

Arkansas Tech University

Online Research Commons @ ATU

ATU Faculty Open Educational Resources

5-2023

Applied Predictive Analytics

Matt Brown

Arkansas Tech University, hbrown11@atu.edu

Follow this and additional works at: https://orc.library.atu.edu/atu_oer



Part of the [Business Analytics Commons](#)

Recommended Citation

Brown, Matt, "Applied Predictive Analytics" (2023). *ATU Faculty Open Educational Resources*. 3.
https://orc.library.atu.edu/atu_oer/3

This Book is brought to you for free and open access by Online Research Commons @ ATU. It has been accepted for inclusion in ATU Faculty Open Educational Resources by an authorized administrator of Online Research Commons @ ATU. For more information, please contact cpark@atu.edu.

Applied Predictive Analytics

Applied Predictive Analytics

By Matt Brown

First Published May 2023

This work is licensed under a Creative Commons Attribution 4.0 International License.



CC BY-NC-SA includes the following elements:




- BY  – Credit must be given to the creator
- NC  – Only noncommercial uses of the work are permitted
- SA  – Adaptations must be shared under the same terms

Table of Contents

About the Contents.....	5
Data Set Information.....	6
Module 1 Introduction to Applied Predictive Analytics and R	7
Introduction to Applied Predictive Analytics	8
Introduction to R.....	11
Module 1 Assignment	25
Module 2 Exploratory Data Analysis Using R	26
What is Exploratory Data Analysis?	27
EDA in R.....	33
Module 2 Assignment	47
Module 3 Building Predictive Models with Regression and Multiple Regression.....	48
Regression for Predictive Models	49
Cross-Validation with Folding	55
Regression with R.....	58
Multiple Regression	69
Multiple Regression in R	69
Module 3 Assignment	74
Module 4 Building Predictive Models with Trees and Random Forests for Regression	75
Introduction to Regression Trees.....	76
Regression Trees with R.....	80
Random Forests	89
Random Forests in R	90
The Dangers of Not Using Test/Validation Data	97
Module 4 Assignment	101
Group Review Project 1	102
Module 5 Introduction to Classification and Logistic Regression	103
Introduction to Classification.....	104

Introduction to Logistic Regression	108
R for Logistic Regression	109
Module 5 Assignment	120
Module 6 Introduction to Trees and Random Forests for Classification.....	121
Introduction to Classification Trees	122
Classification Trees in R.....	127
Random Forests (for Classification)	132
Random Forests in R (for Classification)	133
Module 6 Assignment	143
Module 7 Introduction to Artificial Neural Networks for Classification	144
Introduction to Artificial Neural Networks	145
Artificial Neural Networks in R.....	150
Module 7 Assignment	155
Module 8 Variable/Model Selection	156
Introduction to Variable Selection	157
Variable Selection in R	160
Module 8 Assignment	169
Module 9 Principal Component Analysis for Predictive Modeling	170
Introduction to Principal Component Analysis	171
An Example of PCA in R for Regression.....	172
Module 9 Assignment	180
Group Review Project 2	181
References.....	183

Explanation of Contents

The materials in this text are a collection of lecture notes for a course entitled “Applied Predictive Analytics” offered at Arkansas Tech University. The purpose of this text is to provide all of the necessary written material to students in this course free of charge. The work to compile the material was supported by a zero-textbook cost / open education resource grant provided by Arkansas Tech University. The text uses R software which is open source and also available to students at no cost. Anyone with interest in additional information regarding this text or in the accompanying data sets may contact the author, Matt Brown, at hbrown11@atu.edu.

About the Author

Dr. Matt Brown is a Professor of Business Data Analytics at Arkansas Tech University where he has taught since 2008. Dr. Matt Brown has taught a variety of data analytics/data science related courses at Arkansas Tech, including classes in predictive modeling, database, data mining, artificial intelligence, statistics, process improvement, application development, Information Systems, using technologies such as SAS, R, Python, Excel, and SQL. Prior to Arkansas Tech, Dr. Brown has nine years’ experience working in industry in the field of data analytics for the Wal-Mart Home Office and Tyson Foods World Headquarters. Dr. Brown’s research interests include a variety applied data analytics topics. Dr. Brown has over fifteen peer-reviewed publications in this field, and he continues to provide data analysis consulting to industry.

Course Structure

The course is organized into nine modules. As it was originally taught, each module corresponded to approximately a week’s worth of material for students. This allowed for one week of review projects and exams three times in the semester and two weeks of group projects. For each predictive modeling technique, the module pattern will typically be a basic example showing how the predictive modeling technique works and two R examples with data sets. The assignment for each module consists of a third data set. Some data sets are used in multiple modules to allow students to compare model results.

Course Prerequisites

Students entering this course were required to have a course in statistics and at least one additional course covering the basics of data analytics.

Data Set Information

The following data sets are referenced in this text:

	Data set	Modules
1	Football2023OffStats.csv	1
2	project.csv	2
3	dataForEDA.csv	2
4	retail EDA.csv	2
5	SeniorSurveyQuestion.csv	2
6	AssignmentdataforEDA.csv	2
7	housing.csv	3, 4
8	tenders.csv	3
9	housing2.csv	3
10	APCuv.csv	3
11	halloweenCandy.csv	3
12	payment.csv	4
13	housing3.csv	4
14	US University Data.csv	4
15	student success.csv	5, 6, 7
16	appt.csv	5, 6
17	titanic.csv	5, 6
18	irs.csv	7
19	product survey.csv	8
20	irs2.csv	8
21	nflcombine.csv	8
22	empRet.csv	9
23	storeCard.csv	9
24	happinessSurvey.csv	9

All data sets were simulated (although often based on real information), with the exception of “US University Data.csv” (data.gov), “titanic.csv” (Kaggle.com), and “happinessSurvey.csv” (data.gov).

Module 1: Introduction to Applied Predictive Analytics and R

Introduction to Applied Predictive Analytics

What does the term “**business data analytics**” mean?

“Business data analytics”, “data analytics”, or “analytics”, is a term used to describe applied data analysis. Generally, if it goes by the name “business data analytics” it is referring to practical data analysis that has direct applications to help businesses and, as a field of study, would be housed in a university’s business curriculum. Data science is almost synonymous, focusing on essentially the exact same tools and techniques. However, data science is typically found in engineering, computing or mathematics curriculum and thus will focus more on theoretical computing and mathematics than the closely related field of analytics. As far as being defined as their own fields of study, analytics and data science are new, emerging shortly after the start of the 21st century. As emerging fields, the scope and definition of these terms is still evolving.

Business data analytics may be best defined by the methodologies used by the field. However, this also creates a difficulty, many of the tools and techniques in analytics are technical and would likely be unknown to someone asking for a definition of “business data analytics”. Regardless, this is still the most accurate way to describe analytics. The following is not a comprehensive list, but should provide a representation of what is meant by the term “business data analytics”.

