Advanced Analytics, Machine Learning and Artificial Intelligence

Sanjeev Kumar

Technology & Operations, Ross School of Business
University of Michigan

This draft book contains teaching material developed by Sanjeev Kumar for Advanced Analytics, Machine Learning and Artificial Intelligence classes at Ross School School of Business, University of Michigan.

University of Michigan Business School

MICHIGAN ROSS

dedicated to the one to whom I owe all

Brief Contents

Contents

List of Figures

Foreword

Preface

We live in a world awash in data. Companies are increasingly turning to data analytics to extract a competitive edge from data, especially large, complex datasets often called "Big Data". However, companies are increasingly facing the challenge posed by the scarcity of analytical talent - people who can turn data into better decisions, people who can extract insights and information from data. There is growing demand for professionals with strong quantitative skills combined with an understanding of how data analytic techniques can be applied to business contexts and managerial decision making. To help my students succeed in this growing field, I teach classes in **Advanced Big Data Analytics** in Ross School of Business, Univ. of Michigan. In these courses I teache advanced analytical, statistical and data mining tools with an applied focus. This book is developed specifically for these classes.

The main focus of this book (and the associated coures) is to prepare students to model and manage business decisions with data analytics and decision models using real life case contexts and datasets. By the end of this book students will have a better understanding of processes, methodologies and tools used to transform the large amount of business data available into useful information and support business decision making. The book will focus on extracting actionable business intelligence by analyzing traditional business data as well as more recently available datasets such as social media, crowdsourcing and recommendation engines. The book focuses on the powerful, open source (and hence free) data analysis environment R.

This book came out of my preparations for the following two courses:

1. **TO414: Advanced Analytics for Management Consulting** - Elective course for Senior and Junior BBA students.
2. **TO628: Advanced Big Data Analytics** - Elective course for second year MBA students.

As I designed these courses, I realized that while there are a lot of references available for doing Advanced Analytics on Big Data using R, there isn't a good reference that approaches the topic from the perspective of business students and is accessible for students who do not have extensive background in statistics or computer programming. I designed my classes to have an applied nature with significant amount of hands-on work on real business Big Data with significant managerial implications. I emphasized aspects of the analytics process that are important for actual practice of data science but are not typically well covered in traditonal textbooks - like data cleaning, managing large datasets and building data dashboards. I ended up creating a significant amount of material for the classes - much of it collated from already available but widely dispersed sources. This book is a collection of these material.

Business students have a unique mixture of tech savvy, super smarts and learning ability with a relative lack of computer programming or coding experience. They further have a great applied sense of statistics and data analysis but typically without the theoretical expertise of a Statistics student. This book is directed towards such students. This book assumes that business students are joining an Advanced Analytics course without any knowledge of computer programming and without any background in R. This book assumes that students

are familiar with basic probability and statistics but their focus is on applied statistics - not on the theory. The book then builds up student's comfort level with R while at the same time makeing progress on key Advanced Analytics materials.

Direct all feedback on this book to the author at email: `sankum at umich.edu` or twitter: `@a_teachr`

Dr. Sanjeev Kumar
Technology & Operations
Ross School of Business, Univ. of Michigan

This document was processed on: 19th Apr, 2022
at 2:15am

Introduction

Welcome to **Advanced Big Data Analytics Using R**: Navingating Business Big Data with R. Let's get started.

Who Is This Book For?

This book is for business students, practitioners and executives who want to have hands-on experience in working with business related big data and want to extract actionable insights from the data using advanced analytical techniques. This book is application oriented and hence focuses on the application of advanced analytical techniques rather than the theoretical details. Consequently, this book is not for readers solely focused on the theoretical, statistical part of data analytics.

This book assumes that the readers have basic familiarity with introductory probability and statistics. This domain is easier to follow and understand if the reader also has *some* exposure to some kind of computer programming although that is not a necessity. This book has been written for a practitioner audience, not an academic one. The book aims to be an exhaustive reference resource for practitioners - that's why it starts with baby steps of introducing R and basic data manipulation and ends at the other end with advanced Machine Learning algorithms and complex model improvement algorithms.

Thank you for placing your trust in this book. This chapter will provider further details about this book and how to best read/use this book.

GPL License usually require that the product (in this case, this book) be made available free or charge; and any subsequent product made using the GPL Licensed product must also be made available free of charge [1].

About This Document

This document, which you are probably seeing as a PDF document or an eBook or even a printed book, has been written entirely using R and RStudio. It uses two R packages `Sweave` and `knitr` to integrated R code inside my favorite document creation system called LaTeX. This document has been created entirely using Open Source tools and has been released back into the Open Source ecosystem for free using the GNU General Public License (GPL). You are free to share this document with others as long as you comply with the GPL license.

This is my first attempt at writing a book size document. I have had to develop a bunch of LaTeX, knitr and R workarounds to make the process work and get a quality output. My LaTeXtemplate for this book and details of several of the workarounds are posted on my personal website `http://ateachr.com`.

I plan to publish these and other details on ateachr.com - but as of this version of the document, I have not been able to. I will revise this note after I have published the material.

This section is still more ambition than execution. I hope to get all this done - but as of now, the details are not all finished.

Associated R Package and Website

This book makes use of several data sources, custom developed R code, question sets, documentation and so on. All of these material are packaged together in an R package hosted on GitHub called `clarifyR` . You can download all the material associated with

I think the name is clever. I was pretty happy with myself when I found out that the clarifyR name was available. :-)

the book by installing the clarifyR package in your R installation. For downloading and installing R packages from GitHub, you will need the `devtools` package. Follow the code below that uses the following commands: `install.packages` for installing a R package, `library` for loading a package into memory and finally the `install_github` command that is part of the devtools package for installing a package from GitHub. Note that the install_github command uses the name of the package and the GitHub username that the package belongs to. As most GitHub packages are public, we do not need a password.

```
#Code to be changed
#Installing clarifyR package from GitHub
#First get devtools package and load into memory
#install.packages("devtools");library(devtools)
#install_github("clarifyr", username="clarifyr")
#library(clarifyr)
```

If you wish to access the book contents outside of the R ecosystem, then you can head on to `http://clarifyr.com`. The website has all the material available as part of the clarifyR package. The website also works as an interaction platform between the author and readers and students. Feel free to ask any questions related to the book content there - I will be happy to answer them.

How is This Book Structured

This book covers a wide range of material. The material is organized in 5 parts, 19 chapters and several appendices.

Part I: Getting Started

We set ourselves for the book - download and install all needed software; figure our way around R and RStudio and learn the basics of the R language. Essentially lay down enough of a foundation that we can start getting productive. For students without a computer programming background, it is essential that they do not rush this part. Our success in later parts depend upon us getting comfortable with the material here.

Part II: Data Exploration and Visualization

In my opinion, data cleaning is the most under-appreciated part of data analytics. It often takes more time and effort than the actual analysis that follows.

Real data analytics begins with us understanding and getting a handle on the data. This typically involves cleaning up the data, generating descriptive statistics to better understand the data and finally creating visualizations that allow us to better understand the underlying complexity of the data.

Data visualization has emerged as a key tool in Big Data Analytics. In this book we devote just one chapter to it - a necessary limitation as we have such a broad variety of topics to cover in this book. We focus our attention here on a subset of data visualization that is important in the business context - creating data dashboards that can help managerial decision making.

Part III: Traditional Statistical Modeling

Even advanced analytic techniques like machine learning algorithms in the next part have their foundations in traditional statistical methodologies. Classical statistical modeling approaches like Linear Regression are still the benchmark given their immense popularity, flexibility and stability. In this part of the book, we explore traditional statistical analysis

tools like Linear Regression, Generalized Linear Models like Logistic and Survival Models, Principal Components and Factor Analysis, Time Series Analysis and so on.

As we assume that you are already familiar with basic probability and statistics, we will focus on application of the methodologies and not the underlying theory. We will also focus on how to tweak these tools for the Big Data world as many of these tools run into trouble when sample sizes are quite large. Bigger is not always better - traditional statisticall methodologies were developed/optimized for smaller sample sizes. Using them for large sample sizes give rise to unique issues and problems - we will discuss how to address them.

Part IV: Machine Learning and Predictive Analytics

This is the largest and the most important part of the book. This is why this book was written in the first place - an overview of data analytics tools specific to Big Data - variously known as Machine Learning, Statistical Learning, Predictive Analytics, Data Mining as so on. This book focuses on tools that help us make sense of Big Data and helps us automate the extraction of managerial decision making insights from Big Data.

As with much of the rest of the book, the focus is on applying the tools rather than their theoretical/mathematical underpinnings. We will discuss enough theory to develop an overall understanding and then devote our energies on making these tools work on real datasets.

Part V: Putting It All Together

Now that we have gone through all the elements individually, we can move forward to create a combined, integrated approach that puts all these pieces together. How to combine different models so that they result in an output better than sum of their parts? How to ensure that our models improve as more data become available?

We will conclude by running a couple of large integrated data analytics projects that will combine elements discussed in the book.

Part V: Appendices, Bibliography and Index

Back matter of the book. The Index in the end has three main components - Key Concepts, R Commands and R Packages. Appendices include syllabus for the two courses this book is primarily used for and other miscellaneous content that did not fit in one of the main matter chapters.

There are a lot of quality, free, online resources available for building up your R and Data Analytics skills before you go through this book or the associated courses. They are listed in Appendix: Key References. A fully fleshed data analysis example has been presented on Appendix: Data Analysis Example to give you an idea of the power and range of what can be accomplished with just a little bit of familiarity with R.

How to Read This Book

You would have noticed by now that this book integrates R code within its text. Most of the time R code takes the form of dedicated code blocks like the one below:

```
#This is a demo code block
print("Hello World")
```

```
## [1] "Hello World"
```

As you can see from above, code blocks are printed in color, in `fixed width font`. There is a color scheme here that you will soon become familiar with - comments are in italics and kind of violet looking, functions are in reddish color, strings appear bluish and so on. The output of the code block appears right after - for example - the output of the `print` command follows the code above.

You would notice that whenever we refer to a R command in a significant way (like `print` in paragraph above), the command is printed in `fixed width font`, in red color with a yellow highlight that makes it easy for you to see which commands are discussed significantly on the page. The highlighting also ensures that an entry for the command is placed in the R Commands section of the Index provided at the end of the book. For times when a command is mentioned in text in a minor way not necessating a margin and Index entry, they will be typed in `fixed width font`.

Like R Commands, this book provides special formatting for R Packages used in the book. R Packages are fomatted like R Commands but in blue color - check our reference of the `clarifyR` package earlier - blue fixed width font text with yellow highlighting and finally an entry in the Packages section of the Index.

I would have liked to make margin notes green - but somehow green color prints a horribly toxic green - so I am defaulting to black until I can find a way to print a darker, pleasanter green.

This is a demo margin note. You will find helpful hints, debugging tips and other thoughts and trivia here.

Lastly, special formatting is provided for key concepts - similar to R Commands and Packages in all respects except that they are in bolded and in black color and their Index entry is in the Key Concepts section. For example: this book focuses on **Open Source** software - that are created by a community of software developers for use by the community, usually made available for free.

The three special formatting elements - commands, packages and key concepts - ensure that you have a summary of significant elements discussed in the page just by looking for highlighted items. Other than these, you would also notice that the book makes extensive use of margin notes aligned to the relevant location on the page. I use these margin notes for interesting tidbits that I want to emphasize, my tangential thoughts or just stuff I couldn't find space for in the main text. I also expect that the expanded margin area will be useful for readers to make notes, include remarks etc.

Important Note on Figure Placements and Other Quirks of LaTeX

This book has tonnes of figures - including images and tables. In a book of this size, its quite an undertaking to ensure that all figures are placed appropriately. I typically let LaTeX place figures where it should be placed given considerations of page break, whitespace and readability. This sometimes results in figures being placed a little earlier or later in the document than where you would expect them. Typically LaTeX prefers to put images at the top of the page. I have taken care to refer to figures with their numbering so that you can always identify which figure I am referring to - for exampple - see Figure: 1 for my most favorite moment in Michigan Football history - Woodson making an amazing one handed interception against the little brothers. As you can see - the placement is weird. Figure numbers help - still, figure placements in this book can be a little disorienting at first - I hope you will get used to it quickly.

Figures are usually horizontly centered on the page with horizontal lines at the top and botton separting it from the main text. All figures have a figure number and a caption attached to them. All figures are referred in the List of Figures by their figure number, caption text and the page number where they appear.

This book has been formatted to show page numbers in the bottom center of the page, chapter number and name on the top header of even pages and section number and name on the top header of odd pages. Default font size for this book is 11 points. Contact the author if you would like to get a copy of the PDF version of the book with a larger font size. The original LaTeX files are available upon request as well.

Figure 1: Woodson is a Super Human

If you would like to know more about LaTeX then I would recommend taking a look through Goossens et al. [2]. A great, free, online resource to start learning about LaTeX is https://en.wikibooks.org/wiki/LaTeX [11].

Is This Book Suitable For You?

Well - you wouldn't know until you spend some time with it. However, to give you a quick flavor of what you can expect from this book, I have included a sample basic data analysis example in Appendix B. If the material in Appendix B catches your fancy, looks interesting - then you are in the right place. Dig in.

A quick word of caution though before you get too deep: this book (and the associated courses) are very hands-on. There is no point in reading this book like a sequence of text. This book should be seen more as an illustrated text - illustrated with relevant R commands and material. You should read this book alongside an RStudio session - trying all the commands there as you read along. You will need to get comfortable with a lot of command line typing, keyboard shortcuts and figuring workarounds to inevitable problems that will arise. We will often get into situations where there will be no tested/optimized/prescribed solution and we will need to figure our way out - often with some trial and error. You should be comfortable with such ambiguity.

Part I

Getting Started

Chapter 1

Advanced Big Data Analytics

In God We Trust; All Others Must Bring Data
– *Willian Edwards Deming*

We live in a world awash in data. All this data can be incredibly useful - it can tell us a lot - help us be better. Consider the case of WalMart in Figure: 1.1 [4]. WalMart collects every bit of customer data it can find - and used this data to figure out what do people buy when a hurricane is about to hit an area. I often ask this question to my students - and you can guess the usual answers - water, canned food, board games. Good guesses - but wrong. These are anticipated needs and most people purchase these items days and weeks in advance. WalMart's analysis told them that the items people buy the most just before a hurricane hits are pop-tarts and beer. This insight has tremendous implications for the bottomline. However, extracting clear insights like this from mountains of raw data is not easy - its quite difficult in fact. We need Advanced Data Analytics techniques to do so.

Success of WalMart in extracting insights from data is being replicated by different companies in different industries. NetFlix is able to predict what movies you will like. Sprint is able to predict when a customer might start considering switching services to a competitor. In a now famous incident, Target even managed to deduce that a teenage girl was pregnant before the girl's family knew [5]. In each of these cases, companies are using specific methodologies, that we refer in this book as **Advanced Analytics** , to transform data into insights for making better decisions.

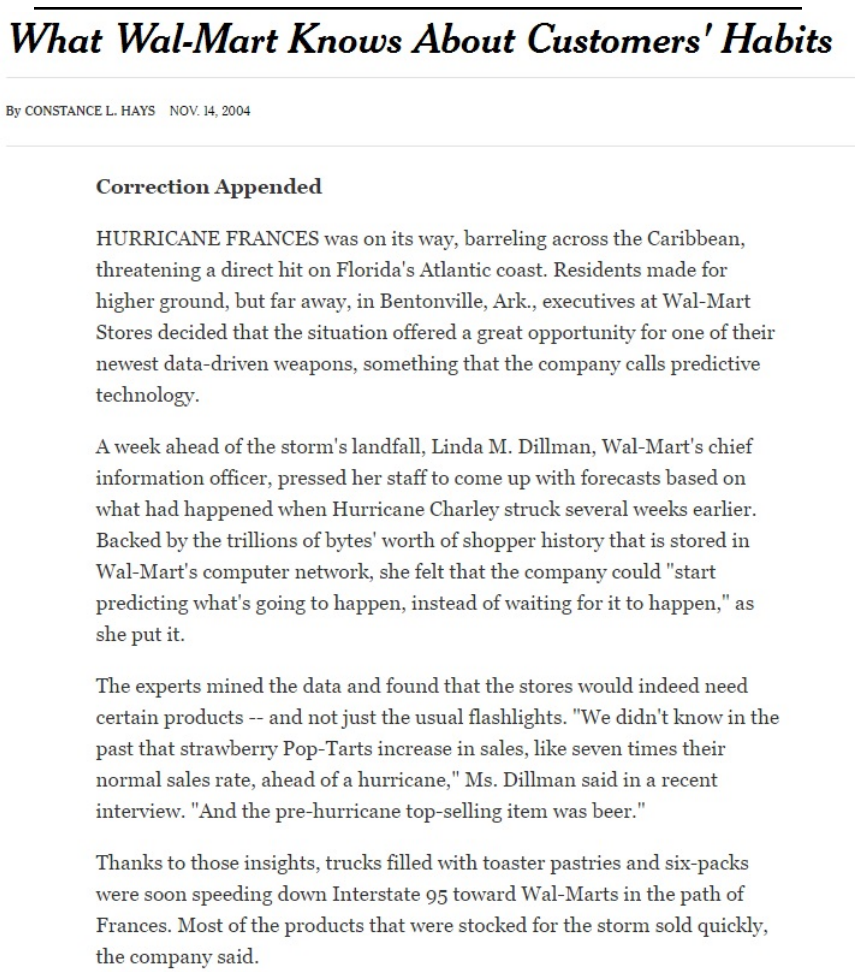
The popular notion that Target knew that the girl was pregnant before even *she* did; is not accurate. Sadly.

Our networked, connected, technology enabled world generates a lot of data that can be used for Analytics. Each time we use our credit cards, click on a web link, send a text message, even talk when your mobile phone is nearby with the Facebook app running - data is generated and stored [3]. As you can imagine, all this data adds up. It adds up to so much that we can not hope to manage, process or analyze this much data with commonly available software in a reasonable amount of time - also known as **Big Data** . So we need Advanced Analytics techniques that can handle Big Data.

The terms Advanced Analytics, Data Analytics and Business Analytics are often used interchangeably. We will consider them synonymous in this book.

1.1 What is the Big Deal with Big Data?

Big Data is obviously big - but size is only one aspect of Big Data that makes analyzing it difficult. Before we aanalyze Big Data, we need to first get familiar these roadblocks. They are usually referred to as the Four Vs: Volume, Velocity, Variety and Veracity [8].



What Wal-Mart Knows About Customers' Habits

By CONSTANCE L. HAYS NOV. 14, 2004

Correction Appended

HURRICANE FRANCES was on its way, barreling across the Caribbean, threatening a direct hit on Florida's Atlantic coast. Residents made for higher ground, but far away, in Bentonville, Ark., executives at Wal-Mart Stores decided that the situation offered a great opportunity for one of their newest data-driven weapons, something that the company calls predictive technology.

A week ahead of the storm's landfall, Linda M. Dillman, Wal-Mart's chief information officer, pressed her staff to come up with forecasts based on what had happened when Hurricane Charley struck several weeks earlier. Backed by the trillions of bytes' worth of shopper history that is stored in Wal-Mart's computer network, she felt that the company could "start predicting what's going to happen, instead of waiting for it to happen," as she put it.

The experts mined the data and found that the stores would indeed need certain products -- and not just the usual flashlights. "We didn't know in the past that strawberry Pop-Tarts increase in sales, like seven times their normal sales rate, ahead of a hurricane," Ms. Dillman said in a recent interview. "And the pre-hurricane top-selling item was beer."

Thanks to those insights, trucks filled with toaster pastries and six-packs were soon speeding down Interstate 95 toward Wal-Marts in the path of Frances. Most of the products that were stocked for the storm sold quickly, the company said.

Figure 1.1: WalMart Figures What to Sell in a Hurricane

Volume

The size of Big data is a constantly moving target - this section will likely be obsolete by the time you get to reading it. Sure Big Data is Big - but most people do not understand just *how* big it is. It is estiated that 90% of all data that exists has been created in last two years [7]. It is estimated that WalMart has collected more than 2.5 petabytes of customer data. A petabyte is 10^{12} or one quadrillion bytes, or the equivalent of about 20 million filing cabinets' worth of text [9]!

This incredible size of Big Data means that such data is usually stored in muliple, distributed machines. It also means that commonly available software like MS-Excel are horribly inadequate to deal with Big Data. The volume issue is actually not fully resolved in R - our tool of choice in the book. How much data we can handle in a single installation of R is usually limited by the size of RAM memory of the machine. In later chapters, we will explore various approaches to enable R to handle larger volume of data.

Velocity

Velocity refers to the speed with which Big Data is growing. It is the classic drinking from the firehose situation. Our data analytics tools have been fast enough to accommodate

this size of incoming traffic to analyze them in real time. Our models have to be robust enough to not get quickly outdated by the new data. However, high velocity also presents a tremendous opportunity as new incoming data provides potential for continuous learning and refinement of our models.

As much of human interaction moves to social media and much of machine to machine interaction moves to Internet of Things (IoT), the velocity of Big Data will only keep growing. Again, our tool of choice - R, faces some problems in managing this deluge of data as R can comparatively slow. In later chapters, we will explore how to make R models faster to be able to handle the high velocity of Big Data.

Variety

Traditional statistical analysis works mainly with quantitative data; typically in the well recognized table format - columns for fields and rows for records. We do have large quantity of such data still - think WalMart transaction data or Nasdaq trading data. However, a large and growing part of today's Big Data is not in neat table format and often not even quantitative. Such data may include text data (blog posts, tweets), rich media data (YouTube videos, Flickr images), sensor data (FitBit, Google Glass), location data (Google Maps, GPS data) etc. Our data analytics methodologies have to be able to make sense of all these different kinds of data.

Veracity

With the staggering volume, supersonic velocity and increasing variety of data come the issues of data veracity - uncertainity regarding the accuracy and trustworthiness of the data. All our data analytic models are not worth anything if they are based on suspect data. Hence, we must ensure data integrity and veracity before building data analytic models to analyze the data.

1.2 Business Data Analytics

Business analytics is the process of transforming data into insights for making better business decisions. Essentially, we want managers to no longer depend on the good old intuition, gut feel or rule of thumb. Instead we want managers to make decisions based on analyzing the available data - the so called Data Driven Decision Making. Following are the three broad types of analytics that are performed as part of business data analytics.

The movie Moneyball is a good example of transitioning from an intuition/gut-feel based decision making to a data driven decision making.

Descriptive Analytics

As the name suggests, Descriptive Analytics attempts to describe what has happened. This can take the form of descriptive statistics, data visualization or data dashboards. The focus is on explanation of an extant fact. We will cover a good part of Descriptive Analytics. Our approach will be focused on R though and will not cover specialized tools like Tableau.

Predictive Analytics

Predictive analytics uses models constructed from past data to predict the future or discover relationships between variables. Examples of such analytical techniques include regression, time series analytics and data mining. This is the major focus of this book - especially Regression and Data Mining.

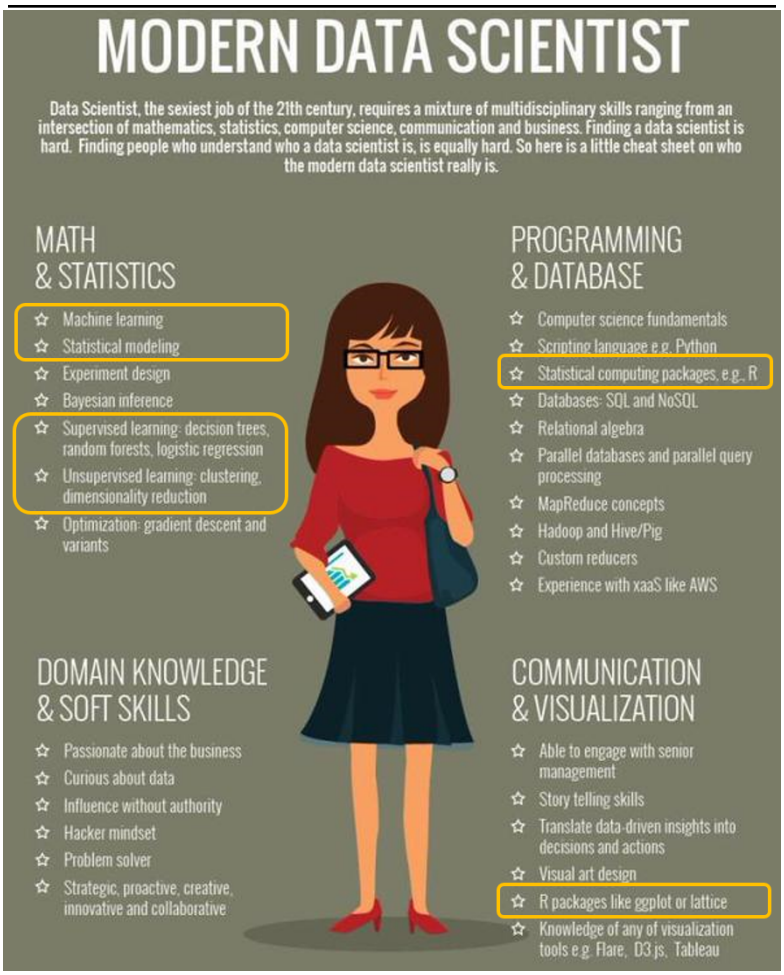


Figure 1.2: Modern Data Scientist

Prescriptive Anaytics

Predictive analytics is concerned with finding the best course of action - including techniques such as optimization and simulation. This is NOT the main focus of this course

Artificial Intelligence

1.3 Data Scientist

Data Scientist is one of the fastest growing job descriptions today - the sexiest of job of 21st century according to Patil and Davenport [10]. However, being a good data scientist is a demanding job. Successful data scientists are knowledgable in statistics, programming, machine learning, data visualization and even the relevant industry domains [6].

See the Figure: 1.2 for typical requirements from a data scientist position. You would note that the requirements span multiple disciplines - math and statistics, programming and databases, and communication and visualization. We will cover a good portion of these requirements in this book and associated courses. I believe that this book is a pretty good start to your journey of becoming a successful data scientist.

Chapter 2

Our Tools: R and RStudio

In God We Trust; All Others Must Bring Data
– *Willian Edwards Deming*

R is awesome. R is also considered pretty difficult to learn. In my opinion, that's not true. In this chapter, we will quickly get out feet wet in the shallow end of the R pool - ready to jump off the deep end in the succeeding chapters.

2.1 Introduction to R and RStudio

For the purpose of this course, R is a Statistical Computing Environment. It provides us with an underlying computer language, analysis tools and software platform for us to work with data. While R is pretty powerful, it is built by geeks for geeks. As a result, it lacks the user-friendliness needed by beginners and casual users. To help mitigate this problem, we do not directly interface with R and instead use the user friendly RStudio software as a wrapper around R. RStudio is often considered an IDE (Integrated Development Environment) for R.

R is a pretty awful name for anything - try googling "R" and see how relevant the search results are! It is called R after the first name of the two authors of R: Robert and Ross. The name kinda makes sense when you consider that R is based on another programming language named S.

Download and Install R and RStudio

R can be downloaded from any of the mirror sites maintained by the Comprehensive R Archive Network ("CRAN"). I recommend the CRAN Mirror maintained by RStudio: `https://cran.rstudio.com/`. Here you will find links to download R for Windows, Mac-OS and Linux. The current version of R available for Windows is R 3.3.1 and measures at around 70MB. Follow the directions to install the starting packages for R - often called "Base R".

Even though we will not use a bare R installtion - lets take a look at it anyway. Launch R once the installation finishes. It will look similar to Figure: 2.1. This window is called the R Console. As you would note - it looks a little scary - no icons, nothing clickable, no menu even. While it is possible for us to do everything we wish to do with R just with the R Console, it really would be nice to make our experience with R a little more user friendly. Lucky for us, we have RStudio to pick up the slack here.

Once you have finished installing R, you can download and install RStudio from: `https://www.rstudio.com/products/rstudio/download/`. There again are installers available for Windows, Mac-OS and Linux. After you are done installing RStudio, you should launch RStudio to check that everything is installed and connected properly. When you launch RStudio for the first time, you will see something similar to the Figure: 2.2.

The larger part of the RStudio window is same as the R Console and this is in-fact R

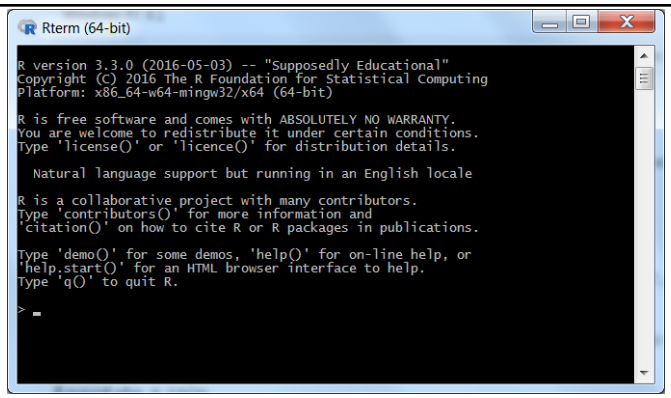


Figure 2.1: Staring Window of Base R Installation

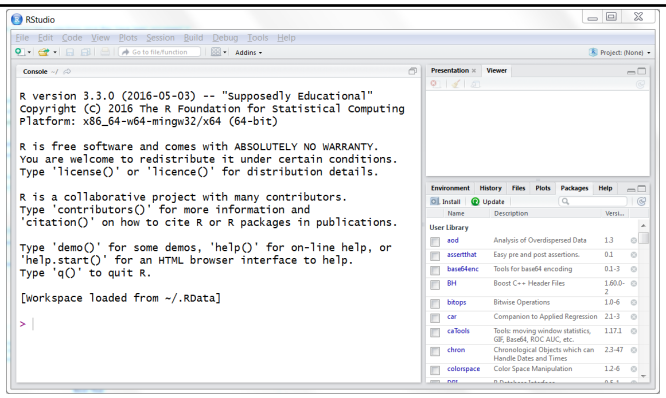


Figure 2.2: Staring Window of RStudio

running inside RStudio. When we launched R independently, this is the prompt that you we got. RStudio combines this direct access to R with other useful elements. In the bottom right, you have access to the file system, you can manage R packages (more about them later), read Help documentation and see various plots and charts that you will soon be building. In the top right, RStudio provides you with details of current R environment and a history of your recent R commands. If you are using Git repository for your files then you will see Git tab here.

Now that we have both R and RStudio set up, it is time for us to get started in making some use of it. Note that the R environment is used primarily by typing commands on the Console prompt. While RStudio provides GUI elements for many common tasks, we will emphasize corresponding command line versions which usually provide better control and flexibility compared to their GUI counterparts.

2.2 Useful Features of RStudio

Console

The console is where you can interact directly with the R system. You can write one line at a time and immediately execute it there by pressing enter. If you wish to write more than one command in one line then you can separate the commands using the semi-colon

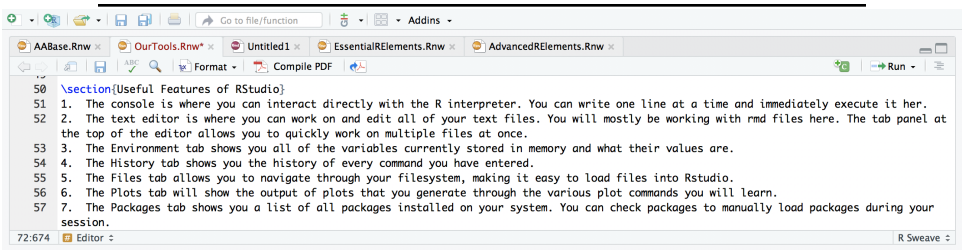


Figure 2.3: Editor Window

character. The console is essentially an R session running inside RStudio. The console shows a command prompt (usually the >) character to indicate that you can type there. You can change the prompt by changing the options setting.

```
options(prompt = "R> ")
```

I once thought it was very clever to have this as my prompt "Sanjeev Says: ". I no longer think so.

