Data Exploration with R

info@atozacademy.nl

Course Overview

Data Exploration with R is a 15 hour intensive program wherein the participants will "learn by doing". In this regard, two case studies will be shared, which will be prepared from

- https://www.kaggle.com/datasets

The idea is that participants apply the techniques that they learn during the first six sessions, on the cases and prepare a presentation of their findings for the last two sessions. Also the participants will be expected to submit their R code before the last two sessions. Each case will require the participants to spend approximately 20-25 hours.

Course Plan

1. **Data Exploration with R (3 hours)** In the introduction sessions we will cover the following
   - Setup R and Introduction to RStudio
   - Reading files in R. Introduction to data frames and basic operation on Data frames
2. **Data Processing with R (1.5 hours)**

   In these sessions we will cover different control structures in R. The participants will learn to write their first "R function" and ways to clean, aggregate and transform data:
3. **Exploratory Data Analysis and Data Visualization with R (3 hours)**

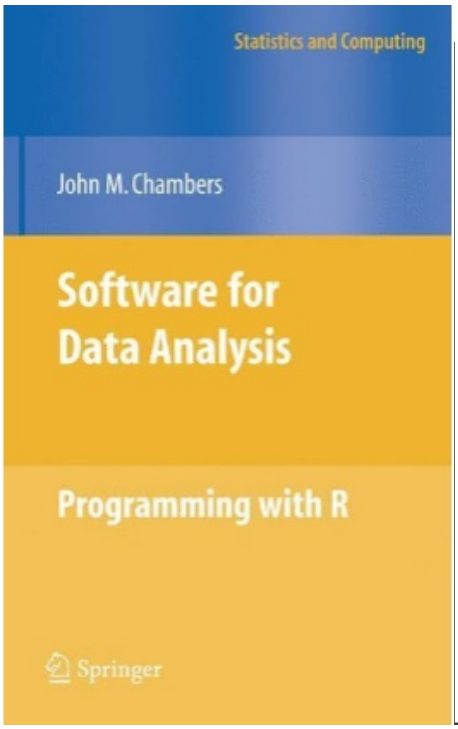
   In these sessions the participants will learn how to compute summary statistics of data in R, visualize data sets and learn to understand and explain "the story of the data set"
4. **Statistical Modelling with R (1.5 hours)**

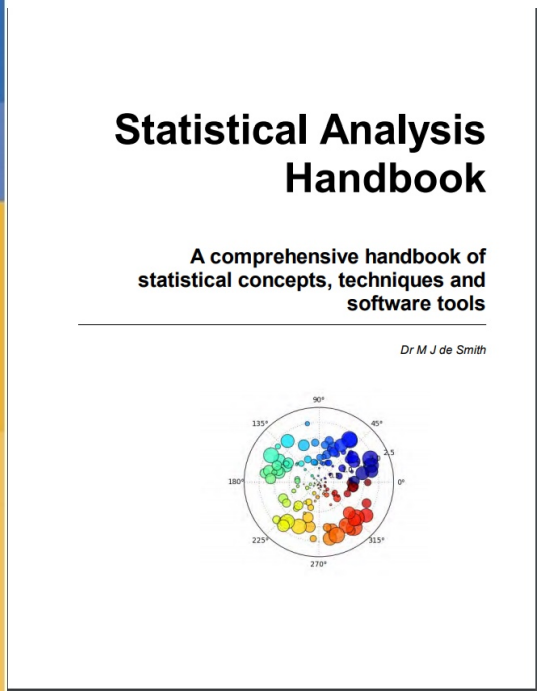
   In these sessions, the participants will learn how to build a linear regression model on a multivariate data set and interpret its results.

5. **Case Analysis (6 hours)**

   In these sessions, the participants will present, analyse and brainstorm on the application of different methods that they have been introduced to in the previous sessions, namely data processing, data exploration, data visualization and predictive modelling to two real world data sets.

The Books





Session	Chapters in Chambers	Chapters in Smith
1	1, 2, 6.1, 6.2	2, 19
2	3, 6.3, 6.4, 6.5, 6.6, 6.7	6
3	7	7
4	6.8, 6.9, 6.10	3, 4, 16.1

Throughout this manual, the participant will come across the following three icons

This icon urges the reader to think about the question that has been asked and such questions will be the main discussion points during sessions.

This icon will be associated with an exercise and the participants are expected to work on these.

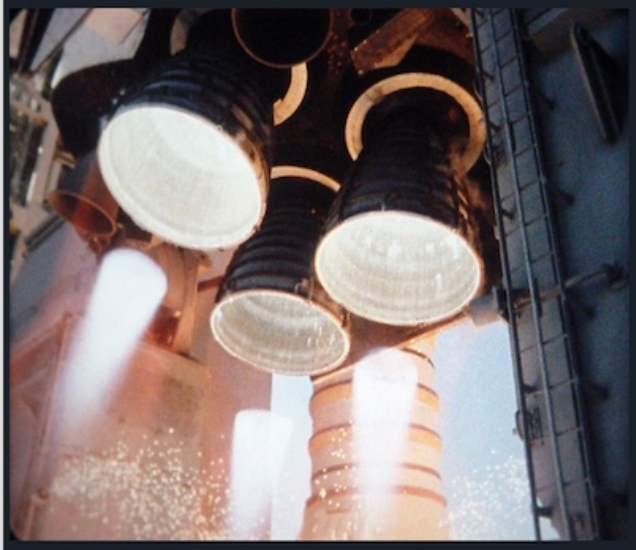
This icon will be used when we want the reader to pay attention to a particular idea.

Contents

Chariot *is on fire !*

01

The Liftoff



Hey,

Whats up?
If you are at this page, you are at T-0 and ready to begin your voyage. Congratulations!!
In chapter 1, we will define the mission, and tell you about the importance of this mission. Moreover, we will introduce you to your chariot that will help you accomplish this mission.

Like, any lift off , you will feel the thrust as we hurl away into the unknown. So buckle your seat belts!!!

Lets Go!

1. The Liftoff

1.1 The Mission

"Data Exploration"

Data Exploration entails the following activities:

1. Formulation of meaningful questions
2. Organization of data in a way to answer the questions

Any real world entity can be treated as an object.

Every object has a set of attributes.

The attributes of an object can be assigned values.

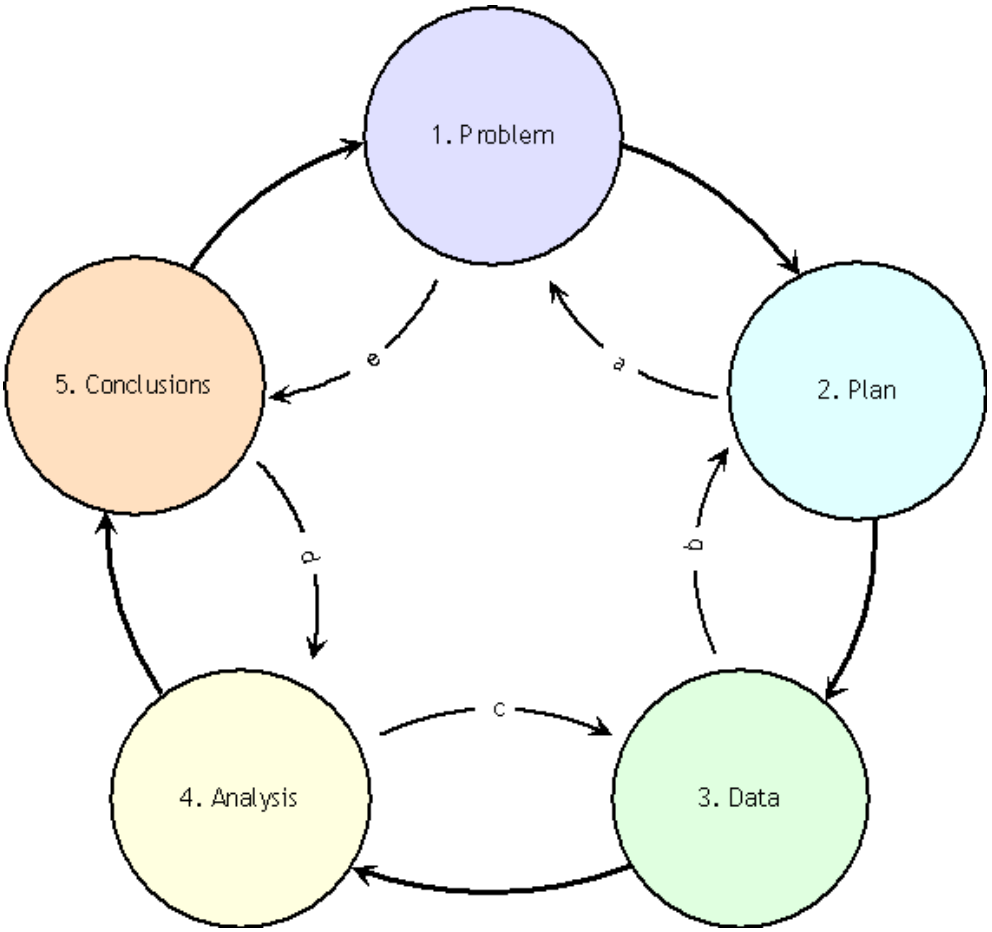
For example, a student is an object. A student can have *age, gender, score* etc.. as attributes. Some of these values can be numeric, for example age and score, while other attributes can be categorical, for example gender. When we collect attribute values for multiple (possibly similar) objects, we end up getting a data set.

The tabular representation of such data is one where the rows correspond to different objects, the columns correspond to the attributes of these objects and each cell corresponds to the attribute value for the object.

Data exploration is the step, in which we try to identify patterns in the data thereby leading us to prove or disprove our hypothesis about the objects. For example, if we constructed a hypothesis that *female students score higher than male students.* Then, in order to prove or disprove it we would have to first collect the data, organize it, and then try to find patterns in the data that would confirm the correctness/incorrectness of the hypothesis.

1.1.1 Stages of Data Exploration

"It is as well to remember the following truths about models: all models are wrong; some models are better than others [George Box said more useful]; the correct model can never be known with certainty; and the simpler a model the better it is!"



"The purpose of the Conclusion stage is to report the results of the study in the language of the Problem. Concise numerical summaries and presentation graphics [tabulations, visualizations] should be used to clarify the discussion. Statistical jargon should be avoided. As well, the Conclusion provides an opportunity to discuss the strengths and weaknesses of the Plan, Data and Analysis especially in regards to possible errors that may have arisen" Mackay and Oldford (2000)

1.1.2 The Geometry of Data

In order to find patterns among objects of interest, we should be able to identify similar objects. One advantage of finding similar objects is that it allows us to find smaller cohorts of objects within which we can try to find patterns, that may have been non-existent in the larger group.

For example, we may observe from the student data that gender does not correlate to score, but if we group the data by age, we may see that for certain age groups, gender does correlate with score.

It is important to compare objects by considering all their attributes. In order to analyze or compute on all attribute of an object, we need to give it an abstract representation. This is achieved by ***representing data in the Cartesian coordinate system*** where each coordinate corresponds to an attribute.

This is fairly easy when the attributes take a numeric value, for example if age = 17 and score = 75, its Cartesian coordinate representation would be (17,25).

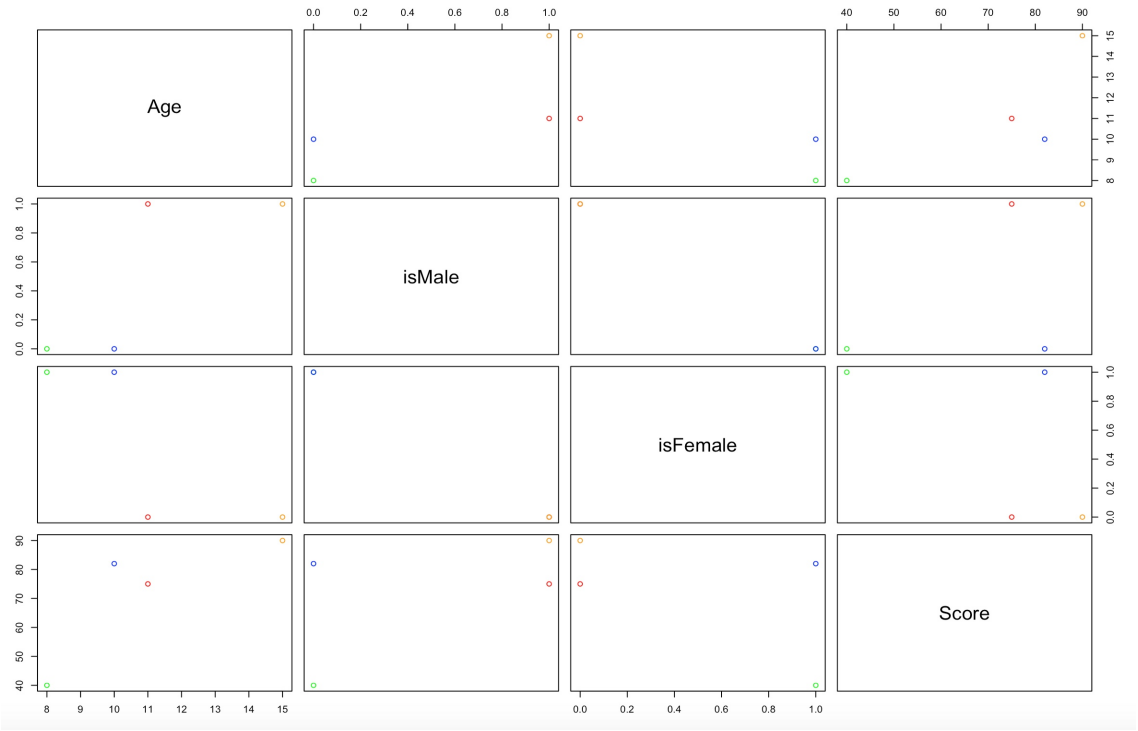
The representation gets a bit complicated for categorical attributes. In order to give it a Cartesian coordinate representation, we need to map each attribute value to a number. However a number system also has the inherent concept of ordering wherein assigning a higher number to something means assigning higher weight to it. So, for the gender attribute, if we were to map the attribute value "Male" to 1 and "Female" to 2, internally this means that we are assigning a higher value to the attribute value "Female"

The correct way to do this is to treat each of the attribute values as representing a coordinate axis, such that when an attribute value is associated with an object it gets a value 1 and all other attribute values are set to 0. This is best explained by an example.

Id	**Age**	**Gender**	**Score**
001	11	M	75
002	8	F	40
003	10	F	82
004	15	M	90

Now, the Cartesian coordinate representation of this data is shown in the table. $X1, X2, X3, X4$ are the coordinate axis.

Id	**Age (X1)**	**isMale (X2)**	**isFemale (X3)**	**Score (X4)**
001	11	1	0	75
002	8	0	1	40
003	10	0	1	82
004	15	1	0	90



Visualization of data in the Cartesian coordinate system , when the number of dimensions is more than three, is quite challenging. A easy workaround is a 2-D pairwise visualization of attributes.

Explain the plot.

1.1.3 Challenges of Data Exploration

"Lies, Damned Lies and Statistics"

The analysis stage of data exploration can be significantly hampered by

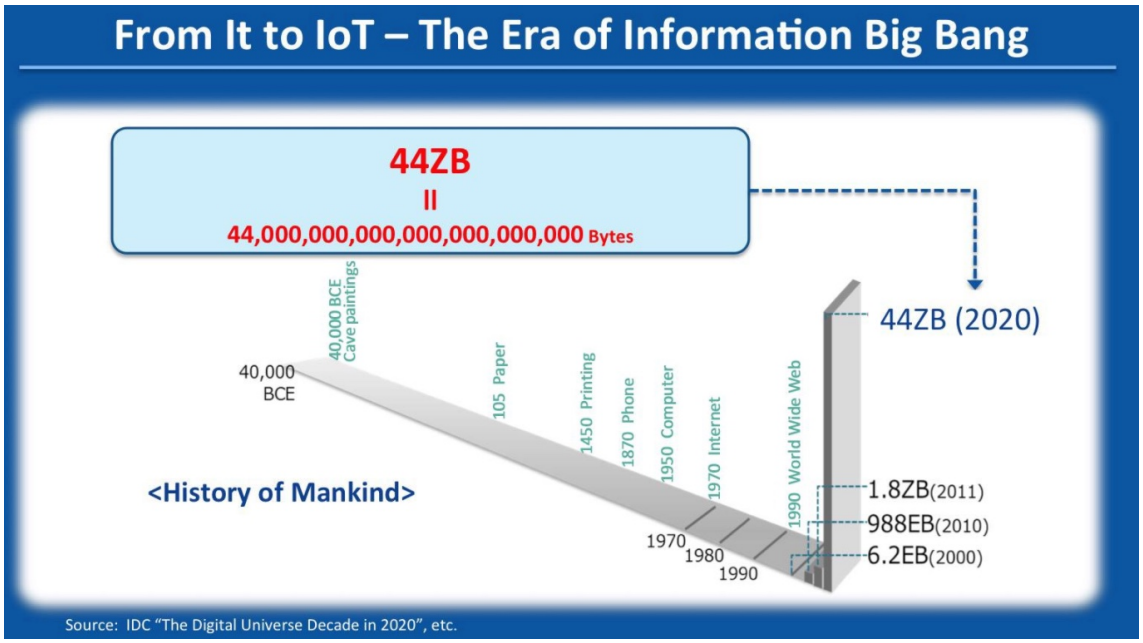
- inadequate or unrepresentative data
- misleading visualization of results

This can lead to inadequate reasoning on the basis of results and therefore drawing of incorrect conclusions.

Argue why the example in section 1.1.2 poses challenges to data exploration.

Explore the data set in exercise1.csv. Which *Stages of Data Exploration* did you apply while exploring the data set in exercise1.csv? What challenges did you face in your exploration?

1.1.4 Data and Society

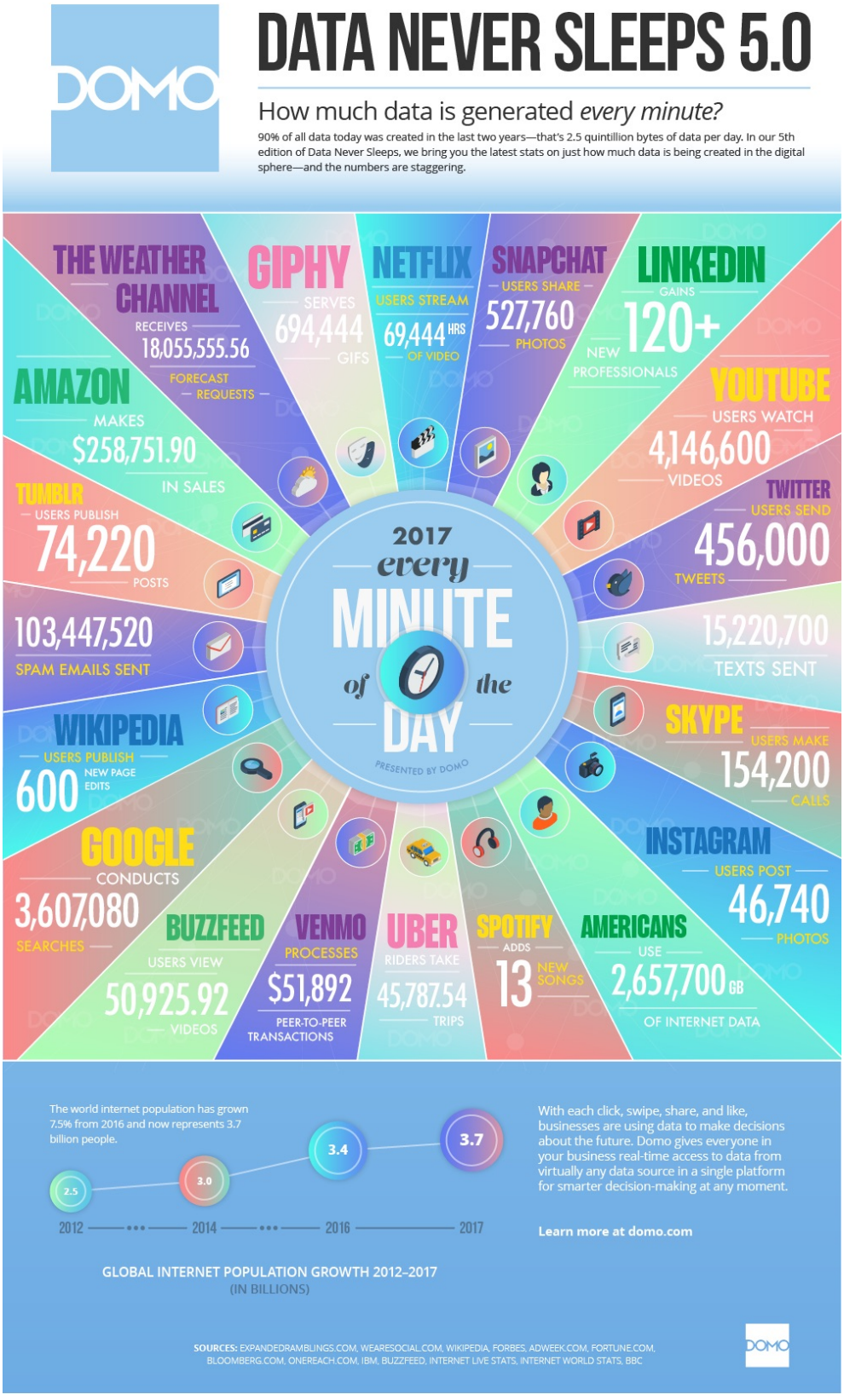


It is true that data is ubiquitous to modern society and is being generated at an unprecedented scale at high ***volumes***, ***velocity*** and ***variety*** (the **3V's**). Try categorizing each of the data source in the info graphic "*Data Never Sleeps 5.0*"

- into one of the **3V's**
- ***high noise-to-signal*** ratio source v/s ***low noise-to-signal*** ratio source
- sources where high volume ***correlates*** to better insights

Read Section 19 of the *Statistical Analysis Handbook*

Argue why the **4'th V** that you have just read about necessitates *data exploration*


DOMO
DATA NEVER SLEEPS 5.0
How much data is generated every minute?
90% of all data today was created in the last two years—that's 2.5 quintillion bytes of data per day. In our 5th edition of Data Never Sleeps, we bring you the latest stats on just how much data is being created in the digital sphere—and the numbers are staggering.
2017 every MINUTE of the DAY
PRESENTED BY DOMO
THE WEATHER CHANNEL
RECEIVES 18,055,555.56 FORECAST REQUESTS
GIPHY
SERVES 694,444 GIFS
NETFLIX
USERS STREAM 69,444 HRS OF VIDEO
SNAPCHAT
USERS SHARE 527,760 PHOTOS
LINKEDIN
GAINS 120+ NEW PROFESSIONALS
AMAZON
MAKES $258,751.90 IN SALES
YOUTUBE
USERS WATCH 4,146,600 VIDEOS
TUMBLR
USERS PUBLISH 74,220 POSTS
TWITTER
USERS SEND 456,000 TWEETS
103,447,520 SPAM EMAILS SENT
15,220,700 TEXTS SENT
WIKIPEDIA
USERS PUBLISH 600 NEW PAGE EDITS
SKYPE
USERS MAKE 154,200 CALLS
GOOGLE
CONDUCTS 3,607,080 SEARCHES
INSTAGRAM
USERS POST 46,740 PHOTOS
BUZZFEED
USERS VIEW 50,925.92 VIDEOS
VENMO
PROCESSES $51,892 PEER-TO-PEER TRANSACTIONS
UBER
RIDERS TAKE 45,787.54 TRIPS
SPOTIFY
ADDS 13 NEW SONGS
AMERICANS
USE 2,657,700 GB OF INTERNET DATA
The world internet population has grown 7.5% from 2016 and now represents 3.7 billion people.
2.5
3.0
3.4
3.7
2012
2014
2016
2017
With each click, swipe, share, and like, businesses are using data to make decisions about the future. Domo gives everyone in your business real-time access to data from virtually any data source in a single platform for smarter decision-making at any moment.
Learn more at domo.com
GLOBAL INTERNET POPULATION GROWTH 2012–2017
(IN BILLIONS)
SOURCES: EXPANDEDRAMBLINGS.COM, WEARESOCIAL.COM, WIKIPEDIA, FORBES, ADWEEK.COM, FORTUNE.COM, BLOOMBERG.COM, ONEREACH.COM, IBM, BUZZFEED, INTERNET LIVE STATS, INTERNET WORLD STATS, BBC
DOMO

1.1.5 Software for Data Exploration

> ***"Trustworthy, Flexible and Efficient Software: The Prime Directives"***
>
> The task of data exploration requires a tool that allows a user to ask meaningful questions about their applications quickly and flexibly. A wide range of techniques is needed to facilitate stages 1-4 of data exploration thereby requiring the data exploration tool to be flexible. Moreover, the tool should be able to provide answers to the questions asked "*within an acceptable time frame*". The **3V's**, and the questions that are posed on the data, require complex transformations and computations on the data, whose correctness cannot be often verified by direct observation of the results in stage 5. Therefore, it is imperative that the underlying implementations of the transformations and computations (which is often ***encapsulated***) are correct.

How does Microsoft Excel provide ***Abstraction via Encapsulation***. Think of an example where the trustworthiness comes into play.

It is extremely hard to achieve *trustworthiness*, *flexibility* and *efficiency* at the same time. As you go ahead in the course try to identify trade-offs between the three directives in R.

1.2 Why R?

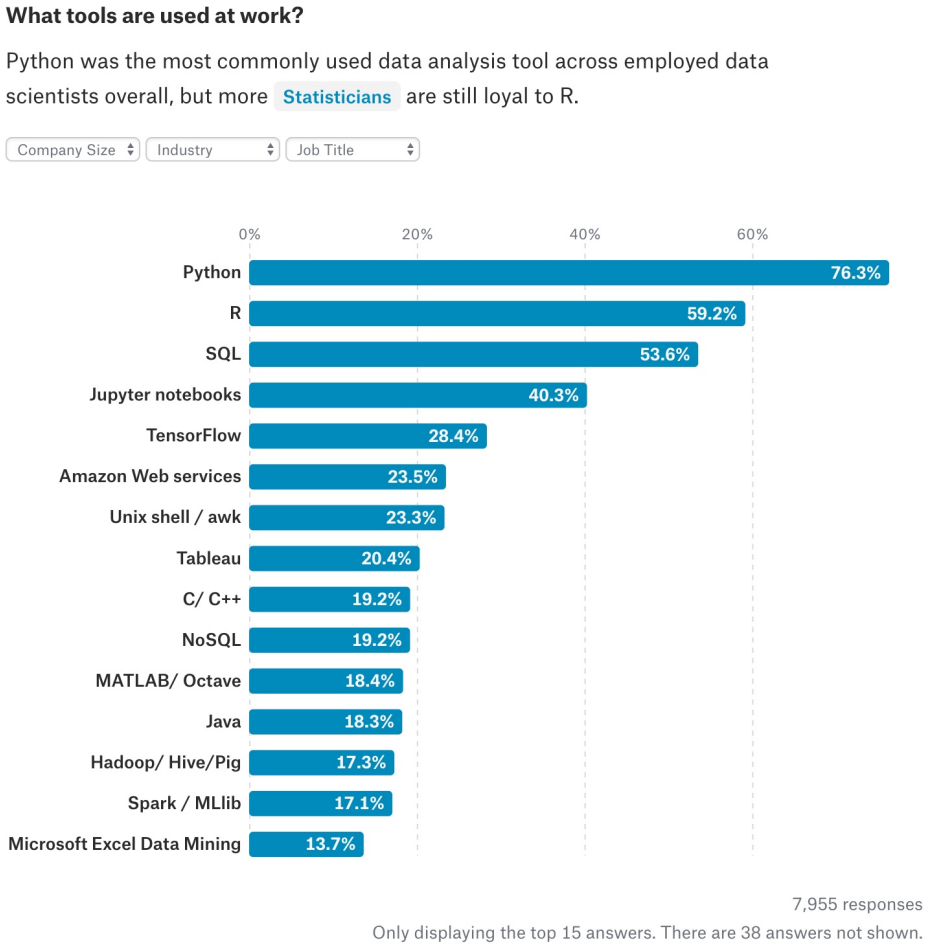
Before delving into the question of *why R?*, let us first take a look at how R fares with respect to to its competitors.The following two plots show the results of two surveys conducted in the year 2017. The first plot shows the result of a similar survey, conducted worldwide, by Kaggle.The second survey was conducted by O'Rielly in Europe and shows the popularity of different tools among data analysts/scientists.As is evident from both plots, R is a highly popular tool among data analysts/scientists.