Knowledge, Tools, and Techniques of Business Data Analytics:

Data

- Collection of data
- Evaluation of data
- Policies and ethical use of data
- Databases

Basic Data Analysis

- Summary statistics
- Visualization
- Exploratory data analysis
- Dashboards

Management Science Techniques

- Optimization
 - Linear programming
 - Genetic algorithms
- Statistical process control
 - Shewhart
 - CUSUM
 - EWMA
- Discrete event simulation
- Expert systems

Hypothesis Tests

- Traditional inferential statistics

- Experiments (designed or observational)
- Continuous/categorical/nonparametric

Predictive modeling

- Methods of model evaluation
- Classification and regression
- Traditional statistics models
 - Regression
 - Logistic Regression
- Machine learning techniques
 - Decision Trees
 - Artificial neural networks
 - Random Forests
 - Support vector machines
 - Naïve Bayes classification

Factor analysis

- Exploratory and confirmatory
- Principle component analysis
- Canonical discriminant analysis
- Structural equation modeling

Data mining

- Many of the previously listed techniques, such as predictive modeling, only for massive quantities of data
- Association analysis
- Clustering
- Anomaly detection

Computing

- Spreadsheets
- Database programming (SQL, non-relational)
- Analytics programming, e.g., R, Python, SAS

Business Context

- Knowledge of basic business terminology and theory
- Writing relevant and ethical data analysis reports

Predictive Modeling

Predictive modeling is the subfield of analytics that focuses on building statistical and machine learning models to make numerical or categorical predictions. Business examples include forecasting sales, predicting the most likely potential customers, determining the correct billing address, examining potential manufacturing changes, among many others. Models often attempt to predict the future or fill in missing information in complex and uncertain situations. This leads to the truism “all models are incorrect”. Indeed, all models have error (if perfect accuracy was possible for a problem, we wouldn’t

use an analytics model), but by using analytics techniques correctly the uncertainty should be accurately quantified for decision makers. Further, not all models perform the same, some models are better than others and often multiple models should be tried and compared to determine the best approach for a problem. For some business problems, the goal is to use a model to improve over a current method of decision making; for others, the goal is to build a model better than what is used by competitors.

Predictive modeling uses techniques from statistics and artificial intelligence's machine learning. While many statistics techniques may be used directly in predictive modeling, it is important to note the differences between statistical hypothesis tests and predictive modeling. In traditional hypothesis tests, the goal is to test a hypothesis using a sample (often as small a sample as possible), controlling for extraneous factors to determine cause and effect. In predictive modeling, while the same techniques may be used, the goal is to make as accurate a prediction as possible. Often in predictive modeling problems, data are more readily available, but there is little or no control over the environment in which the data were collected. In predictive modeling, emphasis is placed on metrics that measure predictive accuracy on data that was *not* used to build the model; often hypothesis tests or metrics calculated using the data used to build the model are of little practical importance to a predictive modeling problem.

Examples of predictive model techniques include: Linear and Multiple Regression, Logistic Regression, Regression Trees, Classification Trees, Random Forests, Artificial Neural Networks, Support Vector Machines, Naïve Bayes classification, Time Series Models, and Spatial Models.

Introduction to R

What is R?

R is an open source implementation of the S language, which was originally created at Bell labs by John Chambers and his colleagues. Ross Ihaka and Robert Gentleman at the University of Auckland in New Zealand initially wrote R. The name "R" comes from the first letter of their names mimicking the one letter title of "S." R is a high-level language designed to make mathematical, statistical, and artificial intelligence algorithms, data mining, data analytics, and graphical techniques easy to implement. Its design goals include being a user-friendly language for data analysis and visualization, and it is widely used in academia, industry, and research. With a vast library of packages and functions, R has become one of the most popular languages for data analysis and visualization, and it has a large and active user community.

Why use R for predictive analytics?

R is a highly effective tool for predictive analytics, offering several advantages over other software packages. The fact that R is a free and open-source product makes it an attractive choice for users of all levels. In addition, the large and active R user community, as well as the abundance of documentation available, ensure that users have access to the resources and support needed to use R. Furthermore, due to the extensive developer base, which primarily comprises well-qualified academic contributors, R often includes state-of-the-art techniques before they are incorporated into proprietary software. In contrast to Python, which is another popular open source option for analytics, R was created by data analysts for data analysts; Python was created by computer scientists for computer scientists to be a general-purpose programming language. Additionally, R possesses the capability to process substantial amounts of data, and various packages are available for interfacing with databases. In summary, R's accessibility, extensive user base, cutting-edge techniques, and data processing capabilities make it a highly recommended choice for predictive analytics.

R is a programming language that supports both functional and object-oriented programming paradigms, as well as imperative and procedural programming. R is an interpreted programming language where very concise programs can lead to quick results. For example, the following R code simulates 300,000 data points from the normal distribution and then plots them using a histogram

```
hist(rnorm(300000))
```

R is designed to be highly extensible: you can define new functions and routines.

Reasons not to use R for predictive analytics

While R is a popular language for data analysis and visualization, some individuals may choose not to use it due to certain limitations. One such limitation is the absence of central support, which means that users do not have access to a dedicated support team. This can be problematic if users encounter issues that they are unable to resolve on their own. Additionally, some R packages are licensed only for non-commercial use, which can restrict the way businesses can utilize the software. Further, if the goal is to

automate or incorporate the predictive analytics into a larger information system, Python may be a better choice since it can also perform many of the same analytics techniques as R yet is also a general-purpose programming language that is already widely used in application development. Ultimately, the choice of language for data analysis will depend on the specific needs and preferences of individual users.

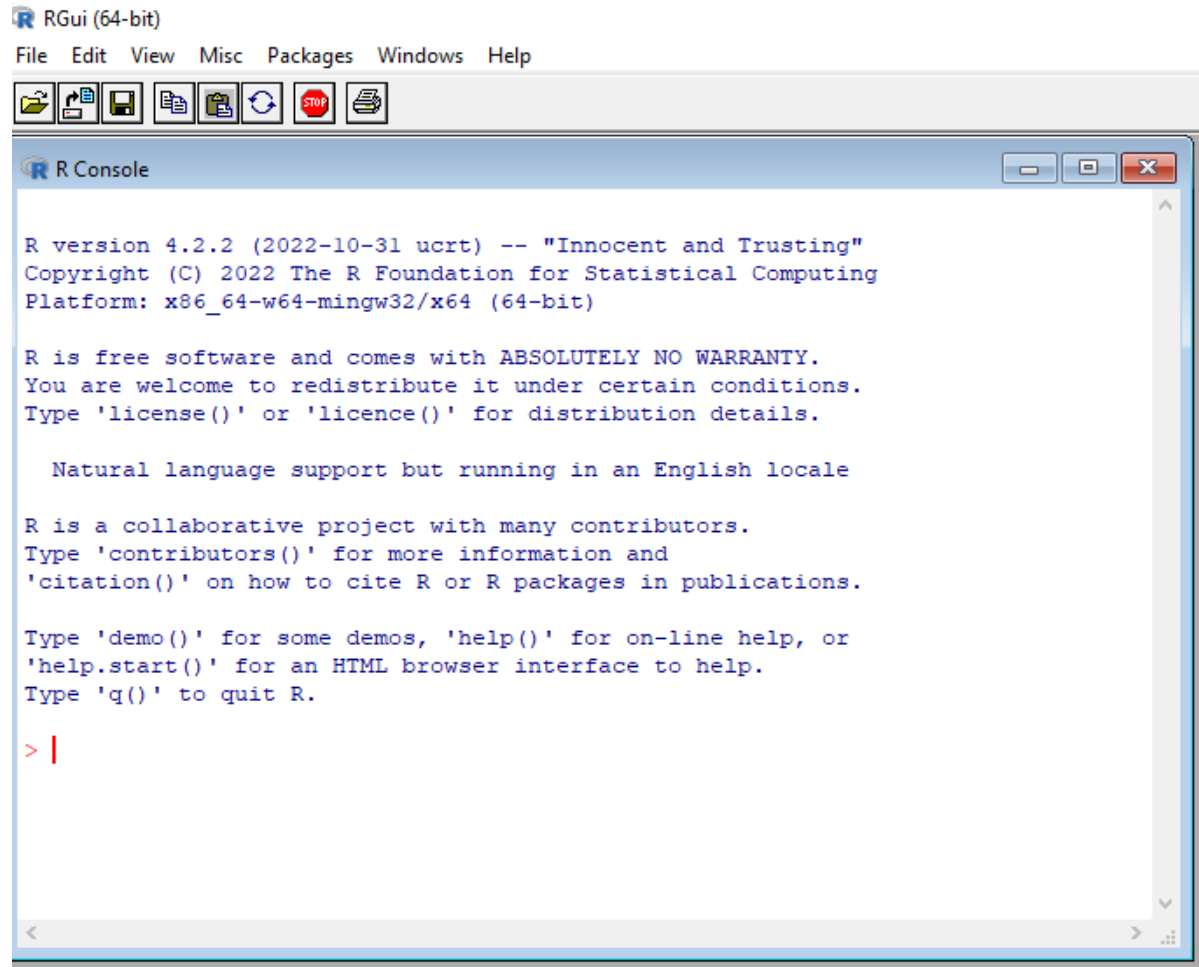
Obtaining and Using R

R can be downloaded and installed from the web at <http://cran.r-project.org/> . R versions exist for Linux, Mac, and Windows operating systems. To install R on a local machine (administrator permissions for the machine are necessary), the steps are as follows: select the appropriate operating system, download the version of R that you need, open the installation file, and follow the instructions—the default options during installations will work for the majority of users. The material presented in this text uses R for Windows. Some differences will exist between versions of R for different operating systems, such as the way files are brought into R since different operating systems reference files differently. Figure 1.1 shows a R session after a successful installation. R commands can be given at the “>” prompt.