Editor

The editor is where you can work on and edit all of your files. This is a pure text editor - so no formatting of any kind. You will mostly be working with RMD files here. The tab panel at the top of the editor allows you to work on multiple files at once. The editor window is hidden when you start RStudio as you usually don't have a file to edit when you begin. Once you create/open a new file then they will open in the editor window and you can edit them. The editor window helpfully shows you the line numbers in left column and the cursor positioning in the line at the bottom left corner. The bottom right corner shows you the file type that you are working on. There are usual icons for saving a file, finding, spell check etc on the top bar. See Figure: 2.3 for an example.

Environment

The Environment tab shows you all of the variables currently stored in memory and what their values are. The tab in fact shows all the objects that currently exist in your R session. Environment tab is a very useful place to have a quick look at the state of your current R session. The environment tab has a handy broom icon for deleting all the objects that exist and clearning your environment. You can, of course, do that using a command as well - `rm` for remove or delete. The command `ls` generates a list of all the objects in the environment.

We haven't said what variables are as yet. Hold on - its coming in the next chapter.

Objects will be discussed very soon, later in this chapter

```
rm(list = ls()) #Delete every object in the environment
```

History

The History tab shows you the history of all the commands you have entered in the Console window. You can use available icons on top to search through the history or delete items from the history. Two very useful icons are for transferring commands to the console and to the source. See details in Figure: 2.4. The `To Console` allows you to select one or more commands from the history and transfers them to the Console window where you can run them by pressing Enter. The `To Source` sends the selected commands to the Editor window, to the currently open and active file.

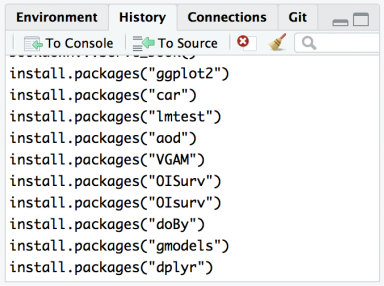


Figure 2.4: History Window

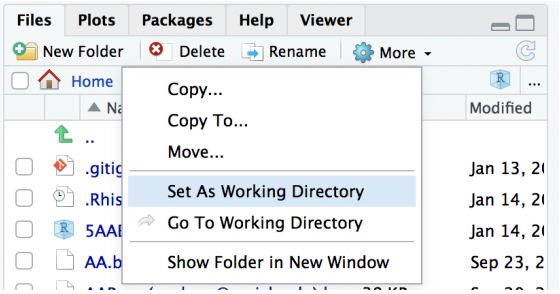


Figure 2.5: Files Window

Files

The Files tab allows you to navigate through your filesystem, making it easy to load files into Rstudio. It shows the content of the current working directory when RStudio starts. You can navigate through the file system of your machine using the available options. The top icons provide ability to create a new folder, delete selected files, rename selected files and refresh file listings. The `More` option reveals further functionalities including two useful ones: the ability to set the currently displayed directory as the working directory and the ability to get the Files tab to display the contents of the current working directory. Figure: 2.5 shows the options.

Plots

Plots made by R are Vector graphics and hence should be saved in vector formats like PNG and not bitmap formats like JPG

the Plots tab shows the plots generated by the commands typed in the Console. Figure: 2.6 show the plot generated by the command `plot(cars)` command in the Plots tab. As you can see, you can use options to zoom in/out and export the plot image as an image file or a PDF document.

Packages

The Packages tab shows you a list of all the packages installed on your system and their version number. You can check the checkbox next to the package name to manually load packages during your R session. You will also find icons for installing and updating packages. A search icon allows for searching among installed packages.

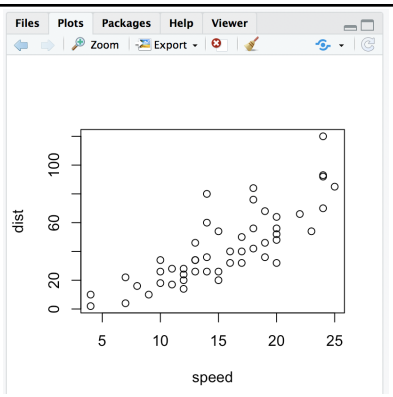


Figure 2.6: Plots Window with a Demo Plot

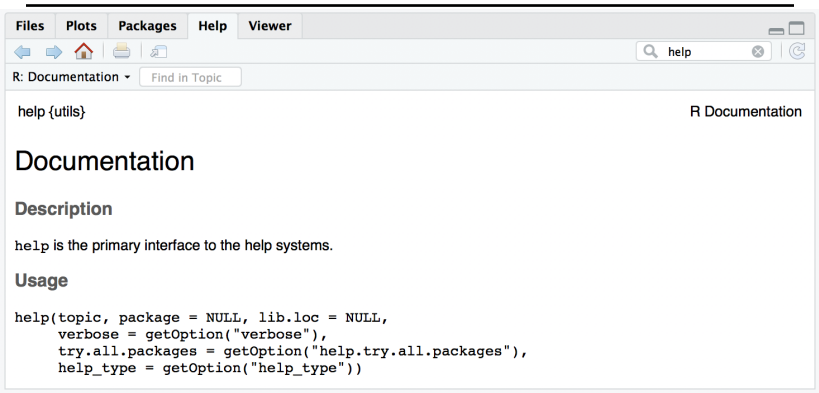


Figure 2.7: Help Windown when looking for help on "help"

Help

As the name suggests, the Help tab can be used to find Help documentation on specific keywords. You can search for a keyword in the search field for looking at corresponding help documentation. If you write a help command in the console, then the output of that will also show in the Help tab. For example, if you type the command below (looking for help on "help"), you will get the help documentation as shown in Figure: 2.7.

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

2.3 Helpful Menu Items and Keyboard Shortcuts

2.4 RMarkdown (RMD) Files

We will use extensive ise of RMarkdown files in the class. Markdown is a simple formatting syntax for authoring documents. RMarkdown allows you to create Markdown documents with embedded R code in them. RMarkdown documents can then be rendered in variety of output formats such as HTML, PDF, and MS Word documents.

RStudio streamlines the process of creating an RMarkdown file. Getting started is easy - in RStudio, go File -> New File -> R Markdown (or Alt-F-F-M). This will start a new

RMD file for you. It needs some startup information to proceed first, as shown in the image below:

Title and Author are self explanatory. As the dialog suggests, you can choose HTML, PDF of Word as default output formats. PDF and Word require you to have a local installed version of TeX or MS-Word. For our class, I recommend that you use HTML as the default output - it is device and platform independent - and capable of handling variety of rich media. We will see later how to include dynamic content like animation and user interaction in these HTML files as well.

Once you say OK, you will be taken to the default start page. This page has some content already to help you started. You should be able to see the preamble right on top that will generate your document headers and also control various options:

```
---
title: "Untitled"
author: "Dr. Sanjeev Kumar"
date: "May 5, 2018"
output: html_document
---
```

You can make changes by either hand-editng the information above or by clicking the Edit Option icon (which looks like a gear) and choosign your options. For example, the preamble for one of my document looks like the following:

```
---
title: "How To Write RMarkdown Files"
author: "Dr. Sanjeev Kumar"
date: "May 5, 2016"
output:
  html_document:
    highlight: tango
    number_sections: yes
---
```

Now you can go ahead and add content to your document. When you want to create the output, click the *Knit* button and a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. We will see later in the document how to embed R code in RMD files.

It is easy to format RMarkdown files and embed R code in them. For more details on using R Markdown see http://rmarkdown.rstudio.com.

2.5 Embedding R Code Blocks in RMD Files

Embedding Options

echo
eval
cache
label
fig.height/width

Embedding Code Blocks of Other Languages

Embedding in-line R Code

Reproducibility of Data Analysis

A key advantage of using an RMD file is that the output contains both the R code and the output. That allows for a precise matching of what steps were performed to reach a specific output. Consider the example of a complex chart made in MS-Excel - you would be hard pressed to figure out what exactly was done to create that chart. Whether the chart has some data manipulation in it, whether the chart inadvertently includes a mistake - there is no transparency - and hence you are likely to not have much trust in the chart. In the case of RMD files on the other hand, the entire process of creating the chart is right there in the form of R commands. Anybody else with the same data and the same commands will reach exactly the same output. Your analysis can be reproduced by anybody else - imporving the reliability and trustworthiness of your analysis.

We discussed "veracity" as one of the four Vs of Big Data. Doing data analytics in a way that is reproducible (e.g through an RMD file) goes a long way towards establishing confidence in the results of the analysis.

Chapter 3

Data Analysis Example

In God We Trust; All Others Must Bring Data
– *Willian Edwards Deming*

First, lets make sure that we have R/RStudio set up to work in the right directory. You can type the command `getwd` to see which directory is R/RStudio working in right now:

```
getwd()
## [1] "D:/Box Sync/2Teaching/AAMLAIBook"
```

If you wish to change the directory you want to work in (remember this is the directory where R/RStudio will save all outputs), then you can do so using the command `setwd` and specifying the directory address. For example, the command below will make `"c:\users\sankum"` the working directory.

Notice that Windows uses the backslash while R uses the forward slash in directory path. This is essentially a hangover from R's Unix days.

```
#setwd("c:/users/sankum") #Command disabled
```

Download the data file (available at: Link to Data File) and save it in the working directory for R/RStudio using the filename `nycflightsjan13.csv`. Now, we need to get the data that we want to analyze into the R environment. We will do so by importing the data file into R using the following command :

```
nycdata <- read.csv("nycflightsjan13.csv")
```

There are a lot of things going on in the command above. Here are the key elements: `"<-"` is called the **Assignment Operator** - it takes the value on its left and assigns that value to the object on its right. On the right of the assignment operator, we have the R function `read.csv()` which reads a CSV or Comma Separated Values files. On the left we have a **Data Frame** that we are calling `nycdata`.

We have used several technical terms that we should define before going forward:
Operators are part of R language and perform certain *operations*. For example, as we saw, the assignment operators assigns the value from its left side to the object on its right side.
Objects are basic building blocks of the R language that may contain both data and logic. R is an **Object Oriented Language** which means that it is organized around objects and their interactions.
Data Frame is one of various types of objects in R that is used to store data. Data frames store data in neat rows and columns (like an Excel table) which makes them easy to use. Typically, columns contain different data fields and rows contain different observations of those data fields.
Functions Function are pieces of code that perform a task and then return (or give back) a value. Functions (or their counterpart Subs - which also perform a task but do not return a value) can be identified by the `()` characters. When the function needs some information to perform the task, then that information is provided to the function by placing them inside

() characters (as in the example above of `read.csv()` function.
read.csv() is one various types of functions that perform the task of importing data into an R object. This function reads a CSV file and outputs the contents of the file in the data frame format.

So, we have used the `read.csv()` function to import data contained in the `nycflightsjan13.csv` file into the data frame named `nycdata`. Let's first peek inside this data frame using the `head()` function that shows first few lines of data.

```
head(nycdata)
```

```
##   X year month day dep_time dep_delay arr_time arr_delay carrier tailnum flight
## 1 1 2013     1   1      517         2      830        11      UA  N14228   1545
## 2 2 2013     1   1      533         4      850        20      UA  N24211   1714
## 3 3 2013     1   1      542         2      923        33      AA  N619AA   1141
## 4 4 2013     1   1      544        -1     1004       -18      B6  N804JB    725
## 5 5 2013     1   1      554        -6      812       -25      DL  N668DN    461
## 6 6 2013     1   1      554        -4      740        12      UA  N39463   1696
##   origin dest air_time distance hour minute
## 1    EWR  IAH      227     1400    5     17
## 2    LGA  IAH      227     1416    5     33
## 3    JFK  MIA      160     1089    5     42
## 4    JFK  BQN      183     1576    5     44
## 5    LGA  ATL      116      762    5     54
## 6    EWR  ORD      150      719    5     54
```

The data shows various information about all the flights departing from one of NYC airports in Jan 2013. Different data fields labels (first row of the data frame) are self explanatory.

If we want to know how many rows and columns there are in our data frame, then we can use the `nrow()` and `ncol()` function. Note that as shown below, you can run multiple R command on the same line by separating them using the end-of-command character `";"`.

```
nrow(nycdata); ncol(nycdata)
```

```
## [1] 27004
## [1] 17
```

We can calculate average arrival delay using the `mean()` function:

```
mean(nycdata$arr_delay, na.rm=TRUE)
```

```
## [1] 6.129972
```

We did two new things - we used the `"$"` character to access an column inside the data frame. Essentially, `X$Y` refers to the `Y` column in the data frame `X`. We asked the `mean()` function to calculate the average of the `arr_delay` column in `nycdata` data frame. However, there are many values in the `arr_delay` column that are missing (think of an empty cell in Excel). These missing values are coded as `NA` in R and the `mean()` function has an optional input `na.rm` which allows you to specify how you would want to consider these `NA` values. `na.rm=TRUE` tells R that it should remove (hence the term `rm`) any `na` values from the calculations.

Similar to the example above, we can calculate other descriptive statistics such as Median, Minimum, Maximum, Standard Deviation, Inter-Quartile Range etc.

```
median(nycdata$arr_delay, na.rm=TRUE); IQR(nycdata$arr_delay, na.rm=TRUE)
```

```
## [1] -3
## [1] 28
```

We can easily plot our data in R. The code for Figure 3.1 creates a histogram using the `hist()` function. We gave this function the `arr_delay` column from the data frame but to

```
hist(nycdata$arr_delay[nycdata$arr_delay < 300])
```

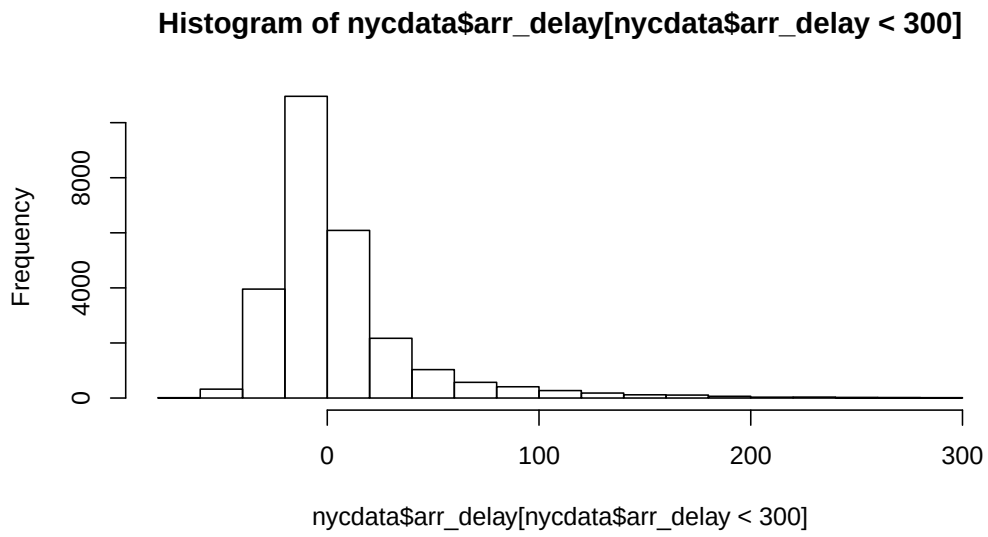


Figure 3.1: Example of a Histogram

make the plot more readable, we **filtered** the data to only show values that are less than 300. As shown above, the filtering condition is presented inside `[]` characters.

The plot shows that most of the flights that departed from NYC in Jan 2013 reached their destination without any significant delay. However, there might be differences between airlines - we would want to see average arrival delays calculated for each different airlines.

```
delaybycarrier <- tapply(nycdata$arr_delay, nycdata$carrier,
                         mean, na.rm = TRUE)
delaybycarrier
##          9E          AA          AS          B6          DL          EV
##  10.2074324   0.9823789   8.9677419   4.7171992  -4.4046512  25.1601917
##          F9          FL          HA          MQ          OO          UA
##  21.8305085   3.3179012  27.4838710   7.8837948 107.0000000   3.1755991
##          US          VX          WN          YV
##   1.4311454 -15.2802548   5.8862944  13.7692308
```

Again we did some interesting things here. We calculated the average of `arr_delay` column for each `carrier` using the `tapply()` function. We saved the output in an object named `delaybycarrier`. When we type just the object name by itself then R essentially prints out the contents of the object - which in this case consists of carrier names and corresponding average arrival delay. We can see that the carrier `VX` had the lowest arrival delay in Jan 2013 while the carrier `OO` had the highest.

To illustrate the point further, we can make a barplot using the `barplot()` function (Figure 3.2) - not much different than the *hist*() function before.

We will end our data analysis example with an example of loading a new package and using that to create a new column in the dataset. R's power and versatility comes from various extensions (called *packages*) created by volunteers. These packages need to be installed and then loaded into memory for them to be used. We will install the *dplyr* package and load it using the following commands.

```
barplot(delaybycarrier)
```

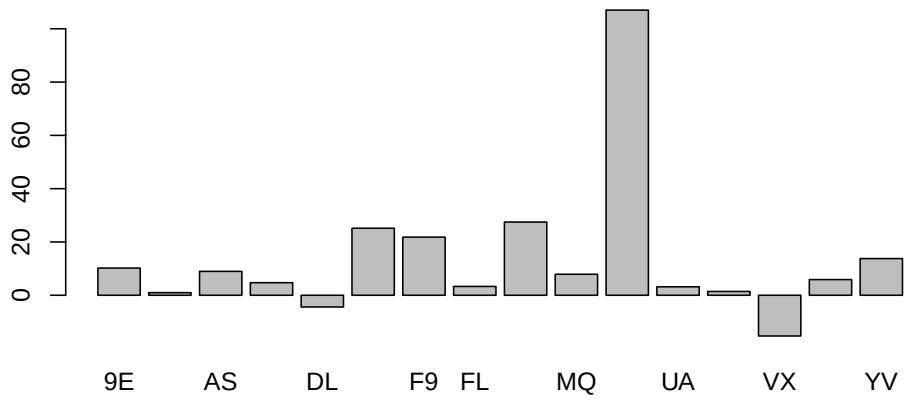


Figure 3.2: Barplot of NYC Flights Data

```
#install.packages("dplyr", repos="https://cran.rstudio.com/")
library(plyr); library(dplyr)
```

Let's say we want to calculate the speed of all the flights in the NYC Flights dataset we looked at earlier. We can use the `mutate` command in the `dplyr` package to do so.

```
nycdata <- mutate(nycdata, avgspeed = distance / air_time)
```

You can use the `head()` function we used earlier to check whether the new column has in fact been created. Note that we could have done the above task without going to `dplyr` at all.

```
nycdata$newspeed = nycdata$distance / nycdata$air_time
```

You can use either of the above two approach - all depends upon your preference. This brings us to our concluding thought of this demo. There are several (sometimes hundreds) of different ways to doing a task in R. Some are more suitable to the given context than others. However, it is up to you choose which to become expert on, which to know peripherally and which to ignore. Each person's interaction and experience with R is unique to that person and his/her choices. I hope this demo has given you an idea of the power, depth and versatility of R and has made you interested in the deep dives into different aspects of R that are about to come in following chapter.

Chapter 4

Essential R Tips & Tricks

`In God We Trust; All Others Must Bring Data`
– Willian Edwards Deming

This chapter is a collection of helpful tips and tricks that you should know. Use thus chapter as a reference - look for what you need when you need it.

4.1 Key Features of R/RStudio Ecosystem

Open Source Software

Open Source Software is a broad term for software that is developed by a community of volunteers and is available free of charge for you and I to use. R is open source with a strong and dedicated community that takes the stewardship of R very very seriously. As R is open source, the underlying computer code (the source code) is available for anyone to view and modify. All these extra eyes looking at the code usually means a more reliable, error free and malware resistant software.

Open source nature of R has both positive and negative impacts. On the positive side, the community helps extend R rapidly by building new packages with new capabilities very quickly. On the negative side, the R platform is built by experts for experts - the focus is on capability and precision - not on ease of use. This makes R a swiss army knife of that is capable of doing almost everything possible in data domain but a knife that can be difficult to learn how to use.

Extensability of R

R is desiged to be easily extended. You, me and everyone else can write new functionalities for R and have it available for everyone in the world to download and use. The R community has been writing extensions for decades now and the result is a library of thousands of packages. If there is anything data analysis related that you want to do, chances are that somebody else wanted to do it too, has already built a package for it that you can download and use.

Easy extensability of R is one of the main reasons why R remained the most capable, cutting edge data analytical platform for decades now. R community has written packages to allow you write SQL commands, C++ commands, build geo-maps, build predictive models - all within the R platform. New developments tend to get incorporated in R much faster than in competing statistical software such as Stata, SAS or SPSS.

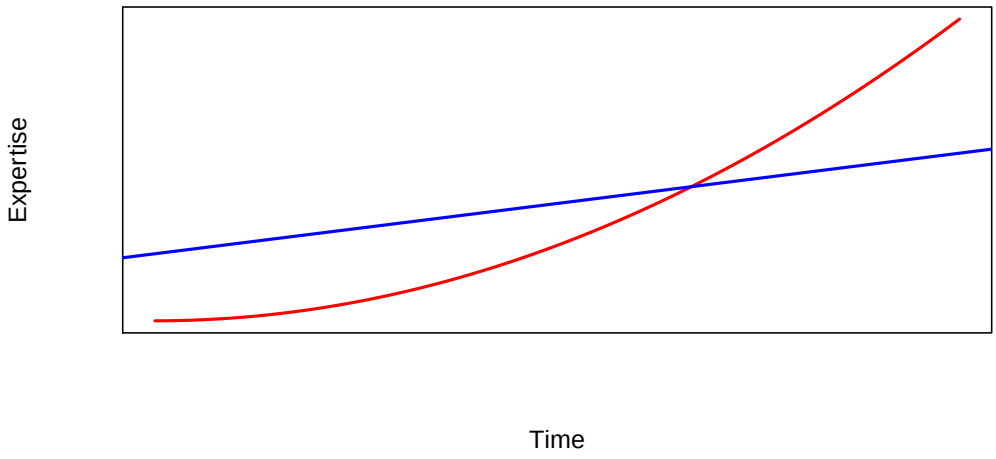


Figure 4.1: Learning Curves for R (red) and alternatives (blue)

Usability of R

R is not graphical. Much of R's capabilities are accessed through typing R commands rather than clicking buttons or accessing menus. R commands allow a level of precision and flexibility that is typically now achievable through a graphical interface. Doing complex tasks successfully in R involves writing a sequence of R commands that are closer to coding in a programming language than using a point and click software.

While R can be considered to have a steep learning curve, it should be noted that once you get through the learning curve and get used to R's syntax, you can slice through complex data analysis with much greater speed and capability than competing alternatives. In my opinion, the learning curves for R and competing graphical alternatives look like Figure: 4.1. R starts slower and has a difficult jumpstart but if you stay with it for a little bit then R handily outperforms the graphical alternatives.

BTW - the plot in Figure: 4.1 was generated using R. Here is the code for it:

```
#Code to generate the Learning Curve Plot
curve(x^2,from = 0,to = 3, xlab = "Time", ylab="Expertise",
      col = "red", lwd=2, xaxt = 'n', yaxt = 'n')
abline(a = 2,b = 1, col = "blue", lwd = 2)
```

R is NOT Verbose

In fact R is very very concise. It will not give you an output until you ask for it. This is very different than most statistical software that tend to overwhelm the user with every possible output possible. R allows you the flexibility to specifically request the levek of detail you need. By default, R will provide only the minimal amount of output.

```
> print("This statement is incomplete"
+ ) #R managed to figure that the command was incomplete
[1] "This statement is incomplete"
>
```

Figure 4.2: Line Continuation for Incomplete Commands

Object Oriented Programming

R Community's Sense of Humor

R community is a hard working one - they are continuously building new and latest packages - making the R ecosystem even better. R community is also a fun one - a clever one at the very least. For example - look at the nickname for the current version:

```
version$nickname
## [1] "Action of the Toes"
```

The R community is very helpful and will answer your questions if you post on relevant forums like StackExchange.com. However, they don't suffer fools and xpect people who ask questions to have done their homework. Over the years, the R community has developed an extensive knowledgebase that is only a google search away for you. As you go further in your exploration of R, attempt to learn from the community and hopefully one day be a bonafide member of the community.

4.2 Common Tips

R is Case Specific and Space Neutral

Everything in R is case specific. In R, upper case letters are considered completely different than their lower case counterparts. R will show no mercy if you mistype something in a difference case. However, R is pretty flexible about spaces and will usually accept multiple (or none) spaces where the default expectation is one space.

Comments and Line Continuation

You can add your comments/remarks in your R code by inserting the `#` character. Any text after the `#` character will be ignored by R during execution. It is essential that sufficient comments are added to your R code so that the code remains understandable and if you share your code with somebody else then they have a way of figuring out what you are trying to do, why and how.

R is smart enough to figure when the command is not finished and allows you to continue typing on the next line. This usually works well when there are un-finished parentheses or operators at the end. R indicates that it is expecting further inputs by showing the character `+` at the beginning of the next line. See Figure: 4.2 for an example.

Naming Conventions

I can not emphasize this enough - so special characters anywhere - not even in the directory path of your files.

Object names in R can consist of any combination of alphanumeric characters (`a-z`, `A-Z`, `0-9`) as well as underscore and period characters. Object names should not include special characters such as `?` or space. Object names must start with a letter or a period. Special care should be taken to not include one of the reserved keywords (like *function* or *help*) as object names.

Useful Keyboard Shortcuts

My quick traning method: tell yourself that you will not be touching the mouse for next one hour. You will do everything through the keyboard. First few minutes will be hard. By the end of the hour, you will be hooked.

I find keyboard shortcuts to be a great productivity aid. Once you use then enough and the muscle memory builds in, then the process becomes almost automatic and really fast.

4.3 Common Issues, Pitfalls, Bugs, Obstacles and Their Solutions

This section is my master list of common issues that students encounter when they start working with R. You may want to not read this in full right now - treat this as a reference section. When you run into an issue, come by here and see if I have a solution for you.

I have an older R installtion. How do I upgrade it?

The R version at the time of writing this document is: 3.6.1. You might have an old, not updated version of R in your machine. You can check the current version of your R by running the `version` command.

```
version
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          3
## minor          6.1
## year           2019
## month          07
## day            05
## svn rev        76782
## language       R
## version.string R version 3.6.1 (2019-07-05)
## nickname       Action of the Toes
version$platform
## [1] "x86_64-w64-mingw32"
```

If you have an older version and if you are in a Windows machine, then upgrading your R installation is easy with the `installr` package. The `installr` package has the handy `updateR` command that will automatically update your R installation to the latest one. A sample code to accomplish this is below:

```
#Sample code, not being evaluated here
install.packages("installr") # install the installr package
library(installr)
updateR() # updating your R version
version # command for checking your current R version
```

I am not going to insult your intelligence by asking you to make sure that you have a live Internet connection!

If you happen to be in a Mac or Linux machine, then you would want to download the current version of R and install. This process is similar to installing a fresh new installation of R as discussion in Section: 2.1.

Help - I am not being able to install any packages!

If you are unable to install R packages, there are a few things we can try. First thing to check

would be your anti-virus software. Several anti-virus software balk at allowing R to perform tasks that may be interpreted as a security risk (for example - downloading anf installing packages from the Internet). You can try to work around this by disabling/uninstalling your anti-virus or adding R to the exceptions list in the anti-virus.

You may also want to check whether you have administrator access in the machine you are working on. Several essential tasks in R require the user to have administrator access.

Lastly, R may be attempting to install packages into a directory that it does not have write permissions on. This can sometimes happen if you are on a computer on a network with mapped drives. You can work around this problem by asking R to install your packages in a safe, editable directory where you have write access. R installs all packages in a library directory - you can find out the current location of the library directory with the `.libPaths` command. You can use the `.libPaths` command to then add a directory location as your personal library location so that R can then install packages in that directory. A sample code to accomplish this is shown below:

```
.libPaths() #Shows the current library path
## [1] "D:/Home/R/win-library/3.6"          "C:/Program Files/R/R-3.6.1/library"
# I have two folders in my library path currently
# You can add your own library folder to the path
#libPaths( c( .libPaths(), "C:/Whatever/Whatever...") )
```

Last command commented as C:/Whatever does not exist obviously - so would have resulted in an error.

Why is the edit command not working for me?

One of the first R commands that gives trouble is the `edit` command. As the name suggests, `edit` is used for editing data - typically it shows the data to be edited in a table format with editable cells. This specific command (and its cousin, the `fix` command) seems to be very sensitive to the keyboard layouts, especially on Apple machines. This problem is hardware oriented and may prove difficult to solve. I usually recommend to ignore these two commands and take care of your data editing needs using alternative methods. If you were using `edit` for only viewing the data, then `View` is a good alternative for that.

Not really - see below

What's wrong with the View command?

I **hate** the command `View` . Here are some of the reasons why I ask my students to not rely on the `View` command. First, note the uppercase *V* in `View`. Its innocuous enough that many students miss that detail and type the command as `view` and of course it doesn't work then. Second, `View` essentially shows you the whole data. That might seem like a good idea at first - but note that we are in the *Big Data* world. Looking at the whole data is not productive. If you want to see a few rows of data then `head` or `tail` are good ideas. Of course, if you are interesed in the *structure* of your dataset then `str` is a much better way to go. See below for examples.

```
head(USArrests) #Shows first 6 rows
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
head(USArrests, n = 3) #You can adjust # of rows
##         Murder Assault UrbanPop Rape
```

```
## Alabama   13.2     236       58 21.2
## Alaska    10.0     263       48 44.5
## Arizona    8.1     294       80 31.0
tail(USArrests) #Like head, but for last rows
##               Murder Assault UrbanPop Rape
## Vermont          2.2      48       32 11.2
## Virginia         8.5     156       63 20.7
## Washington       4.0     145       73 26.2
## West Virginia    5.7      81       39  9.3
## Wisconsin        2.6      53       66 10.8
## Wyoming          6.8     161       60 15.6
str(USArrests) #Structure of the dataset
## 'data.frame': 50 obs. of  4 variables:
##  $ Murder  : num  13.2 10 8.1 8.8 9 7.9 3.3 5.9 15.4 17.4 ...
##  $ Assault : int  236 263 294 190 276 204 110 238 335 211 ...
##  $ UrbanPop: int  58 48 80 50 91 78 77 72 80 60 ...
##  $ Rape    : num  21.2 44.5 31 19.5 40.6 38.7 11.1 15.8 31.9 25.8 ...
```

In case you are wondering, `USArrests` is a dataset of number of arrests per 100,000 population for several violent crimes for all 50 states in the USA. It is one of the many built-in datasets in the base R.

Argh... My R is SO SLOW!

Guess how manay MacBooks equal the same processing power as the Apollo rockets that went to the Moon and came back? Answer is less than 2. You have half of a sparefaring rocket in front of you - and what do you do with it? Email, chat, browse. It is made for much more and its time you actually started using it for the grand ambitions it was built for. make it sweat - its a good thing.

This is not a bad problem to have. You are finally making your laptop work hard - extracting some value from the couple grands you dropped on a machine when you came to school.

Well - waiting and waiting for your code to run does get old pretty soon. So what can you do? Obvious step is to upgrade your machine - some additional RAM goes a long way. Most of the performance issue in R comes from the fact that R loads all the data into the RAM of your machine so if there isn't enough RAM to go around then things grind to a halt. Use common sense computing practices to optimize RAM usage:

- Close all programs you are not currently using - especially Chrome tabs. Each Chrome tab eats RAM - and you may have dozens of them open.
- It is a good idea to restart your machine occassionally.
- When you begin a new R project, you would want to clear out all the old data from your session. You can use the broom icon on the Environment tab of RStudio to do that. Or use the following command: `rm(list = ls())`

Help - my code can't find my data file!

This is the most common bug my students face in the first few weeks of my classes. You typically get an error message like *R can not establish a connection*. This happens because the R session really could not find your data file. Remember - it is looking for the data file in a very specific place:

- If you are running your code in the R Console, then it is looking in the current working diretory. If you have not specifically changes your working directory, then your current working directory is same as your Home directory.
- If you are running your code in a .R or .Rmd file, then it is looking in the directory where the .R or .Rmd file is kept.
- If you are working in a RStudio Project, then it is looking in the project directory.

