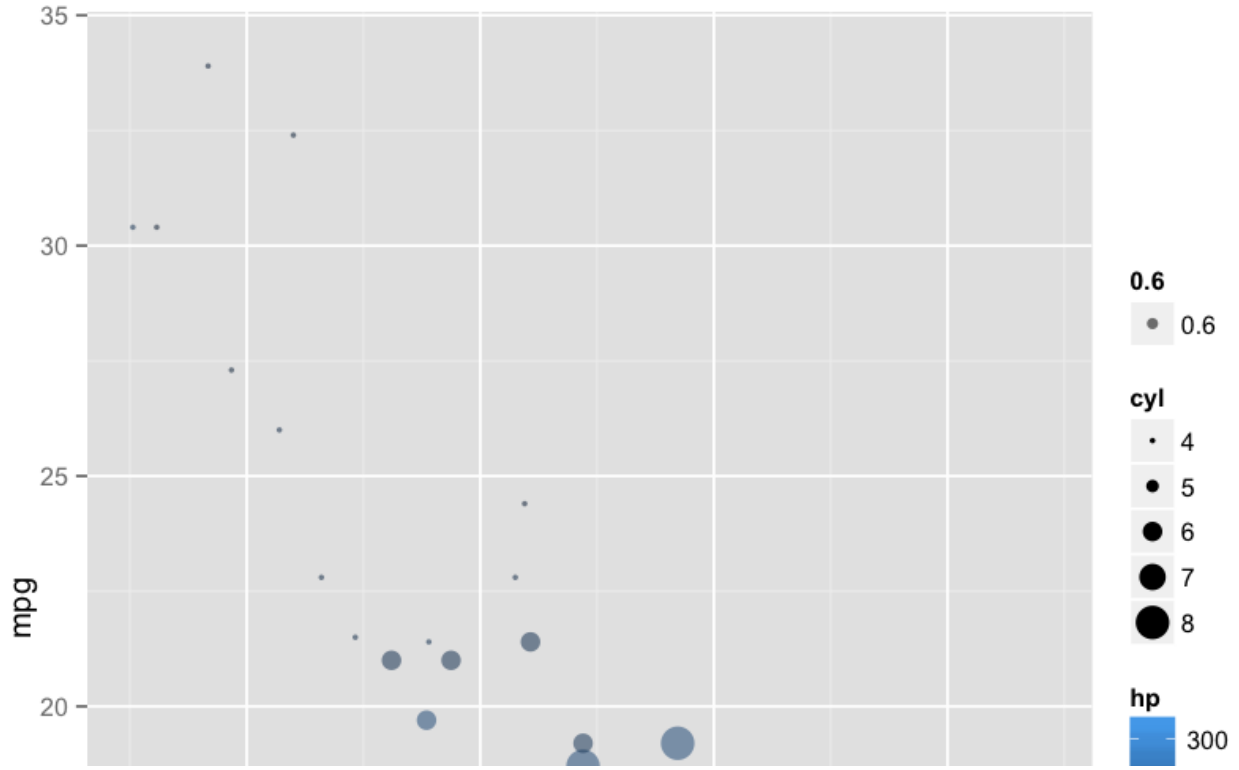
```
qplot(wt,mpg,data=df,size=cyl,color=cyl)
```

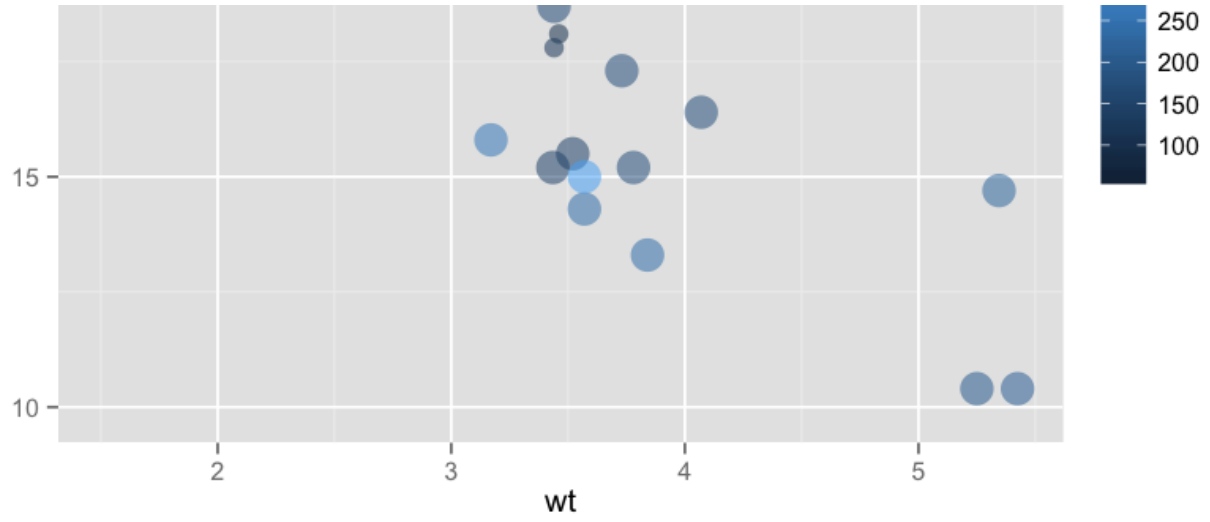




In [21]:

```
# Show 4 features (this gets messy)
qplot(wt,mpg,data=df,size=cyl,color=hp,alpha=0.6)
```



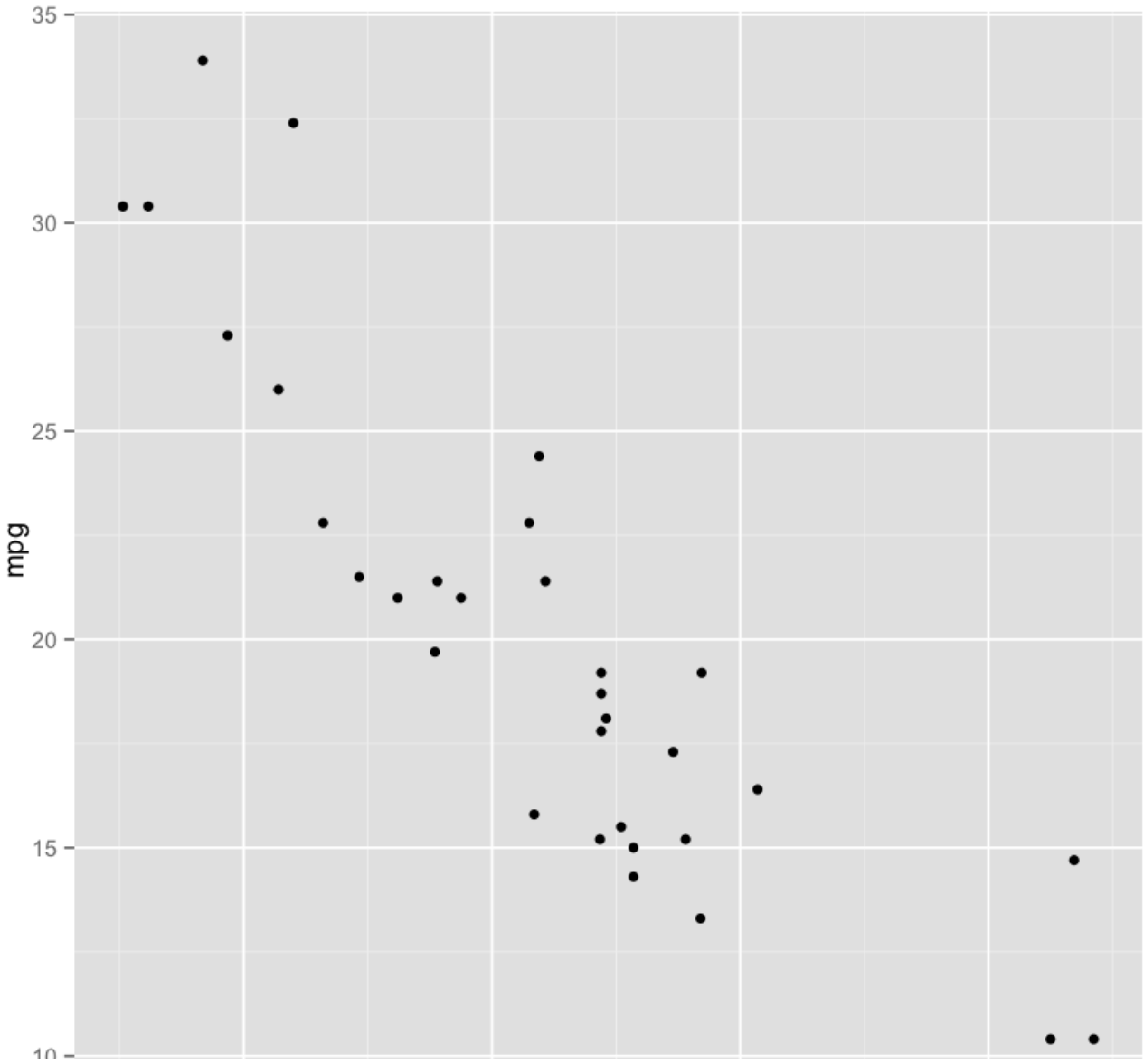


ggplot()

Now let's see hwo to get more control by using ggplot():

In [30]:

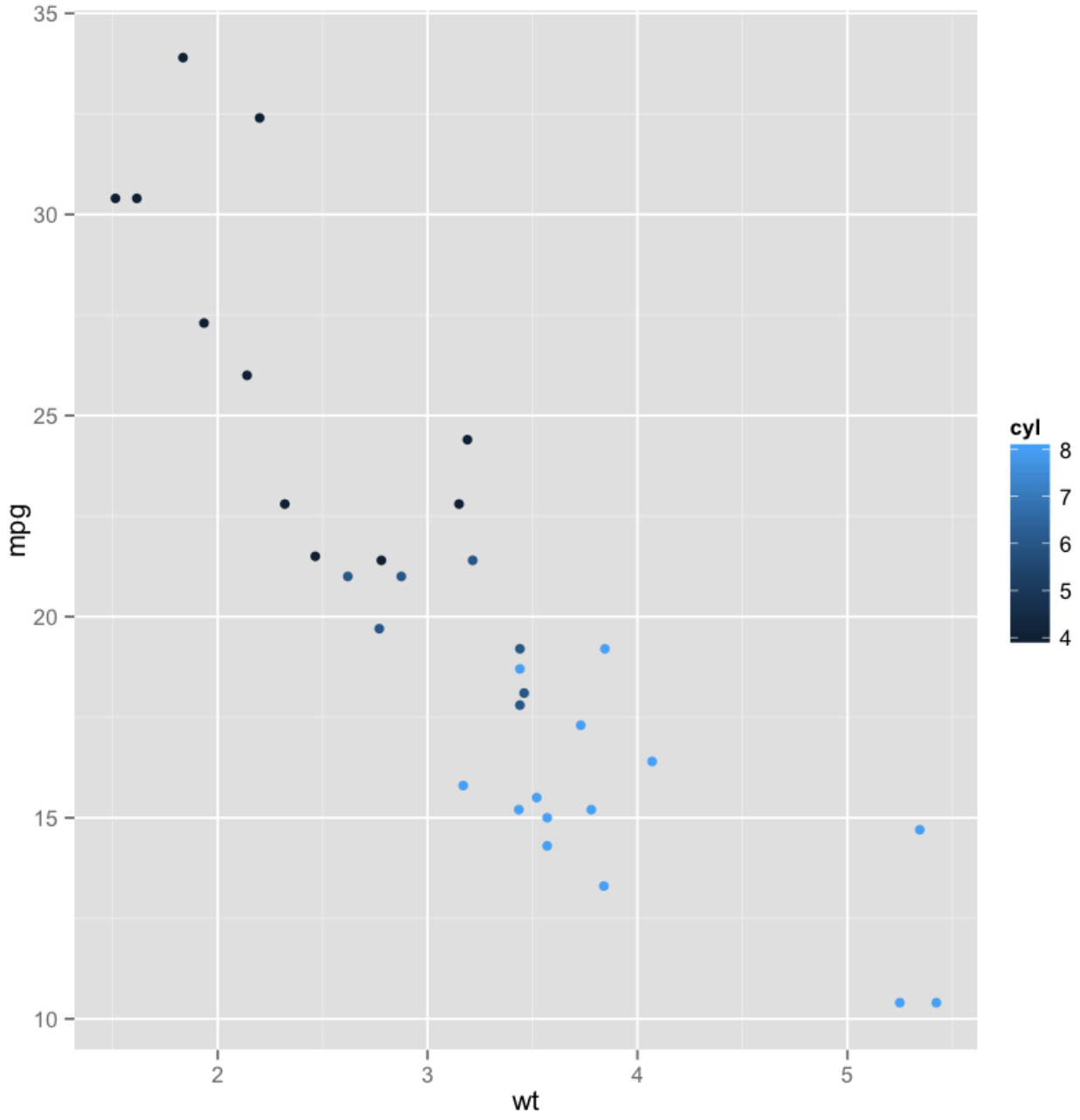
```
pl <- ggplot(data=df,aes(x = wt,y=mpg))
pl + geom_point()
```



Adding 3rd feature

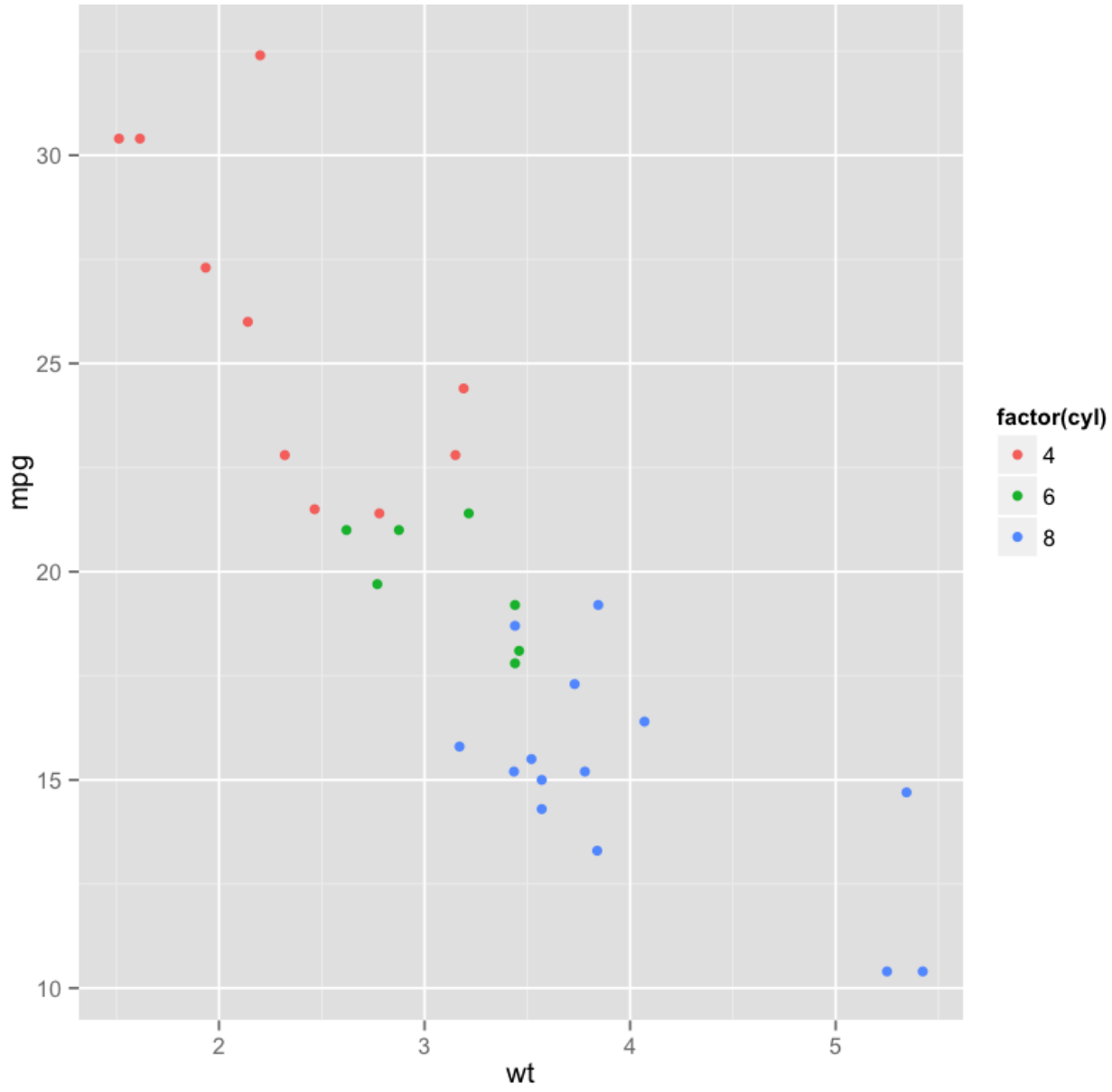
In [32]:

```
pl <- ggplot(data=df,aes(x = wt,y=mpg))
pl + geom_point(aes(color=cyl))
```



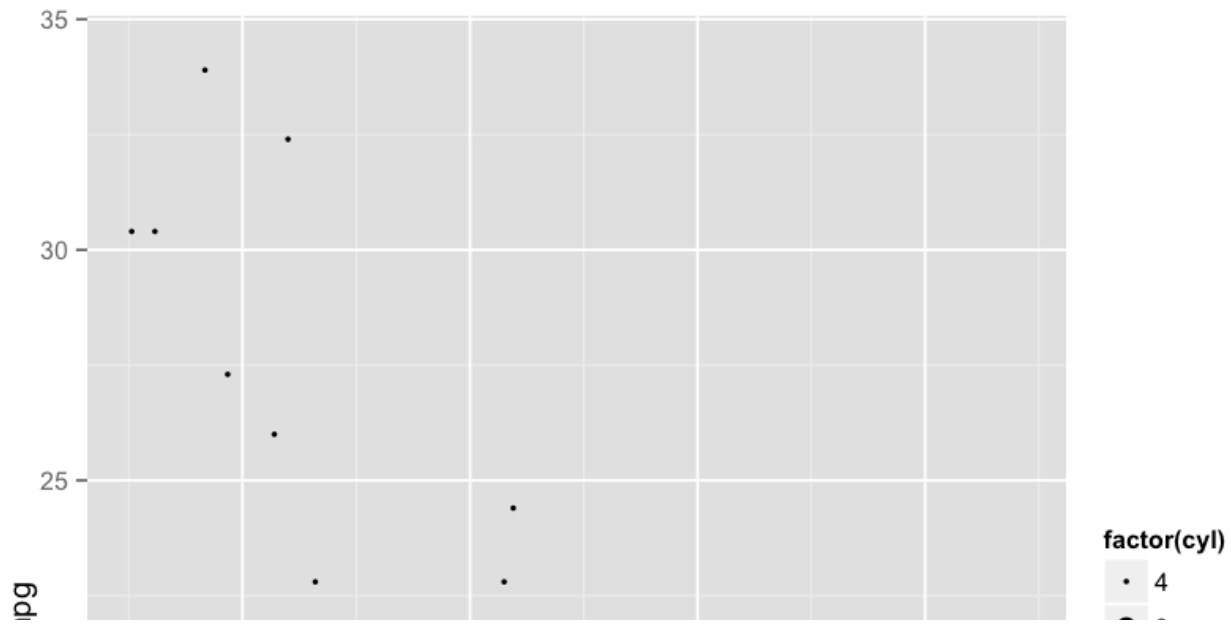
In [37]:

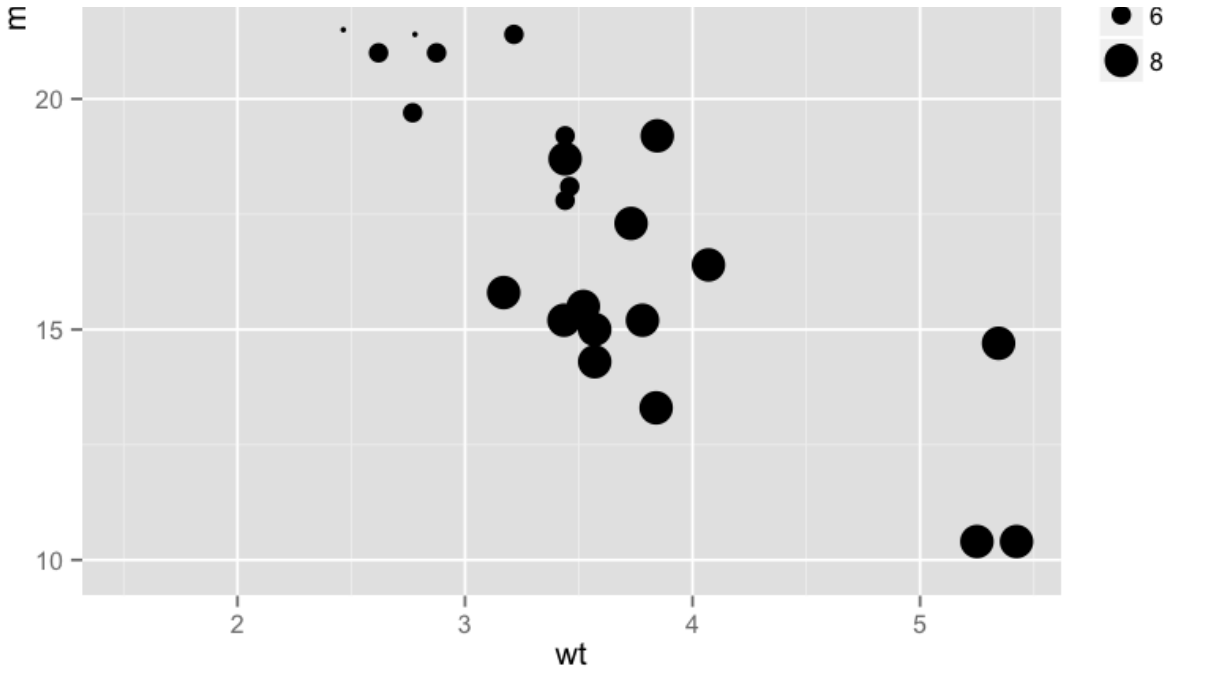
```
pl <- ggplot(data=df,aes(x = wt,y=mpg))

pl + geom_point(aes(color=factor(cyl)))
```



In [38]:

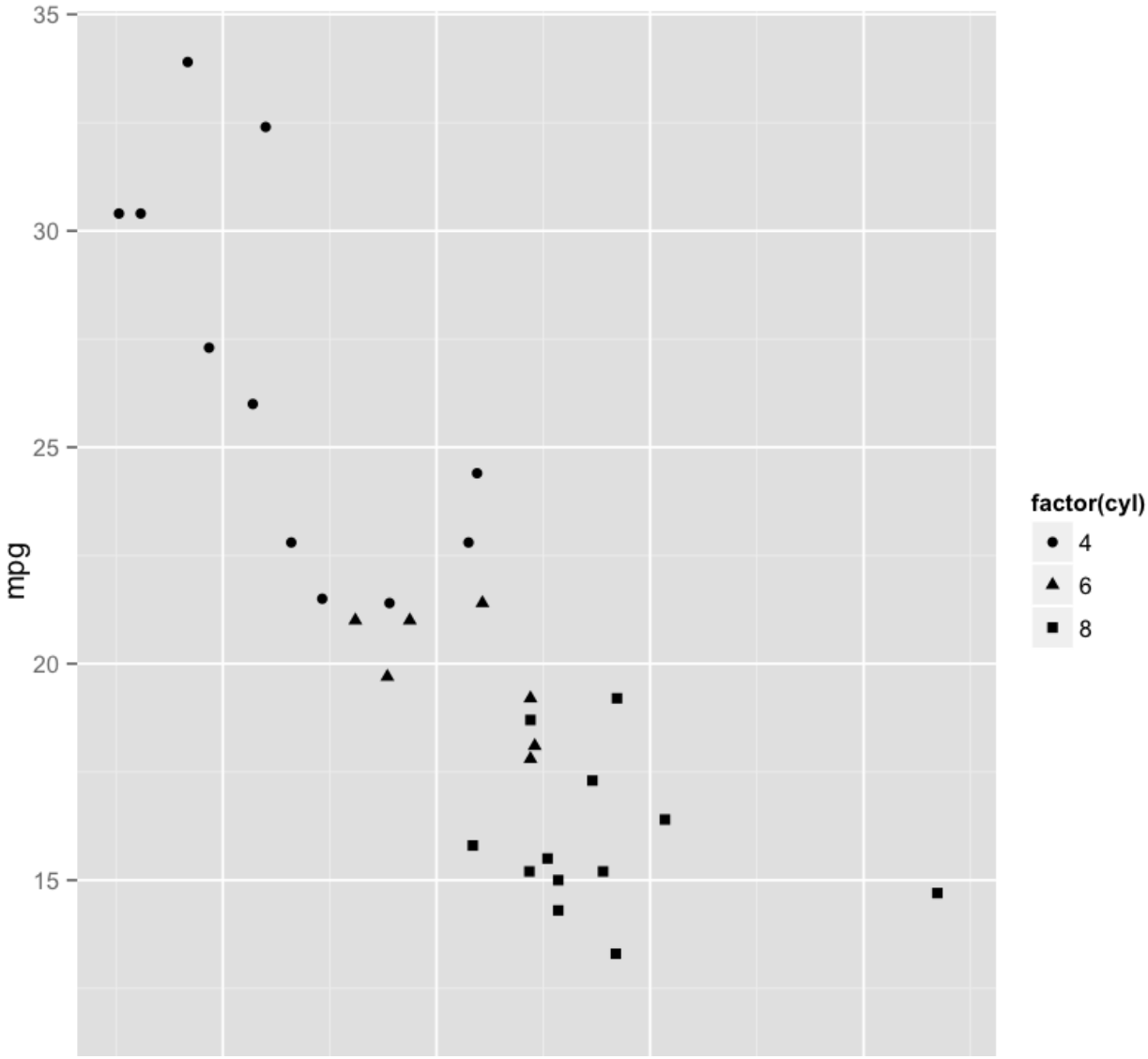
```
pl <- ggplot(data=df,aes(x = wt,y=mpg))

pl + geom_point(aes(size=factor(cyl)))
```





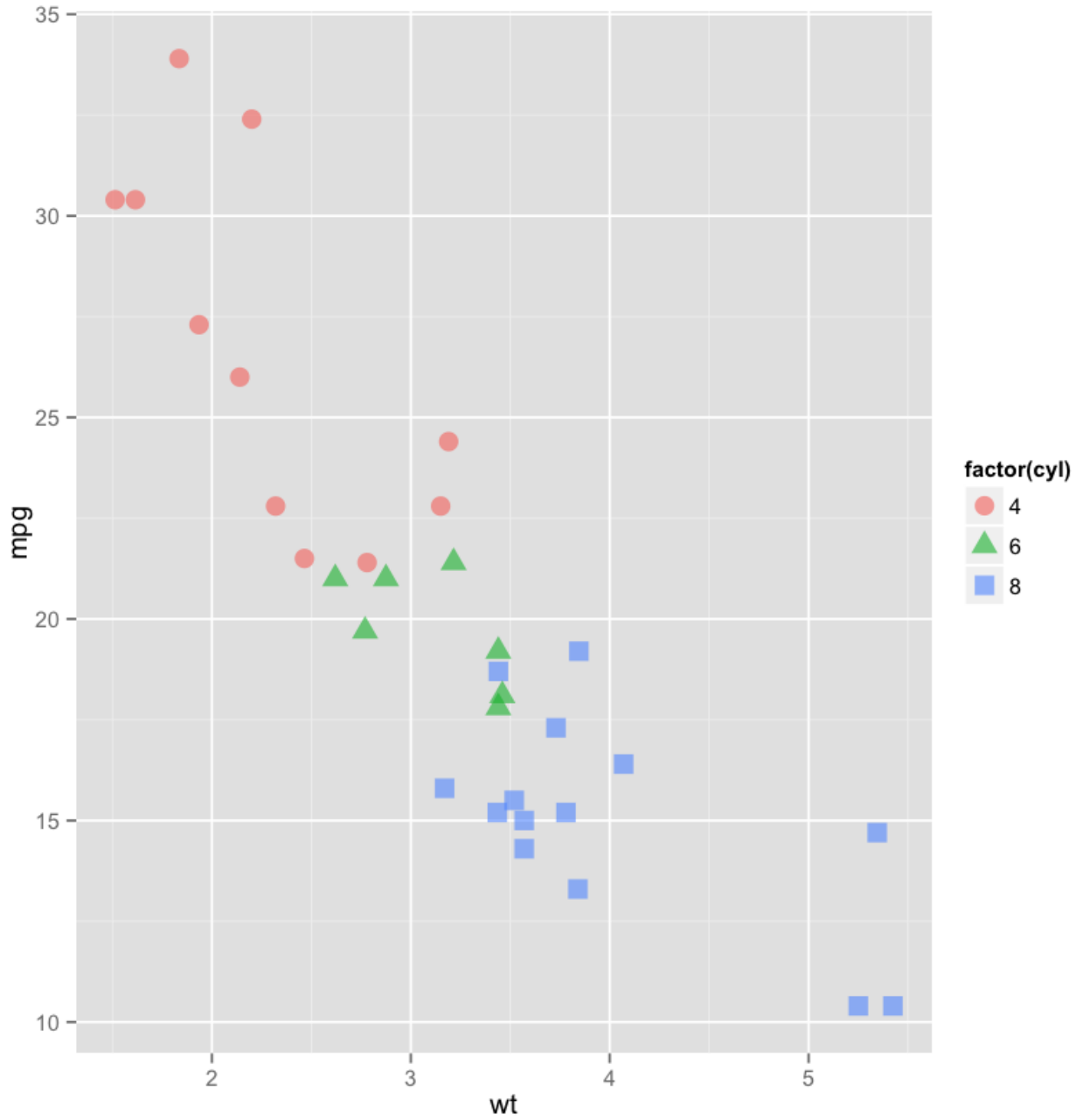
In [47]:

```
# With Shapes
pl <- ggplot(data=df,aes(x = wt,y=mpg))

pl + geom_point(aes(shape=factor(cyl)))
```



In [49]:

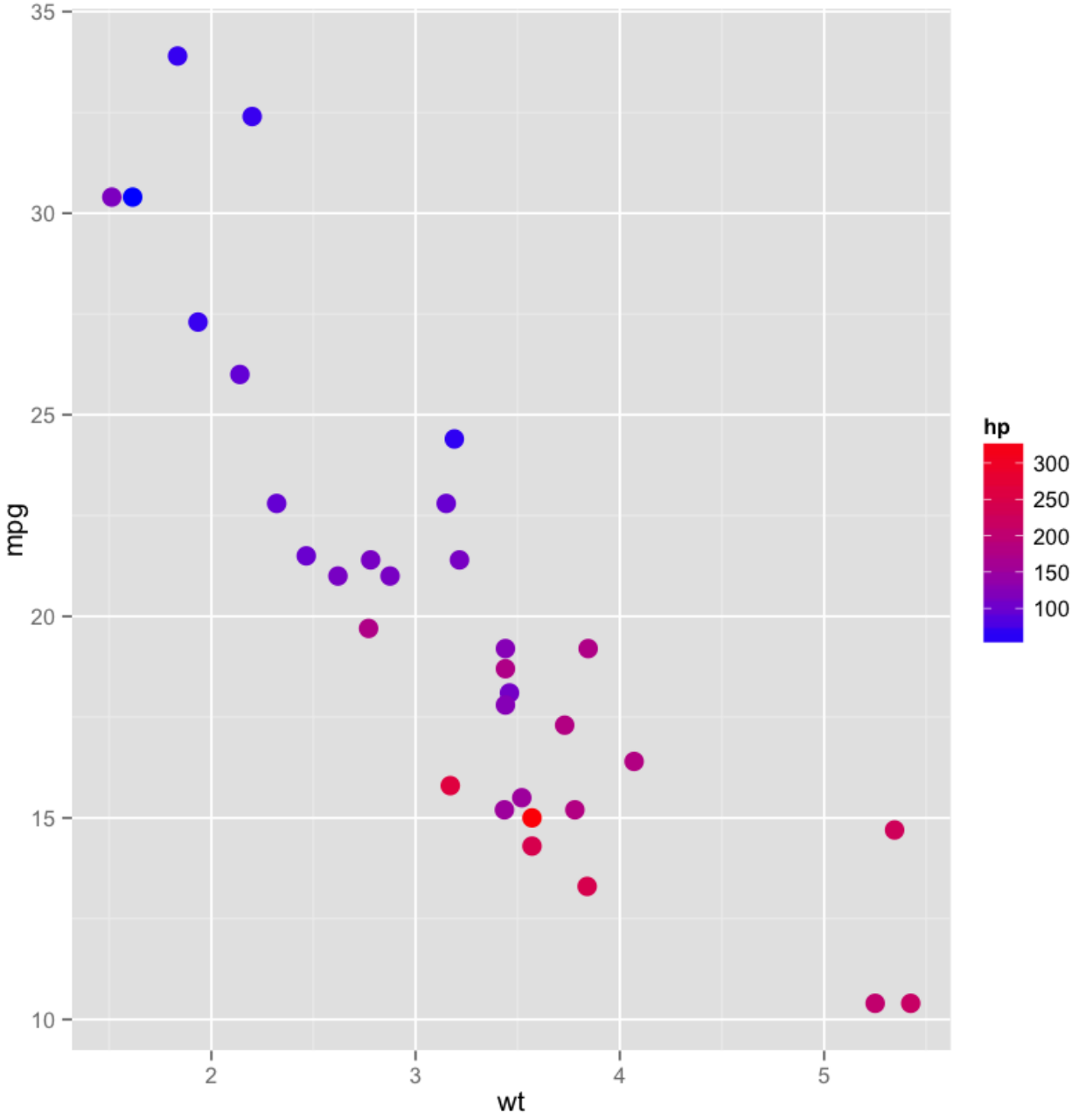
```
# Better version
# With Shapes
pl <- ggplot(data=df,aes(x = wt,y=mpg))

pl + geom_point(aes(shape=factor(cyl),color=factor(cyl)),size=4,alpha=0.6)
```



Gradient Scales

In [50]:

```
pl + geom_point(aes(colour = hp),size=4) + scale_colour_gradient(high='red',low = "blue")
```


35
30
25
20
15
10
mpg
2
3
4
5
wt
hp
300
250
200
150
100

Histograms with ggplot2

Let's go over how to create histograms with ggplot2. Refer to the video for the full explanation! Also a quick note, we are going to be showing a lot of what ggplot *can* do, but **not** what you *should* do!

Load Data

We'll use the movie dataset that comes with ggplot:

In [17]:

```
library(ggplot2)
df <- movies <- movies[sample(nrow(movies), 1000), ]
```

In [18]:

```
head(df)
```

Out[18]:

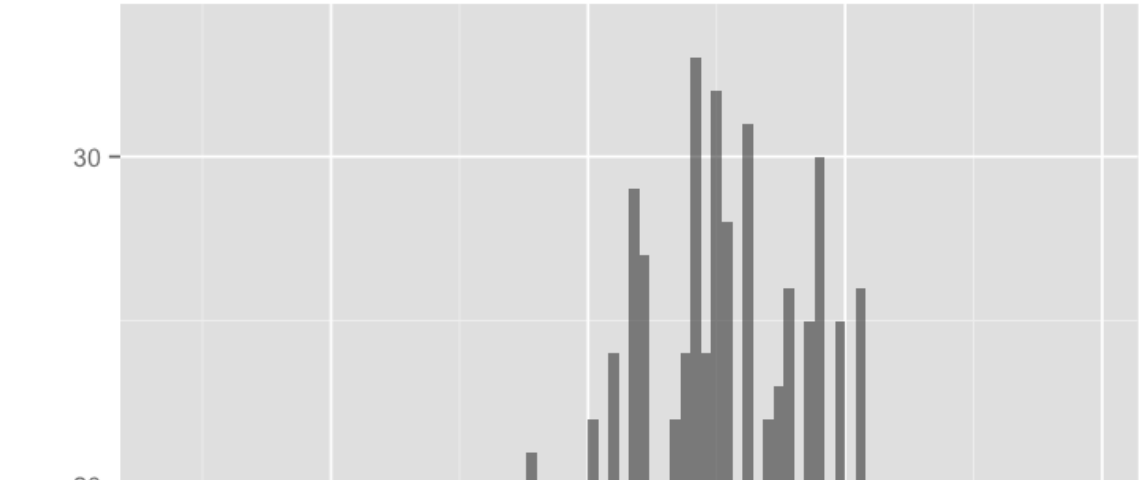
	title	year	length	budget	rating	votes	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	mpaa	Actio
21569	Gunan il guerriero	1983	81	NA	2	38	44.5	14.5	4.5	14.5	14.5	0	4.5	4.5	4.5	4.5		1
39820	Phantom Brother	1988	92	NA	2	10	45.5	14.5	0	0	0	24.5	14.5	0	0	14.5		0
47662	Snow	1996	81	NA	7.2	10	0	0	14.5	0	0	44.5	14.5	24.5	0	24.5		0
41080	Prelude	2000	6	NA	6.3	18	4.5	0	0	14.5	4.5	0	24.5	24.5	4.5	24.5		0
58741	Zwaarmoedige verhalen voor bij de centrale verwarming	1975	95	NA	5	27	14.5	0	0	4.5	14.5	14.5	14.5	14.5	14.5	24.5		0
47430	Sleepy-Time Tom	1951	7	NA	6.7	32	0	0	0	14.5	4.5	4.5	24.5	14.5	24.5	4.5		0

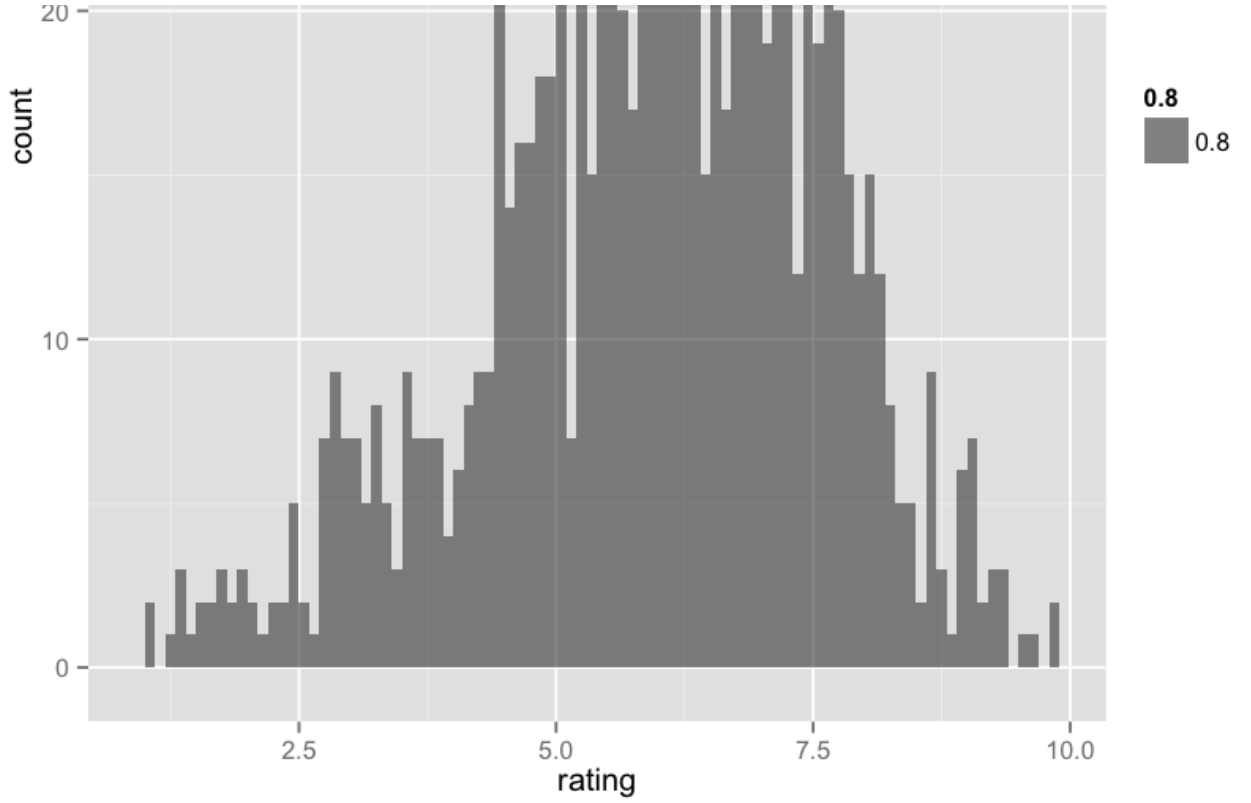
Using qplot()

Basics

In [21]:

```
qplot(rating,data=df,geom='histogram',binwidth=0.1,alpha=0.8)
```





Using ggplot()

Let's see how we can really expand on this by using ggplot! They syntax starts off with the base plot:

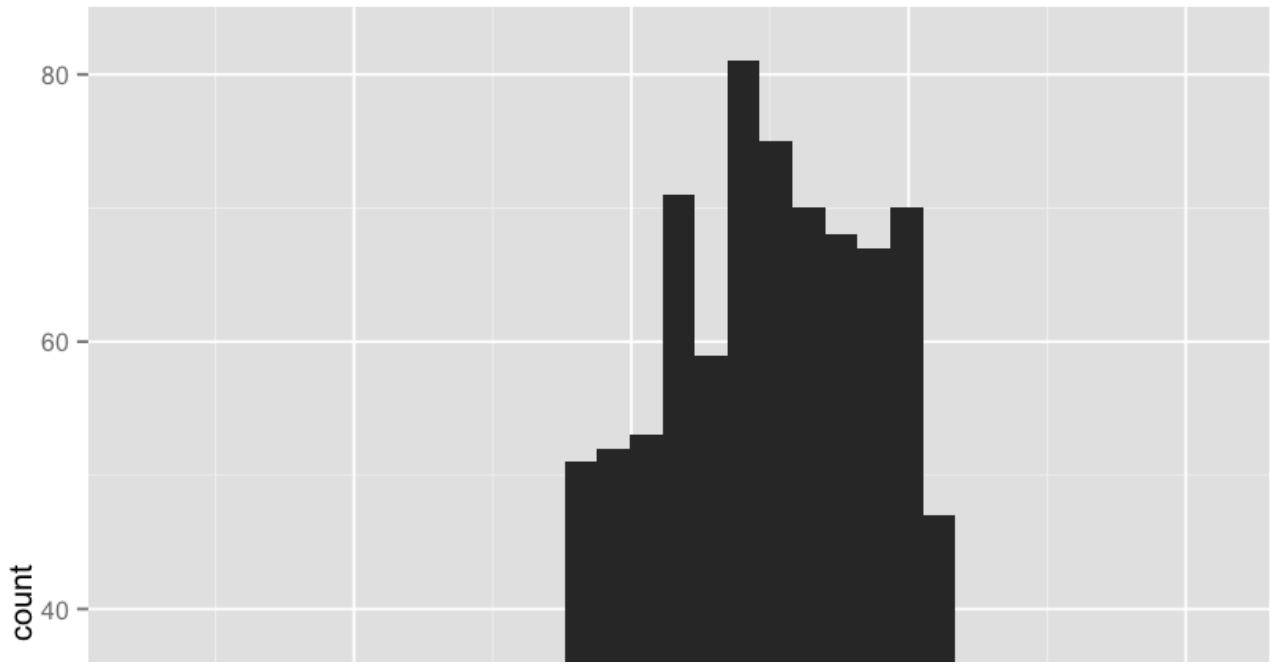
In [23]:

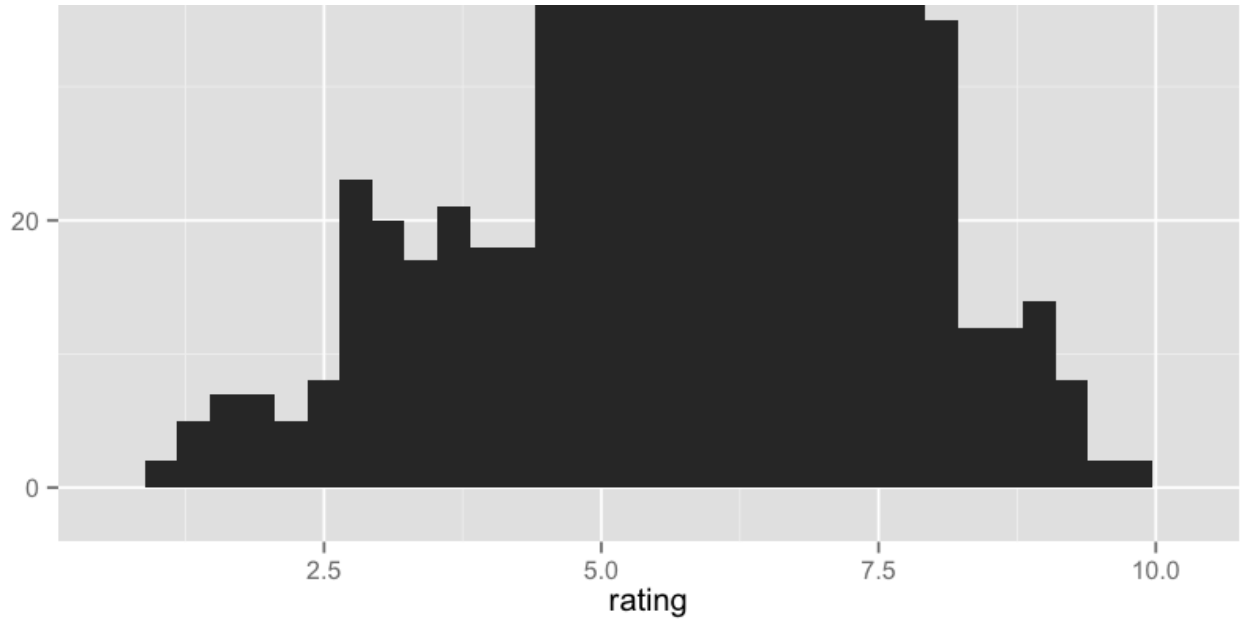
```
# ggplot(data, aesthetics)
pl <- ggplot(df,aes(x=rating))
```

In [24]:

```
# Add Histogram Geometry
pl + geom_histogram()
```

```
stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
```