Think like an Analyst: Criticize the plots in terms of their inadequacy to provide the complete information. Argue, why and how would this lead to drawing of incorrect conclusions.

Read the reports

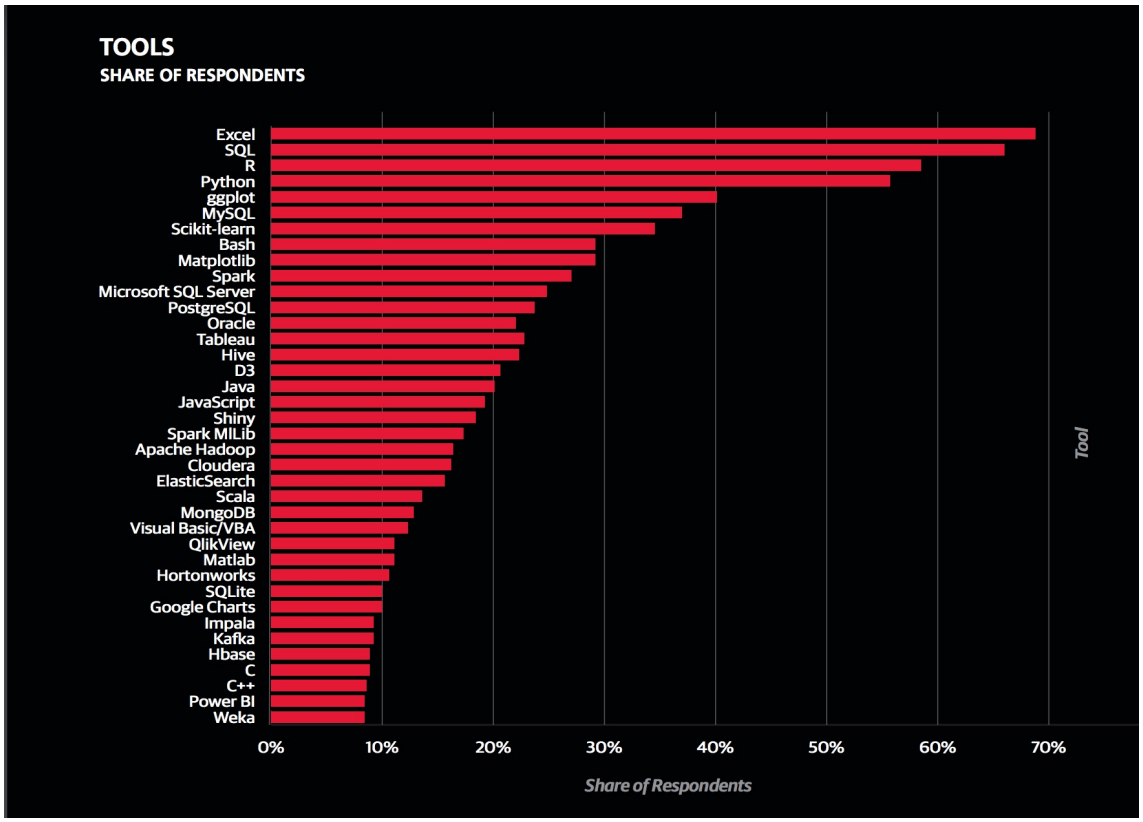
1. **"European Data Science Salary Survey"** published by O'REILLY in 2017
2. **"The State of Data Science & Machine Learning"** published by Kaggle in 2017

and get yourself acquainted with *stage 5* of the data exploration process.



1.2.1 A Brief History of R

- **1976**: Statistical programming language S developed at Bell Labs since 1976 (at the same time as UNIX)
- **1993**: Research project in Auckland, NZ by R. Gentleman and R. Ihaka
- **1995**: Released as an open-source software - generally compatible with the "S" language
- **1997**: R core group formed
- **2000**: R 1.0 released
- **2004**: First international user conference in Vienna
- **2013**: R 3.0.0 released
- **2017**: R at version 3.4.3



1.2.2 Concepts for Programming with R

Functional Programming

Software in R is written in a *functional style* that emphasizes on *encapsulating computations via functions* thereby allowing the programmer to create *abstractions by separating behavior from implementation.*

If you are given a set of N numbers whose average needs to be computed, think about how would the functional programming approach allow you to achieve abstraction and what would be its benefit? *Hint:* Think about how would the computation of average differ when N numbers are given in one go (***batched input***) as opposed to when you get every number one after the other (***streaming input***)

Classes and Methods

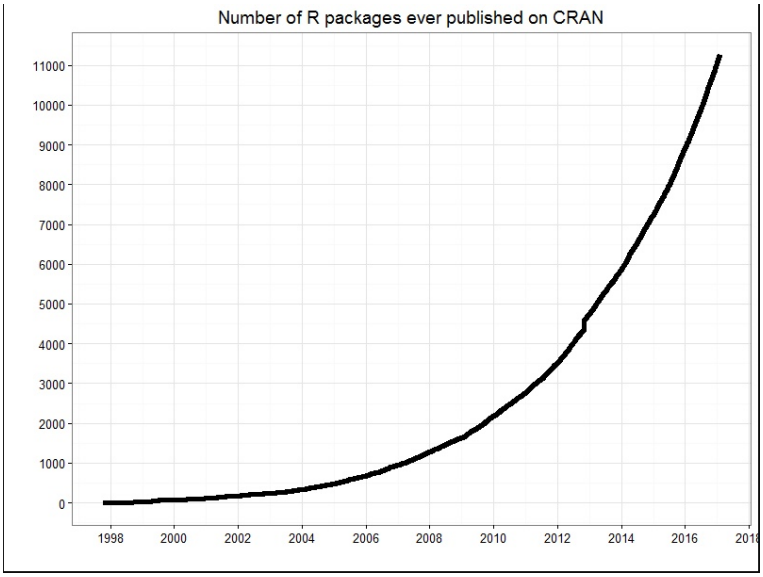
"Functions in R return objects". Therefore any kind of ***data in R is always an object***. While functions facilitate abstraction by encapsulating implementation of computation, *classes encapsulate objects. Methods in R knit together functions and classes.*

Data Frames

A *data frame* is organization of data as observations (rows) and attributes (columns). Data frames are one of the predominant data structures in R. Moreover, they are synonymous with the tabular representation of data as is found in spreadsheets and relational database systems. Such similarity in representation of data enables seamless integration between R and most spreadsheets and RDBMS systems.

Ecosystem

R is an open-source software, thereby providing access to source code sufficient to generate a working version of the software. R is distributed under a version of GPL (Gnu Public License). An effect of this is that R has a thriving community of contributors and active users. Users, seek out existing implementation of data analysis/modeling techniques from a repository called ***CRAN*** (https://cran.r-project.org/). When they do not find an existing implementation that suits their needs, they create their own implementations, *package* these implementations and contribute them back to CRAN. As a result of this CRAN today boasts a total of 12000+ packages. Please see the plot below that shows the growth in number of CRAN packages over time. This has resulted in R being unparalleled in the number of options it provides for data analysis. Moreover, R is the primary tool used for statistical research (and has been so for 20 years). Therefore, when new methods are developed, they are not just published as a paper - they are also published as an R package. This means R is always on the cutting edge of new technologies. Furthermore, R was designed as an interface language - a means to present a consistent language interface for algorithms written in other languages. Many packages work by providing R language bindings to other open-source software, making R a convenient hub for all kinds of algorithms and methods.



Did you notice the sudden discontinuity in the curve around the year 2013? Can you think of what might have caused this?

Argue ***Why R*** satisfies *the prime directives* and should therefore be our tool of choice for our mission of data exploration

1.3 RStudio

Watch the video on
https://www.rstudio.com/resources/webinars/rstudio-essentials-webinar-series-part-1/
and install R and RStudio on the computer that you will be accessing during the sessions.

It is important that the participants have R and RStudio installed on their computers before Session 1.

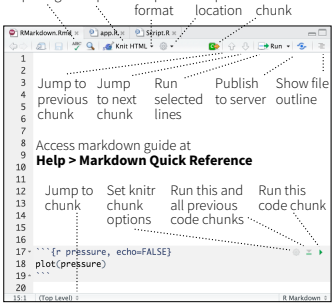
Download the RStudio IDE Cheat Sheet and use it as a reference while working with RStudio. In order to download the Cheat Sheet, *Open RStudio > Go to Help Menu > Cheatsheets > RStudio IDE Cheat Sheet*

RStudio IDE : : **CHEAT SHEET**

Documents and Apps

Open Shiny, R Markdown, knitr, Sweave, LaTeX, .Rd files and more in Source Pane

Check spelling | Render output | Choose output format | Choose output location | Insert code chunk

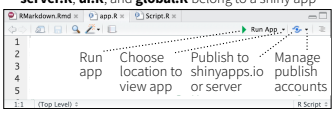


Jump to previous chunk | Jump to next chunk | Run selected lines | Publish to server | Show file outline

Access markdown guide at **Help > Markdown Quick Reference**

Jump to chunk | Set knitr chunk options | Run this and all previous code chunks | Run this code chunk

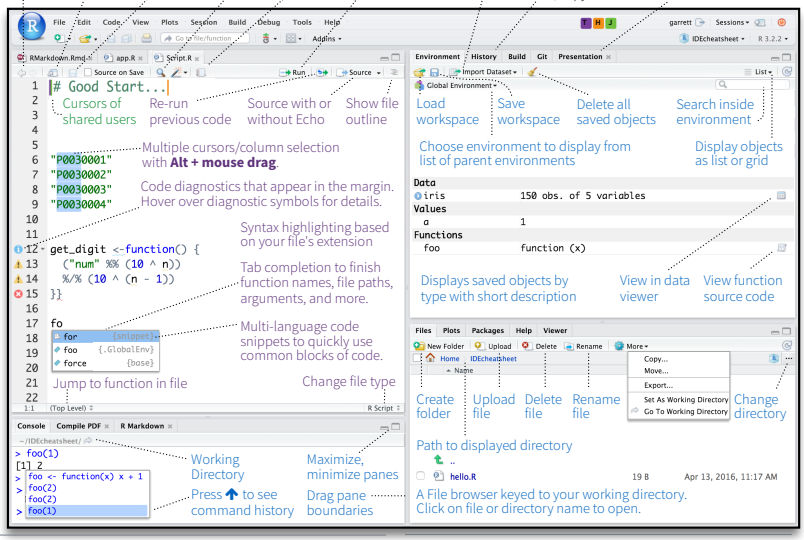
RStudio recognizes that files named **app.R**, **server.R**, **ui.R**, and **global.R** belong to a shiny app



Run app | Choose location to view app | Publish to shinyapps.io or server | Manage publish accounts

Write Code

Navigate tabs | Open in new window | Save | Find and replace | Compile as notebook | Run selected code

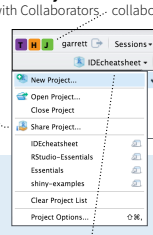


R Support

Import data with wizard | History of past commands to run/copy | Display .RPres slideshows **File > New File > R Presentation**

Pro Features

Share Project with Collaborators | Active shared collaborators



PROJECT SYSTEM

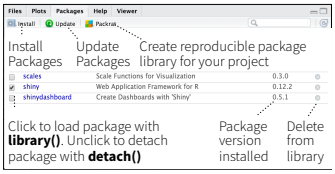
File > New Project

RStudio saves the call history, workspace, and working directory associated with a project. It reloads each when you re-open a project.

RStudio opens plots in a dedicated Plots pane

Navigate recent plots | Open in window | **Export plot** | Delete plot | Delete all plots

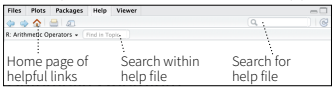
GUI Package manager lists every installed package



Install Packages | Update Packages | Create reproducible package library for your project

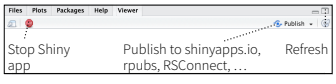
Click to load package with **library()**. Unclick to detach package with **detach()** | Package version installed | Delete from library

RStudio opens documentation in a dedicated Help pane



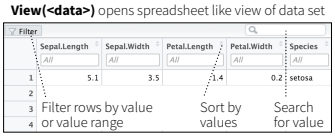
Home page of helpful links | Search within help file | Search for help file

Viewer Pane displays HTML content, such as Shiny apps, RMarkdown reports, and interactive visualizations



Stop Shiny app | Publish to shinyapps.io, rpubs, RSConnect, ... | Refresh

View(<data>) opens spreadsheet like view of data set



Filter rows by value or value range | Sort by values | Search for value

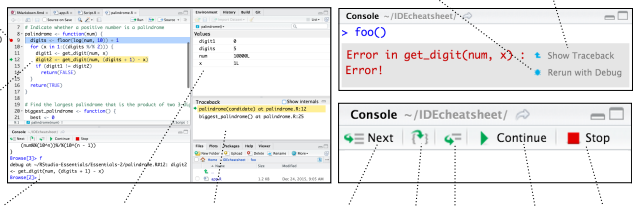
Debug Mode

Open with **debug()**, **browse()**, or a breakpoint. RStudio will open the debugger mode when it encounters a breakpoint while executing code.

Launch debugger mode from origin of error | Open traceback to examine the functions that R called before the error occurred

Click next to line number to add/remove a breakpoint.

Highlighted line shows where execution has paused

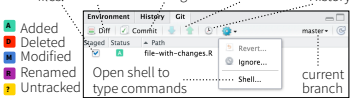


Run commands in environment where execution has paused | Examine variables in executing environment | Select function in traceback to debug | Step through code one line at a time | Step into and out of functions to run | Resume execution | Quit debug mode

Version Control with Git or SVN

Turn on at **Tools > Project Options > Git/SVN**

Stage files | Show file diff | Commit staged files | Push/Pull to remote | View History

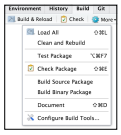


Package Writing

File > New Project > New Directory > R Package

Turn project into package, Enable roxygen documentation with **Tools > Project Options > Build Tools**

Roxygen guide at **Help > Roxygen Quick Reference**



RStudio

Chariot
is on boosters !

02

The Ascent

Hey,

Whats up?
Good job during liftoff!!!
As we leave behind the known and venture into the unknown, we will have to rely on the capabilities of our chariot.
In chapter 2, we will learn about and use our chariot's boosters to power us on.
The ride will be bumpy but definitely exhilarating!!!

Lets Continue!

2. The Ascent - PART A

2.1 Using R

R is an *interactive* environment, wherein the users gets immediate feedback (*results*) for their instructions (*expressions*). This is in contrast to programming languages like C++ or Java wherein a set of instructions (the program) has to be created and compiled before the results can be generated.In R, the interaction between the user and the system constitutes an *R Session*. During an R session, the user provides *expressions* to R for doing computations, displaying results, and creating objects for further use. R, first evaluates the expressions for its syntactic correctness and then performs the task, as specified by the expression. We will examine some basic expressions in the following sub-sections.

At the *console*, the user finds the > prompt where the user responds by typing and expression. Hereon, in this section every expression/set of expressions is mentioned in the gray box and is to be issued at the > prompt in the console. Moreover, following the motto of this course "learn by doing", the reader is strongly encouraged to try out each of these expressions and *read the corresponding help manual for each expression*. As is evident from the examples shown in this section, expressions are predominantly ***function calls with a set of arguments***.

R, like Python, MATLAB etc is a *dynamically typed language* which means that you won't have to define the type of the variable (which stores your data). As a result, the user can only focus on the *data stored by the variable* rather than having to worry about *how the data is stored in the variable*. This is in stark contrast to *statically typed languages* like C++ and Java.

How does a R *data frame* exemplify the *dynamic typing* feature of R?

The dynamic typing feature of R provides great flexibility but at what cost?

2.2 Basic Data Types

Objects are the center of computations in R, along with the function calls that create and use those objects.

What are objects in R? Why are functions and objects dual to each other?

The *class*() function

Every object in R has a *type* associated with it, which signifies the kind of data it contains e.g numbers, strings etc. The *class*() function accepts an object as an argument and returns its type.

2.2.1 Data Type for Boolean

The "logical" type

```
class(TRUE)

## [1] "logical"

class(FALSE)

## [1] "logical"
```

2.2.2 Data Type for Numbers

The "numeric" type

```
class(1)

## [1] "numeric"

class(3.141593)

## [1] "numeric"
```

2.2.3 Data Type for Strings

The "character" type

```
class("Carpe Diem")
```

```
## [1] "character"
```

2.2.4 Concatenating Strings

The *paste*() function

```
paste("Data Analysis","with R",sep=",")
```

```
## [1] "Data Analysis,with R"
```

2.3 Coercion

logical, integer, numeric, character

An ordering of data types from simple to complex. *Coercion* is the conversion from one type to another. However, coercion from any type to another is neither possible nor valid. The general rule for coercion is: *Coercion from a a simple type to a more complex type is valid*

The "is" and "as" methods

The *is* method can be used to test whether an object is from a given class.

The *as* method can be used to coerce an object from one class to another.

2.3.1 Coercion between Boolean and Numbers

```
is.logical(TRUE) ##check if the  data is of type logical
```

```
## [1] TRUE
```

```r
as.numeric(TRUE) ##coerce from logical to numeric
```

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

```r
is.numeric(10) ##check if the data is of type numeric
```

```
## [1] TRUE
```

```r
as.logical(10) ##coerce form numeric to logical.
```

```
## [1] TRUE
```

```r
##Any number that is not equal to 0 will be coerced to FALSE
as.logical(0) ##coerce from numeric to logical.
```

```
## [1] FALSE
```

```r
##Any number that is equal to 0 will be coerced to TRUE
as.logical(0.013) ##coerce from numeric to logical
```

```
## [1] TRUE
```

Give *real world* examples of when would you need to coerce between Boolean and numbers.

2.3.2 Coercion between Boolean and Strings

```r
is.logical(TRUE) ##check if the data is of type boolean
```

```
## [1] TRUE
```

```r
as.character(TRUE) ##coerce boolean to strings
```

```
## [1] "TRUE"
```

```r
is.character("TRUE")  ##check if the data is of type string
```

```
## [1] TRUE

as.logical("TRUE") ##coerce from string to boolean

## [1] TRUE

as.logical("Data") ##coerce any arbitrary string to logical

## [1] NA
```

As shown in the last example, R does not prevent the coercion of an arbitrary string to logical. It just generates NA.

Give *real world* examples of when would you need to coerce between Boolean and strings.

2.3.3 Coercion between Numbers and Strings

```
as.character(1) ## coerce from number to string

## [1] "1"

as.character(1.0023) ## coerce from number to string

## [1] "1.0023"

as.numeric("1") ## coerce from string to number

## [1] 1

as.numeric("Data") ## coerce from string to number

## Warning:  NAs introduced by coercion

## [1] NA
```

Note the difference in output in the last two cases. When a number is passed as a string to *as.numeric*(), it its able to detect the same and correctly coerces it to a number.

Give *real world* examples of when would you need to coerce between Boolean and strings.

2.4 Operators

In R, an operator is a function which can take one argument (unary operator) or two arguments (binary operator). Like any function in R, *an operator too has a return type.*

2.4.1 Arithmetic Operators

```
3 + 2 ## the '+' operator to add two numbers

## [1] 5

3 - 2 ## the '-' operator to substract two numbers

## [1] 1

3 * 2 ## the '*' operator to multiply two numbers

## [1] 6

3 / 2 ## the '/' operator to divide two numbers

## [1] 1.5

3 ^ 2 ## the '^' operator to calculate exponent

## [1] 9

3 %% 2 ## the '%%' operator to calculate modulus

## [1] 1
```

What is the return type for arithmetic operators

Try using the arithmetic operators '+' and '*' on operands of type logical. Explain your observations

Try using the arithmetic operators on operands of type character.

2.4.2 Comparison Operators

```
3 < 2 ## less than comparison operator
```

```
## [1] FALSE
```

```
3 <= 3 ## less than equal to comparison operator
```

```
## [1] TRUE
```

```
3 > 2 ## greater than comparison operator
```

```
## [1] TRUE
```

```
3 >= 3 ## greater than equal to comparison operator
```

```
## [1] TRUE
```

```
3 == 3 ## equality comparison operator
```

```
## [1] TRUE
```

```
3 != 2 ## inequality comparison operator
```

```
## [1] TRUE
```

What is the return type for comparison operators

Try using the comparison operators on strings. Explain your observations

Give *real world* examples of when would you need to use the comparison operator on strings

2.4.3 Logical Operators

```
(3 > 2) & (3 >= 2) ## the AND operator
```

```
## [1] TRUE
```

```
(3 > 2) & (3 < 2)
```

```
## [1] FALSE
```

```
(3 > 2) | (3 < 2) ## the OR operator
```

```
## [1] TRUE
```

```
!(3 > 2) ##the NOT operator
```

```
## [1] FALSE
```

What are the arguments to logical operators? What is its return type?

Which arithmetic operators can be used as a substitute for the logical operators

2.4.4 Special Operators

Want Help? Use ? operator

The **?** operator displays help for the topic that follows it.

```
?as.numeric
```

Another way of displaying documentation for a topic is by calling the *help*() function with the topic as its argument.

```
help("~")
```

Want to Write a Formula? Use $\sim$ operator

R, has its roots in statistical modelling and is extensively used for the purpose of creating data driven models. Therefore, one of the highlights of R is its ability to evaluate and process expressions written in the form of formulas. R, facilitates this with the $\sim$ operator. The following example shows how you can write an equation of the form $y = x^2$

```
y ~ x^2

## y ~ x^2

class(y ~ x^2)

## [1] "formula"
```

Want to Assign a Value? Use $<-$ operator

A number of times we will need to assign the object returned by a function to another object. This is done using the $<-$ operator

```
result <- 3 + 2
result

## [1] 5
```

The $<-$ operator can be used for both *assignment* and *replacement* expressions. Give a few examples wherein you would replace an object of a certain type with an object of another type.

Explain how the *dynamic typing* feature of R is evident here.

2.5 Vectors

A ***vector*** is a collection of *indexable* objects.

Vectors are one of the most important topics in R. So, the reader is urged to pay close attention!

2.5.1 Generating Sequences

The : operator

Before, we start computing on collection of objects i.e *vectorizing*, we should know how to generate a collection of objects (*vectors*).

```
1:10 #generate a sequence of numbers from 1 to 10

##  [1]  1  2  3  4  5  6  7  8  9 10

10:1 #generate a sequence in reverse order

##  [1] 10  9  8  7  6  5  4  3  2  1
```

Now, if we wish to generate a sequence of numbers, but with any step size other than 1, we can do the following

```
seq(1,10,by=2) #generate a sequence of numbers with a step size of 2

## [1] 1 3 5 7 9
```

Try generating

- the sequence in reverse order with a step size of 2
- a sequence of all even numbers between 1 and 100
- a sequence of numbers between 0 and 1 with a step size of 0.1

2.5.2 Replicating Numbers

A number can be generated multiple times using the *rep*() function. The f*irst argument is the number* to be replicated while the *second argument is the number of times* it has to be replicated.

```
rep(2,10) ## replicating 2 ten times


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

2.5.3 Vectors and their Type

If we assign the generated sequence (which is a collection of objects) to another object, the new object is a ***vector***

```
x<-seq(1,10,by=2) ##generate a sequence from 1 to 10 with step
##size of 2 and assign it to an object x.
x


## [1] 1 3 5 7 9
```

Get the type of x. Explain your observation

Generate a vector of 10 element where each element has type logical.Moreover, the vector should be an alternating sequence of TRUE/FALSE. You are only allowed to use *seq*() and coercion

Ind order to create a vector of strings we use the *c*() function

```
y<- c("Apple","Orange","Pear") ##create a vector of characters
y


## [1] "Apple"  "Orange" "Pear"
```

Generate a vector where elements are a mix of type

- logical and character. What is the type of the resulting vector?
- logical and numeric. What is the type of the resulting vector?
- character and numeric. What is the type of the resulting vector?

Generate the sequence Apple_1 Apple_2 Apple_3 ... Apple_10

2.5.4 Length of a Vector

The *length*() function

Once we have selected an attribute, we can count the number of elements in it using the *length*() function and passing the attribute object as an argument to it. This function is synonymous to using $COUNT*$ in your SQL query.

```
x <- seq(1,10) ##create a vector
length(x) ##find the length of the vector


## [1] 10
```

2.5.5 Distinct Elements of a Vector

The *unique*() function

We can get the list of unique elements belonging to an attribute by using the *unique* function. This function is synonymous to using the $DISTINCT()$ function in your SQL query

```
x<- rep(2,5) ##create a vector containing the number 2 , five times
unique(x) ##find unique element in the vector


## [1] 2
```

2.5.6 Sum of a Vector

The *sum*() function

Elements of a numeric vector can be easily summed up by passing it as an argument to the *sum*() function

```
x<-1:10 ##create a vector
sum(x) ## get the sum of elements in the vector


## [1] 55
```

The cumulative sum of elements in a numeric vector can be computed by passing it as an argument to the *cumsum*() function

```
x<-1:10 ##create a vector
cumsum(x) ##get the cumulative/running sum of the vector


##  [1]  1  3  6 10 15 21 28 36 45 55
```

2.5.7 Coercing Vectors

```
x<-1:10 ## create a vector of 10 numbers
x


##  [1]  1  2  3  4  5  6  7  8  9 10


class(x) ## check the type of vector X


## [1] "integer"


as.character(x) ## coerce the numeric vector to character


##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"


as.numeric(as.character(x)) ##coerce a vector of characters to numeric


##  [1]  1  2  3  4  5  6  7  8  9 10


y<-as.logical(as.character(x)) ##coerce a vector of characters to logicals
```

Look at the rule for coercion presented in Coercion and explain each of the above examples.

What does the vector y contain? As shown in the last example, for invalid coercion, R does not throw an error. On the contrary, it generates NA which signifies missing values. On large data sets, while doing multiple coercion and assigning them to objects, it is very easy to make this mistake and it does happen often!.

2.5.8 Indexing Vectors

The [] operator allows us to *retrieve an element or a subset of elements from a vector* by specifying an index or a vector of indices. Moreover, objects can be passed to the [] operator thereby allowing for functions and comparison operators to be used in [].