You would want to make sure that your data file is in the appropriate directory. If you

are not sure what is your working directory, then you can use the commands: `getwd` and `path.expand()` . The character `tilde` is shortcut for the Home directory.

```
getwd()
## [1] "D:/Box Sync/2Teaching/AAMLAIBook"
path.expand("~")
## [1] "D:/Home"
```

Chapter 5

Essential R Language Elements

In God We Trust; All Others Must Bring Data
– *Willian Edwards Deming*

In this chapter we will start playing with some basic R functionalities and start getting comfortable with typing commands in the R console. All code examples you see in the chapter are opportunities for you to type them in R Console yourself and see whether you get the same output. There is no better way to find your way around R than to get your hands dirty - and no better time than now. So lets get started.

5.1 Arithmatic Operators

You can use R as a glorified calculator. All arithmatic calculations can be easily performed using **Arithmatic Operators** like + (addition), - (subtraction), * (multiplication), / (division) and ^ (exponent).

```
10  +    20  #As you can see - spaces don't matter, mostly.
## [1] 30
20/5
## [1] 4
16 ^ 0.5  # Usual Arithmatic Operators: +, -, * , /, ^
## [1] 4
```

You would note that the outputs to the commands have this `[1]` before them. The number indicates how many values there are in the output. Since the commands above lead to a single output value - you see the `[1]` there.

When you have complicated Arithmatic Expressions, R follows usual **Arithmatic Operator Precedance** : Brackets, Exponent, Division and Multiplication, Addition and Subtraction - in that order. Go Left to Right for the same precedance. Of course it is preferable to put in enough paratheses so that you are not relying on R's operator precedance for accurate execution of your commands.

```
5 + 8 / 2 * 4 - 3 # First * then / then + then -
## [1] 18
# You should include enough parantheses
(5 + 8) /((2 * 4) - 3)
## [1] 2.6
```

Two arithmatic operators that many are not familiar with are: **Integer Division** and

Remainder operators. The `Integer Division` operator: `%/%`, provides just the quotient for a division operation while the `remainder operator`: `%%`, provides just the remainder for a division operation.

```
10 %/% 3 # Integer division - only gives the quotient as the output
## [1] 3
10 %% 3 # The counterpart to %/%, gives remainder as the output
## [1] 1
```

5.2 Logical Operators

Note the upper case format for the two keywords. The letters T and F are also acceptable values as shorthand for TRUE and FALSE

Just as arithmatic expressions evaluate to a numerical result, logical expressions evaluate to a logical result - either `TRUE` or `FALSE`. Logical expressions can be created using **Logical Operators** : equal to (==), greate than (>), less than (<), greater than or equal to (>=), less than or equal to (<=) and finally, not equal to (!=).

```
10 > 20 #Should evaluate to FALSE
## [1] FALSE
12.37 <= 23.66 #Should evaluate to TRUE
## [1] TRUE
(10/3) == (30/9) #Should evaluate to TRUE
## [1] TRUE
```

Logical values can be combined using **AND** and **OR** constructs. When combining two logical values using `OR`, the resulting value is `TRUE` if any of the two values are `TRUE`. When combining two logical values using `AND`, the resulting value is `TRUE` only of both the values are `TRUE`. The command for `AND` is `&` while the command for `OR` is the `|` character.

```
(10 > 20) | (12.37 <= 23.66) #Should evaluate to TRUE
## [1] TRUE
(10 > 20) & (12.37 <= 23.66) #Should evaluate to FALSE
## [1] FALSE
```

You can use **NOT** operator to convert a `TRUE` value to `FALSE` and vice versa - the corresponding command is `!` character. You can use logical operators to compare non-numeric values as well.

```
!TRUE #Should evaluate to FALSE
## [1] FALSE
"Dog" > "Cat" #As everybody know, this is TRUE
## [1] TRUE
```

We have not discussed different data types yet. Values enclosed in quote marks are text values, typically called Strings

As the example above shows, comparing non-numeric values can result in interesting results. When two strings are compared, they are essentially compared one character at a time. In the example above, the character C is compared with character D. The comparison is based on the Unicode number associated with the characters. As D has a higher Unicode number than C, "Dog" is considered higher than "Cat".

Interesting thought experiment:
1000 > 50: True or False?
"1000" > "50": True or False?

5.3 Variables

R keeps all data in **Variables** . Variables can be thought of as a space in computer's memory where R can store data. Once we have created a variable, we can then use the

variable name to access, manipulate and work with the data kept in the variable. Variables are created using the **Assignment Operator** written as `<-` . The arrow indicates the direction of assignment. Operators also exist to do assignment in the opposite direction (->) and do the same operation as assignment operator. R also supports the assignment operator command used by many other programming languages (=) but it is customary to use the right-to-left version of assignment operator.

I find the R version of assignment operator to be much more intuitive as the direction of value assignment is clearly specified. The = charater usually confuses beginners into thinking that we are "equating" values instead of the correct interpretation of "assigning" value from one side to the other.

```
revenue <- 100 #Typical way of creating a variable and assigning value
# Above is the recommended approach
revenue = 100 #Alternate way of doing same as above
#100 -> revenue #Reverse direction assignment
revenue <- revenue + 300 #Performing calculations
print(revenue) #Printing/showing the value inside the variable
## [1] 400
```

The first three line of the code above should be read as the value 100 is assigned to the variable *revenue*. In the fourth line, we add 300 to the current value of variable *revenue* and the resulting value (in this case 400) is then assigned to the variable *revenue*. So the value in the variable *revenue* is now 400, which is shown as output in the last line.

A numeric variable like *revenue* above can be used just like a number to perform all usual arithmetic operations.

5.4 Variable Data Types

Variables can store different types of data. We saw integer numerical data in the examples above. Other common data types include floating point numbers (fractional numbers), characters data (i.e. a single character), text data (called strings - as in string of characters) and logical data (true or false). We can find the data type of a variable by using the `class` command.

```
numvalue <- 20.56 #Creating a numeric variable
class(numvalue) #Finding data type of numvalue variable
## [1] "numeric"
```

Numeric Data

Variables of numeric data type can store both integers and floating point values. Unlike many programming languages, R does not have various different kinds of integers and floating point numbers defined to optimize memory usage.

String Data

String or text data is written within quote marks (" ") to differentiate them from text that represent things like object names. An empty string (a text data with no characters) is usually represented by two quote marks with nothing within - like "".

Text is nothing but a string of characters, hence the name String

```
day.name <- "Sunday" #Assigns the string Sunday to variable day.name
someday <- "" #Empty string assigned to variable someday
```

String is a very useful data type as all other types of data can be saved as a string. For example - we can save numeric data ("23.37"), date data ("12/27/1976"), logical data ("TRUE") etc. all as string. R has several useful commands for working with string data. You can connect two strings together using the **String Concatenation** function `paste` . You can calculate the number of characters in a string using the `nchar` function.

Many other strings related commands are discussed later

```
nchar(day.name); nchar(someday)
## [1] 6
## [1] 0
message <- paste(day.name, " is a great day!"); message
## [1] "Sunday  is a great day!"
```

Logical Data

Variables can hold logical values using the two keywords `TRUE` and `FALSE`. Logical values typically result from the use of logical operators such as the equality operator `==`. Note the two = signs there which differetiates it from the version of assignment operator we saw before.

```
4 == 6 #Is 4 equal to 6, result FALSE
## [1] FALSE
someday == "Tuesday" #Again result shoukd be FALSE
## [1] FALSE
```

Converting Between Data Types

Its easy to convert variables from one data type to another. Conversion functions usually take the form of `as.xxx` where `xxx` is the desired datatype. For example: `as.string` converts to a String, `as.numeric` converts to a numeric data type and so on.

```
x <- "12.36" #Creating a String variable
y <- as.numeric(x); y #Converting to a numeric value and displaying
## [1] 12.36
```

5.5 Vectors

Variables store a single value. If you need to store multiple values then you need a **Vector** . Vectors can be created in the same was a variables are created, except that we need to assign multiple values to a vector. A handy function for combining multiple values in one vector is the `combine` function written as `c` .

```
grades <- c(3.2, 3,1, 2.7, 3.9, 4.0) #Grades of five students
class(grades) #A vector with numeric values
## [1] "numeric"
```

The code above created a vector named `grades` that has 5 `elements`. Vectors are very useful because R provides several easy ways to interact with different elements of a vector. We can access individual elements of a vector using the `[ ]` notation - called **Logical Indexing** . We can use the `length` function to find out how many elements there are in the vector.

```
grades[2] #Gets the second element of the vector
## [1] 3
grades[4] <- 3 #Changes 4th element of the vector to 3
length(grades) #Gets the length (number of elements) of the vector
## [1] 6
grades - 0.1 #Adds 2 to *each* element of the vector
```

```
## [1] 3.1 2.9 0.9 2.9 3.8 3.9
grades #Displays current elements of the vector
## [1] 3.2 3.0 1.0 3.0 3.9 4.0
```

Vectors are also useful for doing element-by-element calculations. For example, if we have another vector for number of hours of work for the week, we can calculate number of hours used for work and sleep as follows:

```
work.per.day <- c(9, 11, 10, 8, 6, 3, 2) #Create new vector
#work.and.sleep <- sleep.per.day + work.per.day
#Added two vectors element by element
#print(work.and.sleep) #Show added values
```

It is easy to access data elements inside a vector. All elements are assigned an index number - unlike many programming languages, R starts counting with 1 rather than 0.

```
#wed.hours <- work.per.day[3] #Extracts third element
#work.per.day[7] <- 6 #Changes 7th element of the vector
```

Functions `edit()` and `fix` can be used to edit existing vectors. Number of elements in a vector can be calculated using the `length()` function.

```
#length(work.per.day)
```

5.6 Factors

5.7 Using Built-In Functions

R includes many `functions`. Functions take some values as inputs (often called `arguments`), perform some calculation and return the result. For example *sqrt*(), the `square root function` takes a value and returns its square root.

```
sqrt(100) #Calculate square root of 100
## [1] 10
```

R includes perhaps thousands of functions for different tasks. Some functions can take several arguments with many of them being optional. Such optional arguments typically have a default value that is used in case a value is not provided for that argument. When supplying several arguments, it is a good practice to used `named arguments` as shown below.

```
#Calling functions with name arguments
round(x = 1.23456789, digits = 4)
## [1] 1.2346
#Arguments passed in order, withour names
round(1.23456789, 4)
## [1] 1.2346
#Using default values for optional arguments
round(1.23456789)
## [1] 1
```

The first line above shows running (or `calling`) the function *round* with explicitly named arguments. x represents the number to be rounded and *digits* represents how many digits after the decimal point should the rounding be done for. We could have called the function without named arguments (like in the second line above) but then we would need to provide all the arguments in the exact order needed. As it is easy to mix-up the order,

it is recommended that named arguments are used when multiple arguments are passed to a function.

The third line in the code above shows what happens when an optional argument is not provided to a function. As we have not specified the number of digits, the function uses the default value of the argument (which happens to be 0 in this case). As a result, the functions rounds the number to an integer.

Your R is only as good as your R Packages - so lets figure that first how to install a package - you can do through RStudio GUI - or use the command below. Note the quote marks around the package name - which, like most other things in R, are case sensitive.

```
install.packages("ggplot2")
```

Installing is only the first step - that brings the package to your local machine but does not load it into the current R session. To do so, you can use the `library` command. You can use the `detach` commans to unload a package from the current R session. There are several thousand packages in R - waiting for us to explore

```
library(ggplot2)
detach(package:ggplot2)
```

You typically work in a directory during a R session. You can find current current working directory or set working directory to a directory of your choice. When setting working directory to the desired location, in Windows use / or \\ instead of \ character as the separator character.

```
getwd()
## [1] "D:/Box Sync/2Teaching/AAMLAIBook"
```

```
setwd("Enter Directory Addess Here")
```

You usually have a **home directory** defined for your R installation. When you start R, your R session will usually start in this home directory. Home directory is usually referred using the ~ character. You can find out the directory assigned the `Home` status using the command `path.expand`

```
path.expand("~")
## [1] "D:/Home"
```

First thing - how to get help when you need it. For example: What the hell is a Vector?

```
#Output supressed for brevity
help("vector") #default approach, note the quote marks
?"vector" #or this simple command works too
example("barplot") #You can also look up examples
```

As you work with R, you will create Objects. You can get a list of current objects using the `objects` command. You can delete objects using the `rm` command (rm is short for remove). BTW - Check of Environment Tab in RStudio - you can see that R/RStudio is keeping track of your objects and their values

```
objects()
##  [1] "day.name"       "delaybycarrier" "grades"         "message"
##  [5] "numvalue"       "nycdata"        "revenue"        "someday"
##  [9] "work.per.day"   "x"              "y"
#Don't like a cluttered workspace, delete all objects by
rm(list = ls())  #ls() gives a list of all the objects
```

When you are done with your current R session, you can quit using the `q` command.

You should save your current session first though.

```
#Commands only for demo, not evaluated.
save.image(file = "FileName.RData")
q()
```

Chapter 6

Advanced R Language Elements

In God We Trust; All Others Must Bring Data
– *Willian Edwards Deming*

6.1 apply functions

6.2 If Statements

6.3 Loops

6.4 R Utilities

In this section, I am collecting all the random R stuff that is useful but not easily categorizable.

Calculating Execution Time

You would often want to calculate how much it takes for R to do something. You can do so by using the command `system.time` , which returns the time takesn by R to execute the argument to the function.

```
#Time taken to delete all objects in memory
system.time(rm(list = ls()))
```

```
##    user  system elapsed
##       0       0       0
```

Part II

Data Exploration and Visualization

Chapter 7

Essential Data Management

```
In God We Trust; All Others Must Bring Data
```
– *Willian Edwards Deming*

In this chapter we will look at how to store and work with diffrent types of data.

7.1 Importing Data into R

7.2 Exporting Data from R

7.3 Cleaning and Manipulating Data

7.4 Using Random Numbers and Sampling

7.5 plyr and dplyr Packages

Chapter 8

Descriptive Statistics

`The average human has one breast and one testicle.`
– Des MacHale

8.1 Summary Statistics

Chapter 9

Data Visualization in R

> If you can't explain it simply, you don't understand it well enough.
> – *Albert Einstein*

Part III

Traditional Statistical Modeling

Chapter 10

Linear Regression

In God We Trust; All Others Must Bring Data
– *Willian Edwards Deming*

Now we get down to the business of actually establishing relationships between variables. As the chapter name suggests, we will first explore linear relationships between variables.

10.1 Simple Linear Regression

Lets start with a simple case of two variables - the departure delay and arrival delay variables from the NYC flights data we have seen before.

```
nyc <- read.csv("nycflightsjan13.csv")
```

```
attach(nyc)
```

```
cor(arr_delay, dep_delay, use="complete.obs")
## [1] 0.9163321
```

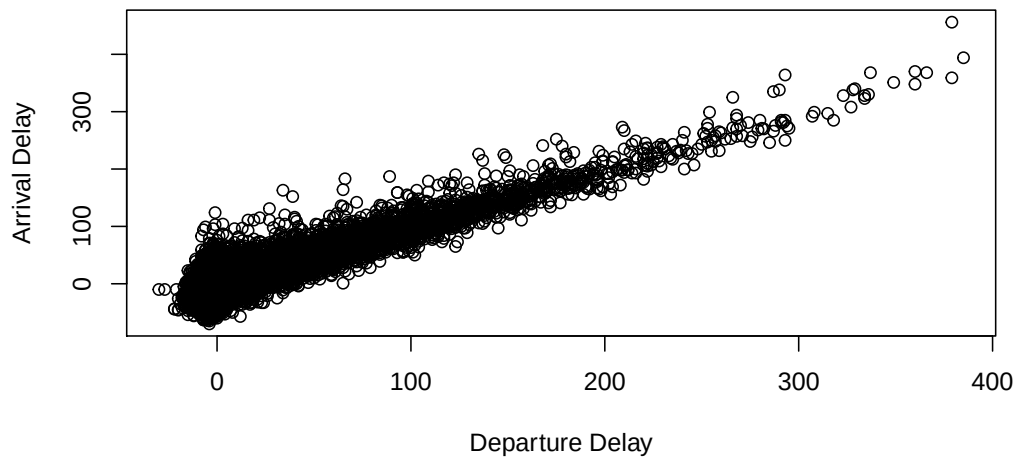
Clearly the two variables are heavily correlated with each other. We can confirm this understanding by taking a quick graphical look at these variables. Figure 10.1 shows the plot of Arrival Delay vs Departure Delay.

What we essentially want to do is to draw a line through the Figure: 10.1 such that the line represents all the scatter points well. There are several ways of drawing this line. The most popular method is called OLS (Ordinary Least Square) that draws a line such that the sum of the square of distances (called **residuals** between the line and all the points is minimized. This method is commonly known as linear regression.

We can now formalize the relationship between the two variables by running a **Linear Regression**. Recall that a linear regression essentially involves modeling the relationship between a predictor variable (x) and a response variable (y) as follows: $y_i = \beta_0 + \beta_1 x_i + \epsilon_i$. This is similar to plotting a line with a y intercept of β_0 and the β_1 as the slope of the line. Through OLS Linear Regression we are trying to estimate the values of βs. Further, to ensure that the analysis is robust and applicable, we are also looking to check whether a) the model itself is statistically significant and b) the coefficients are statistically significant.

The term **Statistically Significant** has a specific meaning here. It means that we can reject the null hypothesis of no effect with more than 95% (or any other level you choose, the default is 95%) confidence. That means that the probability that what we are observing is because of pure chance and randomness is less than 5%. Note that the probability of the analysis being flat out wrong is non-zero. That's why we need to be careful in interpreting the results of a regression analysis.

```
plot(x=dep_delay[dep_delay < 400], y=arr_delay[dep_delay < 400],
     xlab="Departure Delay", ylab="Arrival Delay")
```




```
# Focusing only on departure delay < 400 mins for more readable plot
```

Figure 10.1: Scatter plot of Arrival Delay vs Departure Delay

We can run the linear regression using the `lm()` command.

```
m <- lm(arr_delay ~ dep_delay)
```

We have saved the result of the model in the object `m`. We can now extract relevant information from this object.

```
coef(m) #provides coefficients
```

```
## (Intercept)   dep_delay
##   -4.057010    1.020178
```

```
confint(m) #provides confidence interval for the coefficients
```

```
##                 2.5 %    97.5 %
## (Intercept) -4.259533 -3.854486
## dep_delay    1.014800  1.025556
```

Now that we know the coefficients, we can now draw the fitted line that models the relationship between Arrival Delay and Departure Delay as shown in Figure 10.2. This is the same line we described in our earlier discussion of OLS - this line minimizes the sum of the squares of the distance between all the points and this line.

As you can see that the line is not a perfect fit for all the points. The model provides a handy statistics called the Coefficient of Determination or the R-Square to check just how good of a fit the line actually is. To get R-Square and other useful information, we can use the command (summary) provides a good balanced amount of details.

```
summary(m)
```

```
##
## Call:
```

```
plot(x=dep_delay[dep_delay < 400], y=arr_delay[dep_delay < 400],
     xlab="Departure Delay", ylab="Arrival Delay")
abline(coef(m), lwd=2, col="red")
```

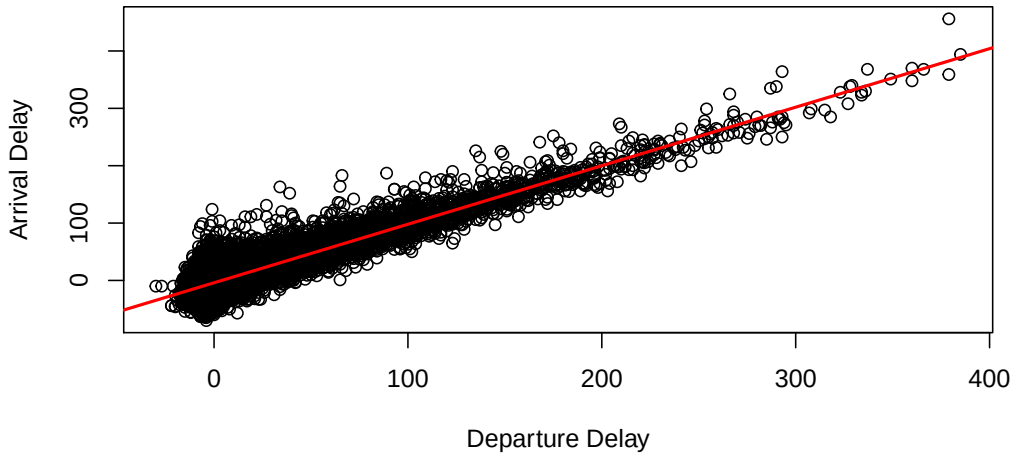


Figure 10.2: Plotting Fitted Values of Regression Model

```
## lm(formula = arr_delay ~ dep_delay)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -65.185  -9.903  -1.024   9.057 132.371
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.057010   0.103326  -39.26   <2e-16 ***
## dep_delay    1.020178   0.002744  371.80   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.19 on 26396 degrees of freedom
##   (606 observations deleted due to missingness)
## Multiple R-squared:  0.8397,Adjusted R-squared:  0.8397
## F-statistic: 1.382e+05 on 1 and 26396 DF,  p-value: < 2.2e-16
```

Important things to note here is that the model itself has a **p-value** of close to zero. p-value can be interpreted as the probability of something happening by chance, by randomness. So first thing we see is that the probability of us getting this result by chance is virtually nil. So that's a good start.

Now we see that the p-value for the coefficient of Departure Delay (or β_1) in our model is again close to zero. This means that the probability that the predictor (in this case Departure Delay) has no impact (in other words $\beta_1 = 0$) on the response (in this case Arrival Delay) is close to zero as well. So we can conclude that Departure Delay has a

statistically significant relationship with Arrival Delay. As the value of the coefficient is 1.02, we can conclude that, on an average, each minute of Departure Delay increases the Arrival Delay by 1.02 minutes.

Finally, the model has an R-Square of 0.8397. This can be interpreted as a goodness-of-fit indicator. We can say that the model explains 83.97% of the variance in the response variable. The R-Square value is susceptible to number of predictor variables - adding more predictor variables, even if they are not significant, increases the R-Square value. To correct for this over-estimation, the Adjusted R-Square can be used instead. As the model summary shows, the Adjusted R-Square value is same as R-Square value as we have only one predictor variable right now.

Multiple Regression

The basic concept of OLS Linear Regression can be easily extended to more than one predictor variables. For illustration, lets look at the following dataset that has measurement of Systolic Blood Pressure, Age (in years) and Weight (in lbs) for patients.

```
bp <- read.csv("bp.csv")
```

We can consider that Blood Pressure of a patient is a function of Age of patient and Weight of patient. So we can specify the following linear regression model.

```
bpmodel <- lm(bp ~ age + weight, data=bp)
summary(bpmodel)
```

```
##
## Call:
## lm(formula = bp ~ age + weight, data = bp)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.3233  -3.6146   0.1924   3.8740  12.0820
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 42.27980    4.75515   8.891 2.38e-16 ***
## age          0.94348    0.08792  10.731  < 2e-16 ***
## weight       0.25135    0.04526   5.553 8.16e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.457 on 217 degrees of freedom
## Multiple R-squared:  0.8394,Adjusted R-squared:  0.8379
## F-statistic: 567.1 on 2 and 217 DF,  p-value: < 2.2e-16
```

We can see in the model summary results that the model itself - our argument that a patient's age and weight have a significant relationship with patient's blood pressure - is indeed statistically significant (model p-value < 0.05). The coefficients of our two predictor variables are statistically significant as well with p-value for both coefficients < 0.05. We can interpret that for each year's increase in age, blood pressure rises by 0.94 points. Further, for each lbs increase in weight, blood pressure increases by 0.25. These two predictor variables explain about 84% of the variance in blood pressure as shown by the R-Square value.

Prediction Based on Regression Model

Once we have a model, we can use the model to predict new values. If, for example, a person is of 60 years age and 140 lb weight, we can predict corresponding blood pressure. We can get either a point prediction or an interval prediction corresponding to a confidence interval.

```
newvalue <- data.frame(age=50, weight=150)
#A Point Prediction
predict(bpmodel, newdata=newvalue)
##        1
## 127.1567
#Confidence Interval
predict(bpmodel, newdata=newvalue, interval="confidence")
##        fit      lwr      upr
## 1 127.1567 124.7148 129.5985
#Prediction Interval
predict(bpmodel, newdata=newvalue, interval="prediction")
##        fit      lwr      upr
## 1 127.1567 116.1275 138.1858
```

As the results show, a person with age 50 years and weight 150 years is predicted to have a blood pressure value of 127.16. This is the point estimate. The 95% confidence interval for the mean blood pressure value is 124.71 - 129.60. The 95% prediction interval for the point estimate is 116.13 - 138.19.

10.2 Regression Diagnostics

OLS Regression depends on the underlying data following several assumptions. If these conditions are not met then regression results need to be further investigated as they may not be accurate. Following are the key requirements:

Normality

Like much of Statistics, OLS works best with data that follows Normal Distribution. Specifically, linear regression assumes that the residuals are normally distributed. The residuals are assumed to be independent and with a mean of zero.

We can plot residuals to check their normality. See Figure: 10.3 for four such plots. First plot is a histogram of residuals - looks pretty well distributed for our case. Second plot is a Q-Q Plot of residuals - again we see that residuals stick to the normal line very well. Plot 3 shows residuals against fitted values while Plot 4 shows Cook's Distance. Visual inspection of these charts does not show specific patterns that may violate normality of residuals. We will use Cook's Distance plot in our discussion of outliers in coming sections.

We can also use the Shapiro-Wilk Normality test as shown below:

```
shapiro.test(residuals(bpmodel))
##
##  Shapiro-Wilk normality test
##
## data:  residuals(bpmodel)
## W = 0.9934, p-value = 0.4379
```

```
par(mfrow=c(2,2))
hist(x = residuals(bpmodel), xlab = "Value of residual", main="")
plot(x=bpmodel, which=2) #QQ Plot
plot(x=bpmodel, which=1) #Residuals vs Fitted
plot(x=bpmodel, which=4) #Cook's Distance
```

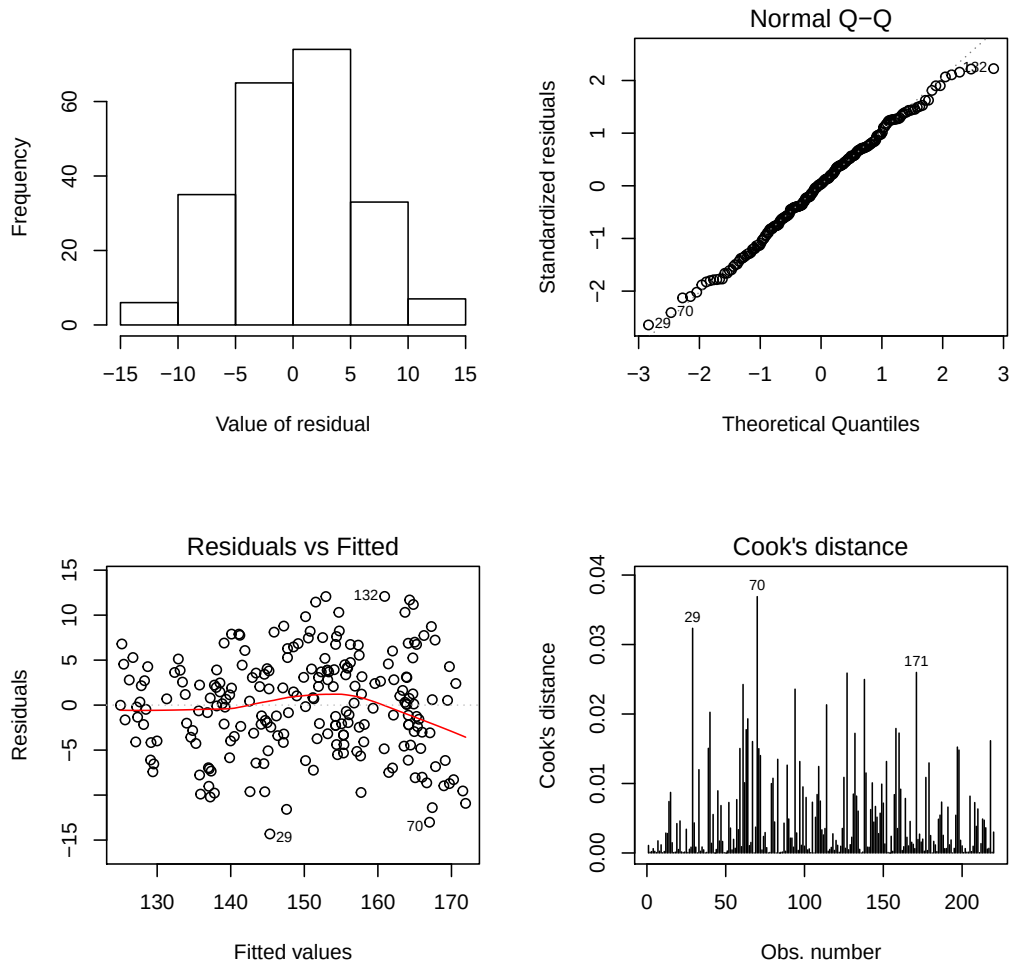


Figure 10.3: Histogram of Residuals

We can see that the null hypothesis of normality was not rejected as the p-value > 0.05. So we can conclude that the assumption of normality is satisfied in our model.

Linearity

As the name suggests, we are only exploring linear relationships between predictors and the response variable. A good graphical way to explore linearity is to check the plots created by the `residualPlots()` function in the `car` package. As Figure: 10.4 suggests, weight seems to have a straight linear relationship with. Age, however, shows some curvature that can be further explored.

It is important to note that OLS only needs residuals to be normal. Predictor variables need not be distributed normally. This allows us to do nearly limitless data transformation (taking square, cube, square-root, log etc.) and still meet the linearity assumption. For example, we could very well have run a regression analysis with log transformed values of age and weight as predictor variables.

```
newbp <- lm(bp ~ log(age) + log(weight), data=bp)
summary(newbp)
```

```
##
## Call:
## lm(formula = bp ~ log(age) + log(weight), data = bp)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.0462  -3.8825   0.4448   3.5703  12.0816
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -338.664     28.593 -11.844  < 2e-16 ***
## log(age)      57.062      5.272  10.823  < 2e-16 ***
## log(weight)   48.124      8.727   5.515 9.89e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.393 on 217 degrees of freedom
## Multiple R-squared:  0.8432,Adjusted R-squared:  0.8417
## F-statistic: 583.3 on 2 and 217 DF,  p-value: < 2.2e-16
```

Key here is to build a model that is theoretically consistent. Running a regression analysis and getting a bunch of coefficients is the easy part - the hard part is to design a model in the first place that makes sense. As they say - Correlation is not Causation. Just because a model is statistically significant, it does not mean that the model is also correct.