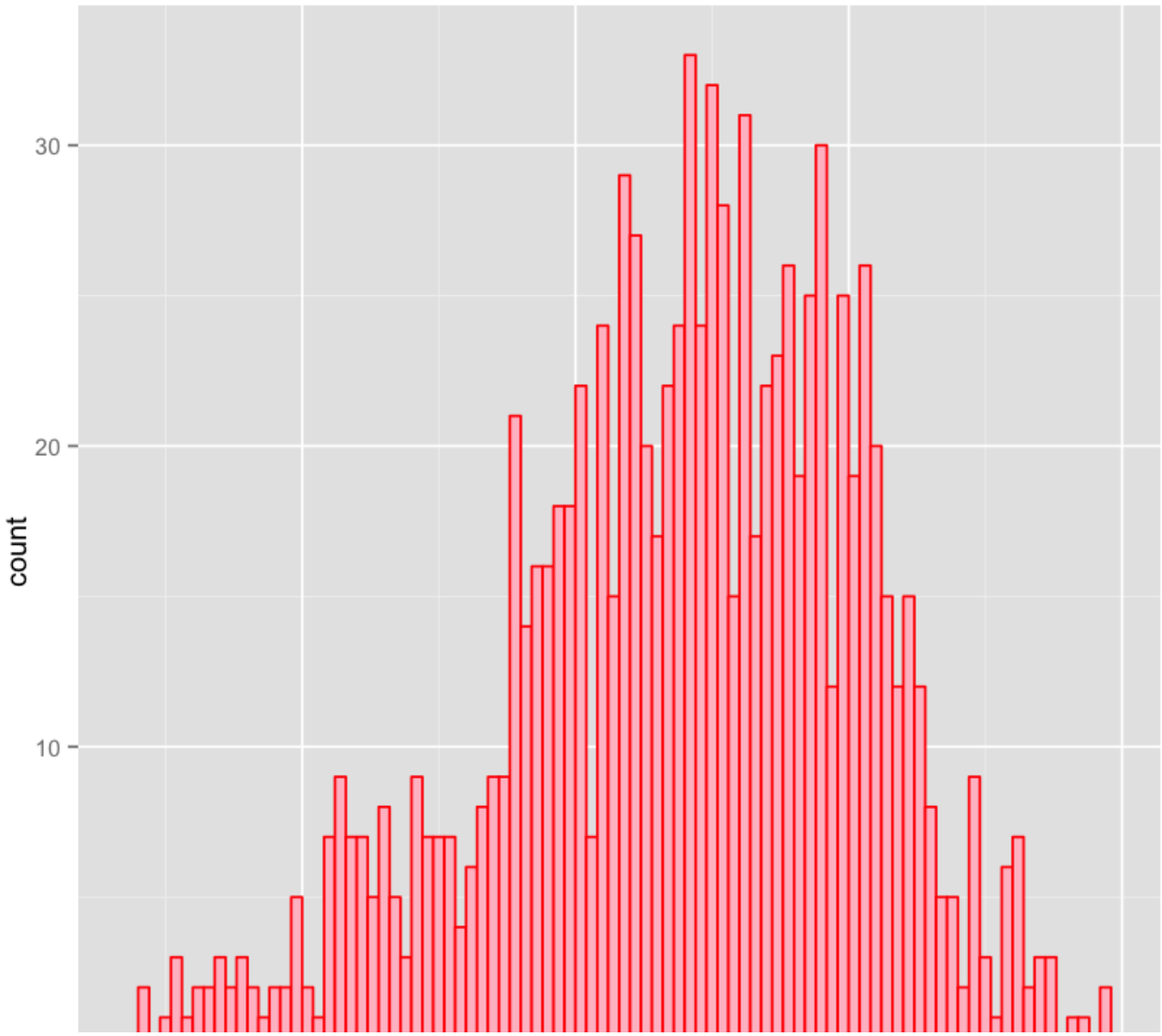


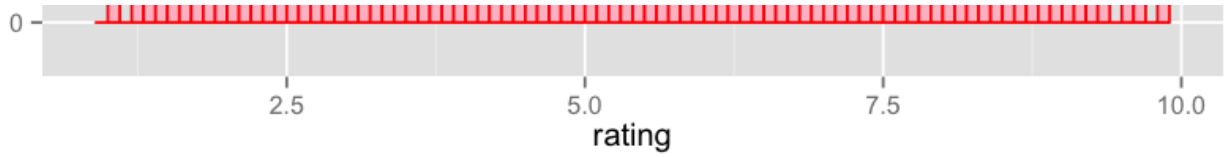


Adding Color

In [41]:

```
pl <- ggplot(df,aes(x=rating))
pl + geom_histogram(binwidth=0.1,color='red',fill='pink')
```

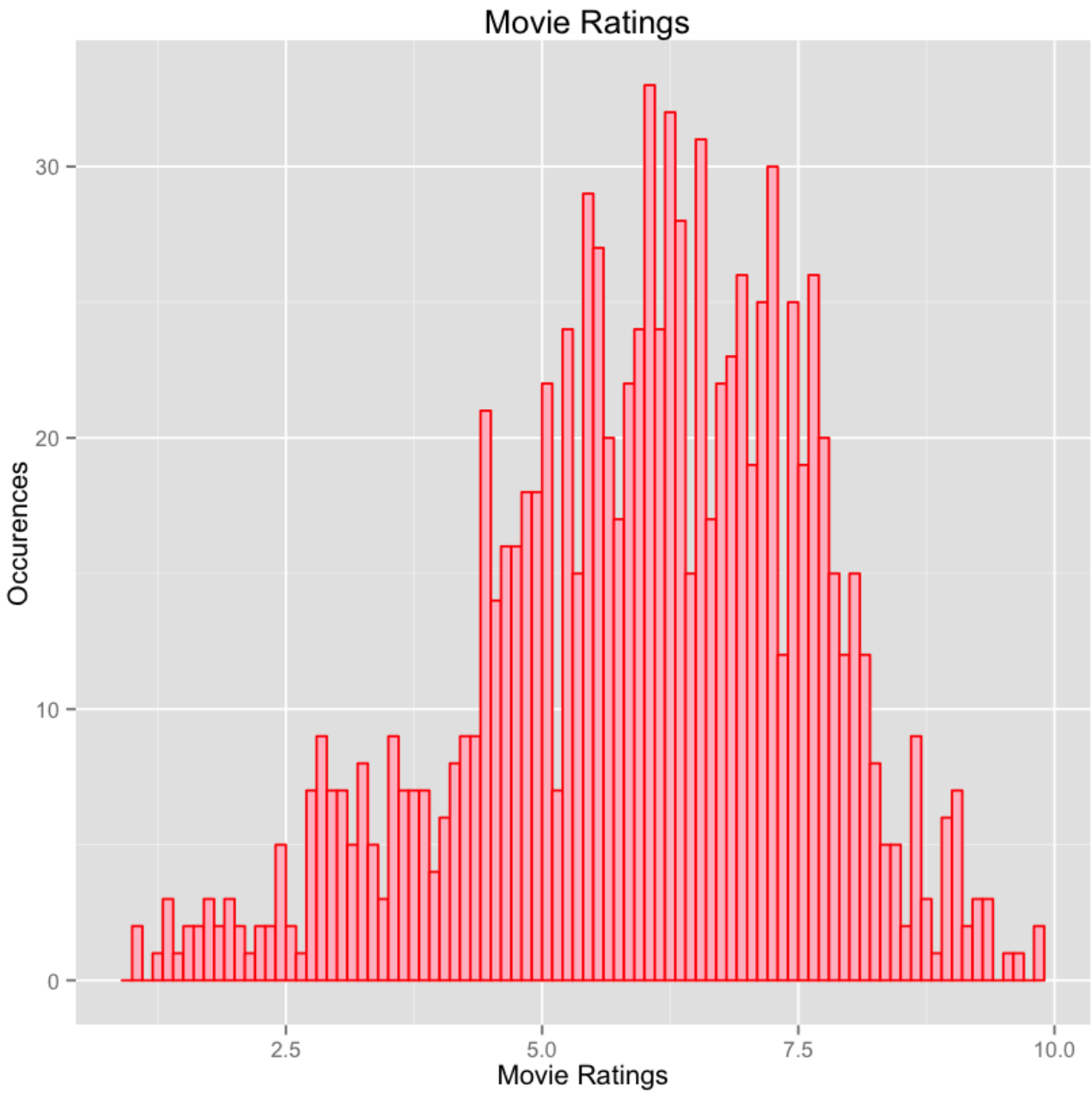




Adding Labels

In [65]:

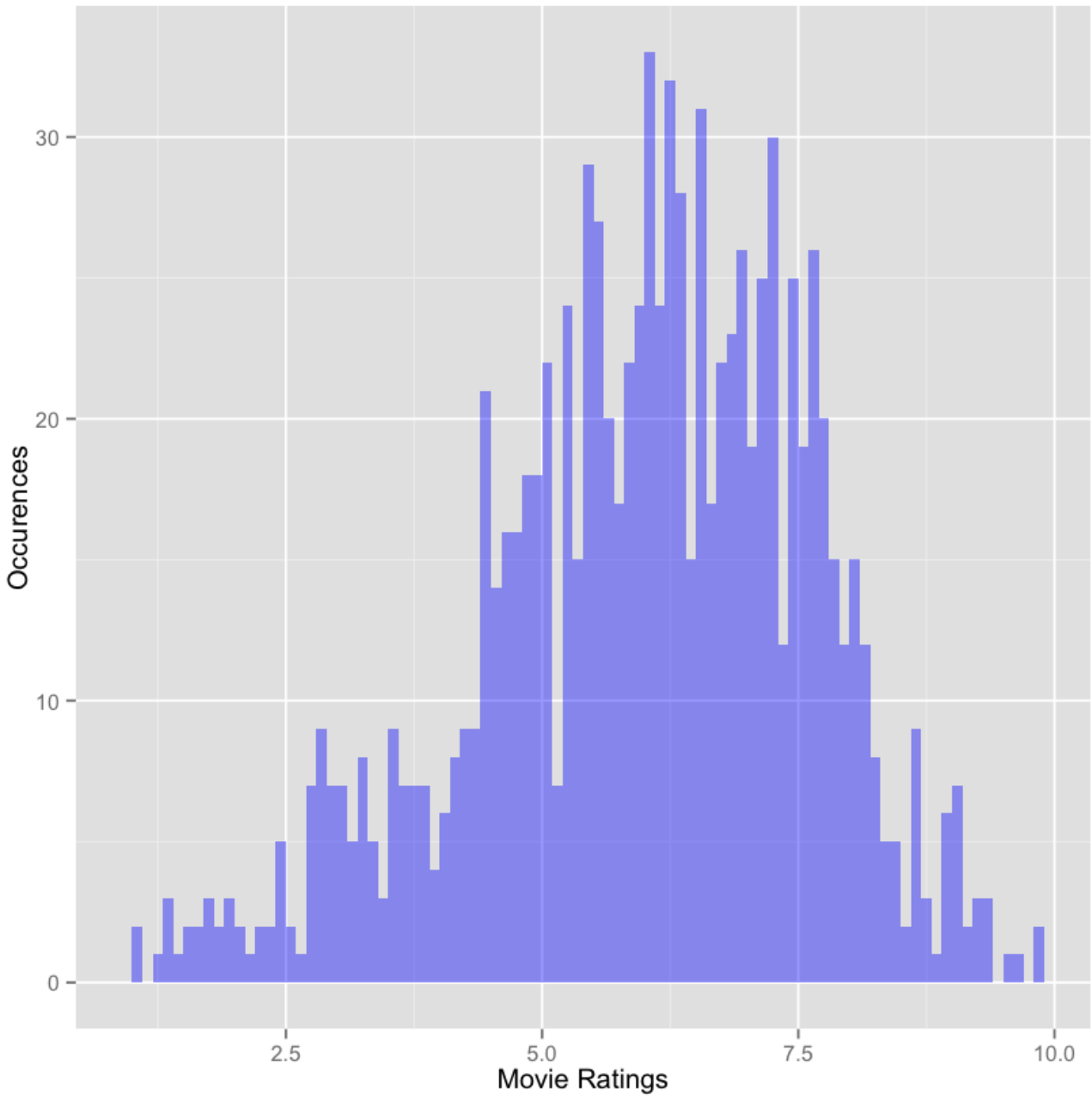
```
pl <- ggplot(df,aes(x=rating))
pl + geom_histogram(binwidth=0.1,color='red',fill='pink') + xlab('Movie Ratings')+ ylab('Occurences
') + ggtitle(' Movie Ratings')
```



Change Alpha (Transparency)

In [49]:

```
pl <- ggplot(df,aes(x=rating))
pl + geom_histogram(binwidth=0.1,fill='blue',alpha=0.4) + xlab('Movie Ratings')+ ylab('Occurences')
```

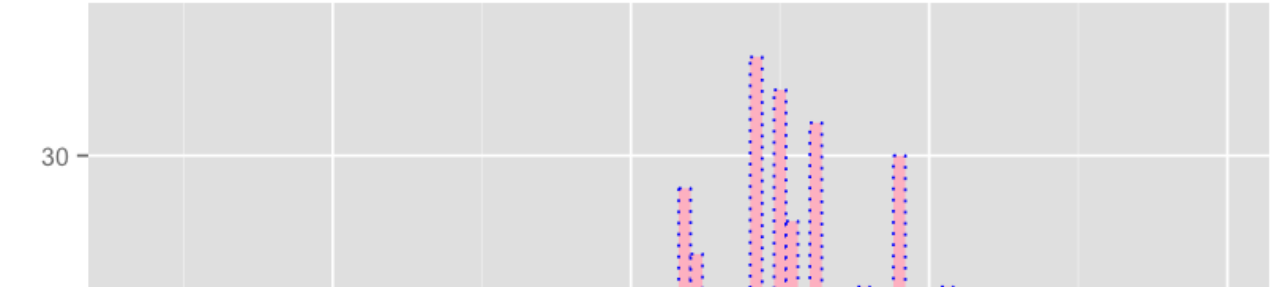


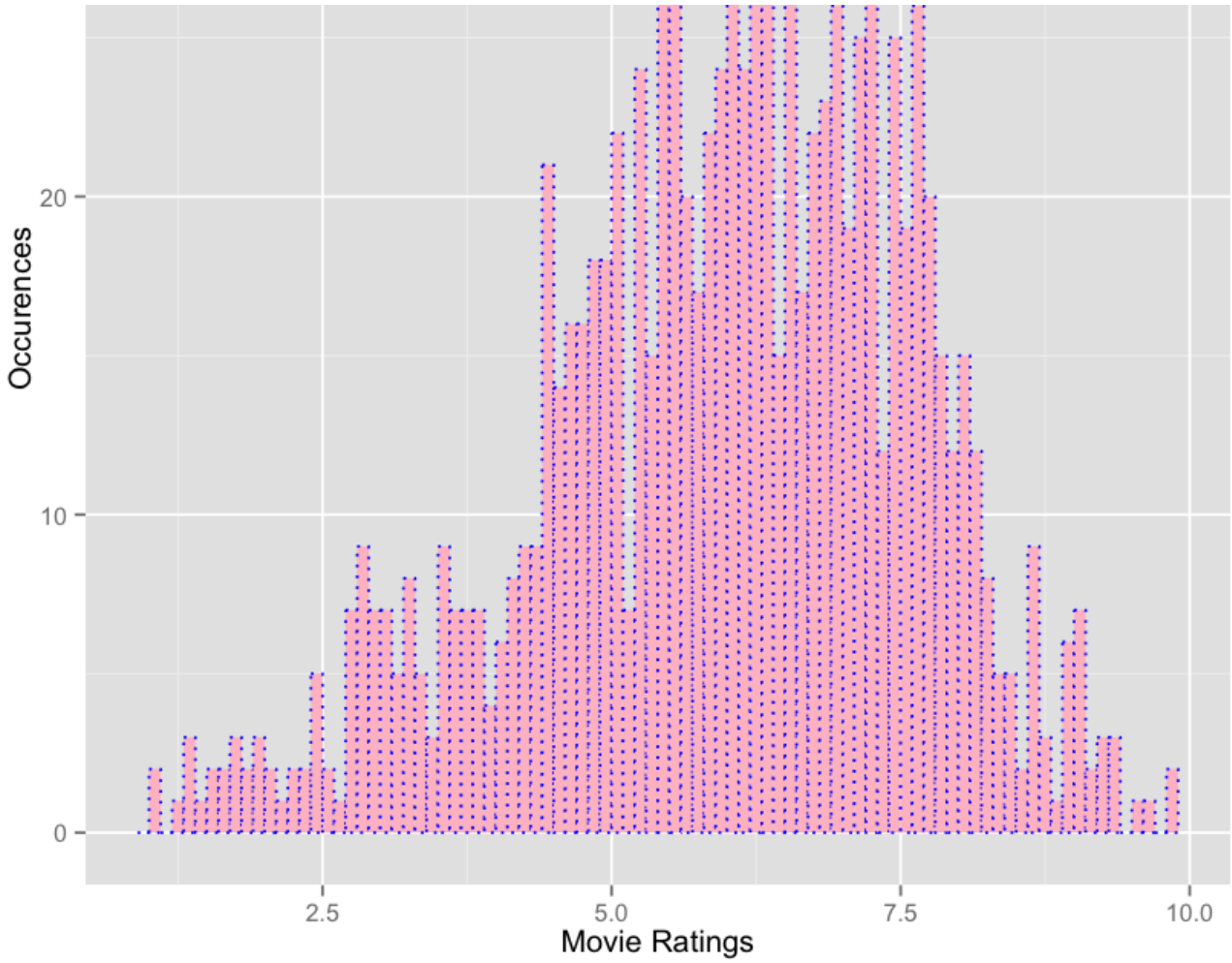
Linetypes

We have the options: "blank", "solid", "dashed", "dotted", "dotdash", "longdash", and "twodash". You would never really use these with a histogram, but just to show your options:

In [52]:

```
pl <- ggplot(df,aes(x=rating))
pl + geom_histogram(binwidth=0.1,color='blue',fill='pink',linetype='dotted') + xlab('Movie Ratings'
)+ ylab('Occurences')
```



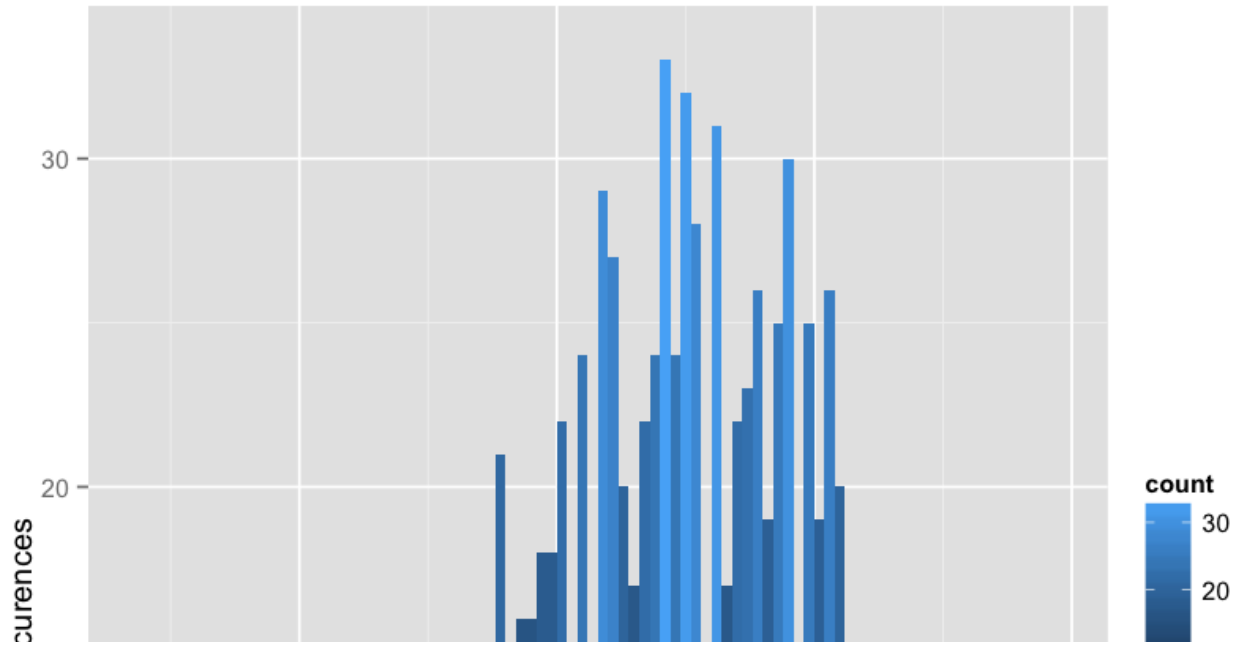


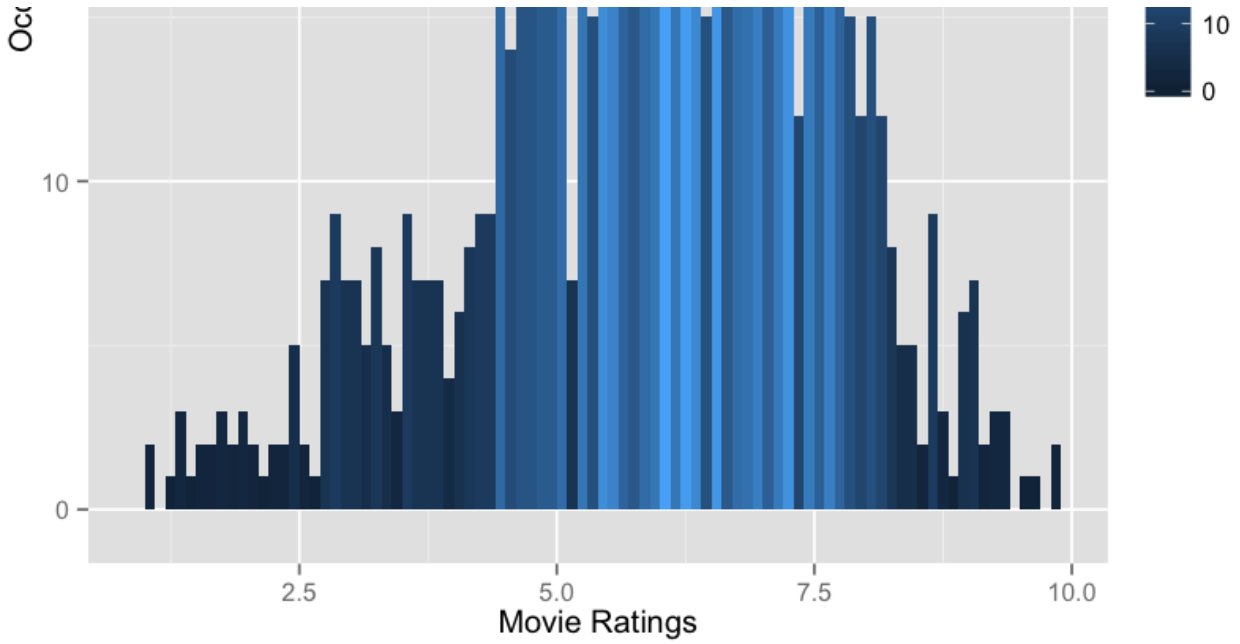
Advanced Aesthetics

We can add a **aes()** argument to the geom_histogram for some more advanced features. We won't go too deep into these, but ggplot gives you the ability to edit color and fill scales.

In [57]:

```
# Adding Labels
pl <- ggplot(df,aes(x=rating))
pl + geom_histogram(binwidth=0.1,aes(fill=..count..)) + xlab('Movie Ratings')+ ylab('Occurences')
```





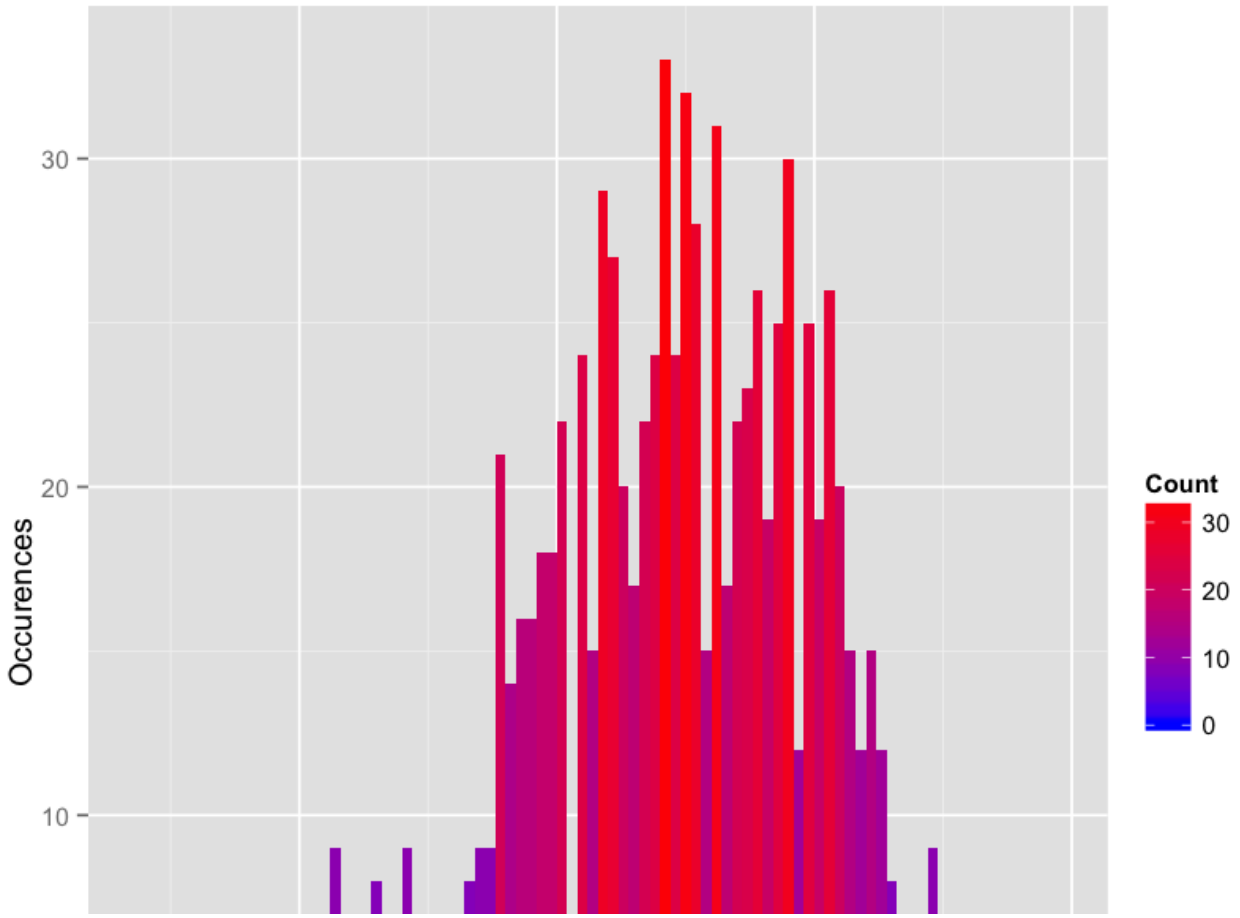
You can further edit this by adding the **scale_fill_gradient()** function to your ggplot objects:

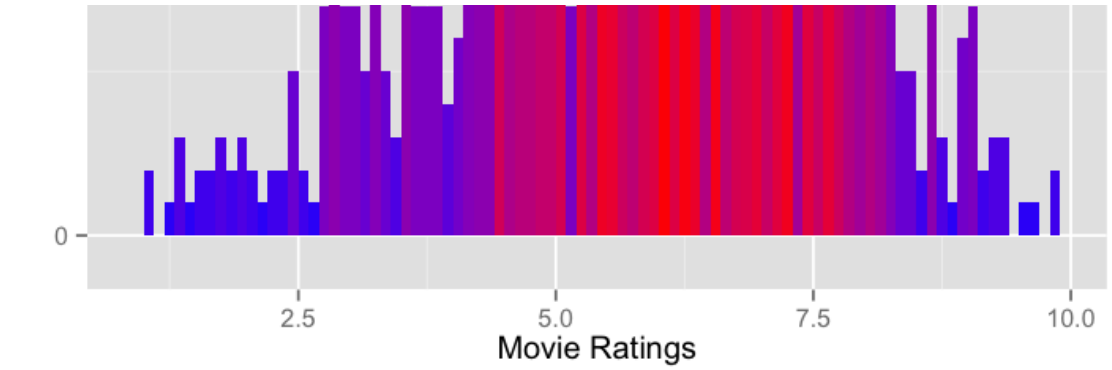
In [63]:

```
# Adding Labels
pl <- ggplot(df,aes(x=rating))
pl2 <- pl + geom_histogram(binwidth=0.1,aes(fill=..count..)) + xlab('Movie Ratings')+ ylab('Occuren
ces')
```

In [59]:

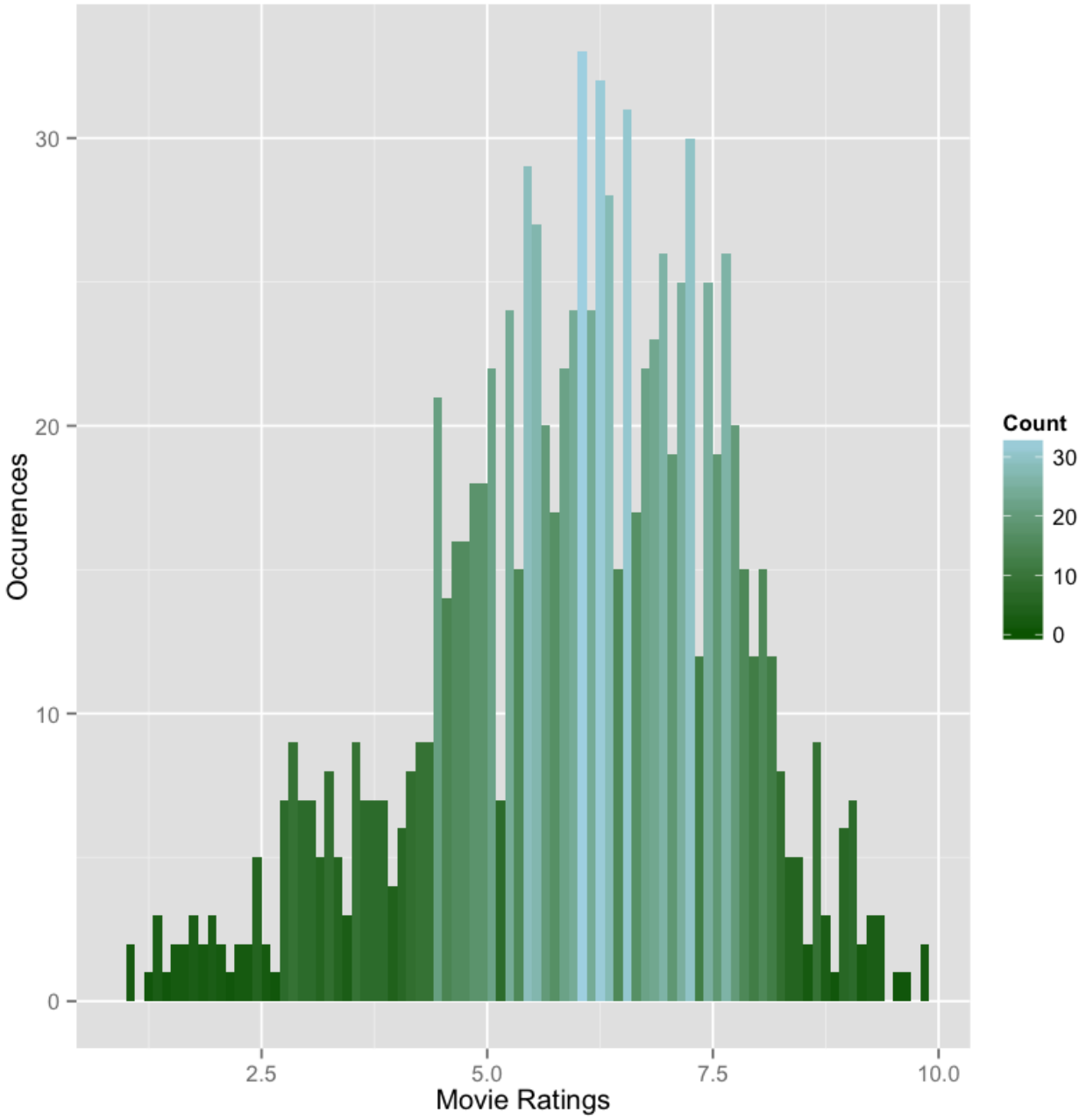
```
# scale_fill_gradient('Label',low=color1,high=color2)
pl2 + scale_fill_gradient('Count',low='blue',high='red')
```





In [62]:

```
# scale_fill_gradient('Label',low=color1,high=color2)
pl2 + scale_fill_gradient('Count',low='darkgreen',high='lightblue')
```



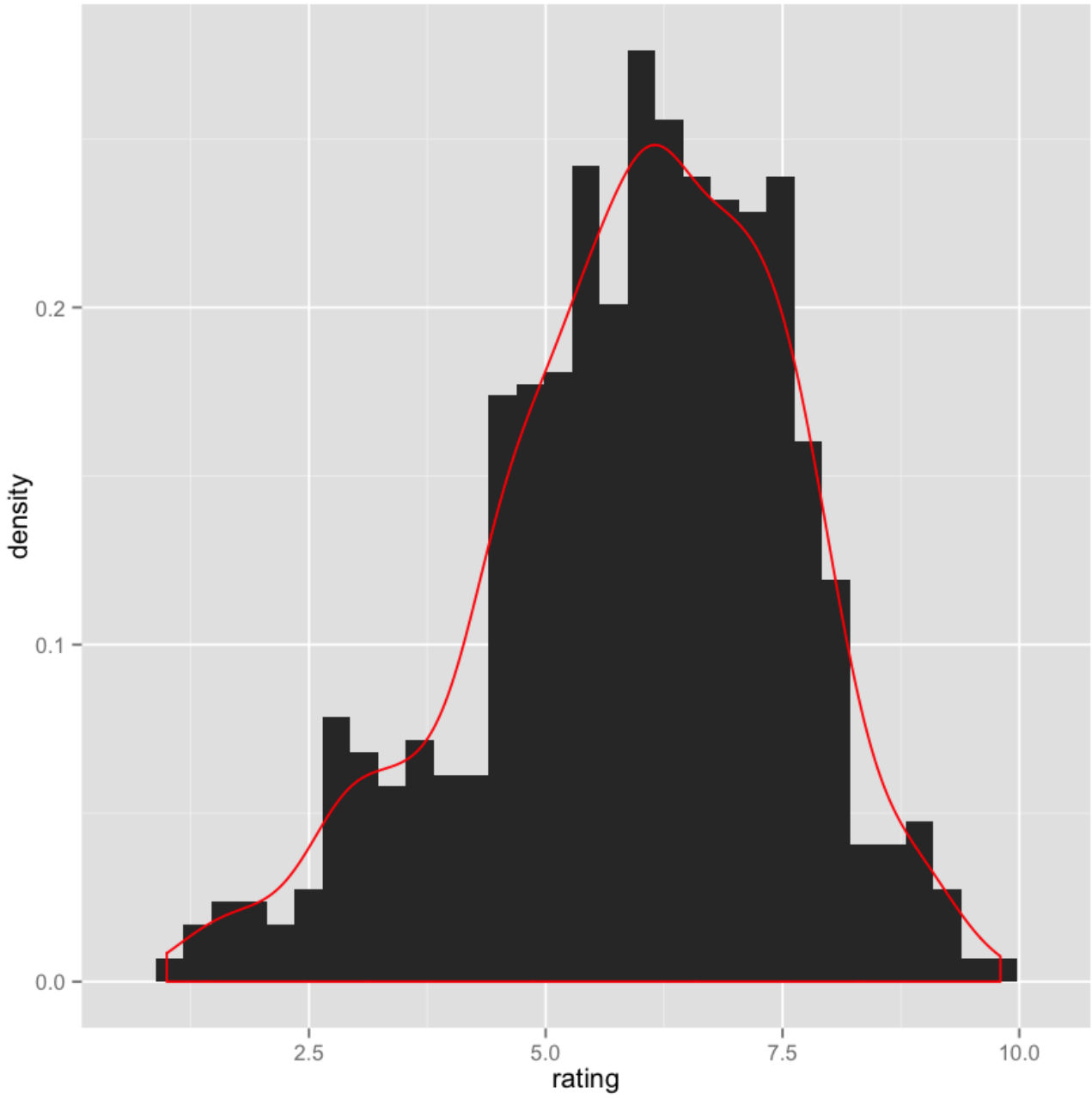
Adding density plot

You can add a kernel density estimation plot

In [68]:

```
# Adding Labels
pl <- ggplot(df,aes(x=rating))
pl + geom_histogram(aes(y=..density..)) + geom_density(color='red')
```

```
stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
```



Boxplots with ggplot2

Boxplots are convenient way of graphically depicting groups of numerical data through their quartiles. Box plots may also have lines extending vertically from the boxes (whiskers) indicating variability outside the upper and lower quartiles, hence the terms box-and-whisker plot and box-and-whisker diagram. Outliers may be plotted as individual points.

Let's see how we can create them with qplot and ggplot!

In [3]:

```
library(ggplot2)
df <- mtcars
```

In [4]:

```
head(df)
```

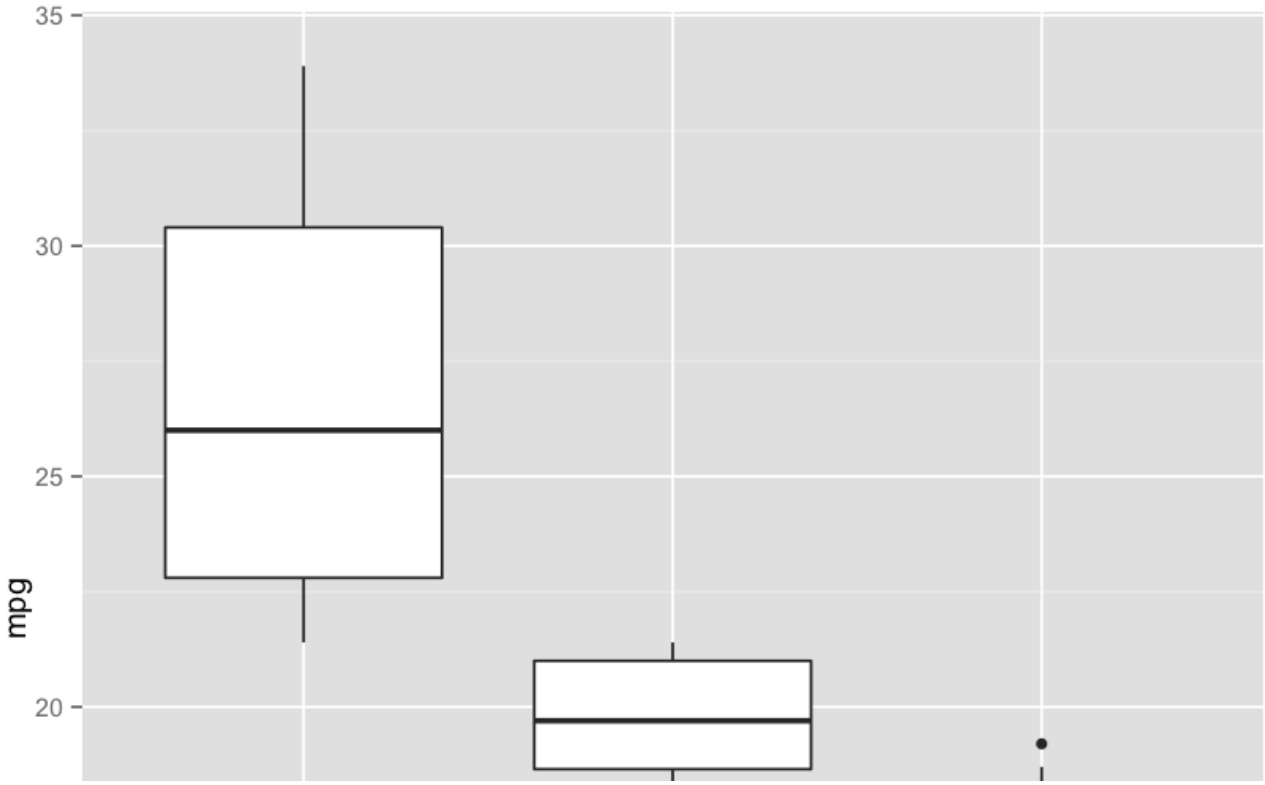
Out[4]:

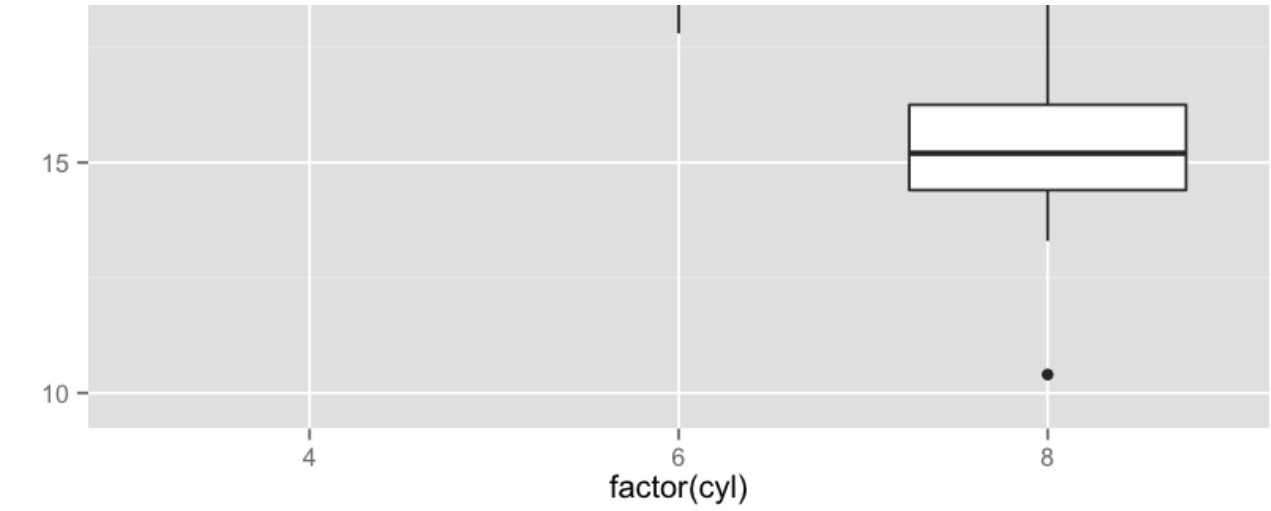
	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.62	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.32	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.44	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.46	20.22	1	0	3	1

qplot

In [10]:

```
qplot(factor(cyl), mpg, data = mtcars, geom = "boxplot")
```

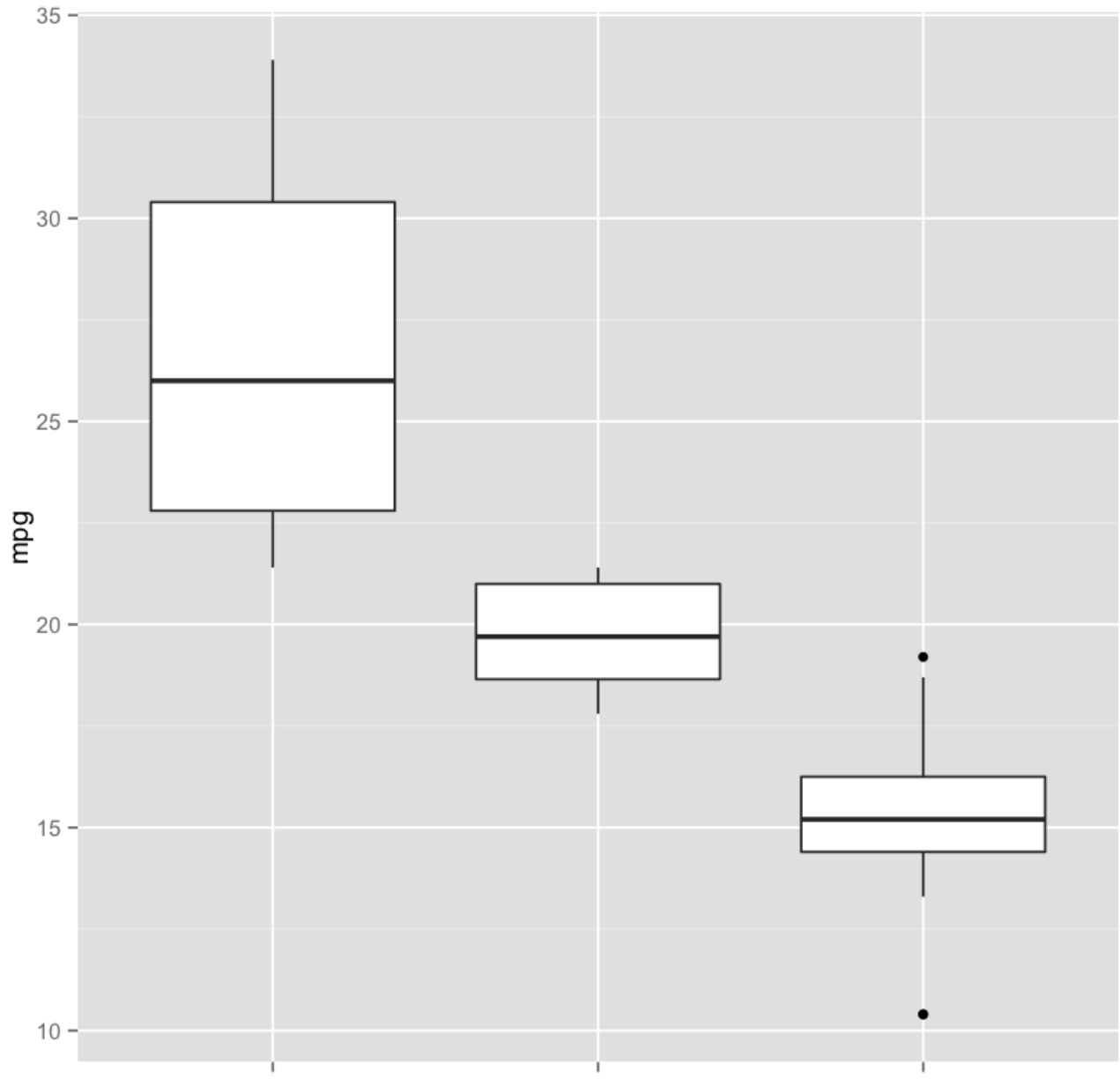


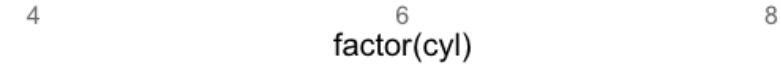


ggplot

In [11]:

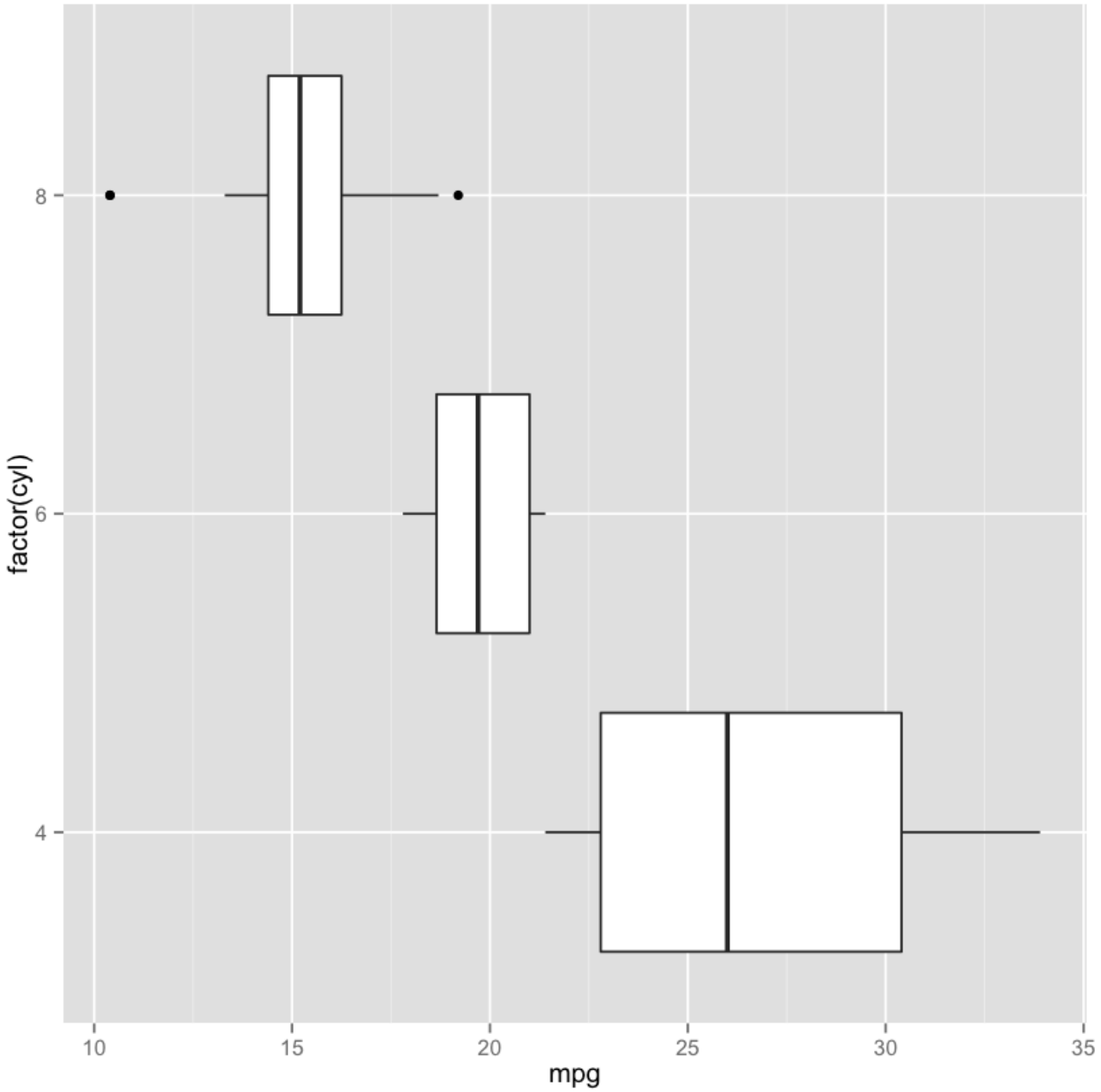
```
pl <- ggplot(mtcars, aes(factor(cyl), mpg))

pl + geom_boxplot()
```





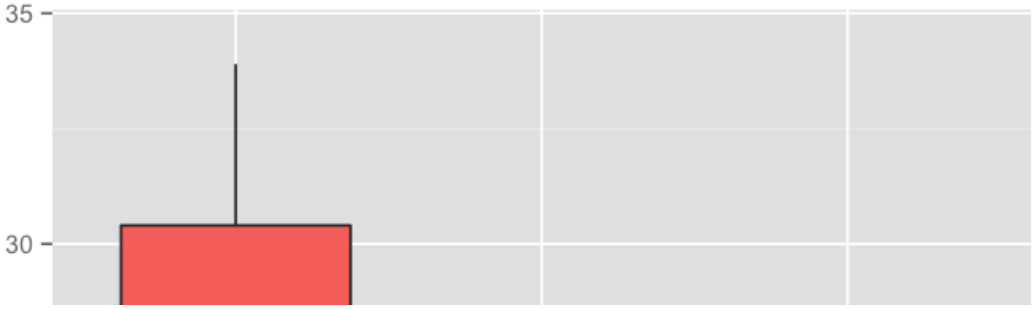
In [13]:

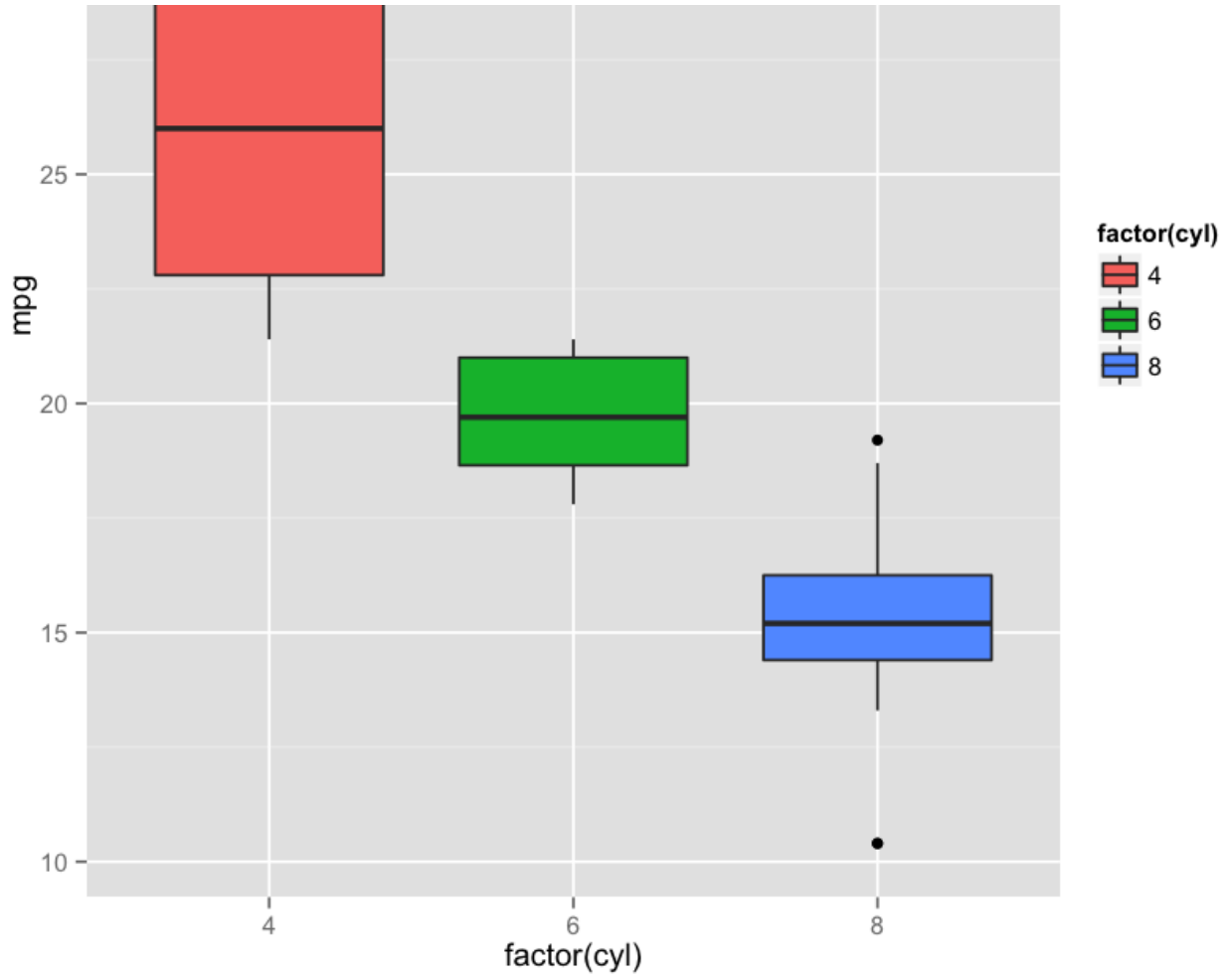
```
pl + geom_boxplot() + coord_flip()
```



In [14]:

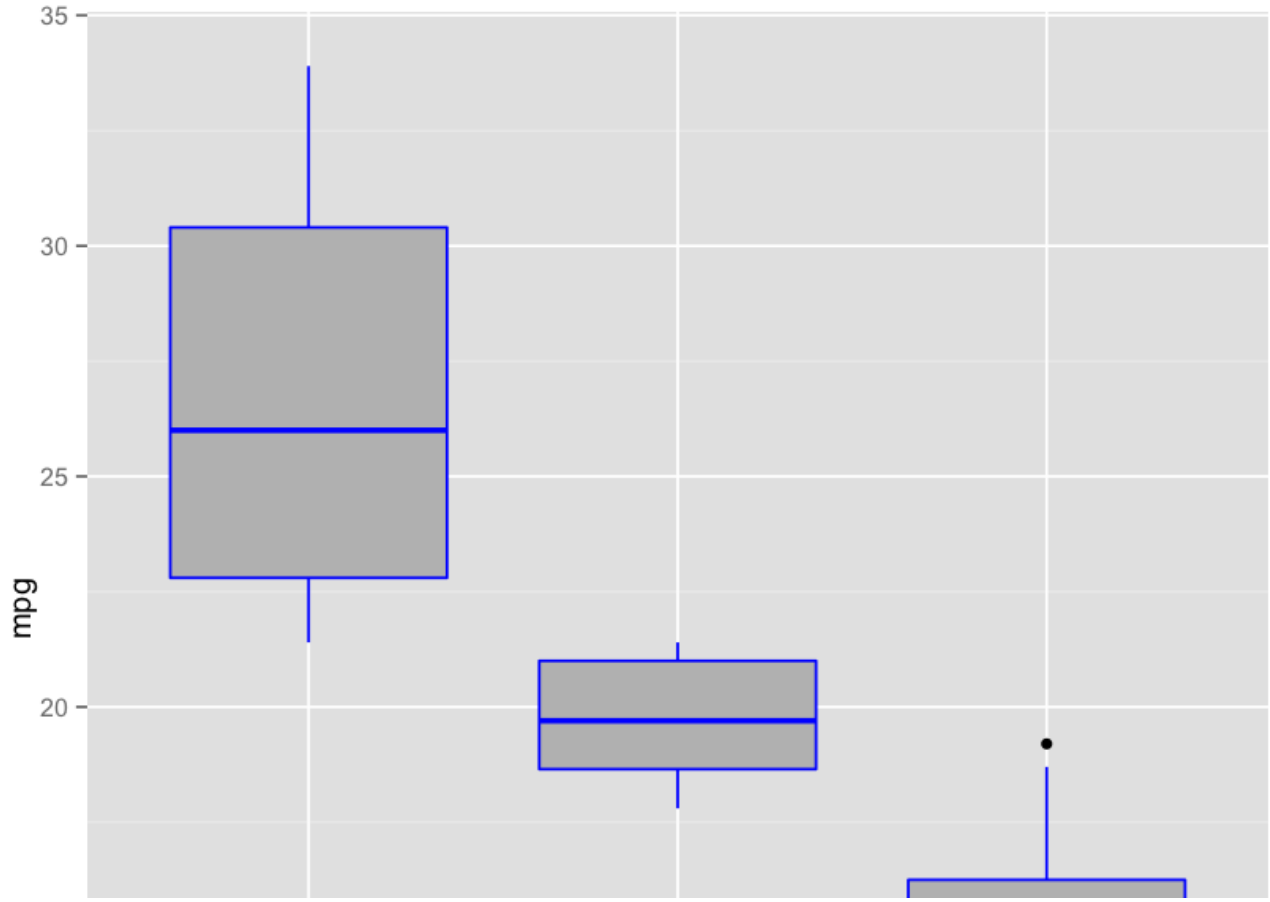
```
pl + geom_boxplot(aes(fill = factor(cyl)))
```

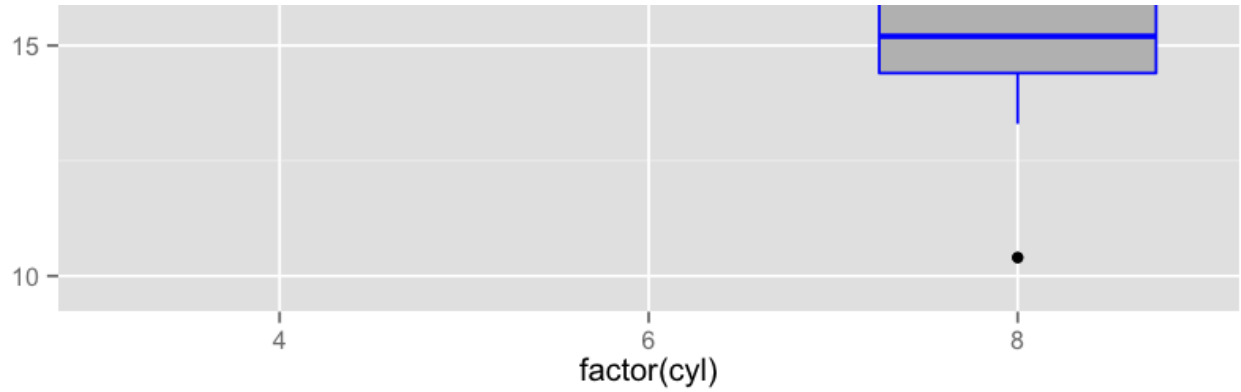




In [16]:

```
pl + geom_boxplot(fill = "grey", color = "blue")
```





That's it for the basics of creating boxplots in ggplot2!

3 Linear Algebra and Statistics-Part-1

Linear Algebra for Data Science

- *What is Linear Algebra?*
- *Why Linear Algebra is important in Data Science?*
- *How Linear Algebra is applied in Data Science?*

What is Linear Algebra?

Linear algebra is the branch of mathematics concerning linear equations, linear functions and their representations through matrices and vector spaces.

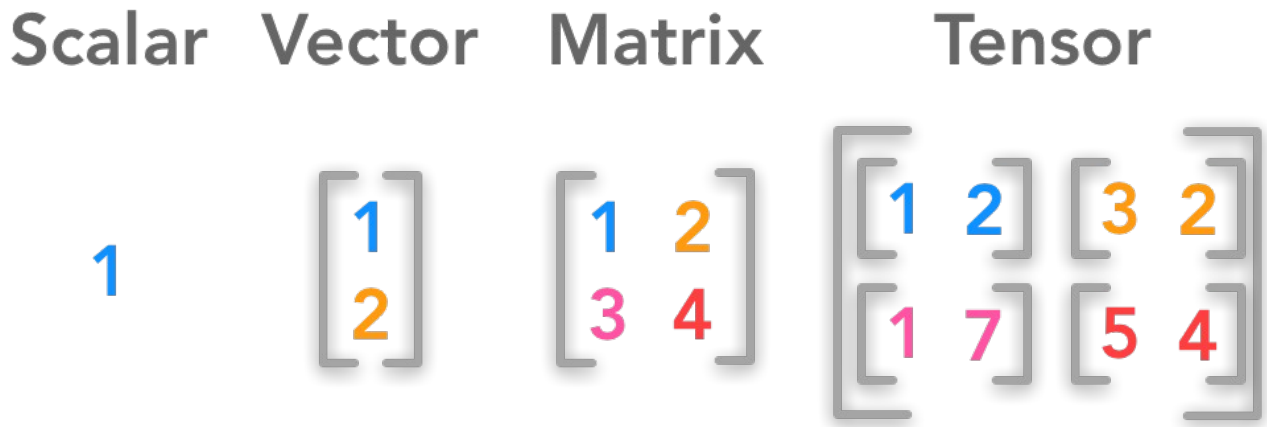
Why Linear Algebra is significant in Data Science?

Linear Algebra is central to almost all areas of mathematics like geometry and functional analysis. Its concepts are a crucial prerequisite for understanding the theory behind Data Science. You don't need to understand Linear Algebra before getting started in Data Science, but at some point, you may want to gain a better understanding of how the different algorithms really work under the hood. So if you really want to be a professional in this field, you will have to master the parts of Linear Algebra that are important for Data Science.

How Linear Algebra is used in Data Science?

- A *vector* is an array of numbers.
- A *scalar* is a single number
- A *matrix* is a 2-D array

- A *tensor* is a n-dimensional array with n>2



Find the mean, median, mode, and range for the following list of values:

13, 18, 13, 14, 13, 16, 14, 21, 13

The mean is the usual average, so I'll add and then divide:

(13 + 18 + 13 + 14 + 13 + 16 + 14 + 21 + 13) ÷ 9 = 15

Note that the mean, in this case, isn't a value from the original list. This is a common result. You should not assume that your mean will be one of your original numbers.

The median is the middle value, so first I'll have to rewrite the list in numerical order:

13, 13, 13, 13, 14, 14, 16, 18, 21

There are nine numbers in the list, so the middle one will be the (9 + 1) ÷ 2 = 10 ÷ 2 = 5th number:

13, 13, 13, 13, 14, 14, 16, 18, 21

So the median is 14.

The mode is the number that is repeated more often than any other, so 13 is the mode.

The largest value in the list is 21, and the smallest is 13, so the range is 21 − 13 = 8.

mean: 15
median: 14
mode: 13
range: 8

Variance And Standard Deviation

Measures of central tendency (mean, median and mode) provide information on the data values at the centre of the data set. Measures of dispersion (quartiles, percentiles, ranges) provide information on the spread of the data around the centre. In this section we will look at two more measures of dispersion called the **variance** and the **standard deviation**.

Variance
Let a population consist of n

elements, $\{x1;x2;\ldots;xn\}$. Write the mean of the data as $\bar{x}$

The variance of the data is the average squared distance between the mean and each data value.

$$\sigma^2 = \frac{\Sigma(x-\mu)^2}{N}$$

Overview of how to calculate standard deviation

$$\mathrm{SD} = \sqrt{\frac{\sum |x - \bar{x}|^2}{n}}$$

where $\sum$ means "sum of", x is a value in the data set, μ is the mean of the data set, and N is the number of data points in the population. The standard deviation formula may look confusing, but it will make sense after we break it down. In the coming sections, we'll walk through a step-by-step interactive example. Here's a quick preview of the steps we're about to follow:

Step 1: Find the mean.
Step 2: For each data point, find the square of its distance to the mean.
Step 3: Sum the values from Step 2.
Step 4: Divide by the number of data points.
Step 5: Take the square root.

Correlation

Correlation is a statistical measure that indicates the extent to which two or more variables fluctuate together. A positive correlation indicates the extent to which those variables increase or decrease in parallel; a negative correlation indicates the extent to which one variable increases as the other decreases.

A correlation coefficient is a statistical measure of the degree to which changes to the value of one variable predict change to the value of another. When the fluctuation of one variable reliably predicts a similar fluctuation in another variable, there's often a tendency to think that means that the change in one causes the change in the other. However, correlation does not imply causation. There may be, for example, an unknown factor that influences both variables similarly.

Here's one example: A number of studies report a positive correlation between the amount of television children watch and the likelihood that they will become bullies. Media coverage often cites such studies to suggest that watching a lot of television causes children to become bullies. However, the studies only report a correlation, not causation. It is likely that some other factor – such as a lack of parental supervision – may be the influential factor.

Mean, Median, Mode, Variance, Standard Deviation

Mean:

Calculate sum of all the values and divide it with the total number of values in the data set.

In [1]:

```
x <- c(1,2,3,4,5,1,2,3,1,2,4,5,2,3,1,1,2,3,5,6)
mean.result = mean(x)
print (mean.result)
```

[1] 2.8

Median:

The middle value of the data set.

In [2]:

```
x <- c(1,2,3,4,5,1,2,3,1,2,4,5,2,3,1,1,2,3,5,6)
median.result = median(x)
print (median.result)
```

```
[1] 2.5
```

Mode:

The most occurring number in the data set. For calculating mode, there is no default function in R. So, we have to create our own custom function.

In [4]:

```
x <- c(1,2,3,4,5,1,2,3,1,2,4,5,2,3,1,1,2,3,5,6)
mode.result = mode(x)
print (mode.result)
```

[1] 1

Variance

How far a set of data values are spread out from their mean.

In [7]:

```
x <- c(1,2,3,4,5,1,2,3,1,2,4,5,2,3,1,1,2,3,5,6)
variance.result = var(x)
print (variance.result)
```

```
[1] 15
```

The most occurring number in the data set. For calculating mode, there is no default function in R. So, we have to create our own custom function.

In [4]:

```
x <- c(1,2,3,4,5,1,2,3,1,2,4,5,2,3,1,1,2,3,5,6)
mode.result = mode(x)
print (mode.result)
```

[1] 1

Standard Deviation

Standard Deviation:

A measure that is used to quantify the amount of variation or dispersion of a set of data values.

In [8]:

```
x <- c(1,2,3,4,5,1,2,3,1,2,4,5,2,3,1,1,2,3,5,6)
sd.result = sqrt(var(x))
print (sd.result)
```

```
[1] "1.576138"
```

SOLVING LINEAR EQUATIONS

Goal: The goal of solving a linear equation is to find the value of the variable that will make the statement (equation) true.

Method: Perform operations to both sides of the equation in order to isolate the variable.

Addition and Subtraction Properties of Equality:
Let a, b, and c represent algebraic expressions.

1. **Addition property of equality:**
   If $a = b$,
   then $a + c = b + c$
2. **Subtraction property of equality:**
   If $a = b$,
   then $a - c = b - c$

Multiplication and Division Properties of Equality:
Let a, b, and c represent algebraic expressions.

1. **Multiplication property of equality:**
   If $a = b$,
   then $ac = bc$
2. **Division property of equality:**
   If $a = b$,
   then $\frac{a}{c} = \frac{b}{c}$ (provided $c \neq 0$)

Clearing Fractions or Decimals in an Equation:
When solving an equation with fractions or decimals, there is an option of clearing the fractions or decimals in order to create a simpler equation involving whole numbers.

1. To clear fractions, multiply both sides of the equation (distributing to all terms) by the LCD of all the fractions.
2. To clear decimals, multiply both sides of the equation (distributing to all terms) by the lowest power of 10 that will make *all* decimals whole numbers.

Steps for Solving a Linear Equation in One Variable:

1. Simplify both sides of the equation.
2. Use the addition or subtraction properties of equality to collect the variable terms on one side of the equation and the constant terms on the other.
3. Use the multiplication or division properties of equality to make the coefficient of the variable term equal to 1.
4. Check your answer by substituting your solution into the original equation.

Note: If when solving an equation, the variables are eliminated to reveal a *true* statement such as, $-13 = -13$, then the solution is **all real numbers.** This type of equation is called an **identity**. On the other hand, if the variables are eliminated to reveal a *false* statement such as, $-7 = 3$, then there is **no solution**. This type of equation is called a **contradiction**. All other linear equations which have only one solution are called **conditional**.

Examples:

A. $x + 5 = -2$
$\quad -5 \quad -5$
$x = -7$

Check:
$(-7) + 5 = -2$
$-2 = -2$ (Solution Checks)

B. $x - 3 = 7$
$\quad +3 \quad +3$
$x = 10$

Check:
$10 - 3 = 7$
$7 = 7$ (Solution Checks)

C. $3x = 24$

$\frac{3x}{3} = \frac{24}{3}$

$x = 8$

Check:

$3(8) = 24$

$24 = 24$ (Solution Checks)

D. $-7x = 28$

$\frac{-7x}{-7} = \frac{28}{-7}$

$x = -4$

Check:

$-7(-4) = 28$

$28 = 28$ (Solution Checks)

E. $\frac{2}{3}x = -4$

$\frac{3}{2} \cdot \frac{2}{3}x = \frac{-4}{1} \cdot \frac{3}{2}$ (Multiply by the reciprocal)

$x = -6$

Check:

$\frac{2}{3}\left(\frac{-6}{1}\right) = -4$

$-4 = -4$ (Solution Checks)

F. $\frac{y}{-5} = 2$

$\frac{-5}{1} \cdot \frac{y}{-5} = 2(-5)$

$y = -10$

Check:

$\frac{-10}{-5} = 2$

$2 = 2$ (Solution Checks)

Identity Example:

G. $2x + 6 = 3(x + 2) - x$

$2x + 6 = 3x + 6 - x$

$2x + 6 = 2x + 6$

$-2x \quad -2x$

$6 = 6$ True Statement

Solution: all real numbers

Contradiction Example:

H. $5x - 3 = 4(x + 2) + x$

$5x - 3 = 4x + 8 + x$

$5x - 3 = 5x + 8$

$-5x \quad -5x$

$-3 = 8$ False Statement

No Solution

I. $10 - 5x = 3(x - 4) - 2(x + 7)$

$10 - 5x = 3x - 12 - 2x - 14$

$10 - 5x = x - 26$

$-x \quad -x$

$10 - 6x = -26$

$-10 \quad -10$

$-6x = -36$

$\frac{-6x}{-6} = \frac{-36}{-6}$

$x = 6$

Check:

$10 - 5(6) = 3(6 - 4) - 2(6 + 7)$

$10 - 30 = 3(2) - 2(13)$

$-20 = 6 - 26$

$-20 = -20$ (Solution Checks)

J. $\frac{x-2}{5} - \frac{x-4}{2} = 2$ (LCD is 10)

$10\left(\frac{x-2}{5} - \frac{x-4}{2}\right) = 2(10)$

$\frac{10}{1} \cdot \frac{(x-2)}{5} - \frac{10}{1} \cdot \frac{(x-4)}{2} = 20$

$2(x-2) - 5(x-4) = 20$

$2x - 4 - 5x + 20 = 20$

$-3x + 16 = 20$

$-16 \quad -16$

$\frac{-3x}{-3} = \frac{4}{-3}$

$x = -\frac{4}{3}$

K. $0.05x + 0.25 = 0.2$

$100(0.05x + 0.25) = 0.2(100)$

$5x + 25 = 20$

$-25 \quad -25$

$\frac{5x}{5} = \frac{-5}{5}$

$x = -1$

Check:

$0.05(-1) + 0.25 = 0.2$

$-0.05 + 0.25 = 0.2$

$0.2 = 0.2$ (Solution Checks)

LINEAR EQUATIONS PRACTICE

1. $4x = 4$
2. $x + 6 = -7$
3. $x - 4 = 7$
4. $\frac{x}{3} = -9$
5. $2x + 4 = 8$
6. $14 = 3 + 2x$
7. $8x - 3 = -19$
8. $6 - x = 9$
9. $-x = -12$
10. $3(x - 2) = 6$
11. $-3(2x - 8) = -12$
12. $4(6 + 2x) = 0$
13. $3x + 2x + 6 = -15$
14. $4 = -2(x + 3)$
15. $27 = 46 + 2x - x$
16. $4x + 6 - 7x + 9 = 18$
17. $4 + 3(x + 2) = 10$
18. $-3 + 3x = -2(x + 1)$
19. $9x - 6 = -3x + 30$
20. $-(x + 2) = 2(3x - 6)$
21. $2x + 6 = 3x + 9 - 3$
22. $-5x + 3 = 2x + 10$
23. $3x - 12x = 24 - 9x$
24. $2(x + 4) = -3(x + 5)$
25. $4(2x - 3) + 4 = 8x - 8$
26. $6x + 11 = -(6x + 5)$
27. $2(x + 7) = 6x + 9 - 4x$
28. $-5(3 - 4x) = -6 + 20x - 9$
29. $4(x - 3) - (x - 5) = 0$
30. $-2(4 - x) = 6(x + 2) + 3x$
31. $\frac{4}{7} = \frac{x}{21}$
32. $\frac{x}{4} = \frac{-20}{16}$
33. $\frac{9c}{10} = \frac{9}{5}$
34. $\frac{1}{4} = \frac{z+1}{4}$
35. $\frac{a}{5} = \frac{a-3}{2}$
36. $\frac{n}{10} = 9 - \frac{n}{5}$
37. $\frac{2}{8} + \frac{3}{4} = \frac{w}{5}$
38. $x - \frac{3}{4} = -2x$
39. $\frac{x}{4} - \frac{x}{6} = \frac{1}{4}$
40. $a - \frac{a}{3} + \frac{a}{5} = 26$
41. $\frac{12}{10} = \frac{z}{25}$
42. $\frac{-2}{6} = \frac{3c}{9}$
43. $\frac{x+4}{7} = \frac{3}{7}$
44. $\frac{4x+5}{6} = \frac{7}{2}$
45. $6 - \frac{x}{4} = \frac{x}{8}$
46. $\frac{x}{3} - \frac{3x}{4} = \frac{1}{12}$
47. $\frac{5}{2} - x = 3x$
48. $\frac{3-5y}{4} = \frac{2-4y}{3}$
49. $\frac{2x-1}{3} - \frac{3x}{4} = \frac{5}{6}$
50. $-\frac{x}{4} = 12$
51. $-x = -12$
52. $-2x = -16$
53. $2x = -14$
54. $\frac{1}{7}x = -8$
55. $\frac{1}{7}x = 2$
56. $-\frac{x}{2} = 4$
57. $-x = 26$
58. $3x = 15$
59. $4x = -32$
60. $\frac{1}{3}x = 5$
61. $\frac{1}{9}x = 5$
62. $-\frac{5}{3}x = -15$
63. $-\frac{6}{5}x = -30$
64. $\frac{6}{5}x = 90$
65. $\frac{1}{3}x = 4$
66. $\frac{7}{6}x = 168$
67. $\frac{1}{6}x = 2$
68. $-\frac{9}{5}x = -45$
69. $-\frac{4}{9}x = -36$
70. $4x - 4 = -40$
71. $9x - 7 = -34$
72. $\frac{7}{8}y - 6 = 8$
73. $10 - x = 6$
74. $-2x - 4x = 1$
75. $-9x - 9x = -9$
76. $\frac{x}{3} - \frac{x}{5} = 2$
77. $\frac{x}{7} - \frac{x}{9} = 2$
78. $\frac{x}{3} - \frac{x}{9} = 6$
79. $\frac{x}{8} - \frac{x}{9} = 1$
80. $\frac{5}{9}y - 4 = 6$
81. $x + 0.4x = 3.5$
82. $5(x - 3) = 45$
83. $-3(x + 7) = 9$
84. $-4(x - 6) = 12$
85. $8 = 2(x - 5) + 6x$
86. $2 = 7(x + 4) + 9x$
87. $1 = 3(x - 2) + 3 - 2x$
88. $3 = 4(x - 2) + 5 - 3x$
89. $3.65 - 7.4x + 1.12 = 21.76$
90. $-8x + 3 - 2x = -6x + 3 - 4x$
91. $10x + 3 + 10x = 13x - 3 + 7x$
92. $6 + 3x = 5(x - 1) - 3(x - 2)$
93. $10 - 5x = 3(x - 4) - 2(x + 7)$
94. $9.2y - 4.3 = 50.9$
95. $0.05z + 0.2 = 0.15z - 10.5$
96. $0.25(60) + 0.10x = 0.15(60 + x)$
97. $0.5(3q + 87) = 1.5q + 43.5$
98. $0.4(y + 10) + 0.6y = 2$
99. $21.1w + 4.6 = 10.9w + 35.2$
100. $0.125x = 0.025(5x + 1)$

LINEAR EQUATIONS PRACTICE ANSWERS

1. $x = 1$
2. $x = -13$
3. $x = 11$
4. $x = -27$
5. $x = 2$
6. $x = \frac{11}{2}$
7. $x = -2$
8. $x = -3$
9. $x = 12$
10. $x = 4$
11. $x = 6$
12. $x = -3$
13. $x = -\frac{21}{5}$
14. $x = -5$
15. $x = -19$
16. $x = -1$
17. $x = 0$
18. $x = \frac{1}{5}$
19. $x = 3$
20. $x = \frac{10}{7}$
21. $x = 0$
22. $x = -1$
23. No Solution
24. $x = -\frac{23}{5}$
25. All real numbers
26. $x = -\frac{4}{3}$
27. No Solution
28. All real numbers
29. $x = \frac{7}{3}$
30. $x = -\frac{20}{7}$
31. $x = 12$
32. $x = -5$
33. $c = 2$
34. $z = 0$
35. $a = 5$
36. $n = 30$
37. $w = 5$
38. $x = \frac{1}{4}$
39. $x = 3$
40. $a = 30$
41. $z = 30$
42. $c = -1$
43. $x = -1$
44. $x = 4$
45. $x = 16$
46. $x = -\frac{1}{5}$
47. $x = \frac{5}{8}$
48. $y = -1$
49. $x = -14$
50. $x = -48$
51. $x = 12$
52. $x = 8$
53. $x = -7$
54. $x = -56$
55. $x = 14$
56. $x = -8$
57. $x = -26$
58. $x = 5$
59. $x = -8$
60. $x = 15$
61. $x = 45$
62. $x = 9$
63. $x = 25$
64. $x = 75$
65. $x = 12$
66. $x = 144$
67. $x = 12$
68. $x = 25$
69. $x = 81$
70. $x = -9$
71. $x = -3$
72. $y = 16$
73. $x = 4$
74. $x = -\frac{1}{6}$
75. $x = \frac{1}{2}$
76. $x = 15$
77. $x = 63$
78. $x = 27$
79. $x = 72$
80. $y = 18$
81. $x = 2.5$
82. $x = 12$
83. $x = -10$
84. $x = 3$
85. $x = \frac{9}{4}$
86. $x = -\frac{13}{8}$
87. $x = 4$
88. $x = 6$
89. $x \approx -2.3$
90. All real numbers
91. No Solution
92. $x = -5$
93. $x = 6$
94. $y = 6$
95. $z = 107$
96. $x = 120$
97. All real numbers
98. $y = -2$
99. $w = 3$
100. No Solution

Statistical Sampling

Each row of data represents an observation about something in the world.

When working with data, we often do not have access to all possible observations. This could be for many reasons; for example:

- It may difficult or expensive to make more observations.
- It may be challenging to gather all observations together.
- More observations are expected to be made in the future.

Observations made in a domain represent samples of some broader idealized and unknown population of all possible observations that could be made in the domain. This is a useful conceptualization as we can see the separation and relationship between observations and the idealized population.

We can also see that, even if we intend to use big data infrastructure on all available data, that the data still represents a sample of observations from an idealized population.

Nevertheless, we may wish to estimate properties of the population. We do this by using samples of observations.

Sampling consists of selecting some part of the population to observe so that one may estimate something about the whole population.

Statistical Hypothesis Testing

Data alone is not interesting. It is the interpretation of the data that we are really interested in.

In statistics, when we wish to start asking questions about the data and interpret the results, we use statistical methods that provide a confidence or likelihood about the answers. In general, this class of methods is called , or significance tests.

The term "*hypothesis*" may make you think about science, where we investigate a hypothesis. This is along the right track.

In statistics, a hypothesis test calculates some quantity under a given assumption. The result of the test allows us to interpret whether the assumption holds or whether the assumption has been violated.

Two concrete examples that we will use a lot in machine learning are:

- A test that assumes that data has a normal distribution.
- A test that assumes that two samples were drawn from the same underlying population distribution.

The assumption of a statistical test is called the null hypothesis, or hypothesis 0 (H0 for short). It is often called the default assumption, or the assumption that nothing has changed.

A violation of the test's assumption is often called the first hypothesis, hypothesis 1 or H1 for short. H1 is really a short hand for "*some other hypothesis*," as all we know is that the evidence suggests that the H0 can be rejected.

- **Hypothesis 0 (H0)**: Assumption of the test holds and is failed to be rejected at some level of significance.
- **Hypothesis 1 (H1)**: Assumption of the test does not hold and is rejected at some level of significance.

Before we can reject or fail to reject the null hypothesis, we must interpret the result of the test.

4 Data Science Software Tools-Orientation

4.1 Languages – R

Why the R Language?

.

- R is designed to operate the way that problems are thought about.
- R is not just a statistics package, it's a language
- R is both flexible and powerful.

Why R for data analysis?

R is not the only language that can be used for data analysis. Why R rather than another? Here is a list:

- interactive language
- data structures
- graphics
- missing values
- functions as first class objects
- packages
- community

Data analysis is inherently an interactive process what you see at one stage determines what you want to do next. Interactivity is important. Language is important. The two together — an interactive language — is even more than their sum. But there is a down-side: compromises between interactive use and programming use are the cause of some user trauma.

R has a fantastic mechanism for creating data structures. Obviously if you are doing data analysis, you want to be able to put your data into a natural form. You don't have to warp your data into a particular structure because that is all that is available.

Graphics should be central to data analysis. Humans are predominantly visual, we don’t intuitively grasp numbers like we do pictures. It is easy to produce graphs for exploring data. The default graphs can be tweaked to get publication-quality graphs.

Real data have missing values. Missing values are an integral part of the R language. Many functions have arguments that control how missing values are to be handled.

Functions, like `mean` and `median`, are objects that you can use like data. You can easily change your analysis to use the median (or some strange estimate you make up on the spot) rather than the mean.

R has a package system that makes it extremely easy for people to add their own functionality so it is indistinguishable from the central part of R. And people have. There are thousands of packages that do all sorts of extraordinary things.

The R community is very strong, and quite committed to improving data analysis.

4.2 Libraries - SciPy

SciPy, pronounced as Sigh Pi, is a scientific python open source, distributed under the BSD licensed library to perform Mathematical, Scientific and Engineering Computations.

The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The SciPy library is built to work with NumPy arrays and provides many user-friendly and efficient numerical practices such as routines for numerical integration and optimization. Together, they run on all popular operating systems, are quick to install and are free of charge. NumPy and SciPy are easy to use, but powerful enough to depend on by some of the world's leading scientists and engineers.

The basic data structure used by SciPy is a multidimensional array provided by the NumPy module. NumPy provides some functions for Linear Algebra, Fourier Transforms and Random Number Generation, but not with the generality of the equivalent functions in SciPy.

4.3 Data Engineering - Profiling, ETL

What Is Data Profiling? Process, Best Practices and Tools

Data processing and analysis can't happen without data profiling —reviewing source data for content and quality. As data gets bigger and infrastructure moves to the cloud, data profiling is increasingly important. Need to achieve big data profiling with limited time and resources?

What is data profiling?

Data profiling is the process of reviewing source data, understanding structure, content and interrelationships, and identifying potential for data projects.

Data profiling is a crucial part of:

- **Data warehouse and business intelligence (DW/BI) projects**—data profiling can uncover data quality issues in data sources, and what needs to be corrected in ETL.
- **Data conversion and migration projects**—data profiling can identify data quality issues, which you can handle in scripts and data integration tools copying data from source to target. It can also uncover new requirements for the target system.
- **Source system data quality projects**—data profiling can highlight data which suffers from serious or numerous quality issues, and the

source of the issues (e.g. user inputs, errors in interfaces, data corruption).

Data profiling involves:

- Collecting descriptive statistics like min, max, count and sum.
- Collecting data types, length and recurring patterns.
- Tagging data with keywords, descriptions or categories.
- Performing data quality assessment, risk of performing joins on the data.
- Discovering metadata and assessing its accuracy.
- Identifying distributions, key candidates, foreign-key candidates, functional dependencies, embedded value dependencies, and performing inter-table analysis.

Types of data profiling

Structure discovery

Validating that data is consistent and formatted correctly, and performing mathematical checks on the data (e.g. sum, minimum or maximum). Structure discovery helps understand how well data is structured—for example, what percentage of phone numbers do not have the correct number of digits.

Content discovery

Looking into individual data records to discover errors. Content discovery identifies which specific rows in a table contain problems, and which systemic issues occur in the data (for example, phone numbers with no area code).

Relationship discovery

Discovering how parts of the data are interrelated. For example, key relationships between database tables, references between cells or tables in a spreadsheet. Understanding relationships is crucial to reusing data; related data sources should be united into one or imported in a way that preserves important relationships.

Open source data profiling tools

1. Quadient DataCleaner—key features include:
   - Data quality, data profiling and data wrangling
   - Detect and merge duplicates
   - Boolean analysis
   - Completeness analysis
   - Character set distribution
   - Date gap analysis
   - Reference data matching

2. **Aggregate Profiler (Open Source Data Quality and Profiling)key features include:**
   - Data profiling, filtering, and governance
   - Similarity checks
   - Data enrichment
   - Real time alerting for data issues or changes
   - Basket analysis with bubble chart validation
   - Single customer view
   - Dummy data creation
   - Metadata discovery

- Anomaly discovery and data cleansing tool
- Hadoop integration

3. Talend Open Studio—a suite of open source tools, data quality features include:

- Customizable data assessment
- A pattern library
- Analytics with graphical charts
- Fraud pattern detection
- Column set analysis
- Advanced matching
- Time column correlation

4.4 Leaflet Data Visualization Framework

At HumanGeo, we're fans of Leaflet, Cloudmade's JavaScript web mapping framework. In the past, we've used other JavaScript mapping frameworks like OpenLayers and Google Maps, and while these frameworks are great, we like Leaflet for its smooth animation, simple API, and good documentation and examples. In fact, we like Leaflet so much, we've been using it in all of our web projects that require a mapping component.Since Leaflet is a relative newcomer in the world of JavaScript mapping frameworks (2011), and since the developers have focused on keeping the library lightweight, there are plenty of opportunities to extend the basic functionality that Leaflet offers.

As a side project at HumanGeo, we've been working on extending Leaflet's capabilities to better support visualizing data, and these efforts have produced HumanGeo's Leaflet Data Visualization Framework. Before I delve into some of the features of the framework, I'd like to provide a little background on why we created the framework in the first place, in particular, I'd like to focus on the challenges that developers face when creating data-driven geospatial visualizations.

When visualizing data on a 2D map we often wish to illustrate differences in data values geographically by varying point, line, and polygon styles (color, fill color, line weight, opacity, etc.) dynamically based on the values of those data. The goal is for users to look at our map and quickly understand geographic differences in the data being visualized. This technique is commonly referred to as thematic mapping, and is a frequently employed technique used in infographics and for illustrating concepts related to human geography. Within the realm of thematic mapping, proportional symbols and choropleth maps are two widely used approaches for illustrating variations in data.

MoneyBall Project - Solutions

Rules of Baseball

You don't need to know much about Baseball to complete this exercise. If you're totally unfamiliar with Baseball, check out this useful explanatory video!

Background

Source: Wikipedia

The 2002 Oakland A's

The Oakland Athletics' 2002 season was the team's 35th in Oakland, California. It was also the 102nd season in franchise history. The Athletics finished first in the American League West with a record of 103-59.

The Athletics' 2002 campaign ranks among the most famous in franchise history. Following the 2001 season, Oakland saw the departure of three key players (the lost boys). Billy Beane, the team's general manager, responded with a series of under-the-radar free agent signings. The new-look Athletics, despite a comparative lack of star power, surprised the baseball world by besting the 2001 team's regular season record. The team is most famous, however, for winning 20 consecutive games between August 13 and September 4, 2002.[1] The Athletics' season was the subject of Michael Lewis' 2003 book Moneyball: The Art of Winning an Unfair Game (as Lewis was given the opportunity to follow the team around throughout that season)

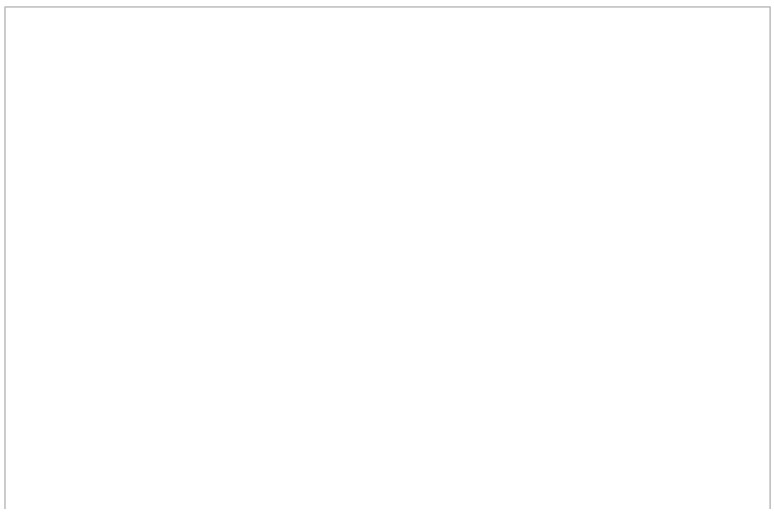
This project is based off the book written by Michael Lewis (later turned into a movie).

Moneyball Book

The central premise of book *Moneyball* is that the collective wisdom of baseball insiders (including players, managers, coaches, scouts, and the front office) over the past century is subjective and often flawed. Statistics such as stolen bases, runs batted in, and batting average, typically used to gauge players, are relics of a 19th-century view of the game and the statistics available at that time. The book argues that the Oakland A's' front office took advantage of more analytical gauges of player performance to field a team that could better compete against richer competitors in Major League Baseball (MLB).

Rigorous statistical analysis had demonstrated that on-base percentage and slugging percentage are better indicators of offensive success, and the A's became convinced that these qualities were cheaper to obtain on the open market than more historically valued qualities such as speed and contact. These observations often flew in the face of conventional baseball wisdom and the beliefs of many baseball scouts and executives.

By re-evaluating the strategies that produce wins on the field, the 2002 Athletics, with approximately US 44 million dollars in salary, were competitive with larger market teams such as the New York Yankees, who spent over US$125 million in payroll that same season.



Because of the team's smaller revenues, Oakland is forced to find players undervalued by the market, and their system for finding value in undervalued players has proven itself thus far. This approach brought the A's to the playoffs in 2002 and 2003.

In this project we'll work with some data and with the goal of trying to find replacement players for the ones lost at the start of the off-season - During the 2001–02 offseason, the team lost three key free agents to larger market teams: 2000 AL MVP Jason Giambi to the New York Yankees, outfielder Johnny Damon to the Boston Red Sox, and closer Jason Isringhausen to the St. Louis Cardinals.

The main goal of this project is for you to feel comfortable working with R on real data to try and derive actionable insights!

Let's get started!

Follow the steps outlined in bold below using your new R skills and help the Oakland A's recruit under-valued players!