R studio is a development environment for R that attempts to make certain tasks easier for R users. R studio can also be downloaded for free. Since R programming commands do not change regardless of whether R studio is used, R studio can certainly be used with this course. However, all examples and assignments made in this text will not require R studio.

R can also be used on the cloud at <https://posit.cloud/> . To use R on the cloud through posit, an account, such as google, is required. A free account is limited to 15 hours of “compute time” each month, which should be enough to complete the requirements for this course. Using posit.cloud is certainly not necessary for this course, but students with difficulty installing or using R on a local machine may prefer this alternative.

Figure 1.1 The R User Interface



Installing Packages in R

If you are new to programming, it should be noted that most programming languages have a limited number of basic commands and capabilities available that are extended through available libraries. This allows storage and memory to be used more effectively since at a given time a user is unlikely to need all of the available features of any given language. As a certain set of programming capabilities are needed, a command to load that library of capabilities is given. R uses this convention as well and installing and loading packages prior to an analysis is a standard part of working with R for analytics.

R comes installed with many capabilities. In addition, numerous “packages” have been developed to extend the capabilities of R. Provided your machine is connected to the Internet, packages (add-on libraries of routines) can be added by typing:

install.packages()

The first time you do this each session, you will be asked to pick a mirror (an R code repository) to install from, you can simply choose “0-cloud” for this option each time. A list of available packages is made available after running the `install.packages()` command. A description of available packages can be

found at: <https://cran.r-project.org/> . Alternatively, you can specify the package you want to install with a command like the following:

`install.packages("nnet")`

The preceding command installs a package for creating artificial neural networks entitled “nnet”. Installing a package downloads the package locally to your machine, so the installation will only need to be done once for that machine. Note that installing a package does not make it available for a particular R session. An additional library command is needed to load the package into memory so that it is available for use:

`library(nnet)`

The nnet package is now loaded and commands from the nnet package will be available until the R session ends. So, if R is closed and reopened, the package will need to be reloaded with an additional library command (but the package will not need to be reinstalled). Note that nnet is enclosed in quotation marks in the install.packages command, but is not enclosed in quotation marks for the library command.

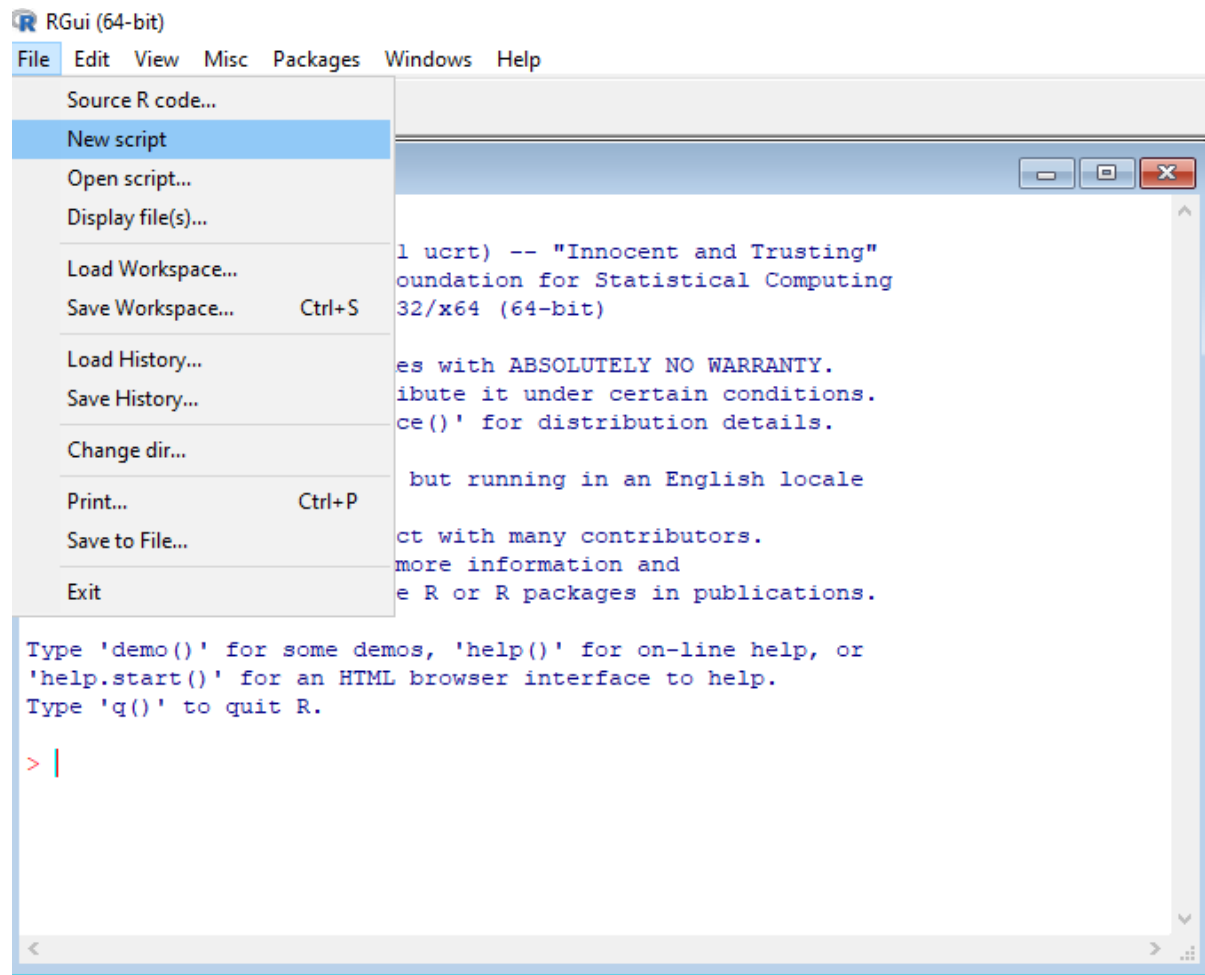
Some R packages require other packages to be loaded in order to be used. When installing a package, you may be prompted to approve additional package dependencies if they are not installed. You may also explicitly install all required dependencies with the following command:

`install.packages('nnet', dependencies = TRUE)`

Getting Started in R

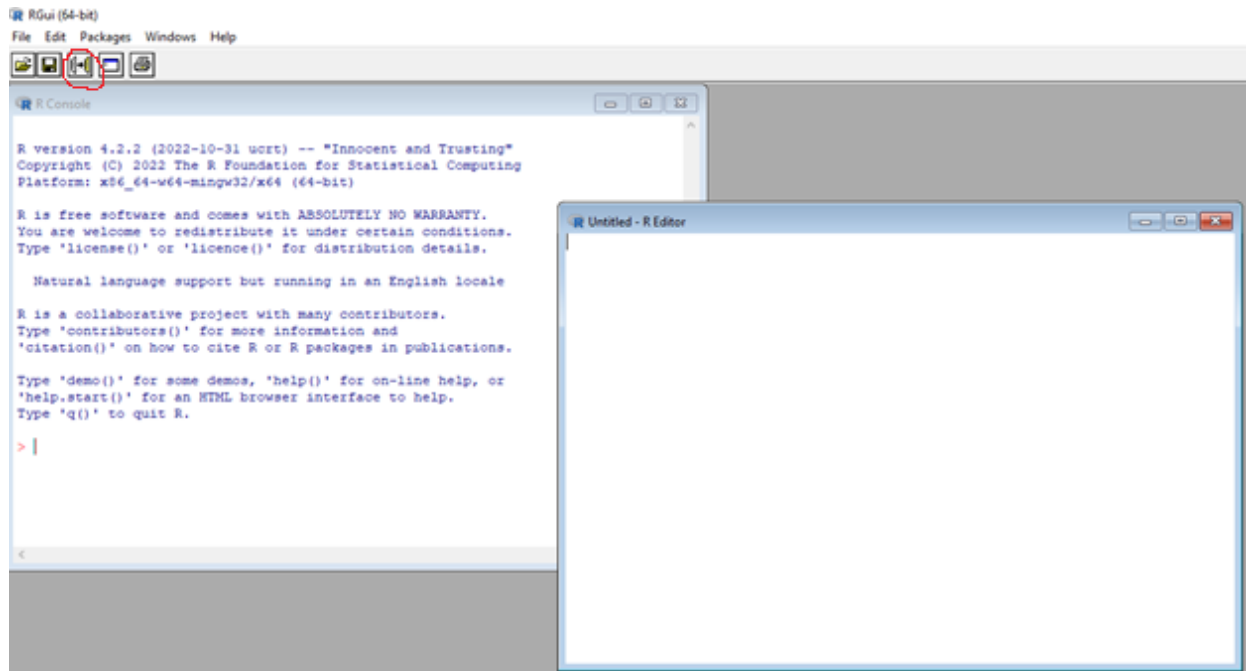
R commands can be directly given at a prompt. For a quick calculation or a quick check of data, the R prompt is the most efficient way of interacting with R. However, for working with larger programs that may need to be edited or debugged, the R script editor is a better way of programming R. For most data analytics projects in this course, you will want to use scripts. To open the script R editor, go to “File” on the R menu and select “New script”, see Figure 1.2 and Figure 1.3:

Figure 1.2 Opening a Script



Commands given at the R prompt (>), will be immediately executed by R when the user hits enter. Commands typed in the script will not be executed until you tell R to execute them, allowing for editing and the execution of several commands at once. To execute a command or several commands in a script, highlight the commands and click on the third icon from the left at the top of the RGui (circled in red in Figure 1.3).

Figure 1.3 An R Session with R Script Editor Open



In the following pages numerous R commands will be given that you should try as you work through this material. To get started with R, try typing the following commands at the prompt, ">" pressing enter after each command:

3*4

12/4

2^2

x<-5

y<-3

z<-x+y

z

#note, anything following a pound symbol is ignored (a way to provide program comments)

The R prompt can be used as a calculator and variables can be assigned with the "<-" symbols (representing an arrow) or with a "=". The results of running the above commands is displayed in Figure 1.4.

Figure 1.4 Example R Session Using the Command Prompt

```
> 3*4
[1] 12
> 12/4
[1] 3
> 2^2
[1] 4
> x<-5
> y<-3
> z<-x+y
> z
[1] 8
> #note, anything following a pound symbol is ignored (a way to provide program$
> |
```

R has built in help that can be accessed by a question mark followed by the command as in

?hist

The above command displays information on the histogram function. If you don't know the exact name of the command you want to use, you can search as follows:

help.search("histogram")

Built-in help and documentation for programming languages such as R have a reputation of being technical and not written for beginners. However, R has a large user base and many example programs and tutorials exist online. An internet search may provide more helpful results for specific questions you have regarding R code. Because a large number of universities use R, for beginning level R information it may also be useful to narrow down your search to .edu sites. This can be done in google by a search such as "Example R program to create a histogram site:.edu". Additionally, there are numerous good general tutorials for R online, for example <https://www.w3schools.com/r/>. Also, large language models recently made available to the public online, such as ChatGPT, can provide R code examples and explanation interactively. However, be aware ChatGPT can make mistakes in R code and R information.

Typing the following command in R, will create a set of data called "testData" with 7 values in it.

testData<-c(88,77,99,55,84,99,83)

This creates a data structure in R that we named "testData". testData is a **vector** in R terminology, meaning it is a structure of data consisting of all the same type of data, testData is also called an **object** in R. Functions like the following can then be used on the data:

summary(testData)

mean(testData)

sd(testData)

plot(testData)

For data objects there are five basic classes: numeric, integer, character, logical, factor. Try the following commands:

```
a<-17
```

```
class(a)
```

```
a<-"Matt"
```

```
class(a)
```

```
class(testData)
```

The class function will identify the class of data in R.

When building models in R, variables are typically either numeric (treated as numbers) or factor (treated as data with categories). Numeric variables are created by default; factors are a special kind of character data structure that must be specified. Try the following commands:

```
testFactor<-c("a", "b", "c", "b", "a", "a", "a")
```

```
#the following tells R to treat testFactor as a factor
```

```
testFactor<-as.factor(testFactor)
```

```
#the following displays the different categories or "levels" in our factor
```

```
levels(testFactor)
```

Common Programming Mistakes to Avoid

R is case-sensitive (capitalization matters), from the previous examples the following commands would have caused errors:

```
#causes an error because "levels" is capitalized:
```

```
Levels(testFactor)
```

```
#causes an error because "Factor" in "testfactor" is not capitalized:
```

```
levels(testfactor)
```

R expects simple quotes, either single or double quotes can be used, but they cannot be the default "fancy" quotations marks that Windows applications use. The following would cause errors:

```
a<-"Matt" #notice the second quotation mark is not a simple quotation mark
```

```
a<-"Matt" #both quotation marks are not simple
```

Spaces cannot appear in the names of data structures, example error:

```
test Data<-c(88,77,99,55) #there cannot be a space between test and Data
```

An equal sign has three uses in R:

1. assignment:

x=6

which is the same as

x<-6

(either can be used).

2. specifying arguments or options in a function:

read.csv(... ,header=TRUE) #this is not the full function, a file path and file name are needed in the ...

3. As a logical operator (to check for equality—notice two equal signs are required in this case):

a==b (this would check if variable a is equal variable b)

A common mistake would be to type a=b, expecting a comparison, but instead reassigning the value of a to the value of b.

Importing and Using a File in R

Data can be read into R in a variety of ways from a variety of formats. In this course we will use comma separated values (.csv) files. This is a common data type that can easily be created from spreadsheet software just by saving the spreadsheet as a csv file. To read a csv file into R entitled Football2023OffStats.csv (note, by default the .csv portion of the file name will not be visible in Windows), you would need to first save the file to a location on your machine where you will be able to locate it. (If you are using a cloud version of R, you will need to upload your file to the cloud.)

For a Windows machine this should appear as follows:

NFL <-read.csv("C:/Users/hbrown11/Desktop/Football2023OffStats.csv",header=TRUE)

The portion of the code highlighted in yellow would change based on the location you specifically saved your file. Not specifying the location and file name correctly when attempting to read a file into R is a common mistake when first learning to use R. Attention must be paid to details like the direction of the “slashes” (which are reversed from the manner in which Windows gives displays a path), capitalization, spaces, etc. If running the above code result in no message being displayed, the R object “NFL” was likely successfully created.