Homegeniety of Variance

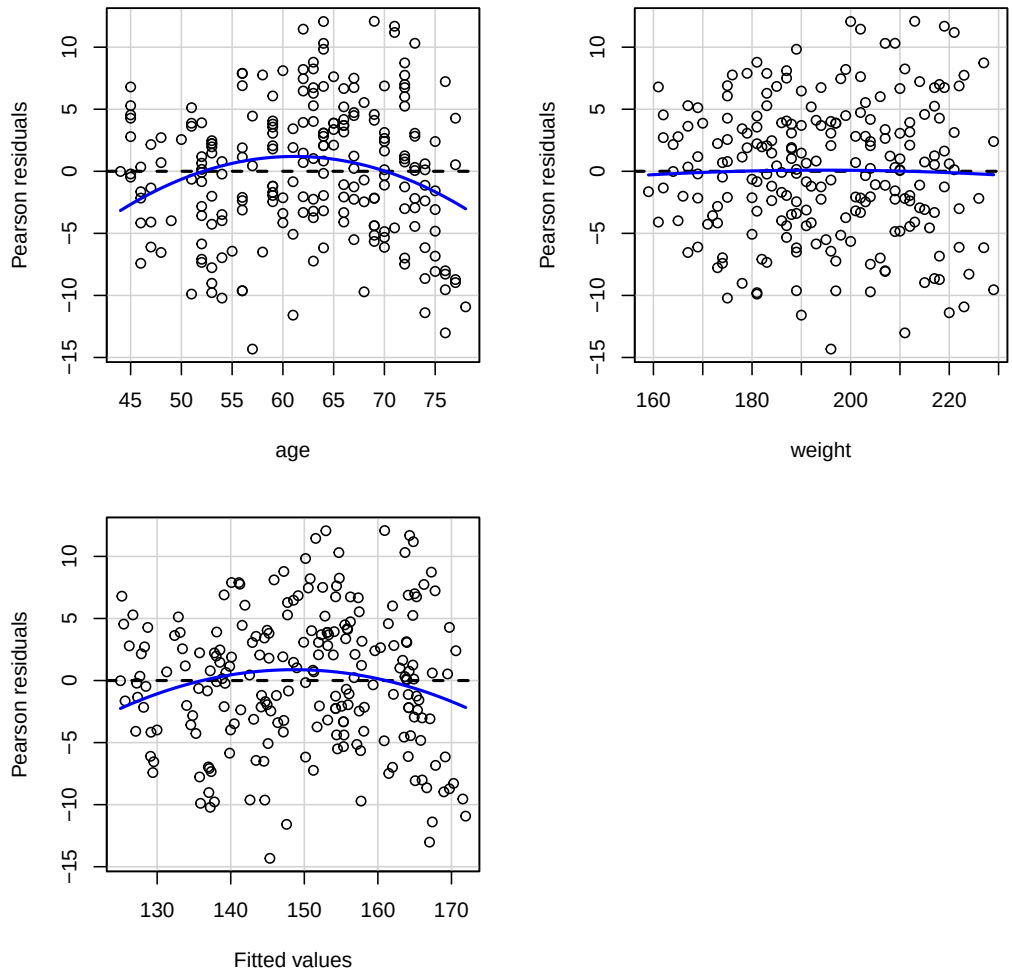
The OLS model assumes that all the residuals have a constant variance. It is conventional to look at the plot in Figure: 10.5 to check for potential heterogeneity in the model, also called heteroscedasticity. As the plot show, there is small trend in the residuals but it may not be significant.

We can further explore the possibility of heteroscedasticity by using the Breuch Pagan test available as the `bptest()` function in the `lmtest` package.

```
bptest(bpmodel)
```

```
##
```

```
residualPlots(bpmodel)
```




```
##             Test stat Pr(>|Test stat|)
## age           -3.2511         0.001334 **
## weight        -0.2617         0.793797
## Tukey test    -2.3594         0.018303 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 10.4: Plots to Check Linearity

```
plot(x=bpmodel, which=3)
```

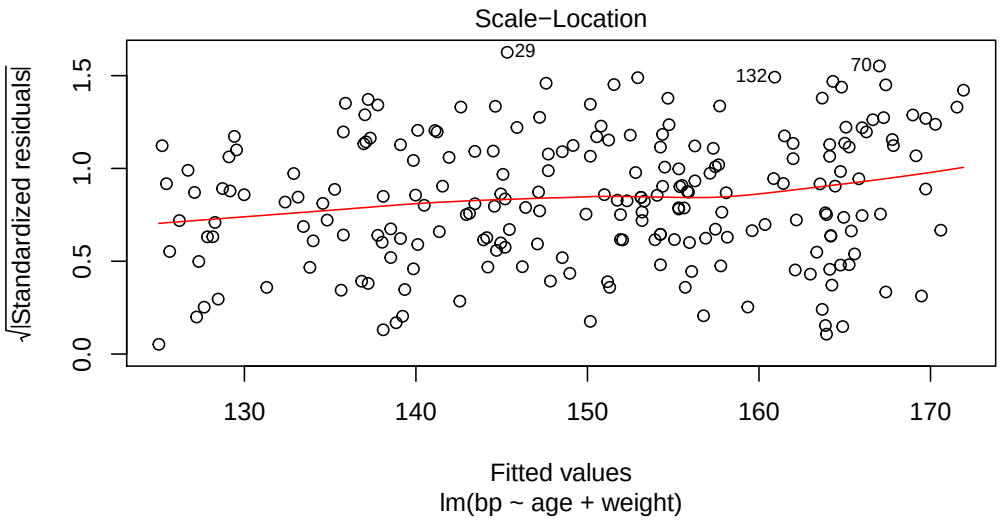


Figure 10.5: Plot to Check for Heteroscedasticity

```
##  studentized Breusch-Pagan test
##
## data:  bpmodel
## BP = 7.83, df = 2, p-value = 0.01994
```

The Breuch Pagan test tests the null hypothesis of no heteroscedasticity. As can be seen from the results, the null hypothesis is rejected at 95% confidence interval and we do **not** have constant variance of residuals in our model.

Uncorrelated Predictors

The OLS model assumes that the predictor variables are independent and not related to each other. Any significant correlation between predictor variables would make the model results less reliable. Correlation between predictor variables, called MultiCollinearity can be estimated using Variance Inflation Factors. Variance Inflation Factors can be calculated using the `vif()` function. A VIF value less than 10 is considered acceptable.

```
vif(bpmodel)
##     age  weight
## 4.51628 4.51628
```

We can see from the output above that correlations between predictor variables is not a concern in our model.

Independent Residuals

We can if residuals are correlated with each other using the Durbin-Watson test. This test is available as function `dwtest` as part of the `lmtest` package. Like the Breuch Pagan test, this test looks at null hypothesis of no autocorrelation and tries to reject it. A p-value >

0.05 shows that the null hypothesis could not be rejected and we do not have statistically significant evidence of autocorrelation.

```
dwtest(bpmodel)
```

```
##
##  Durbin-Watson test
##
## data:  bpmodel
## DW = 2.1197, p-value = 0.8192
## alternative hypothesis: true autocorrelation is greater than 0
```

Outliers

Outlier values are those that differ greatly from other values of predictors or the response variables. Outliers can skew the regression line and can have significant influence on the regression results. Outliers can be visually detected in various diagnostic plots we have done so far. Formal detection of outliers usually considers two approaches: Leverage and Influence. Leverage statistics are designed to identify observations which have predictor values that are far away from the rest of the data. Influence statistics look at which values play too large a role in the regression model.

We can measure the change in the regression results as a result of deleting an observation. This value is called `DFBETAs`. It can be calculated using the `dfbetas` function.

```
dfb <- dfbetas(bpmodel)
head(dfb)
```

```
##    (Intercept)           age       weight
## 1 -0.039521410  0.0040605801  0.017320382
## 2 -0.011804765 -0.0053884549  0.009100589
## 3 -0.010800476 -0.0186179048  0.016753416
## 4  0.010021037 -0.0212482412  0.006672112
## 5  0.003553237  0.0065673756 -0.004930005
## 6 -0.008912326  0.0003038909  0.004934012
```

Output above shows that the first data point, if removed, will change the intercept by -0.039. Similarly We can measure the influence that an observation has on its fitted value with the function `dffits`. DFFITS can be interpreted in the same way as DFBETAS.

```
dff <- dffits(bpmodel)
head(dff)
```

```
##           1           2           3           4           5           6
## -0.05685922 -0.01827616 -0.02433017 -0.04307776  0.02656855  0.01340052
```

While the DFFITS are good for measuring the influence on a single fitted value, Cook's Distance measures the influence an observation on all of the fitted values simultaneously. We saw a plot of Cook's Distance in 10.3. We can also generate Cook's Distance values using the `cooks.distance` function.

```
cooksd <- cooks.distance(bpmodel)
head(cooksd)
```

```
##            1            2            3            4            5            6
## 1.081293e-03 1.118300e-04 1.981837e-04 6.209621e-04 2.362273e-04 6.013163e-05
```

We can also formally test for outliers using the `outlierTest()` function in the `car` package.

```
attach(trees)
plot(Girth, Volume)
```

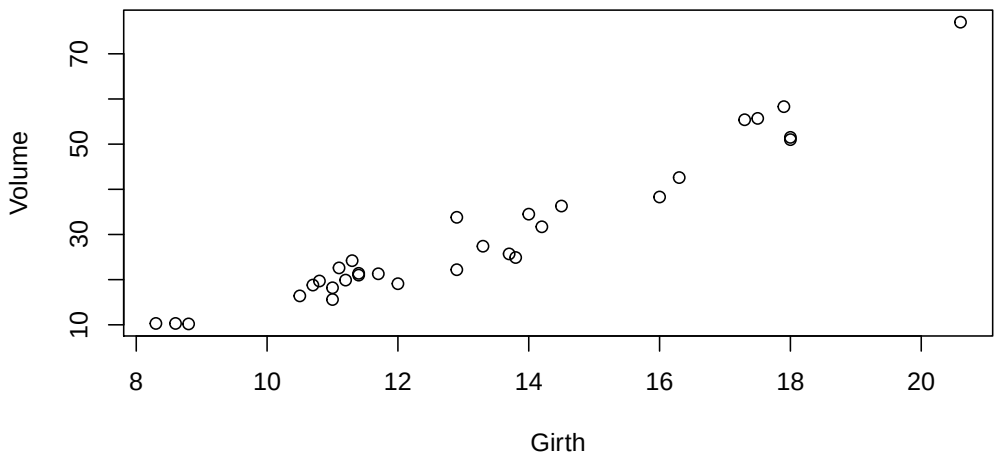


Figure 10.6: Volume vs Girth of Tree Trunks

```
outlierTest(bpmodel)
```

```
## No Studentized residuals with Bonferroni p < 0.05
## Largest |rstudent|:
##    rstudent unadjusted p-value Bonferroni p
## 29 -2.68033          0.0079222           NA
```

This function identifies significant outliers. As we can see, our current dataset does not have any significant outliers. However, there still would be observations that will have more influence on the model than other. These influential observations can be identified using the `influence.measure()` function.

```
influence.measures(bpmodel) #Output removed to save space
```

10.3 Improving Regression Model

In this section we will explore various techniques for building the most appropriate regression model. We will start by checking how we can specify our model to have higher order variables like square of values.

Polynomial Regression

We will consider a new dataset called *trees* for this portion. The dataset has the following columns: *Volume*, *Girth* and *Height*. Here *Volume* is the response variable and *Girth* and *Height* are the predictor variables. We can quickly explore this data by building a scatterplot shown in Figure: 10.6.

As we can see, the data does not seem linear. Values towards the right are curving upwards. We can try to capture this non-linear effect by specifying a square term in the model as follows: $y = \beta_0 + \beta_1 x_1 + \beta_1 x_1^2 + \epsilon$. However, including a square term usually adds a large dose of multicollinearity (correlation between predictor variables) to the model. To reduce the negative impact of multicollinearity, it is suggested that data should be standardized - rescaled to have a mean = 0. We can do this in R as follows:

```
treemodel <- lm(Volume ~ scale(Girth) + I(scale(Girth)^2))
summary(treemodel)
```

```
##
## Call:
## lm(formula = Volume ~ scale(Girth) + I(scale(Girth)^2))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.4889 -2.4293 -0.3718  2.0764  7.6447
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)         27.7452     0.8161  33.996  < 2e-16 ***
## scale(Girth)        14.5995     0.6773  21.557  < 2e-16 ***
## I(scale(Girth)^2)    2.5067     0.5729   4.376 0.000152 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.335 on 28 degrees of freedom
## Multiple R-squared:  0.9616,Adjusted R-squared:  0.9588
## F-statistic: 350.5 on 2 and 28 DF,  p-value: < 2.2e-16
```

Results show that both $Girth$ and $Girth^2$ have significant coefficients. The model is strongly significant as well and has a very high R-Square value. We can graphically see the how model fits in Figure: 10.7. As we can see, the model fits the data very well.

When including higher order variables, we should be mindful of the principle of `parsimony` which states that if a higher order term x^m is part of the model then all lower order terms x, x^2 through x^{m-1} should be part of the model as well.

Interaction Between Predictors

Even though predictor variables are assumed to be independent of each other, they may interact in their influence on the response variable. That is - the impact of one predictor variable in the response variable may depend on the value of a different predictor variable. We can capture such interaction effects by including an additional term in our model. Such interaction terms are usually formed by multiplying one predictor variable by another. This results in model specifications like: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{1:2} x_1 x_2 + \epsilon$ where $\beta_{1:2}$ indicate a coefficient of an interaction term. We can do this in R as follows:

```
treeint <- lm(Volume ~ Girth + Height + Girth:Height)
summary(treeint)
```

```
##
## Call:
## lm(formula = Volume ~ Girth + Height + Girth:Height)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
plot(scale(Girth), Volume)
lines(fitted(treemodel) ~ scale(Girth))
```

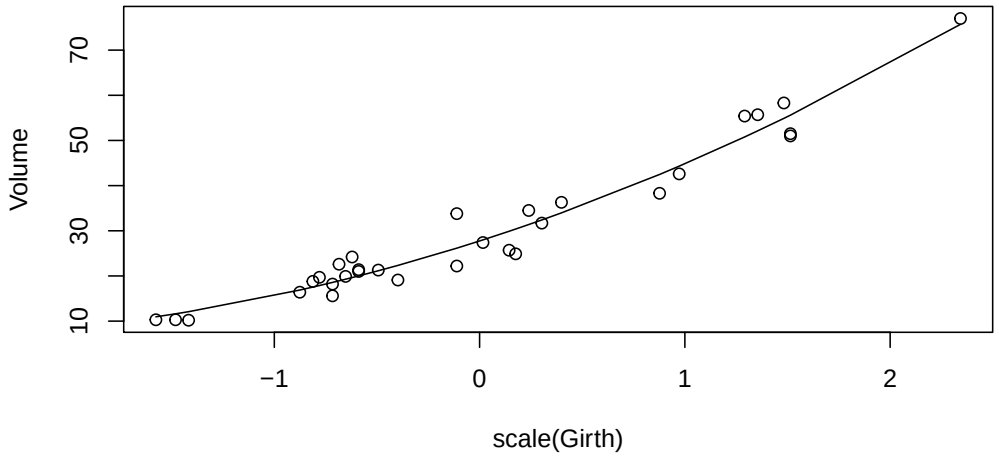


Figure 10.7: Quadratic Model of Volume vs Girth of Tree Trunks

```
## -6.5821 -1.0673  0.3026  1.5641  4.6649
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  69.39632   23.83575   2.911  0.00713 **
## Girth        -5.85585    1.92134  -3.048  0.00511 **
## Height       -1.29708    0.30984  -4.186  0.00027 ***
## Girth:Height  0.13465    0.02438   5.524 7.48e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.709 on 27 degrees of freedom
## Multiple R-squared:  0.9756,Adjusted R-squared:  0.9728
## F-statistic: 359.3 on 3 and 27 DF,  p-value: < 2.2e-16
```

We can see that the interaction term has a significant and positive coefficients. This means that the impact of *Girth* on *Volume* is higher for higher values of *Height*. Note that we can also interpret the results the other way around - that the impact of *Height* on *Volume* is higher for higher values of *Girth*. Continuing with the principle of parsimony discussed earlier, it is recommended to include all the constituents of the interaction effect (called main effects) be included in the model as well.

Categorical Predictor Variables

So far all our predictor variables have been numeric and continuous. However, often we get data that is Qualitative or Categorical in nature. In R vocabulary - a *factor* with many *levels*. These *levels* may be unordered (nominal values) or ordered (ordinal values).

Our *trees* dataset does not have any categorical variables - but we can create one. We will recode the *Height* variable into a categorical variable *is.tall* which will take the value *yes* for $Height > 80$ and the value *no* for $Height <= 80$.

```
trees$is.tall <- cut(trees$Height, breaks = c(-Inf, 80, Inf),
                     labels = c("no", "yes"))
class(trees$is.tall)
```

```
## [1] "factor"
```

Now we want to include *is.tall* variable in our regression model. R handles such categorical variables by converting them to a binary, 0/1 variable called a `dummy variable`. We can think of this as an *indicator* variable - it indicates whether the tree is tall or not. Essentially, we have a model $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$ where x_2 will take values 0 or 1. Note that when the value is 0, the model reduces to $y = \beta_0 + \beta_1 x_1 + \epsilon$. OTOH, when the value is 1, the model becomes $y = (\beta_0 + \beta_2) + \beta_1 x_1 + \epsilon$. The net effect is of just changing the y-intercept by the amount of β_2. We can run this model in R:

```
treesdummy <- lm(Volume ~ Girth + is.tall, data=trees)
summary(treesdummy)
```

```
##
## Call:
## lm(formula = Volume ~ Girth + is.tall, data = trees)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.8961 -2.3250  0.7692  1.7068  7.1240
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -35.8044     3.0530 -11.728 2.56e-12 ***
## Girth         4.8986     0.2303  21.271  < 2e-16 ***
## is.tallyes    4.7695     1.7003   2.805  0.00904 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.823 on 28 degrees of freedom
## Multiple R-squared:  0.9495,Adjusted R-squared:  0.9459
## F-statistic: 263.3 on 2 and 28 DF,  p-value: < 2.2e-16
```

Regression results show that the model is significant and all coefficient estimates are statistically significant as well. We can conclude that the response variable differs for trees with *is.tall* = yes and trees with *is.tall* = no. The mean amount of difference is 4.7695. We can explore the result graphically as shown in Figure: 10.8.

Non-Linear Regression

We have really been sidestepping the fact that we really know how tree's trunk volume is related to its girth and height. Volume of a cylinder of radius r and height h is simply $\pi r^2 h$. So ideally we would want a regression model where *Girth* and *Height* are multiplied together - $y = \beta_0 x_1^{\beta_1} x_2^{\beta_2}$. This form of course does not meet the linearity assumption. However, we can transform the model into a linear one by taking log of both sides. This will give us the following model: $logy = Ln\beta_0 + \beta_1 logx_1 + \beta_2 logx_2$. Lets see how this looks in R.

```
treeslog <- lm(log(Volume) ~ log(Girth) + log(Height), data = trees)
summary(treeslog)
```

```
treestall <- split(trees, trees$is.tall)
treestall[["yes"]]$fit <- predict(treesdummy, treestall[["yes"]])
treestall[["no"]]$fit <- predict(treesdummy, treestall[["no"]])
plot(Volume ~ Girth, data = trees, type = "n")
points(Volume ~ Girth, data = treestall[["yes"]], pch = 1)
points(Volume ~ Girth, data = treestall[["no"]], pch = 2)
lines(fit ~ Girth, data = treestall[["yes"]] )
lines(fit ~ Girth, data = treestall[["no"]])
```

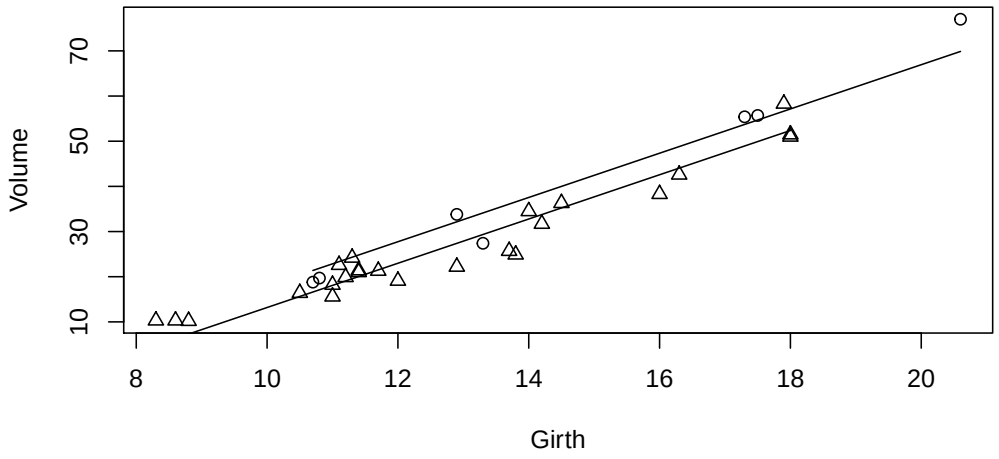


Figure 10.8: Impact of Dummy Variable

```
##
## Call:
## lm(formula = log(Volume) ~ log(Girth) + log(Height), data = trees)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.168561 -0.048488  0.002431  0.063637  0.129223
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -6.63162    0.79979  -8.292 5.06e-09 ***
## log(Girth)   1.98265    0.07501  26.432  < 2e-16 ***
## log(Height)  1.11712    0.20444   5.464 7.81e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.08139 on 28 degrees of freedom
## Multiple R-squared:  0.9777,Adjusted R-squared:  0.9761
## F-statistic: 613.2 on 2 and 28 DF,  p-value: < 2.2e-16
```

As we can see, this is our best model yet!

10.4 Model Selection

When we have several predictor variables available, the important question arises of which of those predictors to include in the regression model. Ideally, variable section should be done based on theoretical criteria and not analytical convenience. However, some times it may be necessary to choose variables to build the best model possible in absence of theoretical considerations. In this section we will look at stepwise regression using the command `step()` as a method for selecting predictor variables. Then we will consider how to compare two different models using the function `anova()`.

step: stepwise method for variable selection

anova: used for comparing two models

The mathematical criterion for deciding whether to include or remove predictor variables is called `AIC` - Akaike Information Criterion - which is defined for a linear regression with K predictor variables as: $AIC = (SS_{res}/\sigma^2) + 2K$. For best model performance, we want the smallest AIC value. As number of predictors grow, without a corresponding significant decline in magnitude of residuals, the AIC goes up. Hence, minimizing AIC ensures that only predictors with significant impact on residuals are included in the model.

Stepwise Regression

We will explore our last model of trees' volume using Stepwise Regression (saved as model *treeslog*. The default stepwise mode is `backward elimination` where we start with a full model (all predictors included) and test removing a predictor variable in each step. The combination with lowest AIC is selected as the preferred model.

```
step(object = treeslog, direction = "backward")
```

```
## Start:  AIC=-152.69
## log(Volume) ~ log(Girth) + log(Height)
##
##               Df Sum of Sq    RSS      AIC
## <none>                      0.1855 -152.685
## - log(Height)  1    0.1978 0.3832 -132.185
## - log(Girth)   1    4.6275 4.8130  -53.743
##
## Call:
## lm(formula = log(Volume) ~ log(Girth) + log(Height), data = trees)
##
## Coefficients:
## (Intercept)   log(Girth)  log(Height)
##      -6.632        1.983        1.117
```

We can see in the output that when all predictors are included, the model has an AIC of -152.69. The output table shows that if any of the predictors are removed, the AIC value increases. Hence, we conclude that the full model is the best mode and we do not need to remove any predictors from our model.

We can also run the stepwise regression using `forward selection`. In this case we start with an empty model (only the intercept) and then additional predictors are added in each step until minimum AIC value is reached.

```
emptymodel <- lm(log(Volume) ~ 1, data=trees) #Start with an empty model
step(object = emptymodel, direction = "forward",
     scope = log(Volume) ~ log(Girth) + log(Height))
```

```
## Start:  AIC=-38.82
## log(Volume) ~ 1
##
##               Df Sum of Sq    RSS      AIC
```

```
## + log(Girth)    1    7.9254 0.3832 -132.185
## + log(Height)   1    3.4957 4.8130  -53.743
## <none>                      8.3087  -38.817
##
## Step:  AIC=-132.19
## log(Volume) ~ log(Girth)
##
##                Df Sum of Sq     RSS     AIC
## + log(Height)   1   0.19778 0.18546 -152.69
## <none>                      0.38324 -132.19
##
## Step:  AIC=-152.69
## log(Volume) ~ log(Girth) + log(Height)
##
## Call:
## lm(formula = log(Volume) ~ log(Girth) + log(Height), data = trees)
##
## Coefficients:
## (Intercept)   log(Girth)  log(Height)
##      -6.632        1.983        1.117
```

Output shows that the empty model (only an intercept) has an AIC of -38.82. In the first step one predictor is added and AIC reaches -132.19; then in the second step second predictor is added and AIC reaches the minimum value of -152.69. We reach the same optimal solution as in the backward elimination process before.

Model Comparison

We can directly compare two models as well. We will create a new model first for comparison purposes.

```
trees$random <- rnorm(nrow(trees)) #Random variable
model.A <- lm(log(Volume) ~ log(Girth) + log(Height),
              data = trees)
model.B <- lm(log(Volume) ~ log(Girth) + log(Height)
              + trees$random, data = trees)
```

Clearly model.B should not be better than model.A as the only difference is a random variable. However, how do we test the relative effectiveness of the two models. First, we can just calculate AIC values using the function `AIC()`. We can see in the output below that model.A has lower AIC values than model.B.

```
AIC(model.A); AIC(model.B)
```

```
## [1] -62.71125
## [1] -61.79706
```

We can also consider the subset model (model.A) as the null hypothesis, the superset model (model.B) as the alternate hypothesis and then use the function `anova()` to check if we can reject the null hypothesis in the favor of the alternate hypothesis.

```
#Null hypothesis first, then alternate hypothesis
anova(model.A, model.B)
```

```
## Analysis of Variance Table
##
## Model 1: log(Volume) ~ log(Girth) + log(Height)
## Model 2: log(Volume) ~ log(Girth) + log(Height) + trees$random
```

```
##   Res.Df     RSS Df Sum of Sq      F Pr(>F)
## 1     28 0.18546
## 2     27 0.17908  1 0.0063836 0.9625 0.3353
```

As the output shows, with a p-value of higher than 0.05, we can not reject the null hypothesis. Thus the analysis shows that model.A is the preferred model.

10.5 Conclusion and Word of Caution

In this chapter, we introduced the idea of linear regression using OLS. We saw how to run linear regression model in R including extracting valuable information from the model. We then looked at various assumptions inherent in OLS and how to test whether those assumptions hold in our model. Finally, we looked at model selection issues including stepwise regression and comparing different models.

Linear Regression is NOT a Silver Bullet

Linear regression is extremely popular. Modern statistical software has made building a linear regression model so easy that we are tempted to use it everywhere. Rarely do regression diagnostics - including testing of linear regression model assumptions, are carried out in a robust and reliable matter. Further, incorrect interpretation of regression model results is endemic.

Linear regression is a specific tool valid only for specific contexts that meet its requirement. The regression analysis is going to be only as useful as the model it tries to estimate. Hence, utmost case must be taken in first designing a theoretically consistent model before the model is evaluated using regression analysis.

Chapter 11

Generalized Linear Models

In God We Trust; All Others Must Bring Data
– *Willian Edwards Deming*

The Multiple Regression model we explored in the last chapter assumed that the error terms were independently and identically distributed as Normal. Generalized linear models ("GLM") extend the traditional multiple regression model to include error terms following different distributions. R provides the function `glm()` to estimate generalized linear models in similar ways as the previously studied `lm()` function. The only additional information required by the `glm()` command is the *family* argument that specifies the distribution of the error term.

We will begin our exploration of GLMs with most popular one - Logit Model or Logistic Regression for a binary response variable.

11.1 Logistic Regression

Logistic regression is used to model dichotomous outcome variables - variables that take binary values - True/False, Yes/No, 1/0 etc. Before we run a regression, the binary response variable needs to be transformed to a continuous variable of wide range. The standard approach is to calculate log odds - log of odds ratio. In the logit model the log odds of the outcome variable is modeled as the response to the linear combination of the predictor variables. Log-odds have the recognizable curve as shown in the Figure 11.1.

For illustrating logistic regression, we will use a dataset of 400 graduate school applications.

```
admit.data <- read.csv("binary.csv")
names(admit.data)
```

```
## [1] "admit" "gre"   "gpa"   "rank"
```

The dataset consists four columns - the outcome variable *admit* is a binary variable. The predictor variables are *gre* (the GRE score), *gpa* (the undergrad GPA) and *rank* (a categorical variable from 1 to 4 indicating the status or prestige of the institution, with 1 denoting the highest prestige).

We will first convert *rank* as a factor and then run a logistic regression.

```
admit.data$rank <- factor(admit.data$rank)
logit.model <- glm(admit ~ gre + gpa + rank, data = admit.data,
                   family = "binomial")
summary(logit.model)
```

```r
probs <- seq(0, 1, 0.01)
odds = probs / (1 - probs); logodds = log(odds)
plot(logodds, probs)
```

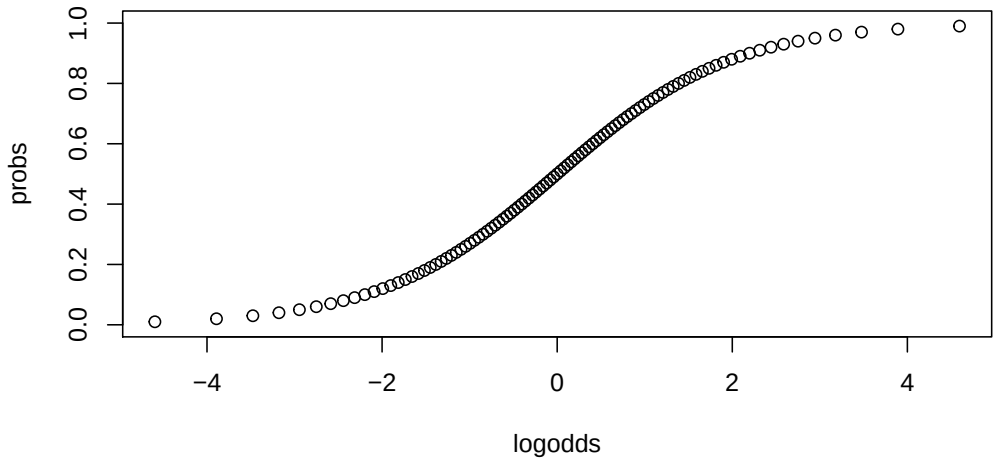


Figure 11.1: Plot of Log of Odds Ratio

```
##
## Call:
## glm(formula = admit ~ gre + gpa + rank, family = "binomial",
##     data = admit.data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6268  -0.8662  -0.6388   1.1490   2.0790
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.989979   1.139951  -3.500 0.000465 ***
## gre          0.002264   0.001094   2.070 0.038465 *
## gpa          0.804038   0.331819   2.423 0.015388 *
## rank2       -0.675443   0.316490  -2.134 0.032829 *
## rank3       -1.340204   0.345306  -3.881 0.000104 ***
## rank4       -1.551464   0.417832  -3.713 0.000205 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 499.98  on 399  degrees of freedom
## Residual deviance: 458.52  on 394  degrees of freedom
## AIC: 470.52
##
```

```
## Number of Fisher Scoring iterations: 4
```

While interpreting the output, keep in mind that the response variable modeled here is not the variable *admit* but the log-odds of *admit*. The output of the model is similar to linear regression - we see residuals, coefficients and p-value - all with similar interpretations as before. We see that both *gre* and *gpa* are statistically significant (p-value < 0.05). The *rank* factor has been converted into three dummy variables - all statistically significant.

Looking at the coefficients, we can see that for 1 unit change in GRE score, log odds of admission increases by 0.002. Similarly, for 1 unit change in GPA, log odds of admission increases by 0.804. Attending an institution of rank 2 vs an institution of rank 1 reduces the log odds of admission by 0.6754.

Given a logistic regression model, we can conduct further significance tests using the `wald test` in the package `aod`. For example, we can test whether the *rank* factors taken together are statistically significant or not. As the output below shows, the three factors taken together have a statistically significant impact.

```
library(aod)
wald.test(b = coef(logit.model), Sigma = vcov(logit.model),
          Terms = 4:6)
```

```
## Wald test:
## ----------
##
## Chi-squared test:
## X2 = 20.9, df = 3, P(> X2) = 0.00011
```

Similarly, we can test whether it makes a difference whether the students comes from an institution of rank 3 of rank 4. While both these ranks had a statistically significant coefficients in the logit model, the magnitude of the coefficients show little difference - so its a relevant question to ask. As we can see from the output below, the effects of rank 3 and rank4 are in fact not statistically distinguishable as far as their impact on log odds of admission.