Data

We'll be using data from Sean Lahaman's Website a very useful source for baseball statistics. The documentation for the csv files is located in the **readme2013.txt** file. You may need to reference this to understand what acronyms stand for.

Use R to open the Batting.csv file and assign it to a dataframe called batting using *read.csv*

In [1]:

```
batting <- read.csv('Batting.csv')
```

Use head() to check out the batting

In [2]:

```
head(batting)
```

Out[2]:

	playerID	yearID	stint	teamID	lgID	G	G_batting	AB	R	H	X2B	X3B	HR	RBI	SB	CS	BB	SO	IBB	HBP	SH	SF	GI
1	aardsda01	2004	1	SFN	NL	11	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	aardsda01	2006	1	CHN	NL	45	43	2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
3	aardsda01	2007	1	CHA	AL	25	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	aardsda01	2008	1	BOS	AL	47	5	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
5	aardsda01	2009	1	SEA	AL	73	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	aardsda01	2010	1	SEA	AL	53	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Use str() to check the structure. Pay close attention to how columns that start with a number get an 'X' in front of them! You'll need to know this to call those columns!

In [3]:

```
str(batting)
```

```
'data.frame': 97889 obs. of  24 variables:
 $ playerID : Factor w/ 18107 levels "aardsda01","aaronha01",..: 1 1 1 1 1 1 1 2 2 2 ...
 $ yearID   : int  2004 2006 2007 2008 2009 2010 2012 1954 1955 1956 ...
 $ stint    : int  1 1 1 1 1 1 1 1 1 1 ...
 $ teamID   : Factor w/ 149 levels "ALT","ANA","ARI",..: 117 35 33 16 116 116 93 80 80 80 ...
 $ lgID     : Factor w/ 6 levels "AA","AL","FL",..: 4 4 2 2 2 2 2 4 4 4 ...
 $ G        : int  11 45 25 47 73 53 1 122 153 153 ...
 $ G_batting: int  11 43 2 5 3 4 NA 122 153 153 ...
 $ AB       : int  0 2 0 1 0 0 NA 468 602 609 ...
 $ R        : int  0 0 0 0 0 0 NA 58 105 106 ...
 $ H        : int  0 0 0 0 0 0 NA 131 189 200 ...
 $ X2B      : int  0 0 0 0 0 0 NA 27 37 34 ...
 $ X3B      : int  0 0 0 0 0 0 NA 6 9 14 ...
 $ HR       : int  0 0 0 0 0 0 NA 13 27 26 ...
```

```
 $ RBI       : int  0 0 0 0 0 0 NA 69 106 92 ...
 $ SB        : int  0 0 0 0 0 0 NA 2 3 2 ...
 $ CS        : int  0 0 0 0 0 0 NA 2 1 4 ...
 $ BB        : int  0 0 0 0 0 0 NA 28 49 37 ...
 $ SO        : int  0 0 0 1 0 0 NA 39 61 54 ...
 $ IBB       : int  0 0 0 0 0 0 NA NA 5 6 ...
 $ HBP       : int  0 0 0 0 0 0 NA 3 3 2 ...
 $ SH        : int  0 1 0 0 0 0 NA 6 7 5 ...
 $ SF        : int  0 0 0 0 0 0 NA 4 4 7 ...
 $ GIDP      : int  0 0 0 0 0 0 NA 13 20 21 ...
 $ G_old     : int  11 45 2 5 NA NA NA 122 153 153 ...
```

Make sure you understand how to call the columns by using the $ symbol.

Call the head() of the first five rows of AB (At Bats) column

In [4]:

```
head(batting$AB)
```

Out[4]:

```
0  2  0  1  0  0
```

Call the head of the doubles (X2B) column

In [5]:

```
head(batting$X2B)
```

Out[5]:

```
0  0  0  0  0  0
```

Quick Note: If you used fread() to use data.table, then you won't need to worry about these X in front of numbers, instead you would use something like:

```
batting[,'2B',with=FALSE]
```

There's a few more ways of doing detailed here.

Alright! Let's move on!

Feature Engineering

We need to add three more statistics that were used in Moneyball! These are:

- Batting Average
- On Base Percentage
- Slugging Percentage

Click on the links provided and search the wikipedia page for the formula for creating the new statistic! For example, for Batting Average, you'll need to scroll down until you see:
$$ AVG = \frac{H}{AB} $$

Which means that the Batting Average is equal to H (Hits) divided by AB (At Base). So we'll do the following to create a new column called **BA** and add it to our data frame:

In [6]:

```
batting$BA <- batting$H / batting$AB
```

After doing this operation, check the last 5 entries of the BA column of your data frame and it should look like this:

In [7]:

```
tail(batting$BA,5)
```

Out[7]:

0.123076923076923 0.274647887323944 0.147058823529412 0.274509803921569 0.213872832369942

Now do the same for some new columns! On Base Percentage (OBP) and Slugging Percentage (SLG). Hint: For SLG, you need 1B (Singles), this isn't in your data frame. However you can calculate it by subtracting doubles,triples, and home runs from total hits (H): 1B = H-2B-3B-HR

- **Create an OBP Column**
- **Create an SLG Column**

In [8]:

```
# On Base Percentage
batting$OBP <- (batting$H + batting$BB + batting$HBP)/(batting$AB + batting$BB + batting$HBP + batt
ing$SF)
```

In [9]:

```
# Creating X1B (Singles)
batting$X1B <- batting$H - batting$X2B - batting$X3B - batting$HR
```

In [10]:

```
# Creating Slugging Average (SLG)
batting$SLG <- ((1 * batting$X1B) + (2 * batting$X2B) + (3 * batting$X3B) + (4 * batting$HR) ) / bat
ting$AB
```

Check the structure of your data frame using str()

In [11]:

```
str(batting)
```

```
'data.frame': 97889 obs. of  28 variables:
 $ playerID : Factor w/ 18107 levels "aardsda01","aaronha01",..: 1 1 1 1 1 1 1 2 2 2 ...
 $ yearID   : int  2004 2006 2007 2008 2009 2010 2012 1954 1955 1956 ...
 $ stint    : int  1 1 1 1 1 1 1 1 1 1 ...
 $ teamID   : Factor w/ 149 levels "ALT","ANA","ARI",..: 117 35 33 16 116 116 93 80 80 80 ...
 $ lgID     : Factor w/ 6 levels "AA","AL","FL",..: 4 4 2 2 2 2 2 4 4 4 ...
 $ G        : int  11 45 25 47 73 53 1 122 153 153 ...
 $ G_batting: int  11 43 2 5 3 4 NA 122 153 153 ...
 $ AB       : int  0 2 0 1 0 0 NA 468 602 609 ...
 $ R        : int  0 0 0 0 0 0 NA 58 105 106 ...
 $ H        : int  0 0 0 0 0 0 NA 131 189 200 ...
 $ X2B      : int  0 0 0 0 0 0 NA 27 37 34 ...
 $ X3B      : int  0 0 0 0 0 0 NA 6 9 14 ...
 $ HR       : int  0 0 0 0 0 0 NA 13 27 26 ...
 $ RBI      : int  0 0 0 0 0 0 NA 69 106 92 ...
 $ SB       : int  0 0 0 0 0 0 NA 2 3 2 ...
 $ CS       : int  0 0 0 0 0 0 NA 2 1 4 ...
 $ BB       : int  0 0 0 0 0 0 NA 28 49 37 ...
 $ SO       : int  0 0 0 1 0 0 NA 39 61 54 ...
 $ IBB      : int  0 0 0 0 0 0 NA NA 5 6 ...
 $ HBP      : int  0 0 0 0 0 0 NA 3 3 2 ...
 $ SH       : int  0 1 0 0 0 0 NA 6 7 5 ...
 $ SF       : int  0 0 0 0 0 0 NA 4 4 7 ...
 $ GIDP     : int  0 0 0 0 0 0 NA 13 20 21 ...
 $ G_old    : int  11 45 2 5 NA NA NA 122 153 153 ...
 $ BA       : num  NaN 0 NaN 0 NaN ...
 $ OBP      : num  NaN 0 NaN 0 NaN ...
 $ X1B      : int  0 0 0 0 0 0 NA 85 116 126 ...
 $ SLG      : num  NaN 0 NaN 0 NaN ...
```

Merging Salary Data with Batting Data

We know we don't just want the best players, we want the most *undervalued* players, meaning we will also need to know current salary information! We have salary information in the csv file 'Salaries.csv'.

Complete the following steps to merge the salary data with the player stats!

Load the Salaries.csv file into a dataframe called sal using read.csv

In [12]:

```
sal <- read.csv('Salaries.csv')
```

Use summary to get a summary of the batting data frame and notice the minimum year in the yearID column. Our batting data goes back to 1871! Our salary data starts at 1985, meaning we need to remove the batting data that occured before 1985.

Use subset() to reassign batting to only contain data from 1985 and onwards

In [13]:

```
summary(batting)
```

Out[13]:

```
      playerID         yearID         stint            teamID        lgID       
 mcguide01:   31   Min.   :1871   Min.   :1.000   CHN     : 4720   AA  : 1890  
 henderi01:   29   1st Qu.:1931   1st Qu.:1.000   PHI     : 4621   AL  :44369  
 newsobo01:   29   Median :1970   Median :1.000   PIT     : 4575   FL  :  470  
 johnto01 :   28   Mean   :1962   Mean   :1.077   SLN     : 4535   NL  :49944  
 kaatji01 :   28   3rd Qu.:1995   3rd Qu.:1.000   CIN     : 4393   PL  :  147  
 ansonca01:   27   Max.   :2013   Max.   :5.000   CLE     : 4318   UA  :  332  
 (Other)  :97717                                  (Other):70727   NA's:  737  
       G             G_batting           AB              R         
 Min.   :  1.00   Min.   :  0.00   Min.   :  0.0   Min.   :  0.00  
 1st Qu.: 13.00   1st Qu.:  7.00   1st Qu.:  9.0   1st Qu.:  0.00  
 Median : 35.00   Median : 32.00   Median : 61.0   Median :  5.00  
 Mean   : 51.65   Mean   : 49.13   Mean   :154.1   Mean   : 20.47  
 3rd Qu.: 81.00   3rd Qu.: 81.00   3rd Qu.:260.0   3rd Qu.: 31.00  
 Max.   :165.00   Max.   :165.00   Max.   :716.0   Max.   :192.00  
                  NA's   :1406     NA's   :6413    NA's   :6413    
       H              X2B             X3B              HR        
 Min.   :  0.00   Min.   : 0.0   Min.   : 0.000   Min.   : 0.000  
 1st Qu.:  1.00   1st Qu.: 0.0   1st Qu.: 0.000   1st Qu.: 0.000  
 Median : 12.00   Median : 2.0   Median : 0.000   Median : 0.000  
 Mean   : 40.37   Mean   : 6.8   Mean   : 1.424   Mean   : 3.002  
 3rd Qu.: 66.00   3rd Qu.:10.0   3rd Qu.: 2.000   3rd Qu.: 3.000  
 Max.   :262.00   Max.   :67.0   Max.   :36.000   Max.   :73.000  
 NA's   :6413     NA's   :6413   NA's   :6413     NA's   :6413    
      RBI              SB               CS              BB        
 Min.   :  0.00   Min.   :  0.000   Min.   : 0.000   Min.   :  0.00  
 1st Qu.:  0.00   1st Qu.:  0.000   1st Qu.: 0.000   1st Qu.:  0.00  
 Median :  5.00   Median :  0.000   Median : 0.000   Median :  4.00  
 Mean   : 18.47   Mean   :  3.265   Mean   : 1.385   Mean   : 14.21  
 3rd Qu.: 28.00   3rd Qu.:  2.000   3rd Qu.: 1.000   3rd Qu.: 21.00  
 Max.   :191.00   Max.   :138.000   Max.   :42.000   Max.   :232.00  
 NA's   :6837     NA's   :7713      NA's   :29867    NA's   :6413    
       SO              IBB              HBP              SH        
 Min.   :  0.00   Min.   :  0.00   Min.   : 0.000   Min.   : 0.000  
 1st Qu.:  2.00   1st Qu.:  0.00   1st Qu.: 0.000   1st Qu.: 0.000  
 Median : 11.00   Median :  0.00   Median : 0.000   Median : 1.000  
 Mean   : 21.95   Mean   :  1.28   Mean   : 1.136   Mean   : 2.564  
 3rd Qu.: 31.00   3rd Qu.:  1.00   3rd Qu.: 1.000   3rd Qu.: 3.000  
 Max.   :223.00   Max.   :120.00   Max.   :51.000   Max.   :67.000  
 NA's   :14251    NA's   :42977    NA's   :9233     NA's   :12751   
       SF              GIDP            G_old             BA        
 Min.   : 0.0    Min.   : 0.00   Min.   :  0.00   Min.   :0.000  
 1st Qu.: 0.0    1st Qu.: 0.00   1st Qu.: 11.00   1st Qu.:0.148  
 Median : 0.0    Median : 1.00   Median : 34.00   Median :0.231  
 Mean   : 1.2    Mean   : 3.33   Mean   : 50.99   Mean   :0.209  
 3rd Qu.: 2.0    3rd Qu.: 5.00   3rd Qu.: 82.00   3rd Qu.:0.275  
 Max.   :19.0    Max.   :36.00   Max.   :165.00   Max.   :1.000  
 NA's   :42446   NA's   :32521   NA's   :5189     NA's   :13520  
      OBP              X1B              SLG        
 Min.   :0.00    Min.   :  0.00   Min.   :0.000  
```

```
 1st Qu.:0.19    1st Qu.:  1.00   1st Qu.:0.179
 Median :0.29    Median :  9.00   Median :0.309
 Mean   :0.26    Mean   : 29.14   Mean   :0.291
 3rd Qu.:0.34    3rd Qu.: 48.00   3rd Qu.:0.397
 Max.   :1.00    Max.   :225.00   Max.   :4.000
 NA's   :49115   NA's   :6413     NA's   :13520
```

In [14]:

```
batting <- subset(batting,yearID >= 1985)
```

Now use summary again to make sure the subset reassignment worked, your yearID min should be 1985

In [15]:

```
summary(batting)
```

Out[15]:

```
      playerID         yearID         stint          teamID      lgID
 moyerja01:   27   Min.   :1985   Min.   :1.00   SDN     : 1313   AA:    0
 mulhote01:   26   1st Qu.:1993   1st Qu.:1.00   CLE     : 1306   AL:17226
 weathda01:   26   Median :2000   Median :1.00   PIT     : 1299   FL:    0
 maddugr01:   25   Mean   :2000   Mean   :1.08   NYN     : 1297   NL:18426
 sierrru01:   25   3rd Qu.:2007   3rd Qu.:1.00   BOS     : 1279   PL:    0
 thomeji01:   25   Max.   :2013   Max.   :4.00   CIN     : 1279   UA:    0
 (Other)  :35498                                 (Other):27879
       G           G_batting           AB              R
 Min.   :  1.0   Min.   :  0.00   Min.   :  0.0   Min.   :  0.00
 1st Qu.: 14.0   1st Qu.:  4.00   1st Qu.:  3.0   1st Qu.:  0.00
 Median : 34.0   Median : 27.00   Median : 47.0   Median :  4.00
 Mean   : 51.7   Mean   : 46.28   Mean   :144.7   Mean   : 19.44
 3rd Qu.: 77.0   3rd Qu.: 77.00   3rd Qu.:241.0   3rd Qu.: 30.00
 Max.   :163.0   Max.   :163.00   Max.   :716.0   Max.   :152.00
                 NA's   :1406     NA's   :4377    NA's   :4377
       H              X2B              X3B              HR
 Min.   :  0.00   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
 1st Qu.:  0.00   1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.: 0.000
 Median :  8.00   Median : 1.000   Median : 0.000   Median : 0.000
 Mean   : 37.95   Mean   : 7.293   Mean   : 0.824   Mean   : 4.169
 3rd Qu.: 61.00   3rd Qu.:11.000   3rd Qu.: 1.000   3rd Qu.: 5.000
 Max.   :262.00   Max.   :59.000   Max.   :23.000   Max.   :73.000
 NA's   :4377     NA's   :4377     NA's   :4377     NA's   :4377
      RBI              SB               CS               BB
 Min.   :  0.00   Min.   :  0.000   Min.   : 0.000   Min.   :  0.00
 1st Qu.:  0.00   1st Qu.:  0.000   1st Qu.: 0.000   1st Qu.:  0.00
 Median :  3.00   Median :  0.000   Median : 0.000   Median :  3.00
 Mean   : 18.41   Mean   :  2.811   Mean   : 1.219   Mean   : 14.06
 3rd Qu.: 27.00   3rd Qu.:  2.000   3rd Qu.: 1.000   3rd Qu.: 21.00
 Max.   :165.00   Max.   :110.000   Max.   :29.000   Max.   :232.00
 NA's   :4377     NA's   :4377      NA's   :4377     NA's   :4377
       SO              IBB              HBP              SH
 Min.   :  0.00   Min.   :  0.000   Min.   : 0.000   Min.   : 0.000
 1st Qu.:  1.00   1st Qu.:  0.000   1st Qu.: 0.000   1st Qu.: 0.000
 Median : 12.00   Median :  0.000   Median : 0.000   Median : 0.000
 Mean   : 27.03   Mean   :  1.171   Mean   : 1.273   Mean   : 1.465
 3rd Qu.: 42.00   3rd Qu.:  1.000   3rd Qu.: 1.000   3rd Qu.: 2.000
 Max.   :223.00   Max.   :120.000   Max.   :35.000   Max.   :39.000
 NA's   :4377     NA's   :4378      NA's   :4387     NA's   :4377
       SF              GIDP            G_old             BA
 Min.   : 0.000   Min.   : 0.00   Min.   :  0.0   Min.   :0.000
 1st Qu.: 0.000   1st Qu.: 0.00   1st Qu.: 11.0   1st Qu.:0.136
 Median : 0.000   Median : 1.00   Median : 32.0   Median :0.233
 Mean   : 1.212   Mean   : 3.25   Mean   : 49.7   Mean   :0.205
 3rd Qu.: 2.000   3rd Qu.: 5.00   3rd Qu.: 77.0   3rd Qu.:0.274
 Max.   :17.000   Max.   :35.00   Max.   :163.0   Max.   :1.000
 NA's   :4378     NA's   :4377    NA's   :5189    NA's   :8905
      OBP              X1B              SLG
 Min.   :0.000   Min.   :  0.00   Min.   :0.000
 1st Qu.:0.188   1st Qu.:  0.00   1st Qu.:0.167
 Median :0.296   Median :  6.00   Median :0.333
 Mean   :0.262   Mean   : 25.66   Mean   :0.304
 3rd Qu.:0.342   3rd Qu.: 42.00   3rd Qu.:0.423
 Max.   :1.000   Max.   :225.00   Max.   :4.000
```

```
 NA's   :8821    NA's   :4377     NA's   :8905
```

Now it is time to merge the batting data with the salary data! Since we have players playing multiple years, we'll have repetitions of playerIDs for multiple years, meaning we want to merge on *both* players and years.

Use the merge() function to merge the batting and sal data frames by c('playerID','yearID'). Call the new data frame combo

In [16]:

```
combo <- merge(batting,sal,by=c('playerID','yearID'))
```

Use summary to check the data

In [17]:

```
summary(combo)
```

Out[17]:

```
      playerID         yearID         stint          teamID.x     lgID.x
 moyerja01:   27   Min.   :1985   Min.   :1.000   LAN     :  940   AA:    0
 thomeji01:   25   1st Qu.:1993   1st Qu.:1.000   PHI     :  937   AL:12292
 weathda01:   25   Median :1999   Median :1.000   BOS     :  935   FL:    0
 vizquom01:   24   Mean   :1999   Mean   :1.098   NYA     :  928   NL:13105
 gaettga01:   23   3rd Qu.:2006   3rd Qu.:1.000   CLE     :  920   PL:    0
 griffke02:   23   Max.   :2013   Max.   :4.000   SDN     :  914   UA:    0
 (Other)  :25250                                  (Other):19823
       G             G_batting           AB              R
 Min.   :  1.00   Min.   :  0.00   Min.   :  0.0   Min.   :  0.00
 1st Qu.: 26.00   1st Qu.:  8.00   1st Qu.:  5.0   1st Qu.:  0.00
 Median : 50.00   Median : 42.00   Median : 85.0   Median :  9.00
 Mean   : 64.06   Mean   : 57.58   Mean   :182.4   Mean   : 24.71
 3rd Qu.:101.00   3rd Qu.:101.00   3rd Qu.:336.0   3rd Qu.: 43.00
 Max.   :163.00   Max.   :163.00   Max.   :716.0   Max.   :152.00
                  NA's   :906      NA's   :2661    NA's   :2661
       H              X2B              X3B              HR
 Min.   :  0.00   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
 1st Qu.:  1.00   1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.: 0.000
 Median : 19.00   Median : 3.000   Median : 0.000   Median : 1.000
 Mean   : 48.18   Mean   : 9.276   Mean   : 1.033   Mean   : 5.369
 3rd Qu.: 87.25   3rd Qu.:16.000   3rd Qu.: 1.000   3rd Qu.: 7.000
 Max.   :262.00   Max.   :59.000   Max.   :23.000   Max.   :73.000
 NA's   :2661     NA's   :2661     NA's   :2661     NA's   :2661
      RBI              SB               CS              BB
 Min.   :  0.00   Min.   :  0.000   Min.   : 0.00   Min.   :  0.00
 1st Qu.:  0.00   1st Qu.:  0.000   1st Qu.: 0.00   1st Qu.:  0.00
 Median :  8.00   Median :  0.000   Median : 0.00   Median :  6.00
 Mean   : 23.56   Mean   :  3.568   Mean   : 1.54   Mean   : 17.98
 3rd Qu.: 39.00   3rd Qu.:  3.000   3rd Qu.: 2.00   3rd Qu.: 29.00
 Max.   :165.00   Max.   :110.000   Max.   :29.00   Max.   :232.00
 NA's   :2661     NA's   :2661      NA's   :2661    NA's   :2661
       SO              IBB              HBP              SH
 Min.   :  0.00   Min.   :  0.000   Min.   : 0.000   Min.   : 0.000
 1st Qu.:  2.00   1st Qu.:  0.000   1st Qu.: 0.000   1st Qu.: 0.000
 Median : 20.00   Median :  0.000   Median : 0.000   Median : 0.000
 Mean   : 33.52   Mean   :  1.533   Mean   : 1.614   Mean   : 1.786
 3rd Qu.: 55.00   3rd Qu.:  2.000   3rd Qu.: 2.000   3rd Qu.: 2.000
 Max.   :223.00   Max.   :120.000   Max.   :35.000   Max.   :39.000
 NA's   :2661     NA's   :2662      NA's   :2670     NA's   :2661
       SF              GIDP             G_old             BA
 Min.   : 0.000   Min.   : 0.000   Min.   :  0.00   Min.   :0.000
 1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.: 20.00   1st Qu.:0.160
 Median : 0.000   Median : 2.000   Median : 47.00   Median :0.242
 Mean   : 1.554   Mean   : 4.127   Mean   : 61.43   Mean   :0.212
 3rd Qu.: 2.000   3rd Qu.: 7.000   3rd Qu.:101.00   3rd Qu.:0.276
 Max.   :17.000   Max.   :35.000   Max.   :163.00   Max.   :1.000
 NA's   :2662     NA's   :2661     NA's   :3414     NA's   :5618
      OBP              X1B              SLG            teamID.y     lgID.y
 Min.   :0.000   Min.   :  0.0   Min.   :0.000   CLE     :  935   AL:12304
 1st Qu.:0.208   1st Qu.:  0.0   1st Qu.:0.200   PIT     :  932   NL:13093
 Median :0.305   Median : 13.0   Median :0.351   PHI     :  931
 Mean   :0.270   Mean   : 32.5   Mean   :0.317   SDN     :  923
 3rd Qu.:0.346   3rd Qu.: 59.0   3rd Qu.:0.432   LAN     :  921
```

```
 Max.   :1.000   Max.   :225.0   Max.   :4.000   CIN     :  912
 NA's   :5562    NA's   :2661    NA's   :5618    (Other):19843
     salary
 Min.   :       0
 1st Qu.:  255000
 Median :  550000
 Mean   : 1879256
 3rd Qu.: 2150000
 Max.   :33000000
```

Analyzing the Lost Players

As previously mentioned, the Oakland A's lost 3 key players during the off-season. We'll want to get their stats to see what we have to replace. The players lost were: first baseman 2000 AL MVP Jason Giambi (giambja01) to the New York Yankees, outfielder Johnny Damon (damonjo01) to the Boston Red Sox and infielder Rainer Gustavo "Ray" Olmedo ('saenzol01').

Use the subset() function to get a data frame called lost_players from the combo data frame consisting of those 3 players. Hint: Try to figure out how to use %in% to avoid a bunch of or statements!

In [18]:

```
lost_players <- subset(combo,playerID %in% c('giambja01','damonjo01','saenzol01') )
```

In [19]:

```
lost_players
```

Out[19]:

	playerID	yearID	stint	teamID.x	lgID.x	G	G_batting	AB	R	H	X2B	X3B	HR	RBI	SB	CS	BB	SO	IBB	H
5135	damonjo01	1995	1	KCA	AL	47	47	188	32	53	11	5	3	23	7	0	12	22	0	1
5136	damonjo01	1996	1	KCA	AL	145	145	517	61	140	22	5	6	50	25	5	31	64	3	3
5137	damonjo01	1997	1	KCA	AL	146	146	472	70	130	12	8	8	48	16	10	42	70	2	3
5138	damonjo01	1998	1	KCA	AL	161	161	642	104	178	30	10	18	66	26	12	58	84	4	4
5139	damonjo01	1999	1	KCA	AL	145	145	583	101	179	39	9	14	77	36	6	67	50	5	3
5140	damonjo01	2000	1	KCA	AL	159	159	655	136	214	42	10	16	88	46	9	65	60	4	1
5141	damonjo01	2001	1	OAK	AL	155	155	644	108	165	34	4	9	49	27	12	61	70	1	5
5142	damonjo01	2002	1	BOS	AL	154	154	623	118	178	34	11	14	63	31	6	65	70	5	6
5143	damonjo01	2003	1	BOS	AL	145	145	608	103	166	32	6	12	67	30	6	68	74	4	2
5144	damonjo01	2004	1	BOS	AL	150	150	621	123	189	35	6	20	94	19	8	76	71	1	2
5145	damonjo01	2005	1	BOS	AL	148	148	624	117	197	35	6	10	75	18	1	53	69	3	2
5146	damonjo01	2006	1	NYA	AL	149	149	593	115	169	35	5	24	80	25	10	67	85	1	4
5147	damonjo01	2007	1	NYA	AL	141	141	533	93	144	27	2	12	63	27	3	66	79	1	2
5148	damonjo01	2008	1	NYA	AL	143	143	555	95	168	27	5	17	71	29	8	64	82	0	1
5149	damonjo01	2009	1	NYA	AL	143	143	550	107	155	36	3	24	82	12	0	71	98	1	2
5150	damonjo01	2010	1	DET	AL	145	145	539	81	146	36	5	8	51	11	1	69	90	2	2
5151	damonjo01	2011	1	TBA	AL	150	150	582	79	152	29	7	16	73	19	6	51	92	1	7
7872	giambja01	1995	1	OAK	AL	54	54	176	27	45	7	0	6	25	2	1	28	31	0	3
7873	giambja01	1996	1	OAK	AL	140	140	536	84	156	40	1	20	79	0	1	51	95	3	5
7874	giambja01	1997	1	OAK	AL	142	142	519	66	152	41	2	20	81	0	1	55	89	3	6
7875	giambja01	1998	1	OAK	AL	153	153	562	92	166	28	0	27	110	2	2	81	102	7	5
7876	giambja01	1999	1	OAK	AL	158	158	575	115	181	36	1	33	123	1	1	105	106	6	7
7877	giambja01	2000	1	OAK	AL	152	152	510	108	170	29	1	43	137	2	0	137	96	6	9
7878	giambja01	2001	1	OAK	AL	154	154	520	109	178	47	2	38	120	2	0	129	83	24	1

	playerID	yearID	stint	teamID.x	lgID.x	G	G_batting	AB	R	H	X2B	X3B	HR	RBI	SB	CS	BB	SO	IBB	H
7879	giambja01	2002	1	NYA	AL	155	155	560	120	176	34	1	41	122	2	2	109	112	4	1
7880	giambja01	2003	1	NYA	AL	156	156	535	97	134	25	0	41	107	2	1	129	140	9	2
7881	giambja01	2004	1	NYA	AL	80	80	264	33	55	9	0	12	40	0	1	47	62	1	8
7882	giambja01	2005	1	NYA	AL	139	139	417	74	113	14	0	32	87	0	0	108	109	5	1
7883	giambja01	2006	1	NYA	AL	139	139	446	92	113	25	0	37	113	2	0	110	106	12	1
7884	giambja01	2007	1	NYA	AL	83	83	254	31	60	8	0	14	39	1	0	40	66	2	8
7885	giambja01	2008	1	NYA	AL	145	145	458	68	113	19	1	32	96	2	1	76	111	5	2
7886	giambja01	2009	1	OAK	AL	83	83	269	39	52	13	0	11	40	0	0	50	72	1	7
7887	giambja01	2009	2	COL	NL	19	19	24	4	7	1	0	2	11	0	0	7	8	0	0
7888	giambja01	2010	1	COL	NL	87	87	176	17	43	9	0	6	35	2	0	35	47	5	6
7889	giambja01	2011	1	COL	NL	64	64	131	20	34	6	0	13	32	0	0	17	45	0	3
7890	giambja01	2012	1	COL	NL	60	NA	89	7	20	4	0	1	8	0	0	20	24	2	2
7891	giambja01	2013	1	CLE	AL	71	71	186	21	34	8	0	9	31	0	1	23	56	0	4
20112	saenzol01	1999	1	OAK	AL	97	97	255	41	70	18	0	11	41	1	1	22	47	1	1
20113	saenzol01	2000	1	OAK	AL	76	76	214	40	67	12	2	9	33	1	0	25	40	2	7
20114	saenzol01	2001	1	OAK	AL	106	106	305	33	67	21	1	9	32	0	1	19	64	1	1
20115	saenzol01	2002	1	OAK	AL	68	68	156	15	43	10	1	6	18	1	1	13	31	1	7
20116	saenzol01	2005	1	LAN	NL	109	109	319	39	84	24	0	15	63	0	1	27	63	1	3
20117	saenzol01	2006	1	LAN	NL	103	103	179	30	53	15	0	11	48	0	0	14	47	1	7
20118	saenzol01	2007	1	LAN	NL	92	92	110	9	21	5	0	4	18	0	0	16	25	0	2

Since all these players were lost in after 2001 in the offseason, let's only concern ourselves with the data from 2001.

Use subset again to only grab the rows where the yearID was 2001.

In [20]:

```
lost_players <- subset(lost_players,yearID == 2001)
```

Reduce the lost_players data frame to the following columns: playerID,H,X2B,X3B,HR,OBP,SLG,BA,AB

In [21]:

```
lost_players <- lost_players[,c('playerID','H','X2B','X3B','HR','OBP','SLG','BA','AB')]
```

In [22]:

```
head(lost_players)
```

Out[22]:

	playerID	H	X2B	X3B	HR	OBP	SLG	BA	AB
5141	damonjo01	165	34	4	9	0.3235294	0.363354	0.2562112	644
7878	giambja01	178	47	2	38	0.4769001	0.6596154	0.3423077	520
20114	saenzol01	67	21	1	9	0.2911765	0.3836066	0.2196721	305

Replacement Players

Now we have all the information we need! Here is your final task - Find Replacement Players for the key three players we lost! However, you have three constraints:

- The total combined salary of the three players can not exceed 15 million dollars.
- Their combined number of At Bats (AB) needs to be equal to or greater than the lost players.

- Their mean OBP had to equal to or greater than the mean OBP of the lost players

Use the combo dataframe you previously created as the source of information! Remember to just use the 2001 subset of that dataframe. There's lost of different ways you can do this, so be creative! It should be relatively simple to find 3 players that satisfy the requirements, note that there are many correct combinations available!

Helpful info on sorting data frames

Example Solution

Note: There are lots of correct answers and ways to solve this!

First only grab available players from year 2001

In [38]:

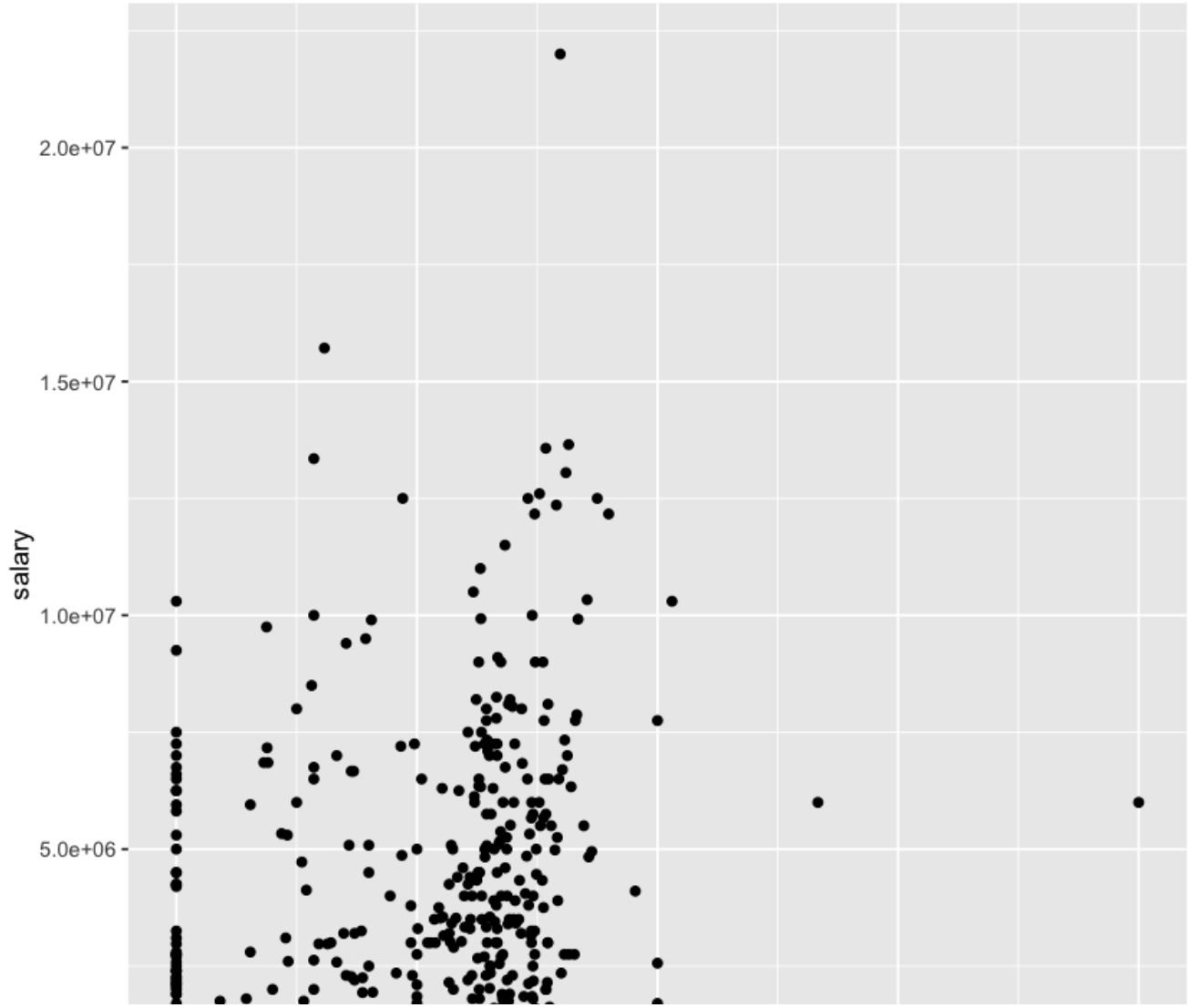
```
library(dplyr)
avail.players <- filter(combo,yearID==2001)
```

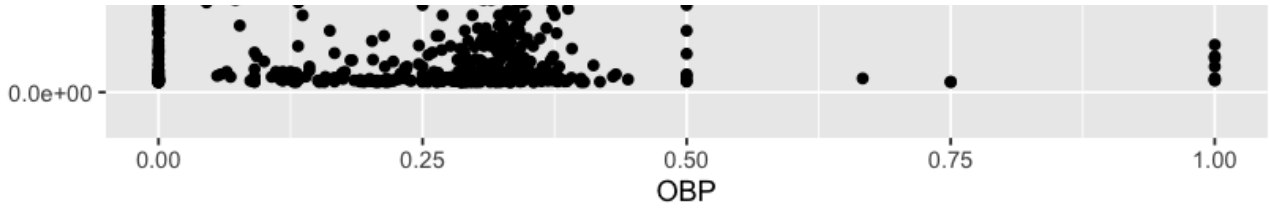
Then I made a quick plot to see where I should cut-off for salary in respect to OBP:

In [39]:

```
library(ggplot2)
ggplot(avail.players,aes(x=OBP,y=salary)) + geom_point()
```

```
Warning message:
: Removed 168 rows containing missing values (geom_point).
```





Looks like there is no point in paying above 8 million or so (I'm just eyeballing this number). I'll choose that as a cutt off point. There are also a lot of players with OBP==0. Let's get rid of them too.

In [41]:

```
avail.players <- filter(avail.players,salary<8000000,OBP>0)
```

The total AB of the lost players is 1469. This is about 1500, meaning I should probably cut off my avail.players at 1500/3= 500 AB.

In [42]:

```
avail.players <- filter(avail.players,AB >= 500)
```

Now let's sort by OBP and see what we've got!

In [44]:

```
possible <- head(arrange(avail.players,desc(OBP)),10)
```

Grab columns I'm interested in:

In [45]:

```
possible <- possible[,c('playerID','OBP','AB','salary')]
```

In [46]:

```
possible
```

Out[46]:

	playerID	OBP	AB	salary
1	giambja01	0.4769001	520	4103333
2	heltoto01	0.4316547	587	4950000
3	berkmla01	0.4302326	577	305000
4	gonzalu01	0.4285714	609	4833333
5	thomeji01	0.4161491	526	7875000
6	alomaro01	0.4146707	575	7750000
7	edmonji01	0.4102142	500	6333333
8	gilesbr02	0.4035608	576	7333333
9	pujolal01	0.402963	590	200000
10	olerujo01	0.4011799	572	6700000

Can't choose giambja again, but the other ones look good (2-4). I choose them!

In [47]:

```
possible[2:4,]
```

Out[47]:

	playerID	OBP	AB	salary
2	heltoto01	0.4316547	587	4950000
3	berkmla01	0.4302326	577	305000
4	gonzalu01	0.4285714	609	4833333

Great, looks like I just saved the 2001 Oakland A's a lot of money! If only I had a time machine and R, I could have made a lot of money in 2001 picking players!

Great Job!

Congratulations on your first project!

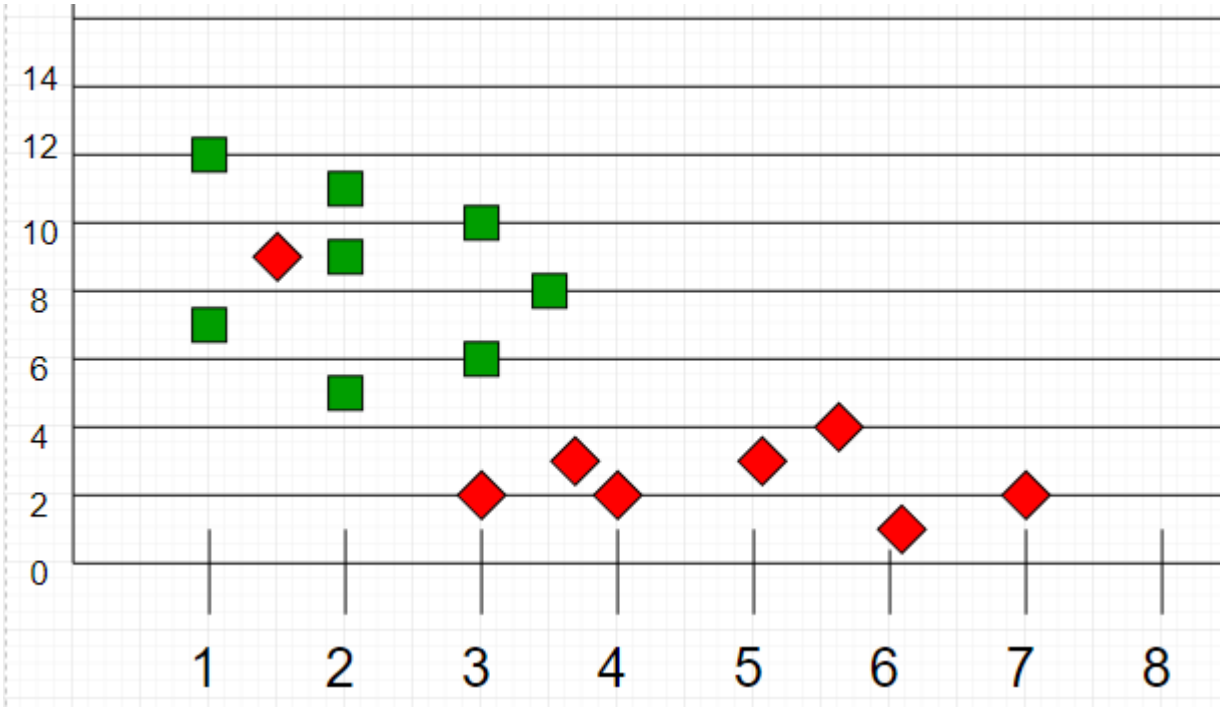
K-Nearest Neighbours

K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection.

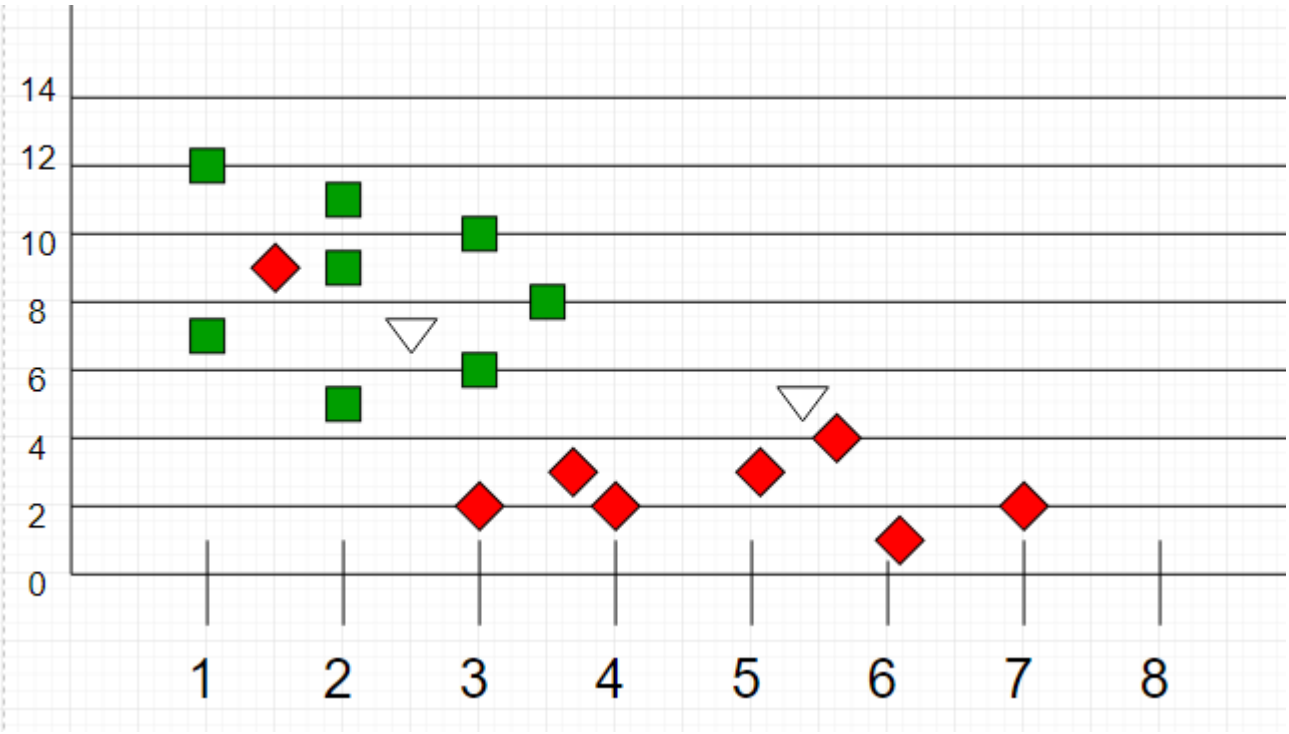
It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data).

We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.

As an example, consider the following table of data points containing two features:



Now, given another set of data points (also called testing data), allocate these points a group by analyzing the training set. Note that the unclassified points are marked as ‘White’.



Intuition

If we plot these points on a graph, we may be able to locate some clusters or groups. Now, given an unclassified point, we can assign it to a group by observing what group its nearest neighbors belong to. This means a point close to a cluster of points classified as ‘Red’ has a higher probability of getting classified as ‘Red’.

Intuitively, we can see that the first point (2.5, 7) should be classified as ‘Green’ and the second point (5.5, 4.5) should be classified as ‘Red’.

Algorithm

Let m be the number of training data samples. Let p be an unknown point.

1. Store the training samples in an array of data points arr[]. This means each element of this array represents a tuple (x, y).

```
2.for i=0 to m:
    Calculate Euclidean distance d(arr[i], p).
```

3. Make set S of K smallest distances obtained. Each of these distances corresponds to an already classified data point.
4. Return the majority label among S.

K can be kept as an odd number so that we can calculate a clear majority in the case where only two groups are possible (e.g. Red/Blue). With increasing K, we get smoother, more defined boundaries across different classifications. Also, the accuracy of the above classifier increases as we increase the number of data points in the training set.

K Nearest Neighbors

Get Data

We'll use the ISLR package to get the data, you can download it with the code below. Remember to call the library as well.

In [2]:

```
#install.packages('ISLR',repos = 'http://cran.us.r-project.org')
```

In [3]:

```
library(ISLR)
```

We will apply the KNN approach to the Caravan data set, which is part of the ISLR library. This data set includes 85 predictors that measure demographic characteristics for 5,822 individuals. The response variable is Purchase, which indicates whether or not a given individual purchases a Caravan insurance policy. In this data set, only 6% of people purchased caravan insurance.