```
X<-seq(1,1.1,by=0.01) ##A generation and assignment  expression
X[1:5] ##get the first 5 elements of the vector X

## [1] 1.00 1.01 1.02 1.03 1.04

X[10:5] ##get the last five element of vector X BUT in

## [1] 1.09 1.08 1.07 1.06 1.05 1.04

##reversed order
X[c(TRUE,FALSE)] ##get elements corresponding to all odd indexes

## [1] 1.00 1.02 1.04 1.06 1.08 1.10

X[c(FALSE,TRUE)] ##get elements corresponding to all even indexes

## [1] 1.01 1.03 1.05 1.07 1.09

X[c(1,5)] ##get the 1st and 5th element of X

## [1] 1.00 1.04

X[-5]  ##get everything EXCEPT the 5th element

##  [1] 1.00 1.01 1.02 1.03 1.05 1.06 1.07 1.08 1.09 1.10

X[-c(1,5)] ## get everything EXCEPT the 1st and the 5th elements

## [1] 1.01 1.02 1.03 1.05 1.06 1.07 1.08 1.09 1.10
```

For the sequence generated in the example try the following

- Pass an index to the [] that is greater than the length of X. As you can see, instead of complaining, R just returns NA. Now if you were assigning the returned values to another vector, this would go completely unnoticed until much later in your data analysis pipeline wherein you start seeing erroneous results.
- Assign a value to a vector index greater than the size of the vector length. Again you will notice that R does not complain (it should!) and just extends the vector until the new index and fills NA in between. In once is not careful, this can lead to a lot of problems!
- Pass NA as an index!!! Again R does not complain (as it should!) but just returns NA for every element of the vector

```
X<-seq(1,1.05,by=0.01)
X

## [1] 1.00 1.01 1.02 1.03 1.04 1.05

X[8]

## [1] NA

X[8]<-5
X

## [1] 1.00 1.01 1.02 1.03 1.04 1.05   NA 5.00

X[NA]

## [1] NA NA NA NA NA NA NA NA
```

The Fibonacci series is given by a sequence of the form $1, 1, 2, 3, 5, 8, ...$ Create a vector of length 2 and use the techniques you have learnt until now to generate the Fibonacci series. Also, you are allowed to create only one vector.

For any arbitrary length vector of 2's, append an arbitrary length vector of 3's

2.5.9 Vectorizing

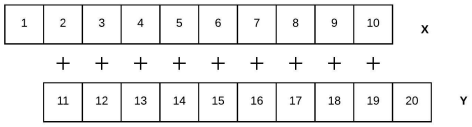
Computing means applying an operator or a set of operators, on objects.

Vectorizing means *computing* on a collection of objects (*vectors*) instead of computing on each object. Over the history of R, there has been a lot of discussion of what is variously called "avoiding loops", "vectorizing computations", or "whole-object computations", in order to improve the efficiency of computations. Operators, when applied to vectors means that the operator is applied to each pair of object in both vectors. However, *the element wise operation is taken care of by the low level implementation of R*, and is conveniently hidden from the user.

Vectorizing with Arithmetic Operators

```
options(digits = 3)
X<-seq(1,10) ## generate a sequence of numbers from 1 to 10
Y<-seq(11,20) ## generate a sequence of numbers from 11 to 20
X+Y  ## Addition of two vectors

##  [1] 12 14 16 18 20 22 24 26 28 30

X-Y  ## Difference between two vectors

##  [1] -10 -10 -10 -10 -10 -10 -10 -10 -10 -10

X*Y  ## Multiplication of two vectors

##  [1]  11  24  39  56  75  96 119 144 171 200

X/Y  ## Division of two vectors

##  [1] 0.0909 0.1667 0.2308 0.2857 0.3333 0.3750 0.4118 0.4444 0.4737 0.5000

exp<-rep(2,10) ##create a vector of 10 two's
X^exp  ## Squaring a set of numbers

##  [1]   1   4   9  16  25  36  49  64  81 100

X%%Y  ## Finding modulus of two vectors

##  [1]  1  2  3  4  5  6  7  8  9 10
```

Try adding the vectors X and Y as shown in the figure



Note that you are allowed to use only one arithmetic operator

Given vector X and Y, compute X^2 without using the $\wedge$ operator

Vectorizing with Comparison Operators

```
X<-seq(1,10) ## generate a sequence of numbers from 1 to 10
Y<-seq(11,20) ## generate a sequence of numbers from 11 to 20
X > Y    ##check if X is greater than Y


##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE


Y[5]<-X[5] ##set the 5th element of Y equal to the 5th element of X
X == Y ##check for equality between X and Y


##  [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
```

What is the return type of applying comparison operators on vectors

Are All Values True?

```
X<-seq(1,10) ## generate a sequence of numbers from 1 to 10
Y<-seq(11,20) ## generate a sequence of numbers from 11 to 20
all(Y > X)


## [1] TRUE


Y[5]<-X[5] ##set the 5th element of Y equal to the 5th element of X
all(Y == X)


## [1] FALSE
```

Which Values Are True?

```
X<-seq(1,10) ## generate a sequence of numbers from 1 to 10
Y<-seq(11,20) ## generate a sequence of numbers from 11 to 20
which(Y > X)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
Y[5]<-X[5] ##set the 5th element of Y equal to the 5th element of X
which(Y == X)
```

```
## [1] 5
```

```
X[X==Y] ##return the element of X that is equal to Y.
```

```
## [1] 5
```

Using one arithmetic operation on Y, transform Y in a way such that $all(X == Y)$ returns TRUE

Using one arithmetic operation on Y, transform Y in a way such that $all(X < Y)$ returns TRUE

Transform only the values of Y that are even numbers, in such a way that they are all less than X. Then compare them with their corresponding values in X to verify if they are less. Try to do this in as few steps as possible.

2.6 Data Frames

2.6.1 Data sets in R

R comes packaged with a list of data sets which can be viewed with the following command:

```
library(help = "datasets")
```

These data sets are a great starting point for new comers to try out different R expressions on, and getting your hands dirty.

2.6.2 View a data set

The *View*() function

The *View*() function can be used to see the contents of a data set wherein the argument to the function is the object that contains the data.

```
View(iris) ##see the contents of a data frame
```

2.6.3 Type of an Data Frame

The *class*() function

We can get the type of an R object by using the *class*() function. This is synonymous to using the *DESC*() function in your SQL query

```
class(iris) ## get the type of a data frame


## [1] "data.frame"
```

2.6.4 Size of a Data Frame

The *dim*() function

For objects that have a tabular structure, the *dim*() function enables us to get the number of rows (observations) and number of columns (attributes)

```
dim(iris) ##get the dimensions (number of rows,number of columns)
```

```
## [1] 150   5
```

Now,try:

```
class(dim(iris)) ##get the class of the object returned by
```

```
## [1] "integer"
```

```
##the call to dim() function
```

Do you see the difference in result when iris is passed as an argument to *class*() v/s when *dim*(*iris*) is passed as an argument to *class*(). Think about what does the output signify?

Did you notice how *class*() takes a data.frame as an argument in one case while takes *dim*(*iris*) in another. Relate this observation to the ideas of objects and encapsulation that we have discussed until now.

Get only the row count of the data set

2.6.5 Attribute of a Data Frame

The $ operator

Attributes of an R object can be accessed by the **$** operator. This operator is synonymous to selecting a column in a spreadsheet or a database table.

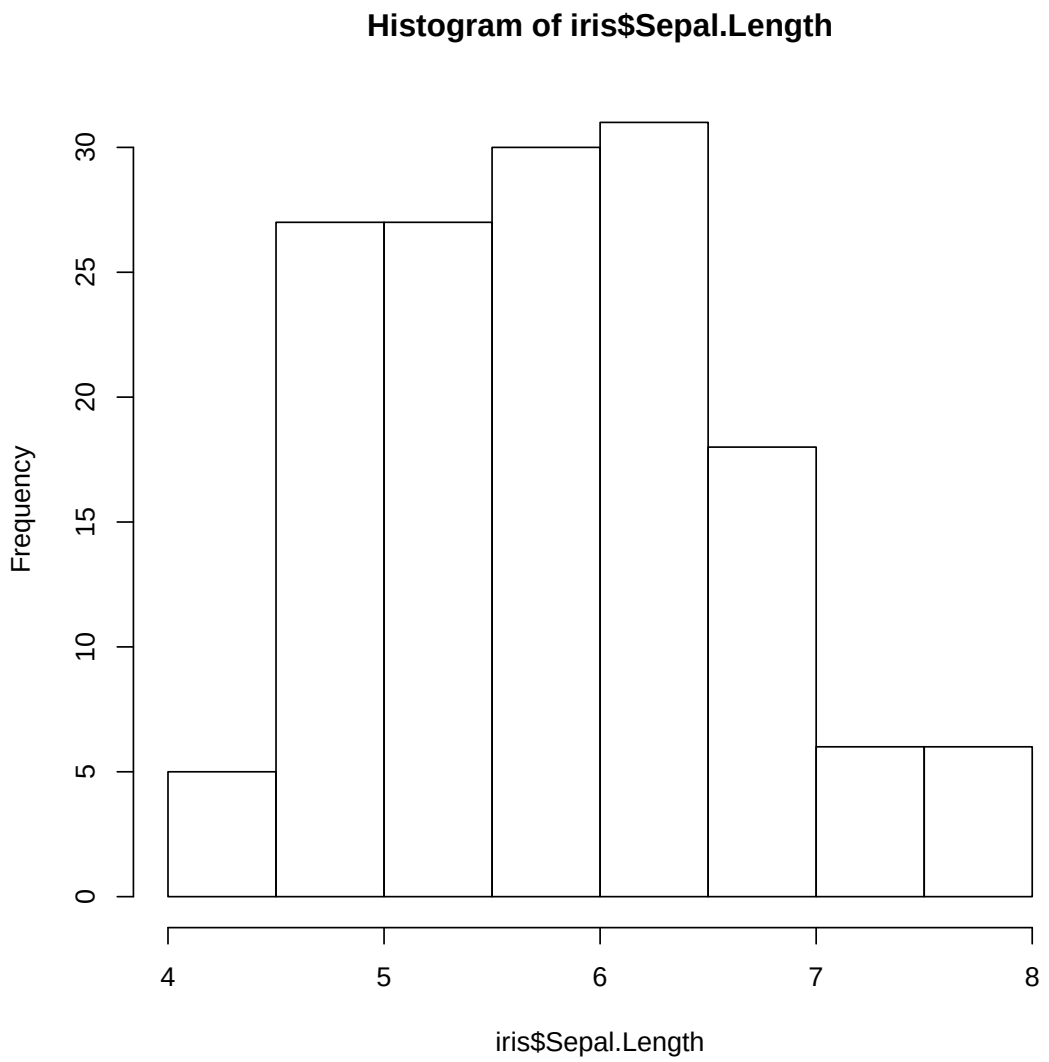
```
resut<-iris$Sepal.Length ##acessing columns of a data frame
```

2.6.6 Create a Histogram

The *hist*() function

A histogram which plots the number of occurrences of distinct elements (***discrete variables***) in the attribute or number of occurrences of distinct intervals (***continuous variables***) in the attribute.

```
hist(iris$Sepal.Length) ##create a histogram of sepal sength which
```




```
##is a vector of the numeric class
```

Is the argument passed to *hist*() a *discrete attribute* or a *continuous attribute*? Explain your choice.

Change the histogram plot to show normalized frequencies

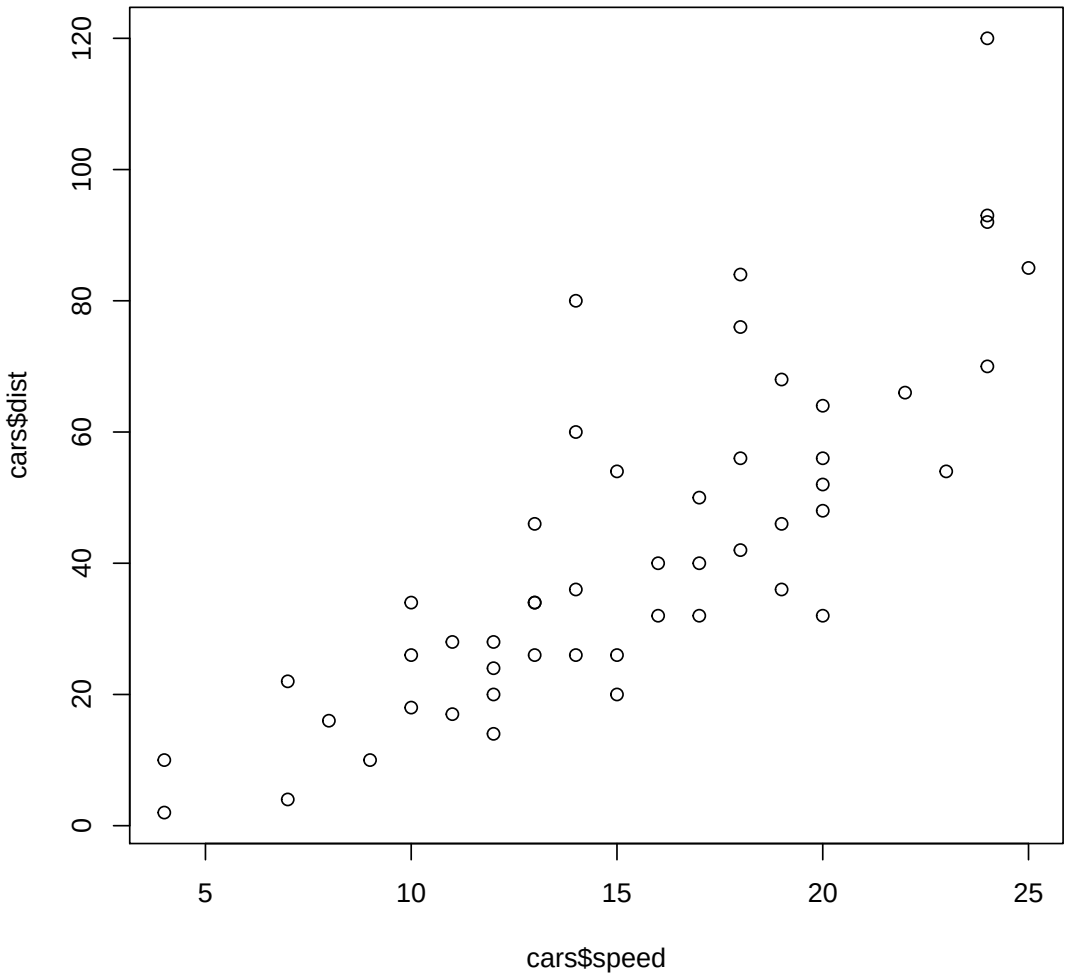
Find the sum of the first 10 and last 10 sepal lengths.

2.6.7 Create a Scatterplot

The *plot*() function

The plot function takes two numeric arrays as arguments and treats each pair of values in the in the arguments as $(X, Y$ coordinates, thereby producing a scatter plot

```
plot(cars$speed,cars$dist) ##create a scatter plot with speed on the x axis
```




```
## and distance on the y axis
```

Can you *explain* the plot that you see above?

Change the markers in the plot from circles to crosses. Moreover change the of the markers

What will you get when you pass a data frame with three attributes as an argument to the *plot*() function

2.6.8 Create a Data Frame

```
X<-data.frame(c(1,2),c("apples","oranges"),
              c(TRUE,FALSE))  ##adding data to a data frame wherein
##each vector represents a column of the deta frame
X


##   c.1..2. c..apples....oranges.. c.TRUE..FALSE.
## 1       1                apples           TRUE
## 2       2               oranges          FALSE


names(X)<-c("id","fruits","has_vitaminC") ##adding attribute names
##to the data frame
X


##   id  fruits has_vitaminC
## 1  1  apples         TRUE
## 2  2 oranges        FALSE


X$fruits


## [1] apples  oranges
## Levels: apples oranges
```

The output for the last expression shows that the attribute fruits has levels. What does this mean?

Pick a data set from the list of available data sets and *explore it* using the *R expressions* you have just learnt.

Create a data frame for the example shown in section 1.1.2 and generate the plot that was shown in the same section.

2.7 Input/Output with External Sources

Reading in data from files like "**.txt**","**.csv**" and "**.xls**" into the R environment has been greatly simplified by RStudio. Just goto Environment tab $->$ Import Dataset and select the file type from which you want to import the data (for *.txt* and *.csv* files choose the "From CSV..." option while for excel sheets choose "From Excel.." option). A new window will open up with a set of options that will guide you to import your data in R. R also provides a set of functions for reading and writing data. An inquisitive reader can read about them in https://www.datacamp.com/community/tutorials/r-tutorial-read-excel-into-r

Though RStudio has simplified the process of reading in data by providing a GUI, it hasn't provided any mechanism for writing data out of the R environment. What do you think this is the reason for this?

Do the following

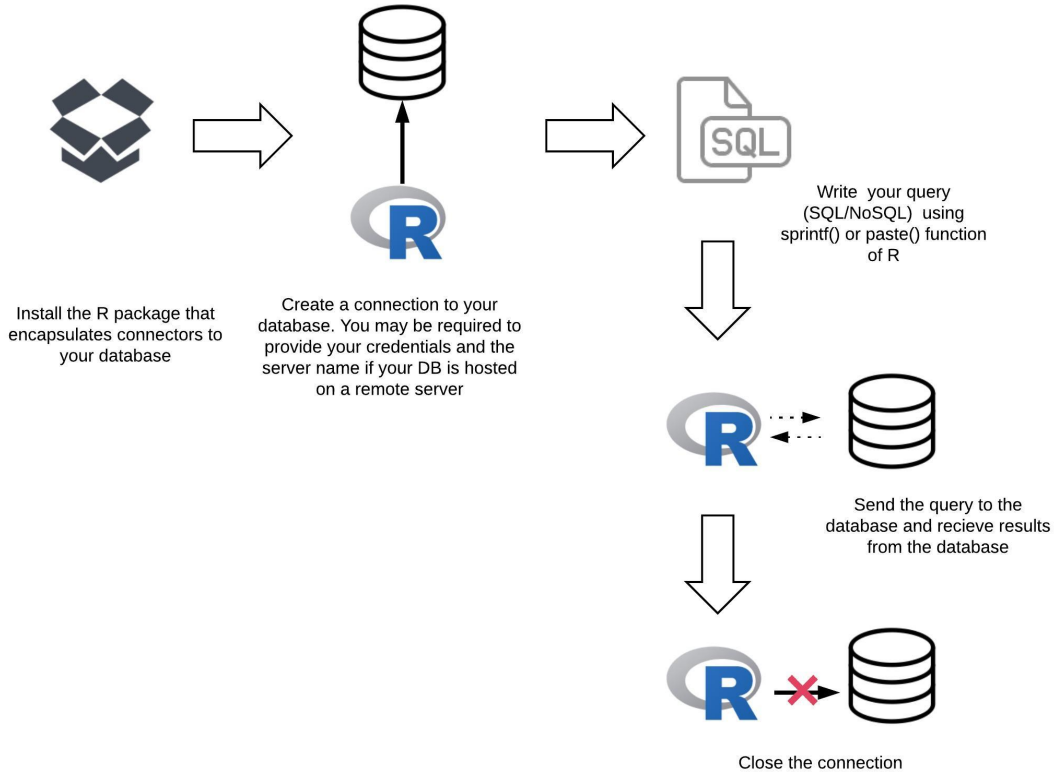
1. Remove the headers from exercise1.csv and try importing it into the R environment. What do you observe?
2. Replace exercise1.csv as exercise1.tsv (i.e change the delimiter between attributes from comma to tab) and try importing it into the R environment.

2.7.1 IO with RDBMS

A large number of different databases exist in the market and each of them has is its own way of setting up and managing connections with client applications. These low level connection details are encapsulated by R packages and the user only has to be aware of an abstract view as shown in the schematic. Details of R packages for connecting to some of the popular databases can be found in the following links:

- How to Connect to an Oracle Database:
  http://rprogramming.net/connect-to-database-in-r/
- How to Connect to a MySQL Database:
  https://www.r-bloggers.com/mysql-and-r/
- How to Connect to MongoDB:
  https://www.r-bloggers.com/r-and-mongodb/

As, shown in the schematic, SQL queries can be constructed with the *sprintf*() and *paste*() function. We give a few examples here:




```
sprintf("select * from names where name in (%s)",
        paste(c("'Andrew","John","Jane'"),collapse="','")
       )


## [1] "select * from names where name in ('Andrew','John','Jane')"


sprintf("insert into networks (id, name) values (%d, '%s');",
                    11, "John"
        )


## [1] "insert into networks (id, name) values (11, 'John');"
```

Yes, it is indeed complicated to write queries in R as compared to writing them in a SQL editor. But the idea is to realize, as a user that one should do complicated aggregations and transformation of the data either solely in the database and store it back in tables where R can query from or within R itself (after having imported the data) but **NEVER** at the interface level. *Complicated interfaces lead to unmanageable software.*When you have large amounts of data, trying to read in all the data of the database into R and then doing the data aggregation and transformation in R is also **NOT** a good idea as it creates a lot of dependency on the connector between R and the database which may have its own limitations based on how it has been implemented.

Chariot
is on boosters !

03

The Ascent

Hey,

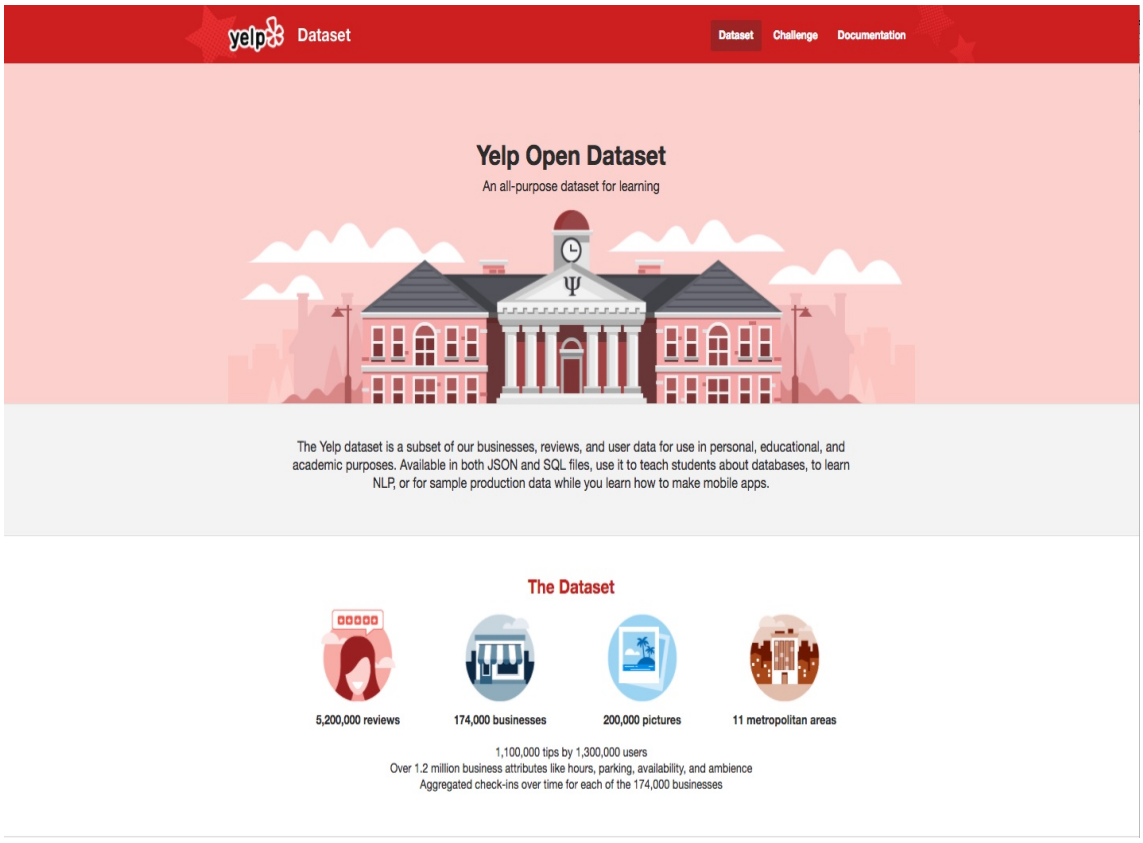
Whats up?
Great work during stage A of Ascent!!
However, the job is only half done!!
Now that you are well aware of your chariot's capabilities, it's time we put them to good use.
We are about to enter a whole new level!!!

Lets Power on!

3. The Ascent - PART B

3.1 The Dataset

Its time to get real!!. We give you the **"Yelp Open Dataset"** . For the rest of our journey, we will be presenting different data manipulation, visualization and modelling techniques using this data set as our example. The idea is to get you acquainted with typically large, and noisy data sets as is prevalent in today's world and to show you how to explore these data sets using R. The data set has been provided to you as set of 11 files, along with this reader. The inter-relation between the attributes can be found at the end of this chapter. Further details of the data set can be found in https://www.yelp.com/dataset



3.2 Data Wrangling

Data Wrangling sometimes referred to as data munging, is the process of transforming and mapping data from its raw form to a form that is suitable for analysis. Exploring data for new insights is a gradual, iterative process. Occasionally we get a sudden insight, but for the most part we try something, look at the results, reflect a bit, and then try something slightly different.

In this section, we will learn about ways to **(a)** Clean **(b)** Transform and **(c)** Aggregate data. Therefore, we will formulate different questions that we wish to ask on the data and write R expressions that will enable us to answer those questions.

3.2.1 The *business* dataset

Import the file ***business.csv*** into your R environment.

Dimension and Attributes

Question: What is the size of the data?

```
dim(business)
```

```
## [1] 156639     12
```

```
##get the dimension of the data
```

Question: What are the attributes?