```
coefflist <- cbind(0,0,0,0,1,-1)
wald.test(b = coef(logit.model), Sigma = vcov(logit.model),
          L = coefflist)
```

```
## Wald test:
## ----------
##
## Chi-squared test:
## X2 = 0.29, df = 1, P(> X2) = 0.59
```

As log odds are not intuitive to interpret, we can calculate coefficients as odds ratios. We can then use our model to predict probability of admission for a given set of input values.

```
exp(coef(logit.model))
```

```
## (Intercept)         gre         gpa       rank2       rank3       rank4
##   0.0185001   1.0022670   2.2345448   0.5089310   0.2617923   0.2119375
```

```
admitnew <- data.frame(gre = 700, gpa = 3.9, rank = 1)
admitnew$rank = factor(admitnew$rank)
predict(logit.model, newdata = admitnew, type = "response")
```

```
##         1
## 0.6749952
```

Now we can say that a unit change in GPA improves the odds of being admitted by a factor of 2.23.

```
library(VGAM)
probs <- seq(0, 1, 0.01)
probitdata <- probit(probs)
                    ## Warning:  'probit' is deprecated.
                        ## Use 'probitlink' instead.
                          ## See help("Deprecated")
plot(probs, probitdata)
```

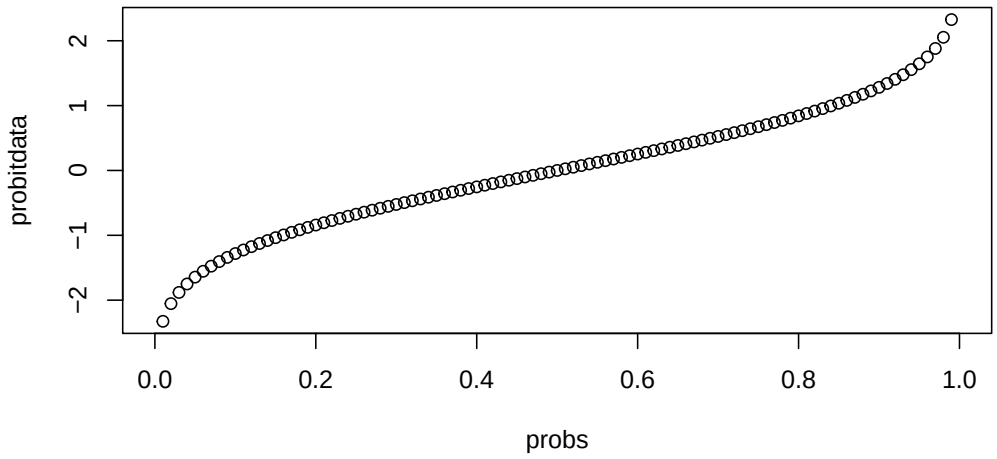


Figure 11.2: Plot of Probit Transformation

11.2 Probit Model

A probit model is also suitable for binary response variables. In the probit model, the inverse standard normal distribution of the probability is modeled as a linear combination of the predictors. Probit transformation has a similar shape as logistic transformation as can be seen in Figure: 11.2.

We can use the same `glm()` command to run a probit model. We need to specify *link* as *probit* to ensure that the model is run on a probit transformed outcome variable.

```
probit.model <- glm(admit ~ gre + gpa + rank, data = admit.data,
                    family = binomial(link="probit"))
summary(probit.model)
```

```
##
## Call:
## glm(formula = admit ~ gre + gpa + rank, family = binomial(link = "probit"),
##     data = admit.data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6163  -0.8710  -0.6389   1.1560   2.1035
##
## Coefficients:
```

```
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.386836   0.673946  -3.542 0.000398 ***
## gre          0.001376   0.000650   2.116 0.034329 *
## gpa          0.477730   0.197197   2.423 0.015410 *
## rank2       -0.415399   0.194977  -2.131 0.033130 *
## rank3       -0.812138   0.208358  -3.898 9.71e-05 ***
## rank4       -0.935899   0.245272  -3.816 0.000136 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 499.98  on 399  degrees of freedom
## Residual deviance: 458.41  on 394  degrees of freedom
## AIC: 470.41
##
## Number of Fisher Scoring iterations: 4
```

Usually, logit and probit models are pretty much interchangeable. We can demonstrate the fact by calculating predicted probability for the same values as for logit before. As we can see from the output below, the predicted probabilities are nearly identical.

```
predict(probit.model, newdata = admitnew,  type = "response")
```

```
##         1
## 0.6697505
```

Much of our discussion of logistic regression is also applicable to probit models as well. The actual logit and probit transformations show some difference only very close to the edges (near probability either 0 or 1). This leads to minor, non-significant differences in model fit. However, logit models have the advantage of an easily interpretable result - log of odds ratio is easier to understand and interpret than the inverse of cumulative normal distribution. As a result, logistic regression is much more popular than probit regression.

11.3 Survival Analysis

In last section we looked at modeling a binary response varioable. In this section, we extend out understanding to include the time taken for the event to happen. Essentially, we want to model the time duration for an event to occur. Examples of such models may include - time taken for a loan to default, time for a machine to fail, time until a stock market crash or time taken for seeds to getminate. Traditionally, survival models have been used to model time to death (hence the name survival). Survival analysis is often called Duration Analysis or Failure Time Analysis as well.

We did not go into much theoretical details for Linear Regression and Logistic Regression as it was assumed that students have had prior exposure to those models (in the course pre-req TO301). As survival analysis is likely to be a new concept for most students, this writeup provides a deeper theoretical exploration than we did for previous topics.

Survival Data

Our main focus in survival analysis is survival time or "time to event". While time to event is a continuous variable, it is difficult to model it using linear regression because of non-normality and censoring. Non-normality occurs from the fact that time to event typically follows exponential distribution. The non-normality aspect of the data violates the normality assumption of OLS linear regression. Time to event data is also typically

right-censored - meaning that the information may be incomplete as the some observations do not get enough time to fail during the period under study. The most common context of survival analysis includes following subjects over time and observe at which point in time they experience the event of interest. It is typical that the study is not long enough to allow the event to occur for all the subjects in the study; resulting in a right censored data. Subjects also drop out of studies for various reasons leading to further right censoring. We use the name right censored because in this case the missing observations are further to the right end of the x-axis (if we plot time on x-axis).

Our main purpose in doing survival analysis is to examine relationships between survival time and one or more predictors (usually called covariates in survival models). The most widely used modeling technique for the purpose is the Cox Proportional Hazard Regression Model. This will the focus of our discussion.

Hazard Rate

We are primarily interested in the Survival Functiion $S(t) = Prob(T > t) = 1 - P(t)$ where T is the survival time and $P(t)$ is the Cumulative Probability Density Function of T. We typically assume that $S(0) = 1$, $S(t) = 0$ as $t \to \infty$ and $dS(t)/dt <= 0$.

Key to survival analysis is the concept of hazard rate - also called hazard function. Hazard rate can be defined as the probability that an event will occur at time `t` assuming that the event has not already happened by then. Another way to think about it is to suppose that a subject has survived until time t and then calculate the probability that it will not survive for an additional time dt with $dt \to \infty$. Essentially, hazard rate at time t, often denoted as $h(t)$ is the rate at which events occur at time t. Hazard rate (most commonly log of hazard rate) is typically used as the response variable in survival analysis.

Mathematically, we can calculate that $h(t) = f(t)/S(t)$ where $f(t)$ is the probability density function of T and $S(t)$, as we saw above, is the survival function. A constant hazard rate, $h(t) = k$ implies an exponential distribution of survival times with probability density function $p(t) = ke^{-kt}$.

Other common hazard function specifications include: $log(h(t)) = k + \rho t$ leading to Gompertz distribution of survival time, and $log(h(t)) = k + \rho log(t)$ leading to Weibull distribution of survival time. While several parametric regresison models exist for estimating specific functional form of hazard rate, we will focus on the `Cox Model` that leaves the baseline hazard function unspecified as the general form $log(h_0(t)) = \alpha(t)$. Including covariates x_1 through x_n, our model to be estimated becomes:

$$log(h(t)) = \alpha(t) + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n$$

or equivalently,

$$h(t) = e^{\alpha(t)} e^{(\beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n)}$$

which reduces to

$$h(t) = e^{\alpha(t)} e^{\eta}$$

where

$$\eta = (\beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n)$$

is the linear predictor. We can see that if we have two observations i and j then the hazard ratio for these two observations

$$\frac{h_i(t)}{h_j(t)} = \frac{e^{\eta_i}}{e^{\eta_j}}$$

is independent of time t. As the hazard rate is proportional to the the linear predictor, this model is also known as the **Cox Proportional Hazards Model**.

Survival Analysis in R

We will use the `OIsurv` packages which loads the other two required packages `survival` (which has the required functions) and `KMsurv` (which has some useful datasets for survival analysis)

```
## Error in library(OIsurv):  there is no package called 'OIsurv'
```

```
install.packages{"OIsurv} Output suppressed for brevity
library(OIsurv); library(survival)
```

Most functions in the *survival* package need survival objects created by the `Surv()` function. For right censored data, survival objects can be created using two arguments to the *Surv* function - the event time or the censoring time and a dummy variable coded 1 if the event occured or 0 of the observation is censored.

We will use a dataset on Recidivism to illustrate the process of survival analysis in R. The dataset contains observations for a year of make prisoners after being released from prison. The dataset has the following variables:

- `week`: Number of weeks to get arrested again, or censoring time in case the subject does not get arrested during study period
- `arrest`: dummy variable to indicate whether the subject was arrested during the study period; 1: Yes, 0: No.
- `fin`: Whether the subject received financial assistance; 1: Yes, 0: No.
- `age`: in years at the time of release from prison.
- `race`: 1 for African American, 0 for all others
- `wexp`: Full time work experience before incarneation; 1: Yes, 0: No
- `mar`: Whether the subject was married at the time of release; 1: Yes, 0: No.
- `paro`: Whether the subject was released on parole; 1: Yes, 0: No.
- `prio`: Number of prior convictions.
- `educ`: Education level; 2: grade 6 or less, 3: grades 6 through 9, 4: grades 10 and 11, 5: grade 12, or 6: some post-secondary.
- `emp1 - emp52`: Dummy variables; 1 if the subject was employed in the corresponding week of the study; 0 otherwise.

We will start by importing the data and taking a quick look inside.

```
rossi <- read.table("Rossi.txt", header = TRUE)
rossi[1:5, 1:15] #First five rows, first 15 columns
```

```
##   week arrest fin age race wexp mar paro prio educ emp1 emp2 emp3 emp4 emp5
## 1   20      1   0  27    1    0   0    1    3    3    0    0    0    0    0
## 2   17      1   0  18    1    0   0    1    8    4    0    0    0    0    0
## 3   25      1   0  19    0    1   0    1   13    3    0    0    0    0    0
## 4   52      0   1  23    1    1   1    1    1    5    0    0    0    0    1
## 5   52      0   0  19    0    1   0    1    3    3    0    0    0    0    0
```

First two columns are most important here. First column is the time stamp of observation in number of weeks; second column is whether the subject was rearrested (`arrest` = 1) or not (`arrest` = 0) at that time. So the first subject was rearrested in week 20th. A `week` = 52 and `arrest` = 0 (like the fourth row) means that the subject was never arrested until the study ended with the censoring time of 52.

Lets take a look at the structure of the data and make needed changes.

```
str(rossi[,1:10])
```

```
## 'data.frame': 432 obs. of  10 variables:
##  $ week  : int  20 17 25 52 52 52 23 52 52 52 ...
##  $ arrest: int  1 1 1 0 0 0 1 0 0 0 ...
```

```
##  $ fin   : int  0 0 0 1 0 0 0 1 0 0 ...
##  $ age   : int  27 18 19 23 19 24 25 21 22 20 ...
##  $ race  : int  1 1 0 1 0 1 1 1 1 1 ...
##  $ wexp  : int  0 0 1 1 1 1 1 1 0 1 ...
##  $ mar   : int  0 0 0 1 0 0 1 0 0 0 ...
##  $ paro  : int  1 1 1 1 1 0 1 1 0 0 ...
##  $ prio  : int  3 8 13 1 3 2 0 4 6 0 ...
##  $ educ  : int  3 4 3 5 3 4 4 3 3 5 ...
```

Seems like several factors are coded as numbers. In contrast to how we approached them before, we will leave them as numbers (its a good exercise to change them to factors and see what breaks in following pages). We first build the survival object.

```
rossi.surv <- Surv(rossi$week, rossi$arrest)
```

We can now use this survival object as the response variable in a Cox model. Cox models can be run using the `coxph()` in the *survival* package. We will start by using all the variables except $emp1 - emp52$ dummy variables. The syntax of $coxph()$ is quite similar to the $lm()$ and $glm()$ syntax we have already seen.

```
mod.first <- coxph(rossi.surv ~ fin + age + race + wexp
                   + mar + paro + prio + educ, data=rossi)
summary(mod.first)
```

```
## Call:
## coxph(formula = rossi.surv ~ fin + age + race + wexp + mar +
##     paro + prio + educ, data = rossi)
##
##   n= 432, number of events= 114
##
##          coef exp(coef) se(coef)      z Pr(>|z|)
## fin  -0.35963   0.69794  0.19180 -1.875  0.06079 .
## age  -0.05768   0.94395  0.02187 -2.638  0.00835 **
## race  0.34554   1.41276  0.30907  1.118  0.26356
## wexp -0.11439   0.89191  0.21311 -0.537  0.59145
## mar  -0.42496   0.65380  0.38209 -1.112  0.26605
## paro -0.08991   0.91401  0.19568 -0.459  0.64589
## prio  0.08469   1.08838  0.02919  2.902  0.00371 **
## educ -0.18578   0.83046  0.13153 -1.412  0.15782
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      exp(coef) exp(-coef) lower .95 upper .95
## fin     0.6979     1.4328    0.4792    1.0164
## age     0.9440     1.0594    0.9044    0.9853
## race    1.4128     0.7078    0.7709    2.5891
## wexp    0.8919     1.1212    0.5874    1.3543
## mar     0.6538     1.5295    0.3092    1.3825
## paro    0.9140     1.0941    0.6229    1.3413
## prio    1.0884     0.9188    1.0279    1.1525
## educ    0.8305     1.2042    0.6417    1.0747
##
## Concordance= 0.656  (se = 0.026 )
## Likelihood ratio test= 35.35  on 8 df,   p=2e-05
## Wald test            = 33.74  on 8 df,   p=5e-05
## Score (logrank) test = 35.1  on 8 df,   p=3e-05
```

```
plot(survfit(mod.first), ylim=c(0.75, 1),
     xlab = "Weeks of Observation",
     ylab="Proportion of Subject Not Rearrested")
```

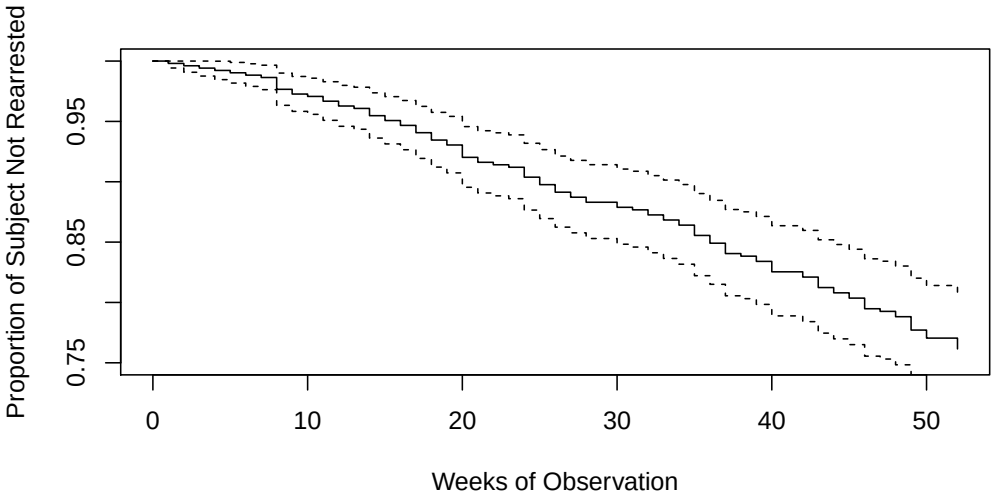


Figure 11.3: Plot of Survival Function

Interpretation of the results is very similar to interpretation of linear regression output. The coefficients show the impact of unit change in predictor variable on the response variable (in this case log hazard ratio). Statistical significance can be checked with the p-value in the last column. So, we can say that the log hazard rate for subjects who were provided financial aid ($fin = 1$) is 0.40 less than log hazard rate for subjects who were not provided financial aid ($fin = 0$); and the effect is statistically significant. Predictors *age* and *prio* (prior convictions) were also found to be statistically significant.

As interpreting log hazard rates can be cumbersome, the output also provides exponentiated coefficients - which can be interpreted as multiplicative effects on the hazard rate. For example, *age* has a coefficient $\beta = -0.05$ and exponentiated coefficient $e^{\beta} = 0.95$. That means that with increase of 1 in *age*, the hazard rate is 0.95 times the original value - a reduction of 5%. Similarly, we can see that an increase of one in prior convictions leads to an increase in log hazard rate of 0.079 and the hazard rate is multiplied by a factor of 1.08 (i.e. an increase of 8% in hazard rate).

Model significance is shown by the Likelihood Ratio Test, Wald Test and Score Test which tests the null hypothesis that all the coefficients are zero. As the results show, the null hypothesis is rejected and the model is found significant. The R-Square value can be interpreted in a similar manner as in linear regression.

Now that we have a suitable model, we can now examine the estimated distribution of survival times using the `survfit()` function that estimates the survival function $S(t)$ at the mean values of all predictors. Figure 11.3 shows the survival function for our analysis including the 95% confidence band.

Time Dependent Covariates

The previous model assumed that the effect of covariates on the survival object was independent of the the time parameter. The *coxph* function allows for time dependent covariates as well. However, in such cases it requires that data for each time period for a subject appear as a separate record (or row) in the data set. For example - we have 52 weeks of employment data in our dataset - but each week is coded as a column. To include employment data (which is time dependent) we will need to first reshape the data in what is commonly known as the "long form".

```
# Change to the long format
rossi.long <- reshape(data = rossi, varying = paste0("emp", 1:52),
                      v.names = "employed", timevar = "time",
                      idvar = "id", direction = "long", sep = "")

# Sort by id and time
library(doBy)
## Error in library(doBy):  there is no package called 'doBy'
rossi.long <- orderBy( ~  + id + time, data = rossi.long)
## Error in orderBy(~+id + time, data = rossi.long):  could not find function
"orderBy"
# Drop rows where emp is NA (time after event/censoring)
rossi.long <- rossi.long[!is.na(rossi.long$emp),]
head(rossi.long)
##     week arrest fin age race wexp mar paro prio educ time employed id
## 1.1   20      1   0  27    1    0   0    1    3    3    1        0  1
## 2.1   17      1   0  18    1    0   0    1    8    4    1        0  2
## 3.1   25      1   0  19    0    1   0    1   13    3    1        0  3
## 4.1   52      0   1  23    1    1   1    1    1    5    1        0  4
## 5.1   52      0   0  19    0    1   0    1    3    3    1        0  5
## 6.1   52      0   0  24    1    1   0    0    2    4    1        0  6
```

For time varing covariates, we will need start and stop time for each time interval and the corresponding value of the time varying covariates for that time interval. We want to check for two such covariates: whether employed during current time period or whether employed during last (lagged by 1) time period. The following code calculated the values in the long form.

```
library(plyr)  #Load plyr package

# Create time variables and various forms of exposure variables
rossi.long <- ddply(
  .data = rossi.long, .variables = c("id"), .drop = TRUE, .fun = function(DF)
  {
  DF$start <- c(0, head(DF$time, -1))  #Start of each interval
  DF$stop <- DF$time  #End of each interval
  DF$event <- 0  #Event indicator for each interval
  DF[nrow(DF),"event"] <- DF[nrow(DF),"arrest"] # Arrest value
  DF$employed.lag1 <- c(rep(NA, 1), head(DF$employed, -1))
  DF  #Return DF
  })
rossi.long[rossi.long$id == 2, c("id","start","stop","event","employed",
                                 "employed.lag1")]
##    id start stop event employed employed.lag1
```

```
## 21  2     0    1     0        0            NA
## 22  2     1    2     0        0             0
## 23  2     2    3     0        0             0
## 24  2     3    4     0        0             0
## 25  2     4    5     0        0             0
## 26  2     5    6     0        0             0
## 27  2     6    7     0        0             0
## 28  2     7    8     0        0             0
## 29  2     8    9     0        0             0
## 30  2     9   10     0        1             0
## 31  2    10   11     0        1             1
## 32  2    11   12     0        1             1
## 33  2    12   13     0        1             1
## 34  2    13   14     0        1             1
## 35  2    14   15     0        0             1
## 36  2    15   16     0        0             0
## 37  2    16   17     1        0             0
```

The data for subject id = 2 shows the time dependent covariates in the long form. We can see that the subject was rearrested in the 17th week. We can also see that the subject was employed from 10th to 14th week.

We are now ready to run a cox model with time dependent covariates. We start by checking whether being employed in a week affects the chances of a subject getting arrested in that week.

```
mod.timed <- coxph(Surv(start, stop, event) ~ fin + age
                   + race + wexp + mar + paro + prio
                   + employed, data = rossi.long)
summary(mod.timed)
```

```
## Call:
## coxph(formula = Surv(start, stop, event) ~ fin + age + race +
##     wexp + mar + paro + prio + employed, data = rossi.long)
##
##   n= 19809, number of events= 114
##
##               coef exp(coef) se(coef)      z Pr(>|z|)
## fin      -0.35672   0.69997  0.19113 -1.866  0.06198 .
## age      -0.04634   0.95472  0.02174 -2.132  0.03301 *
## race      0.33866   1.40306  0.30960  1.094  0.27402
## wexp     -0.02555   0.97477  0.21142 -0.121  0.90380
## mar      -0.29375   0.74546  0.38303 -0.767  0.44314
## paro     -0.06421   0.93781  0.19468 -0.330  0.74156
## prio      0.08514   1.08887  0.02896  2.940  0.00328 **
## employed -1.32832   0.26492  0.25072 -5.298 1.17e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##          exp(coef) exp(-coef) lower .95 upper .95
## fin         0.7000     1.4286    0.4813    1.0180
## age         0.9547     1.0474    0.9149    0.9963
## race        1.4031     0.7127    0.7648    2.5740
## wexp        0.9748     1.0259    0.6441    1.4753
## mar         0.7455     1.3414    0.3519    1.5793
## paro        0.9378     1.0663    0.6403    1.3735
```

```
## prio        1.0889     0.9184    1.0288    1.1525
## employed    0.2649     3.7747    0.1621    0.4330
##
## Concordance= 0.708  (se = 0.023 )
## Likelihood ratio test= 68.65  on 8 df,   p=9e-12
## Wald test            = 56.15  on 8 df,   p=3e-09
## Score (logrank) test = 64.48  on 8 df,   p=6e-11
```

The result show that the time dependent covariate - *employed* has a large effect on the hazaes of rearrest. The hazard rate for a time period is smaller by a factor of 0.2649 (i.e. 73.5% less) when the subject is employed during the said time period. This large effect is a bit misleading though as it may be a result of reverse causality - the subject can't work if he has been arrested. So a better model would be to check whether being employed in the previous time period has an impact on the hazard rate of getting rearrested in the next time period.

```
mod.lagtimed <- coxph(Surv(start, stop, event) ~ fin + age
                      + race + wexp + mar + paro + prio +
                        employed.lag1, data  = rossi.long)
summary(mod.lagtimed)
```

```
## Call:
## coxph(formula = Surv(start, stop, event) ~ fin + age + race +
##     wexp + mar + paro + prio + employed.lag1, data = rossi.long)
##
##   n= 19377, number of events= 113
##    (432 observations deleted due to missingness)
##
##                    coef exp(coef) se(coef)      z Pr(>|z|)
## fin           -0.35130   0.70377  0.19181 -1.831 0.067035 .
## age           -0.04977   0.95144  0.02189 -2.274 0.022969 *
## race           0.32147   1.37915  0.30912  1.040 0.298369
## wexp          -0.04764   0.95348  0.21323 -0.223 0.823207
## mar           -0.34476   0.70839  0.38322 -0.900 0.368310
## paro          -0.04710   0.95399  0.19630 -0.240 0.810375
## prio           0.09199   1.09635  0.02880  3.194 0.001402 **
## employed.lag1 -0.78689   0.45526  0.21808 -3.608 0.000308 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##               exp(coef) exp(-coef) lower .95 upper .95
## fin              0.7038     1.4209    0.4832    1.0250
## age              0.9514     1.0510    0.9115    0.9932
## race             1.3792     0.7251    0.7525    2.5278
## wexp             0.9535     1.0488    0.6278    1.4481
## mar              0.7084     1.4116    0.3343    1.5013
## paro             0.9540     1.0482    0.6493    1.4016
## prio             1.0964     0.9121    1.0362    1.1600
## employed.lag1    0.4553     2.1966    0.2969    0.6981
##
## Concordance= 0.67  (se = 0.026 )
## Likelihood ratio test= 47.16  on 8 df,   p=1e-07
## Wald test            = 43.37  on 8 df,   p=7e-07
## Score (logrank) test = 46.4  on 8 df,   p=2e-07
```

As the results above show, being employed in the previous week reduces the hazard rate

of being rearrested by a factor of 0.455 (i.e. a reduction of 54.5%). A large and statistically significant result (p-value < 0.05).

Model Diagnostics for Cox Regression

Just like for linear regression, it is essential that we check that the data fits the basic assumptions inherent in running a Cox regression model. We will do model disgnostics on a smaller model with only the significant coefficients.

```
mod.small <- coxph(Surv(week, arrest) ~ fin + age + prio,
                   data=rossi)
mod.small
```

```
## Call:
## coxph(formula = Surv(week, arrest) ~ fin + age + prio, data = rossi)
##
##          coef exp(coef) se(coef)      z        p
## fin  -0.34695   0.70684  0.19025 -1.824 0.068197
## age  -0.06711   0.93510  0.02085 -3.218 0.001289
## prio  0.09689   1.10174  0.02725  3.555 0.000378
##
## Likelihood ratio test=29.05  on 3 df, p=2.189e-06
## n= 432, number of events= 114
```

Assumption of Proportional Hazards

Tests for proportional hazards are based on residuals of the model (specifically Scaled Schoenfeld Residuals). The function `cox.zph()` function calculates tests of the proportional-hazards assumption for each covariate, by correlating the corresponding set of scaled Schoenfeld residuals with time.

```
cox.zph(mod.small)
```

```
##             rho   chisq      p
## fin    -0.00657 0.00507 0.9433
## age    -0.20976 6.54147 0.0105
## prio   -0.08004 0.77288 0.3793
## GLOBAL       NA 7.13046 0.0679
```

Results show a strong evidence of non-proportional hazards for age, while the global test is not fully statistically significant. These tests are exploring whether there is a linear trends in the hazard rate. We can get further clarity by plotting the scaled Schoenfeld residuals as shown in Figure: 11.4.

We are looking for systematic departures from the expected horizontal line. The plots confirm the result of the *cox.zph*() function - we see string evidence of time dependent hazard for *age* while the impact of *fin* and *prio* on the hazard rate seens more or less constant against time.

One way to accommodate non-proportional hazards is to build explicitely include interaction terms between covariates and time into the Cox regression model. For example, based on the diagnostics test and the plot, we can include a linear interaction of time and age in the model. As these interactions need to be treated as time-dependent covariates, we will need to include them in the long form data.

```
mod.int <- coxph(Surv(start, stop, event) ~ fin + age + prio
                 + age:stop, data=rossi.long)
mod.int
```

```
par(mfrow=c(2,2)); plot(cox.zph(mod.small))
```

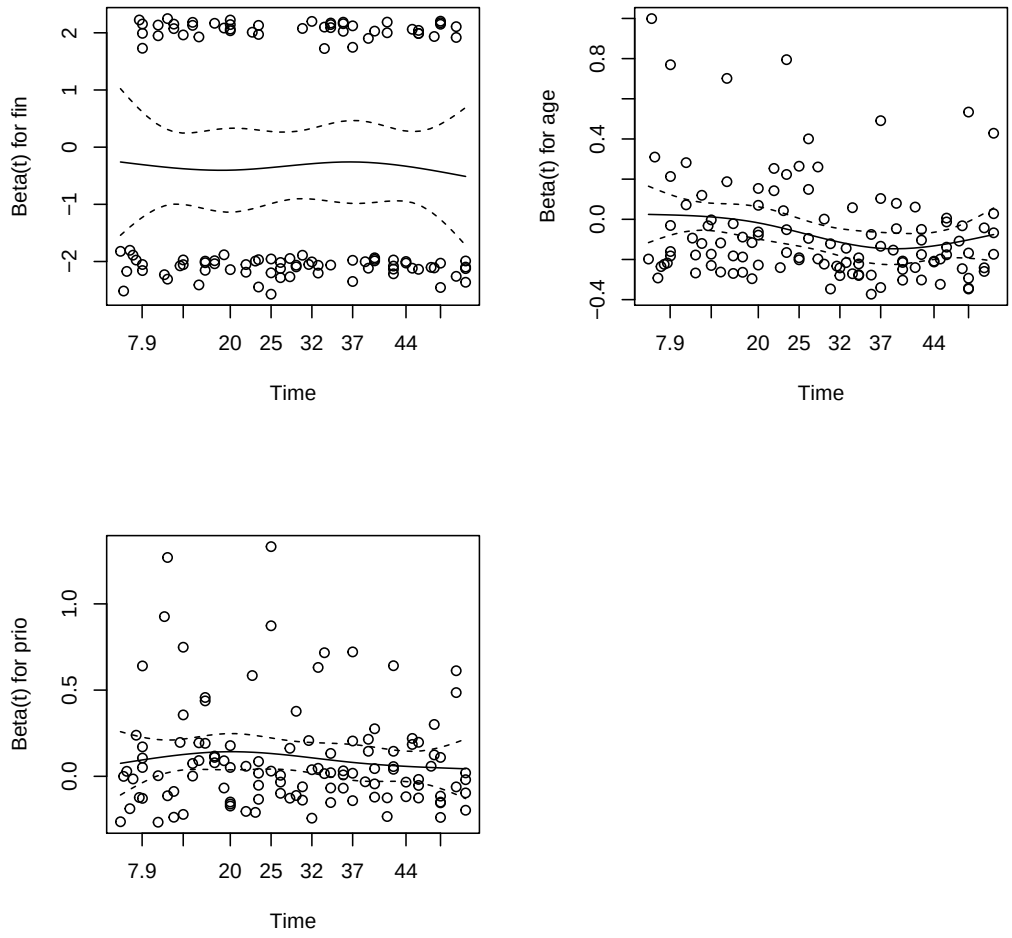


Figure 11.4: Diagnostic Plots for Proportional Hazard Assumption

```
## Call:
## coxph(formula = Surv(start, stop, event) ~ fin + age + prio +
##     age:stop, data = rossi.long)
##
##               coef exp(coef)  se(coef)      z        p
## fin      -0.348562  0.705702  0.190229 -1.832 0.066902
## age       0.032277  1.032803  0.039429  0.819 0.413015
## prio      0.098179  1.103160  0.027257  3.602 0.000316
## age:stop -0.003834  0.996173  0.001468 -2.612 0.008991
##
## Likelihood ratio test=36.03  on 4 df, p=2.846e-07
## n= 19809, number of events= 114
```

As the results show, the interaction between *age* and *time* is highly statistically significant with a negative coefficient. This means that the impact of *age* on hazard rate reduces with passage of time. We can now check again whether including the interaction effect helped the model meet the assumption of proportional hazards.