Let's look at the structure:

In [4]:

```
str(Caravan)
```

```
'data.frame': 5822 obs. of  86 variables:
 $ MOSTYPE : num  33 37 37 9 40 23 39 33 33 11 ...
 $ MAANTHUI: num  1 1 1 1 1 1 2 1 1 2 ...
 $ MGEMOMV : num  3 2 2 3 4 2 3 2 2 3 ...
 $ MGEMLEEF: num  2 2 2 3 2 1 2 3 4 3 ...
 $ MOSHOOFD: num  8 8 8 3 10 5 9 8 8 3 ...
 $ MGODRK  : num  0 1 0 2 1 0 2 0 0 3 ...
 $ MGODPR  : num  5 4 4 3 4 5 2 7 1 5 ...
 $ MGODOV  : num  1 1 2 2 1 0 0 0 3 0 ...
 $ MGODGE  : num  3 4 4 4 4 5 5 2 6 2 ...
 $ MRELGE  : num  7 6 3 5 7 0 7 7 6 7 ...
 $ MRELSA  : num  0 2 2 2 1 6 2 2 0 0 ...
 $ MRELOV  : num  2 2 4 2 2 3 0 0 3 2 ...
 $ MFALLEEN: num  1 0 4 2 2 3 0 0 3 2 ...
 $ MFGEKIND: num  2 4 4 3 4 5 3 5 3 2 ...
 $ MFWEKIND: num  6 5 2 4 4 2 6 4 3 6 ...
 $ MOPLHOOG: num  1 0 0 3 5 0 0 0 0 0 ...
 $ MOPLMIDD: num  2 5 5 4 4 5 4 3 1 4 ...
 $ MOPLLAAG: num  7 4 4 2 0 4 5 6 8 5 ...
 $ MBERHOOG: num  1 0 0 4 0 2 0 2 1 2 ...
 $ MBERZELF: num  0 0 0 0 5 0 0 0 1 0 ...
 $ MBERBOER: num  1 0 0 0 4 0 0 0 0 0 ...
 $ MBERMIDD: num  2 5 7 3 0 4 4 2 1 3 ...
 $ MBERARBG: num  5 0 0 1 0 2 1 5 8 3 ...
 $ MBERARBO: num  2 4 2 2 0 2 5 2 1 3 ...
 $ MSKA    : num  1 0 0 3 9 2 0 2 1 1 ...
 $ MSKB1   : num  1 2 5 2 0 2 1 1 1 2 ...
 $ MSKB2   : num  2 3 0 1 0 2 4 2 0 1 ...
 $ MSKC    : num  6 5 4 4 0 4 5 5 8 4 ...
 $ MSKD    : num  1 0 0 0 0 2 0 2 1 2 ...
 $ MHHUUR  : num  1 2 7 5 4 9 6 0 9 0 ...
 $ MHKOOP  : num  8 7 2 4 5 0 3 9 0 9 ...
 $ MAUT1   : num  8 7 7 9 6 5 8 4 5 6 ...
 $ MAUT2   : num  0 1 0 0 2 3 0 4 2 1 ...
 $ MAUT0   : num  1 2 2 0 1 3 1 2 3 2 ...
 $ MZFONDS : num  8 6 9 7 5 9 9 6 7 6 ...
 $ MZPART  : num  1 3 0 2 4 0 0 3 2 3 ...
 $ MINKM30 : num  0 2 4 1 0 5 4 2 7 2 ...
 $ MINK3045: num  4 0 5 5 0 2 3 5 2 3 ...
 $ MINK4575: num  5 5 0 3 9 3 3 3 1 3 ...
 $ MINK7512: num  0 2 0 0 0 0 0 0 0 1 ...
 $ MINK123M: num  0 0 0 0 0 0 0 0 0 0 ...
```

```
 $ MINKGEM : num  4 5 3 4 6 3 3 3 2 4 ...
 $ MKOOPKLA: num  3 4 4 4 3 3 5 3 3 7 ...
 $ PWAPART : num  0 2 2 0 0 0 0 0 0 2 ...
 $ PWABEDR : num  0 0 0 0 0 0 0 0 0 0 ...
 $ PWALAND : num  0 0 0 0 0 0 0 0 0 0 ...
 $ PPERSAUT: num  6 0 6 6 0 6 6 0 5 0 ...
 $ PBESAUT : num  0 0 0 0 0 0 0 0 0 0 ...
 $ PMOTSCO : num  0 0 0 0 0 0 0 0 0 0 ...
 $ PVRAAUT : num  0 0 0 0 0 0 0 0 0 0 ...
 $ PAANHANG: num  0 0 0 0 0 0 0 0 0 0 ...
 $ PTRACTOR: num  0 0 0 0 0 0 0 0 0 0 ...
 $ PWERKT  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ PBROM   : num  0 0 0 0 0 0 0 3 0 0 ...
 $ PLEVEN  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ PPERSONG: num  0 0 0 0 0 0 0 0 0 0 ...
 $ PGEZONG : num  0 0 0 0 0 0 0 0 0 0 ...
 $ PWAOREG : num  0 0 0 0 0 0 0 0 0 0 ...
 $ PBRAND  : num  5 2 2 2 6 0 0 0 0 3 ...
 $ PZEILPL : num  0 0 0 0 0 0 0 0 0 0 ...
 $ PPLEZIER: num  0 0 0 0 0 0 0 0 0 0 ...
 $ PFIETS  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ PINBOED : num  0 0 0 0 0 0 0 0 0 0 ...
 $ PBYSTAND: num  0 0 0 0 0 0 0 0 0 0 ...
 $ AWAPART : num  0 2 1 0 0 0 0 0 0 1 ...
 $ AWABEDR : num  0 0 0 0 0 0 0 0 0 0 ...
 $ AWALAND : num  0 0 0 0 0 0 0 0 0 0 ...
 $ APERSAUT: num  1 0 1 1 0 1 1 0 1 0 ...
 $ ABESAUT : num  0 0 0 0 0 0 0 0 0 0 ...
 $ AMOTSCO : num  0 0 0 0 0 0 0 0 0 0 ...
 $ AVRAAUT : num  0 0 0 0 0 0 0 0 0 0 ...
 $ AAANHANG: num  0 0 0 0 0 0 0 0 0 0 ...
 $ ATRACTOR: num  0 0 0 0 0 0 0 0 0 0 ...
 $ AWERKT  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ ABROM   : num  0 0 0 0 0 0 0 1 0 0 ...
 $ ALEVEN  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ APERSONG: num  0 0 0 0 0 0 0 0 0 0 ...
 $ AGEZONG : num  0 0 0 0 0 0 0 0 0 0 ...
 $ AWAOREG : num  0 0 0 0 0 0 0 0 0 0 ...
 $ ABRAND  : num  1 1 1 1 1 0 0 0 0 1 ...
 $ AZEILPL : num  0 0 0 0 0 0 0 0 0 0 ...
 $ APLEZIER: num  0 0 0 0 0 0 0 0 0 0 ...
 $ AFIETS  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ AINBOED : num  0 0 0 0 0 0 0 0 0 0 ...
 $ ABYSTAND: num  0 0 0 0 0 0 0 0 0 0 ...
 $ Purchase: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
```

In [5]:

```
summary(Caravan$Purchase)
```

Out[5]:

No
5474
Yes
348

Cleaning Data

Since we are just using this data as a simple example, we won't worry about feature engineering. Let's just remove any NA values by dropping the rows with them.

In [6]:

```
any(is.na(Caravan))
```

Out[6]:

FALSE

Looks like we're good! Let's move on the standardize the variable features:

Standardize Variables

Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. Any variables that are on a large scale will have a much larger effect on the distance between the observations, and hence on the KNN classifier, than variables that are on a small scale.

For example, let's check out the variance of two features:

In [7]:

```
var(Caravan[,1])
```

Out[7]:

165.037847395189

In [8]:

```
var(Caravan[,2])
```

Out[8]:

0.164707781931954

Clearly the scales are different! We are now going to standarize all the X variables except Y (Purchase). The Purchase variable is in column 86 of our dataset, so let's save it in a separate variable because the knn() function needs it as a separate argument.

In [9]:

```
# save the Purchase column in a separate variable
purchase <- Caravan[,86]

# Standarize the dataset using "scale()" R function
standardized.Caravan <- scale(Caravan[,-86])
```

Let's check the variance again:

In [10]:

```
var(standardized.Caravan[,1])
```

Out[10]:

1

In [11]:

```
var(standardized.Caravan[,2])
```

Out[11]:

1

We can see that now that all independent variables (X's) have a mean of 1 and standard deviation of 0. Great, then let's divide our dataset into testing and training data. We'll just do a simple split of the first 1000 rows as a test set:

In [12]:

```
# First 100 rows for test set
test.index <- 1:1000
test.data <- standardized.Caravan[test.index,]
test.purchase <- purchase[test.index]
```

In [13]:

```
# Rest of data for training
train.data <- standardized.Caravan[-test.index,]
train.purchase <- purchase[-test.index]
```

Using KNN

Rememeber that we are trying to come up with a model to predict whether someone will purchase or not. We will use the knn() function to do so, and we will focus on 4 of its arguments that we need to specify. The first argument is a data frame that contains the training data set(remember that we don't have the Y here), the second argument is a data frame that contains the testing data set (again no Y variable), the third argument is the train.purchase column (Y) that we save earlier, and the fourth argument is the k (how many neighbors). Let's start with k = 1. knn() function returns a vector of predicted Y's.

In [15]:

```
library(class)
```

In [16]:

```
set.seed(101)
predicted.purchase <- knn(train.data,test.data,train.purchase,k=1)
head(predicted.purchase)
```

Out[16]:

No No No No No No

Now let's evaluate the model we trained and see our misclassification error rate.

In [17]:

```
mean(test.purchase != predicted.purchase)
```

Out[17]:

0.116

Choosing K Value

Let's see what happens when we choose a different K value:

In [18]:

```
predicted.purchase <- knn(train.data,test.data,train.purchase,k=3)
mean(test.purchase != predicted.purchase)
```

Out[18]:

0.073

Interesting! Our Misclassification rate went down! What about k=5?

In [19]:

```
predicted.purchase <- knn(train.data,test.data,train.purchase,k=5)
mean(test.purchase != predicted.purchase)
```

Out[19]:

0.066

Should we manually change k and see which k gives us the minimal misclassification rate? NO! we have computers, so let's automate the process with a for() loop. A loop in R repeats the same command as much as you specify. For example, if we want to

automate the process with a for() loop. A loop in R repeats the same command as much as you specify. For example, if we want to check for k =1 up to 100, then we have to write 3 x 100 lines of code, but with a for loop, you just need 4 lines of code, and you can repeat those 3 lines up to as many as you want. (Note this may take awhile because you're running the model 20 times!)

Note on NULL versus NA

In [20]:

```
predicted.purchase = NULL
error.rate = NULL

for(i in 1:20){
    set.seed(101)
    predicted.purchase = knn(train.data,test.data,train.purchase,k=i)
    error.rate[i] = mean(test.purchase != predicted.purchase)
}
```

In [21]:

```
print(error.rate)
```

```
 [1] 0.116 0.107 0.074 0.070 0.066 0.064 0.062 0.061 0.058 0.058 0.059 0.058
[13] 0.059 0.059 0.059 0.059 0.059 0.059 0.059 0.059
```

Elbow Method

We can plot out the various error rates for the K values. We should see an "elbow" indicating that we don't get a decrease in error rate for using a higher K. This is a good cut-off point:

In [22]:

```
library(ggplot2)
```

In [23]:

```
k.values <- 1:20
```

In [24]:

```
error.df <- data.frame(error.rate,k.values)
```

In [25]:

```
error.df
```

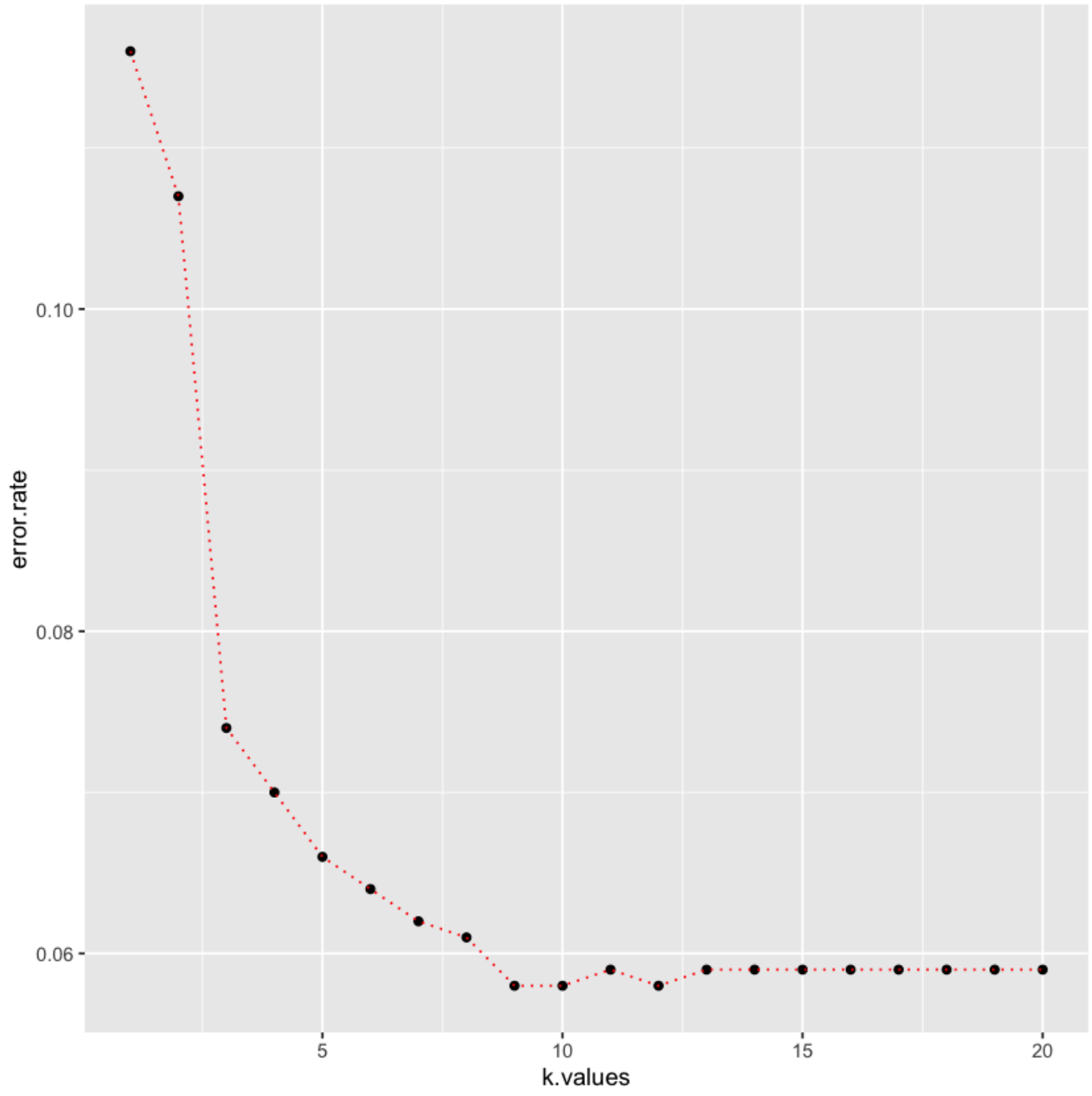
Out[25]:

	error.rate	k.values
1	0.116	1
2	0.107	2
3	0.074	3
4	0.07	4
5	0.066	5
6	0.064	6
7	0.062	7
8	0.061	8
9	0.058	9
10	0.058	10
11	0.059	11
12	0.058	12

	error.rate	k.values
13	0.059	13
14	0.059	14
15	0.059	15
16	0.059	16
17	0.059	17
18	0.059	18
19	0.059	19
20	0.059	20

In [26]:

```
ggplot(error.df,aes(x=k.values,y=error.rate)) + geom_point()+ geom_line(lty="dotted",color='red')
```



Here we can clearly see that increasing beyond K=9 does not help our misclassification at all. So we can set that as the K for our model during training.

Great Job!

K means Clustering – Introduction

We are given a data set of items, with certain features, and values for these features (like a vector). The task is to categorize those items into groups. To achieve this, we will use the kMeans algorithm; an unsupervised learning algorithm.

Overview

(It will help if you think of items as points in an n-dimensional space). The algorithm will categorize the items into k groups of similarity. To calculate that similarity, we will use the euclidean distance as measurement.

The algorithm works as follows:

1. First we initialize k points, called means, randomly.
2. We categorize each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that mean so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.

The "points" mentioned above are called means, because they hold the mean values of the items categorized in it. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another method is to initialize the means at random values between the boundaries of the data set (if for a feature *x* the items have values in [0,3], we will initialize the means with values for *x* at [0,3]).

The above algorithm in pseudocode:

```
Initialize k means with random values

For a given number of iterations:
    Iterate through items:
        Find the mean closest to the item
        Assign item to mean
```

```
        Update mean
```

Read Data

We receive input as a text file (‘data.txt’). Each line represents an item, and it contains numerical values (one for each feature) split by commas. You can find a sample data set here.

We will read the data from the file, saving it into a list. Each element of the list is another list containing the item values for the features. We do this with the following function:

K-means Clustering

Method Used

K Means Clustering is an unsupervised learning algorithm that tries to cluster data based on their similarity. Unsupervised learning means that there is no outcome to be predicted, and the algorithm just tries to find patterns in the data. In k means clustering, we have the specify the number of clusters we want the data to be grouped into. The algorithm randomly assigns each observation to a cluster, and finds the centroid of each cluster. Then, the algorithm iterates through two steps:

Reassign data points to the cluster whose centroid is closest. Calculate new centroid of each cluster. These two steps are repeated till the within cluster variation cannot be reduced any further. The within cluster variation is calculated as the sum of the euclidean distance between the data points and their respective cluster centroids.

Get the Data

We will use the iris data set. The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by Ronald Fisher in his 1936 paper *The use of multiple measurements in taxonomic problems as an example of linear discriminant analysis.*

The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimetres.

In [1]:

```
library(datasets)
```

In [2]:

```
head(iris)
```

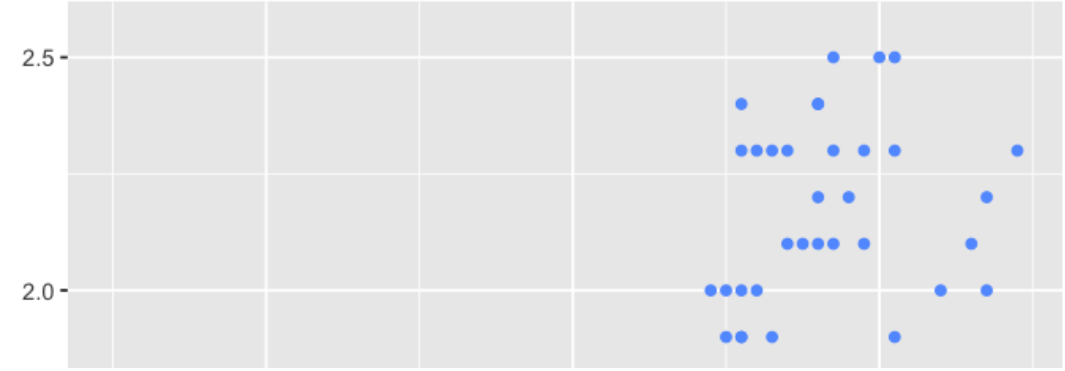
Out[2]:

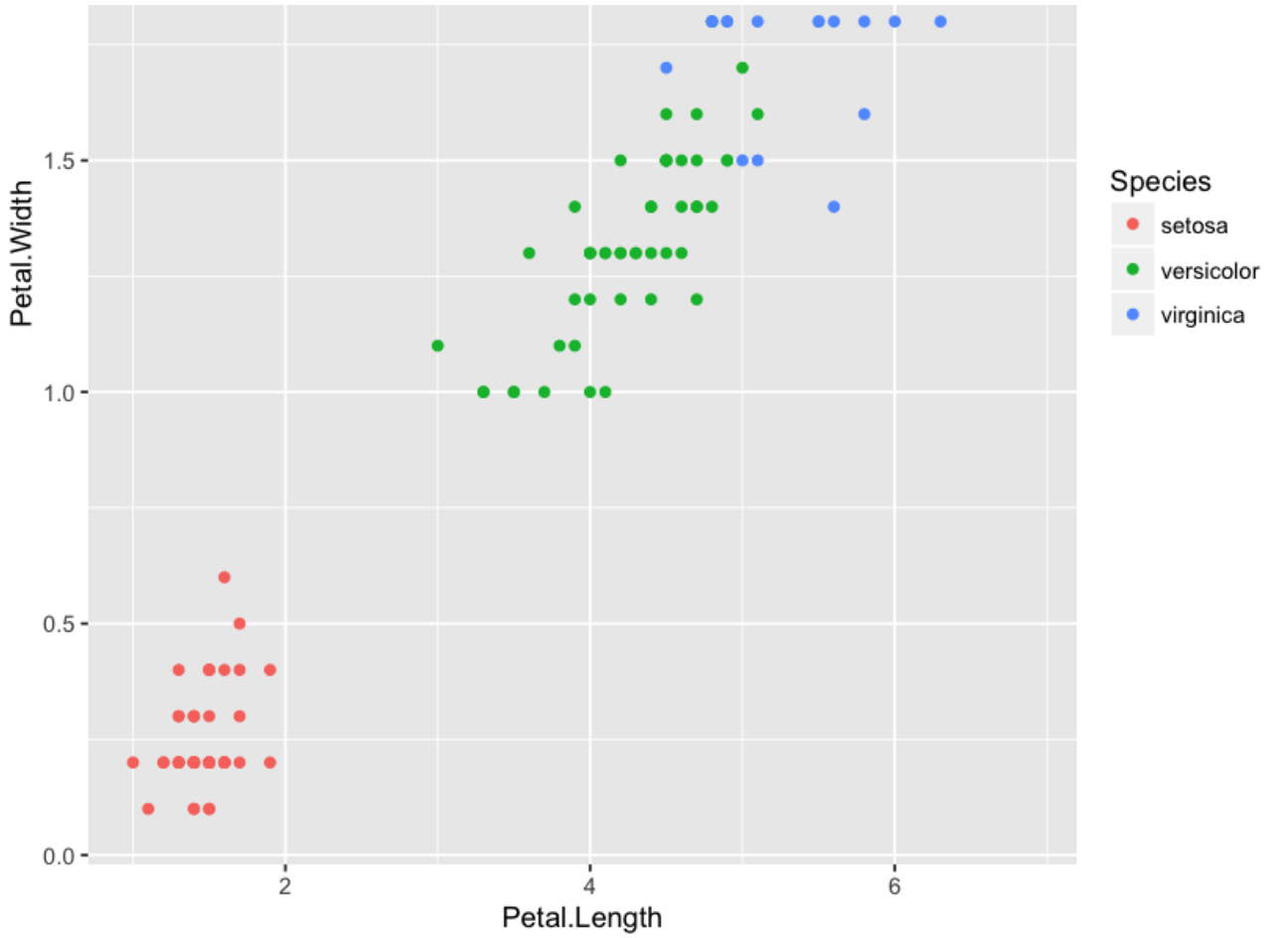
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

EDA

In [3]:

```
library(ggplot2)
ggplot(iris, aes(Petal.Length, Petal.Width, color = Species)) + geom_point()
```





Clustering

Now let's attempt to use the K-means algorithm to cluster the data. Remember that this is an unsupervised learning algorithm, meaning we won't give it any information on the correct labels:

In [9]:

```
set.seed(101)
```

In [10]:

```
help(kmeans)
```

Out[10]:

kmeans {stats}	R Documentation

K-Means Clustering

Description

Perform k-means clustering on a data matrix.

Usage

```
kmeans(x, centers, iter.max = 10, nstart = 1,
       algorithm = c("Hartigan-Wong", "Lloyd", "Forgy",
                     "MacQueen"), trace=FALSE)
## S3 method for class 'kmeans'
fitted(object, method = c("centers", "classes"), ...)
```

Arguments

	numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame

`x`	numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns).
`centers`	either the number of clusters, say *k*, or a set of initial (distinct) cluster centres. If a number, a random set of (distinct) rows in `x` is chosen as the initial centres.
`iter.max`	the maximum number of iterations allowed.
`nstart`	if `centers` is a number, how many random sets should be chosen?
`algorithm`	character: may be abbreviated. Note that `"Lloyd"` and `"Forgy"` are alternative names for one algorithm.
`object`	an **R** object of class `"kmeans"`, typically the result `ob` of `ob <- kmeans(..)`.
`method`	character: may be abbreviated. `"centers"` causes `fitted` to return cluster centers (one for each input point) and `"classes"` causes `fitted` to return a vector of class assignments.
`trace`	logical or integer number, currently only used in the default method (`"Hartigan-Wong"`): if positive (or true), tracing information on the progress of the algorithm is produced. Higher values may produce more tracing information.
`...`	not used.

Details

The data given by `x` are clustered by the *k*-means method, which aims to partition the points into *k* groups such that the sum of squares from points to the assigned cluster centres is minimized. At the minimum, all cluster centres are at the mean of their Voronoi sets (the set of data points which are nearest to the cluster centre).

The algorithm of Hartigan and Wong (1979) is used by default. Note that some authors use *k*-means to refer to a specific algorithm rather than the general method: most commonly the algorithm given by MacQueen (1967) but sometimes that given by Lloyd (1957) and Forgy (1965). The Hartigan–Wong algorithm generally does a better job than either of those, but trying several random starts (`nstart`> *1*) is often recommended. In rare cases, when some of the points (rows of `x`) are extremely close, the algorithm may not converge in the "Quick-Transfer" stage, signalling a warning (and returning `ifault = 4`). Slight rounding of the data may be advisable in that case.

For ease of programmatic exploration, *k=1* is allowed, notably returning the center and `withinss`.

Except for the Lloyd–Forgy method, *k* clusters will always be returned if a number is specified. If an initial matrix of centres is supplied, it is possible that no point will be closest to one or more centres, which is currently an error for the Hartigan–Wong method.

Value

`kmeans` returns an object of class `"kmeans"` which has a `print` and a `fitted` method. It is a list with at least the following components:

`cluster`	A vector of integers (from `1:k`) indicating the cluster to which each point is allocated.
`centers`	A matrix of cluster centres.
`totss`	The total sum of squares.
`withinss`	Vector of within-cluster sum of squares, one component per cluster.
`tot.withinss`	Total within-cluster sum of squares, i.e. `sum(withinss)`.
`betweenss`	The between-cluster sum of squares, i.e. `totss-tot.withinss`.
`size`	The number of points in each cluster.
`iter`	The number of (outer) iterations.
`ifault`	integer: indicator of a possible algorithm problem – for experts.

References

Forgy, E. W. (1965) Cluster analysis of multivariate data: efficiency vs interpretability of classifications. *Biometrics* **21**, 768–769.

Hartigan, J. A. and Wong, M. A. (1979). A K-means clustering algorithm. *Applied Statistics* **28**, 100–108.

Lloyd, S. P. (1957, 1982) Least squares quantization in PCM. Technical Note, Bell Laboratories. Published in 1982 in *IEEE Transactions on Information Theory* **28**, 128–137.

MacQueen, J. (1967) Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, eds L. M. Le Cam & J. Neyman, **1**, pp. 281–297. Berkeley, CA: University of

California Press.

Examples

```
require(graphics)

# a 2-dimensional example
x <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),
           matrix(rnorm(100, mean = 1, sd = 0.3), ncol = 2))
colnames(x) <- c("x", "y")
(cl <- kmeans(x, 2))
plot(x, col = cl$cluster)
points(cl$centers, col = 1:2, pch = 8, cex = 2)

# sum of squares
ss <- function(x) sum(scale(x, scale = FALSE)^2)

## cluster centers "fitted" to each obs.:
fitted.x <- fitted(cl);  head(fitted.x)
resid.x <- x - fitted(cl)

## Equalities : ----------------------------------
cbind(cl[c("betweenss", "tot.withinss", "totss")], # the same two columns
         c(ss(fitted.x), ss(resid.x),    ss(x)))
stopifnot(all.equal(cl$ totss,        ss(x)),
   all.equal(cl$ tot.withinss, ss(resid.x)),
   ## these three are the same:
   all.equal(cl$ betweenss,    ss(fitted.x)),
   all.equal(cl$ betweenss, cl$totss - cl$tot.withinss),
   ## and hence also
   all.equal(ss(x), ss(fitted.x) + ss(resid.x))
   )

kmeans(x,1)$withinss # trivial one-cluster, (its W.SS == ss(x))

## random starts do help here with too many clusters
## (and are often recommended anyway!):
(cl <- kmeans(x, 5, nstart = 25))
plot(x, col = cl$cluster)
points(cl$centers, col = 1:5, pch = 8)
```

[Package *stats* version 3.2.2]

In [11]:

```
# Luckily we already know how many clusters to expect
irisCluster <- kmeans(iris[, 1:4], 3, nstart = 20)
irisCluster
```

Out[11]:

```
K-means clustering with 3 clusters of sizes 62, 50, 38

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.901613    2.748387     4.393548    1.433871
2     5.006000    3.428000     1.462000    0.246000
3     6.850000    3.073684     5.742105    2.071053

Clustering vector:
  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [75] 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 3 3 3 3 1 3 3 3 3
[112] 3 3 1 1 3 3 3 3 1 3 1 3 1 3 3 1 1 3 3 3 3 3 1 3 3 3 3 1 3 3 3 1 3 3 3
[149] 3 1

Within cluster sum of squares by cluster:
```

```
within cluster sum of squares by cluster:
[1] 39.82097 15.15100 23.87947
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

In [13]:

```
table(irisCluster$cluster, iris$Species)
```

Out[13]:

```
    setosa versicolor virginica
  1      0         48        14
  2     50          0         0
  3      0          2        36
```

We can also grab additional information about the clusters.

In [14]:

```
irisCluster
```

Out[14]:

```
             Length Class  Mode
cluster      150    -none- numeric
centers       12    -none- numeric
totss          1    -none- numeric
withinss       3    -none- numeric
tot.withinss   1    -none- numeric
betweenss      1    -none- numeric
size           3    -none- numeric
iter           1    -none- numeric
ifault         1    -none- numeric
```

Looks like we correctly grouped the setosa, with some crossover with versicolor and virginica.

Cluster Visualizations

We can plot the clusters out:

In [20]:

```
library(cluster)
```

In [36]:

```
help(clusplot)
```

Out[36]:

clusplot {cluster}	R Documentation

Bivariate Cluster Plot (of a Partitioning Object)

Description

Draws a 2-dimensional “clusplot” (clustering plot) on the current graphics device. The generic function has a default and a `partition` method.

Usage

```
clusplot(x, ...)
```

```
## S3 method for class 'partition'
clusplot(x, main = NULL, dist = NULL, ...)
```

Arguments

x	an R object, here, specifically an object of class "partition", e.g. created by one of the functions pam, clara, or fanny.
main	title for the plot; when NULL (by default), a title is constructed, using x$call.
dist	when x does not have a diss nor a data component, e.g., for pam(dist(*), keep.diss=FALSE), dist must specify the dissimilarity for the clusplot.
...	optional arguments passed to methods, notably the clusplot.default method (except for the diss one) may also be supplied to this function. Many graphical parameters (see par) may also be supplied as arguments here.

Details

The clusplot.partition() method relies on clusplot.default.

If the clustering algorithms pam, fanny and clara are applied to a data matrix of observations-by-variables then a clusplot of the resulting clustering can always be drawn. When the data matrix contains missing values and the clustering is performed with pam or fanny, the dissimilarity matrix will be given as input to clusplot. When the clustering algorithm clara was applied to a data matrix with NAs then clusplot will replace the missing values as described in clusplot.default, because a dissimilarity matrix is not available.

Value

For the partition (and default) method: An invisible list with components Distances and Shading, as for clusplot.default, see there.

Side Effects

a 2-dimensional clusplot is created on the current graphics device.

See Also

clusplot.default for references; partition.object, pam, pam.object, clara, clara.object, fanny, fanny.object, par.

Examples

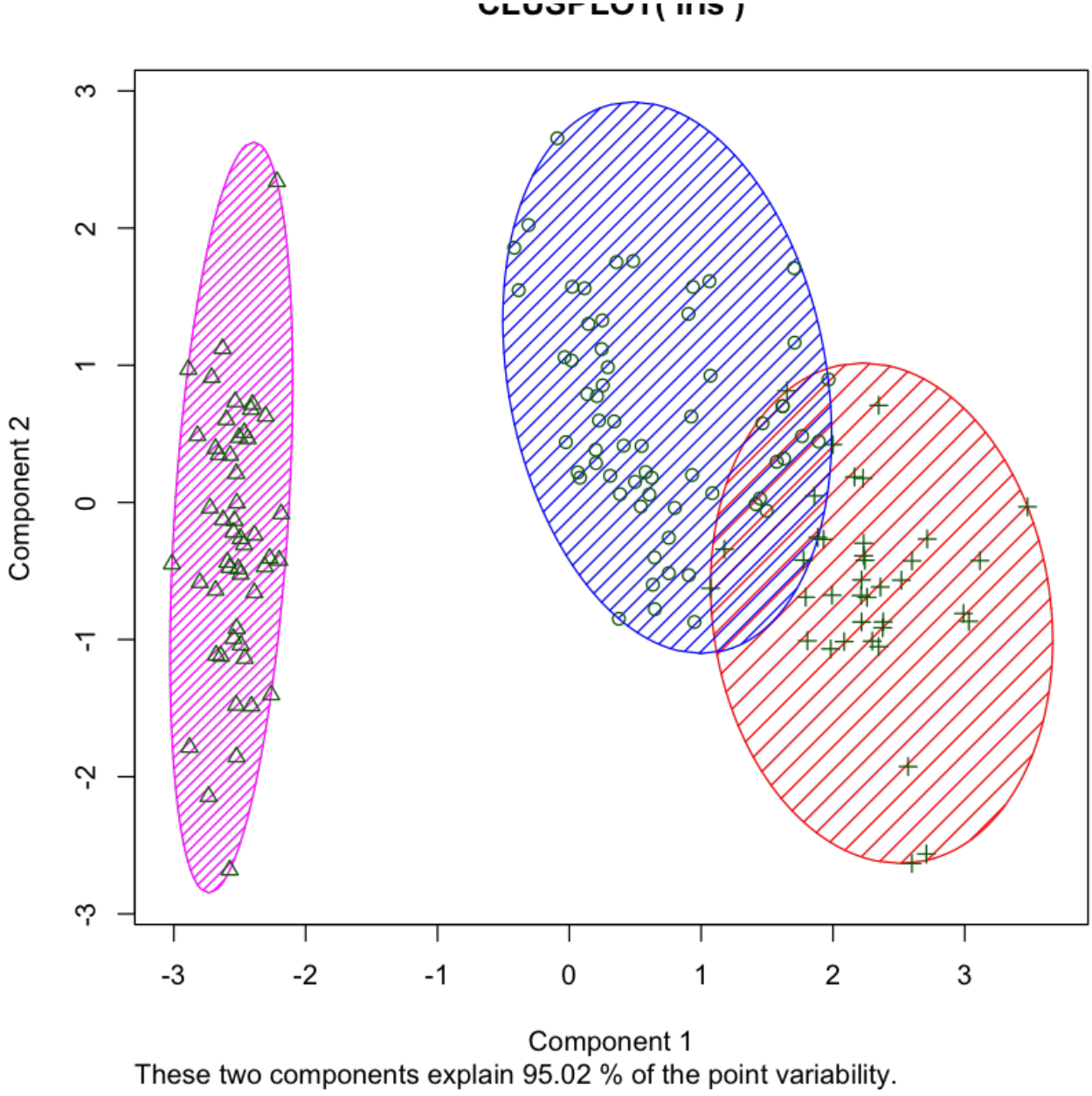
```
 ## For more, see ?clusplot.default

## generate 25 objects, divided into 2 clusters.
x <- rbind(cbind(rnorm(10,0,0.5), rnorm(10,0,0.5)),
           cbind(rnorm(15,5,0.5), rnorm(15,5,0.5)))
clusplot(pam(x, 2))
## add noise, and try again :
x4 <- cbind(x, rnorm(25), rnorm(25))
clusplot(pam(x4, 2))
```

[Package *cluster* version 2.0.3]

Here we can plot two components and the cluster regions versus the actual real data labels:

In [35]:

```
library(cluster)
clusplot(iris, irisCluster$cluster, color=TRUE, shade=TRUE, labels=0,lines=0, )
```


Component 2
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3
Component 1
These two components explain 95.02 % of the point variability.

Great Job!

Linear Regression for Machine Learning

Linear regression is perhaps one of the most well known and well understood algorithms in statistics and machine learning.

In this post you will discover the linear regression algorithm, how it works and how you can best use it in on your machine learning projects. In this post you will learn:

- Why linear regression belongs to both statistics and machine learning.
- The many names by which linear regression is known.
- The representation and learning algorithms used to create a linear regression model.
- How to best prepare your data when modeling using linear regression.

You do not need to know any statistics or linear algebra to understand linear regression. This is a gentle high-level introduction to the technique to give you enough background to be able to use it effectively on your own problems.

Let’s get started.

Isn’t Linear Regression from Statistics?

Before we dive into the details of linear regression, you may be asking yourself why we are looking at this algorithm.

Isn’t it a technique from statistics?

Machine learning, more specifically the field of predictive modeling is primarily concerned with minimizing the error of a model or making the most accurate predictions possible, at the expense of explainability. In applied machine learning we will borrow, reuse and steal algorithms from many different fields, including statistics and use them towards these ends.

As such, linear regression was developed in the field of statistics and is studied as a model for understanding the relationship between input and output numerical variables, but has been borrowed by machine learning. It is both a statistical algorithm and a machine learning algorithm.

Next, let’s review some of the common names used to refer to a linear regression model.

Many Names of Linear Regression

When you start looking into linear regression, things can get very confusing.

The reason is because linear regression has been around for so long (more than 200 years). It has been studied from every possible angle and often each angle has a new and different name.

Linear regression is a **linear model**, e.g. a model that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, that y can be calculated from a linear combination of the input variables (x).

When there is a single input variable (x), the method is referred to as **simple linear regression**. When there are **multiple input variables**, literature from statistics often refers to the method as multiple linear regression.

Different techniques can be used to prepare or train the linear regression equation from data, the most common of which is called **Ordinary Least Squares**. It is common to therefore refer to a model prepared this way as Ordinary Least Squares Linear Regression or just Least Squares Regression.

Now that we know some names used to describe linear regression, let’s take a closer look at the representation used.

Linear Regression Model Representation

Linear regression is an attractive model because the representation is so simple.

The representation is a linear equation that combines a specific set of input values (x) the solution to which is the predicted output for that set of input values (y). As such, both the input values (x) and the output value are numeric.

The linear equation assigns one scale factor to each input value or column, called a coefficient and represented by the capital Greek letter Beta (B). One additional coefficient is also added, giving the line an additional degree of freedom (e.g. moving up and down on a two-dimensional plot) and is often called the intercept or the bias coefficient.

For example, in a simple regression problem (a single x and a single y), the form of the model would be:

$$y = B0 + B1*x$$

In higher dimensions when we have more than one input (x), the line is called a plane or a hyper-plane. The representation therefore is the form of the equation and the specific values used for the coefficients (e.g. B0 and B1 in the above example).

It is common to talk about the complexity of a regression model like linear regression. This refers to the number of coefficients used in the model.

When a coefficient becomes zero, it effectively removes the influence of the input variable on the model and therefore from the prediction made from the model (0 * x = 0). This becomes relevant if you look at regularization methods that change the learning algorithm to reduce the complexity of regression models by putting pressure on the absolute size of the coefficients, driving some to zero.

Now that we understand the representation used for a linear regression model, let's review some ways that we can learn this representation from data.

Linear Regression Learning the Model

Learning a linear regression model means estimating the values of the coefficients used in the representation with the data that we have available.

In this section we will take a brief look at four techniques to prepare a linear regression model. This is not enough information to implement them from scratch, but enough to get a flavor of the computation and trade-offs involved.

There are many more techniques because the model is so well studied. Take note of Ordinary Least Squares because it is the most common method used in general.

Also take note of Gradient Descent as it is the most common technique taught in machine learning classes.

1. Simple Linear Regression

With simple linear regression when we have a single input, we can use statistics to estimate the coefficients.

This requires that you calculate statistical properties from the data such as means, standard deviations, correlations and covariance. All of the data must be available to traverse and calculate statistics.

This is fun as an exercise in excel, but not really useful in practice.

2. Ordinary Least Squares

When we have more than one input we can use Ordinary Least Squares to estimate the values of the coefficients.

The Ordinary Least Squares procedure seeks to minimize the sum of the squared residuals. This means that given a regression line through the data we calculate the distance from each data point to the regression line, square it, and sum all of the squared errors together. This is the quantity that ordinary least squares seeks to minimize.

This approach treats the data as a matrix and uses linear algebra operations to estimate the optimal values for the coefficients. It means that all of the data must be available and you must have enough memory to fit the data and perform matrix operations.

It is unusual to implement the Ordinary Least Squares procedure yourself unless as an exercise in linear algebra. It is more likely that you will call a procedure in a linear algebra library. This procedure is very fast to calculate.

3. Gradient Descent

When there are one or more inputs you can use a process of optimizing the values of the coefficients by iteratively minimizing the error of the model on your training data.

This operation is called Gradient Descent and works by starting with random values for each coefficient. The sum of the squared errors are calculated for each pair of input and output values. A learning rate is used as a scale factor and the coefficients are updated in the direction towards minimizing the error. The process is repeated until a minimum sum squared error is achieved or no further improvement is possible.

When using this method, you must select a learning rate (alpha) parameter that determines the size of the improvement step to take on each iteration of the procedure.

Gradient descent is often taught using a linear regression model because it is relatively straightforward to understand. In practice, it is useful when you have a very large dataset either in the number of rows or the number of columns that may not fit into memory.

4. Regularization

There are extensions of the training of the linear model called regularization methods. These seek to both minimize the sum of the squared error of the model on the training data (using ordinary least squares) but also to reduce the complexity of the model (like the number or absolute size of the sum of all coefficients in the model).

Two popular examples of regularization procedures for linear regression are:

- Lasso Regression: where Ordinary Least Squares is modified to also minimize the absolute sum of the coefficients (called L1 regularization).
- Ridge Regression: where Ordinary Least Squares is modified to also minimize the squared absolute sum of the coefficients (called L2 regularization).

These methods are effective to use when there is collinearity in your input values and ordinary least squares would overfit the training data.

Now that you know some techniques to learn the coefficients in a linear regression model, let's look at how we can use a model to make predictions on new data.

Making Predictions with Linear Regression

Given the representation is a linear equation, making predictions is as simple as solving the equation for a specific set of inputs.

Let's make this concrete with an example. Imagine we are predicting weight (y) from height (x). Our linear regression model representation for this problem would be:

$$y = B0 + B1 * x1$$

or

$$weight = B0 + B1 * height$$

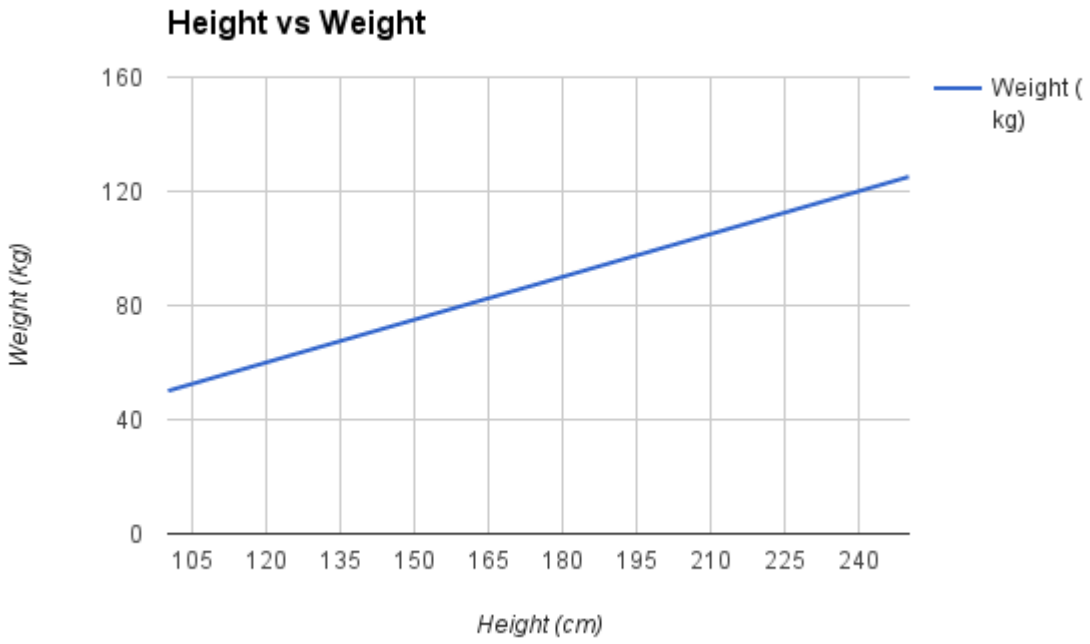
Where B0 is the bias coefficient and B1 is the coefficient for the height column. We use a learning technique to find a good set of coefficient values. Once found, we can plug in different height values to predict the weight.

For example, lets use B0 = 0.1 and B1 = 0.5. Let's plug them in and calculate the weight (in kilograms) for a person with the height of 182 centimeters.

$$weight = 0.1 + 0.5 * 182$$

$$weight = 91.1$$

You can see that the above equation could be plotted as a line in two-dimensions. The B0 is our starting point regardless of what height we have. We can run through a bunch of heights from 100 to 250 centimeters and plug them to the equation and get weight values, creating our line.



Preparing Data For Linear Regression

Linear regression is been studied at great length, and there is a lot of literature on how your data must be structured to make best use of the model.

As such, there is a lot of sophistication when talking about these requirements and expectations which can be intimidating. In practice, you can uses these rules more as

rules of thumb when using Ordinary Least Squares Regression, the most common implementation of linear regression.

Try different preparations of your data using these heuristics and see what works best for your problem.

- **Linear Assumption**. Linear regression assumes that the relationship between your input and output is linear. It does not support anything else. This may be obvious, but it is good to remember when you have a lot of attributes. You may need to transform data to make the relationship linear (e.g. log transform for an exponential relationship).
- **Remove Noise**. Linear regression assumes that your input and output variables are not noisy. Consider using data cleaning operations that let you better expose and clarify the signal in your data. This is most important for the output variable and you want to remove outliers in the output variable (y) if possible.
- **Remove Collinearity**. Linear regression will over-fit your data when you have highly correlated input variables. Consider calculating pairwise correlations for your input data and removing the most correlated.
- **Gaussian Distributions**. Linear regression will make more reliable predictions if your input and output variables have a Gaussian distribution. You may get some benefit using transforms (e.g. log or BoxCox) on you variables to make their distribution more Gaussian looking.
- **Rescale Inputs**: Linear regression will often make more reliable predictions if you rescale input variables using standardization or normalization.

Linear Regression Lecture

Remember that Linear Regression is a supervised learning algorithm, meaning we'll have labeled data and try to predict new labels on unlabeled data. We'll explore some of the following concepts:

- Get our Data
- Exploratory Data Analysis (EDA)
- Clean our Data
- Review of Model Form
- Train and Test Groups
- Linear Regression Model

Get our Data

We will use the Student Performance Data Set from UC Irvine's Machine Learning Repository! Download this data our just use the supplied csv files in the notebook repository. We'll specifically look at the math class (student-mat.csv). Make sure to take note that the delimiter is a semi-colon.

In [1]:

```
# Read CSV, note the delimiter (sep)
df <- read.csv('student-mat.csv',sep=';')
```

In [2]:

```
head(df)
```

Out[2]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason	guardian	traveltime	studytime	fa
1	GP	F	18	U	GT3	A	4	4	at_home	teacher	course	mother	2	2	0
2	GP	F	17	U	GT3	T	1	1	at_home	other	course	father	1	2	0
3	GP	F	15	U	LE3	T	1	1	at_home	other	other	mother	1	2	3
4	GP	F	15	U	GT3	T	4	2	health	services	home	mother	1	3	0
5	GP	F	16	U	GT3	T	3	3	other	other	home	father	1	2	0
6	GP	M	16	U	LE3	T	4	3	services	other	reputation	mother	1	2	0

In [3]:

```
summary(df)
```

Out[3]:

```
 school   sex          age        address famsize   Pstatus      Medu
 GP:349   F:208   Min.   :15.0   R: 88   GT3:281   A: 41   Min.   :0.000
 MS: 46   M:187   1st Qu.:16.0   U:307   LE3:114   T:354   1st Qu.:2.000
                  Median :17.0                             Median :3.000
                  Mean   :16.7                             Mean   :2.749
                  3rd Qu.:18.0                             3rd Qu.:4.000
                  Max.   :22.0                             Max.   :4.000
      Fedu             Mjob           Fjob            reason      guardian
 Min.   :0.000   at_home : 59   at_home : 20   course    :145   father: 90
 1st Qu.:2.000   health  : 34   health  : 18   home      :109   mother:273
 Median :2.000   other   :141   other   :217   other     : 36   other : 32
 Mean   :2.522   services:103   services:111   reputation:105
 3rd Qu.:3.000   teacher : 58   teacher : 29
 Max.   :4.000
   traveltime      studytime        failures      schoolsup famsup     paid
 Min.   :1.000   Min.   :1.000   Min.   :0.0000   no :344   no :153   no :214
 1st Qu.:1.000   1st Qu.:1.000   1st Qu.:0.0000   yes: 51   yes:242   yes:181
 Median :1.000   Median :2.000   Median :0.0000
 Mean   :1.448   Mean   :2.035   Mean   :0.3342
```

```
Mean   :1.440   Mean   :2.035   Mean   :0.3342
3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:0.0000
Max.   :4.000   Max.   :4.000   Max.   :3.0000
activities nursery   higher    internet  romantic      famrel
no :194    no : 81   no : 20   no : 66   no :263   Min.   :1.000
yes:201    yes:314   yes:375   yes:329   yes:132   1st Qu.:4.000
                                                   Median :4.000
                                                   Mean   :3.944
                                                   3rd Qu.:5.000
                                                   Max.   :5.000
   freetime         goout           Dalc            Walc
Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
1st Qu.:3.000   1st Qu.:2.000   1st Qu.:1.000   1st Qu.:1.000
Median :3.000   Median :3.000   Median :1.000   Median :2.000
Mean   :3.235   Mean   :3.109   Mean   :1.481   Mean   :2.291
3rd Qu.:4.000   3rd Qu.:4.000   3rd Qu.:2.000   3rd Qu.:3.000
Max.   :5.000   Max.   :5.000   Max.   :5.000   Max.   :5.000
    health         absences           G1              G2
Min.   :1.000   Min.   : 0.000   Min.   : 3.00   Min.   : 0.00
1st Qu.:3.000   1st Qu.: 0.000   1st Qu.: 8.00   1st Qu.: 9.00
Median :4.000   Median : 4.000   Median :11.00   Median :11.00
Mean   :3.554   Mean   : 5.709   Mean   :10.91   Mean   :10.71
3rd Qu.:5.000   3rd Qu.: 8.000   3rd Qu.:13.00   3rd Qu.:13.00
Max.   :5.000   Max.   :75.000   Max.   :19.00   Max.   :19.00
      G3
Min.   : 0.00
1st Qu.: 8.00
Median :11.00
Mean   :10.42
3rd Qu.:14.00
Max.   :20.00
```

Attribute Information

Here is the attribute information for our data set: Attribute Information:

Attributes for both student-mat.csv (Math course) and student-por.csv (Portuguese language course) datasets:

- 1 school - student's school (binary: 'GP' - Gabriel Pereira or 'MS' - Mousinho da Silveira)
- 2 sex - student's sex (binary: 'F' - female or 'M' - male)
- 3 age - student's age (numeric: from 15 to 22)
- 4 address - student's home address type (binary: 'U' - urban or 'R' - rural)
- 5 famsize - family size (binary: 'LE3' - less or equal to 3 or 'GT3' - greater than 3)
- 6 Pstatus - parent's cohabitation status (binary: 'T' - living together or 'A' - apart)
- 7 Medu - mother's education (numeric: 0 - none, 1 - primary education (4th grade), 2 â€“ 5th to 9th grade, 3 â€“ secondary education or 4 â€“ higher education)
- 8 Fedu - father's education (numeric: 0 - none, 1 - primary education (4th grade), 2 â€“ 5th to 9th grade, 3 â€“ secondary education or 4 â€“ higher education)
- 9 Mjob - mother's job (nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at_home' or 'other')
- 10 Fjob - father's job (nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at_home' or 'other')
- 11 reason - reason to choose this school (nominal: close to 'home', school 'reputation', 'course' preference or 'other')
- 12 guardian - student's guardian (nominal: 'mother', 'father' or 'other')
- 13 traveltime - home to school travel time (numeric: 1 - less than 15 min., 2 - 15 to 30 min., 3 - 30 min. to 1 hour, or 4 - more than 1 hour)
- 14 studytime - weekly study time (numeric: 1 - less than 2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - more than 10 hours)
- 15 failures - number of past class failures (numeric: n if between 1 and 3 , else 4)
- 16 schoolsup - extra educational support (binary: yes or no)
- 17 famsup - family educational support (binary: yes or no)
- 18 paid - extra paid classes within the course subject (Math or Portuguese) (binary: yes or no)
- 19 activities - extra-curricular activities (binary: yes or no)
- 20 nursery - attended nursery school (binary: yes or no)
- 21 higher - wants to take higher education (binary: yes or no)
- 22 internet - Internet access at home (binary: yes or no)
- 23 romantic - with a romantic relationship (binary: yes or no)
- 24 famrel - quality of family relationships (numeric: from 1 - very bad to 5 - excellent)
- 25 freetime - free time after school (numeric: from 1 - very low to 5 - very high)
- 26 goout - going out with friends (numeric: from 1 - very low to 5 - very high)

- 27 Dalc - workday alcohol consumption (numeric: from 1 - very low to 5 - very high)
- 28 Walc - weekend alcohol consumption (numeric: from 1 - very low to 5 - very high)
- 29 health - current health status (numeric: from 1 - very bad to 5 - very good)
- 30 absences - number of school absences (numeric: from 0 to 93)

these grades are related with the course subject, Math or Portuguese:

- 31 G1 - first period grade (numeric: from 0 to 20)
- 31 G2 - second period grade (numeric: from 0 to 20)
- 32 G3 - final grade (numeric: from 0 to 20, output target)

Clean the Data

Next we have to clean this data. This data is actually already cleaned for you, But here are some things you may want to consider doing for other data sets:

Check for NA values

Let's see if we have any NA values:

In [4]:

```
any(is.na(df))
```

Out[4]:

FALSE

Great! Most real data sets will probably have NA or Null values, so its always good to check! Its up to you how to deal with them, either dropping them if they aren't too many, or imputing other values, like the mean value.

Categorical Features

Moving on, let's make sure that categorical variables have a factor set to them. For example, the MJob column refers to categories of Job Types, not some numeric value from 1 to 5. R is actually really good at detecting these sort of values and will take of this work for you a lot of the time, but always keep in mind the use of **factor()** as a possible. Luckily this is basically already, we can check this using the str() function:

In [5]:

```
str(df)
```

```
'data.frame': 395 obs. of  33 variables:
 $ school    : Factor w/ 2 levels "GP","MS": 1 1 1 1 1 1 1 1 1 1 ...
 $ sex       : Factor w/ 2 levels "F","M": 1 1 1 1 1 2 2 1 2 2 ...
 $ age       : int  18 17 15 15 16 16 16 17 15 15 ...
 $ address   : Factor w/ 2 levels "R","U": 2 2 2 2 2 2 2 2 2 2 ...
 $ famsize   : Factor w/ 2 levels "GT3","LE3": 1 1 2 1 1 2 2 1 2 1 ...
 $ Pstatus   : Factor w/ 2 levels "A","T": 1 2 2 2 2 2 2 1 1 2 ...
 $ Medu      : int  4 1 1 4 3 4 2 4 3 3 ...
 $ Fedu      : int  4 1 1 2 3 3 2 4 2 4 ...
 $ Mjob      : Factor w/ 5 levels "at_home","health",..: 1 1 1 2 3 4 3 3 4 3 ...
 $ Fjob      : Factor w/ 5 levels "at_home","health",..: 5 3 3 4 3 3 3 5 3 3 ...
 $ reason    : Factor w/ 4 levels "course","home",..: 1 1 3 2 2 4 2 2 2 2 ...
 $ guardian  : Factor w/ 3 levels "father","mother",..: 2 1 2 2 1 2 2 2 2 2 ...
 $ traveltime: int  2 1 1 1 1 1 1 2 1 1 ...
 $ studytime : int  2 2 2 3 2 2 2 2 2 2 ...
 $ failures  : int  0 0 3 0 0 0 0 0 0 0 ...
 $ schoolsup : Factor w/ 2 levels "no","yes": 2 1 2 1 1 1 1 2 1 1 ...
 $ famsup    : Factor w/ 2 levels "no","yes": 1 2 1 2 2 2 1 2 2 2 ...
 $ paid      : Factor w/ 2 levels "no","yes": 1 1 2 2 2 2 1 1 2 2 ...
 $ activities: Factor w/ 2 levels "no","yes": 1 1 1 2 1 2 1 1 1 2 ...
 $ nursery   : Factor w/ 2 levels "no","yes": 2 1 2 2 2 2 2 2 2 2 ...
 $ higher    : Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...
 $ internet  : Factor w/ 2 levels "no","yes": 1 2 2 2 1 2 2 1 2 2 ...
 $ romantic  : Factor w/ 2 levels "no","yes": 1 1 1 2 1 1 1 1 1 1 ...
 $ famrel    : int  4 5 4 3 4 5 4 4 4 5 ...
 $ freetime  : int  3 3 3 2 3 4 4 1 2 5 ...
 $ goout     : int  4 3 2 2 2 2 4 4 2 1 ...
```

```
 $ Dalc      : int  1 1 2 1 1 1 1 1 1 1 ...
 $ Walc      : int  1 1 3 1 2 2 1 1 1 1 ...
 $ health    : int  3 3 3 5 5 5 3 1 1 5 ...
 $ absences  : int  6 4 10 2 4 10 0 6 0 0 ...
 $ G1        : int  5 5 7 15 6 15 12 6 16 14 ...
 $ G2        : int  6 5 8 14 10 15 12 5 18 15 ...
 $ G3        : int  6 6 10 15 10 15 11 6 19 15 ...
```

Exploratory Data Analysis

Let's use ggplot2 to explore the data a bit. Feel free to expand on this section:

In [6]:

```
library(ggplot2)
library(ggthemes)
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

Correlation and CorrPlots

From Wikipedia, correlation is defined as:

- In statistics, dependence or association is any statistical relationship, whether causal or not, between two random variables or two sets of data. Correlation is any of a broad class of statistical relationships involving dependence, though in common usage it most often refers to the extent to which two variables have a linear relationship with each other. Familiar examples of dependent phenomena include the correlation between the physical statures of parents and their offspring, and the correlation between the demand for a product and its price.

Correlation plots are a great way of exploring data and seeing if there are any interaction terms. Let's start off by just grabbing the numeric data (we can't see correlation for categorical data):

In [7]:

```
# Grab only numeric columns
num.cols <- sapply(df, is.numeric)

# Filter to numeric columns for correlation
cor.data <- cor(df[,num.cols])

cor.data
```

Out[7]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goo
age	1.000000000	-0.163658419	-0.163438069	0.070640721	-0.004140037	0.243665377	0.053940096	0.016434389	0.12
Medu	-0.163658419	1.000000000	0.623455112	-0.171639305	0.064944137	-0.236679963	-0.003914458	0.030890867	0.06
Fedu	-0.163438069	0.623455112	1.000000000	-0.158194054	-0.009174639	-0.250408444	-0.001369727	-0.012845528	0.04
traveltime	0.070640721	-0.171639305	-0.158194054	1.000000000	-0.100909119	0.092238746	-0.016807986	-0.017024944	0.02
studytime	-0.004140037	0.064944137	-0.009174639	-0.100909119	1.000000000	-0.173563031	0.039730704	-0.143198407	-0.06

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goo
failures	0.24366538	0.23667996	0.25040844	0.09223875	-0.17356303	1.00000000	-0.04433663	0.09198747	0.12
famrel	0.053940096	-0.003914458	-0.001369727	-0.016807986	0.039730704	-0.044336626	1.000000000	0.150701444	0.06
freetime	0.01643439	0.03089087	-0.01284553	-0.01702494	-0.14319841	0.09198747	0.15070144	1.00000000	0.28
goout	0.126963880	0.064094438	0.043104668	0.028539674	-0.063903675	0.124560922	0.064568411	0.285018715	1.00
Dalc	0.131124605	0.019834099	0.002386429	0.138325309	-0.196019263	0.136046931	-0.077594357	0.209000848	0.26
Walc	0.11727605	-0.04712346	-0.01263102	0.13411575	-0.25378473	0.14196203	-0.11339731	0.14782181	0.42
health	-0.062187369	-0.046877829	0.014741537	0.007500606	-0.075615863	0.065827282	0.094055728	0.075733357	-0.00
absences	0.17523008	0.10028482	0.02447289	-0.01294378	-0.06270018	0.06372583	-0.04435409	-0.05807792	0.04
G1	-0.06408150	0.20534100	0.19026994	-0.09303999	0.16061192	-0.35471761	0.02216832	0.01261293	-0.1
G2	-0.14347405	0.21552717	0.16489339	-0.15319796	0.13588000	-0.35589563	-0.01828135	-0.01377714	-0.1
G3	-0.16157944	0.21714750	0.15245694	-0.11714205	0.09781969	-0.36041494	0.05136343	0.01130724	-0.1

While this is fantastic information, it's hard to take it all in. Let's visualize all this data. There are lots of amazing 3rd party packages to do this, let's use and install the 'corrgram' package and the corrplot package. This will also install a bunch of dependencies for the package.

In [8]:

```
#install.packages('corrgram',repos = 'http://cran.us.r-project.org')
#install.packages('corrplot',repos = 'http://cran.us.r-project.org')
```

In [9]:

```
library(corrplot)
library(corrgram)
```

```
Warning message:
: replacing previous import by 'magrittr::%>%' when loading 'dendextend'
```

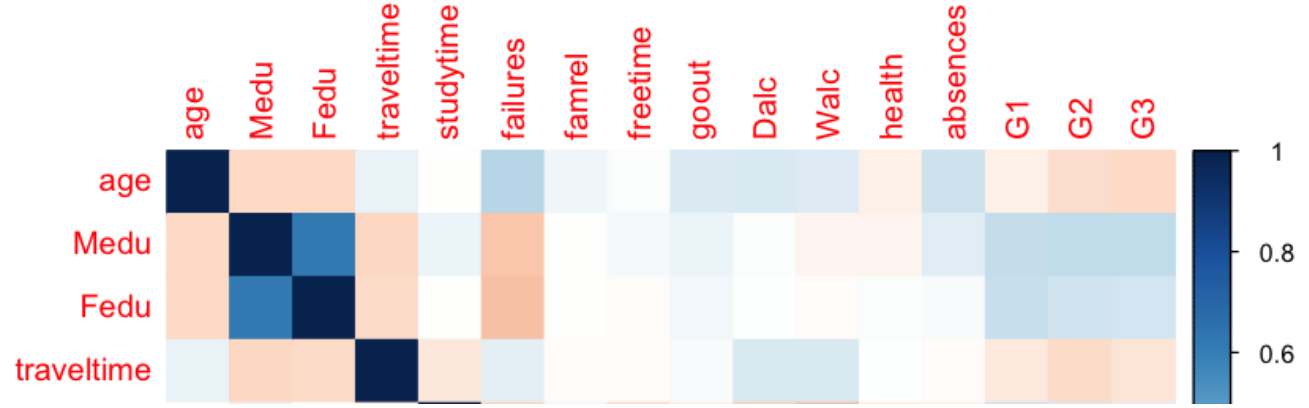
Let's start by using corrplot, the most common one. [Here's a really nice documentation page on the package.](#) I encourage you to play around with it.

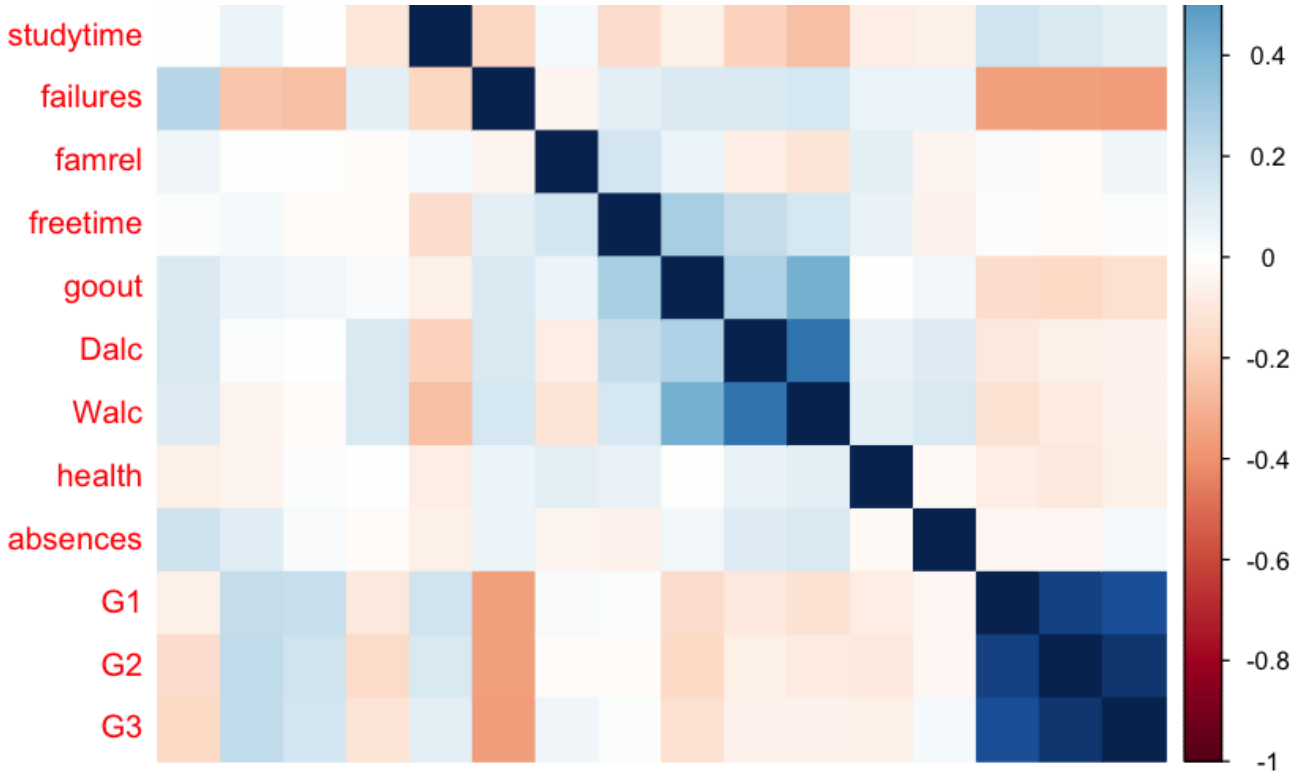
In [10]:

```
#help(corrplot)
```

In [11]:

```
corrplot(cor.data,method='color')
```





Cleary we have very high correlation between G1, G2, and G3 which makes sense since those are grades:

- 31 G1 - first period grade (numeric: from 0 to 20)
- 31 G2 - second period grade (numeric: from 0 to 20)
- 32 G3 - final grade (numeric: from 0 to 20, output target)

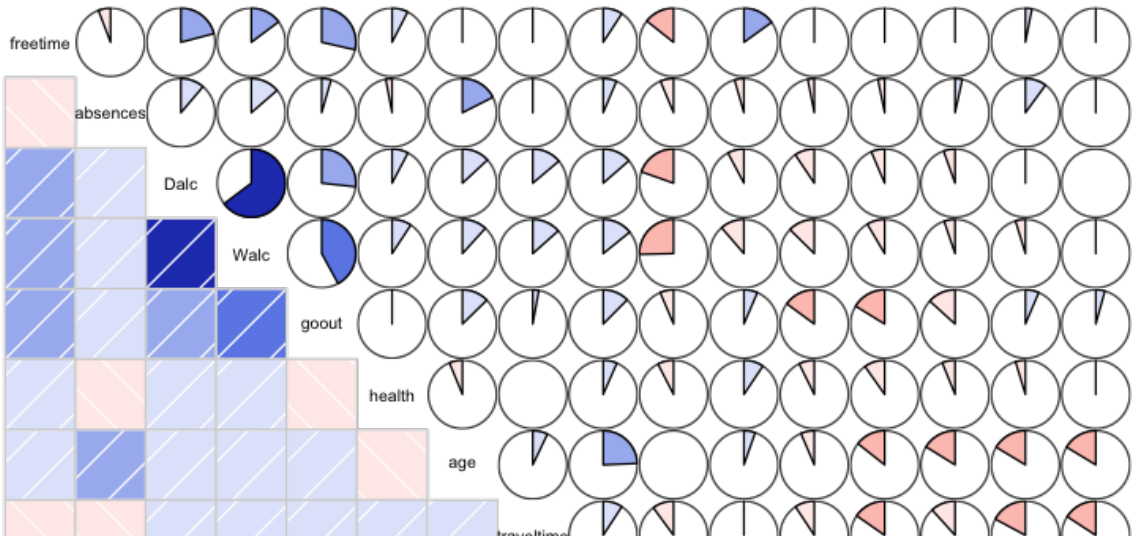
Meaning good students do well each period, and poor students do poorly each period, etc. Also a high G1,G2, or G3 value has a negative correlation with failure (number of past class failures).

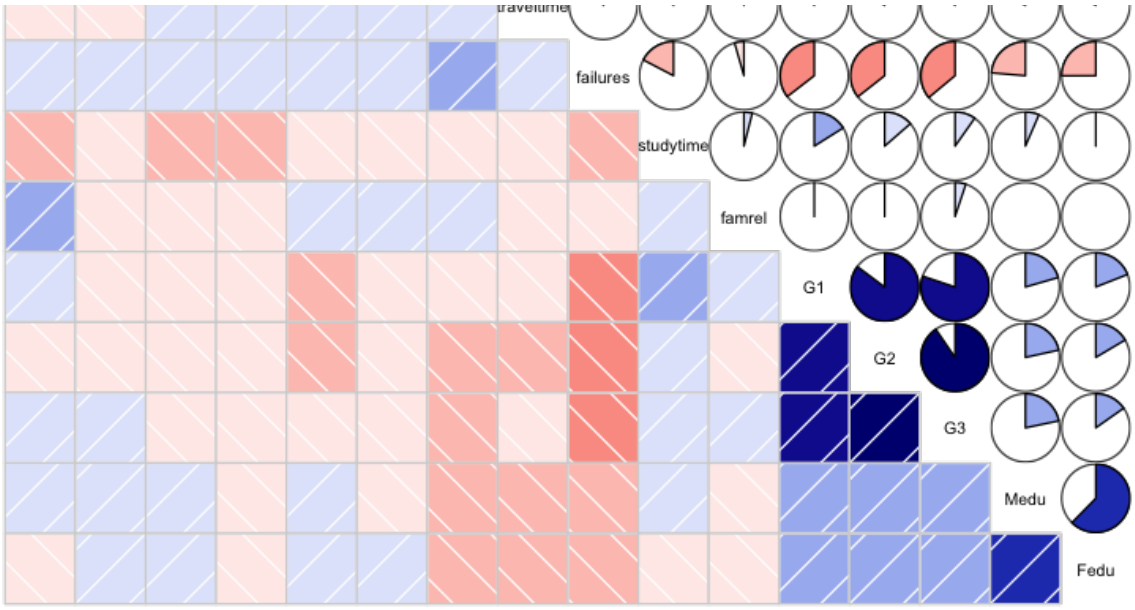
Also Mother and Father education levels are correlated, which also makes sense.

We can also use the corrgram which allows to just automatically do these type of figures by just passing in the dataframe directly. There's a lot going on here, so reference the documentation of corrgram for more info.

In [12]:

```
corrgram(df,order=TRUE, lower.panel=panel.shade,
  upper.panel=panel.pie, text.panel=panel.txt)
```

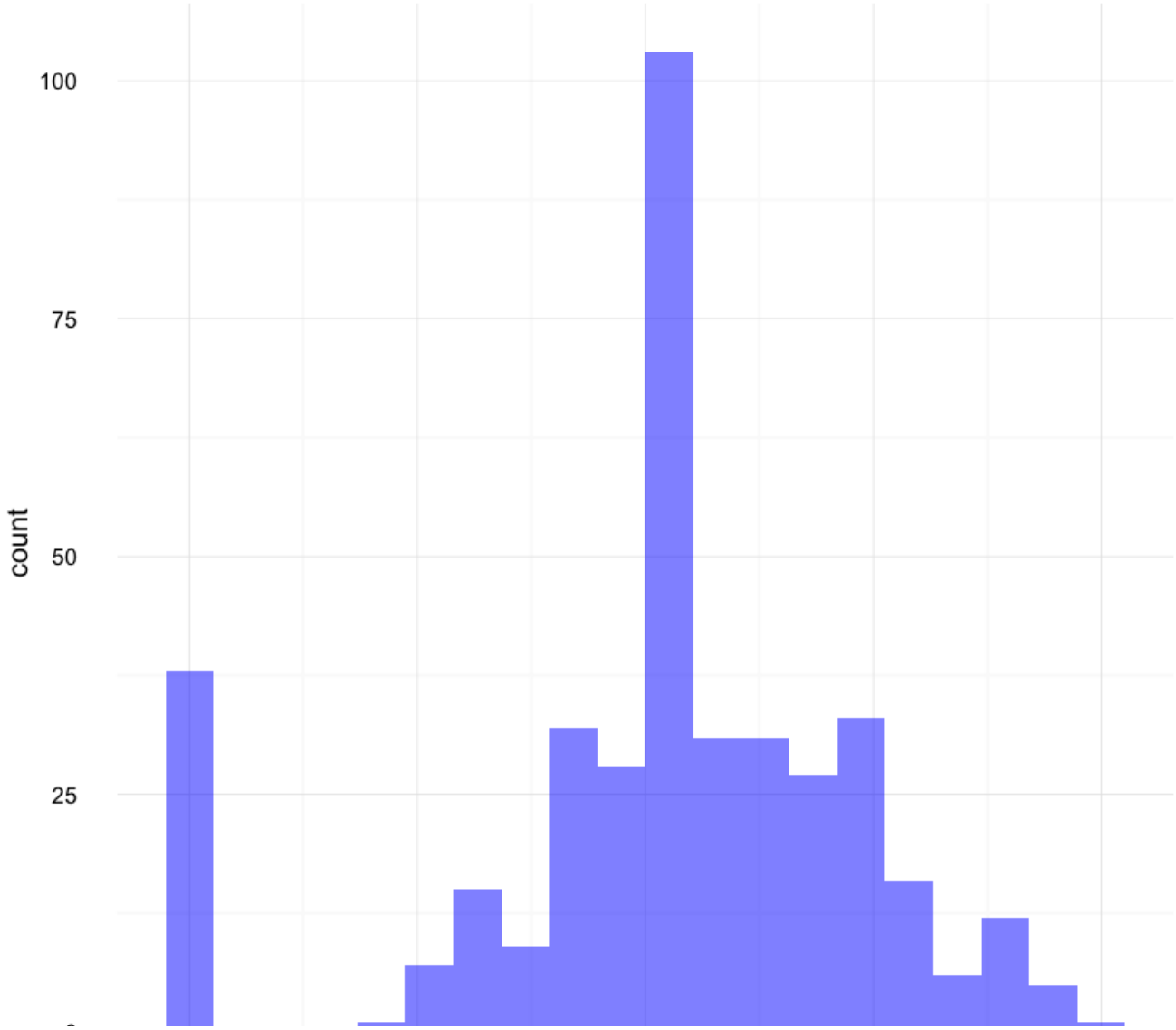


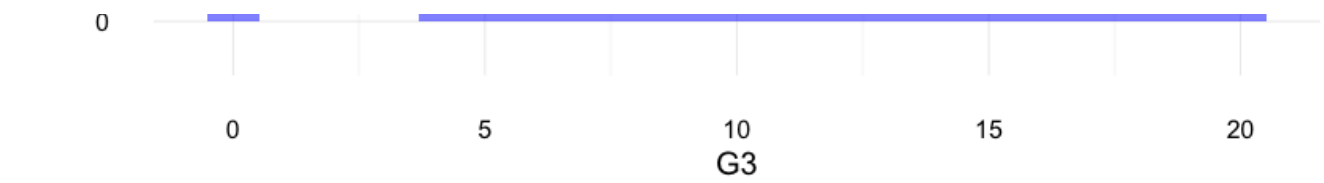


Since we're going to eventually try to predict the G3 score let's see a histogram of these scores:

In [13]:

```
ggplot(df,aes(x=G3)) + geom_histogram(bins=20,alpha=0.5,fill='blue') + theme_minimal()
```





Looks like quite a few students get a zero. This is a good place to ask questions, like are students missing the test? Also why is the mean occurence so high? Is this test curved?

Let's continue by building a model

Building a Model

General Form:

The general model of building a linear regression model in R looks like this:

```
model <- lm(y ~ x1 + x2,data)

or to use all the features in your data

model <- lm(y ~. , data) # Uses all features
```

Train and Test Data

We'll need to split our data into a training set and a testing set in order to test our accuracy. We can do this easily using the caTools library:

In [14]:

```
# Import Library
library(caTools)
# Set a random see so your "random" results are the same as this notebook
set.seed(101)

# Split up the sample, basically randomly assigns a booleans to a new column "sample"
sample <- sample.split(df$age, SplitRatio = 0.70) # SplitRatio = percent of sample==TRUE

# Training Data
train = subset(df, sample == TRUE)

# Testing Data
test = subset(df, sample == FALSE)
```

Training our Model

Let's train out model on our training data, then ask for a summary of that model:

In [15]:

```
model <- lm(G3 ~ .,train)
```

In [16]:

```
summary(model)
```

Out[16]:

```
Call:
lm(formula = G3 ~ ., data = train)

Residuals:
    Min      1Q  Median      3Q     Max
-7.7681 -0.6423  0.2294  1.0691  4.5942
```

```
Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)      -1.329568   2.474569  -0.537 0.591574
schoolMS          0.838581   0.470545   1.782 0.076016 .
sexM              0.034883   0.275586   0.127 0.899382
age              -0.214994   0.119579  -1.798 0.073472 .
addressU          0.067190   0.326035   0.206 0.836905
famsizeLE3       -0.111068   0.283228  -0.392 0.695302
PstatusT         -0.153653   0.401679  -0.383 0.702417
Medu              0.279949   0.171111   1.636 0.103164
Fedu             -0.221275   0.151103  -1.464 0.144422
Mjobhealth        0.002065   0.610532   0.003 0.997304
Mjobother         0.509947   0.403195   1.265 0.207209
Mjobservices      0.475476   0.435332   1.092 0.275857
Mjobteacher       0.285345   0.550640   0.518 0.604802
Fjobhealth        0.433172   0.774191   0.560 0.576343
Fjobother        -0.296792   0.577217  -0.514 0.607611
Fjobservices     -0.311595   0.593628  -0.525 0.600148
Fjobteacher      -0.321205   0.712695  -0.451 0.652628
reasonhome       -0.431435   0.319907  -1.349 0.178755
reasonother       0.159612   0.454480   0.351 0.725755
reasonreputation -0.051845   0.317894  -0.163 0.870589
guardianmother    0.267462   0.311371   0.859 0.391226
guardianother    -0.157335   0.554872  -0.284 0.777003
traveltime        0.274301   0.197865   1.386 0.166968
studytime        -0.140650   0.155149  -0.907 0.365577
failures         -0.185333   0.211040  -0.878 0.380739
schoolsupyes      0.562716   0.379303   1.484 0.139268
famsupyes         0.369402   0.268848   1.374 0.170745
paidyes           0.060643   0.270971   0.224 0.823107
activitiesyes    -0.286006   0.247519  -1.155 0.249063
nurseryyes       -0.426858   0.315064  -1.355 0.176774
higheryes         0.503353   0.677346   0.743 0.458148
internetyes      -0.097405   0.331040  -0.294 0.768834
romanticyes      -0.243837   0.268414  -0.908 0.364577
famrel            0.494479   0.136663   3.618 0.000363 ***
freetime         -0.139869   0.138297  -1.011 0.312879
goout             0.078871   0.128794   0.612 0.540879
Dalc             -0.248633   0.178366  -1.394 0.164651
Walc              0.221434   0.139178   1.591 0.112950
health            0.027495   0.095100   0.289 0.772748
absences          0.060830   0.017915   3.396 0.000804 ***
G1                0.162966   0.076956   2.118 0.035255 *
G2                0.994677   0.066450  14.969  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.868 on 235 degrees of freedom
Multiple R-squared:  0.8616, Adjusted R-squared:  0.8374
F-statistic: 35.68 on 41 and 235 DF,  p-value: < 2.2e-16
```

Model Interpretation

Understanding requires a general understanding of statistics, check out Wikipedia for overviews on some of these topics, as well as the ISLR book. Here's a quick guide on understanding the model summary:

#	Name	Description
1	Residuals	The residuals are the difference between the actual values of the variable you're predicting and predicted values from your regression--$y - \hat{y}$. For most regressions you want your residuals to look like a normal distribution when plotted. If our residuals are normally distributed, this indicates the mean of the difference between our predictions and the actual values is close to 0 (good) and that when we miss, we're missing both short and long of the actual value, and the likelihood of a miss being far from the actual value gets smaller as the distance from the actual value gets larger. Think of it like a dartboard. A good model is going to hit the bullseye some of the time (but not everytime). When it doesn't hit the bullseye, it's missing in all of the other buckets evenly (i.e. not just missing in the 16 bin) and it also misses closer to the bullseye as opposed to on the outer edges of the dartboard.
2	Significance Stars	The stars are shorthand for significance levels, with the number of asterisks displayed according to the p-value computed. *** for high significance and * for low significance. In this case, *** indicates that it's unlikely that no

		relationship exists b/w absences and G3 scores.
3	Estimated Coeffecient	The estimated coefficient is the value of slope calculated by the regression. It might seem a little confusing that the Intercept also has a value, but just think of it as a slope that is always multiplied by 1. This number will obviously vary based on the magnitude of the variable you're inputting into the regression, but it's always good to spot check this number to make sure it seems reasonable.
4	Standard Error of the Coefficient Estimate	Measure of the variability in the estimate for the coefficient. Lower means better but this number is relative to the value of the coefficient. As a rule of thumb, you'd like this value to be at least an order of magnitude less than the coefficient estimate.
5	t-value of the Coefficient Estimate	Score that measures whether or not the coefficient for this variable is meaningful for the model. You probably won't use this value itself, but know that it is used to calculate the p-value and the significance levels.
6	Variable p-value	Probability the variable is *NOT* relevant. You want this number to be as small as possible. If the number is *really* small, R will display it in scientific notation.
7	Significance Legend	The more punctuation there is next to your variables, the better. Blank=bad, Dots=pretty good, Stars=good, More Stars=very good
8	Residual Std Error / Degrees of Freedom	The Residual Std Error is just the standard deviation of your residuals. You'd like this number to be proportional to the quantiles of the residuals in #1. For a normal distribution, the 1st and 3rd quantiles should be 1.5 +/- the std error. The Degrees of Freedom is the difference between the number of observations included in your training sample and the number of variables used in your model (intercept counts as a variable).
9	R-squared	Metric for evaluating the goodness of fit of your model. Higher is better with 1 being the best. Corresponds with the amount of variability in what you're predicting that is explained by the model. WARNING: While a high R-squared indicates good correlation, correlation does *not* always imply causation.
10	F-statistic & resulting p-value	Performs an F-test on the model. This takes the parameters of our model (in our case we only have 1) and compares it to a model that has fewer parameters. In theory the model with more parameters should fit better. If the model with more parameters (your model) doesn't perform better than the model with fewer parameters, the F-test will have a high p-value (probability *NOT* significant boost). If the model with more parameters is better than the model with fewer parameters, you will have a lower p-value. The DF, or degrees of freedom, pertains to how many variables are in the model. In our case there is one variable so there is one degree of freedom.

Looks like Absences, G1, and G2 scores are good predictors. With age and activities also possibly contributing to a good model.

Visualize our Model

We can visualize our linear regression model by plotting out the residuals, the residuals are basically a measure of how off we are for each point in the plot versus our model (the error).

In [17]:

```
# Grab residuals
res <- residuals(model)

# Convert to DataFrame for gglpot
res <- as.data.frame(res)

head(res)
```

Out[17]:

	res
1	0.9678451
5	1.182998
6	-1.409605
7	0.1125706
9	0.381467

	res
10	0.3221204

Why Plot Residuals?

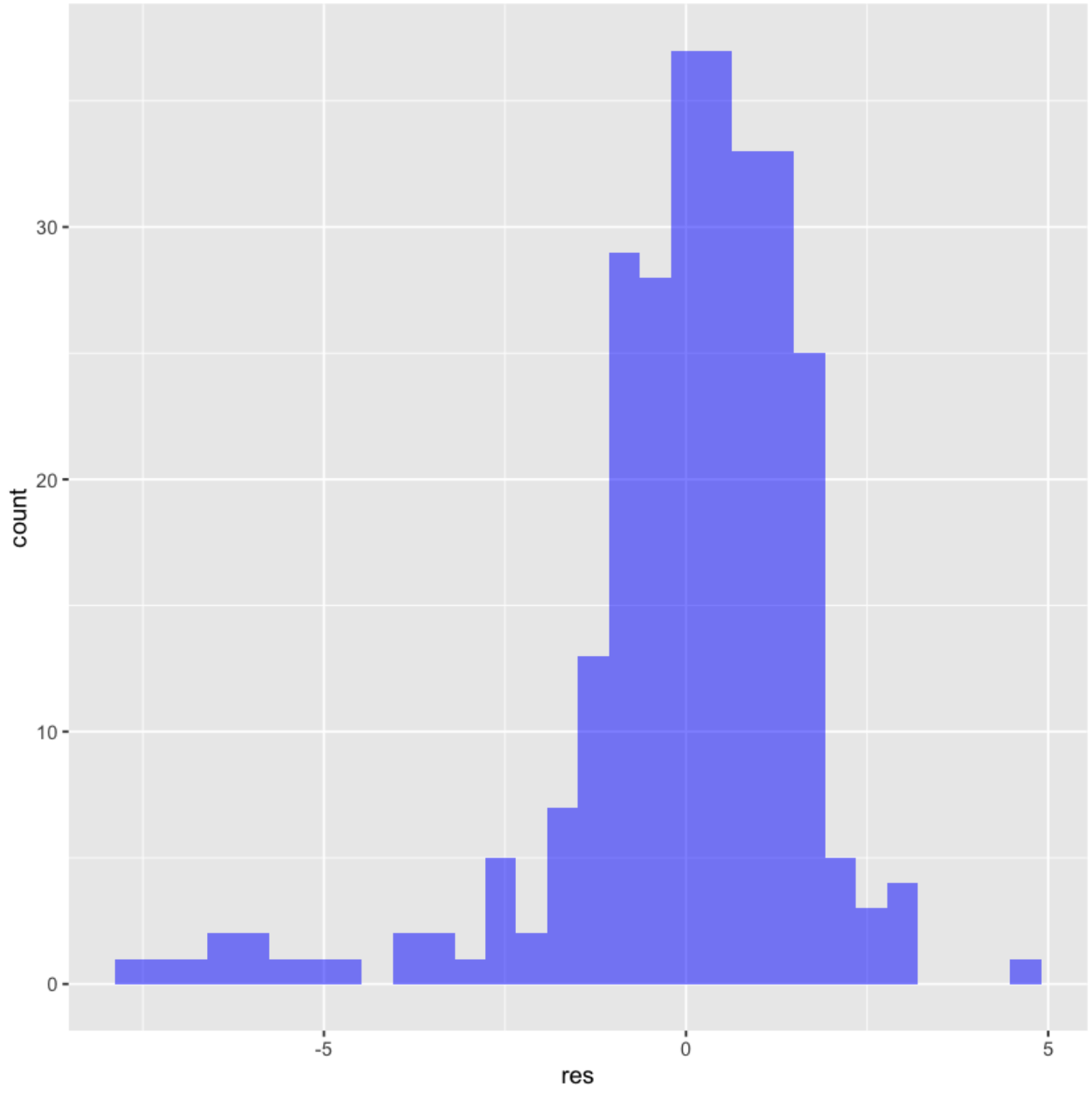
We want a histogram of our residuals to be normally distributed, something with a strong bimodal distribution may be a warning that our data was not a good fit for lienar regression. However, this can also be hidden from out model. A famous example is Anscombe's Quartet

Using ggplot

In [18]:

```
# Histogram of residuals
ggplot(res,aes(res)) +  geom_histogram(fill='blue',alpha=0.5)
```

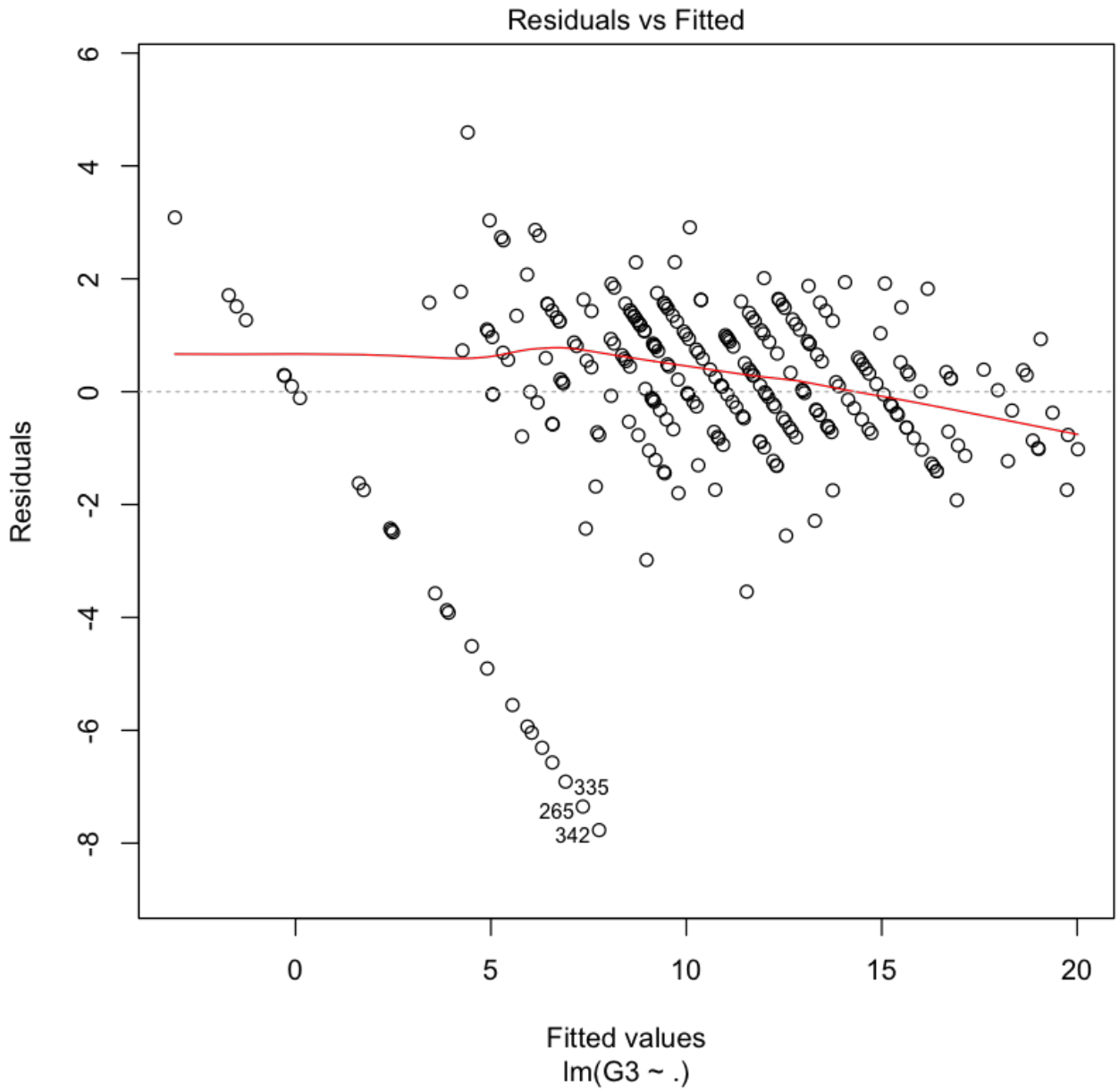
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

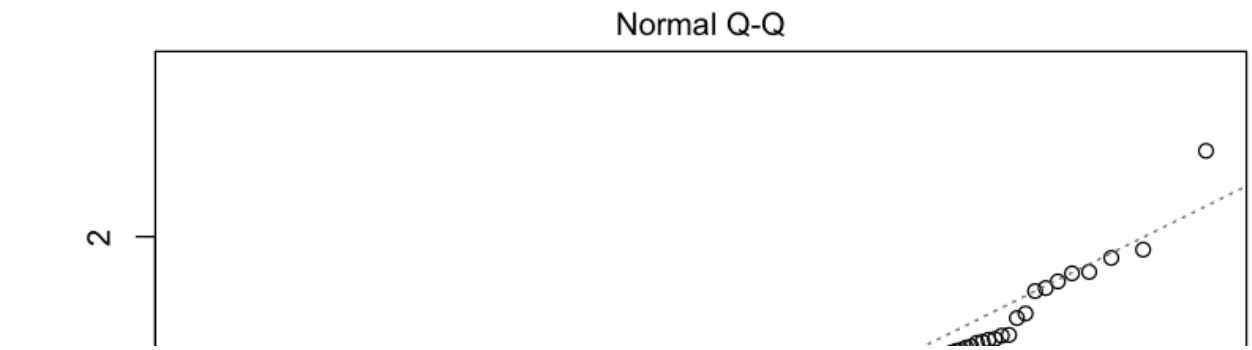


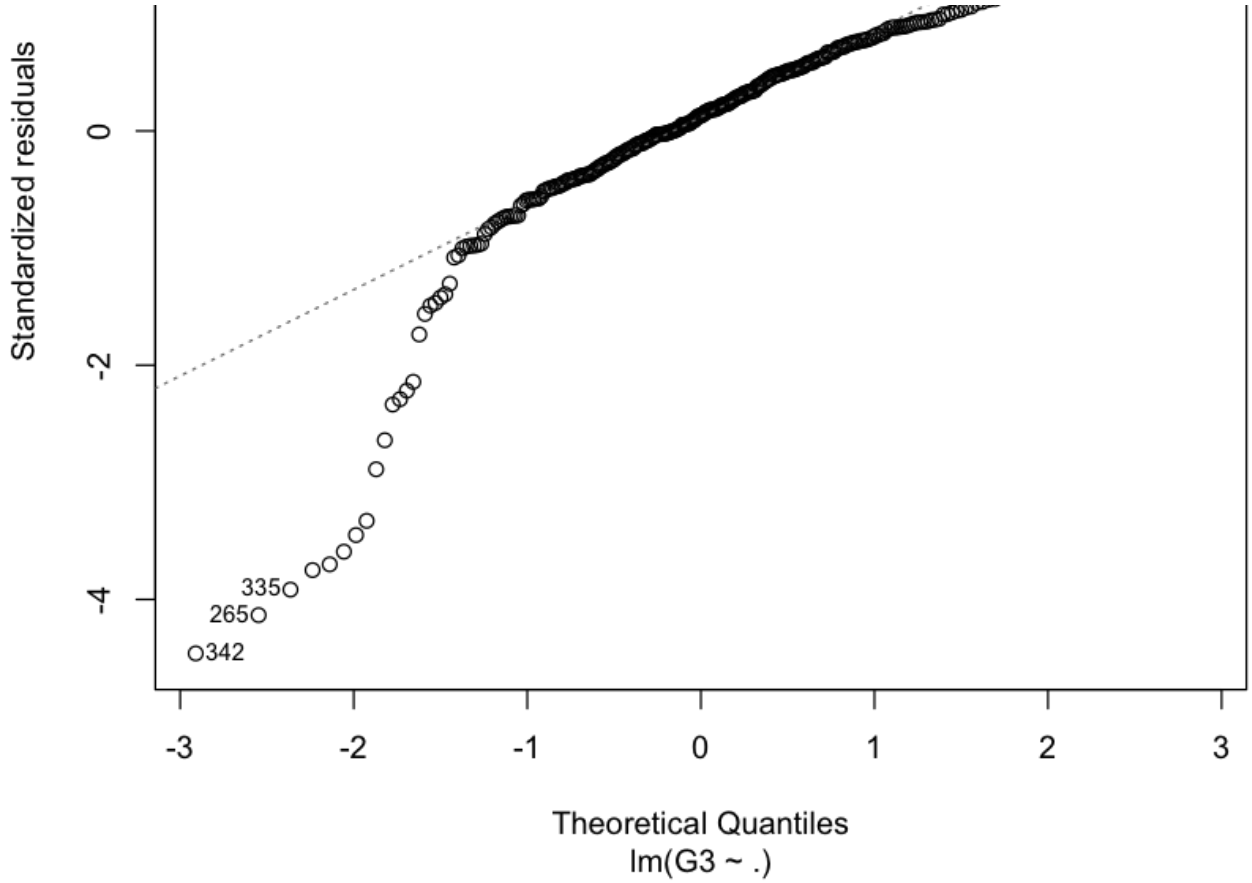
Looks like there are some suspicious residual values that have a value less than -5. We can further explore this by just calling plot on our model. What these plots represent is outside the course of this lecture, but it's covered in ISLR, as well as the Wikipedia page on Regression Validation.