```
colnames(business)
```

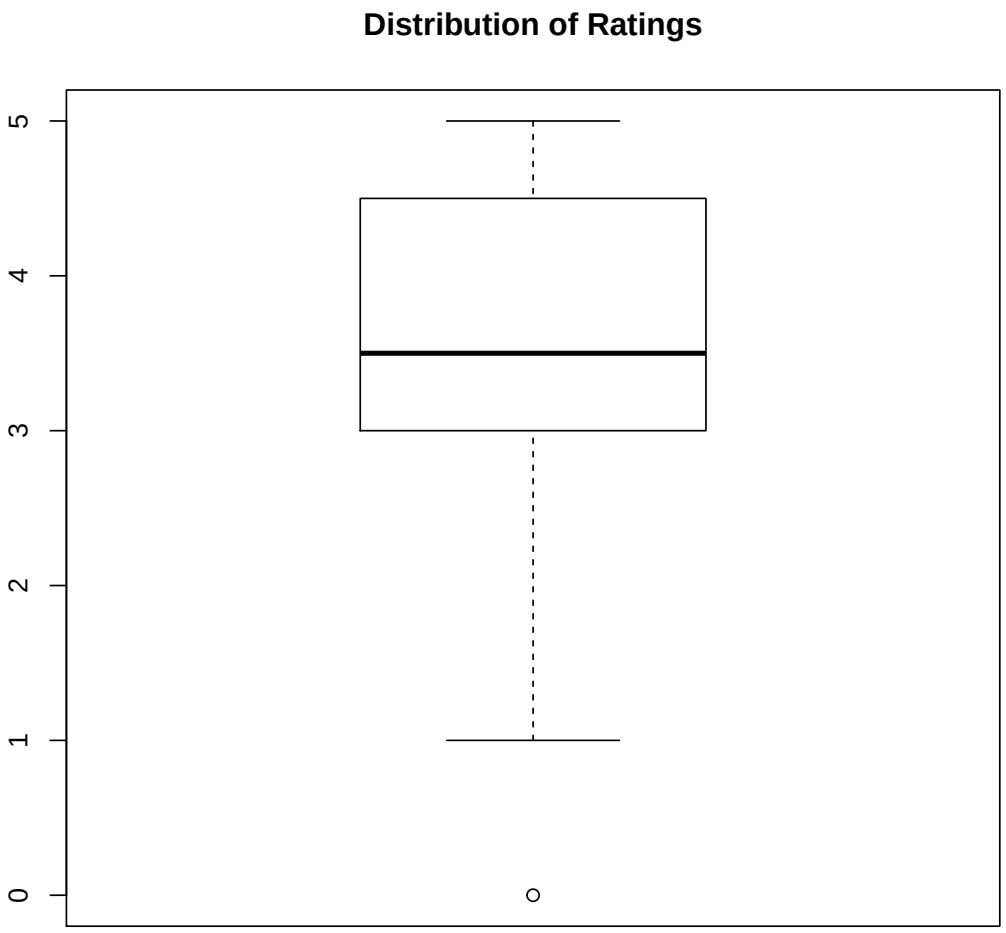
```
##  [1] "id"           "name"         "neighborhood" "address"
##  [5] "city"         "state"        "postal_code"  "latitude"
##  [9] "longitude"    "stars"        "review_count" "is_open"
```

```
##get the attributes of the data
```

Uni-variate Distribution

Question: What is the distribution of ratings in the data?

```
boxplot(business$stars,main = "Distribution of Ratings")
```



Explain the plot

What is the distribution of review count in the data?

Sorting and Selection

Question: Which are the top 10 business by rating?

```
head(business[order(business$stars,
                    decreasing = TRUE
                    ),
                c("name","state","stars")
                ],
     n=10)
## sort the data by number of  stars and get the top 10 businesses
```

Which are the bottom 10 businesses by review count?

Aggregation

Question: What is the number of business entities per state?

```
countPerState <- aggregate(business$state,
                           by=list(business$state),
                           FUN=length
                           )
names(countPerState)<-c("state","count")
View(countPerState)
##count number of occurences of businesses in each state
```

Read the help manual for the *aggregate()* function.

- Why have we passed a *list()* to the **by** argument?
- What do you get if you just set $by = business\$state$?

Take a close look at the values in the **countPerState$state** column. Do you notice any irregularities?

Filtering

Question: How can we filter out incorrect values?

```
View(countPerState[is.na(
                         as.numeric(countPerState$state)
                         ),
                   ]
     )
##filter out the entries that are of type "numeric"
```

Explain how are we taking advantage of coercion in the above expression.

When we count the number of hotels per state, we see that some states are labelled as 'C'. Find the list of all businesses that belong to this state

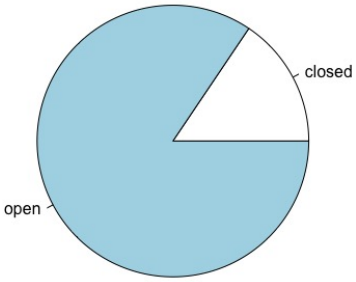
The bounding box of United States in terms of (lat,lon) is (-125.0011, 24.9493) and (-66.9326, 49.5904). Find all businesses that are located in US. Moreover, find which state in US has the highest number of 5 star businesses.

Bi-variate Distribution

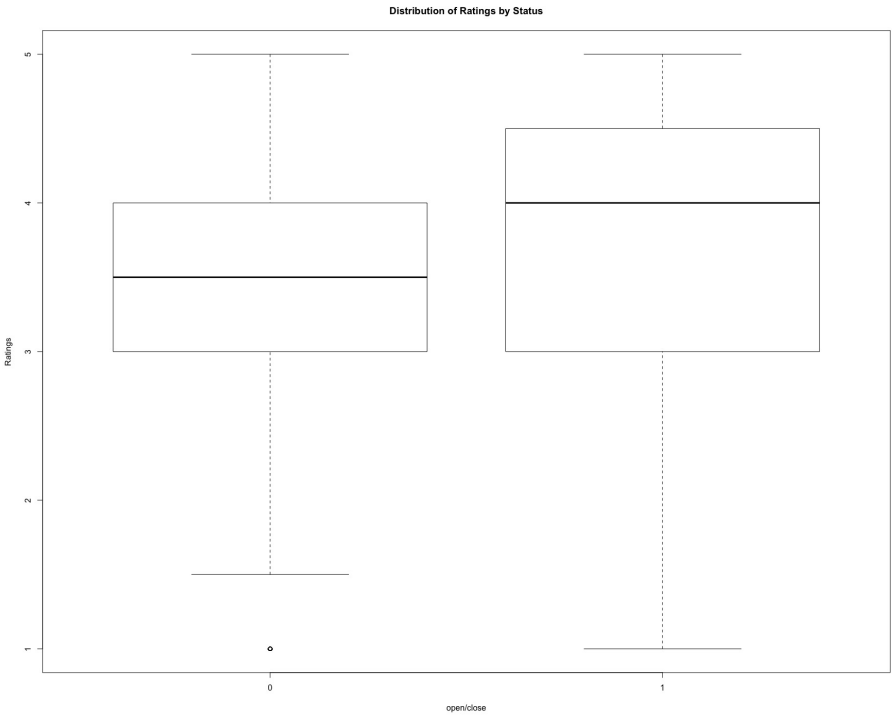
Question: What is the distribution of open v/s closed business?

```
table(business$is_open)
## get frequency of open v/s closed businesses
```

Refer to https://www.statmethods.net/graphs/pie.html and generate the pie chart as shown below



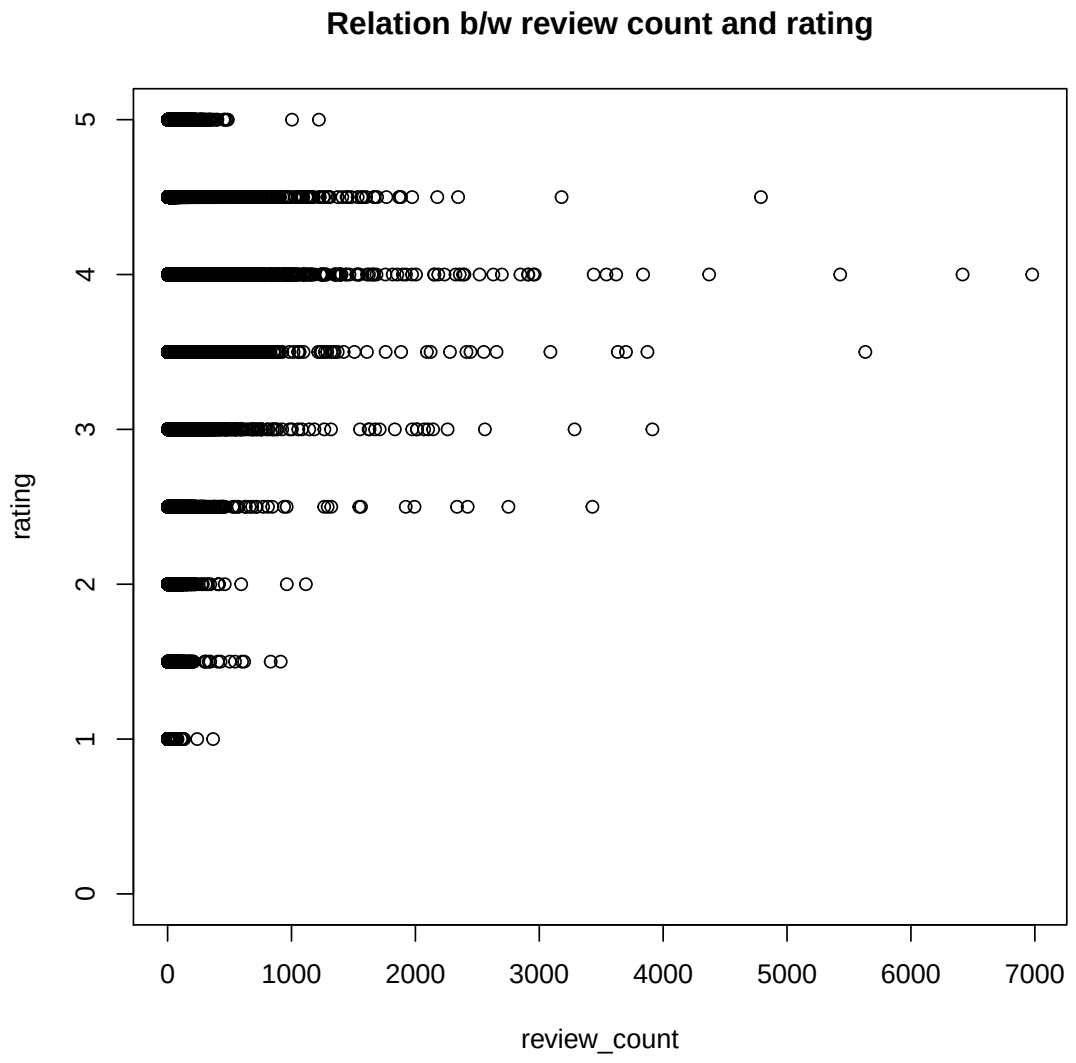
Refer to https://www.statmethods.net/graphs/boxplot.html and generate the boxplot as shown below. Moreover explain the plot



Relationship among Attributes

Question: Does there exist a relationship between ratings and number of reviews?

```
plot(business$review_count,business$stars,
      main = "Relation b/w review count and rating",
     xlab = "review_count", ylab = "rating")
```




```
##scatterplot of review count and ratings
```

Explain the plot

Generate the scatter-plot that plots the relationship between review count and ratings for all open businesses and has number of reviews ≤ 1000.

3.2.2 The *category* dataset

Import the file ***category.csv*** into your R environment.

Dimension and Attributes

Question: What is the dimension of the data and what are the various attributes?

```
dim(category)


## [1] 590290      2


##get dimension of the data
colnames(category)


## [1] "business_id" "category"


##get attributes of the data
```

Sorting and Selection

Question: Which are the 10 most frequent categories?

```
countPerCategory <- aggregate(category$category,
                              by=list(category$category),
                              length
                             )
##count the number of occurence of each category
names(countPerCategory) <- c("category","count")

top10Categories<-head(countPerCategory[
                             order(countPerCategory$count,
                                   decreasing = TRUE
                                   ),
                            ],
         n=10
     )
##sort the frequency data frame in descending order
##and list the top 10
top10Categories
```

Question: Which business belong to the 10 most frequent categories?

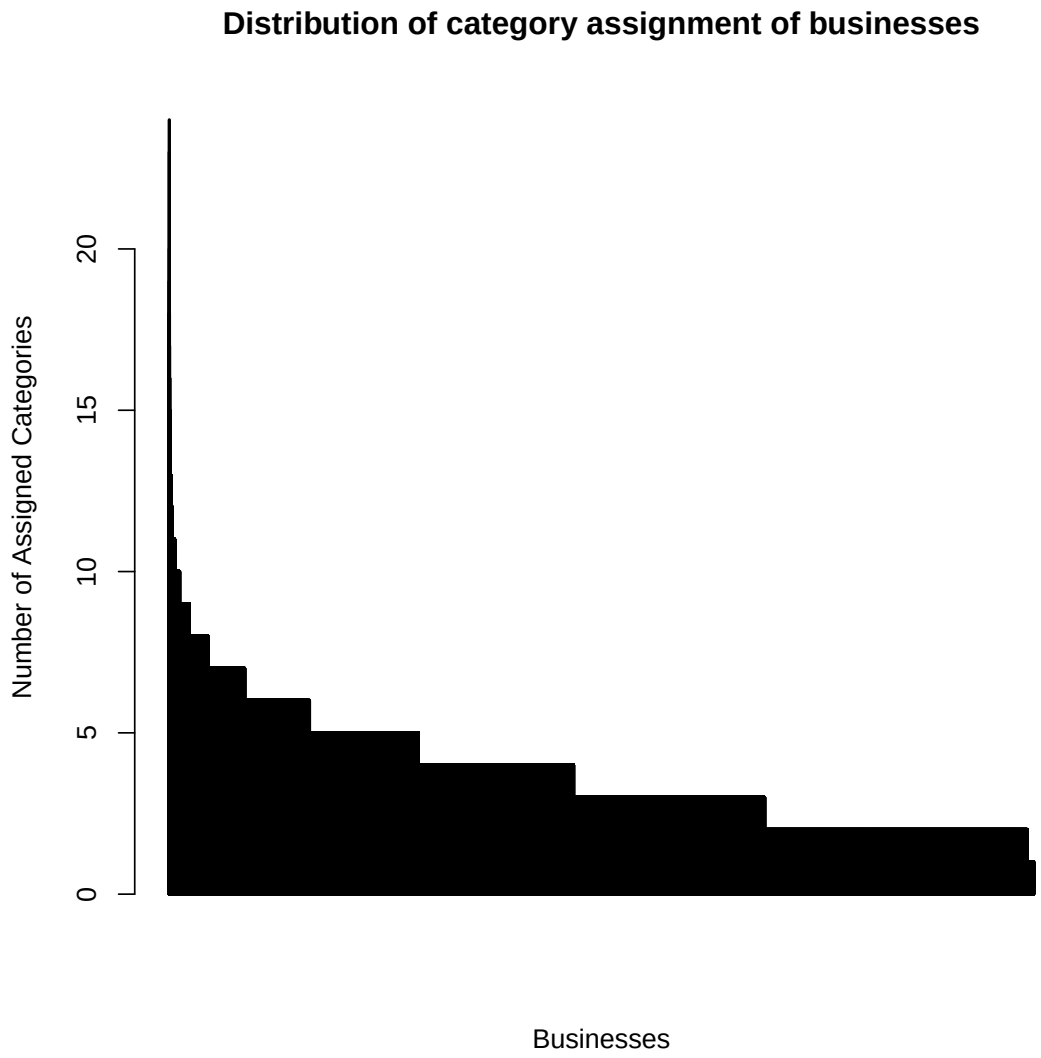
```
businesses_top10Categories <- category[category$category %in%
                                       top10Categories[,1],]
##list all entries in the category file that
#belong to one of the top 10 categories
```

How would you check if the filtering criteria applied in the last step returns the correct result

Distribution of the Data

Question: Are businesses assigned to multiple categories? How many such businesses exist?

```
businessIdFreq <- table(category$business_id)
##get number of categories each business has been assigned to
length(businessIdFreq > 0) > 0


## [1] TRUE


##check if there are businesses assigned to multiple categories
plot(businessIdFreq[order(
                          businessIdFreq,
                          decreasing = TRUE
                          )
                    ],
     xlab = "Businesses",
     ylab="Number of Assigned Categories",
     main = "Distribution of category assignment of businesses",
     xaxt = 'n')
```




```
##sort the frequency of category assignment in descending
##order and plot the same
```

Explain the plot

Joining Data sets

Question: Which business belong to the category "shopping"?

```
shopping <- business[business$id %in%
                         businesses_top10Categories
                             [businesses_top10Categories$category
                                 == "Shopping",
                             ]$business_id,
         ]
```

```
##list all businesses from the business dataframe
##that belong to the category "shopping"
shopping
```

Try using

```
businesses_top10Categories[
          businesses_top10Categories$category
          == "Shopping",
          "business_id"
          ]
```

instead to get the list of businesses labelled with category = "Shopping". Explain the difference

Filter all businesses from the business data frame that belong to one of the top 10 categories

Calculate the average reviews count for businesses belonging to the top 10 categories and compare it against the average review count for businesses belonging to the bottom 10 categories

3.2.3 The *attribute* dataset

Import the file ***attribute.csv*** into your R environment.

Dimension and Attributes

Question: What is the dimension of the data and what are the various attributes?

```
dim(attribute)
```

```
## [1] 1229805       2
```

```
##get dimension of the data
colnames(attribute)
```

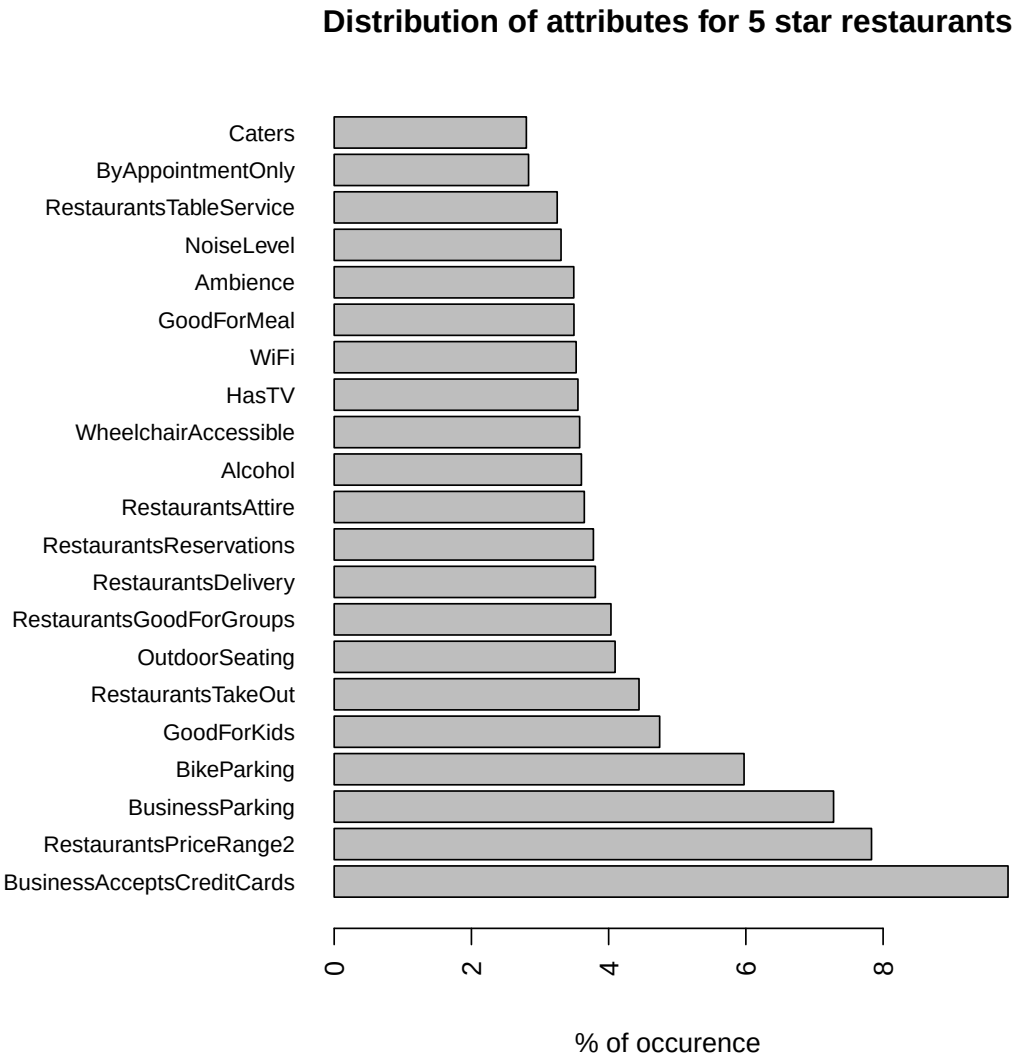
```
## [1] "business_id" "name"
```

```
##get attributes of the data
```

Filtering, Sorting and Joining

Question: Which attributes are prevalent for restaurants with 5 star rating?

```
fiveStarRestaurants<-business[(business$id %in%
                  category[
                           category$category == "Restaurants",
                         ]$business_id
          )
         &
          (business$stars == max(business$stars)),
       ]$id
##get id's of 5 star restaurants

attributeCount <- table(attribute[
                                  fiveStarRestaurants
                                  %in%
                                  attribute$business_id,
                                  "name"]
                        )
##find attributes corresponding to the matching business
##and count their number of occurences

sortedAttributeCount <- attributeCount[order(
                                        attributeCount,
                                        decreasing = TRUE
                                        )
                                      ]
##sort the attributes in descending order of their occurence

sortedAttributePercent <- sortedAttributeCount / sum(sortedAttributeCount) * 100
##normalize the counts

par(las=2)
##make lable text perpendicular to the axis

par(mar=c(5,12,4,2))
##increase y-axis margin

barplot(sortedAttributePercent[sortedAttributePercent > 1],
        horiz=TRUE,
        cex.names = 0.8,
        xlab = "% of occurence",
        main = "Distribution of attributes for 5 star restaurants")
```




```
##plot the distribution of attributes
##that occur at least 1% of the time
```

Pay close attention to the ***par()*** function in the example

Read the help manual of the ***par()*** function

- Which attributes are prevalent for restaurants with 3 star rating? Compare the same with attributes of restaurants that have a rating of 5 stars. Explain your observation
- Do the same analysis for businesses which have category as "Shopping"

3.3 Control Structures

3.3.1 The *checkin* dataset

Import the file ***checkin.csv*** into your R environment.

Dimension and Attributes

Question: What is the dimension of the data and what are the various attributes?

```
dim(checkin)
```

```
## [1] 3738750       3
```

```
##get dimension of the data
colnames(checkin)
```

```
## [1] "business_id" "date"        "count"
```

```
##get attributes of the data
```

Splitting Strings

Question: How can we split the checkin date (DayOfWeek-HH:MM) into "day of week" and "time"?

```
checkinTime <- strsplit(checkin$date,split = "-")
## split the date string into day of week and time
```

Read the help manual of the ***strsplit()*** function

What is the type of the object "checkinTime"?

3.3.2 The *for* loop

for(name in range) body

Iterative Vectorization

Question: How can we create three vectors for "day of week", "hour of day" and "minutes" respectively?

```
numCheckins <- length(checkinTime)
##the length of the list that contains
## lists of the form [[<day of week>,<hour of day>,<minute>],..]
dow <- vector(mode = "character", length = numCheckins)
## a character vector of size = numCheckins
hod <- vector(mode = "integer", length = numCheckins)
## an integer vector of size = numCheckins
minute <- vector(mode = "integer", length = numCheckins)
## an integer vector of size = numCheckins
index = 1 ##set increment counter to 1
for (dayTime in checkinTime){
      ## iterateve over each list in the list
      dow[index]<-dayTime[1]
      ## extract the first element of the list (day of week)
      ## and assign it to the corresponding position in dow
      HH_MM_split <- strsplit(dayTime[2],split = ":")
      ## split the second element
      ## of the list (HH:MM) into HH and MM which in turn
      ##creates a list of lists
      hod[index] <- as.integer(HH_MM_split[[1]][1])
      ##access the first element (hour)
      ## of the list of lists  and assign it to the
      ##corresponding position in hod
      minute[index] <- as.integer(HH_MM_split[[1]][2])
      ##access the second element (minute)
      ## of the list of lists and assign it to the
      ##corresponding position in minute
      index = index + 1 ##increment the value of index
}
index = 1 ##reset index to 1
```

- Explain the meaning of the [[]] in the context of the example
- What happens if we set the mode of the vectors "hod" and "minute" to ***numeric***?

- Install the package **rbenchmark** and read the help manual for the ***benchmark()*** function.
- Try the above example without initializing the length/type of the vector. Benchmark the same against the given example. Explain why do you see a difference in the execution time
- Do not reset the index value to 1 at the end of the example and run the code block twice. Observe the size of the vectors "dow", "hod" and "minute". Explain your observation.

3.3.3 The *if* statement

if(test) expression1 else expression2

Conditional Assignment

Question: How can we map days of week to their position in the week? e.g. Sunday = 0, Monday = 1, ...

```
pow <- vector(mode = "integer", length = length(dow))
## create a vector of type integer where the length is
## equal to the number of elements in the dow
index = 1 ## set the increment counter to 1
for(item in dow){ ##iterate over element of the the
  ## day of week vector
       if (item == "Sunday"){
         pow[index] <- 0 ## if the day of week is sunday assign
         ## value 0 to the corresponding position in the pow vector
      }
      if (item == "Monday"){
         pow[index] <- 1 ## if the day of week is monday assign
         ## value 1 to the corresponding position in the pow vector
      }
      else if (item == "Tuesday"){
         pow[index] <- 2 ## if the day of week is tuesday assign
         ## value 2 to the corresponding position in the pow vector
      }
      else if (item == "Wednesday"){
         pow[index] <- 3 ## if the day of week is wednesday assign
         ## value 3 to the corresponding position in the pow vector
      }
      else if (item == "Thursday"){
         pow[index] <- 4 ## if the day of week is thursday assign
```

```
        ## value 4 to the corresponding position in the pow vector
      }
    else if (item == "Friday"){
        pow[index] <- 5 ## if the day of week is friday assign
        ## value 5 to the corresponding position in the pow vector
    }
    else if (item == "Saturday"){
        pow[index] <- 6 ## if the day of week is saturday assign
        ## value 6 to the corresponding position in the pow vector
    }
    index = index + 1 ##increment the value of index
}
index = 1  ##reset index to 1
```

- How can you achieve the same result without using the *for loop* and *if statement* at all! Hint: try the ***which()*** function. Benchmark your implementation (without the control statements) against the example. Do you notice a difference in performance? Explain your observation.
- Create a data frame named *checkins* that has the following attributes

```
colnames(checkins)
```

```
## [1] "business_id"      "day_of_week"      "position_of_week"
## [4] "hour"             "minute"           "count"
```

Validation

Question: How can we check for correct assignment of position to day of week?