```
cox.zph(mod.int)
```

```
##              rho   chisq     p
## fin      -0.0068 0.00544 0.941
## age      -0.0471 0.36703 0.545
## prio     -0.0849 0.86449 0.352
## age:stop  0.0567 0.55701 0.455
## GLOBAL        NA 1.41901 0.841
```

All terms now show no statistically significant non-proportionality in hazards. Including the interaction term did help.

Outliers

Checking for outliers can be done using the `dfbeta` metric we have seen before in Linear Regression. Plot of *dfbeta*, as shown in Figure: 11.5 can help us see any significant outliers. We can see that, compared to the coeficient values, corresponding dfbeta values are pretty small. We can, hence, conclude that the model does not suffer from significant outlier problem.

Acknowledgement

Much of the material for this section has been taken from John Fox's excellent article "Cox Proportional-Hazards Regression for Survival Data" available at:
`https://socserv.socsci.mcmaster.ca/jfox/Books/Companion-1E/`

`appendix-cox-regression.pdf`. Code for converting data to long form has been taken from:
`https://rpubs.com/kaz_yos/bio223-recidivism`.

```
dfbeta <- residuals(mod.small, type="dfbeta"); par(mfrow=c(2,2))
for (j in 1:3) {
  plot(dfbeta[,j], ylab=names(coef(mod.small))[j])
  abline(h=0, lty=2)
  }
```

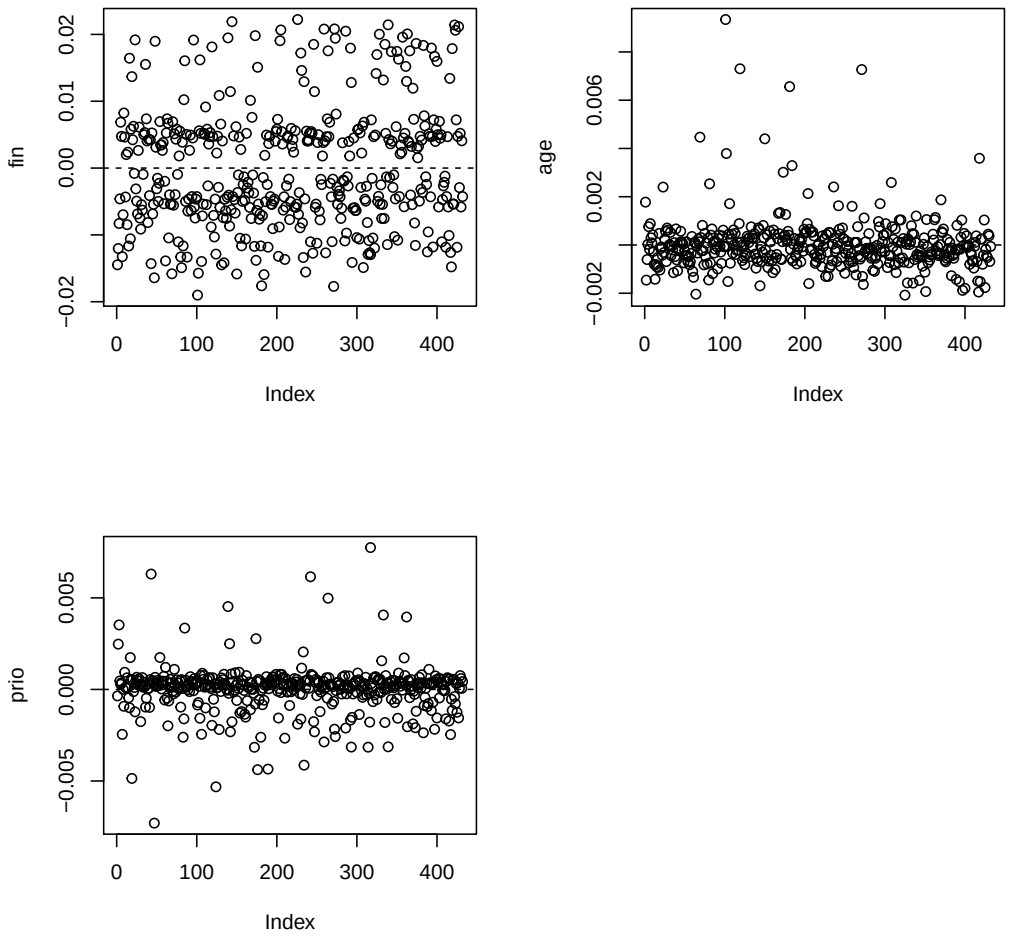


Figure 11.5: DFBeta Plots for Checking for Outliers

Part IV

Machine Learning and Predictive Analytics

Chapter 12

Introduction to Machine Learning Algorithms

Big Data is like teenage sex: everyone talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it, so everyone claims they are doing it.

– *Dan Ariely*

Chapter 13

k-Nearest-Neighbors Classification

`In God We Trust; All Others Must Bring Data`
– Willian Edwards Deming

We will begin our exploration of machine learning algorithms today. We will begin with one of so called *lazy learning* algorithms - Classification using Nearest Neighbors. It is essentially a process of discovery - classifying data by placing it in the category with the most similar or *nearest* neighbors.

Despite the simple basic idea, nearest neighbor methods are extremely powerful and popular. Popular success examples like Netflix Challenge, Amazon Prediction Engine etc have used versions of this algorithm. This algorithm works best for classification tasks where relationships between features and target classes are complex and difficult to understand but the class types themselves are fairly homogeneous. If the groups are not well differentiated then nearest neighbor algorithm does not provide optimal results.

13.1 kNN Algorithm

kNN algorithm starts with a training dataset where records are already classified into target classes. We now attempt to classify a test dataset which is unclassified but otherwise has same features as the training dataset. For each record in the test dataset, kNN algorithm identifies `k` records in the training data that are "nearest" or closest in features - here `k` is an integer. The test record is assigned the class of the majority of the `k` nearest neighbors.

kNN algorithm treats dataset features as coordinates in a multidimensional feature space. Two dimensional feature spaces can be easily visualized as a scatter plot. Higher dimensional features spaces are difficult to visualize but the essential idea remains the same. When trying to categorize a new record, we need to calculate its distance to records in the training dataset. Distances can be calculated as the Cartesian coordinate distance - also known as Euclidean Distance.

13.2 Calculating Distance

If we have two records p and q; and p_i and q_i refer to the i^th feature of p and q respectively; then the distance between p and q can be calculated as:

$$dist(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2}$$

Once we have distances with different records, we can sort them in increasing order. If we were to use $k = 1$ then we will just assign the record to the same category as the record

with the smallest distance. If we use $k = 3$ then we look at the closest three neighbors and assign the record to the category that is most applicable within the three neighbors.

13.3 Choosing An Appropriate k

Choosing an appropriate k should balance between overfitting and underfitting the training data. This problem is commonly known as the **bias-variance tradeoff**. Choosing a large k reduces the impact or variance caused by noisy data - but runs the risk of being biased against smaller but important patterns. Typically, smaller values allow for more complex decision boundaries that fit the training data better. In practice, choosing k depends upon the difficulty of the classification task and the number of records in the training data. Typically, k is set somewhere between 3 and 10. A common practice is to set k equal to the square root of the number of records in training dataset.

13.4 Rescaling Data for kNN

As we depend on Euclidean Distance, the scale of variables matter a lot. Before we can apply kNN, we must transform variable to comparable scale. Following are the common methods of feature rescaling:

Min-Max Normalization Convert all values on a 0 to 1 scale using the formula below to get normalized values.

$$X_{new} = \frac{X - min(X)}{max(X) - min(X)}$$

z-Score Standardization Subtract mean value of the feature with the feature value and then divide by the standard deviation of the feature.

$$X_{new} = \frac{X - Mean(X)}{StdDev(X)}$$

z-Score essentially provides the number of standard deviations from the mean. Z-scores have a mean of 0 and distribution of 1. Unlike the normalized values above, z-score values are unbounded with a range of $-\infty$ to ∞.

Dummy Coding For nominal values, we use dummy coding of 1 or 0. If the feature has several levels then it will require several dummy variables (one less than the number of features). Dummy coding has the advantage of being on the same scale as normalized data.

13.5 Illustrated Example: Breast Cancer Diagnosis

We will illustrate kNN using "Breast Cancer Wisconsin Diagnostic" dataset from the UCI Machine Learning Repository. The dataset includes 569 examples of cancer biopsies each with 32 features. Of these 32, one is identification number and another is the cancer diagnosis (the target classification). Rest 30 are the features we need to use to figure out kNN classification.

```
#Import Data
wbcd <- read.csv("wisc_bc_data.csv", stringsAsFactors = FALSE)
#str(wbcd) # Lets check what we have in this dataset
# Str command commented for brevity in output
```

As the first variable *id* is only a unique identifier, it is not likely to add any value to our model, so we can safely drop it from the dataset.

```
wbcd <- wbcd[-1]
```

The second variable, *diagnosis* is of significant interest as this is the target classification - in this case a variable with two values B for Benign and M for Malignant. We can check details of this variable. We will also convert the variable as a factor to recognize its essential nature.

```
table(wbcd$diagnosis)
```

```
##
##   B   M
## 357 212
```

```
wbcd$diagnosis <- factor(wbcd$diagnosis, levels = c("B", "M"),
                         labels = c("Benign", "Malignant"))
round(prop.table(table(wbcd$diagnosis)) * 100, 1)
```

```
##
##    Benign Malignant
##      62.7      37.3
```

Rescaling the Data

We need to now figure whether our data has a problem of differing scales. Exploring three variables here:

```
summary(wbcd[c("radius_mean", "area_mean", "smoothness_mean")])
```

```
##   radius_mean       area_mean      smoothness_mean
##  Min.   : 6.981   Min.   : 143.5   Min.   :0.05263
##  1st Qu.:11.700   1st Qu.: 420.3   1st Qu.:0.08637
##  Median :13.370   Median : 551.1   Median :0.09587
##  Mean   :14.127   Mean   : 654.9   Mean   :0.09636
##  3rd Qu.:15.780   3rd Qu.: 782.7   3rd Qu.:0.10530
##  Max.   :28.110   Max.   :2501.0   Max.   :0.16340
```

We can clearly see that different variables differ significantly in their scale. So we will need to normalize our data so that we rescale variables to a standard range of values. We can write our own function for normalizing and then use *lapply* to normalize our dataset.

```
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
wbcd_n <- as.data.frame(lapply(wbcd[2:31], normalize))
summary(wbcd_n[c("radius_mean", "area_mean", "smoothness_mean")])
```

```
##   radius_mean       area_mean      smoothness_mean
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.2233   1st Qu.:0.1174   1st Qu.:0.3046
##  Median :0.3024   Median :0.1729   Median :0.3904
##  Mean   :0.3382   Mean   :0.2169   Mean   :0.3948
##  3rd Qu.:0.4164   3rd Qu.:0.2711   3rd Qu.:0.4755
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
```

As we can now see, our three variables in question are now distributed on a standard scale; and are comparable to each other as far as euclidean distance is concerned.

Dividing Data Into Training and Testing Sets

Alright - now we need to train our model. Of course, we shouldn't use the entire dataset to train as then we will have no way to test the model. Lets keep 100 observations as the test dataset and use the rest for training.

```
wbcd_train <- wbcd_n[1:469, ]
wbcd_test <- wbcd_n[470:569, ]
```

We would also want to extract the target variable in factors. These factors would be useful for us in a little bit when we train and test out model.

```
wbcd_train_labels <- wbcd[1:469, 1]
wbcd_test_labels <- wbcd[470:569, 1]
```

Running the Model

Now is the time to train our model on the training data. We will need a package named *class*. Class package includes several classification algorithms.

```
#install.packages("class") #Commented
library(class)
```

We will use the *knn*() function. The function call is pretty simple. It needs a training dataset, a testing dataset, a factor vector with the classification for each row in training data and an integer value corresponding to the number of nearest neighbors k. Since we have 469 observations in the training data, we might want to start with $k = 21$ as it is closest odd number to square root of 469. The function returns a factor vector of predicted values for each record in the test dataset.

```
wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test,
                      cl = wbcd_train_labels, k=21)
```

Model Accuracy

Alright - so we have run a model - is it any good? We should evaluate model performance. In this case it is easy since we already know what should have been the classification. The *CrossTable*() function in the *gmodels* package is a good choice to do this calculation.

```
#install.packages("gmodels") #Comments
library(gmodels)
## Warning:  package 'gmodels' was built under R version 3.6.3
CrossTable(x = wbcd_test_labels, y = wbcd_test_pred,
           prop.chisq=FALSE)
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  100
```

```
##
##
##                   | wbcd_test_pred
## wbcd_test_labels |    Benign | Malignant | Row Total |
## -----------------|-----------|-----------|-----------|
##           Benign |        61 |         0 |        61 |
##                  |     1.000 |     0.000 |     0.610 |
##                  |     0.968 |     0.000 |           |
##                  |     0.610 |     0.000 |           |
## -----------------|-----------|-----------|-----------|
##        Malignant |         2 |        37 |        39 |
##                  |     0.051 |     0.949 |     0.390 |
##                  |     0.032 |     1.000 |           |
##                  |     0.020 |     0.370 |           |
## -----------------|-----------|-----------|-----------|
##     Column Total |        63 |        37 |       100 |
##                  |     0.630 |     0.370 |           |
## -----------------|-----------|-----------|-----------|
##
##
```

As we can see, there were 69 true negative cases - the mass was benign - and the kNN algorithm correctly identified each of the cases. There were 37 true positive cases where the model and the test data agree on identification of malignant cases. The model presents 2 false negatives and 0 false positives. So we have a 98% accuracy - only two cases were wrongly classified.

Improving the Model with z-Scores

We have two ways to see if we can improve this model - we can try different values of k and try a different rescaling technique. Lets start by using the z-score as the rescaling method. We can use the *scale*() function for this.

```
wbcd_z <- as.data.frame(scale(wbcd[-1]))
```

We follow the same process as before for creating training and testing data.

```
wbcd_train <- wbcd_z[1:469, ]; wbcd_test <- wbcd_z[470:569, ]
```

We are ready to run a new model now.

```
wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test,
                      cl = wbcd_train_labels, k=21)
```

Let us see how we did this time.

```
CrossTable(x = wbcd_test_labels, y = wbcd_test_pred,
           prop.chisq=FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
```

```
##
##
## Total Observations in Table:  100
##
##
##                  | wbcd_test_pred
## wbcd_test_labels |    Benign | Malignant | Row Total |
## -----------------|-----------|-----------|-----------|
##           Benign |        61 |         0 |        61 |
##                  |     1.000 |     0.000 |     0.610 |
##                  |     0.924 |     0.000 |           |
##                  |     0.610 |     0.000 |           |
## -----------------|-----------|-----------|-----------|
##        Malignant |         5 |        34 |        39 |
##                  |     0.128 |     0.872 |     0.390 |
##                  |     0.076 |     1.000 |           |
##                  |     0.050 |     0.340 |           |
## -----------------|-----------|-----------|-----------|
##     Column Total |        66 |        34 |       100 |
##                  |     0.660 |     0.340 |           |
## -----------------|-----------|-----------|-----------|
##
##
```

Well - the new model is actually a little bit worse. We now have 95% accuracy. False negatives actually increased - and that is actually a worse mistake to make than a false positive - so all in all - not a change that worked for us.

Trying Different k Values

Okay - we still have the option of trying different k values. Following are the results for different k values using normalized training data.

k value	False Positives	False Negatives	Accuracy Percentage
1	1	3	96%
5	2	0	98%
11	3	0	97%
15	3	0	97%
21	2	0	98%
27	4	0	96%

Figure 13.1: Different k Values

What do you think is the best model. $k = 1$ is better for avoiding false negatives. Original $k = 21$ was better as far as overall accuracy is concerned.

Chapter 14

k-Means Clustering

In God We Trust; All Others Must Bring Data

– *Willian Edwards Deming*

Chapter 15

Finding Patterns using Association Rules

In God We Trust; All Others Must Bring Data

– *Willian Edwards Deming*

Chapter 16

Classification using Support Vector Machines

```
In God We Trust; All Others Must Bring Data
```

– Willian Edwards Deming

Chapter 17

Deep Learning using Neural Networks

In God We Trust; All Others Must Bring Data
– *Willian Edwards Deming*

Chapter 18

Decision Trees and Random Forests

In God We Trust; All Others Must Bring Data

– *Willian Edwards Deming*

Part V

Putting It All Together

Chapter 19

Improving and Combining Models

In God We Trust; All Others Must Bring Data
– *Willian Edwards Deming*

Chapter 20

Some Last Words

Torture the data, and it will confess to anything.
– *Ronald Coase, Economics Nobel Laureate*

Part VI

Artificial Intelligence

Chapter 21

Introduction to Artificial Intelligence

Chapter 22

Inside of an AI: Decision Making

Chapter 23

Natural Language Processing

Chapter 24

Hardware Side of AI: IoT, Autonomous Vehicles, Robotics

Chapter 25

Strategic Implications of AI

Chapter 26

Challenges, Risks and Ethical Considerations

Part VII

Appendices, Bibliography and Index

Appendix A

Data Exploration Example - Titanic

In God We Trust; All Others Must Bring Data
– *Willian Edwards Deming*

```
# TO567 Data Mining Week 1
# Exploring data using R
# Example: Titanic


#=======================================================
#
#Task 1. Data Loading Part
#
#=======================================================

# First, set the working directory.
# Make sure that titanic data file is in the same folder as this script file.
# Session -> Set Working Directory -> To the source file location
# getwd() for checking the working directory location
getwd()
## [1] "D:/Box Sync/2Teaching/AAMLAIBook"
# Read the file and store the data in an object called titanic

#read.csv("filename.csv") for reading a CSV file into R
titanic <- read.csv("datasets/titanic.csv")

# Note that we read the data into a dataframe named titanic


#=======================================================
#
#Task 2. Seeing the data
#
#=======================================================

#titanic will show you the entire data.
#head allows you to preview first 5-6 entries of the data.
#tali allows you to preview last 5-6 entries of the data
#
```

```
# titanic
head(titanic)
```

```
##   PassengerId Survived Pclass
## 1           1        0      3
## 2           2        1      1
## 3           3        1      3
## 4           4        1      1
## 5           5        0      3
## 6           6        0      3
##                                                 Name    Sex Age SibSp Parch
## 1                            Braund, Mr. Owen Harris   male  22     1     0
## 2 Cumings, Mrs. John Bradley (Florence Briggs Thayer) female  38     1     0
## 3                             Heikkinen, Miss. Laina female  26     0     0
## 4       Futrelle, Mrs. Jacques Heath (Lily May Peel) female  35     1     0
## 5                           Allen, Mr. William Henry   male  35     0     0
## 6                                   Moran, Mr. James   male  NA     0     0
##             Ticket    Fare Cabin Embarked
## 1        A/5 21171  7.2500              S
## 2         PC 17599 71.2833   C85        C
## 3 STON/O2. 3101282  7.9250              S
## 4           113803 53.1000  C123        S
## 5           373450  8.0500              S
## 6           330877  8.4583              Q
```

```
tail(titanic)
```

```
##     PassengerId Survived Pclass                                      Name    Sex
## 886         886        0      3     Rice, Mrs. William (Margaret Norton) female
## 887         887        0      2                    Montvila, Rev. Juozas   male
## 888         888        1      1             Graham, Miss. Margaret Edith female
## 889         889        0      3 Johnston, Miss. Catherine Helen "Carrie" female
## 890         890        1      1                    Behr, Mr. Karl Howell   male
## 891         891        0      3                      Dooley, Mr. Patrick   male
##     Age SibSp Parch     Ticket   Fare Cabin Embarked
## 886  39     0     5     382652 29.125              Q
## 887  27     0     0     211536 13.000              S
## 888  19     0     0     112053 30.000   B42        S
## 889  NA     1     2 W./C. 6607 23.450              S
## 890  26     0     0     111369 30.000  C148        C
## 891  32     0     0     370376  7.750              Q
```

```
# str() shows the data type of each column
str(titanic)
```

```
## 'data.frame': 891 obs. of  12 variables:
##  $ PassengerId: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Survived   : int  0 1 1 1 0 0 0 0 1 1 ...
##  $ Pclass     : int  3 1 3 1 3 3 1 3 3 2 ...
##  $ Name       : Factor w/ 891 levels "Abbing, Mr. Anthony",..: 109 191 358 277 16 559 5
##  $ Sex        : Factor w/ 2 levels "female","male": 2 1 1 1 2 2 2 2 1 1 ...
##  $ Age        : num  22 38 26 35 35 NA 54 2 27 14 ...
##  $ SibSp      : int  1 1 0 1 0 0 0 3 0 1 ...
##  $ Parch      : int  0 0 0 0 0 0 0 1 2 0 ...
##  $ Ticket     : Factor w/ 681 levels "110152","110413",..: 524 597 670 50 473 276 86 39
##  $ Fare       : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ Cabin      : Factor w/ 148 levels "","A10","A14",..: 1 83 1 57 1 1 131 1 1 1 ...
```

```
##  $ Embarked   : Factor w/ 4 levels "","C","Q","S": 4 2 4 4 4 3 4 4 4 2 ...
# View() or fix() opens a data viewer
# Not recommended. You should get out of the habit of
# watching your data like you are a primitive Excel user!
# View(titanic)
# fix(titanic)


# names(data) or colnames shows you the column titles.
# nrow(data), ncol(data), dim(data) show the size of the data
# is.data.frame(data) answers whether data is stored in a data frame.

# Try the follwing commands and see what happens.

# names(titanic)
# colnames(titanic)
# nrow(titanic)
# ncol(titanic)
# dim(titanic)
# is.data.frame(titanic)


#=========================================================
#
#Task 3. Summarize and tabulate the data
#
#=========================================================


# Let's summarize entire data set
# summary(titanic)


# Getting information about a particular column
# We can use "titanic$Survived" to refer to the "Survived" column of "titanic" data frame.

# Example - let's do a summary of the Survived column of titanic data frame
# summary(titanic$Survived)



# Tabulate the data
# The "table" function will then go through the column and count the occurrence of each value (in this
# try summary() and table()
# see what these functions generate.

titanic$Survived
##   [1] 0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 1 1 0 1 0 0 1 0 0 1 1 0 0 0 1
##  [38] 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 1 1 0 1 1 0 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0
##  [75] 1 0 0 0 1 1 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 1 0
## [112] 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 0
## [149] 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1
## [186] 0 1 1 0 0 1 0 1 1 1 1 1 0 0 1 0 0 0 0 0 1 0 0 1 1 1 0 1 0 0 0 1 1 0 1 0 1 0
```

```
## [223] 0 0 1 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 1
## [260] 1 0 1 0 0 0 0 0 1 1 1 0 1 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 0 1 1 1 1 0 0 0 0
## [297] 0 0 1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 0 0 1 1 0 1 1 0 0 1 1 0 1 0 1 1 1 1 0 0
## [334] 0 1 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 1 1 1
## [371] 1 0 0 0 0 0 1 1 0 0 0 1 1 0 1 0 0 0 1 0 1 1 1 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0
## [408] 1 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 0 0 1 0 1 0 0 1 0 0 1
## [445] 1 1 1 1 1 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0
## [482] 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 1 0 1 1 0 1 1 0 0 1 0
## [519] 1 0 1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0 0 1 1
## [556] 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 1 1 0 1 1 1 0 0 0 1 0 1 0 0 0 1
## [593] 0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1 0
## [630] 0 1 0 1 0 0 1 0 0 0 0 0 1 0 1 1 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0
## [667] 0 0 0 1 1 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0 1 1 0
## [704] 0 0 0 1 1 1 1 1 0 1 0 0 0 1 1 0 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0
## [741] 1 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0
## [778] 1 0 1 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 0
## [815] 0 0 0 0 0 0 1 1 0 1 0 0 0 1 1 1 1 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0
## [852] 0 0 1 0 1 1 1 1 0 0 0 1 0 0 1 1 0 0 1 0 1 0 0 1 1 0 0 0 1 1 0 0 0 0 0 0 1
## [889] 0 1 0
```

```
summary(titanic$Survived)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0000  0.3838  1.0000  1.0000
```

```
table(titanic$Survived)
```

```
##
##   0   1
## 549 342
```

```
# The function prop.table will convert the table to proportions.
prop.table(table(titanic$Survived))
```

```
##
##         0         1
## 0.6161616 0.3838384
```

```
# We can round the proportion table to 3 decimal points.
round(prop.table(table(titanic$Survived)), digits = 3)
```

```
##
##     0     1
## 0.616 0.384
```

```
################################################################################################
#
# Excercise: Repeat this for titanic$Sex
# The function "summary" summarizes the information contained in this argument. Let us sum
# Do the same thing for the variable sex -- titanic$sex
#
```

```
#############################################################################################
# Answers
# titanic$Sex
# summary(titanic$Sex)
# table(titanic$Sex)
# prop.table(table(titanic$Sex))
# round(prop.table(table(titanic$Sex)), digits = 3)



#Question: How come summary(titanic$Sex) shows counts but summary(titanic$Survived) does not?
#Data type: Sex is stored as a factor (categorical variable), Survived is sotred as a number.
str(titanic)
## 'data.frame': 891 obs. of  12 variables:
##  $ PassengerId: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Survived   : int  0 1 1 1 0 0 0 0 1 1 ...
##  $ Pclass     : int  3 1 3 1 3 3 1 3 3 2 ...
##  $ Name       : Factor w/ 891 levels "Abbing, Mr. Anthony",..: 109 191 358 277 16 559 520 629 417 58
##  $ Sex        : Factor w/ 2 levels "female","male": 2 1 1 1 2 2 2 2 1 1 ...
##  $ Age        : num  22 38 26 35 35 NA 54 2 27 14 ...
##  $ SibSp      : int  1 1 0 1 0 0 0 3 0 1 ...
##  $ Parch      : int  0 0 0 0 0 0 0 1 2 0 ...
##  $ Ticket     : Factor w/ 681 levels "110152","110413",..: 524 597 670 50 473 276 86 396 345 133 ...
##  $ Fare       : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ Cabin      : Factor w/ 148 levels "","A10","A14",..: 1 83 1 57 1 1 131 1 1 1 ...
##  $ Embarked   : Factor w/ 4 levels "","C","Q","S": 4 2 4 4 4 3 4 4 4 2 ...
#=======================================================
#
#Task 4. Create a 2 dimensional contingency table
#
#=======================================================

# We can also create a multi dimensional table, say a table that distributes the passengers according t

# Create a 2 by 2 table (Sex, Survived)
# Then, let's convert the numbers into proportions.
#
# table(titanic$Sex, titanic$Survived)
# prop.table(table(titanic$Sex, titanic$Survived))
# Round to 3 decimal places

table(titanic$Sex, titanic$Survived)
##
##            0   1
##   female  81 233
```

```
##   male   468 109
```

```
table1 <-  prop.table(table(titanic$Sex, titanic$Survived))
round(table1, 3)
```

```
##
##              0     1
##   female 0.091 0.262
##   male   0.525 0.122
```

```
# Let's use prop.table so that the proportions add up to 100% in each row.
# Round the results to 3 decimal places

table2 <- prop.table(table(titanic$Sex, titanic$Survived),1) #1: each row adds up to 1.
round(table2, 3)
```

```
##
##              0     1
##   female 0.258 0.742
##   male   0.811 0.189
```

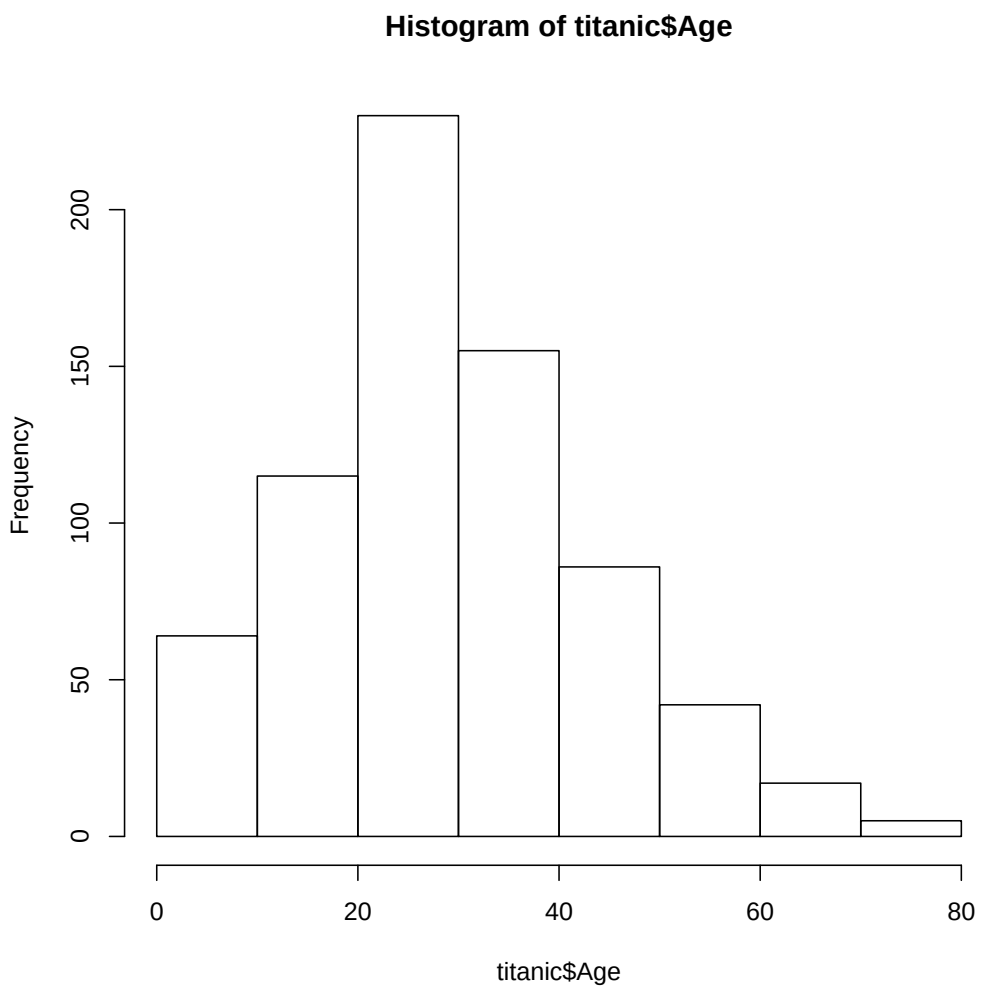
```
table3 <- prop.table(table(titanic$Sex, titanic$Survived),2) #2: each column adds up to 1.
round(table3, 3)
```