The full file that was read into R could be viewed by the following command (just the name of the R object):

NFL

See Figure 1.5. If the file that was read in is large, it may be much more prudent to only view the first few rows of the file, this can be accomplished with the following command:

head(NFL)

Figure 1.5. Reading a CSV File into R

```
> NFL <-read.csv("C:/Users/hbrown11/Desktop/Football2023OffStats.csv",header=TRUE)
> NFL
```

	Team	RshTD	RecTD	TotTD	TwoPt
1	49ers	18	26	46	2
2	Bears	16	18	34	0
3	Bengals	13	34	47	3
4	Bills	14	32	47	3
5	Broncos	8	16	24	1
6	Browns	20	19	41	3
7	Buccaneers	6	25	32	1
8	Cardinals	14	16	33	5
9	Chargers	15	24	40	1
10	Chiefs	14	40	56	1
11	Colts	7	17	26	0
12	Commanders	8	24	33	3
13	Cowboys	23	28	51	1
14	Dolphins	12	30	44	2
15	Eagles	32	27	61	2
16	Falcons	16	15	35	1
17	Giants	19	16	36	2
18	Jaguars	14	24	40	2
19	Jets	12	15	28	2
20	Lions	22	29	53	4
21	Packers	12	26	41	3
22	Panthers	17	14	33	0
23	Patriots	12	16	34	1
24	Raiders	13	27	41	1
25	Rams	13	18	32	2
26	Ravens	13	17	31	2
27	Saints	14	23	38	1
28	Seahawks	13	29	44	1
29	Steelers	14	10	25	3
30	Texans	6	17	24	2
31	Titans	15	15	32	0
32	Vikings	17	30	48	1

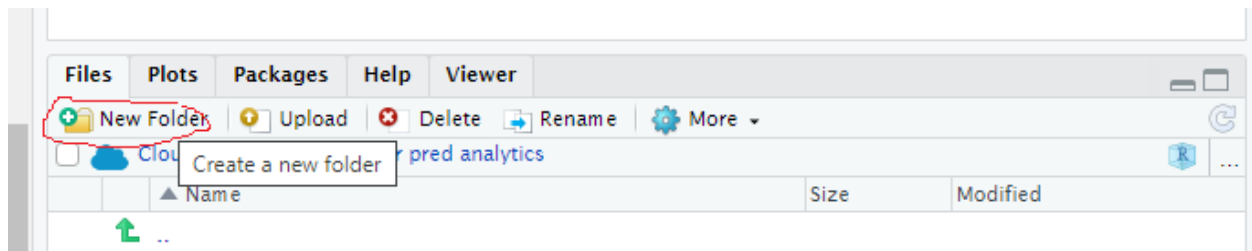
```
> |
```

Reading a File into R Studio Cloud (Optional)

If you are using R through posit's R studio cloud, the following steps can be used to read in the CSV file:

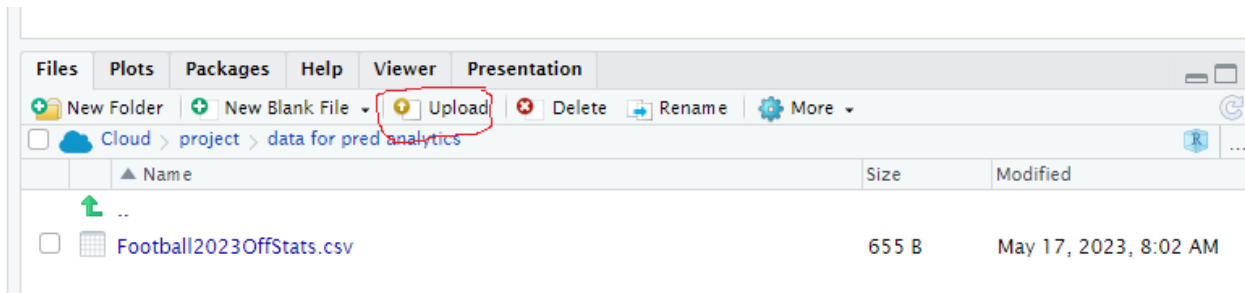
1. Log into R studio cloud
2. Create a new project
3. Locate the "Files Tab" in the pane at the bottom right of the screen
4. Create a "New Folder" (see Figure 1.6) (you can name it anything)

Figure 1.6 Creating a Folder in R Studio Cloud



5. Click on “Upload” and find the file you saved on your machine “Football2023OffStats.csv”
6. Save the uploaded file in the folder you just created, when completed it should appear as follows in Figure 1.7

Figure 1.7 Uploading a File to R Cloud



7. The file can now be loaded into R with the following command typed in the console:

```
NFL<-read.table("/cloud/project/data for pred analytics/Football2023OffStats.csv",header=TRUE)
```

Working with R Data Frames

In predictive modeling we will most often be working with a data structure in R called a data frame (or *data.frame*). Data frames are basically tables, each column represents a different variable that is typically either numeric or a factor. By default, a data frame is created when reading in a file using `read.csv`.

Try the following (after you have read in the `Football2023OffStats.csv` file):

```
str(NFL) #examines the data structure
```

```
head(NFL) #shows the first 6 rows
```

```
View(NFL) #spreadsheet like view of data
```

```
summary(NFL) #five number summary with mean
```

```
NFL$TotTD #return just the column TotTD
```

mean(NFL\$TotTD) #calculate the mean of TotTD

A common requirement after creating a data frame prior to beginning predictive modeling in R is to specify the columns that are factors, example:

NFL\$Team<-as.factor(NFL\$Team)

This does not noticeably change the way the data appear, but behind the scenes in R it designates the column "Team" as a particular data structure that is necessary for some analysis.

Summary of a Few Basic R Commands

The remainder of this module are lists of a few basic R commands. These lists are not comprehensive, but intended to be an introduction to commonly used commands. To "know" a programming language is less about memorizing commands and more about practice and experience writing programs in that language. While not a necessity for being able to build predict analytic models, if you have interest in becoming proficient in R, it is recommended that you try several, if not all, of these commands and perhaps others that you find online.

Commands to work with packages and functions:

install.packages() #install a package

library(package) #load a package

sessionInfo() #see what packages are currently loaded

?function #get help on a specific function

help.search("topic ") #get help on a specific topic

objects() #list the current objects in memory

ls() #list the current objects in memory (same as objects())

Sys.time() #display current data and time

Combining data:

c(data,data,data) #combines elements separated by commas

paste(data,data,data) #concatenates character vectors

cbind(data,data,data) #bind vectors by column

rbind(data,data,data) #bind vectors by rows

Explore the structure of data

head(data) #return first 6 rows

tail(data) #return last 6 rows

str(data) #examines the data structure

View(data) #display data as spreadsheet

length(data) #give the number of elements in a vector

dim(data) #give the number of rows and columns (in a data frame or matrix)

summary(data) #mean and five number summary or a count of the unique levels of a factor

Math/stat functions

sqrt(n) #square root of n, n^{.5} also works

log(n) #natural logarithm

exp(n) #anti natural logarithm

mean(data) #calculate the mean

var(data) #calculate the sample variance

sd(data) #calculate the same standard deviation

max(data) #return the maximum value

min(data) #return the minimum value

median(data) #return the median

round(x,n) #round the numbers in x to n places

Ordering and sequences

sort(n) #sort from least to greatest

rev(n) #reverse order of elements

rep(x,n) #repeat x n times

seq(a, b, c) #create a sequence starting at a and ending at b by c

a:b #create an integer sequence from a to b

Miscellaneous

```
read.csv(file) #read a csv file into a data frame  
read.table(file) #read a text file (tab delimited) into a data frame  
write.csv(dataframe) #write a data frame to a csv file  
ifelse(condition,a,b) #process each element of a data structure, if the condition is meet  
output a, otherwise output b  
#for (i in range){action} loop, example  
or (i in 1:10){print("hi")}  
sample(data) #create a random sample  
#Arithmetic Operations:  
#+, -, *, /, ^ are the standard arithmetic operators.  
#Matrix Arithmetic.  
# * is element wise multiplication  
# %*% is matrix multiplication  
#Assignment  
# To assign a value to a variable use "<-"  
# Common logical operators in R include:  
# != (not equal to), < (less than), > (greater than), <= (less than or equal to), >= (greater than or  
equal to), | (or), & (and), %in% (in).
```

Module 1 Assignment

This assignment is given as a discussion board.