```
table(checkins$day_of_week,checkins$position_of_week)
```

```
##
##                 0      1      2      3      4      5      6
##   Friday        0      0      0      0      0 558419      0
##   Monday        0 478652      0      0      0      0      0
##   Saturday      0      0      0      0      0      0 624409
##   Sunday   542932      0      0      0      0      0      0
##   Thursday      0      0      0      0 520651      0      0
##   Tuesday       0      0 499441      0      0      0      0
##   Wednesday     0      0      0 514246      0      0      0
```

```
##co-occurence count of day of week and position
```

```
sum(table(checkins$day_of_week,checkins$position_of_week)) ==
  dim(checkins)[1]


## [1] TRUE


##check if every day of week has been mapped to one position
```

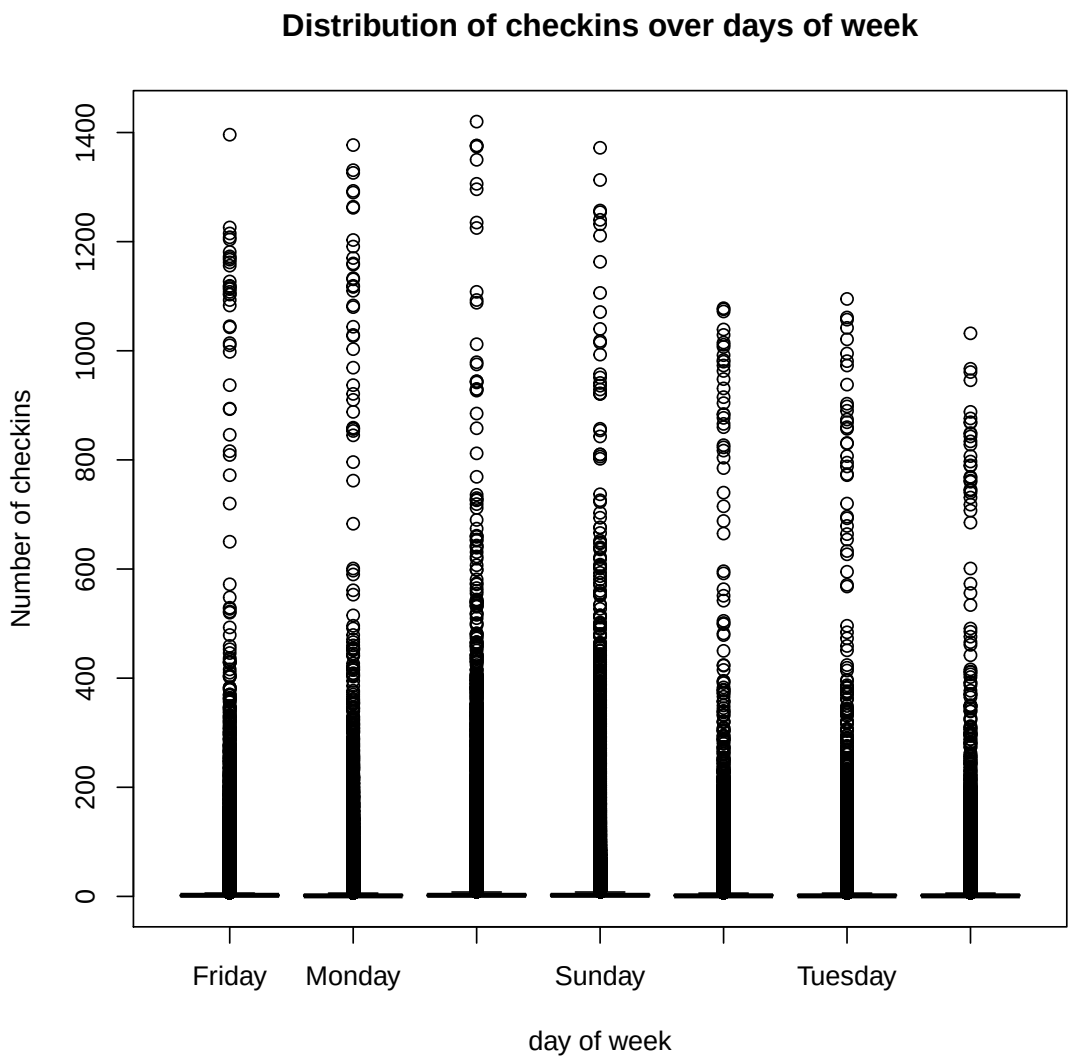
Explain the output of the code snippet

Compute the total number of checkins aggregated by hour and day of week. Plot the same

Conditional Distribution

Question: What is the distribution of checkins over each day of the week?

```
summary(checkins$count)


##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1       1       2       4       3    1420


##generate summary statistic for checkin count
boxplot(checkins$count ~ checkins$day_of_week,
        main = "Distribution of checkins over days of week",
        xlab = "day of week",ylab = "Number of checkins")
```



```
##distribution of count , given, day of week
```

Explain the plot

- Regenerate the plot but only for checkins $\geq$ 3rd Quartile
- Generate a plot that shows the relationship between hour of day and number of checkins. Explain if you notice a relationship between the two

3.4 Functions

function(arguments) body

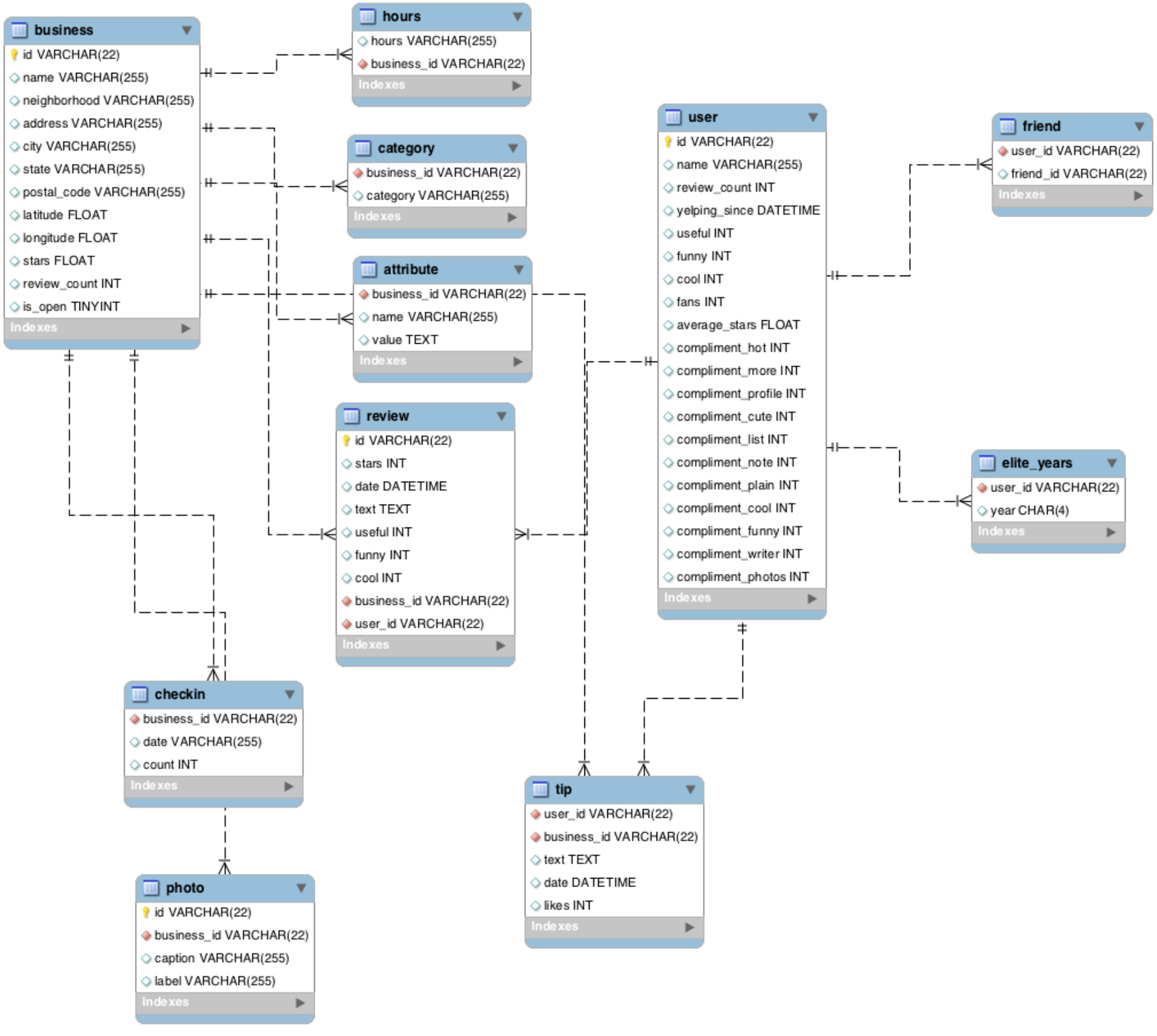
Question: How can we split the "date" attribute into "day of week", "hour of day" and "minutes" respectively?

```
## a generic splitting function that accepts as arguments
## - an input of type characer that is to be split
## - the delimiter for day
## - the delimiter for time
## returns : a list of the form
##           [<day of week>, <hour of day>, <minute>]
getDOW_HOD <- function(dateString,dowSplitter = "",timeSplitter = ""){
            splitValues <- strsplit(dateString,split = dowSplitter)
            dow <- splitValues[[1]][1]
            HH_MM_split <- strsplit(splitValues[[1]][2],
                                    split = timeSplitter)
            hod <- as.numeric(HH_MM_split[[1]][1])
            mod <- as.numeric(HH_MM_split[[1]][2])
            return(list(dayOfWeek = dow,
                        hourOfDay = hod,
                        minute = mod))
}

## for every elment in checkin\£date apply the
## function getDOW_HOD
result <- lapply(checkin$date,
                 getDOW_HOD,
                 dowSplitter = '-',timeSplitter = ':'
                )
```

Read the help manual for ***lapply()***. Try using ***apply()*** instead of ***lapply()***.

The reader is urged to work on the following exercise!

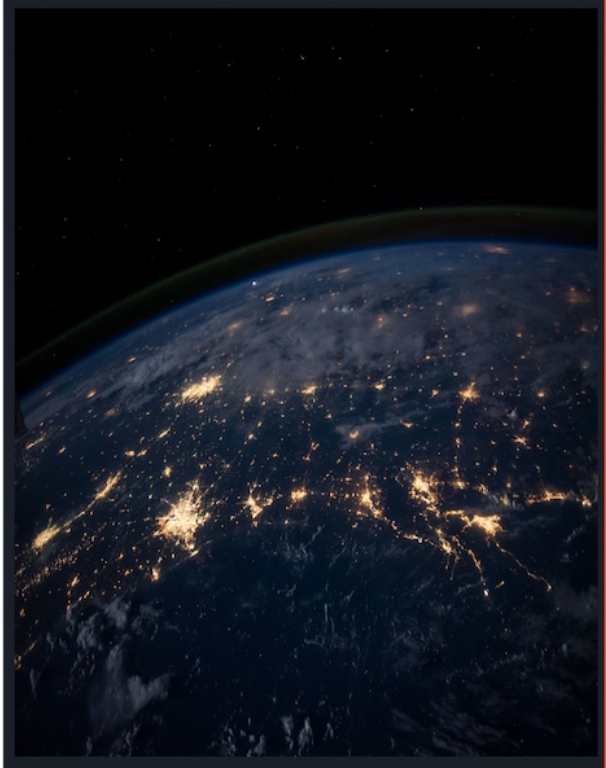
Write a R function that: Given a category and time of week e.g (weekday/weekend),(morning/evening), the function should compute the number of checkins per state. Moreover, the function should also check if there exists a relationship between the rating of a business and the number of checkins The function should also generate plots that visualize the results.


business
id VARCHAR(22)
name VARCHAR(255)
neighborhood VARCHAR(255)
address VARCHAR(255)
city VARCHAR(255)
state VARCHAR(255)
postal_code VARCHAR(255)
latitude FLOAT
longitude FLOAT
stars FLOAT
review_count INT
is_open TINYINT
Indexes
hours
hours VARCHAR(255)
business_id VARCHAR(22)
Indexes
category
business_id VARCHAR(22)
category VARCHAR(255)
Indexes
attribute
business_id VARCHAR(22)
name VARCHAR(255)
value TEXT
Indexes
review
id VARCHAR(22)
stars INT
date DATETIME
text TEXT
useful INT
funny INT
cool INT
business_id VARCHAR(22)
user_id VARCHAR(22)
Indexes
checkin
business_id VARCHAR(22)
date VARCHAR(255)
count INT
Indexes
photo
id VARCHAR(22)
business_id VARCHAR(22)
caption VARCHAR(255)
label VARCHAR(255)
Indexes
user
id VARCHAR(22)
name VARCHAR(255)
review_count INT
yelping_since DATETIME
useful INT
funny INT
cool INT
fans INT
average_stars FLOAT
compliment_hot INT
compliment_more INT
compliment_profile INT
compliment_cute INT
compliment_list INT
compliment_note INT
compliment_plain INT
compliment_cool INT
compliment_funny INT
compliment_writer INT
compliment_photos INT
Indexes
friend
user_id VARCHAR(22)
friend_id VARCHAR(22)
Indexes
elite_years
user_id VARCHAR(22)
year CHAR(4)
Indexes
tip
user_id VARCHAR(22)
business_id VARCHAR(22)
text TEXT
date DATETIME
likes INT
Indexes

Chariot
is in orbit !

04

Orbiting



Hey,

Whats up?

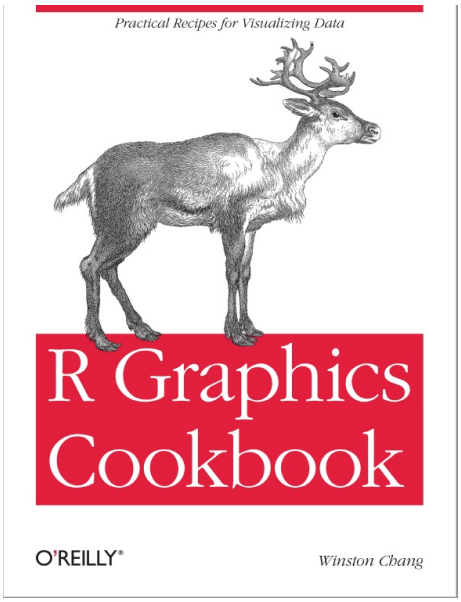
For the first time ever, we start seeing the grander scheme. This is where we use our chariot to explore, to go where no one has gone before. It is time to start taking giant leaps forward !!

4. The Orbiting

4.1 The Grammar of Graphics

The Grammar of Graphics provides a formal, structured perspective on how to describe data graphics. Therefore, the grammar of graphics is a *mapping from properties of data to visual properties of graphics*. The data properties are typically numerical or categorical values, while the visual properties include the x and y position of points, colors of lines, heights of bars and so on.

Over the past years, ***ggplot2*** has become the de-facto package for data visualization in R. In this chapter, we will familiarize the reader with the basic concepts of ggplot2 and show basic data visualization techniques using the same. For a more detailed and thorough treatment please refer to the following book. Moreover, a large number of examples using ggplot2 can be found at `http://r-statistics.co/Complete-Ggplot2-Tutorial-Part1-With-R-Code.html`. Furthermore, a cheatsheet for ggplot2() is available in the RStudio IDE and can be found at the end of this chapter as well.



4.2 The ggplot2 Package

ggplot2 is not bundled with the base packages of R. So the user will have to install ggplot2 before being able to use it.

ggplot2 Terminology

1. ***Data*** is what we want to visualize. It consists of *variables* which are stored as columns in a data frame.
2. ***Geoms*** are the geometric objects that are drawn to represent the data, such as bars, lines, and points.
3. ***Aesthetics*** are visual properties of geoms, such as x and y position, line color, point shapes, etc.
4. ***Mappings*** represent the relationship between data values and aesthetics.
5. ***Scales*** control the mapping from the the values in the data space, to values in the aesthetics space.
6. ***Guides*** show the viewer how to map the visual properties back to the data space. Commonly used guides are tick marks and labels on an axis.

ggplot2 requires the data to be represented as a data frame wherein each variable, that is mapped to an asthetic must be represented as a column in the data frame.

ggplot2 uses the '+' operator to add, each of the components described above, to a visualization. Recall that the '+' operator is a function in R and the operands (in this case ggplot2 functions) are arguments to this function.

The reader is urged to pay close attention to the comments section of the code snippets in this chapter, as the workings of ggplot2 has been explained in those comments.

4.2.1 A Step By Step Guide To Visualization with ggplot2

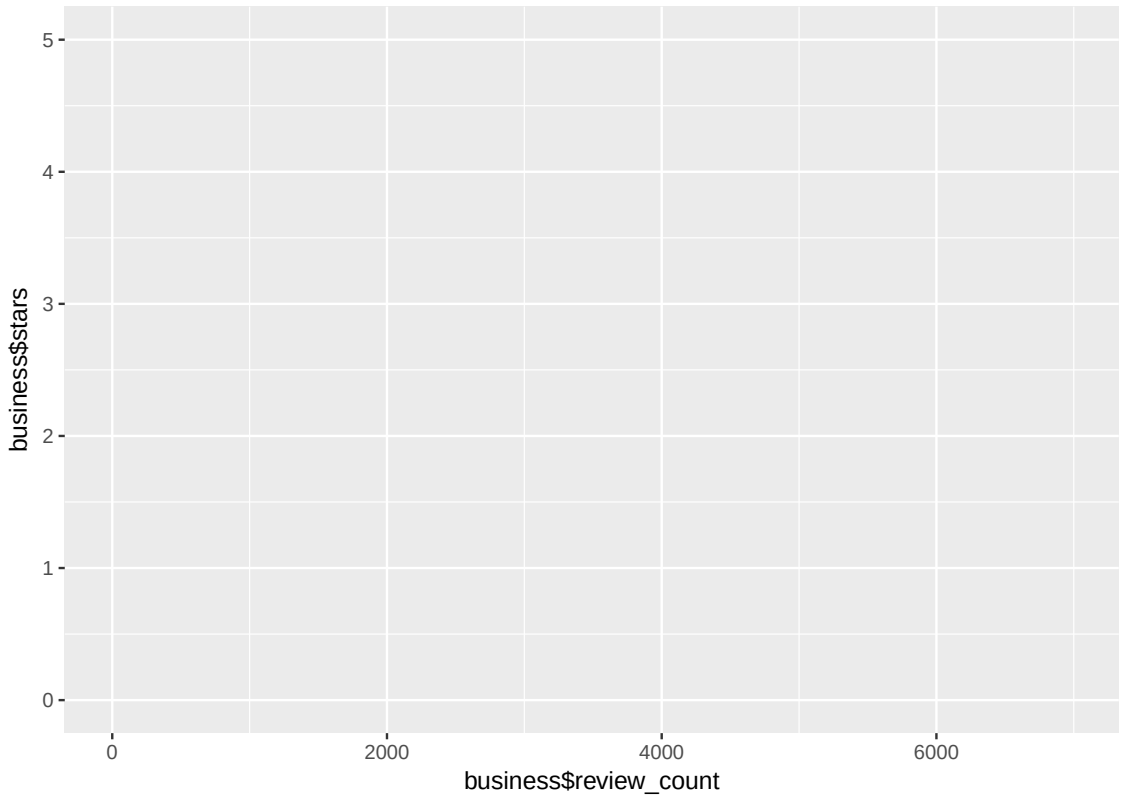
Step 1: Add the *data frame* that you wish to visualize

```
graph <- ggplot(business)
##pass data frame to ggplot2
graph
```


Passing a data frame to the ggplot() function initializes a ggplot visualization without any axis or points i.e without any aesthetics. The next step is to add aesthetics to the plot.

Step 2: Set the *Aesthetics* of the visualization

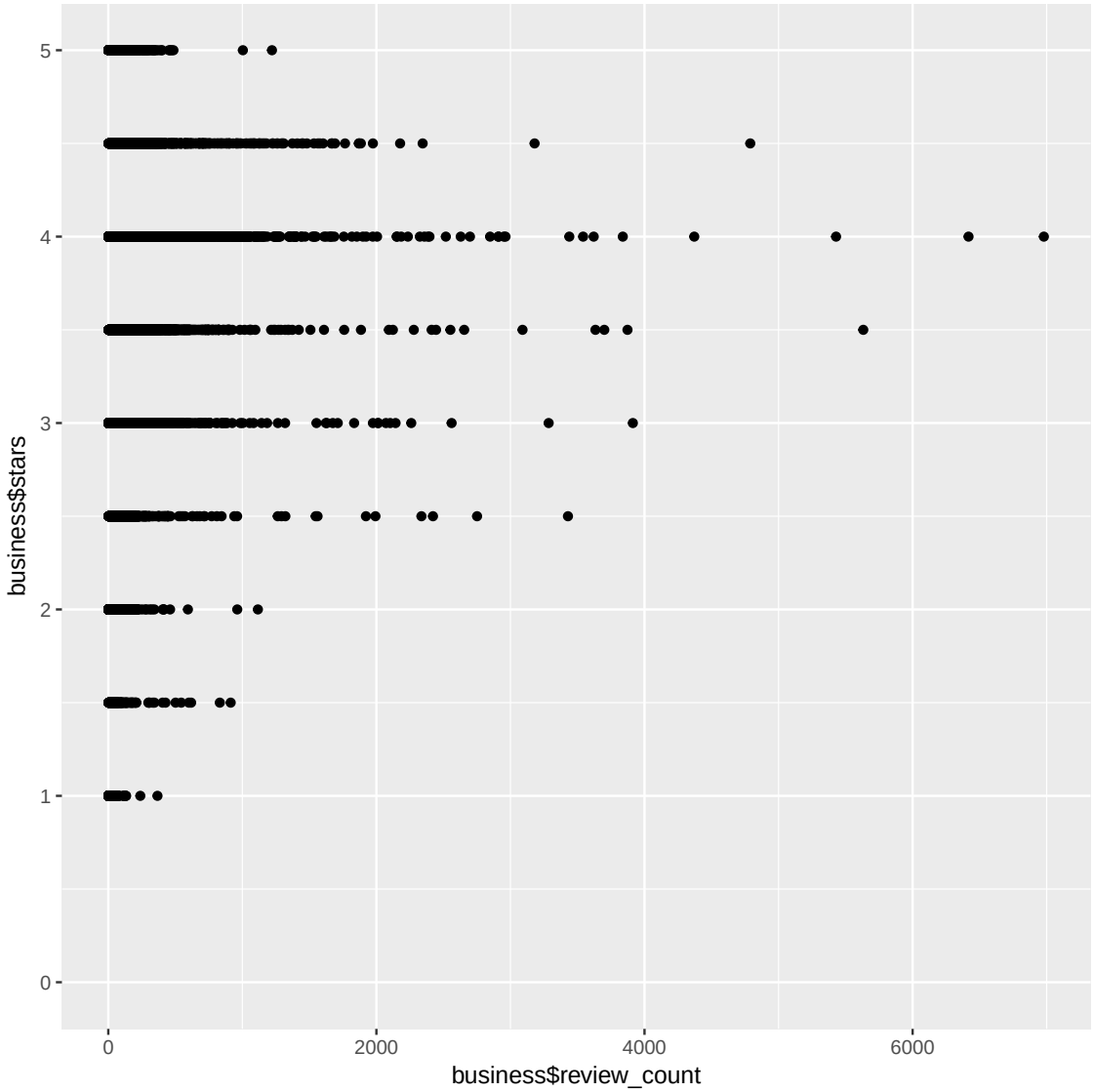
```
graph <- graph +
aes( x = business$review_count,
     y = business$stars
)
##set the aesthetics of the x and y axis by mapping
## review count to the x axis and stars to the y axis
graph
```



Adding aesthetics to the plot results in addition of coordinate axis to the plot. As in the case of base graphics in R, one may need to create both or at least one axis using aesthetics. Note, that the argument passed to the *aes()* function defines a *mapping* from the columns in the data frame to its corresponding visual representation. Moreover, the axis labels are set to the column names by default. We will change this in the next steps.

Step 3: Add *Geometry* to the visualization

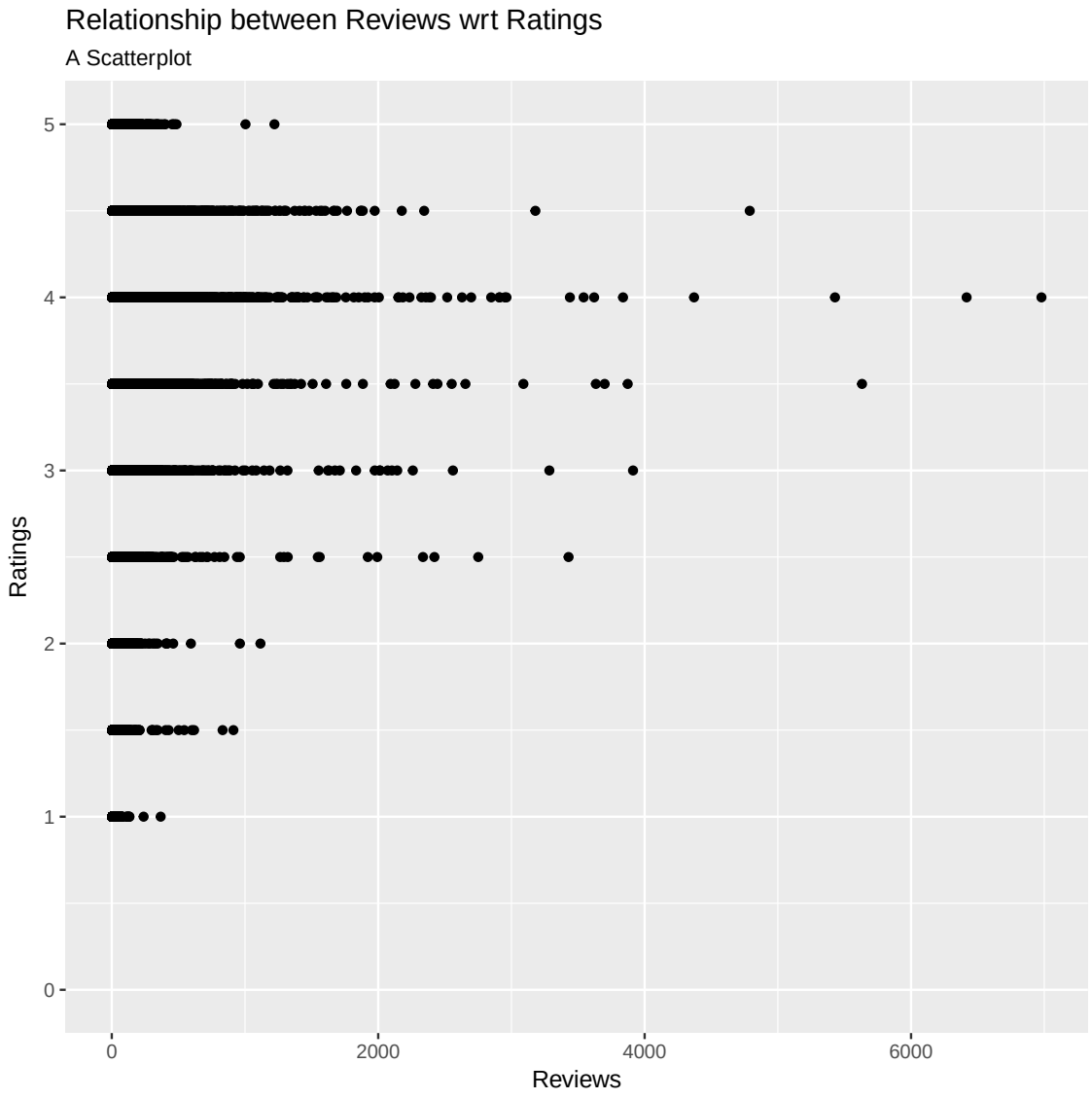
```
graph <- graph + geom_point()
## add geometry and more specifically tell ggplot
## to add points for (x,y) values
graph
```



Adding geometry, instructs the visualization system to generate a particular type of plot. For example *geom_point()* would generate a scatter plot, *geom_bar()* would generate a bar plot, *geom_hist()* would generate a histogram etc.

Step 4: Add *Guides* to the visualization

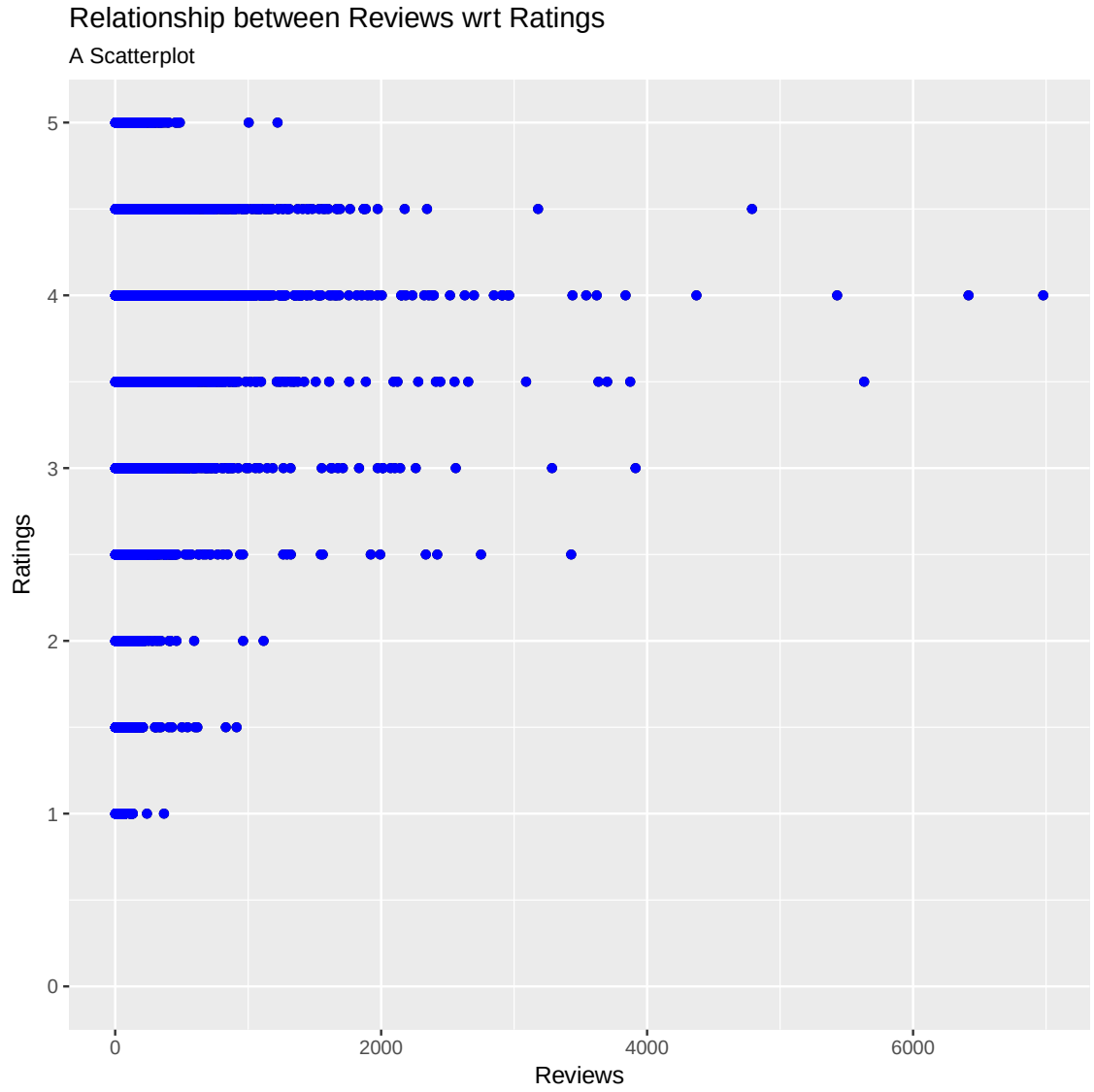
```
graph <- graph +
ggtitle( "Relationship between Reviews wrt Ratings",
           subtitle = "A Scatterplot" ) +
xlab("Reviews") + ylab("Ratings")
## add title and subtitle to the plot
## add labels to the axis
graph
```



Adding guides to the visualization, implies adding a title (Note the *ggtitle()* function), labels (Note the *xlab()* and *ylab()* functions) and legends (Using the *labs()* function) to the plot.

Step 5: Set *Aesthetics* to the *Geoms*