```
##
##              0     1
##   female 0.148 0.681
##   male   0.852 0.319
```

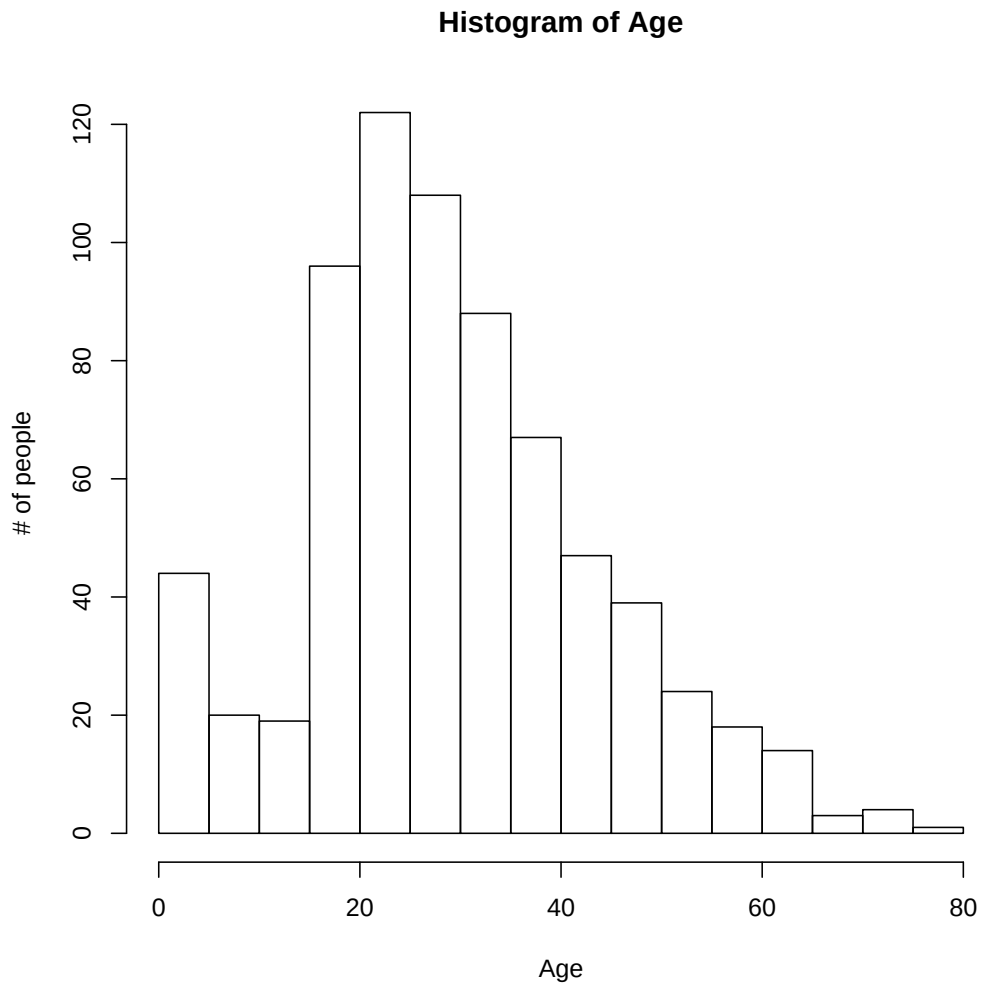
```
################################################################################################
#
# Excercise: Create tables for titanic$PClass and titanic$Survived
# 1. Create count tables (Pclass, Survived)
# 2. Create a proportion table
# 3. Modify a proportion table so that each row adds up to 1.
# 4. Modify a proportion table so that each column adds up to 1.
```

```
################################################################################################
# Answers
# table(titanic$Pclass, titanic$Survived)
# round(prop.table(table(titanic$Pclass, titanic$Survived)),3)
#
# round(prop.table(table(titanic$Pclass, titanic$Survived), 1),3)  #1: each row adds up to
# round(prop.table(table(titanic$Pclass, titanic$Survived), 2),3)  #2: each column adds up
```

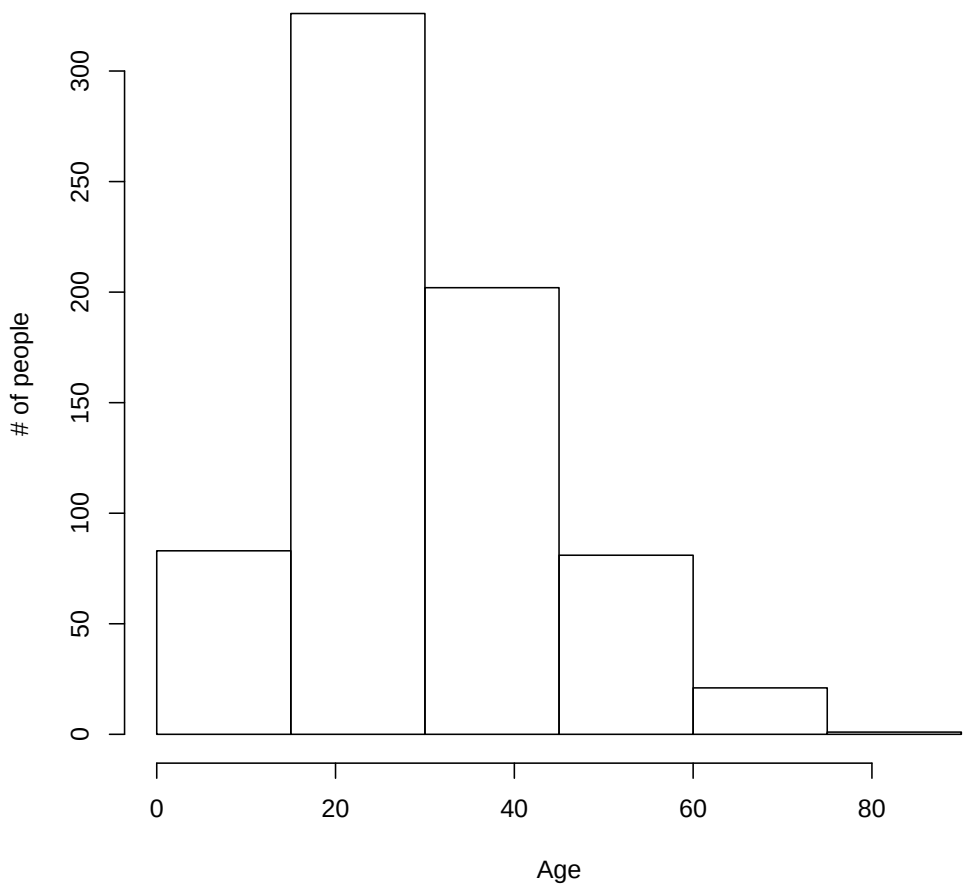
```r
#=========================================================
#
#Task 5. Histogram
#
#=========================================================

# Now, let us summarize the information in the Age column.

summary(titanic$Age)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    0.42   20.12   28.00   29.70   38.00   80.00     177
# Let's draw a histogram of age distribution

hist(titanic$Age)
```




```r
hist(titanic$Age, xlab="Age", ylab="# of people",  main = "Histogram of Age", breaks =12)
```




```
hist(titanic$Age, xlab="Age", ylab="# of people",  main = "Histogram of Age", breaks=seq(0
```




```
#################################################################################################
# Exercise
# 1. Summarize titanic$Fare
# 2. Draw a histogram of $Fare
# 3. Draw a histogram of $Fare with 12 break points
# 4. Draw a histogram of $Fare with breaks = seq(0,1000,by 50)









#################################################################################################
# Answers
# summary(titanic$Fare)
# hist(titanic$Fare)
# hist(titanic$Fare, xlab="Fare", ylab="# of people", col="red", main = "Histogram of Fare", breaks =12
# hist(titanic$Fare, xlab="Fare", ylab="# of people", col="red", main = "Histogram of Fare", breaks=seq
```

```
#=========================================================
#
#Task 6. Adding a column (new variables)
#
#=========================================================

# We will add another column to our data frame.
# This column will be titled "Child" and it will be equal to 1 if the passenger is less th



titanic$Child <- 0 #Default value = 0
titanic$Child[titanic$Age<18] <- 1 #Change the value to 1 if Age< 18
titanic$Child
##   [1] 0 0 0 0 0 0 0 1 0 1 1 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
##  [38] 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0 0 1 0 0
##  [75] 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [112] 1 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1
## [149] 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 1
## [186] 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
## [223] 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [260] 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
## [297] 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
## [334] 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [371] 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [408] 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0
## [445] 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1
## [482] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [519] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0
## [556] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [593] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
## [630] 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [667] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0
## [704] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
## [741] 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
## [778] 1 0 0 1 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1
## [815] 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1
## [852] 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
## [889] 0 0 0
################################################################################################
# Exercise
#
```

```r
# You will add a column called FClass whose value is 1 if the passenger is in the first class and 0 oth
```

```r
################################################################################
# Answers
# titanic$FClass <- 0
# titanic$FClass[titanic$Pclass == 1] <- 1
# titanic$FClass
```

```r
#=========================================================
#
#Task 7. aggregate() and barplot()
#
#=========================================================

# We will use "aggregate" function to count how many passengers survived, grouped by their Sex and whet

# The "aggregate" function divides the data into groups according to the variable to the right of "~" a
# In this case, it will add the numbers (0s and 1s in the Survived column) for four different groups of

# aggregate(Survived ~ Child, data=titanic, sum)
# This means that R will sum Survived column for two subgroups Child=1 or Child =0

# aggregate(Survived ~ Child + Sex, data=titanic, sum)
# This means that R will sum the values in Survived and show the results grouped by Child (0 or 1) and

aggregate(Survived ~ Child, data=titanic, sum)
##   Child Survived
## 1     0      281
## 2     1       61
aggregate(Survived ~ Child, data=titanic, mean)
##   Child  Survived
## 1     0 0.3611825
## 2     1 0.5398230
aggregate(Survived ~ Child + Sex, data=titanic, sum)
```

```
##   Child    Sex Survived
## 1     0 female      195
## 2     1 female       38
## 3     0   male       86
## 4     1   male       23
```

```
aggregate(Survived ~ Child + Sex, data=titanic, mean)
```

```
##   Child    Sex  Survived
## 1     0 female 0.7528958
## 2     1 female 0.6909091
## 3     0   male 0.1657033
## 4     1   male 0.3965517
```

```
#Above statement is the same as
# aggregate(titanic$Survived, list(Child = titanic$Child, Sex= titanic$Sex), sum)
# aggregate(titanic$Survived, list(Child = titanic$Child, Sex= titanic$Sex), mean)
```

```
# We will use "aggregate" function again, this time to count how many passengers belong to

# Note that legnth measures how many people are in each subgroup (regardless of their surv
# sum measures the number of people survived.
# mean  will be the sample proportion of each category

# If you do length, it just counts the number of observations in the survived columns for
# not the number of people survived.
```

```
aggregate(Survived ~ Child + Sex, data=titanic, length)
```

```
##   Child    Sex Survived
## 1     0 female      259
## 2     1 female       55
## 3     0   male      519
## 4     1   male       58
```

```
# Since Survived = 1 or 0, if you use sum, it counts the number of people survived for eac
# The number of people survived
aggregate(Survived ~ Child + Sex, data=titanic, sum)
```

```
##   Child    Sex Survived
## 1     0 female      195
## 2     1 female       38
## 3     0   male       86
## 4     1   male       23
```

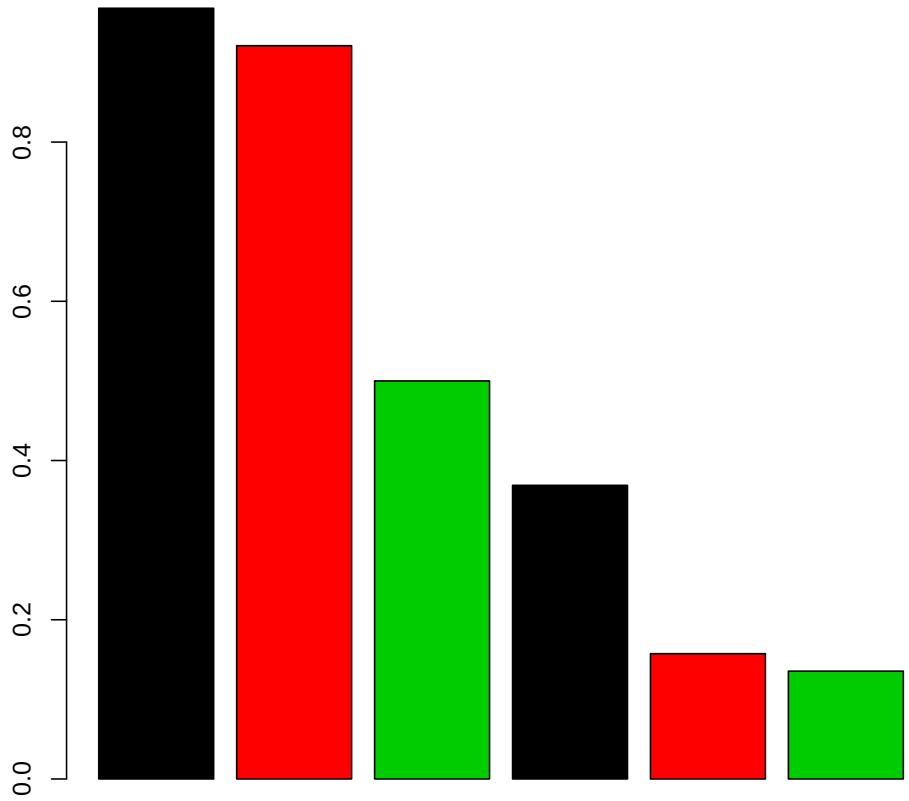
```
# Proportion of people survived
temp_t <- aggregate(Survived ~ Child + Sex, data=titanic, mean)
temp_t
```

```
##   Child    Sex  Survived
## 1     0 female 0.7528958
## 2     1 female 0.6909091
## 3     0   male 0.1657033
## 4     1   male 0.3965517
```

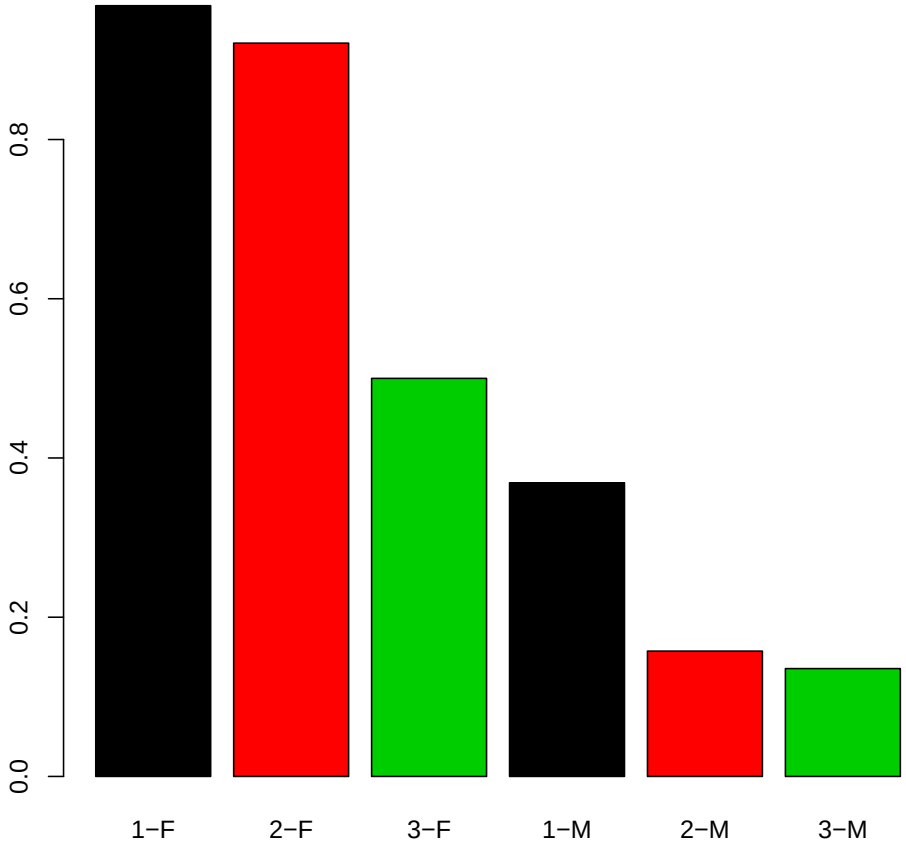
```
temp_t$Survived <- round(temp_t$Survived,3)
```

```
temp_t
##   Child    Sex Survived
## 1     0 female    0.753
## 2     1 female    0.691
## 3     0   male    0.166
## 4     1   male    0.397
################################################################################################
# Exercise
#
# 1. Aggregate and compute the sum of Fare,  grouped by Child (0 or 1)
# 2. Aggregate and compute the average of Fare by Child and Sex









################################################################################################
# Answer
#
# aggregate(Fare ~ Child, data=titanic, sum)
# aggregate(Fare ~ Child + Sex, data=titanic, mean)









#=======================================================
#
#Task 7. aggregate() and barplot()
#
#=======================================================

# Finally, we check how passengers did when we group them according to PClass, and Sex. We use the "agg

# Do the following

breakdown <- aggregate(Survived ~ Pclass + Sex, data=titanic, mean)

# Observe
breakdown
##   Pclass    Sex  Survived
## 1      1 female 0.9680851
## 2      2 female 0.9210526
## 3      3 female 0.5000000
## 4      1   male 0.3688525
## 5      2   male 0.1574074
## 6      3   male 0.1354467
colnames(breakdown)
## [1] "Pclass"   "Sex"      "Survived"
```

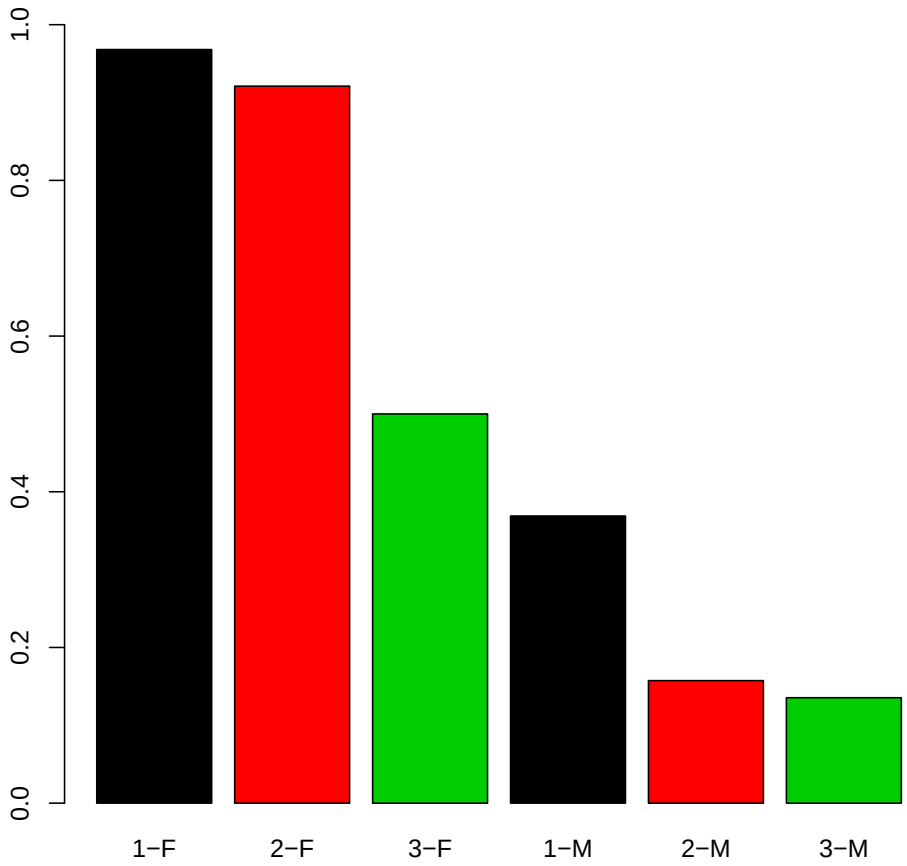
```
barplot(breakdown$Survived, col=breakdown$Pclass)
```




```
# Add a label to each bar using names.arg

barplot(breakdown$Survived, col=breakdown$Pclass, names.arg = c("1-F", "2-F", "3-F", "1-M"
```




```
# Next week, you will learn how to enhance a graph as above.

# Change the y-axis to (0,1) using ylim = c(0,1)
barplot(breakdown$Survived, col=breakdown$Pclass, names.arg = c("1-F", "2-F", "3-F", "1-M", "2-M", "3-M
```




```
##############################################################################
# Exercise
#
# We want to examine how survials are related to fares.
# Since Fare is a continuous variable, aggreate(Survived ~ Fare, data=titanic, mean) is no
# To see this, try

aggregate(Survived ~ Fare + Sex, data=titanic,mean)
##         Fare    Sex   Survived
## 1     6.7500 female 0.00000000
## 2     7.2250 female 1.00000000
## 3     7.2292 female 1.00000000
## 4     7.2500 female 1.00000000
## 5     7.4958 female 1.00000000
## 6     7.5500 female 0.50000000
## 7     7.6292 female 0.00000000
## 8     7.6500 female 1.00000000
## 9     7.7333 female 1.00000000
## 10    7.7375 female 1.00000000
## 11    7.7500 female 0.66666667
```

```
## 12    7.7750 female 0.33333333
## 13    7.7875 female 1.00000000
## 14    7.8292 female 1.00000000
## 15    7.8542 female 0.50000000
## 16    7.8792 female 1.00000000
## 17    7.8958 female 0.00000000
## 18    7.9250 female 0.60000000
## 19    8.0292 female 1.00000000
## 20    8.0500 female 0.00000000
## 21    8.1375 female 0.00000000
## 22    8.6625 female 0.00000000
## 23    8.6833 female 1.00000000
## 24    8.8500 female 0.00000000
## 25    9.3500 female 1.00000000
## 26    9.4750 female 0.00000000
## 27    9.5875 female 0.50000000
## 28    9.8250 female 0.00000000
## 29    9.8375 female 0.00000000
## 30    9.8417 female 1.00000000
## 31   10.4625 female 0.00000000
## 32   10.5000 female 0.87500000
## 33   10.5167 female 0.00000000
## 34   11.1333 female 1.00000000
## 35   11.2417 female 1.00000000
## 36   12.0000 female 1.00000000
## 37   12.2875 female 1.00000000
## 38   12.3500 female 1.00000000
## 39   12.4750 female 1.00000000
## 40   12.6500 female 1.00000000
## 41   13.0000 female 0.85714286
## 42   13.4167 female 1.00000000
## 43   13.5000 female 1.00000000
## 44   13.7917 female 1.00000000
## 45   13.8583 female 1.00000000
## 46   14.4000 female 0.00000000
## 47   14.4542 female 0.20000000
## 48   14.4583 female 0.00000000
## 49   14.5000 female 0.50000000
## 50   15.2458 female 0.33333333
## 51   15.5000 female 0.75000000
## 52   15.7417 female 1.00000000
## 53   15.7500 female 1.00000000
## 54   15.8500 female 1.00000000
## 55   16.0000 female 1.00000000
## 56   16.1000 female 0.66666667
## 57   16.7000 female 1.00000000
## 58   17.4000 female 1.00000000
## 59   17.8000 female 0.00000000
## 60   18.0000 female 0.00000000
## 61   18.7500 female 1.00000000
## 62   19.2583 female 1.00000000
## 63   19.5000 female 1.00000000
## 64   20.2125 female 0.00000000
```

```
## 65   20.2500 female 1.00000000
## 66   20.5250 female 1.00000000
## 67   21.0000 female 0.66666667
## 68   21.0750 female 0.00000000
## 69   22.0250 female 1.00000000
## 70   22.3583 female 1.00000000
## 71   23.0000 female 1.00000000
## 72   23.2500 female 1.00000000
## 73   23.4500 female 0.00000000
## 74   24.0000 female 1.00000000
## 75   24.1500 female 0.33333333
## 76   25.4667 female 0.00000000
## 77   25.9292 female 1.00000000
## 78   26.0000 female 0.85714286
## 79   26.2500 female 1.00000000
## 80   26.2833 female 1.00000000
## 81   26.5500 female 1.00000000
## 82   27.0000 female 1.00000000
## 83   27.7208 female 1.00000000
## 84   27.7500 female 1.00000000
## 85   27.9000 female 0.00000000
## 86   28.7125 female 0.00000000
## 87   29.0000 female 1.00000000
## 88   29.1250 female 0.00000000
## 89   30.0000 female 1.00000000
## 90   30.0708 female 1.00000000
## 91   31.0000 female 1.00000000
## 92   31.2750 female 0.00000000
## 93   31.3875 female 1.00000000
## 94   32.5000 female 1.00000000
## 95   33.0000 female 1.00000000
## 96   34.3750 female 0.00000000
## 97   39.0000 female 1.00000000
## 98   39.4000 female 1.00000000
## 99   39.6000 female 1.00000000
## 100  39.6875 female 0.00000000
## 101  41.5792 female 1.00000000
## 102  46.9000 female 0.00000000
## 103  49.5000 female 1.00000000
## 104  49.5042 female 1.00000000
## 105  51.4792 female 1.00000000
## 106  51.8625 female 1.00000000
## 107  52.0000 female 1.00000000
## 108  52.5542 female 1.00000000
## 109  53.1000 female 1.00000000
## 110  55.0000 female 1.00000000
## 111  55.9000 female 1.00000000
## 112  56.9292 female 1.00000000
## 113  57.0000 female 1.00000000
## 114  57.9792 female 1.00000000
## 115  59.4000 female 1.00000000
## 116  65.0000 female 1.00000000
## 117  66.6000 female 1.00000000
```

```
## 118  69.3000 female 1.00000000
## 119  69.5500 female 0.00000000
## 120  71.0000 female 1.00000000
## 121  71.2833 female 1.00000000
## 122  75.2500 female 1.00000000
## 123  76.2917 female 1.00000000
## 124  76.7292 female 1.00000000
## 125  77.9583 female 1.00000000
## 126  78.2667 female 1.00000000
## 127  78.8500 female 1.00000000
## 128  79.2000 female 1.00000000
## 129  79.6500 female 1.00000000
## 130  80.0000 female 1.00000000
## 131  82.1708 female 1.00000000
## 132  83.1583 female 1.00000000
## 133  83.4750 female 1.00000000
## 134  86.5000 female 1.00000000
## 135  89.1042 female 1.00000000
## 136  90.0000 female 1.00000000
## 137  91.0792 female 1.00000000
## 138  93.5000 female 1.00000000
## 139 106.4250 female 1.00000000
## 140 108.9000 female 1.00000000
## 141 110.8833 female 1.00000000
## 142 113.2750 female 1.00000000
## 143 120.0000 female 1.00000000
## 144 133.6500 female 1.00000000
## 145 134.5000 female 1.00000000
## 146 135.6333 female 1.00000000
## 147 146.5208 female 1.00000000
## 148 151.5500 female 0.33333333
## 149 153.4625 female 1.00000000
## 150 164.8667 female 1.00000000
## 151 211.3375 female 1.00000000
## 152 227.5250 female 1.00000000
## 153 247.5208 female 1.00000000
## 154 262.3750 female 1.00000000
## 155 263.0000 female 1.00000000
## 156 512.3292 female 1.00000000
## 157   0.0000   male 0.06666667
## 158   4.0125   male 0.00000000
## 159   5.0000   male 0.00000000
## 160   6.2375   male 0.00000000
## 161   6.4375   male 0.00000000
## 162   6.4500   male 0.00000000
## 163   6.4958   male 0.00000000
## 164   6.7500   male 0.00000000
## 165   6.8583   male 0.00000000
## 166   6.9500   male 0.00000000
## 167   6.9750   male 0.50000000
## 168   7.0458   male 0.00000000
## 169   7.0500   male 0.00000000
## 170   7.0542   male 0.00000000
```

```
## 171   7.1250   male 0.00000000
## 172   7.1417   male 1.00000000
## 173   7.2250   male 0.10000000
## 174   7.2292   male 0.15384615
## 175   7.2500   male 0.00000000
## 176   7.3125   male 0.00000000
## 177   7.4958   male 0.00000000
## 178   7.5208   male 0.00000000
## 179   7.5500   male 0.00000000
## 180   7.6500   male 0.00000000
## 181   7.7250   male 0.00000000
## 182   7.7292   male 0.00000000
## 183   7.7333   male 0.00000000
## 184   7.7375   male 0.00000000
## 185   7.7417   male 0.00000000
## 186   7.7500   male 0.10526316
## 187   7.7750   male 0.15384615
## 188   7.7958   male 0.33333333
## 189   7.8000   male 0.00000000
## 190   7.8292   male 0.00000000
## 191   7.8542   male 0.11111111
## 192   7.8750   male 0.00000000
## 193   7.8875   male 0.00000000
## 194   7.8958   male 0.02702703
## 195   7.9250   male 0.38461538
## 196   8.0500   male 0.12195122
## 197   8.1125   male 1.00000000
## 198   8.1583   male 0.00000000
## 199   8.3000   male 0.00000000
## 200   8.3625   male 0.00000000
## 201   8.4042   male 0.00000000
## 202   8.4333   male 0.00000000
## 203   8.4583   male 0.00000000
## 204   8.5167   male 1.00000000
## 205   8.6542   male 0.00000000
## 206   8.6625   male 0.09090909
## 207   8.7125   male 0.00000000
## 208   9.0000   male 0.00000000
## 209   9.2167   male 0.00000000
## 210   9.2250   male 0.00000000
## 211   9.3500   male 0.00000000
## 212   9.4833   male 0.00000000
## 213   9.5000   male 0.22222222
## 214   9.8458   male 0.00000000
## 215  10.1708   male 0.00000000
## 216  10.5000   male 0.12500000
## 217  11.1333   male 1.00000000
## 218  11.2417   male 1.00000000
## 219  11.5000   male 0.00000000
## 220  12.2750   male 0.00000000
## 221  12.3500   male 0.00000000
## 222  12.4750   male 1.00000000
## 223  12.5250   male 0.00000000
```

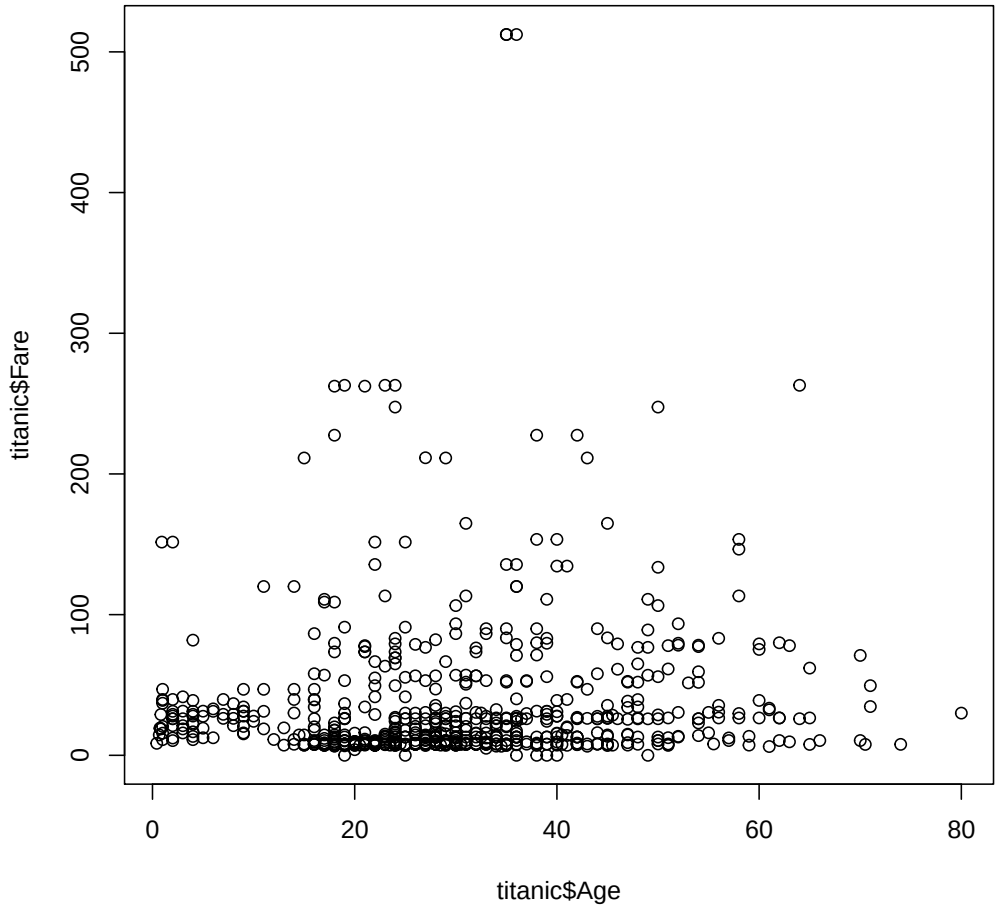
```
## 224  12.8750   male 0.00000000
## 225  13.0000   male 0.14285714
## 226  13.5000   male 0.00000000
## 227  13.8625   male 1.00000000
## 228  14.0000   male 0.00000000
## 229  14.1083   male 0.00000000
## 230  14.4000   male 0.00000000
## 231  14.4542   male 0.00000000
## 232  14.4583   male 0.00000000
## 233  14.5000   male 0.20000000
## 234  15.0000   male 0.00000000
## 235  15.0458   male 0.00000000
## 236  15.0500   male 0.00000000
## 237  15.1000   male 0.00000000
## 238  15.2458   male 1.00000000
## 239  15.5000   male 0.00000000
## 240  15.5500   male 0.00000000
## 241  15.7417   male 1.00000000
## 242  15.8500   male 0.00000000
## 243  15.9000   male 1.00000000
## 244  16.1000   male 0.00000000
## 245  17.8000   male 0.00000000
## 246  18.0000   male 0.00000000
## 247  18.7500   male 1.00000000
## 248  18.7875   male 0.50000000
## 249  19.9667   male 0.00000000
## 250  20.2125   male 0.00000000
## 251  20.2500   male 0.00000000
## 252  20.5250   male 0.50000000
## 253  20.5750   male 0.50000000
## 254  21.0000   male 0.00000000
## 255  21.0750   male 0.00000000
## 256  21.6792   male 0.00000000
## 257  22.5250   male 0.00000000
## 258  23.2500   male 1.00000000
## 259  23.4500   male 0.00000000
## 260  24.0000   male 0.00000000
## 261  24.1500   male 0.00000000
## 262  25.4667   male 0.00000000
## 263  25.5875   male 0.00000000
## 264  25.9250   male 0.00000000
## 265  26.0000   male 0.17647059
## 266  26.2500   male 0.00000000
## 267  26.2875   male 1.00000000
## 268  26.3875   male 1.00000000
## 269  26.5500   male 0.50000000
## 270  27.0000   male 0.00000000
## 271  27.7208   male 0.00000000
## 272  27.7500   male 0.00000000
## 273  27.9000   male 0.00000000
## 274  28.5000   male 0.00000000
## 275  29.0000   male 1.00000000
## 276  29.1250   male 0.00000000
```