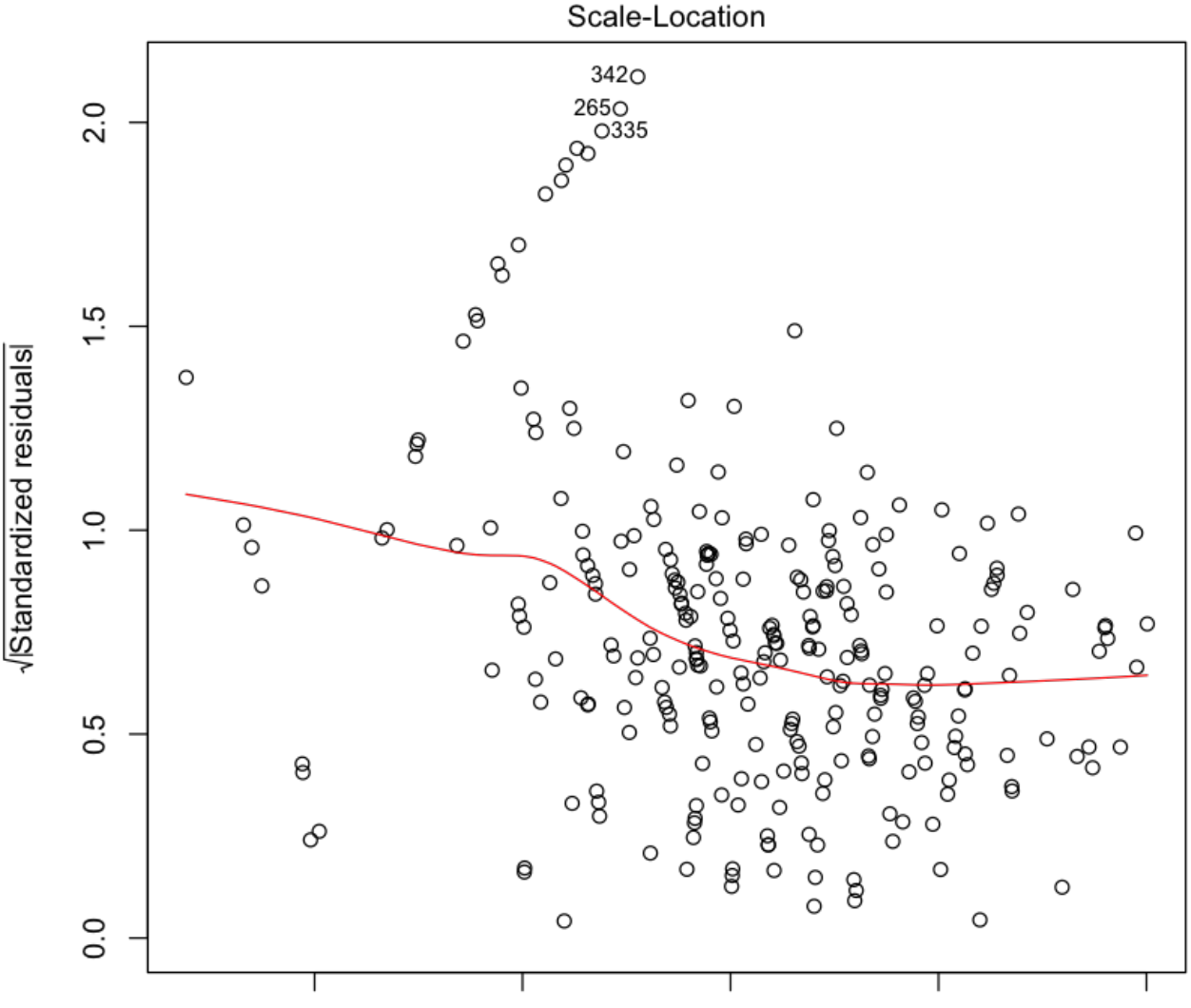
In [19]:

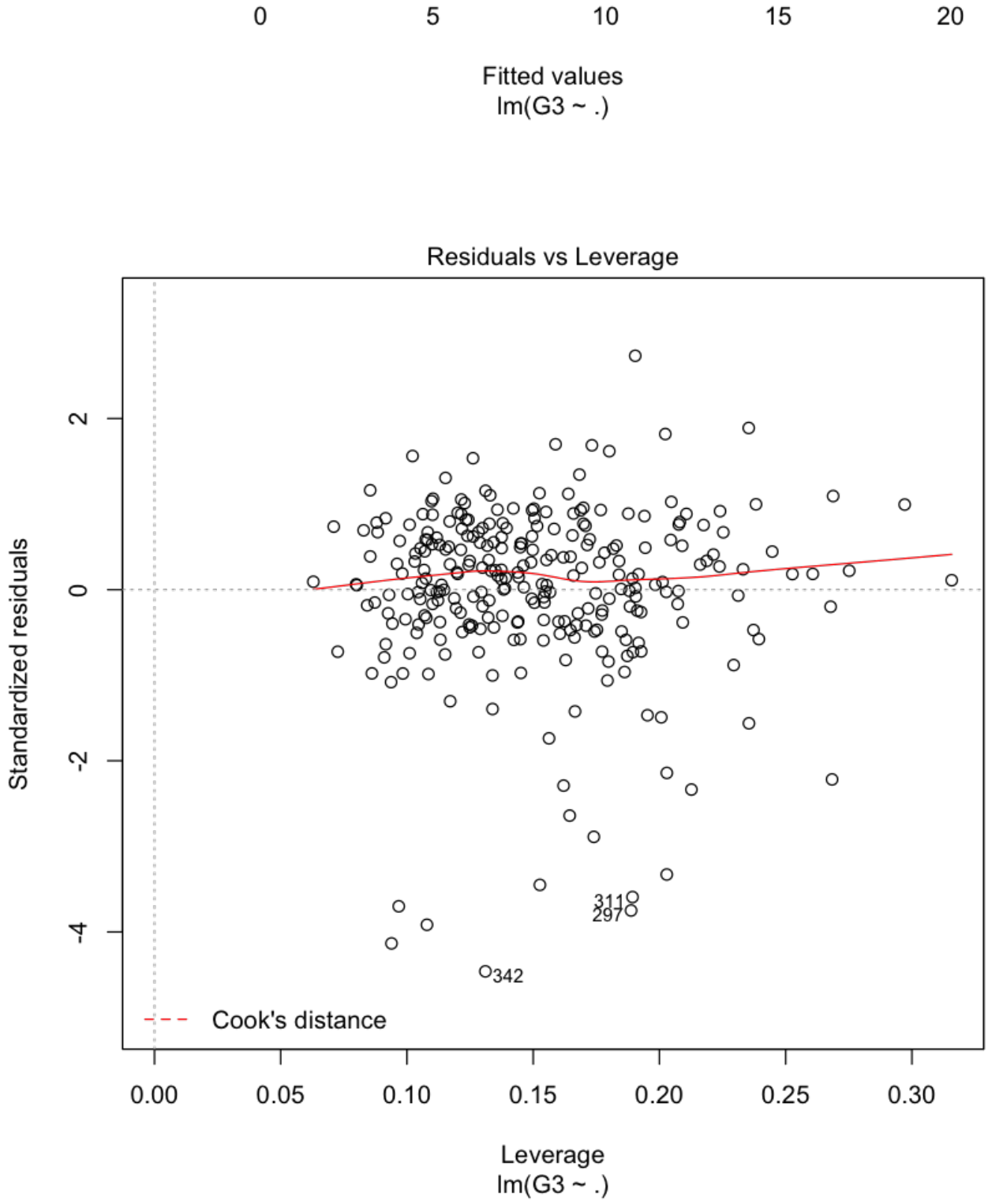
```
plot(model)
```






Standardized residuals
0
-2
-4
335
265
342
-3
-2
-1
0
1
2
3
Theoretical Quantiles
lm(G3 ~ .)


Scale-Location
√|Standardized residuals|
342
265
335
2.0
1.5
1.0
0.5
0.0



Basically after looking at these plots what you will realize is that our model (behaving as a continuous line, predicted students would get **negative** scores on their test! Let's make these all zeros when running our results against our predictions.

Predictions

Let's test our model by predicting on our testing set:

In [20]:

```
G3.predictions <- predict(model,test)
```

Now we can get the root mean squared error, a standardized measure of how off we were with our predicted values:

In [21]:

```
results <- cbind(G3.predictions,test$G3)
colnames(results) <- c('pred','real')
results <- as.data.frame(results)
```

Now let's take care of negative predictions! Lot's of ways to this, here's a more complicated way, but its a good example of creating a custom function for a custom problem:

In [22]:

```
to_zero <- function(x){
    if  (x < 0){
        return(0)
    }else{
        return(x)
    }
}
```

In [23]:

```
results$pred <- sapply(results$pred,to_zero)
```

There's lots of ways to evaluate the prediction values, for example the MSE (mean squared error):

In [24]:

```
mse <- mean((results$real-results$pred)^2)
print(mse)
```

```
[1] 4.411405
```

Or the root mean squared error:

In [25]:

```
mse^0.5
```

Out[25]:

2.10033451255583

Or just the R-Squared Value for our model (just for the predictions)

In [26]:

```
SSE = sum((results$pred - results$real)^2)
SST = sum( (mean(df$G3) - results$real)^2)

R2 = 1 - SSE/SST
R2
```

Out[26]:

0.777902260014962

Conclusion

You should now feel comfortable with the R syntax for a Linear Regression. If some of the plots or math did not make sense to you, make sure to review ISLR and the relevant Wikipedia pages. There is no real substitute for taking the time to read about this material

Up next is an exercise to test your knowledge!

Logistic Regression Lecture

The Data

We can begin by loading in our training data into data frames:

In [181]:

```
df.train <- read.csv('titanic_train.csv')
```

In [182]:

```
head(df.train)
```

Out[182]:

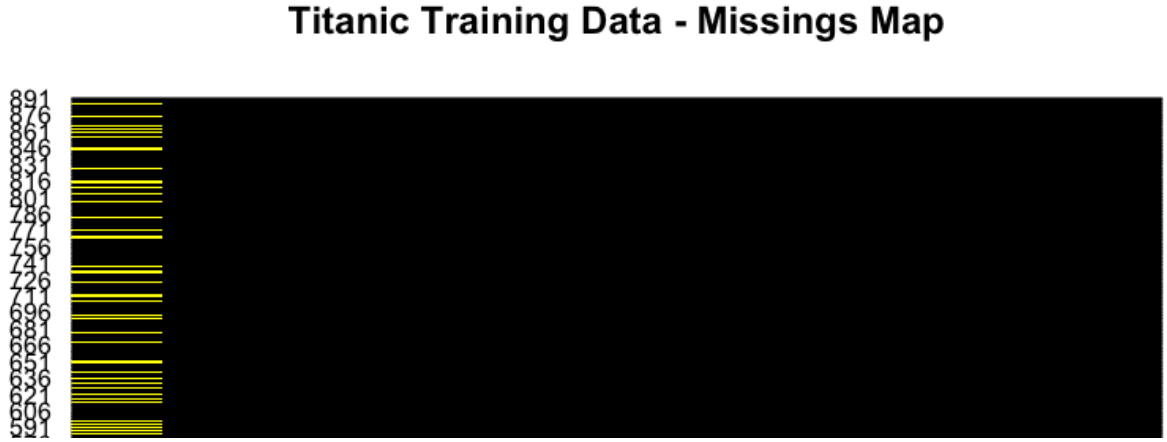
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25		S
2	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	71.2833	C85	C
3	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.925		S
4	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
5	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05		S
6	6	0	3	Moran, Mr. James	male	NA	0	0	330877	8.4583		Q

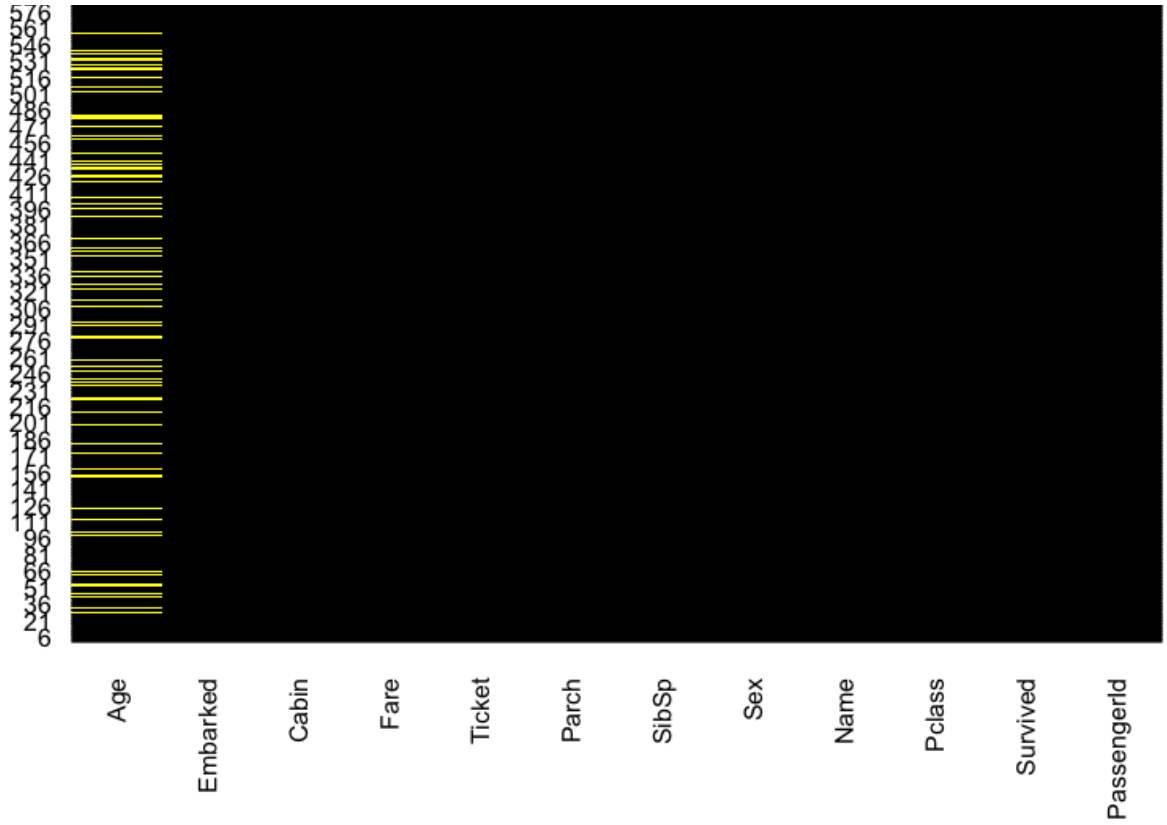
Exploratory Data Analysis (EDA)

Let's explore how much missing data we have, we can use the Amelia pacakge for this. Install it if you want to follow along, you'll need to install it later for you logistic regression project.

In [183]:

```
library(Amelia)
missmap(df.train, main="Titanic Training Data - Missings Map",
        col=c("yellow", "black"), legend=FALSE)
```





Roughly 20 percent of the Age data is missing. The proportion of Age "missings" is likely small enough for reasonable replacement with some form of imputation.

Let's continue on by visualizing some of the data.

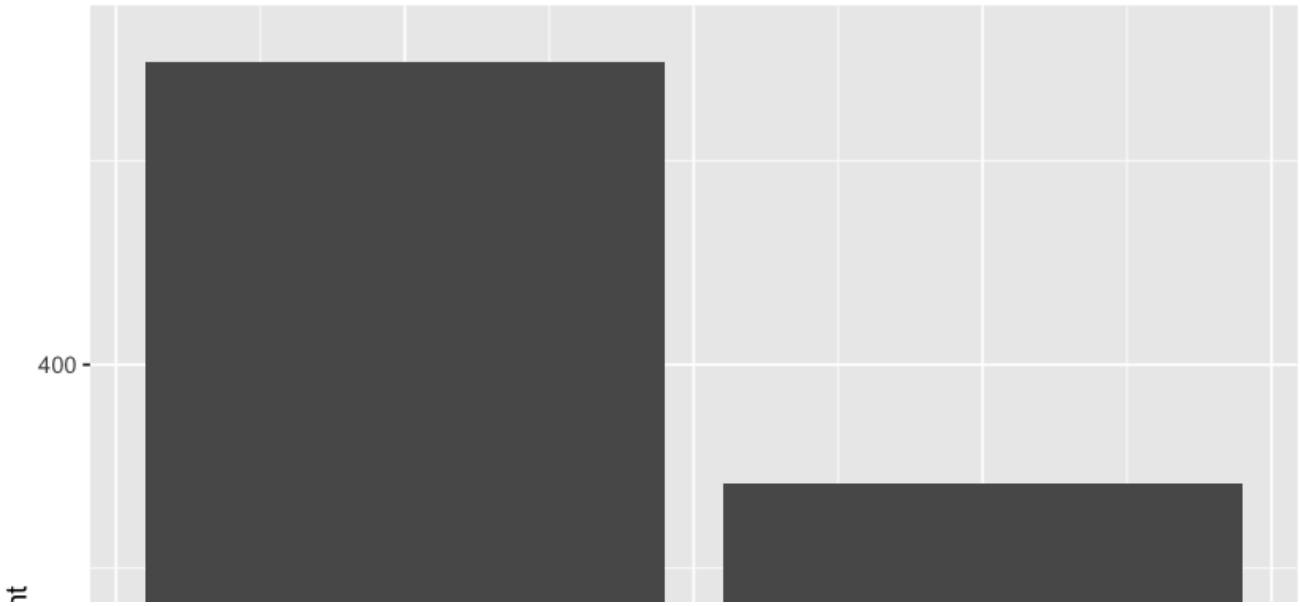
Data Visualization with ggplot2

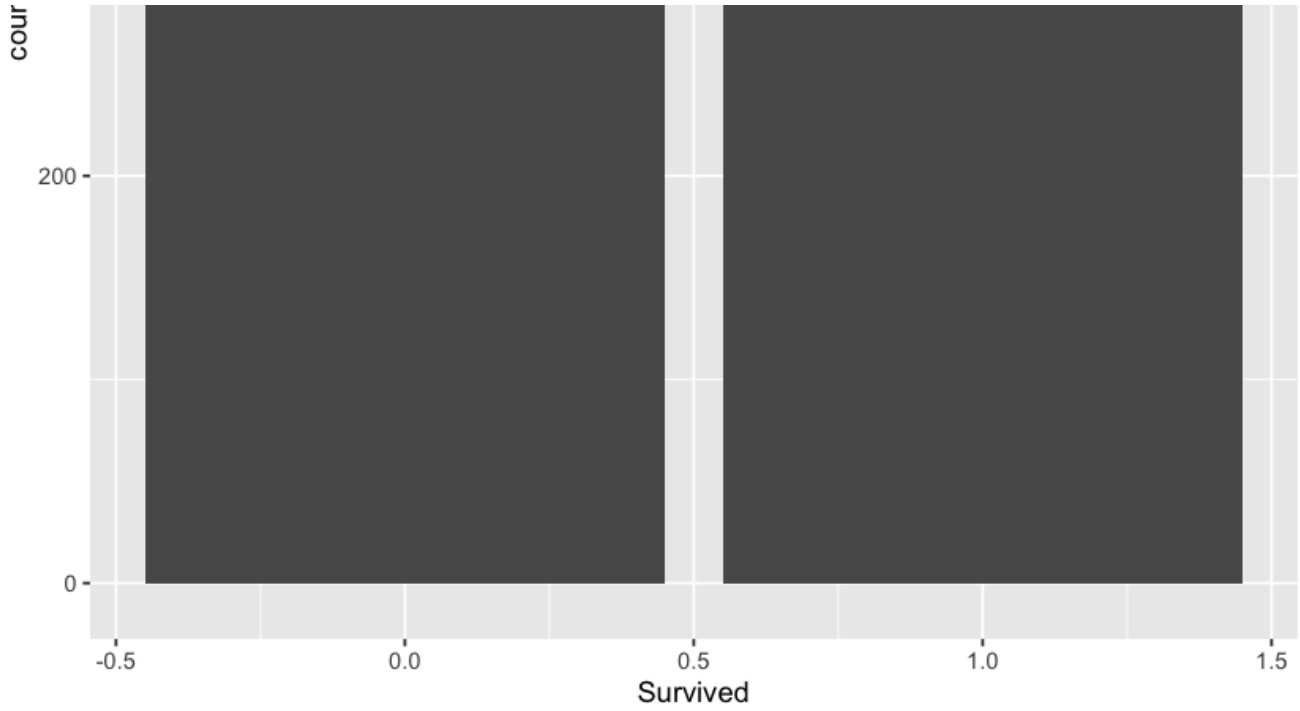
In [184]:

```
library(ggplot2)
```

In [185]:

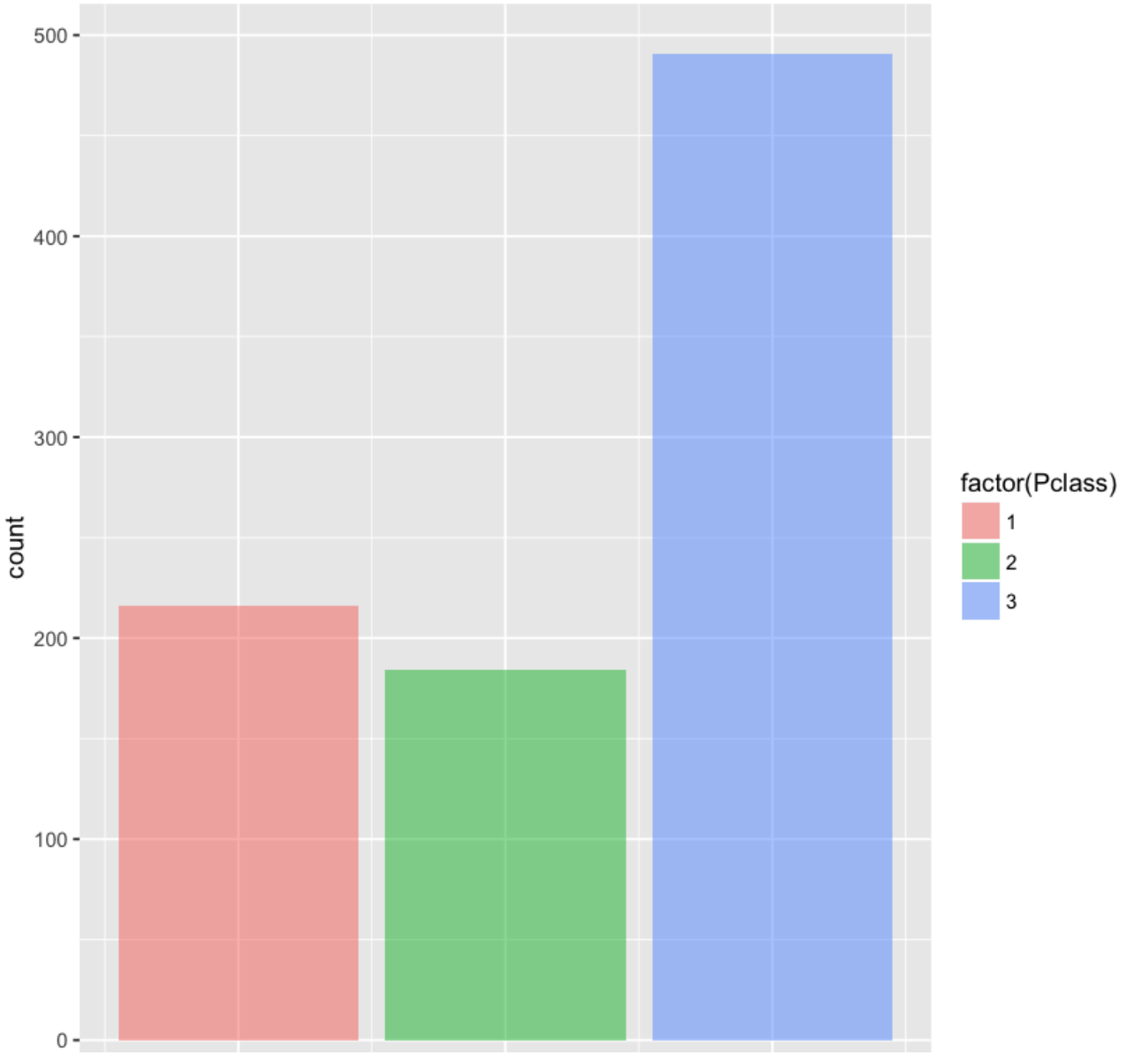
```
ggplot(df.train,aes(Survived)) + geom_bar()
```

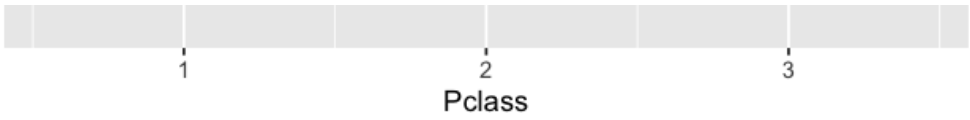




In [186]:

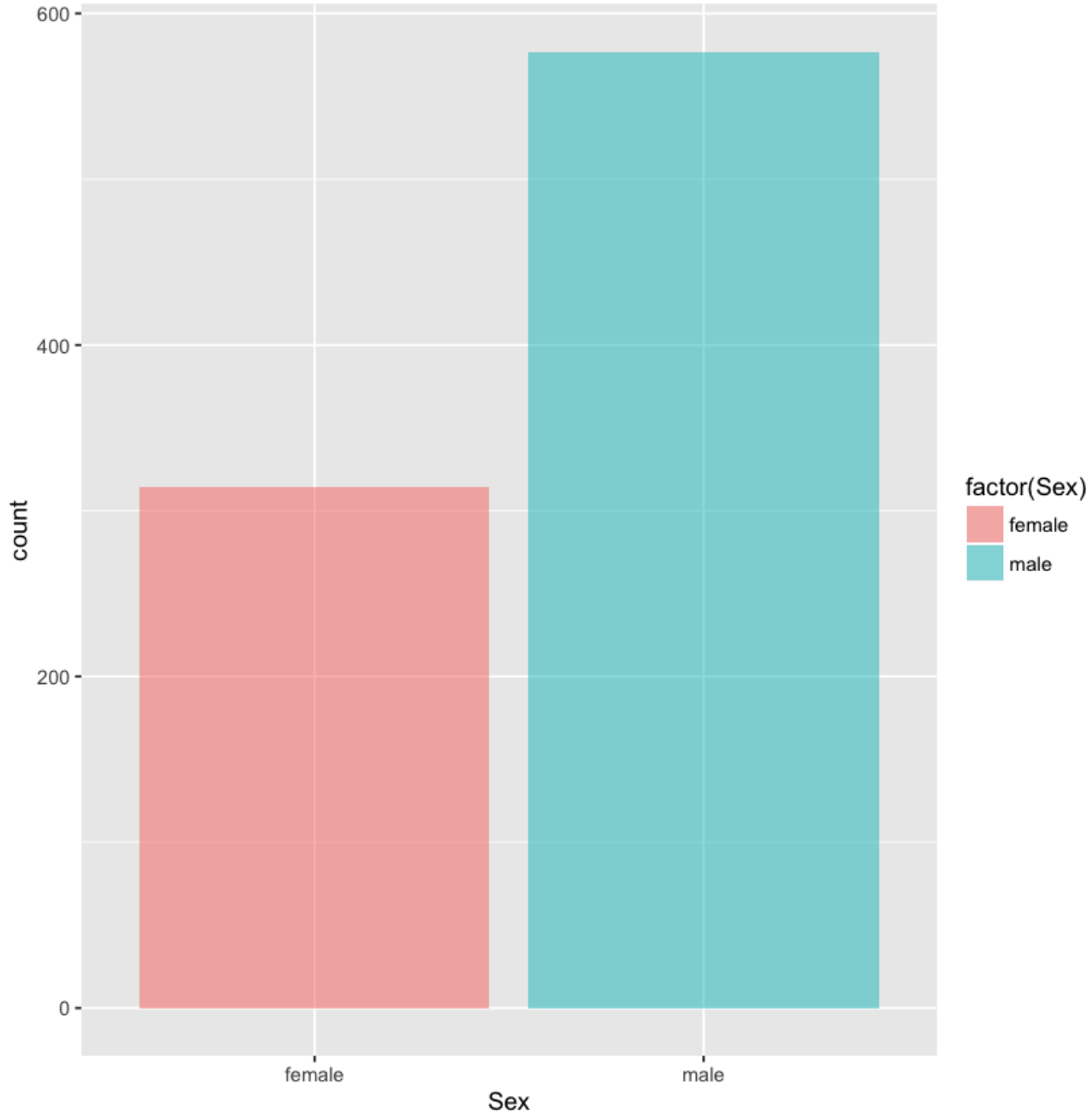
```
ggplot(df.train,aes(Pclass)) + geom_bar(aes(fill=factor(Pclass)),alpha=0.5)
```





In [187]:

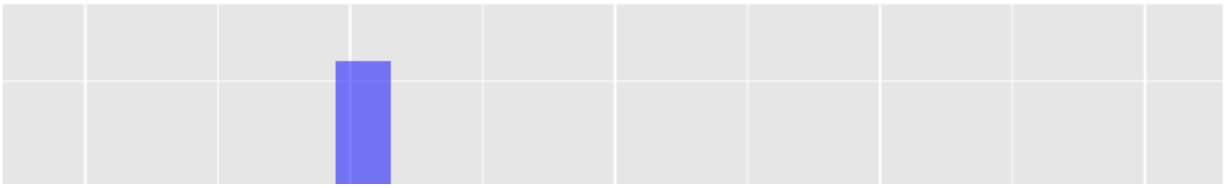
```
ggplot(df.train,aes(Sex)) + geom_bar(aes(fill=factor(Sex)),alpha=0.5)
```

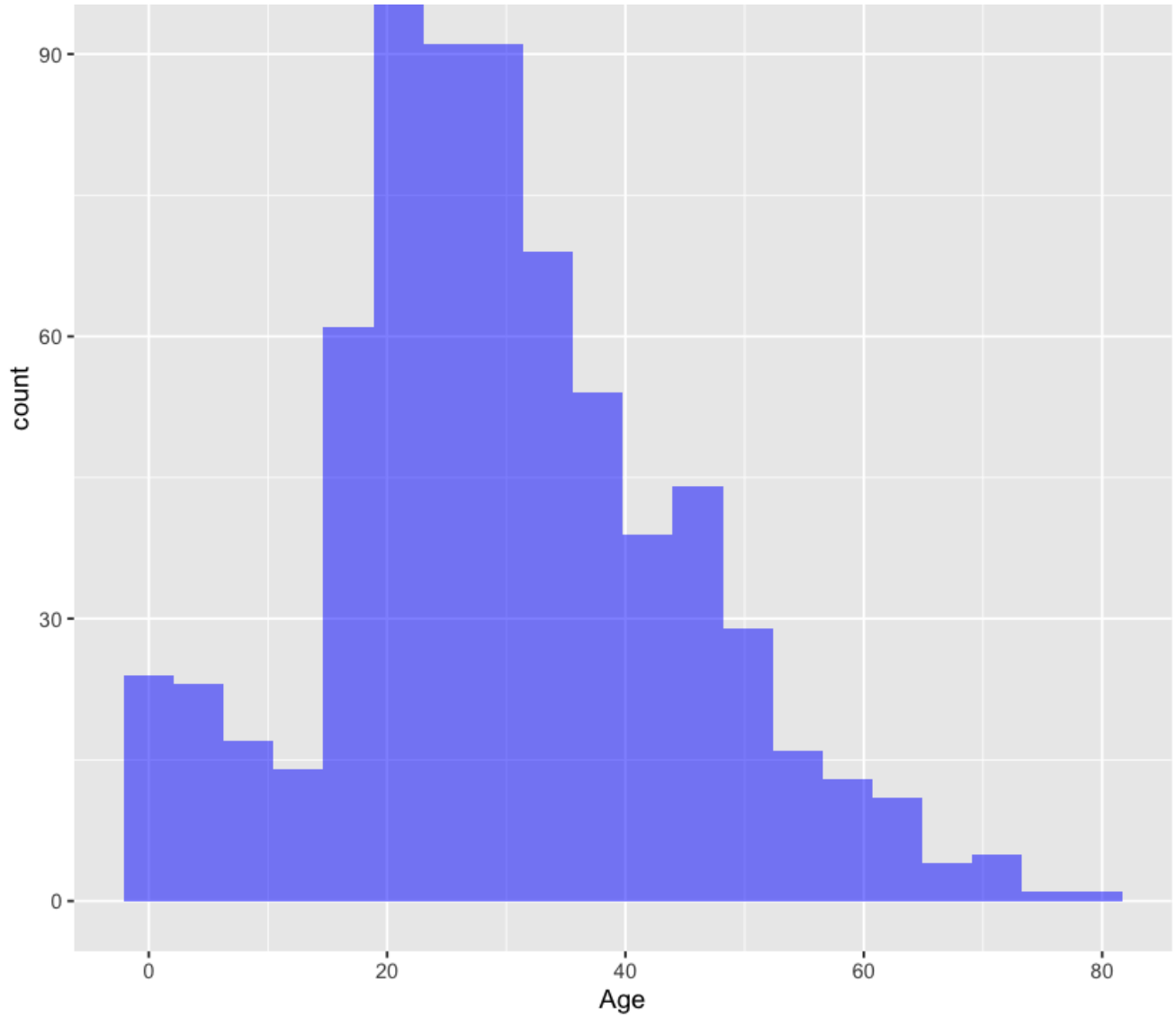


In [188]:

```
ggplot(df.train,aes(Age)) + geom_histogram(fill='blue',bins=20,alpha=0.5)
```

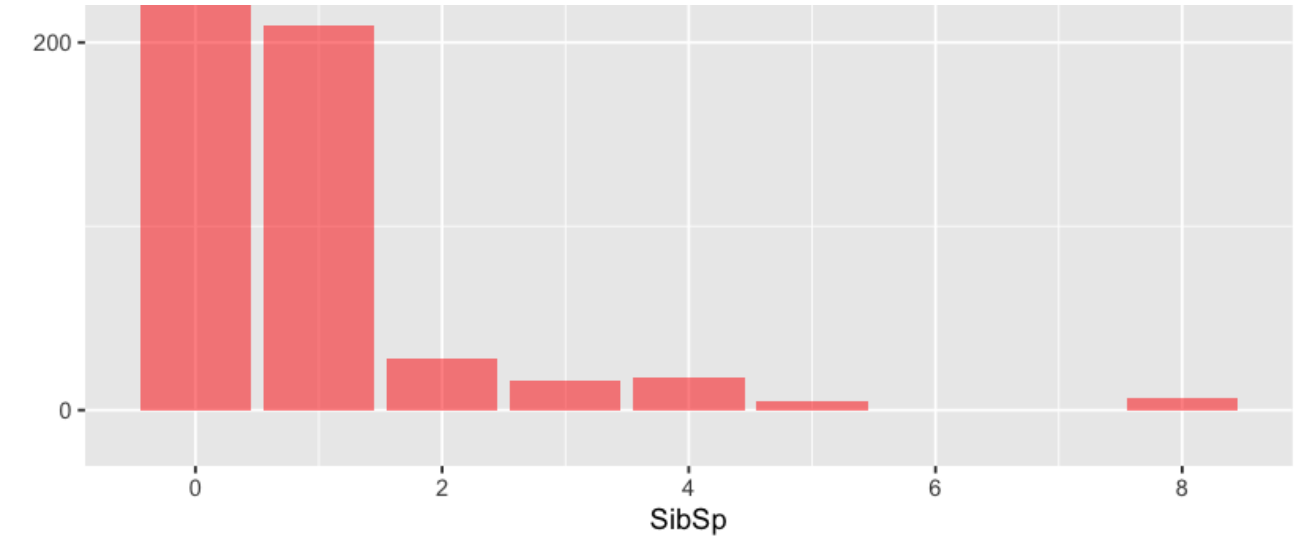
```
Warning message:
: Removed 177 rows containing non-finite values (stat_bin).
```





In [189]:

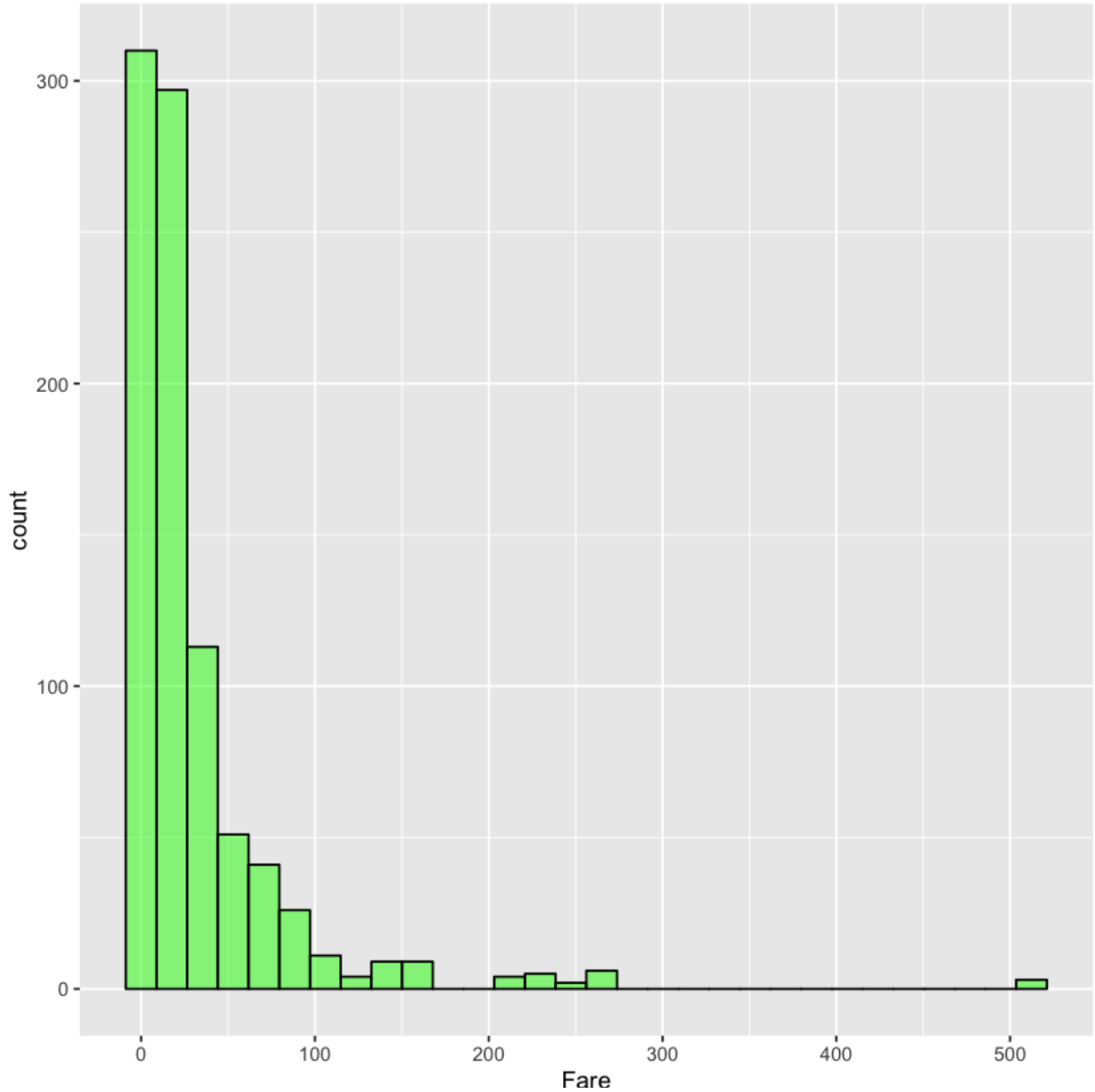
```
ggplot(df.train,aes(SibSp)) + geom_bar(fill='red',alpha=0.5)
```



In [190]:

```
ggplot(df.train,aes(Fare)) + geom_histogram(fill='green',color='black',alpha=0.5)
```

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



Data Cleaning

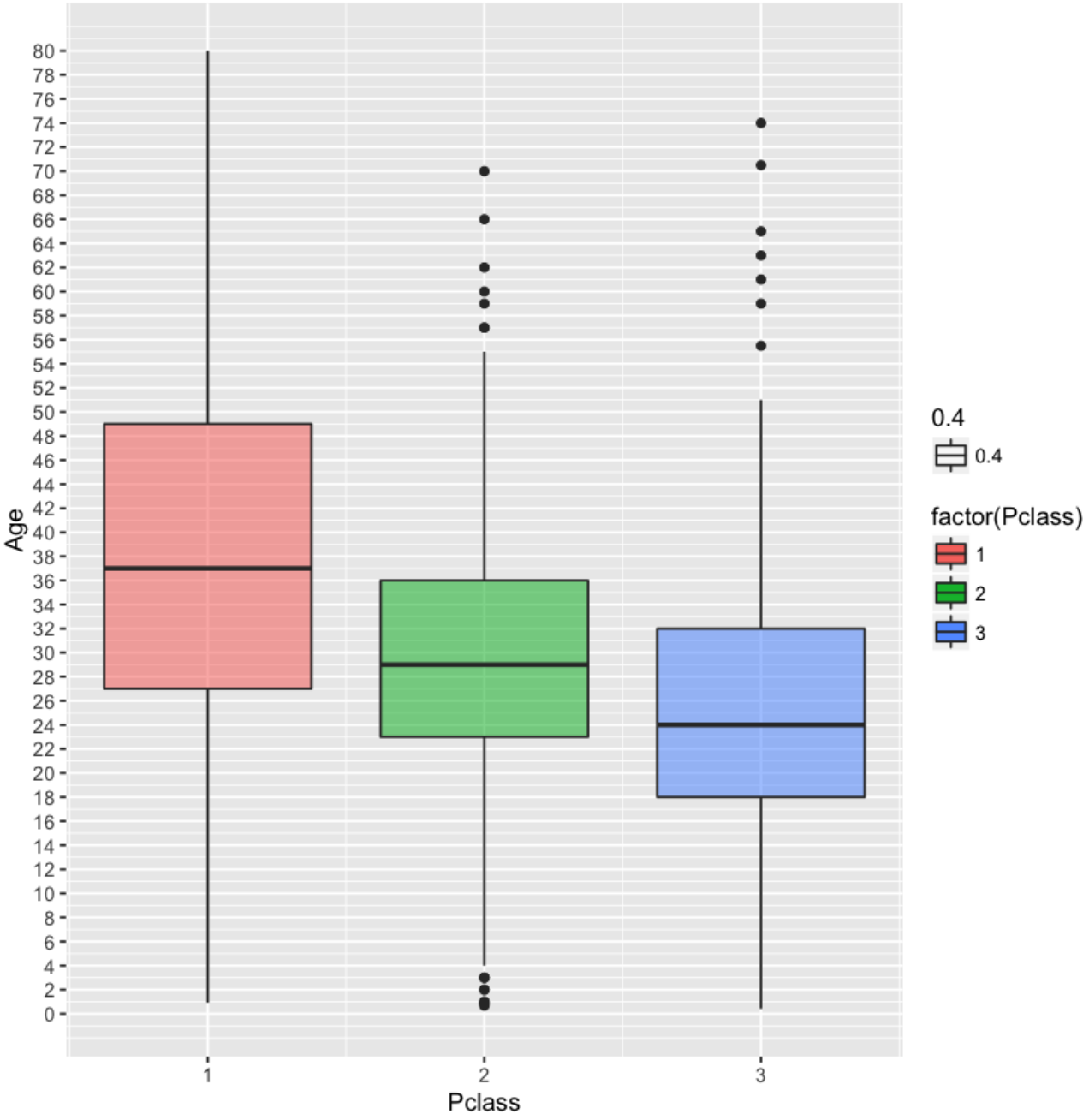
We want to fill in missing age data instead of just dropping the missing age data rows. One way to do this is by filling in the mean age of all the passengers (imputation).