1. Briefly introduce yourself. Please give your name and a short professional bio.
2. After reviewing the material for week 1, create a simple R program. The program must create a variable, create a vector, or create a data frame and do something with that variable, vector, or data frame. Describe what you did using comments in R.
3. Respond to one other student's post by adding to their program, using the variable, vector, or data frame they created, add another feature to their program.

Module 2: Exploratory Data Analysis Using R

What is Exploratory Data Analysis?

Exploratory data analysis (EDA) is a crucial step in the data analysis process that involves examining and understanding the characteristics of the data before formal inference or modeling is performed. In EDA, a primary goal is to gain insights and an understanding of the data by investigating its key features, such as the variation and distribution of the variables, outliers, and missing values. By analyzing the variation in the data, we can determine what is expected and what is not expected, and identify any potential sources of variability. Additionally, by exploring the relationships or potential relationships between variables, we can gain a deeper understanding of the underlying structure of the data and uncover any patterns or trends that may be of interest. EDA helps to inform the choice of statistical methods and models that can be used to analyze the data and can also provide insights for formulating hypotheses for further investigation

Another important use of EDA is to uncover potential information quality issues. Information quality issues can arise in a variety of ways, from manual data entry errors to sensor malfunction or software errors. As a data analyst, finding information quality issues is inevitable. These issues must be dealt with—the adage “garbage in, garbage out” certainly applies to the creation of predictive models. Further, dealing with information quality does not mean simply deleting outliers. In fact, deleting an outlier that legitimately belongs to a set of data can do as much harm to a model as failing to address an information quality issue. Information quality issues are addressed through investigation. EDA may be used to identify suspicious data; an analyst must track down the source and determine if the data arise from an error or are a legitimate record.

In summary, when asked to build a predictive model as an analyst, EDA should be the first step to both get a feel for the data to determine what analysis can be done and to determine if there are potential information quality issues that could impact the validity of the model.

Summary Statistics for EDA

Simple summary statistics are a standard tool for EDA. Commonly used summary statistics include measures for central tendency, including the mean, median, and mode; measures of variability, including, max, min, quantiles/quartiles, standard deviation/variation; and the correlation coefficient as a measure of linear similarity. For categorical data, frequency is used in the calculation mean, standard deviation, and correlation. These simple statistics are extensively documented, and their details can be found in any number of online sources for more information.

The mean and standard deviation and correlation coefficient are three of the most common summary statistics:

$$\mathit{mean}(x) = \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

$$\mathit{standard\ deviation}(x) = s_x = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}$$

$$\text{correlation}(x, y) = r_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

The mean and standard deviation can provide a quick identification of potential outliers. For a variety of reasons, approximately three standard deviations from the mean for a given variable generally provides a good decision boundary for flagging potential outliers. Specifically, any point exceeding three standard deviations may warrant further investigation; the further away from three standard deviations the more unusual. However, the size and distribution of the data also should be taken into consideration. The more data points, the less concern for points exceeding three standard deviations from the mean (for 100,000 values from even a very stable normally distributed process, it would be unusual not to have a few points exceeding three standard deviations). Also, for very skewed data, such as exponentially distributed data, more points would be expected to exceed three standard deviations. For this reason, EDA discoveries are usually based on several facts observed in the data and may involve both summary statistics and visualization of the data.

The correlation coefficient is calculated on data pairs and there must be a natural pairing of the data to calculate correlation. Correlation is a measure of the linear relationship between two variables. Values close to -1 or 1 indicate a strong linear relationship; values close to 0 indicate no linear relationship. Positive correlation indicates as one variable increases, the other variable also increases. Negative correlation indicates as one variable increases, the other variable decreases. Correlation may be visualized with a scatter plot (see Figures 2.1 and 2.2).

Figure 2.1 A Scatterplot Where $r = 0.87$

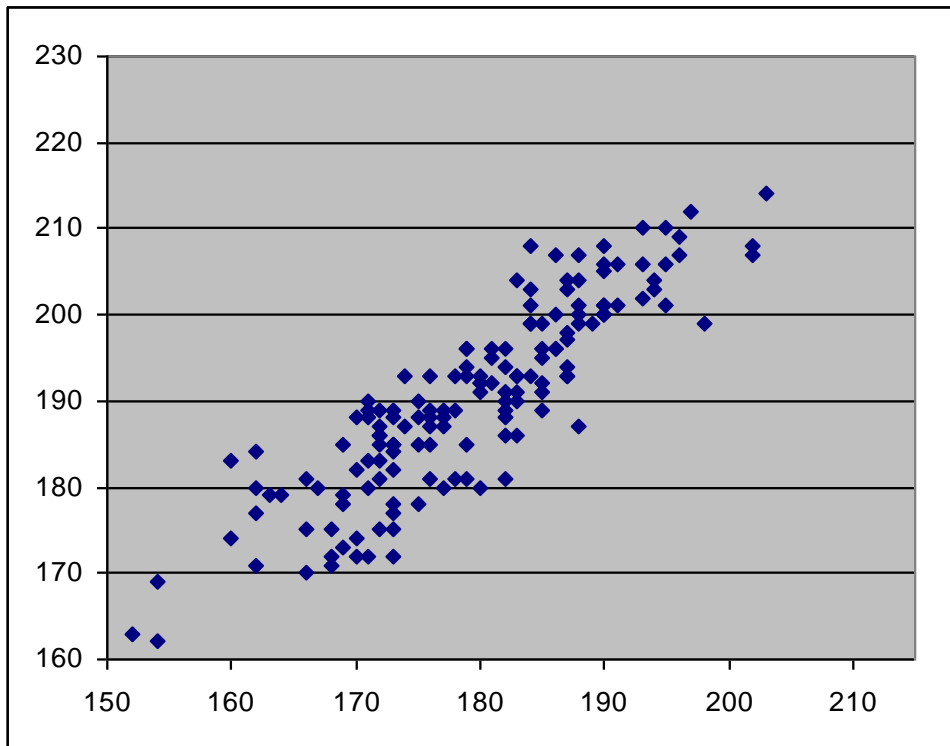
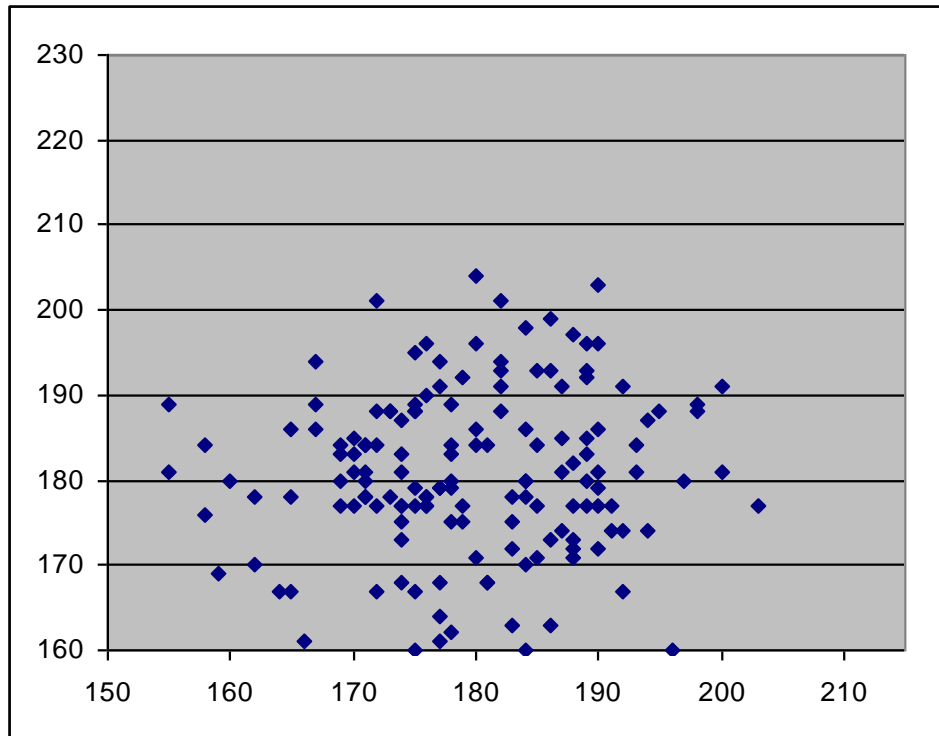