```
## 277  29.7000   male 0.33333333
## 278  30.0000   male 0.75000000
## 279  30.0708   male 0.00000000
## 280  30.5000   male 0.80000000
## 281  30.6958   male 0.00000000
## 282  31.0000   male 0.50000000
## 283  31.2750   male 0.00000000
## 284  31.3875   male 0.50000000
## 285  32.3208   male 0.00000000
## 286  33.0000   male 0.00000000
## 287  33.5000   male 0.00000000
## 288  34.0208   male 0.00000000
## 289  34.3750   male 0.00000000
## 290  34.6542   male 0.00000000
## 291  35.0000   male 0.00000000
## 292  35.5000   male 0.75000000
## 293  36.7500   male 0.50000000
## 294  37.0042   male 0.50000000
## 295  38.5000   male 0.00000000
## 296  39.0000   male 0.50000000
## 297  39.6000   male 0.00000000
## 298  39.6875   male 0.00000000
## 299  40.1250   male 0.00000000
## 300  41.5792   male 0.00000000
## 301  42.4000   male 0.00000000
## 302  46.9000   male 0.00000000
## 303  47.1000   male 0.00000000
## 304  49.5042   male 0.00000000
## 305  50.0000   male 0.00000000
## 306  50.4958   male 0.00000000
## 307  51.8625   male 0.00000000
## 308  52.0000   male 0.20000000
## 309  52.5542   male 1.00000000
## 310  53.1000   male 0.33333333
## 311  55.4417   male 1.00000000
## 312  55.9000   male 0.00000000
## 313  56.4958   male 0.71428571
## 314  56.9292   male 1.00000000
## 315  57.0000   male 1.00000000
## 316  61.1750   male 0.00000000
## 317  61.3792   male 0.00000000
## 318  61.9792   male 0.00000000
## 319  63.3583   male 1.00000000
## 320  66.6000   male 0.00000000
## 321  69.5500   male 0.00000000
## 322  71.0000   male 0.00000000
## 323  73.5000   male 0.00000000
## 324  76.7292   male 1.00000000
## 325  77.2875   male 0.00000000
## 326  78.8500   male 0.00000000
## 327  79.2000   male 0.33333333
## 328  79.6500   male 0.00000000
## 329  81.8583   male 1.00000000
```

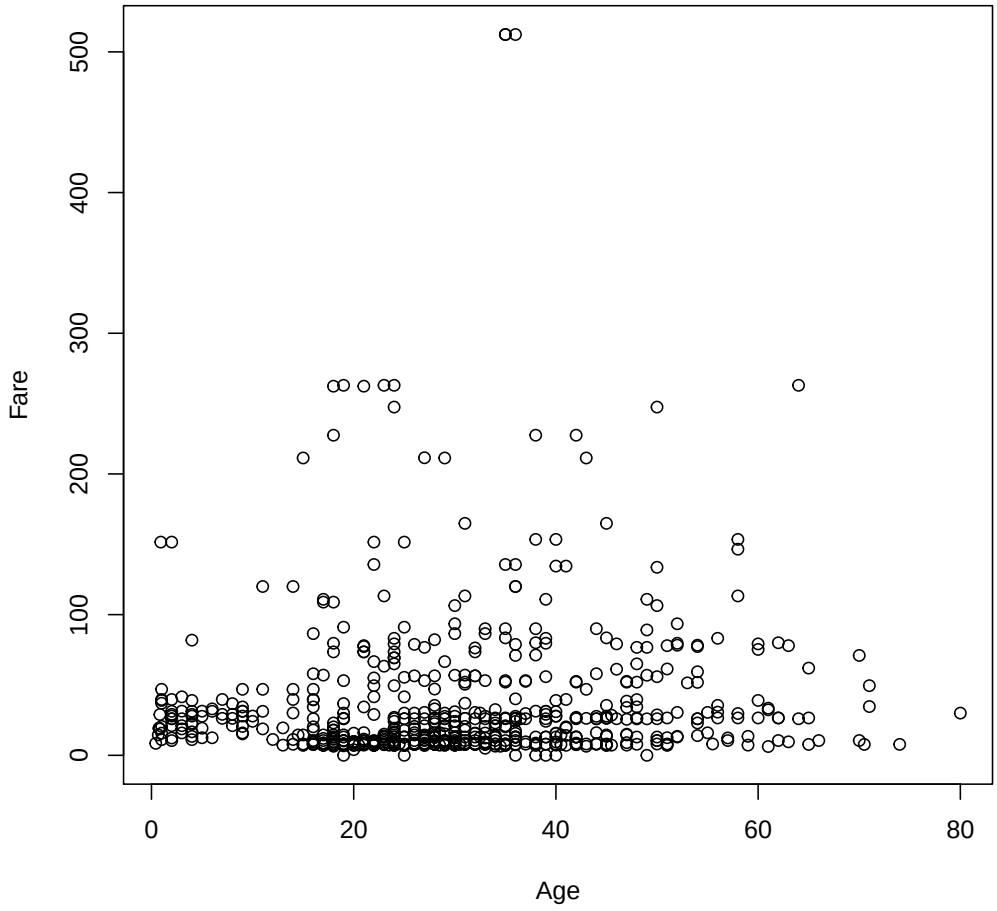
```
## 330  82.1708   male 0.00000000
## 331  83.4750   male 0.00000000
## 332  89.1042   male 1.00000000
## 333  90.0000   male 0.50000000
## 334  91.0792   male 1.00000000
## 335 106.4250   male 0.00000000
## 336 108.9000   male 0.00000000
## 337 110.8833   male 0.50000000
## 338 113.2750   male 0.00000000
## 339 120.0000   male 1.00000000
## 340 133.6500   male 1.00000000
## 341 135.6333   male 0.00000000
## 342 151.5500   male 1.00000000
## 343 153.4625   male 0.00000000
## 344 211.5000   male 0.00000000
## 345 221.7792   male 0.00000000
## 346 227.5250   male 0.00000000
## 347 247.5208   male 0.00000000
## 348 263.0000   male 0.00000000
## 349 512.3292   male 1.00000000
```

```
# Instead,
# we add a new column to our data frame, called "Fare2", to indicate how expensive the fare was for eac
# We will group them into four:
#           those who paid $30+,
#           between 20 and 30,
#           between 10 and 20,
#           and less than 10.

titanic$Fare2<-"30+" #default
titanic$Fare2[titanic$Fare < 30 & titanic$Fare >= 20] <- "20-30"
titanic$Fare2[titanic$Fare < 20 & titanic$Fare >= 10] <- "10-20"
titanic$Fare2[titanic$Fare < 10] <- "<10"


# Now Create a breakdown2:
# breakdown2 <- aggregate(Survived ~ Fare2 + Sex, data=titanic,mean)




# Draw a barplot of breakdown2$Survived



# Try this and see the changes
barplot(breakdown2$Survived, col=breakdown2$Sex,  names.arg =rownames(breakdown2), legend =rownames(brea
## Error in barplot(breakdown2$Survived, col = breakdown2$Sex, names.arg = rownames(breakdown2),
:  object 'breakdown2' not found
# Try the following and see what has changed.
 barplot(breakdown2$Survived, col=breakdown2$Sex, legend =rownames(breakdown2) , names.arg =paste(breako
## Error in barplot(breakdown2$Survived, col = breakdown2$Sex, legend = rownames(breakdown2),
:  object 'breakdown2' not found
```

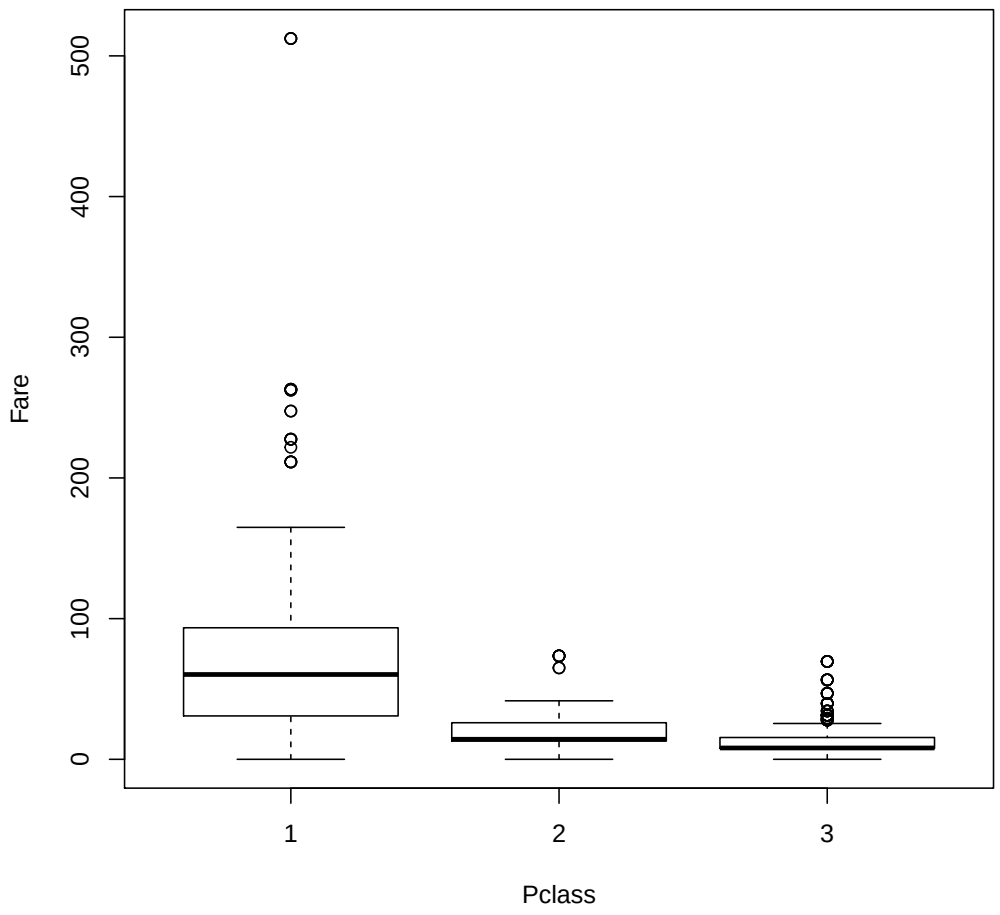
```
#=========================================================
#
#Task 8. Scatter plot, Correlation, Boxplot
#
#=========================================================

str(titanic)
## 'data.frame': 891 obs. of  14 variables:
##  $ PassengerId: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Survived   : int  0 1 1 1 0 0 0 0 1 1 ...
##  $ Pclass     : int  3 1 3 1 3 3 1 3 3 2 ...
##  $ Name       : Factor w/ 891 levels "Abbing, Mr. Anthony",..: 109 191 358 277 16 559 5
##  $ Sex        : Factor w/ 2 levels "female","male": 2 1 1 1 2 2 2 2 1 1 ...
##  $ Age        : num  22 38 26 35 35 NA 54 2 27 14 ...
##  $ SibSp      : int  1 1 0 1 0 0 0 3 0 1 ...
##  $ Parch      : int  0 0 0 0 0 0 0 1 2 0 ...
##  $ Ticket     : Factor w/ 681 levels "110152","110413",..: 524 597 670 50 473 276 86 39
##  $ Fare       : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ Cabin      : Factor w/ 148 levels "","A10","A14",..: 1 83 1 57 1 1 131 1 1 1 ...
##  $ Embarked   : Factor w/ 4 levels "","C","Q","S": 4 2 4 4 4 3 4 4 4 2 ...
##  $ Child      : num  0 0 0 0 0 0 0 1 0 1 ...
##  $ Fare2      : chr  "<10" "30+" "<10" "30+" ...
#Scatter plot plot(X,Y) or plot(Y~X, data)
plot(titanic$Age,titanic$Fare,col="black")
```




```
plot(Fare~Age, data=titanic,col="black")
```




```
cor(titanic$Fare,titanic$Age, use="complete.obs")
## [1] 0.09606669
#Some observations are missing: use="complete.obs"

#Boxplot of Fare grouped by Class
boxplot(Fare~Pclass,data=titanic)
```




```
##############################################################################################
# Exercise
#
# We want to examine how survials are related to fares.
# Since Fare is a continuous variable, aggreate(Survived ~ Fare, data=titanic, mean) is not useful
# To see this, try


# 1. Draw a scatter plot X: Fare, Y: Survived
# 2. Compute the correlation between Fare and Survived
# 3. Boxplot of Fare grouped by Survived
```

```
#############################################################################################
# Answers
# plot(titanic$Fare,titanic$Survived,col="black")
# plot(Survived~Fare, data=titanic,col="black")
# cor(titanic$Fare,titanic$Survived, use="complete.obs")
# Some observations are missing: use="complete.obs"
# boxplot(Fare~Survived,data=titanic)



# Now answers the questions in the class note.
```

Appendix B

R Graphics Example - Movie Death Count

In God We Trust; All Others Must Bring Data

– *Willian Edwards Deming*

```
# TO567 Movie Body Count
# Plotting data using R
# Example: Film death counts

# Download TO567_Body_Count_Practice.R and filmdeathcounts.csv into the same directory.
# Set working directory.
# Open the R file.
# Task 1: Load CSV File and View in a Data Editor
# Read the data from the file "filmdeathcounts.csv" into a data frame called "BodyCountData".


# Task 1: Load the csv data
#
#

BodyCountData <- read.csv("datasets/filmdeathcounts.csv")
str(BodyCountData)
## 'data.frame': 545 obs. of  8 variables:
##  $ Film          : Factor w/ 537 levels "24 Hour Party People",..: 1 2 3 5 6 4 7 8 9 10 ...
##  $ Year          : int  2002 2002 2007 2007 2007 2007 1999 1986 1987 1977 ...
##  $ Body_Count    : int  7 53 212 67 600 45 1 65 199 243 ...
##  $ MPAA_Rating   : Factor w/ 10 levels "Approved","G",..: 8 8 8 8 8 8 8 8 8 6 ...
##  $ Genre         : Factor w/ 208 levels "Action|Adventure",..: 136 199 199 200 88 115 166 62 62 178
##  $ Director      : Factor w/ 330 levels "Aaron Norris",..: 213 49 168 62 330 129 143 158 158 248 ...
##  $ Length_Minutes: int  117 113 100 113 117 122 123 95 105 175 ...
##  $ IMDB_Rating   : num  7.3 7.6 7 6.6 7.7 7.8 6.4 7.5 7.3 7.4 ...
# Task 2: See the columnnames and change it be replacing with the following values
colnames(BodyCountData) <-c("Film","Year","Bodies","MPAA","Genre","Director","Minutes","IMDB")
# Let's change the names of the columns to shorter names.
# We use "colnames" function and assign a vector of new names. The function c() indicates the list of n
```

```
# Task 3: Adding Columns
# Lets Add a column called Filmcount
# Let's add another column, titled "FilmCount", to our data frame and fill it with "1" (nu
# Type BodyCountData["FilmCount"] <- c(1) or BodyCountData$FilmCount <- 1
# Add another column called "BodyPerMin" by BodyCountData[,3] / BodyCountData[,7] (Bodies/
# Add another colunm called "BodyPerHour" by multiplying BodyPerMin by 60




# Task 4: Descriptive Statistics
# Summary the entire  data set
# Summary BodyPerMin
# Find average, median, stdv of BodyPerMin
# Do the same things for Bodies


# Task 5: Histogram of BodyPerMin and Bodies









# Task 6: Bodycounts by each year using aggreate
# Next, we create a table that will show the total number of bodies for each year.
# We use aggreate function.
# For each year, it will "sum" together the numbers in the "Bodies" column of the films ma
```

```
# Task 7: Draw a scatter plot: Y= BodyCountData$Bodies, X= BodyCountData$BodyPerMin.
# Compute the correlation between the two variables.
# Basic grammar for basic scatter plot
# plot(Data name$x-var, Data name$y-var, options)
#   Or
# plot(y-var ~ x-var, data=Data Name, options)


# 1. Draw a scatter plot: BodyPerMin (x) and Bodies (y)
```

```
# 2. Draw a scatter plot: BodyPerMin (x) and BodyPerHour (y)
# 3. Draw a scatter plot: IMDB (x) and BodyPerMin (y)


# Draw a scatter plot IMDB (x) and BodyPerHour (y)
# You should get a straight line.









# Task 8. Use Table Function to Fount
# Step 1. Use table function to find out how many films fell into each MPAA category
# Step 2. Use table functions to create a count table by IMDB rating


# IMDB is a continuous variable. If you make a table it will have many bins
# A better approach is to bin the data. In order to do that, you first need to set the break point.
# Then cut the data according to the breakpoints
# We store this binned data to IMDB_Cut
# Make a table using IMDB_Cut

# table_breaks <- 1:10  # breakpoints =1,2,3,4...10: This will create 10 bins
# BodyCountData$IMDB_Cut <- cut(BodyCountData$IMDB, table_breaks)
# table_IMDB_Cut <- table(BodyCountData$IMDB_Cut)
# table_IMDB_Cut







# Task 9: Box Plot
# We can create a box plot by boxplot(Y~X, data)
# Create a box plot of bodies, grouped by MPAA rating  for BodyPerMin




# Task 10: BPM and MPAA rating
# a)  Use aggregate function and compute the mean of BodyPerMin for each MPAA rating.
# b)  Draw a barplot of BPM (X-axis: MPAA rating, height: Avg BodyPerMin for each rating.
```

Appendix C

Syllabus for TO414 Fall 2016

In God We Trust; All Others Must Bring Data
– *Willian Edwards Deming*

C.1 Course Description

We live in a world awash in data. Companies are increasingly turning to data analytics to extract a competitive edge from data, especially large, complex datasets often called "Big Data". However, companies are increasingly facing the challenge posed by the scarcity of analytical talent - people who can turn data into better decisions, people who can extract insights and information from data. There is growing demand for professionals with strong quantitative skills combined with an understanding of how data analytic techniques can be applied to business contexts and managerial decision making. To help students succeed in this growing field, this course teaches advanced analytical, statistical and data mining tools with an applied focus.

The main focus of this course is to prepare students to model and manage business decisions with data analytics and decision models using real life case contexts and datasets. By the end of this course students will have a better understanding of processes, methodologies and tools used to transform the large amount of business data available into useful information and support business decision making. The course will focus on extracting actionable business intelligence by analyzing traditional business data as well as more recently available datasets such as social media, crowdsourcing and recommendation engines. The course focuses on the powerful, open source (and hence free) data analysis environment R – prior experience with R is NOT required. As this is an applied course, the focus is on application of various statistical and analytical tools and the student is assumed to be familiar with basic statistics at TO 301 level. A working knowledge of spreadsheet applications like MS-Excel is assumed.

C.2 Class Details

- Class Schedule: Tue/Thu 10AM-11:30AM, R1240
- Instructor Information: Dr. Sanjeev Kumar, R3443, sankum@umich.edu, @theSanjeev, 734-615-7064
- Office Hours: TBD, by appointment, my office is always open for my students - come by *anytime*.
- Class Materials: Distributed through the Canvas site for the course, through clarifyR GitHub page and through http://clarifyR.com.

- Teaching Assistants: Class has two dedicated teaching assistants with extensive experience with R. TAs will attend all classes and help students with class exercises. TAs will also hold office hours (times TBD) to help students with individual and group assignments.

C.3 Course Materials

All course materials will be distributed through the Canvas site for the course. Each class will usually have a reading assigned to it and a dataset for students to work on. There is no assigned textbook for the course - the instructor has developed a custom textbook for the class. All datasets, reading materials, assignments and exercises will be distributed through the *clarifyR* packages available on GitHub as an R package.

The class will require students to bring their own laptops to class - any Windows, Mac or Linux machines are fine. However, Chromebooks are not suitable for the class as it is not usually possible to run R/RStudio on Chromebooks effectively on them. In case some students do not have access to suitable laptops or in case of students facing temporary technical problems, about 10 laptops will be provided during each class session for students to use.

Apart from the Canvas site for the course, course material will also be available through http://clarifyR.com, the website for the custom textbook and custom R package for the course available through GitHub.

C.4 Course Grading

- In-Class Quizzes, Exercises, Assignments and Class Participation: 20%
- Individual Homework Assignments: 10 x 4% each = 40%
  1. Basic tasks in R, creating RMarkdown documents and RShiny apps
  2. Data manipulation, data cleaning
  3. Descriptive statistics, basic R graphics
  4. Linear regression
  5. Logistic regression
  6. Principal Component / Factor analysis
  7. Classification using kNN
  8. Clustering using k-Means-Clustering
  9. Classification using black box models
  10. Decision Trees, Random Forests
- Group Projects: 20%. Group project deliverable include a data analysis report (RMD file and html/pdf output, RShiny dashboard) and a 10 min class presentation for Group Project 2.
  - Group Project 1: 7% for Project Report on Data exploration, cleaning, visualization and descriptive statistics
  - Group Project 2: 7% for Project Report + 6% for Project Presentation on Integrated Models for Predictive Analytics - building the best predictive machine learning model
- Individual Final Assessment: 20%

Course grading will follow the usual Ross grading curve for elective courses. Typical grade curve allows for less than 60% of students in A range, less than 90% of students in B and above range and at least 10% of students in B-Minus and below. All efforts will be made to assign students as high a grade as possible within the Ross elective grade curve.

C.5 Class Schedule

All individual assignments are typically due before class in one weeks time. Group projects are typically due before class in two weeks time. The class schedule below provides specific details for submission due dates. Further details will be posted on the Canvas site for the course.

Module 1: Setting the Foundation

Session 1: 09/06 Tue : Course introduction, Big Data and Advanced Analytics, What? Why? How? Our plan for next 14 weeks.
Session 2: 09/08 Thu : Introducing the tools - R, RStudio, RMarkdown, RShiny, clarifyR - Assignment 1 released
Session 3: 09/13 Tue : Managing large datasets, Data manipulation in R
Session 4: 09/15 Thu : R language elements for data manipulation - apply functions, plyr, dplyr, if statements, loops - Assignment 2 released, Assignment 1 due
Session 5: 09/20 Tue : Exploring your data - descriptive statistics
Session 6: 09/22 Thu : Graphics in R - Assignment 3 released, Assignment 2 due
Session 7: 09/27 Tue : Data visualization, ggplot2 graphics
Session 8: 09/29 Thu : Data dashboards, RShiny - Group Assignment 1 released, Assignment 3 due
Session 9: 10/04 Tue : Interactive graphics in RShiny
Session 10: 10/06 Thu : Putting it all together - data exploration, data manipulation, creating graphics, interactive graphics, data dashboards

Module 2: Advanced Data Analytics - Traditional Statistical Methods

Session 11: 10/11 Tue : Linear Regression, Ordinary Least Squares
Session 12: 10/13 Thu : Regression Diagnostics, Interaction effects, Dummy variables - *No class on Tue 10/18* - Assignment 4 released
Session 13: 10/20 Thu : Advanced topics in Linear Regression, Model selection
Session 14: 10/25 Tue : Generalized Linear Models, Logistics Regression - Assignment 5 released, Assignment 4 due
Session 15: 10/27 Thu : Principal Component Analysis, Factor Analysis
Session 16: 11/01 Tue : Review session for Module 2 - Assignment 6 released, Assignment 5 due

Module 3: Big Data Analytics - Machine Learning

Session 17: 11/03 Thu : Introduction to Machine Learning Algorithms
Session 18: 11/08 Tue : Classification algorithms - k-Nearest-Neighbors - Assignment 7 released, Assignment 6 due
Session 19: 11/10 Thu : Clustering algorithms - k-Means-Clustering
Session 20: 11/15 Tue : Classification algorithms - Naive Bayes - Assignment 8 released, Assignment 7 due
Session 21: 11/17 Thu : Finding patterns - Association Rules
Session 22: 11/22 Tue : Classification algorithms - Black box models - Support Vector Machines and Artificial Neural Networks - Assignment 9 released, Assignment 8 due - *No class 11/24 Thu*
Session 23: 11/29 Tue : Decision Trees, Bagging, Boosting, Random Forests - Group Assignment 2 released
Session 24: 12/01 Thu : Combining models for best prediction, Ensemble methods - Assignment 10 released, Assignment 9 due

Session 25: 12/06 Tue : Advanced topics in Machine Learning

Module 4: Review, Assessment and Presentation

Session 26: 12/08 Thu : Reserved class for guest speaker - Assignment 10 due
Session 27: 12/13 Tue : Group Assignment 2 Presentation (13 groups, 5 mins each). Group project reports due before class.
Final Assessment : During exam week at the assigned exam time.

C.6 Class Policies

Laptops You may use your laptop (or tablet or smart phone) during class time, but only for taking notes or retrieving material from Canvas. This is a discussion-oriented class and you are expected to stay engaged with the class material. Anyone using their laptops or other devices to do email, check Facebook, or for any other non-class purpose will be appropriately disciplined, which will count against your participation grade. If you struggle with information overload or related issues, we are happy to discuss positive strategies in office hours.

Attendance Class participation is important. There will be a sign-in sheet, short quiz or class exercise for each class.

Required Readings You should read the material listed in the schedule before the class in which they will be discussed.

Grading Grade distributions will adhere to the Ross School of Business grading policies.

Course LMS We will use Canvas extensively. Whenever possible, class materials will be distributed electronically on Canvas rather than in paper form. Assignments will be submitted there, too. We will also use Canvas announcements to keep you informed of class plans or changes.

Collaboration The RSB Honor Policy is to be observed in all respects. Individual assignments are meant to be individual. For team assignments, there should be no collaboration or communication between teams.

Services for Students with Disabilities The University of Michigan is committed to providing equal opportunity for participation in all programs, services and activities. Students wishing to receive testing accommodations must register with the UM SSD (Services for Students with Disabilities) as soon as possible. It is the student's responsibility to submit their written request and Verified Individualized Services and Accommodations (VISA) form to the instructor as early as possible, but no later than two weeks prior to the test or quiz for which accommodations are requested. In rare cases, the need for an accommodation arises after the two week deadline has passed (for example, a broken wrist). In these cases the student should still contact SSD and the instructor - however, due to logistical constraints we cannot guarantee that an accommodation can be made after the two week deadline has passed. Requests only need to be made once for each class. Requests may be made through the Testing Accommodations survey (posted in Canvas site) and must include a scanned or photographed copy of the VISA form.

Religious Holidays We are aware of the religious holidays scheduled to take place this semester. We are also aware of, and fully supportive of, the University's policies on religious holidays and academic conflict. Please get in touch with the instructor for your specific requests regarding religious holidays accommodation.

Grading Final grades will be determined by applying the Ross grading policy to class point totals. No points will be received for work not completed as scheduled, unless special circumstances apply. In-class exercises generally cannot be made up. If you must be absent, please notify instructor in advance.

C.7 Frequently Asked Questions (FAQs)

1. **How do I get R, RStudio, clarifyR etc?**
   R can be downloaded and installed on student's personal machines for free. Access to R is also available through Univ. of Michigan's Statistics & Computation Service at http://www.itcs.umich.edu/scs.
   RStudio is also available for free for non-commercial usage. The instructor runs a custom RStudio server for students to need in case they temporarily do not have access to RStudio.
   The clarifyR package that includes all course materials including datasets, reading material, assignments, class code files etc can be downloaded from Instructor's GitHub page for the package and also from the webpage for the package http://clarifyR.com.
2. **My laptop is not very powerful - is that a problem?**
   For the most part, R/RStudio applications will not demand much computing power from your laptop - any ordinary laptop will be sufficient. However, some assignments, especially the group assignments deall with complex algorithms on large amounts of data. These tasks will require machines with powerful CPUs and large RAM. It is not uncommon for group assignment tasks to run for several hours on normal laptops.
   To help students run computing intensive tasks, the course provides the following options:
   - Access to Univ. of Michigan's Statistics & Computation Service at http://www.itcs.umich.edu/scs.
   - Access to custom RStudio server run by the instructor.
   - Access to cloud based R servers.
3. **Do I need to know how to program/code before I join the course?**
   It is not essential but it surely is helpful. The course will teach basics of programming in R. This course includes significant amount of technically intensive material. You will find it easier to pick up and understand the content if you have some background in computer programming.
   Experience in any kind of computer programming - even something as elementary as MS-Excel VBA is quite helpful in getting a running start into the course.
4. **More questions will be added to this FAQ as the course proceeds.**

Appendix D

Syllabus for TO628 Winter-B 2017

In God We Trust; All Others Must Bring Data
– *Willian Edwards Deming*

D.1 Course Description

D.2 Course Details

D.3 Course Materials

D.4 Course Grading

D.5 Class Schedule

D.6 Class Policies

D.7 Frequently Asked Questions (FAQs)

Bibliography

[1] Free Software Foundation. The gnu general public license v3.0 - gnu project - free software foundation. https://www.gnu.org/licenses/gpl-3.0.en.html, 2007. (Accessed on 09/23/2016).

[2] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

[3] Andrew Griffin. Facebook is using smartphones to listen to what people say, professor suggests. http://www.independent.co.uk/life-style/gadgets-and-tech/news/facebook-using-people-s-phones-to-listen-in-on-what-they-re-saying-claims/-professor-a7057526.html, 2016. (Accessed on 09/23/2016).

[4] Constance L. Hays. What wal-mart knows about customers' habits - the new york times, Nov 14 2004.

[5] Kashmir Hill. How target figured out a teen girl was pregnant before her father did. http://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/, 2012. (Accessed on 09/23/2016).

[6] Dave Holtz. 8 skills you need to be a data scientist | udacity. http://blog.udacity.com/2014/11/data-science-job-skills.html, 2014. (Accessed on 09/23/2016).

[7] IBM. Ibm - what is big data? https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html, . (Accessed on 09/23/2016).

[8] IBM. Infographic: The four v's of big data | ibm big data & analytics hub. http://www.ibmbigdatahub.com/infographic/four-vs-big-data, . (Accessed on 09/23/2016).

[9] Andrew McAfee, Erik Brynjolfsson, Thomas H Davenport, DJ Patil, and Dominic Barton. Big data: The management revolution. *Harvard Bus Rev*, 90(10):61–67, 2012.

[10] TH Patil and DJ Davenport. Data scientist: The sexiest job of the 21st century. *Harvard Business Review*, 2012.

[11] Wikibooks. Latex - wikibooks, open books for an open world. https://en.wikibooks.org/wiki/LaTeX. (Accessed on 09/23/2016).

Index of R Commands, Packages and Key Concepts