```
graph <- graph + geom_point(colour = "blue")
## set the colour of the points in the plot
graph
```

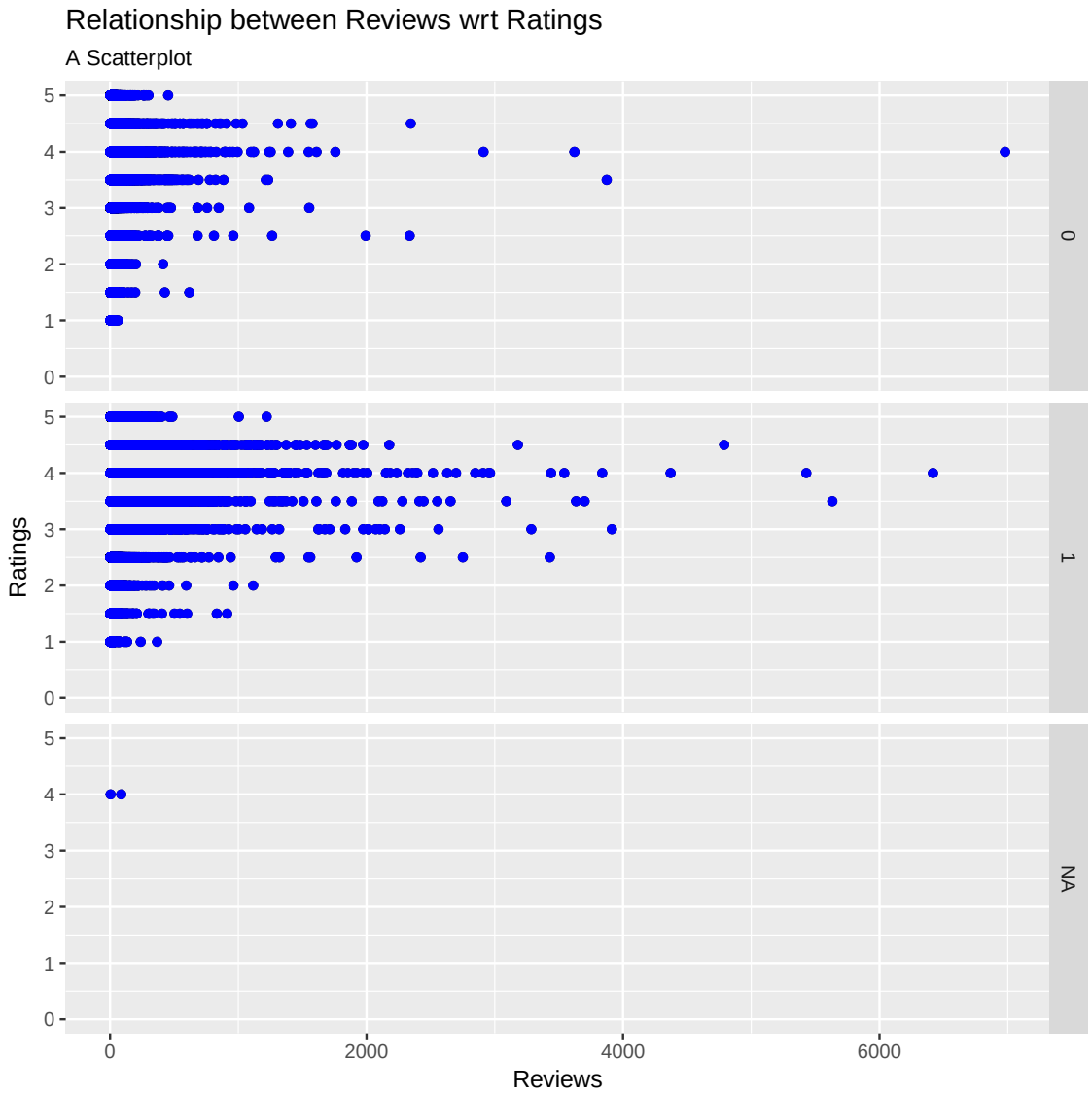


ggplot allows us to set the aesthetics of the *geoms* as shown here. However, there is a difference between **aesthetics setting** and **aesthetics mapping**. What is demonstrated here is an example of aesthetics setting

Pass aes(colour = "blue") as an argument to *geom_point()*. What do you observe?

Step 6: Add *Facets* to the visualization

```
graph + facet_grid( business$is_open ~ . )
```



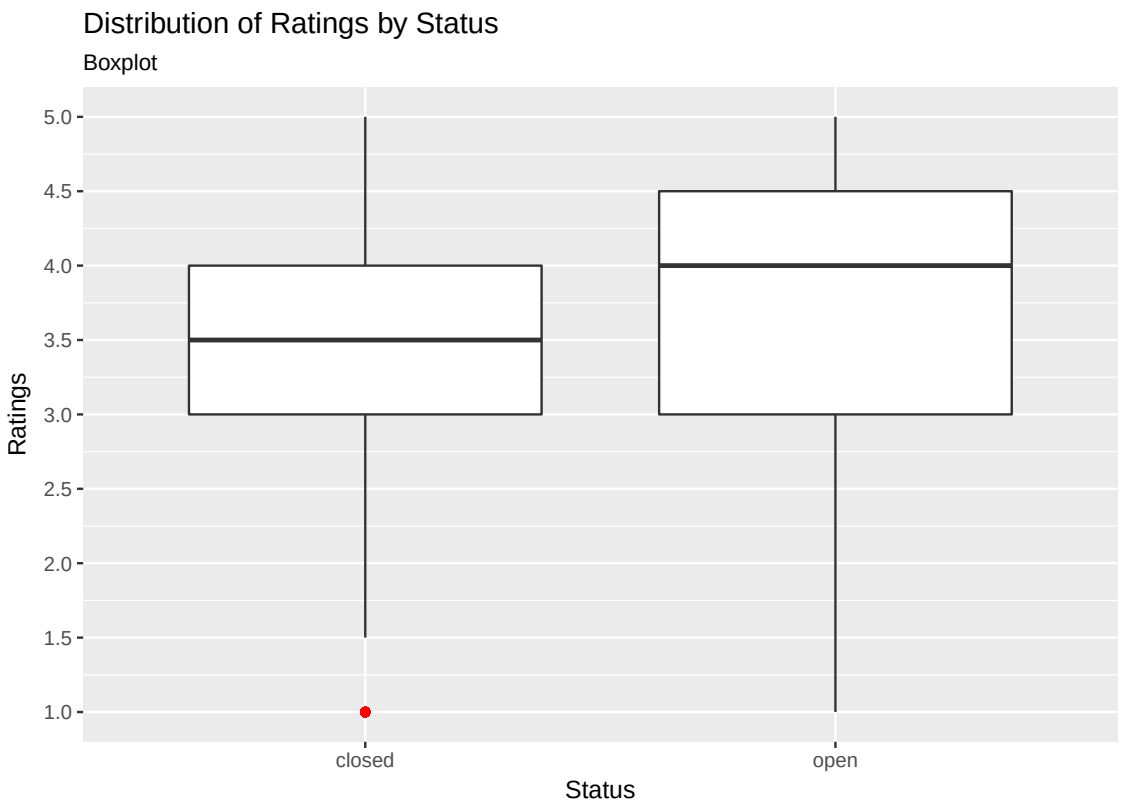

```
## Visualize Reviews by open/closed status
```

Remove entries from the data frame that have reviews as NA. Moreover, add another column to the data frame that maps the is_open values of "0" to "close" and "1" to "open". Finally, regenerate the plot, but now faceted on the new column. Name the new data frame as "cleansedBusiness" and the new column as "status_char".

4.2.2 Generating a *Boxplot*

Question: What is the distribution of ratings by status?

```
graph <- ggplot( cleansedBusiness ) ## attach the new df to plotting env
interval <- seq( 0, max(cleansedBusiness$stars), 0.5 ) ## create a sequence
## from 0 to max value of stars column with an increment value of 0.5
graph <- graph +
aes( x = cleansedBusiness$status_char,
     ## map the new column to the x axis
     y = cleansedBusiness$stars
     ## map the stars column to the y axis
   ) +
geom_boxplot( outlier.color = "red" ) +
##add geom to the plot in this case a box plot and set the colour of the
##outlier values to red
ggtitle( "Distribution of Ratings by Status ", subtitle = "Boxplot" ) +
xlab( "Status" ) + ylab( "Ratings" )  ##add guides to the plot
graph + scale_y_continuous( breaks = interval )
```




```
##change the resolution of the y axis
```

Pay close attention the scale_y_continuous() function and the argument passed to it

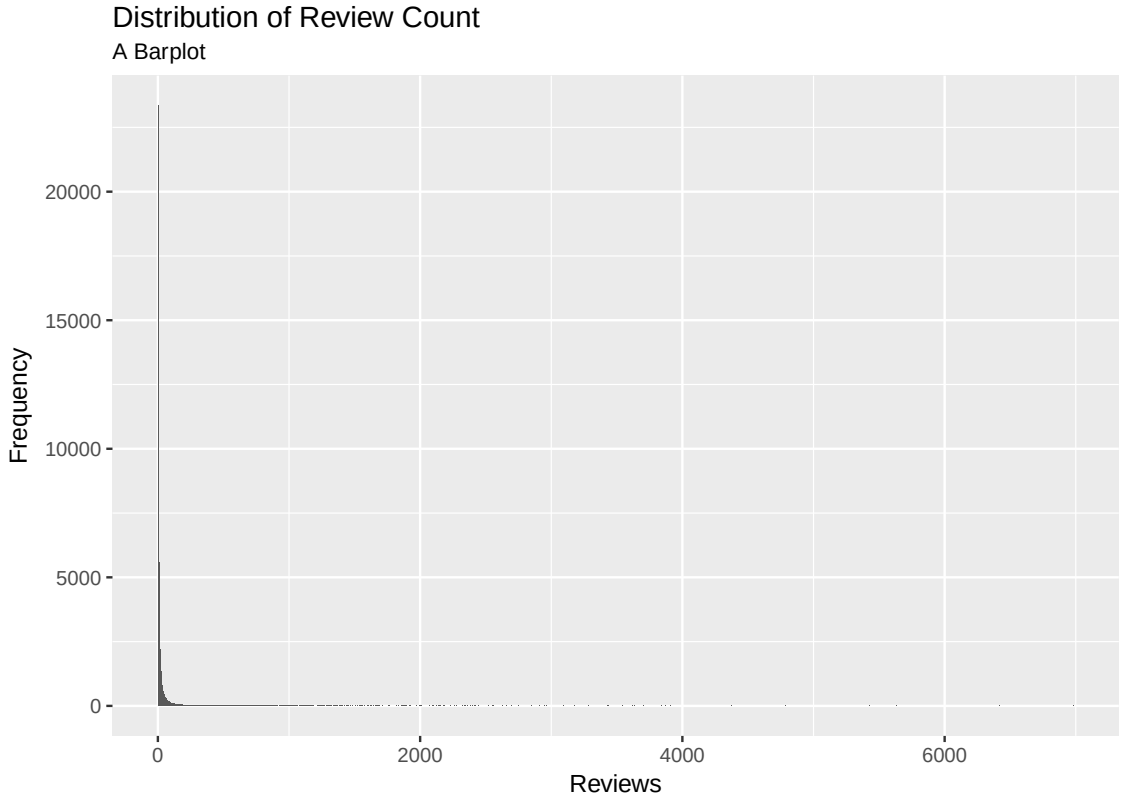
Note the outlier.color parameter that is set and passed to geom_boxplot

Read the help manual for scale_x_discrete() and argue why is scale_y_continuous() a better choice in this case.

4.2.3 Generating a *Barplot*

Question: What is the distribution of reviews?

```
graph <-  ggplot(cleansedBusiness) ## add data to the viz environment
graph +
aes( x = cleansedBusiness$review_count )  +
## map the review count column to the x axis . Since its a bar plot, the
## y axis is a count for each  element in the x axis
geom_bar() +
## add geom to the plot in this case a bar plot
ggtitle( "Distribution of Review Count",
         subtitle = "A Barplot" ) +
xlab("Reviews") + ylab("Frequency") # add guides to the viz environment
```

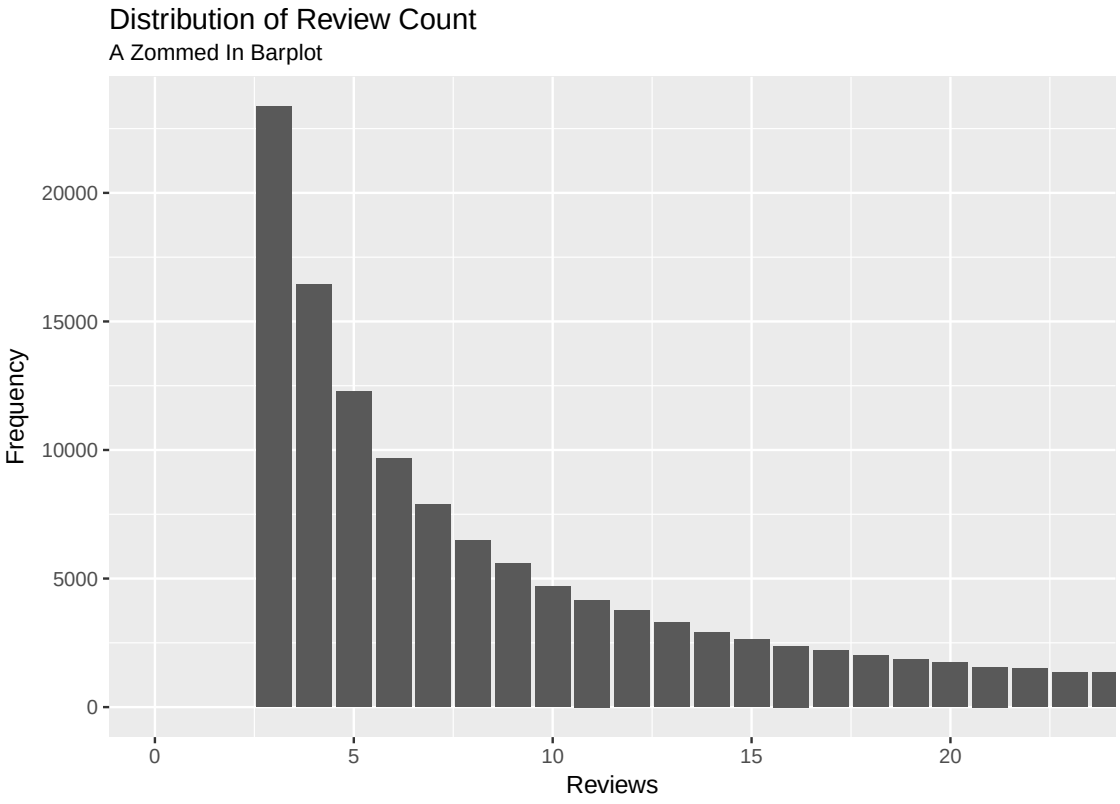


Read the help manual for both geom_bar() and geom_histogram(). Try using geom_histogram() instead of geom_bar(). Explain your observation.

4.2.4 Zooming into a Plot

Question: What is the distribution of reviews within the 3rd quartile?

```
graph <-  ggplot(cleansedBusiness) ## add data to the viz environment
thirdQuartile <- summary(cleansedBusiness$review_count)[5]
## The third quartile of the review counts
graph <- graph +
aes(x = cleansedBusiness$review_count)  +
 ## map the review count column to the x axis . Since its a bar plot,
## the y axis is a count for each  element in the x axis
geom_bar() +
## add geom to the plot in this case a bar plot
ggtitle( "Distribution of Review Count",
          subtitle = "A Zommed In Barplot" ) +
xlab( "Reviews" ) + ylab( "Frequency" )
## add guides to the plot
graph <- graph +
coord_cartesian( xlim = c(0,thirdQuartile) )  ## scale the x axis by
## limiting it to  show values only till the third quartile
graph
```

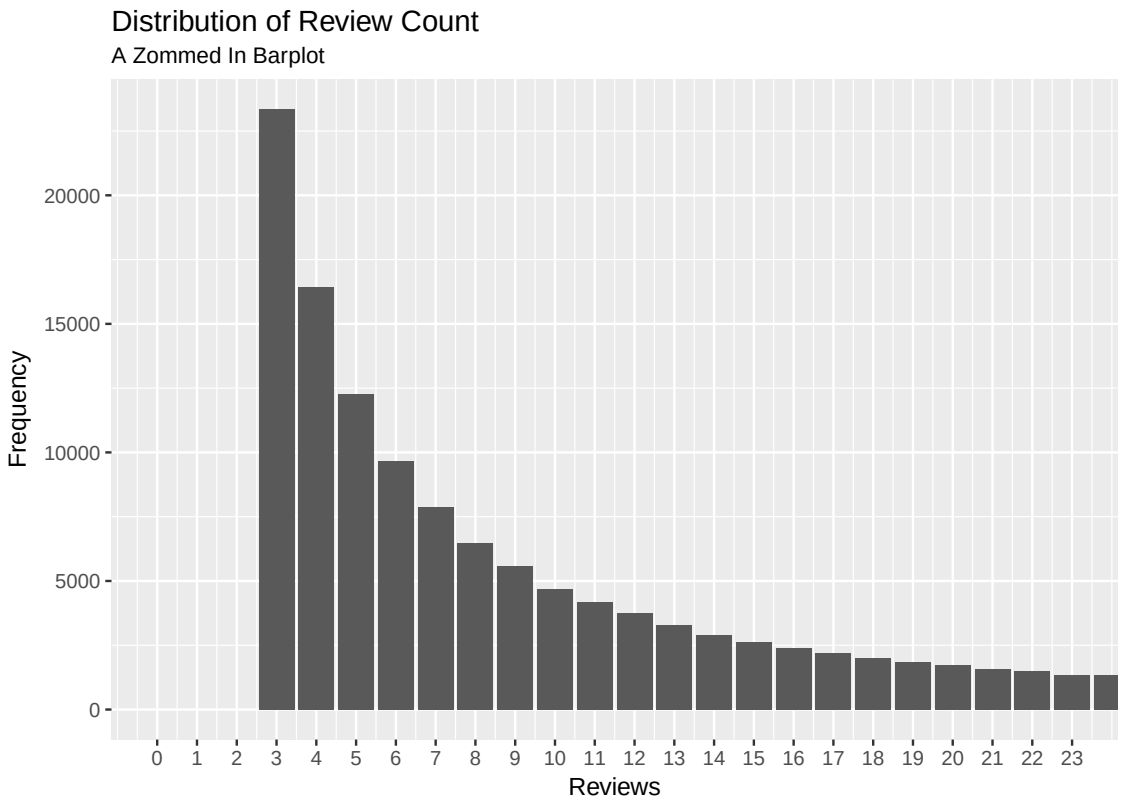


Read the help manual for both coord_cartesian(). Try the above example without the coord_cartesian() function. Explain your observation.

4.2.5 Changing Axis Resolution

Question: How do we change the resolution of the x axis to interval of 1?

```
graph1 <- graph +
scale_x_continuous(
                    breaks = seq(0,thirdQuartile,1)
                   )  ## change the resolution of the
## x-axis to interval of 1
graph1
```



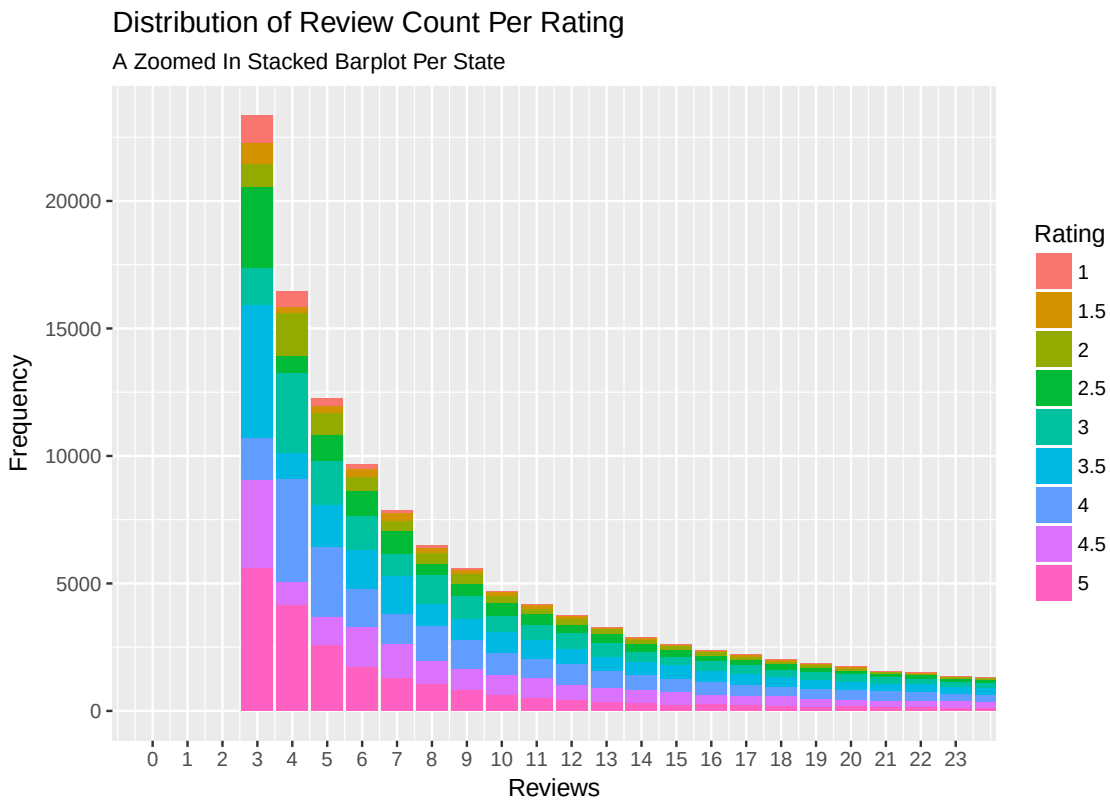
Generate a box plot for the same data and with the same zoomed in view as the barplot

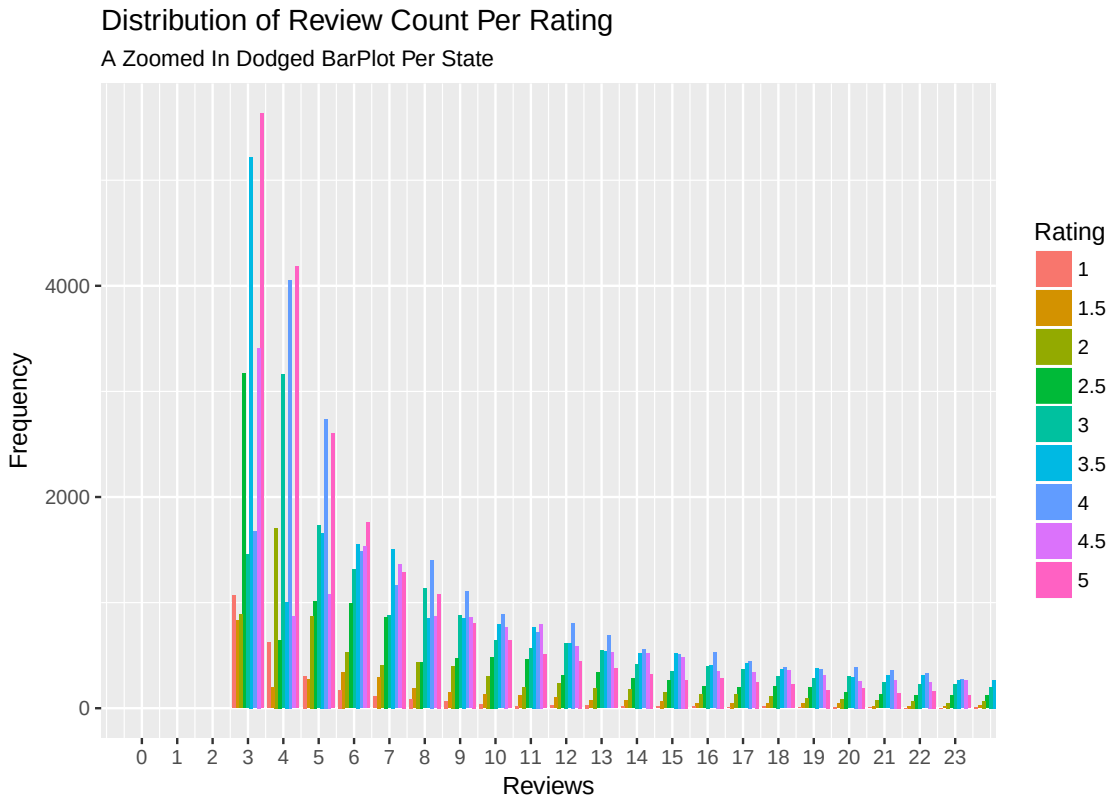
4.2.6 Generating Stacked Barplot and Dodged Barplot

Question: What is the distribution of reviews grouped by ratings?