Figure 2.1 A Scatterplot Where $r = 0.08$



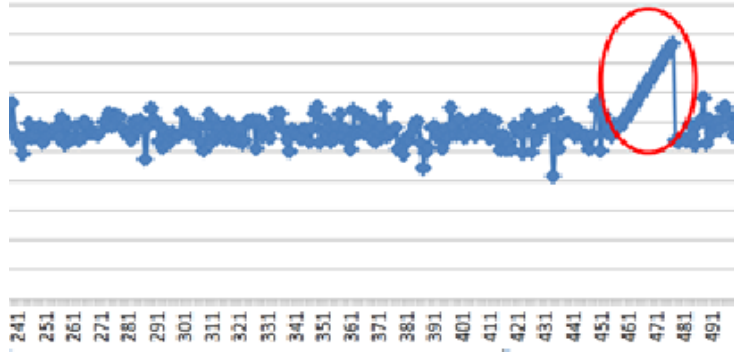
Correlation can be misunderstood and misused in many ways. In EDA, correlation does not imply one variable causes the other; such determinations must be made with carefully controlled experiments. In addition, correlation only measures linear relationships and will miss other more complex patterns in data. Further, correlation is a relative measure without absolutes. In the context of one business problem, a correlation coefficient of 0.39 may be a substantial and potentially profitable relationship; in the context of another, a correlation coefficient of 0.78 may not be strong enough to act upon. Statistical significance of correlation coefficients can likewise be misleading, for larger quantities of data (which are typical for many business data sets) all correlation coefficients, even those close to zero, will be found statistically significant. Finally, decisions about whether to include a variable in a model should generally not be made based on correlation alone due to the potential interaction between variables. Despite these potential issues, correlation coefficients can provide valuable insight into the nature of the relationships that exist in the data.

Data Visualization for EDA

There are numerous data visualizations that are useful for EDA. In fact, entire books and courses are often dedicated to data visualization. However, for the purpose of exploring data (and not creating eye-catching reports for the purpose of attracting customer attention) using just a handful of tools can be quite effective. Run charts, histograms, scatter plots, and box whisker plots are briefly detailed in this section.

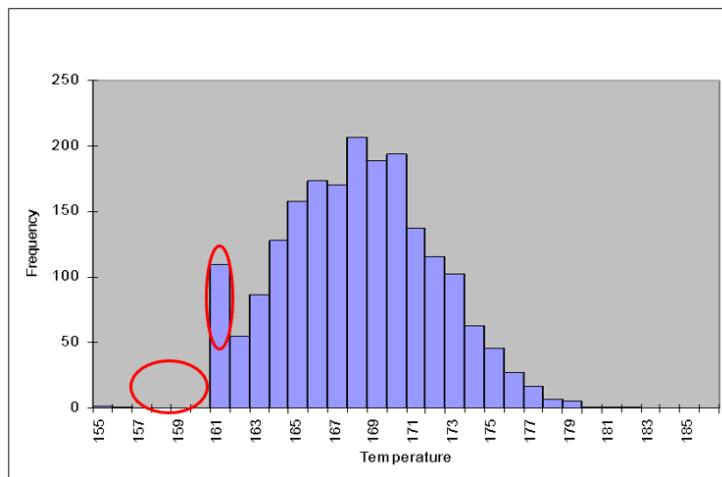
A run chart (sometimes called a line chart) is a plot of a variable over time or in the order the data were collected. It is useful for determining patterns over time and should only be used if there is a sequence or time element to the order data appear in a column. The run chart can make discoveries other visualization tools cannot. The run chart can identify non-random patterns over time, patterns indicating seasonal or time trends or potential information quality issues (see Figure 2.3).

Figure 2.3 A Run Chart Showing a Period of Time with a Non-Random Pattern



A histogram is a bar chart of a variable showing the frequency of values in given ranges. The histogram is useful for showing the overall statistical distribution of data. It can also be used to diagnose problems in data, such as missing values or non-typical distributions (see Figure 2.4).

Figure 2.4 A Histogram Showing an Unexpected Peak and Gap in Data



The statistical distributions that are identified in histograms are important to understand for data analysis. A distribution describes how data are spread or distributed across different values. Incorporating summary statistics, a distribution is typically characterized by a measure of central tendency, such as the mean or median, along with a measure of variability, such as the standard deviation or variance. By understanding the distribution of data, we can gain insights into the patterns and characteristics of the data, including any outliers or extreme values. Most statistical distributions

are described using a probability density function, which represents the theoretical line that would appear on top of a histogram of the data. Knowing the underlying distribution of data can be important for many analytics tasks, such as hypothesis testing, predictive modeling, and simulation modeling. By understanding the distribution of the data, more accurate predictions and more meaningful conclusions can be drawn from predictive modeling. There are numerous statistical distributions. Figures 2.5 and 2.6 show a Normal distribution and Exponential distribution by way of example.

Figure 2.5 Example of a Histogram Depicting a Normal Distribution

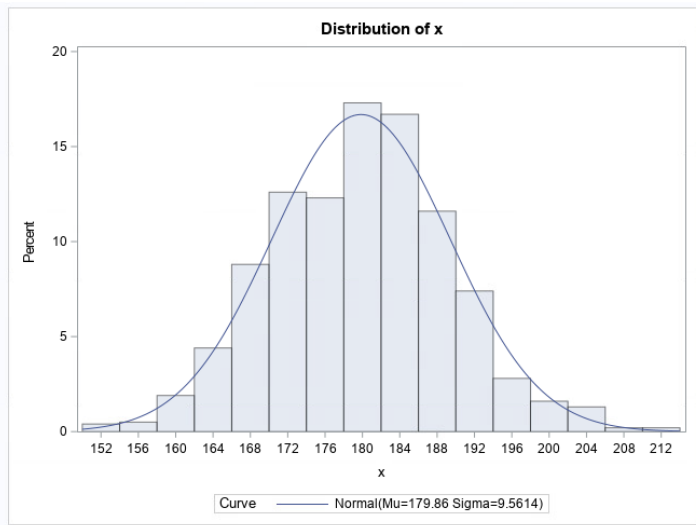
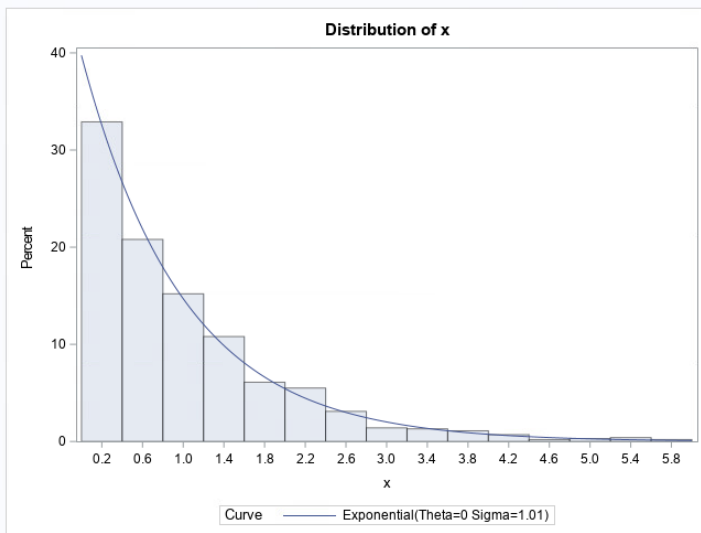
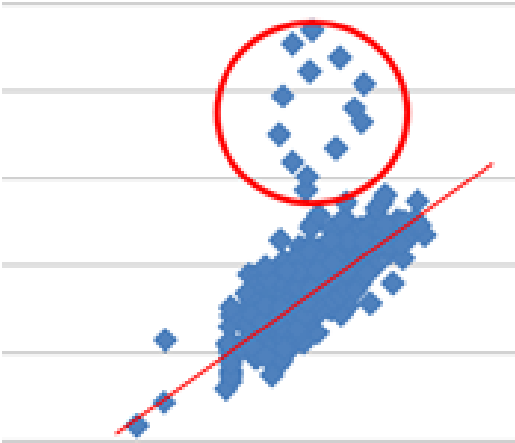


Figure 2.6 Example of a Histogram Depicting an Exponential Distribution



A scatterplot shows the relationship between two variables. The scatterplot can identify more complex relationships than correlation coefficients alone. Further, scatterplots can identify outliers in two dimensions that cannot be identified in either run charts or histograms (see Figure 2.7).