However we can be smarter about this and check the average age by passenger class. For example:

In [191]:

```
pl <- ggplot(df.train,aes(Pclass,Age)) + geom_boxplot(aes(group=Pclass,fill=factor(Pclass),alpha=0.
4))
pl + scale_y_continuous(breaks = seq(min(0), max(80), by = 2))
```

```
Warning message:
: Removed 177 rows containing non-finite values (stat_boxplot).
```



We can see the wealthier passengers in the higher classes tend to be older, which makes sense. We'll use these average age values to impute based on Pclass for Age.

In [192]:

```
impute_age <- function(age,class){
    out <- age
    for (i in 1:length(age)){

        if (is.na(age[i])){

            if (class[i] == 1){
                out[i] <- 37

            }else if (class[i] == 2){
                out[i] <- 29

            }else{
                out[i] <- 24
            }
        }else{
            out[i]<-age[i]
        }
    }
    return(out)
}
```

In [193]:

```
fixed.ages <- impute_age(df.train$Age,df.train$Pclass)
```

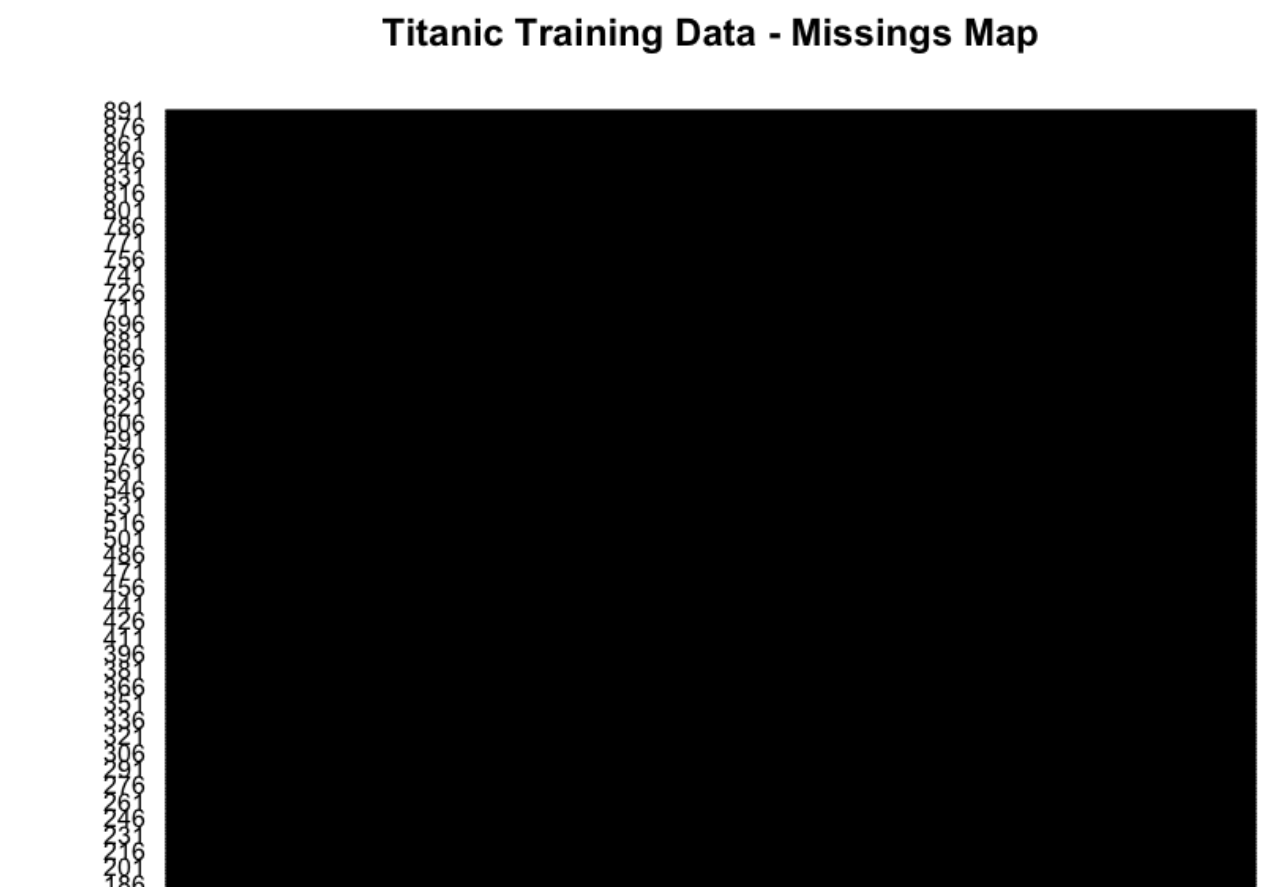
In [194]:

```
df.train$Age <- fixed.ages
```

Now let's check to see if it worked:

In [195]:

```
missmap(df.train, main="Titanic Training Data - Missings Map",
        col=c("yellow", "black"), legend=FALSE)
```



Great let's continue with building our model!

Building a Logistic Regression Model

Now it is time to build our model! Let's begin by doing a final "clean-up" of our data by removing the features we won't be using and making sure that the features are of the correct data type.

In [197]:

```
str(df.train)
```

```
'data.frame': 891 obs. of  12 variables:
 $ PassengerId: int  1 2 3 4 5 6 7 8 9 10 ...
 $ Survived   : int  0 1 1 1 0 0 0 0 1 1 ...
 $ Pclass     : int  3 1 3 1 3 3 1 3 3 2 ...
 $ Name       : Factor w/ 891 levels "Abbing, Mr. Anthony",..: 109 191 358 277 16 559 520 629 417
581 ...
 $ Sex        : Factor w/ 2 levels "female","male": 2 1 1 1 2 2 2 2 1 1 ...
 $ Age        : num  22 38 26 35 35 24 54 2 27 14 ...
 $ SibSp      : int  1 1 0 1 0 0 0 3 0 1 ...
 $ Parch      : int  0 0 0 0 0 0 0 1 2 0 ...
 $ Ticket     : Factor w/ 681 levels "110152","110413",..: 524 597 670 50 473 276 86 396 345 133 .
..
 $ Fare       : num  7.25 71.28 7.92 53.1 8.05 ...
 $ Cabin      : Factor w/ 148 levels "","A10","A14",..: 1 83 1 57 1 1 131 1 1 1 ...
 $ Embarked   : Factor w/ 4 levels "","C","Q","S": 4 2 4 4 4 3 4 4 4 2 ...
```

Let's remove what we won't use:

In [198]:

```
head(df.train,3)
```

Out[198]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25		S
2	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	71.2833	C85	C
3	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.925		S

Let's select the relevant columns for training:

In [173]:

```
library(dplyr)
```

In [199]:

```
df.train <- select(df.train,-PassengerId,-Name,-Ticket,-Cabin)
```

In [201]:

```
head(df.train,3)
```

Out[201]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
1	0	3	male	22	1	0	7.25	S
2	1	1	female	38	1	0	71.2833	C
3	1	3	female	26	0	0	7.925	S

Now let's set factor columns.

In [202]:

```
str(df.train)
```

```
'data.frame': 891 obs. of  8 variables:
 $ Survived: int  0 1 1 1 0 0 0 0 1 1 ...
 $ Pclass  : int  3 1 3 1 3 3 1 3 3 2 ...
 $ Sex     : Factor w/ 2 levels "female","male": 2 1 1 1 2 2 2 2 1 1 ...
 $ Age     : num  22 38 26 35 35 24 54 2 27 14 ...
 $ SibSp   : int  1 1 0 1 0 0 0 3 0 1 ...
 $ Parch   : int  0 0 0 0 0 0 0 1 2 0 ...
 $ Fare    : num  7.25 71.28 7.92 53.1 8.05 ...
 $ Embarked: Factor w/ 4 levels "","C","Q","S": 4 2 4 4 4 3 4 4 4 2 ...
```

In [203]:

```
df.train$Survived <- factor(df.train$Survived)
df.train$Pclass <- factor(df.train$Pclass)
df.train$Parch <- factor(df.train$Parch)
df.train$SibSp <- factor(df.train$SibSp)
```

Train the Model

Now let's train the model!

In [205]:

```
log.model <- glm(formula=Survived ~ . , family = binomial(link='logit'),data = df.train)
```

In [207]:

```
summary(log.model)
```

Out[207]:

```
Call:
glm(formula = Survived ~ ., family = binomial(link = "logit"),
    data = df.train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.8158  -0.6134  -0.4138   0.5808   2.4896

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.845e+01  1.660e+03   0.011 0.991134
Pclass2     -1.079e+00  3.092e-01  -3.490 0.000484 ***
Pclass3     -2.191e+00  3.161e-01  -6.930 4.20e-12 ***
```

```
Sexmale     -2.677e+00  2.040e-01 -13.123  < 2e-16 ***
Age         -3.971e-02  8.758e-03  -4.534 5.79e-06 ***
SibSp1       8.135e-02  2.245e-01   0.362 0.717133
SibSp2      -2.897e-01  5.368e-01  -0.540 0.589361
SibSp3      -2.241e+00  7.202e-01  -3.111 0.001862 **
SibSp4      -1.675e+00  7.620e-01  -2.198 0.027954 *
SibSp5      -1.595e+01  9.588e+02  -0.017 0.986731
SibSp8      -1.607e+01  7.578e+02  -0.021 0.983077
Parch1       3.741e-01  2.895e-01   1.292 0.196213
Parch2       3.862e-02  3.824e-01   0.101 0.919560
Parch3       3.655e-01  1.056e+00   0.346 0.729318
Parch4      -1.586e+01  1.055e+03  -0.015 0.988007
Parch5      -1.152e+00  1.172e+00  -0.983 0.325771
Parch6      -1.643e+01  2.400e+03  -0.007 0.994536
Fare         2.109e-03  2.490e-03   0.847 0.397036
EmbarkedC   -1.458e+01  1.660e+03  -0.009 0.992995
EmbarkedQ   -1.456e+01  1.660e+03  -0.009 0.993001
EmbarkedS   -1.486e+01  1.660e+03  -0.009 0.992857
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1186.66  on 890  degrees of freedom
Residual deviance:  763.41  on 870  degrees of freedom
AIC: 805.41

Number of Fisher Scoring iterations: 15
```

We can see clearly that Sex,Age, and Class are the most significant features. Which makes sense given the women and children first policy.

Predicting using Test Cases

Let's make a test set out of our training set, retrain on the smaller version of our training set and check it against the test subset.

In [212]:

```
library(caTools)
set.seed(101)

split = sample.split(df.train$Survived, SplitRatio = 0.70)

final.train = subset(df.train, split == TRUE)
final.test = subset(df.train, split == FALSE)
```

Now let's rerun our model on only our final training set:

In [213]:

```
final.log.model <- glm(formula=Survived ~ . , family = binomial(link='logit'),data = final.train)
```

In [215]:

```
summary(final.log.model)
```

Out[215]:

```
Call:
glm(formula = Survived ~ ., family = binomial(link = "logit"),
    data = final.train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.8288  -0.5607  -0.4096   0.6174   2.4898

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.777e+01  2.400e+03   0.007 0.994091
Pclass2     -1.230e+00  3.814e-01  -3.225 0.001261 **
```

```
Pclass3     -2.160e+00  3.841e-01  -5.624 1.87e-08 ***
Sexmale     -2.660e+00  2.467e-01 -10.782  < 2e-16 ***
Age         -3.831e-02  1.034e-02  -3.705 0.000212 ***
SibSp1      -2.114e-02  2.755e-01  -0.077 0.938836
SibSp2      -4.000e-01  6.463e-01  -0.619 0.536028
SibSp3      -2.324e+00  8.994e-01  -2.584 0.009765 **
SibSp4      -1.196e+00  8.302e-01  -1.440 0.149839
SibSp5      -1.603e+01  9.592e+02  -0.017 0.986666
SibSp8      -1.633e+01  1.004e+03  -0.016 0.987019
Parch1       7.290e-01  3.545e-01   2.056 0.039771 *
Parch2       1.406e-01  4.504e-01   0.312 0.754892
Parch3       7.919e-01  1.229e+00   0.645 0.519226
Parch4      -1.498e+01  1.552e+03  -0.010 0.992300
Parch5      -9.772e-03  1.378e+00  -0.007 0.994343
Parch6      -1.635e+01  2.400e+03  -0.007 0.994563
Fare         3.128e-03  3.091e-03   1.012 0.311605
EmbarkedC   -1.398e+01  2.400e+03  -0.006 0.995353
EmbarkedQ   -1.387e+01  2.400e+03  -0.006 0.995386
EmbarkedS   -1.431e+01  2.400e+03  -0.006 0.995243
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 829.60  on 622  degrees of freedom
Residual deviance: 530.63  on 602  degrees of freedom
AIC: 572.63

Number of Fisher Scoring iterations: 15
```

Now let's check our prediction accuracy!

In [222]:

```
fitted.probabilities <- predict(final.log.model,newdata=final.test,type='response')
```

Now let's calculate from the predicted values:

In [223]:

```
fitted.results <- ifelse(fitted.probabilities > 0.5,1,0)
```

In [224]:

```
misClasificError <- mean(fitted.results != final.test$Survived)
print(paste('Accuracy',1-misClasificError))
```

```
[1] "Accuracy 0.798507462686567"
```

Looks like we were able to achieve around 80% accuracy, where as random guessing would have just been 50% accuracy. Let's see the confusion matrix:

In [225]:

```
table(final.test$Survived, fitted.probabilities > 0.5)
```

Out[225]:

```
   FALSE TRUE
0    140   25
1     29   74
```

Great Job!

Support Vector Machines

Get the Data

We'll use the iris data again since we are already familiar with it.

In [1]:

```
library(ISLR)
```

In [2]:

```
head(iris)
```

Out[2]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Building the Model

We'll need the **e1071** library. More info here.

In [5]:

```
#install.packages('e1071',repos = 'http://cran.us.r-project.org')
```

In [14]:

```
library(e1071)
```

We can use the svm() function to build and train a support vector machine model:

In [15]:

```
help(svm)
```

Out[15]:

svm {e1071}	R Documentation

Support Vector Machines

Description

`svm` is used to train a support vector machine. It can be used to carry out general regression and classification (of nu and epsilon-type), as well as density-estimation. A formula interface is provided.

Usage

```
## S3 method for class 'formula'
```

```
svm(formula, data = NULL, ..., subset, na.action =
na.omit, scale = TRUE)
## Default S3 method:
svm(x, y = NULL, scale = TRUE, type = NULL, kernel =
"radial", degree = 3, gamma = if (is.vector(x)) 1 else 1 / ncol(x),
coef0 = 0, cost = 1, nu = 0.5,
class.weights = NULL, cachesize = 40, tolerance = 0.001, epsilon = 0.1,
shrinking = TRUE, cross = 0, probability = FALSE, fitted = TRUE,
..., subset, na.action = na.omit)
```

Arguments

`formula`	a symbolic description of the model to be fit.
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'svm' is called from.
`x`	a data matrix, a vector, or a sparse matrix (object of class `Matrix` provided by the Matrix package, or of class `matrix.csr` provided by the SparseM package, or of class `simple_triplet_matrix` provided by the slam package).
`y`	a response vector with one label for each row/component of `x`. Can be either a factor (for classification tasks) or a numeric vector (for regression).
`scale`	A logical vector indicating the variables to be scaled. If `scale` is of length 1, the value is recycled as many times as needed. Per default, data are scaled internally (both `x` and `y` variables) to zero mean and unit variance. The center and scale values are returned and used for later predictions.
`type`	`svm` can be used as a classification machine, as a regression machine, or for novelty detection. Depending of whether `y` is a factor or not, the default setting for `type` is `C-classification` or `eps-regression`, respectively, but may be overwritten by setting an explicit value. Valid options are: • `C-classification` • `nu-classification` • `one-classification` (for novelty detection) • `eps-regression` • `nu-regression`
`kernel`	the kernel used in training and predicting. You might consider changing some of the following parameters, depending on the kernel type. **linear:** *u'*v* **polynomial:** *(gamma*u'*v + coef0)^degree* **radial basis:** *exp(-gamma*|u-v|^2)* **sigmoid:** *tanh(gamma*u'*v + coef0)*
`degree`	parameter needed for kernel of type `polynomial` (default: 3)
`gamma`	parameter needed for all kernels except `linear` (default: 1/(data dimension))
`coef0`	parameter needed for kernels of type `polynomial` and `sigmoid` (default: 0)
`cost`	cost of constraints violation (default: 1)—it is the 'C'-constant of the regularization term in the Lagrange formulation.
`nu`	parameter needed for `nu-classification`, `nu-regression`, and `one-classification`
`class.weights`	a named vector of weights for the different classes, used for asymmetric class sizes. Not all factor levels have to be supplied (default weight: 1). All components have to be named.
`cachesize`	cache memory in MB (default 40)
`tolerance`	tolerance of termination criterion (default: 0.001)
`epsilon`	epsilon in the insensitive-loss function (default: 0.1)
`shrinking`	option whether to use the shrinking-heuristics (default: `TRUE`)

`cross`	if a integer value k>0 is specified, a k-fold cross validation on the training data is performed to assess the quality of the model: the accuracy rate for classification and the Mean Squared Error for regression
`fitted`	logical indicating whether the fitted values should be computed and included in the model or not (default: `TRUE`)
`probability`	logical indicating whether the model should allow for probability predictions.
`...`	additional parameters for the low level fitting function `svm.default`
`subset`	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)

Details

For multiclass-classification with k levels, k>2, `libsvm` uses the 'one-against-one'-approach, in which k(k-1)/2 binary classifiers are trained; the appropriate class is found by a voting scheme.

`libsvm` internally uses a sparse data representation, which is also high-level supported by the package SparseM.

If the predictor variables include factors, the formula interface must be used to get a correct model matrix.

`plot.svm` allows a simple graphical visualization of classification models.

The probability model for classification fits a logistic distribution using maximum likelihood to the decision values of all binary classifiers, and computes the a-posteriori class probabilities for the multi-class problem using quadratic optimization. The probabilistic regression model assumes (zero-mean) laplace-distributed errors for the predictions, and estimates the scale parameter using maximum likelihood.

Value

An object of class `"svm"` containing the fitted model, including:

`SV`	The resulting support vectors (possibly scaled).
`index`	The index of the resulting support vectors in the data matrix. Note that this index refers to the preprocessed data (after the possible effect of `na.omit` and `subset`)
`coefs`	The corresponding coefficients times the training labels.
`rho`	The negative intercept.
`sigma`	In case of a probabilistic regression model, the scale parameter of the hypothesized (zero-mean) laplace distribution estimated by maximum likelihood.
`probA, probB`	numeric vectors of length k(k-1)/2, k number of classes, containing the parameters of the logistic distributions fitted to the decision values of the binary classifiers (1 / (1 + exp(a x + b))).

Note

Data are scaled internally, usually yielding better results.

Parameters of SVM-models usually *must* be tuned to yield sensible results!

Author(s)

David Meyer (based on C/C++-code by Chih-Chung Chang and Chih-Jen Lin)
David.Meyer@R-project.org

References

- Chang, Chih-Chung and Lin, Chih-Jen:
  LIBSVM: a library for Support Vector Machines
  http://www.csie.ntu.edu.tw/~cjlin/libsvm
- Exact formulations of models, algorithms, etc. can be found in the document:
  Chang, Chih-Chung and Lin, Chih-Jen:
  LIBSVM: a library for Support Vector Machines

http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.ps.gz

- More implementation details and speed benchmarks can be found on: Rong-En Fan and Pai-Hsune Chen and Chih-Jen Lin: *Working Set Selection Using the Second Order Information for Training SVM*
  http://www.csie.ntu.edu.tw/~cjlin/papers/quadworkset.pdf

See Also

`predict.svm` `plot.svm` `tune.svm` `matrix.csr` (in package SparseM)

Examples

```
data(iris)
attach(iris)

## classification mode
# default with factor response:
model <- svm(Species ~ ., data = iris)

# alternatively the traditional interface:
x <- subset(iris, select = -Species)
y <- Species
model <- svm(x, y)

print(model)
summary(model)

# test with train data
pred <- predict(model, x)
# (same as:)
pred <- fitted(model)

# Check accuracy:
table(pred, y)

# compute decision values and probabilities:
pred <- predict(model, x, decision.values = TRUE)
attr(pred, "decision.values")[1:4,]

# visualize (classes by color, SV by crosses):
plot(cmdscale(dist(iris[,-5])),
     col = as.integer(iris[,5]),
     pch = c("o","+")[1:150 %in% model$index + 1])

## try regression mode on two dimensions

# create data
x <- seq(0.1, 5, by = 0.05)
y <- log(x) + rnorm(x, sd = 0.2)

# estimate model and predict input values
m   <- svm(x, y)
new <- predict(m, x)

# visualize
plot(x, y)
points(x, log(x), col = 2)
points(x, new, col = 4)

## density-estimation

# create 2-dim. normal with rho=0:
X <- data.frame(a = rnorm(1000), b = rnorm(1000))
attach(X)

# traditional way:
m <- svm(X, gamma = 0.1)
```

```
# formula interface:
m <- svm(~., data = X, gamma = 0.1)
# or:
m <- svm(~ a + b, gamma = 0.1)

# test:
newdata <- data.frame(a = c(0, 4), b = c(0, 4))
predict (m, newdata)

# visualize:
plot(X, col = 1:1000 %in% m$index + 1, xlim = c(-5,5), ylim=c(-5,5))
points(newdata, pch = "+", col = 2, cex = 5)

# weights: (example not particularly sensible)
i2 <- iris
levels(i2$Species)[3] <- "versicolor"
summary(i2$Species)
wts <- 100 / table(i2$Species)
wts
m <- svm(Species ~ ., data = i2, class.weights = wts)
```

[Package *e1071* version 1.6-7]

In [7]:

```
model <- svm(Species ~ ., data=iris)
```

In [9]:

```
summary(model)
```

Out[9]:

```
Call:
svm(formula = Species ~ ., data = iris)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.25

Number of Support Vectors:  51

 ( 8 22 21 )


Number of Classes:  3

Levels:
 setosa versicolor virginica
```

Example Predictions

We have a small data set, so instead of splitting it into training and testing sets (which you should always try to do!) we'll just score out model against the same data it was tested against:

In [11]:

```
predicted.values <- predict(model,iris[1:4])
```

In [13]:

```
table(predicted.values,iris[,5])
```

Out[13]:

```
predicted.values setosa versicolor virginica
      setosa         50          0         0
      versicolor      0         48         2
      virginica       0          2        48
```

Just as we've seen before with the iris data set, the setosa values are easily separated with some noise between versicolor and virginica.

Advanced - Tuning

We can try to tune parameters to attempt to improve our model, you can refer to the help() documentation to understand what each of these parameters stands for. We use the tune function:

In [19]:

```
# Tune for combos of gamma 0.5,1,2
# and costs 1/10 , 10 , 100
tune.results <- tune(svm,train.x=iris[1:4],train.y=iris[,5],kernel='radial',
                  ranges=list(cost=10^(-1:2), gamma=c(.5,1,2)))
```

In [20]:

```
summary(tune.results)
```

Out[20]:

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
    1   0.5

- best performance: 0.04666667

- Detailed performance results:
    cost gamma      error dispersion
1    0.1   0.5 0.05333333 0.06885304
2    1.0   0.5 0.04666667 0.04499657
3   10.0   0.5 0.05333333 0.05258738
4  100.0   0.5 0.06000000 0.04919099
5    0.1   1.0 0.06666667 0.05443311
6    1.0   1.0 0.06000000 0.06629526
7   10.0   1.0 0.06000000 0.06629526
8  100.0   1.0 0.07333333 0.06629526
9    0.1   2.0 0.10000000 0.06478835
10   1.0   2.0 0.06000000 0.06629526
11  10.0   2.0 0.06000000 0.06629526
12 100.0   2.0 0.06666667 0.07027284
```

We can now see that the best performance occurs with cost=1 and gamma=0.5. You could try to train the model again with these specific parameters in hopes of having a better model:

In [21]:

```
tuned.svm <- svm(Species ~ ., data=iris, kernel="radial", cost=1, gamma=0.5)
```

In [22]:

```
summary(tuned.svm)
```

Out[22]:

```
Call:
svm(formula = Species ~ ., data = iris, kernel = "radial", cost = 1,
    gamma = 0.5)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.5

Number of Support Vectors:  59

 ( 11 23 25 )


Number of Classes:  3

Levels:
 setosa versicolor virginica
```

In [26]:

```
tuned.predicted.values <- predict(tuned.svm,iris[1:4])
```

In [27]:

```
table(tuned.predicted.values,iris[,5])
```

Out[27]:

```
tuned.predicted.values setosa versicolor virginica
            setosa         50          0         0
            versicolor      0         48         2
            virginica       0          2        48
```

Looks like we weren't able to improve on our model! The concept of trying to tune for parameters by just trying many combinations in generally known as a grid search. In this case, we likely have too little data to actually improve our model through careful parameter selection.

Great Job!

Data Science Career Opportunities:

In a world where 2.5 quintillion bytes of data is produced every day, a professional who can organize this humongous data to provide business solutions is indeed the hero! Much has been spoken about why Big Data is here to stay and why Big Data Analytics is the best career move. Building on what's alreadybeen written and said, let's discuss Data Science career opportunities and why 'Data Scientist' is the sexiest job title of the 21st century.

Data Science Career Opportunities

A Data Scientist, according to Harvard Business Review, "is a high-ranking professional with the training and curiosity to make discoveries in the world of Big Data". Therefore it comes as no surprise that Data Scientists are coveted professionals in the Big Data Analytics and IT industry.

With experts predicting that 40 zettabytes of data will be in existence by 2020 (Source), Data Science career opportunities will only shoot through the roof! Shortage of skilled professionals in a world which is increasingly turning to data for decision making has also led to the huge demand for Data Scientists in start-ups as well as well-established companies. A McKinsey Global Institute study states that by 2018, the US alone will face

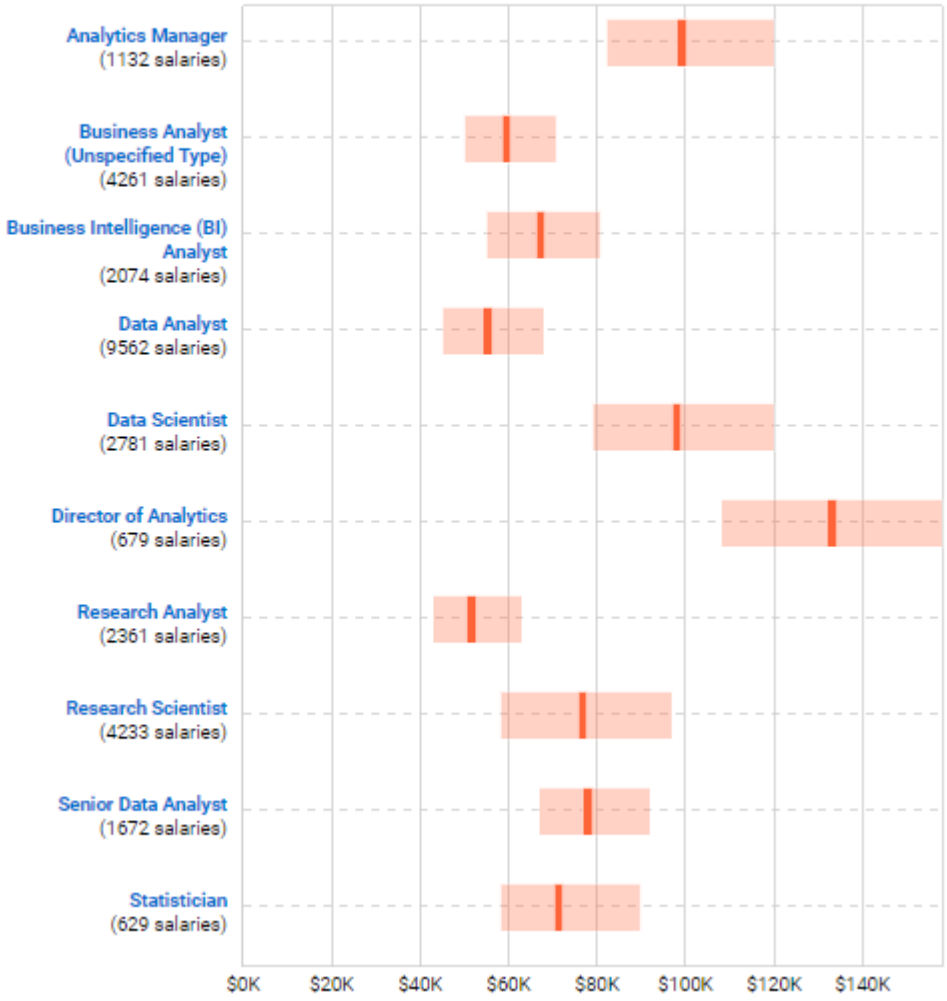
a shortage of about 190,000 professionals with deep analytical skills. With the Big Data wave showing no signs of slowing down, there’s a rush among global companies to hire Data Scientists to tame their business-critical Big Data.

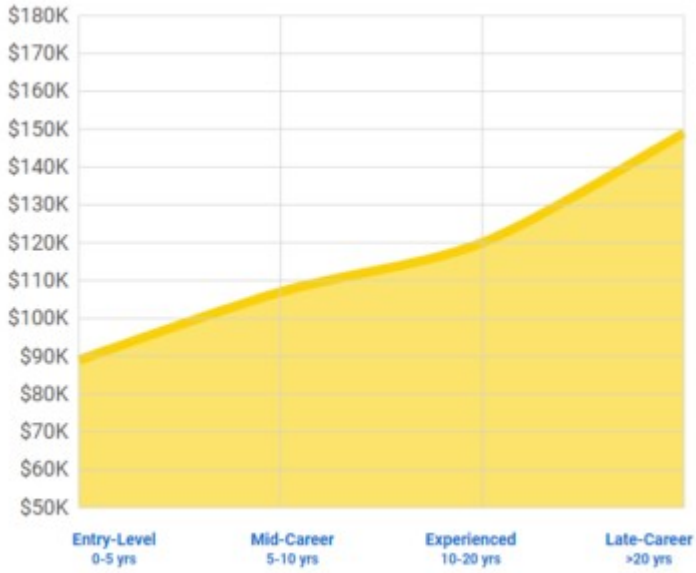
Data Scientist Salary Trends

A report by Glassdoor shows that Data scientists lead the pack for the best jobs in America. The report goes on to say that the median salary for a Data Scientist is an impressive $91,470 in the US and ₹622,162 and there are over 2300 job openings posted on the site (Source).

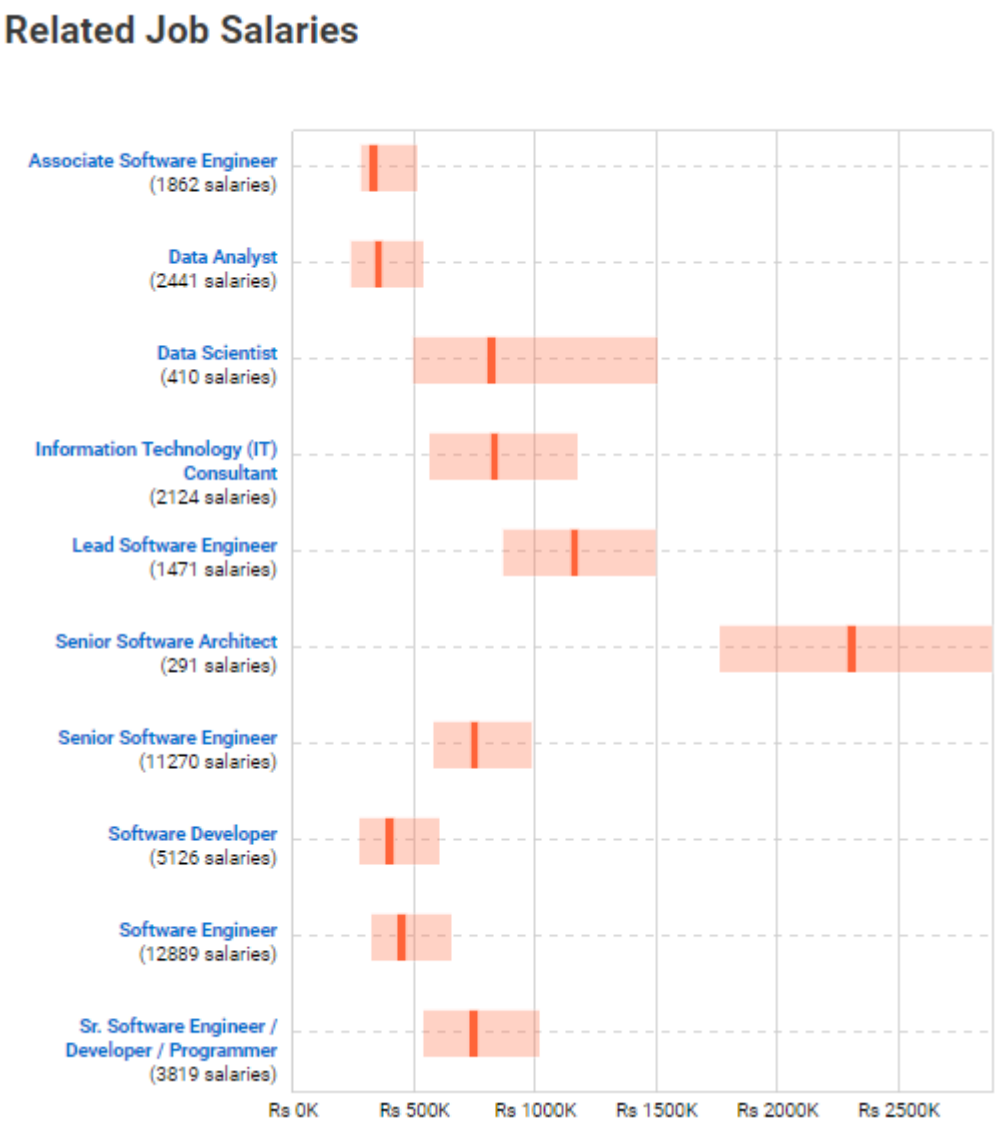
On Indeed.com, the average Data Scientist salaries for job postings in the US are 80% higher than average salaries for all job postings nationwide, as of May 2019.

Related Job Salaries





In India the trend is no different; as of May 2019, the median salary for a Data Scientist role is Rs. 622,162 according to Payscale.com.



Data Scientist Job Roles

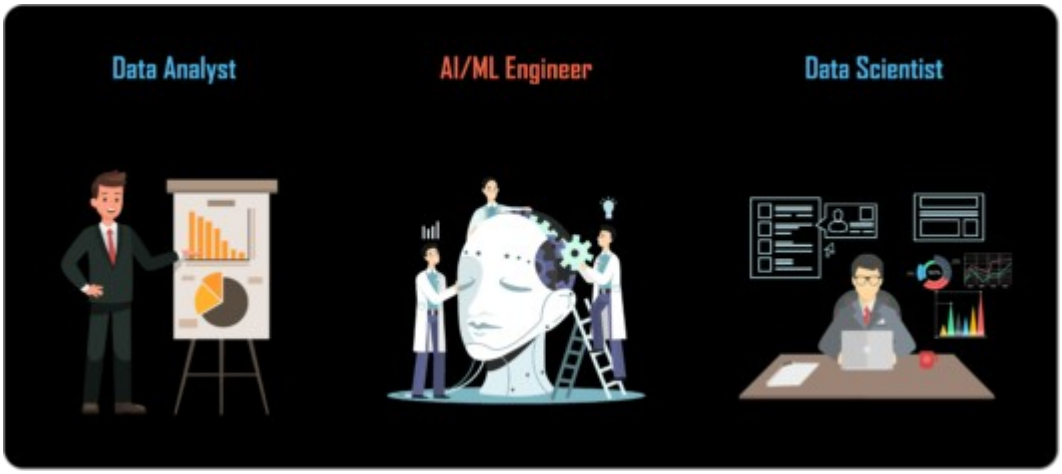
A Data Scientist dons many hats in his/her workplace. Not only are Data Scientists responsible for business analytics, they are also involved in building data products and software platforms,

along with developing visualizations and machine learning algorithms.

Some of the prominent Data Scientist job titles are:

- •Data Scientist
- •Data Architect
- •Data Administrator
- •Data Analyst
- •Business Analyst
- •Data/Analytics Manager
- •Business Intelligence Manager

TOP DATA SCIENCE: PROFILES

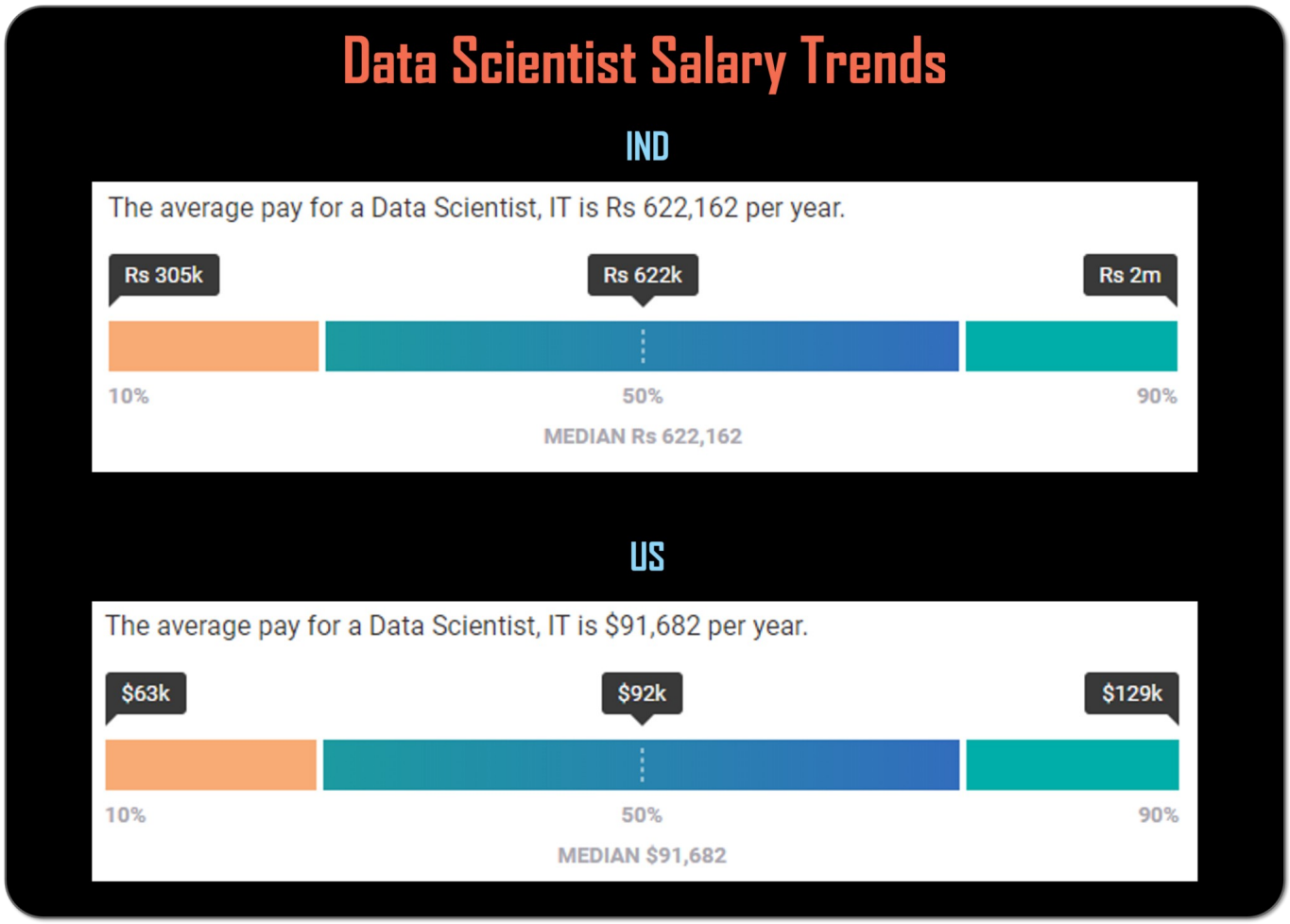


Hot Data Science Skills

Coding skills clubbed with knowledge of statistics and the ability to think critically, make up the arsenal of a successful data scientist. Some of the in-demand Data Scientist skills that will fetch big career opportunities in Data Science are:

- Programming Languages: R/Python/Java
- Statistics and Applied Mathematics
- Working Knowledge of Hadoop and Spark
- Databases: SQL and NoSQL
- Machine Learning and Neural Networks
- Proficiency in Deep Learning Frameworks: TensorFlow, Keras, Pytorch
- Creative Thinking & Industry Knowledge

The Payscale.com chart below shows the average **Data Scientist Salary** by skills in the USA and India.



Currency: India – ₹, US – $

The upward swing in Data Science career opportunities is expected to continue for a long time to come. As data pervades our life and companies try to make sense of the data generated, skilled Data Scientists will be continued to be wooed by businesses big and small. Case in point, a look at the jobs board on Indeed.com reveals top companies competing with each other to hire Data Scientists. A few big names include Facebook, Twitter, Airbnb, Apple, LinkedIn, IBM and PayPal among others.

The time is ripe to up-skill in Data Science and Big Data Analytics to take advantage of the Data Science career opportunities that come your way.

Data Science Trends in 2019

When it comes to major Data Science trends to watch in 2019, the co-founder and CEO of Kaggle, Anthony Goldbloom. has predicted that very soon Data Centers will be replaced by departmental or business-specific Data Science teams.

As discussed in *Data Science Trends in 2018*, last year's major trends continued from 2017 as the growth of Big Data, AI, Machine Learning (ML), Edge Computing, Blockchain, and Digital Twins all amplified. Big Data and Data Science, the two resounding themes of 2017, were drowned by the recurrent theme of 2018: "the meshing of the physical and digital world." In 2019, this mesh will continue to become a mainstream reality in businesses.

What Does This Mean?

•The year 2019 will witness businesses across the globe implementing and exploring the "meshing of the physical and the virtual world."
•The field of Data Science is currently in transitional mode, and 2019 may well set the stage for advanced data technologies to take over routine business processes for higher efficiency and productivity while the human data scientists tackle more complex challenges. The roles and responsibilities of data scientists will continue to evolve and the title "data scientist" will become too limiting for such diverse roles
•Increased business automation will not lead to the obsolescence of data scientists because smart tools will offer higher capabilities to the human experts. The data scientist is becoming a "reinvented" scientist with sufficient free time in the future to explore complex business problems, while advanced technologies take over the routine processes.

•Automated ML systems promise to take Predictive Analytics to the next level, but also come with steep learning curves.

•Chatbots, Virtual Reality (VR), and Augmented Reality (AR) will together revolutionize product and service marketing. AI-enabled technologies will deliver truly personalized customer experience through interactive demos, live simulations, and visualization of custom solutions.

•Blockchain will go mainstream, encompassing diverse industry sectors from banking and finance to health insurance. Blockchain not only promises data protection and fraud detection but also smart contracts and automated governance in Data Management.

•According to *The Future Impact of Data Science on Business Analytics*, data scientists will play a leadership role in Business Analytics, leading the connected machines through unexplored paths. Augmented Analytics will revolutionize predictive intelligence.

Now, let's see how the above trends unfold in 2019.

What are the roles and responsibilities of a data scientist?

Data scientists are big data wranglers. They take an enormous mass of messy data points (unstructured and structured) and use their formidable skills in math, statistics and programming to clean, massage and organize them. Then they apply all their analytic powers – industry knowledge, contextual understanding, skepticism of existing assumptions – to uncover hidden solutions to business challenges.

Data Scientist Responsibilities -

"**A data scientist is someone who is better at statistics than any software engineer and better at software engineering than any statistician.**"

On any given day, a data scientist may be required to:

Conduct undirected research and frame open-ended industry questions

Extract huge volumes of data from multiple internal and external sources

Employ sophisticated analytics programs, machine learning and statistical methods to prepare data for use in predictive and prescriptive modeling

Thoroughly clean and prune data to discard irrelevant information

Explore and examine data from a variety of angles to determine hidden weaknesses, trends and/or opportunities

Devise data-driven solutions to the most pressing challenges

Invent new algorithms to solve problems and build new tools to automate work

Communicate predictions and findings to management and IT departments through effective data visualizations and reports

Recommend cost-effective changes to existing procedures and strategies

Every company will have a different take on job tasks. Some treat their data scientists as glorified data analysts or combine their duties with data engineers; others need top-level analytics experts skilled in intense machine learning and data visualizations.

As data scientists achieve new levels of experience or change jobs, their responsibilities invariably change. For example, a person working alone in a mid-size company may spend a good portion of the day in data cleaning and munging. A high-level employee in a business that offers data-based services may be asked to structure big data projects or create new products.