```
graph    <-  ggplot(cleansedBusiness) ## Attach data to ggplot
base_graph <- graph +
aes(x =
      cleansedBusiness$review_count ## set the aesthetics by mapping
## the  review count to the x axis
    ) +
coord_cartesian(
            xlim = c(0,thirdQuartile) ## scale the x axis by limiting
##it to show values only till the third quartile
             )  +
scale_x_continuous(
           breaks = seq(0,thirdQuartile,1) ## change the resolution of
## the x-axis to interval of 1
           ) +
xlab("Reviews") + ylab("Frequency") ## add titles to the coordinate axes
graph1 <- base_graph +
geom_bar(
          aes(fill =
              as.factor(cleansedBusiness$stars)
              ## map groups of reviews by stars to the aesthetics of
             ) ## fill colour in segments of the bar plot by the
              ## size of each group
       )  +
## By default the bar plot is a stacked barplot
ggtitle( "Distribution of Review Count Per Rating",
         subtitle = "A Zoomed In Stacked Barplot Per State"
       ) ## add titles and subtitles
graph2 <- base_graph +
geom_bar(
         aes(fill =
               as.factor(cleansedBusiness$stars)
           ),
         position = "dodge" ## change the position argument of geom_bar()
## to create a dodged bar plot where the height of each bar is
## size of each   group
        ) +
ggtitle( "Distribution of Review Count Per Rating",
        subtitle = "A Zoomed In Dodged BarPlot Per State")
gridExtra::grid.arrange(
              graph1 + labs(fill = "Rating"),
              graph2 + labs(fill = "Rating"), ##The labs() function
## sets the labels of the plot. Note that we set the argument fill to
## the label name as that is the same argument we have used in aes().
              nrow = 2
```

```
    )  ## generate both plots together. The nrow parameter
```





```
## specifies the row-wise split of the visual environment.
```

- Note the outlier.color parameter that is set and passed to geom_boxplot
- Note the use of **aesthetic mapping** when aes() is passed as an argument to geom_bar()
- Node the use of labs() to set the legend of the plot

What would be the outcome if we used geom_histogram() instead of geom_bar()

Why did we pass the argument *fill* to the labs() function. What would be the outcome if we passed *colour* instead?

Read the help manual for grid.arrange() function.

4.2.7 Merging Data Frames

Question: How can we create a data frame of businesses in top10 categoriesm, having an additional attribute called "category"?

```
countPerCategory <- aggregate( category$category ,
                               by = list(category$category),
                               FUN = length
                              ) ## group the categories in the category
##data setby their count
names(countPerCategory) <- c("category","count")
## add names to the resulting data frame
top10Categories <- head(
                        countPerCategory[
                        order(countPerCategory$count,
                              decreasing = TRUE),
                        ## order the countPerCategory data frame in
                        ## decreasing order
                       ],
                       n=10 ## list only the top 10 elements in the
                       ## sorted data frame
                      )
businesses_top10Categories <-
category[
          category$category
          %in%
          top10Categories[,1], ## search for matching category
          ## in the category dataframe and list all
          ## matching entries in category
      ]

businesses_top10Merged <- merge( cleansedBusiness,
                           businesses_top10Categories,
                           by.x = "id",by.y = "business_id"
                          )
## merging two data frames  by business id
```

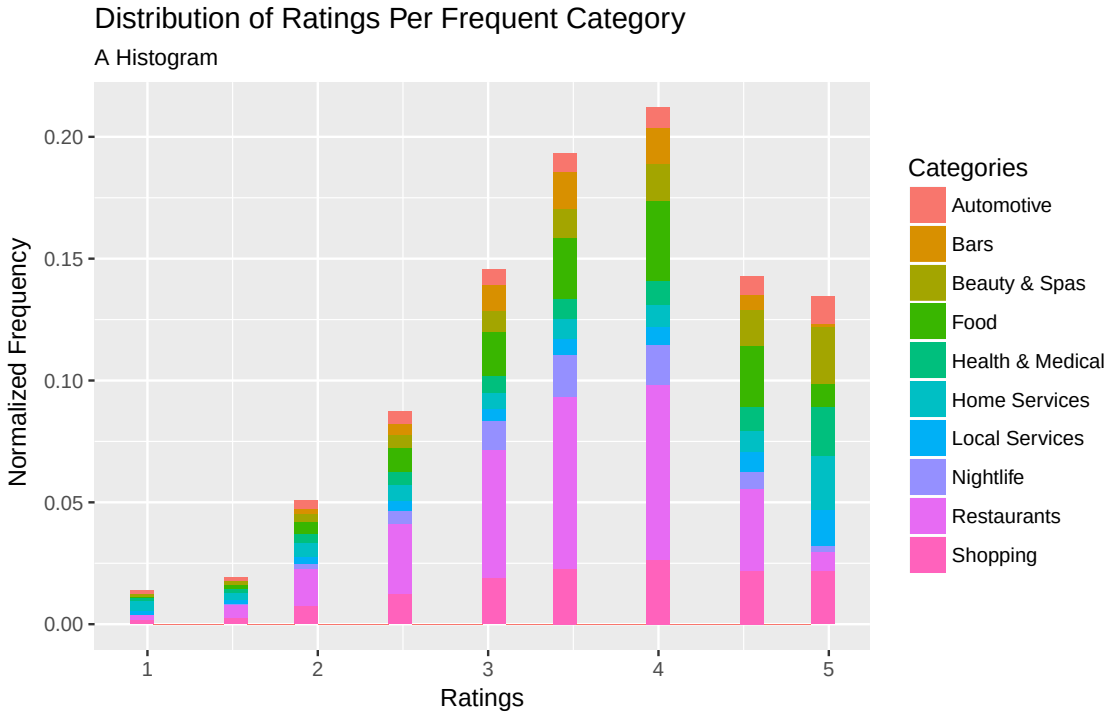
Read the help manual for merge() function.

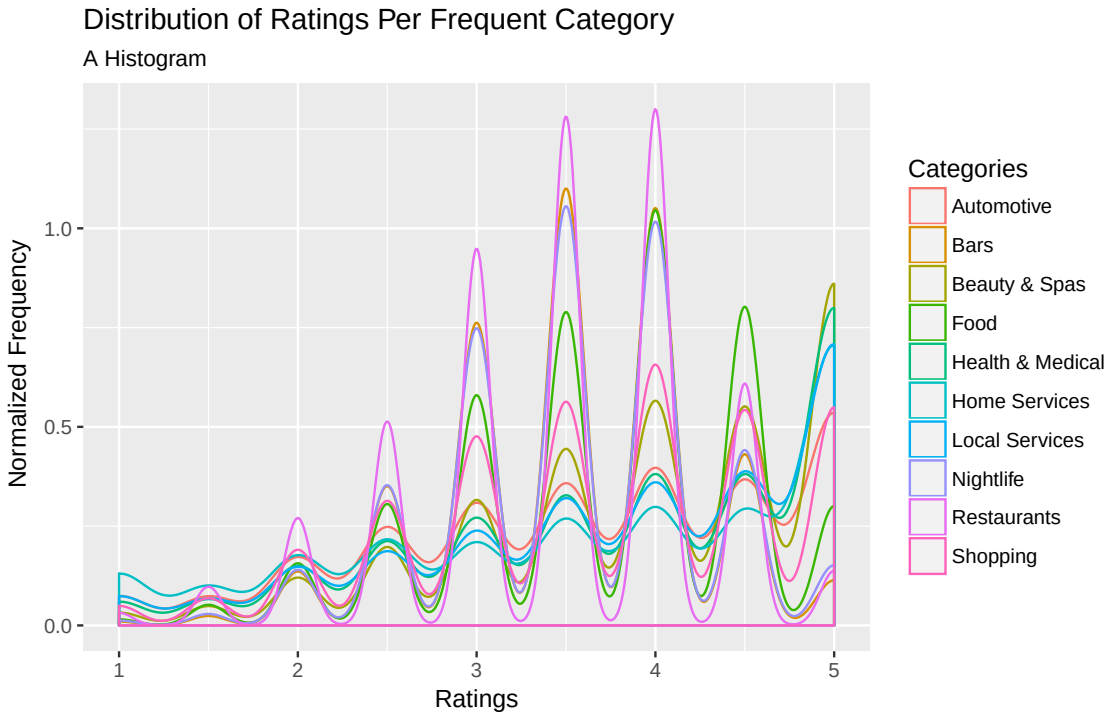
4.2.8 Generating a *Densityplot*

Question: What is the distribution of ratings for each of the top 10 most frequent categories?

```
graph <- ggplot(businesses_top10Merged) ## attached the merged dataframe
## to the viz environment. This new new data frame contains only businesses
## belonging to the top 10 categories and has an additional column called
## category
base_graph <- graph +
aes( x = businesses_top10Merged$stars ## map the attrbute stars to the
     ## x axis
   )  +
xlab("Ratings") +
ylab(" Normalized Frequency") +
ggtitle("Distribution of Ratings Per Frequent Category",
        subtitle = "A Histogram")
## added guides to the visualization
graph1 <- base_graph +
geom_histogram(
                  aes(
                       y = ..count.. / sum(..count..),
                       ## geom_histogram() contains an outcome variable
                       ## called count "..count.." tells geom_histogram()
                       ## to refer to its own outcome variable oand not treat
                       ## count as a user defined variable. Here, we are mapping
                       ## the frequency of each element in the x-axis to y-axis
                       fill = businesses_top10Merged$category
                       ## groups the y-values by category and maps it to the
                       ## the aesthetics of filling colour in
                       ## segments of the plot by the  size of
                       ## each group
                     )
               ) +
labs( fill = "Categories" )  #The labs() function sets
## the labels of the plot. Note that we set the argument fill to
## the label name as that is the same argument we have used in aes().
graph2 <- base_graph +
geom_density(
               aes(color =
                     as.factor(businesses_top10Merged$category)
                 ) ## groups the y-values by category and maps it to the
                   ## the aesthetics of colouring each
                   ## density function by category.
           ) +
 ##the density plot based on the top10 categories
labs( color = "Categories" ) ##setting the label
## to categories. Please note the use setting of the
```

```
## color aesthetics since we did the same in geom_density
gridExtra::grid.arrange(
                         graph1 ,graph2,
                        nrow = 2)## generate both plots together.


## `stat_bin()` using `bins = 30`.  Pick better value with `binwidth`.
```





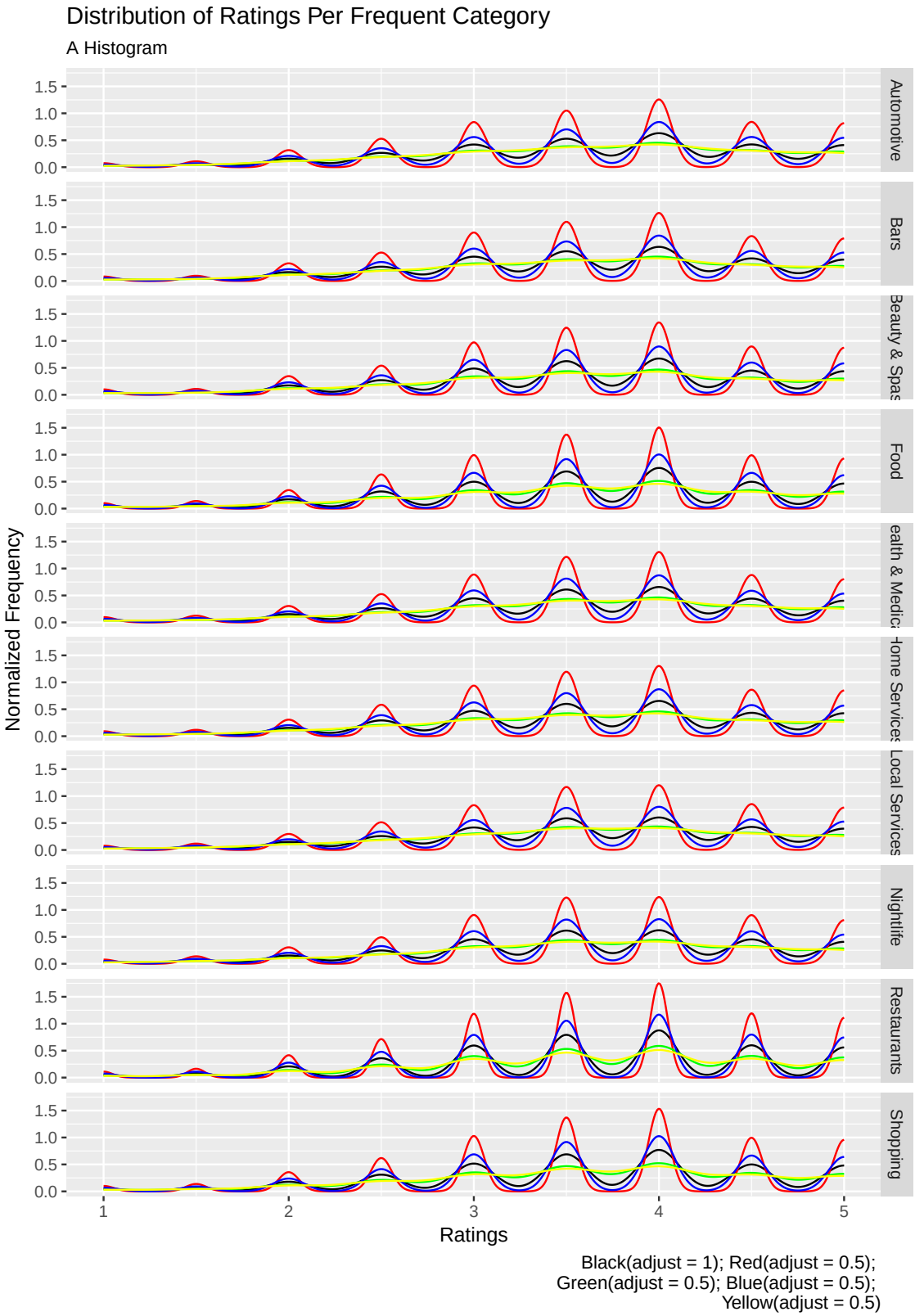
```
## The nrow parameter specifies the row-wise split
## of the visual environment.
```

Note that for geom_density() we map the "color" aesthetics instead of the "fill" aesthetics. Try mapping the "fill" aesthetics instead. What do you observe?

4.2.9 Smoothing Density Plots

Question: How does the density estimate change under different smoothing parameters?

```
base_graph + geom_line( stat = "density" ) + ## geom_line() connects the
## data in their order along the x-axis. Moreover setting the stat argument
## tells ggplot to create a density plot
geom_line( stat = "density", colour = "red", adjust = 0.5 ) + ## the adjust
## parameter sets the smoothing factor of the density estimate
geom_line( stat = "density", colour = "green", adjust = 1.5 ) +
geom_line( stat = "density", colour = "blue", adjust = 0.75 ) +
geom_line( stat = "density", colour = "yellow", adjust = 1.75 ) +
facet_grid(
            as.factor( businesses_top10Merged$category ) ~ .
            ## generates a density plot per category
          ) +
labs( caption = "Black(adjust = 1); Red(adjust = 0.5);
                 Green(adjust = 0.5); Blue(adjust = 0.5);
                 Yellow(adjust = 0.5)"
    )  ## adds a caption to the plot
```


Distribution of Ratings Per Frequent Category
A Histogram
Normalized Frequency
Ratings
Automotive
Bars
Beauty & Spas
Food
ealth & Medic
Home Services
Local Services
Nightlife
Restaurants
Shopping
Black(adjust = 1); Red(adjust = 0.5);
Green(adjust = 0.5); Blue(adjust = 0.5);
Yellow(adjust = 0.5)

4.2.10 Combining Multiple Plots

Centering and Scaling the Data

$$\frac{X-\mu}{\sigma}$$

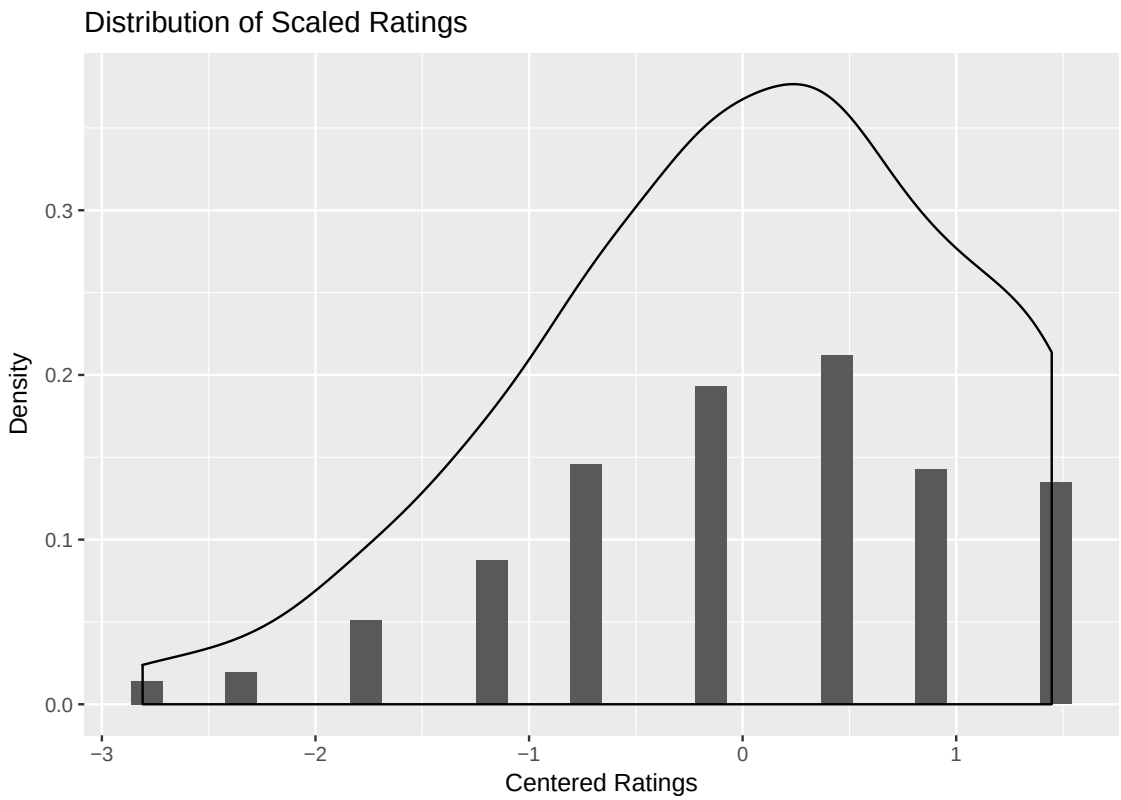
where X is an observation in the dataset, μ and σ are the mean and standard deviation respectively.

```
centered <- ( businesses_top10Merged$stars  -
                mean( businesses_top10Merged$stars )
              ## compute the difference between each value in the stats
              ## attrbute and its mean
             ) / sd( businesses_top10Merged$stars ) ## divide the difference
            ## by the standard deviation of stars
businesses_top10Merged <- data.frame( businesses_top10Merged ,
                                centered_ratings = centered )
```

Try centering the review attribute and generate two boxplots ,arranged side by side. The first boxplot should be on the uncentered review counts while the second boxplot should be on the centered review counts. Explain your observation.

Question:What is the distribution of Ratings once it has been centered?

```
graph <- ggplot( businesses_top10Merged )
graph1 <- graph +
aes( x = businesses_top10Merged$centered_ratings ,
     ## map centered_rating attribute to the x-axis
     y = ..density.. ##map the density variable of
     ## geom_density() to the y-axis
   ) +
geom_histogram(aes( y =
                      ..count.. / sum(..count..)
                    ## count is the computed variable of
                    ## geom_histogram()
                  )
               ) +
geom_density( adjust = 4 ) + ## adjust the smoothness
## of the density plot
xlab( "Centered Ratings" ) + ylab( "Density" ) +
ggtitle( "Distribution of Scaled Ratings" )  ## add
## guides to the plot
graph1


## `stat_bin()` using `bins = 30`.  Pick better value with `binwidth`.
```

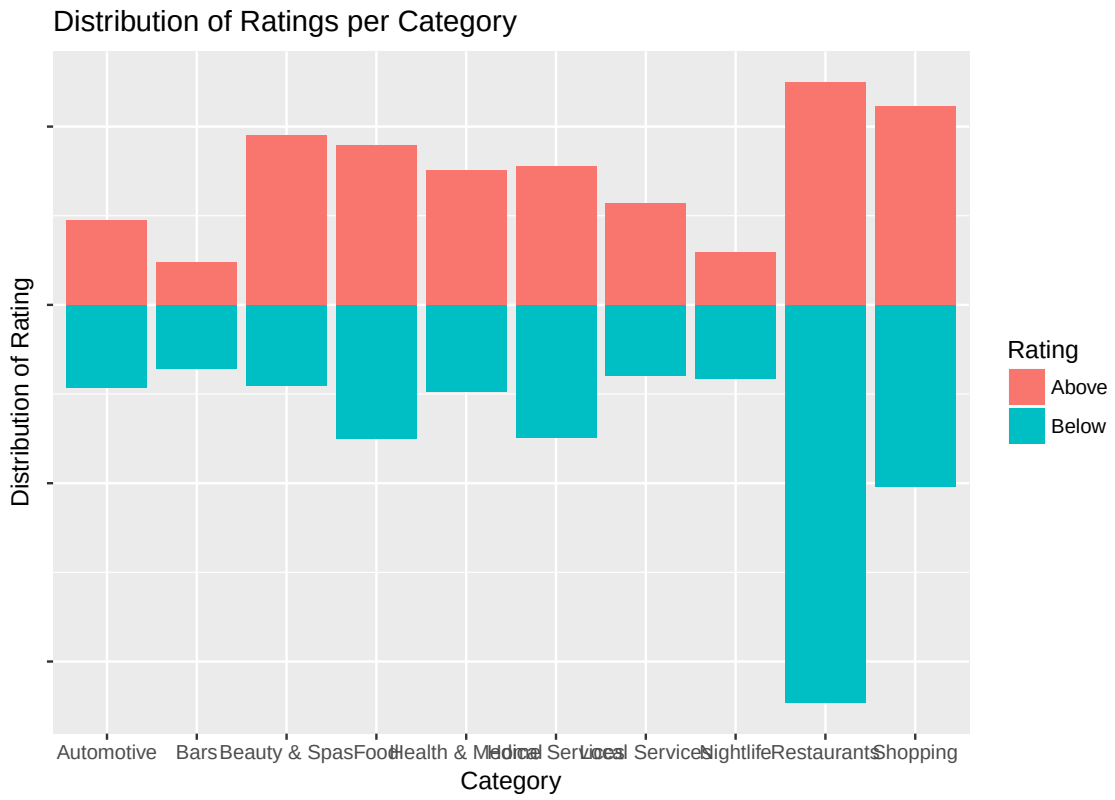


4.2.11 Generating a *Columnplot*

Question:What is the distribution of data Ratings per top 10 category, once it has been centered?

```
rating_categorization <- ifelse(
                            businesses_top10Merged$centered_ratings > 0,
                            "Above","Below")
## Note the use of the ifelse statement, which checks if the
## centered ratings are above or be low 0 and creates a vector
## of containing "Above" / "Below"
businesses_top10Merged <- data.frame(businesses_top10Merged,
  rating_categorization =
          rating_categorization ) ## add the new attribute to
## the business_top10Merged data frame
graph <- ggplot(businesses_top10Merged)
graph + aes(x = as.factor(
                             businesses_top10Merged$category
                           ),
                ## map the x-axis to the category
                ## attribute of the businesses_top10Merged
                ## data frame
            y =  businesses_top10Merged$centered_ratings )  +
                ## map the y-axis to the centered_ratings
```

```
                ## attribute of the businesses_top10Merged df
geom_col( aes( fill =
                as.factor(businesses_top10Merged$rating_categorization)
              ) ## set the aesthetics mapping to group the business
            ## categories by whether they are "Above" / "Below".
          ) +
theme( axis.text.y = element_blank() ) +
## Disable the labels on the y-axis
labs( fill = "Rating" ) +
xlab("Category") + ylab("Distribution of Rating") +
ggtitle("Distribution of Ratings per Category") ## add guides to the
```




```
## plot
```

4.3 Statistical Modeling

4.3.1 Binary Vector Representation of Categorical Attributes

As explained in 1.1.2, each categorical value needs to be mapped to a different dimension. This requires us to represent categorical values as a binary vector.

Question: What is the binary vector representation of the top10 categories?

```
## a function to create a binary vector
## the function accepts the following arguments
## category_ =  a category of type character whose corresponding position
##   in the binary vector is set to TRUE
## categoryNames_ = a vector of category names that defines the
##   total number of categories and hence the output dimension
## returns: a vector of type logical which has dimensions
##            length(categoryNames_) x 1
createBinaryCategoryVector <- function(category_,categoryNames_){
  colIdx <- which(categoryNames_ == category_)
  ## find the index of the categoryNames_ vector that
  ## matches the value in category_. The logical value
  ## for this position has to be set to TRUE
  leftVec <- rep ( FALSE,(colIdx-1) )
  ## create a vector for the left of the matching position
  rightVec <- rep ( FALSE, length(categoryNames_) - colIdx )
  ## create a vector for the right of the matching position
  categoryVec <- c(leftVec, TRUE, rightVec)
  ##concatenate the left and right vectors to the value
  return(categoryVec)
}

CategoriesCount <- aggregate(category$category,
                             by = list(category$category),
                             length)
## find the fereency of occurence of each category from the
## category dataset
names(CategoriesCount) <- c("category","frequency")
## assign attribute names to the CategoriesCount df
mostFrequentCategories <- head(
                           CategoriesCount[
                                       order(CategoriesCount$frequency,
                                             decreasing = TRUE),
                                     ],
                           ## order the CategoriesCount dataframe in
                           ## decreasing order of occurence of
                           ## each category
```

```
                            n = 10
                          )
##  get the 10 most frequent categories
business_mostFrequentCategories <- category[
                                            category$category %in%
                                            mostFrequentCategories$category,
                                           ]
## get all business id's belonging to the top 10 categories

## length(unique(business_mostFrequentCategories£business_id)) <
##    length(business_mostFrequentCategories£business_id)
## check if businesses are assigned to more than one category

categoryMatrix <- t(
                      sapply(business_mostFrequentCategories$category,
                            FUN=createBinaryCategoryVector,
                            categoryNames_=mostFrequentCategories$category
                           )
                    )
## for every value in business_mostFrequentCategories£category apply
## the function createBinaryCategoryVector()  and transpose it to get
## a binary vector of dimension
## 1 X length(mostFrequentCategories£category)
## Thefore for all applying the function to all elements of
## business_mostFrequentCategories£category will
## result in a matrix of dimension:
## length(business_mostFrequentCategories£category)  X
##      length(mostFrequentCategories£category)

row.names(categoryMatrix) <- NULL
## set the row names of the category matrix to NULL to remove
## duplicate row names as conversion of a matrix to a data.frame
## expects unique rownames

catVecDF <- as.data.frame(categoryMatrix)
## convert the matrix to a data frame
names(catVecDF) <- mostFrequentCategories$category
## assign names to the data frame

catVecDF <- data.frame(business_id =
                         business_mostFrequentCategories$business_id,
                       catVecDF)
## add business_id to the data frame

numCols <- dim(catVecDF)[2]
## get the number of columns of the data frame

CategoriesBinaryRep <- aggregate(catVecDF[,2:numCols],
```

```
                                    by = list(catVecDF$business_id),
                                    FUN = sum)
## aggregate the data frame by business_id such that an OR operation
## takes place beetween two or more rows belonging to the same
## business id

names(CategoriesBinaryRep)[1] <- "business_id"
## assign attribute name to the first column of the data frame

result <- apply(CategoriesBinaryRep[,2:11],2,max)
## validate whether there were any duplicate rows in
## the data. If that was the case, the max per column
## obtained by the apply command would be greater than 1
```

- Create a binary vector representation of the attributes, corresponding to the business_id values in **CategoriesBinaryRep**. Name this data frame as **AttributesBinaryRep**. How many business id's could be found? List the business id's that could not be found.
- Use the checkins data frame that we created in 3.3.3 to compute the total number of checkins per day of week. Match the business_id in this data frame to business_id in **CategoriesBinaryRep**. For the ones that match,give it the same binary vector representation as **AttributesBinaryRep** but instead of 0/1, it should contain number of checkins per attribute. Name this data frame as **CheckinsBinaryRep**. Moreover, name each attribute in this data frames as numCheckins_Monday,numCheckins_Tuesday,....
- Merge all three data frame you created to obtain a new data frame. Name this as **dataSet**.

4.3.2 Creating a Training and Validation Set

In order to build a predictive model, we need to split the data into two or more **mutually exclusive** data sets, namely:

1. ***Training Set(s)*** A randomly selected subset of the data, that is used to fit a model and further tune its parameters.
2. ***Validation Set(s)*** A randomly selected subset of the data, whose outcomes we will predict, using our model and compare the predicted outcomes with the actual outcomes (the **ground truth**).

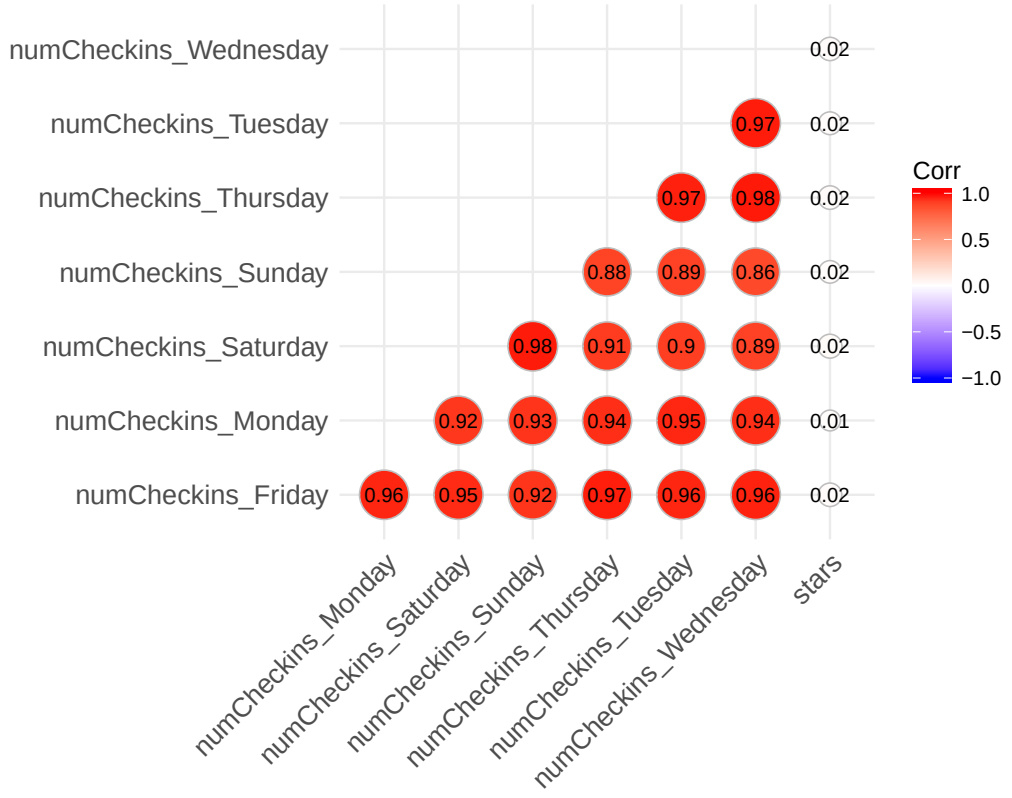
Question: How can we create a training set and a test set from a given data set?

```
dim(dataSet)


## [1] 113845     60


numRecords <- dim(dataSet)[1]
Idx <- seq(1:numRecords)
## create a vector of same length as number of examples in
## the training set
trainingIdx <- sample(Idx, 0.8 * numRecords, replace = FALSE)
## sample 80% of the values from the vector, without replacement
## this results in an index for the training set
testIdx <- Idx[!(Idx %in% trainingIdx)]
## find the values in Idx that are not present in training Idx
## This forms the inex of the validation set
trainingSet <- dataSet[trainingIdx,]
## Select rows from the dataSet that correspond to
##  indexes in trainingIdx
testSet <- dataSet[testIdx,]
## Select rows from the dataSet that correspond to
##  indexes in testIdx
```

4.3.3 Feature Analysis and Selection

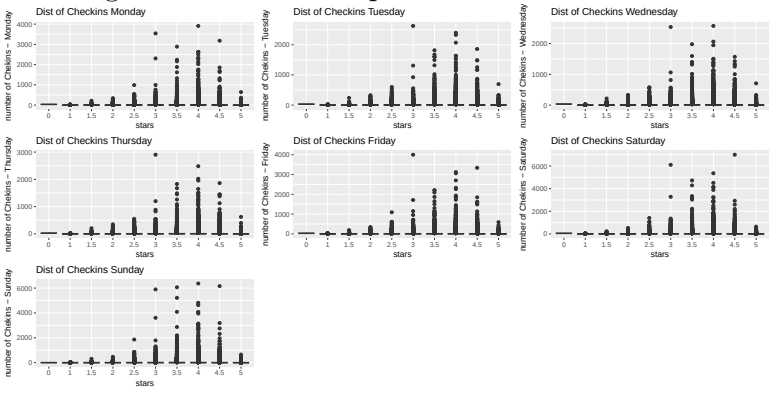
Question: How can we detect outliers in the data?

```
## A function to create a boxplot
## The function expects the following
## arguments
## base_graph : the ggplot object with the dataframe
##              attached to it
## aesMap : the maping of the attrbutes to the x and y
##          axis
## xlabel : the x axis label
## ylabel : the y axis label
## title  : title of the plot
## return : the boxplot object
boxplotMatrix <- function(base_graph,aesMap,xlabel,
                          ylabel,title){
  graph <- base_graph +
           aesMap +
           geom_boxplot() +
           ggtitle(title) +
           xlab(xlabel) +
           ylab(ylabel)
   return(graph)
}

base_graph <- ggplot(trainingSet)
graph1 <- boxplotMatrix(base_graph,
                        aes(x =
                              as.factor(trainingSet$stars),
                            y = trainingSet$numCheckins_Monday
                        ),
                        "stars","number of Chekins - Monday",
                        "Dist of Checkins Monday")

subsetCols <- names(trainingSet)[2:8]
subsetCols <- append(subsetCols,"stars")
ggcorrplot(cor(trainingSet[,subsetCols]),
            method = "circle",
           type ="lower",lab = TRUE,
           lab_size = 3)
```



In order to use ggcorrplot(), the user will need to install the package as follows:
devtools::install_github("kassambara/ggcorrplot")

- Generate the boxplot for all seven days and generate the plot as shown below

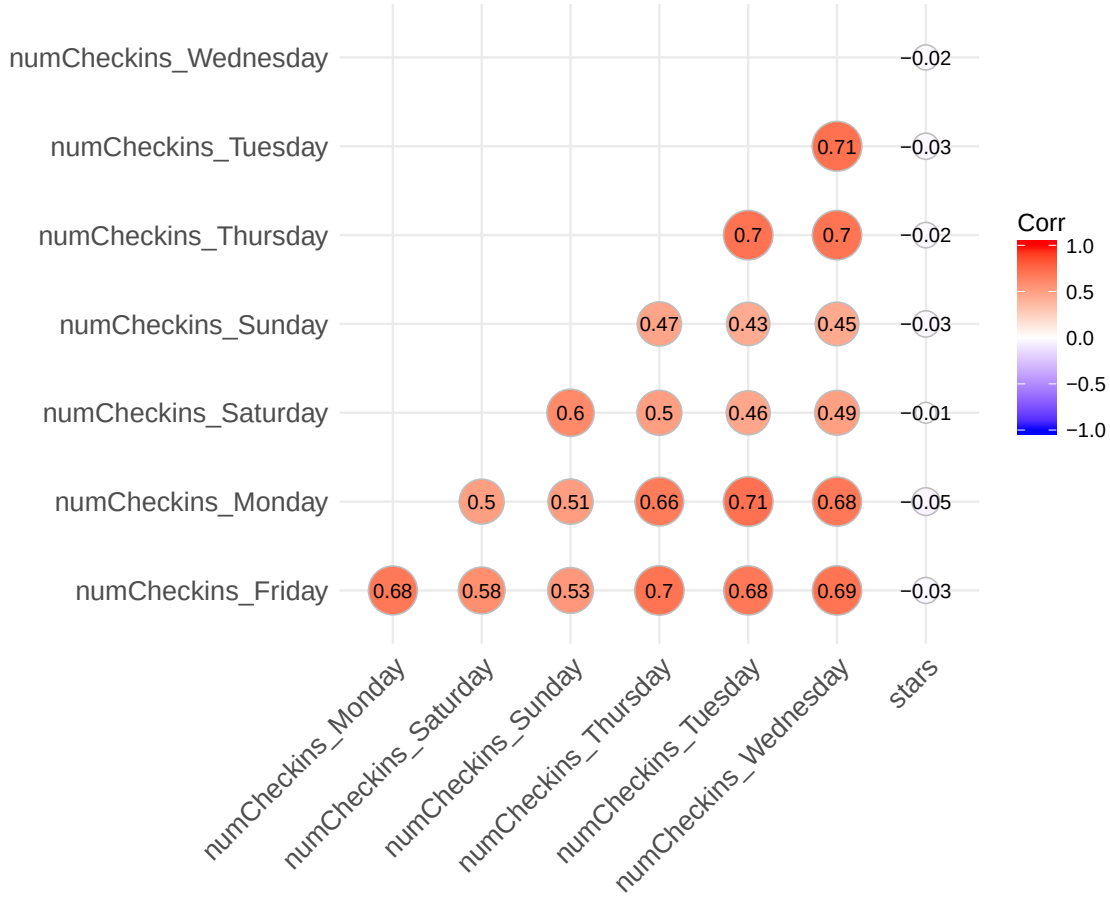


- Do the same for reviews i.e detect outliers in checkins while using reviews as a grouping variable. Moreover generate the correlation plot matrix and box plot matrix for the same

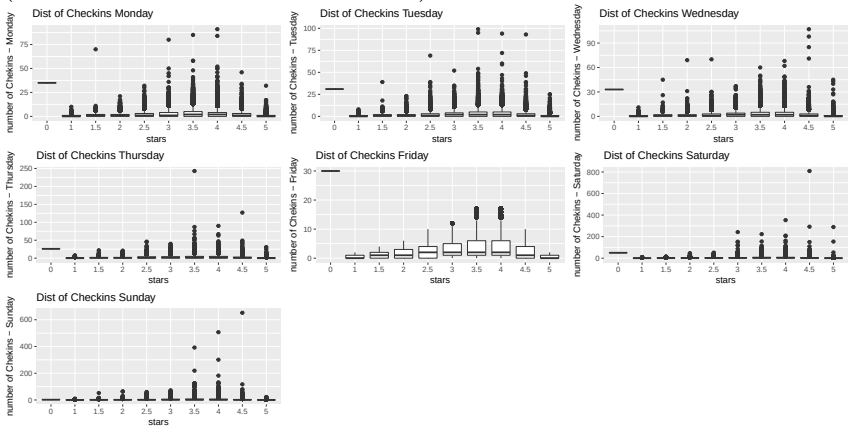
Question: How can we remove outliers from the data?

```
## a function to returns the third quartile value grouped
## by another variable
## the function accepts the following arguments
## input_ =  the vector whose summary statistics needs to
##           be computed
## groupingVec = the vector containing values on which
##          we need to group the  values in input_ vector
## returns: the third quartile of each group
getOutlierThresholds <- function(input_,groupingVec){
  result <- aggregate(input_,by = list(groupingVec),summary)
  ## compute the summary statistics of input_ when grouped
  ## by the vales in groupingVec
  ## returns a list where the first element is the grouping
  ## values while the second element is the
  summaryStats <- result[,2]
  groupingVals <- result[,1]
  return(summaryStats[,5])
}


thresholdsPerRating <- apply(trainingSet[,2:8],
                             2,
                             getOutlierThresholds,
                             groupingVec=trainingSet$stars)
##apply the function getOutlierThresholds() on each of the
## numChekins__ columns of trainingSet and group by
## the attribute "stars"

thresholdsPerRating <- data.frame("stars" =
                                    sort(unique(trainingSet$stars)),
                                  thresholdsPerRating)
## add the attribute "stars" to the data and create a data frame

## a function to filter data based on
## input conditions
## the function accepts the following arguments
## input_ =  A vector of size 2 that contains
##           (a) attribute filter
##           (b) value threshold
## dataVec1 = the vector on which the
##        attribute filter has to be applied
## dataVec2 = the vector on which the
##        threshold filter has to be applied
## returns: vector of logicals, where the values
## in positions that satisfy the conditions
## are set to FALSE
```

```
removeOutliers <- function(input_,dataVec1,dataVec2){
  rating <- input_[1]
  threshold <- input_[2]
  idxVec <-  rep(TRUE,length(dataVec1))
  idxVec[dataVec2 == rating & dataVec1 > threshold] <- FALSE
  return(idxVec)
}

result <- apply(thresholdsPerRating[,c("stars","numCheckins_Friday")],
                1,removeOutliers,
                dataVec1 = trainingSet$numCheckins_Friday,
                dataVec2 = trainingSet$stars)
## apply the function removeOutliers() to each row of the
## first two columns of the thresholdsPerRating data frame
## The vectors on which the conditions have to be applied
## are passed on dataVec1 and dataVec2.
## the result is a logical matrix of dimensions
##  length(stars) X dim(thresholdsPerRating)[1]
## the columns in result corresponds to
## the value of star

outliers_numCheckins_Friday <- apply(result,1,prod)
## For a given row of result, if either one of
## the condition is satisfied
## (i.e one of the columns = FALSE),
## the value should be filtered out. Therefore
## we take a columnwise product

cleanedTrainingSet <- trainingSet[as.logical(outliers_numCheckins_Friday),]
## Filter all records from the trainingSet that ae not
## outliers

graph1 <- boxplotMatrix(base_graph, ##the base graph
                        aes(x =
                              as.factor(cleanedTrainingSet$stars),
                            y = cleanedTrainingSet$numCheckins_Monday
                        ), ## aesthetics mapping
                        "stars","number of Chekins - Monday",
                        "Dist of Checkins Monday")
ggcorrplot(cor(cleanedTrainingSet[,subsetCols]), ##
            method = "circle",
           type ="lower",lab = TRUE,
           lab_size = 3)
```



Generate the boxplot for all seven days (without the outliers) as shown below



- Generate the correlation matrix for rest of the features wrt the outcome variable. What do you observe?
- Are there features that have low correlation among themselves but high correlation with the outcome variable? Make a list of these features.
- Using the function that your wrote in 3.4, add the following features to the data i.e

  1. number of checkins on weekdays
  2. number of checkins on weekends
  3. number of checkins during morning
  4. number of checkins during evening
  5. top5 states

- Analyze the new features, namely

  1. Check for presence of outliers wrt to the outcome variable.Remove the outliers. Generate relevant plots to demonstrate the same.
  2. Check the correlation among these feature and wrt to the outcome variable. Explain your observation.

4.3.4 Setting up a Linear Regression Model

Question: How can we do linear regression?

```
modelFull <- lm(stars ~ . , data = trainingSet[,2:59])
## fit a linear regression model on the full data where the
## outcome variable is the number of stars
predictedStars <- predict.lm(modelFull,testSet[,2:58])
## use the fitted model to predict the star of a business
squaredError <- (testSet$stars - predictedStars)^2
MeanSquaredError <- mean(squaredError)
## Calculate the mean squared error between the acutal rating
## and predicted rating
MeanSquaredError


## [1] 0.733


modelOutlierRemoved <- lm(stars ~ . ,
                          data = cleanedTrainingSet[,2:59])
## fit a linear regression model on the outlier removed data
## outcome variable is the number of stars
predictedStars <- predict.lm(modelOutlierRemoved,testSet[,2:58])
## use the fitted model to predict the star of a business
squaredError <- (testSet$stars - predictedStars)^2
MeanSquaredError <- mean(squaredError)
## Calculate the mean squared error between the acutal rating
## and predicted rating
MeanSquaredError


## [1] 1.11
```

Why do we see a higher mean squared error even after removing the outliers?

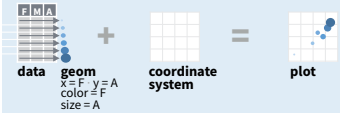
- Setup a linear regression model by only considering the features that you have found useful in the previous section. Do you observe a change in the RMSE?
- Do the same by considering the outcome variable as reviews
- In 4.2.10, we converted ratings to a binary vector indicating whether it is above or below the average. Setup a regression model using this as the outcome variable. Make sure you do feature analysis wrt this outcome variable.

Data Visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.



Complete the template below to build a graph.

```
ggplot (data = <DATA>) +                                   required
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
 stat = <STAT> , position = <POSITION> ) +                 Not required, sensible defaults supplied
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

ggplot(data = mpg, **aes(**x = cty, y = hwy**))** Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings | data | geom

qplot(x = cty, y = hwy, data = mpg, geom = "point**")** Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot() Returns the last plot

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1,curvature=z**))** - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round", linemitre=1**)**
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group**))**
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1**))** - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900**))** - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1**))**
b + geom_hline(aes(yintercept = lat**))**
b + geom_vline(aes(xintercept = long**))**

b + geom_segment(aes(yend=lat+1, xend=long+1**))**
b + geom_spoke(aes(angle = 1:1155, radius = 1**))**

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)
```

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5**)** x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy**))** x, y, alpha, color, fill, linetype, size, weight

discrete

```
d <- ggplot(mpg, aes(fl))
```

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x , continuous y

```
e <- ggplot(mpg, aes(cty, hwy))
```

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)** x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2**)**
x, y, alpha, color, fill, shape, size

e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"**)**, x, y, alpha, color, linetype, size

e + geom_smooth(method = lm**)**, x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)**, x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

```
f <- ggplot(mpg, aes(class, hwy))
```

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"**)**, x, y, alpha, color, fill, group

f + geom_violin(scale = "area"**)**, x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y

```
g <- ggplot(diamonds, aes(cut, color))
```

g + geom_count(), x, y, alpha, color, fill, shape, size, stroke

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))
```

h + geom_bin2d(binwidth = c(0.25, 500)**)**
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function

```
i <- ggplot(economics, aes(date, unemploy))
```

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))
```

j + geom_crossbar(fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar(), x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

```
data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))
```

k + geom_map(aes(map_id = state), map = map**)**
+ expand_limits(x = map$long, y = map$lat**)**,
map_id, alpha, color, fill, linetype, size

THREE VARIABLES

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))l <- ggplot(seals, aes(long, lat))
```

l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE**)**
x, y, alpha, fill

l + geom_tile(aes(fill = z)**)**, x, y, alpha, color, fill, linetype, size, width

Stats An alternative way to build a layer

A stat builds new variables to plot (e.g., count, prop).

data **stat** **geom** (x = x, y = ..count..) **coordinate system** **plot**

Visualize a stat by changing the default stat of a geom function, **geom_bar(stat="count")** or by using a stat function, **stat_count(geom="bar")**, which calls a default geom to make a layer (equivalent to a geom function). Use **..name..** syntax to map stat variables to aesthetics.

geom to use | stat function | geommappings

i + stat_density2d(aes(fill = ..level..), geom = "polygon")

variable created by stat

c + stat_bin(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..

c + stat_count(width = 1) **x, y,** | ..count.., ..prop..

c + stat_density(adjust = 1, kernel = "gaussian")
x, y, | ..count.., ..density.., ..scaled..

e + stat_bin_2d(bins = 30, drop = T)
x, y, fill | ..count.., ..density..

e + stat_bin_hex(bins=30) **x, y, fill** | ..count.., ..density..

e + stat_density_2d(contour = TRUE, n = 100)
x, y, color, size | ..level..

e + stat_ellipse(level = 0.95, segments = 51, type = "t")

l + stat_contour(aes(z = z)) **x, y, z, order** | ..level..

l + stat_summary_hex(aes(z = z), bins = 30, fun = max)
x, y, z, fill | ..value..

l + stat_summary_2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

f + stat_boxplot(coef = 1.5) **x, y** | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..

f + stat_ydensity(kernel = "gaussian", scale = "area") **x, y** | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

e + stat_ecdf(n = 40) **x, y** | ..x.., ..y..

e + stat_quantile(quantiles = c(0.1, 0.9), formula = y ~ log(x), method = "rq") **x, y** | ..quantile..

e + stat_smooth(method = "lm", formula = y ~ x, se=T, level=0.95) **x, y** | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + stat_function(aes(x = -3:3), n = 99, fun = dnorm, args = list(sd=0.5)) **x** | ..x.., ..y..

e + stat_identity(na.rm = TRUE)

ggplot() + stat_qq(aes(sample=1:100), dist = qt, dparam=list(df=5)) **sample, x, y** | ..sample.., ..theoretical..

e + stat_sum() x, y, size | ..n.., ..prop..

e + stat_summary(fun.data = "mean_cl_boot")

h + stat_summary_bin(fun.y = "mean", geom = "bar")

e + stat_unique()

Scales

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.

(n <- d + geom_bar(aes(fill = fl)))

scale_ | aesthetic to adjust | prepackaged scale to use | scale-specific arguments

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

GENERAL PURPOSE SCALES

Use with most aesthetics

scale_*_continuous() - map cont' values to visual ones
scale_*_discrete() - map discrete values to visual ones
scale_*_identity() - use data values **as** visual ones
scale_*_manual(values = c()) - map discrete values to manually chosen visual ones
scale_*_date(date_labels = "%m/%d", date_breaks = "2 weeks") - treat data values as dates.
scale_*_datetime() - treat data x values as date times. Use same arguments as scale_x_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

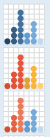
Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

COLOR AND FILL SCALES (DISCRETE)

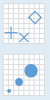
n <- d + geom_bar(aes(fill = fl))
n + scale_fill_brewer(palette = "Blues")
For palette choices:
RColorBrewer::display.brewer.all()
n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")

COLOR AND FILL SCALES (CONTINUOUS)



o <- c + geom_dotplot(aes(fill = ..x..))
o + scale_fill_distiller(palette = "Blues")
o + scale_fill_gradient(low="red", high="yellow")
o + scale_fill_gradient2(low="red", high="blue", mid = "white", midpoint = 25)
o + scale_fill_gradientn(colours=topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES



p <- e + geom_point(aes(shape = fl, size = cyl))
p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7))
p + scale_radius(range = c(1,6))
p + scale_size_area(max_size = 6)

Coordinate Systems

r <- d + geom_bar()

r + coord_cartesian(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system

r + coord_fixed(ratio = 1/2)
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

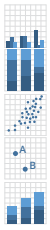
r + coord_polar(theta = "x", direction=1)
theta, start, direction
Polar coordinates

r + coord_trans(ytrans = "sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

π + coord_quickmap()
π + coord_map(projection = "ortho", orientation=c(41, -74, 0))projection, orienztation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Position Adjustments

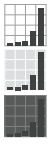
Position adjustments determine how to arrange geoms that would otherwise occupy the same space.



s <- ggplot(mpg, aes(fl, fill = drv))
s + geom_bar(position = "dodge")
Arrange elements side by side
s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height
e + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting
e + geom_label(position = "nudge")
Nudge labels away from points
s + geom_bar(position = "stack")
Stack elements on top of one another

Each position adjustment can be recast as a function with manual **width** and **height** arguments
s + geom_bar(position = position_dodge(width = 1))

Themes



r + theme_bw()
White background with grid lines
r + theme_gray()
Grey background (default theme)
r + theme_dark()
dark for contrast

r + theme_classic()
r + theme_light()
r + theme_linedraw()
r + theme_minimal()
Minimal themes
r + theme_void()
Empty theme

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(. ~ fl)
facet into columns based on fl
t + facet_grid(year ~ .)
facet into rows based on year
t + facet_grid(year ~ fl)
facet into both rows and columns
t + facet_wrap(~ fl)
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

t + facet_grid(drv ~ fl, scales = "free")
x and y axis limits adjust to individual facets
"free_x" - x axis limits adjust
"free_y" - y axis limits adjust

Set **labeller** to adjust facet labels

t + facet_grid(. ~ fl, labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(fl ~ ., labeller = label_bquote(alpha ^ .(fl)**))**

α^c	α^d	α^e	α^p	α^r

t + facet_grid(. ~ fl, labeller = label_parsed)

c	d	e	p	r

Labels

t + labs(**x** = "New x axis label", **y** = "New y axis label",
title ="Add a title above the plot",
subtitle = "Add a subtitle below title",
caption = "Add a caption below plot",
<AES> = "New <AES> legend title")

Use scale functions to update legend labels

t + annotate(geom = "text", x = 8, y = 9, label = "A")

geom to place | manual values for geom's aesthetics

Legends

n + theme(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right"
n + guides(fill = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)
n + scale_fill_discrete(name = "Title",
labels = c("A", "B", "C", "D", "E"))
Set legend title and labels with a scale function.

Zooming

Without clipping (preferred)
t + coord_cartesian(
xlim = c(0, 100), ylim = c(10, 20))

With clipping (removes unseen data points)
t + xlim(0, 100) **+ ylim(**10, 20)
t + scale_x_continuous(limits = c(0, 100)) **+**
scale_y_continuous(limits = c(0, 100))

RStudio