Figure 2.7 A Scatterplot Showing a Linear Relationship and Unexpected Points That Do Not Follow the Relationship



A box plot, also known as a box-and-whisker plot, is a graphical representation of the distribution of a dataset that is commonly used in EDA. It displays the median, quartiles, and range of the data. The box in the plot represents the middle 50% of the data, with the bottom and top edges of the box representing the first and third quartiles (25% and 75% quantiles of the data), respectively. The line inside the box represents the median of the data. The whiskers extend from the box to show a relative measure of the “range” of the data, typically defined as 1.5 times the interquartile range. Box plots are useful for comparing the distributions of different datasets and for identifying outliers. Note that outliers are more aggressively identified by the box plot than other techniques and may not accurately represent unexpected values. Figures 2.8 and 2.9 show a single box plot and a boxplot chart that compares groups. Figure 2.8 shows a box plot of an exponential distribution and identifies many “outliers” that are actually expected values for an exponentially distributed variable.

Figure 2.8 A Single Box Plot Showing Skewed Data

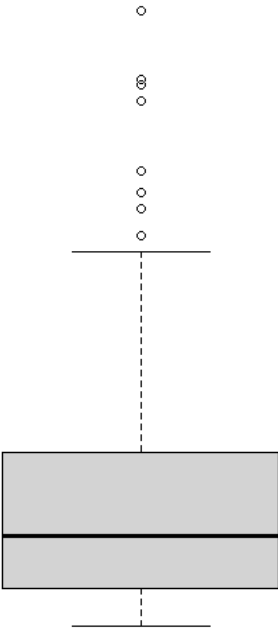
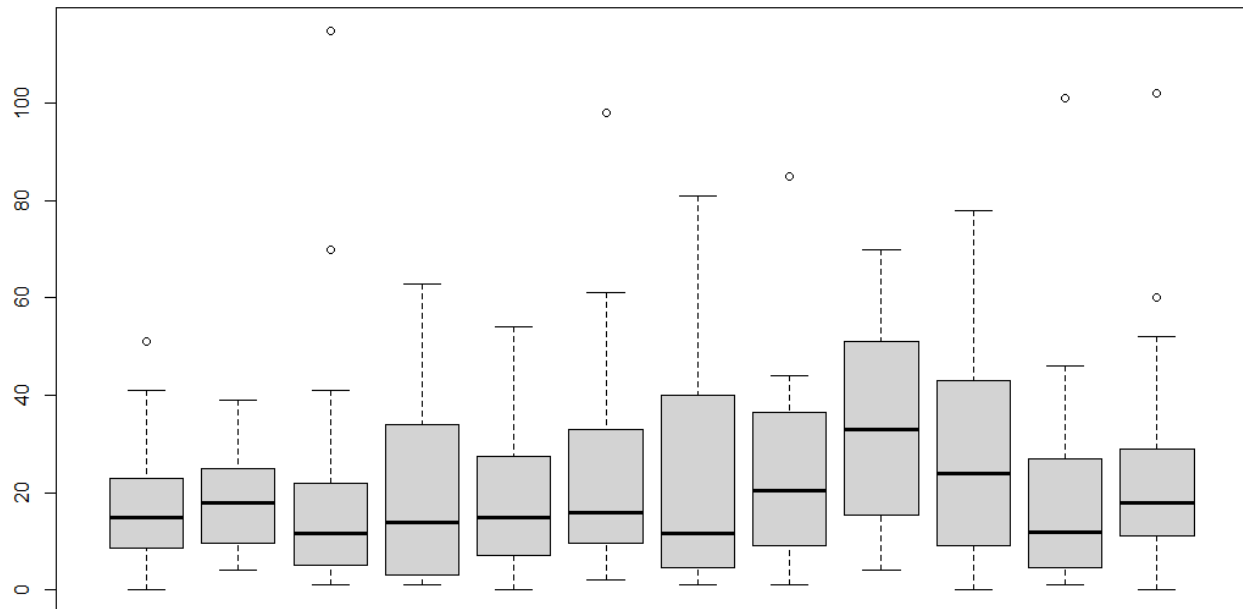


Figure 2.9 Box Plots That Compare Different Groups of Data



There are issues with using data visualizations for EDA. Unless annotated with additional statistical calculations, graphs do not show statistical significance and can often be misinterpreted as if they do. Further, for the large quantities of data that are increasingly common in data analysis, many otherwise effective visualizations can become ineffective or difficult to create due to amount of data involved. It may become necessary to sample data to effectively visualize it, but sampling potentially masks outliers and information quality issues. Using both a combination of both summary stats, visualizations, and using and understanding statistical hypothesis tests can overcome these limitations.

EDA in R

R has numerous functions and packages for exploratory data analysis. In this section, a sampling of these feature will be described. Note, many of these R commands were introduced in Module 1; here they are revisited in context of EDA.

The first set of R EDA examples use the file “project.csv”. The project data set can be loaded into R as follows

```
project <- read.csv("C:/Users/hbrown11/Desktop/project.csv", header=TRUE)
```

```
#confirm the file loaded correctly and see what the file looks like:
```

```
head(project)
```

The highlighted code will change based on the location that you saved the project csv file. Included in the project data set (Figure 2.10) are business project data with the following columns:

Event, a numerical identifier uniquely identifying each project,

emp_id, a numerical identifier uniquely identifying each employee,
days_duration, the number of days each project lasted,
project_cost, the total cost of the project

Figure 2.10 The Project Data Set

Event	emp_id	days_duration	project_cost
1	10008	13	189311
2	10007	11	287022
3	10009	6	14428
4	10003	41	122507
5	10005	15	29172
6	10004	63	45874

Summary Statistics in R

To summarize all columns in project:

```
summary(project)
```

Note this includes columns “Event” and “emp_id”, for which numerical summary statistics are nonsense as these are identifiers without numerical interpretation. To just summarize the columns of interest, each can be specified as follows:

```
summary(project$days_duration)
```

```
summary(project$project_cost)
```

To calculate the mean and standard deviation of all columns the following code can be used:

```
apply(project,2,mean)
```

```
apply(project,2,sd)
```

The code above tells R to apply the mean functions to the table project for all columns. Note in the second argument of the “apply” function: 2=columns, 1=rows. Again, this gives stats for the “id” columns of Event and emp_id, alternatively each column can be specified:

```
mean(project$days_duration)
```

```
sd(project$days_duration)
```

```
mean(project$project_cost)
```

```
sd(project$project_cost)
```

Correlations of all columns can be found by the following command:

```
cor(project)
```

Correlation of just days_duration and project_cost can be found as follows:

```
cor(project$days_duration,project$project_cost)
```

Histograms in R

Histograms may be created as follows:

```
hist(project$days_duration)
```

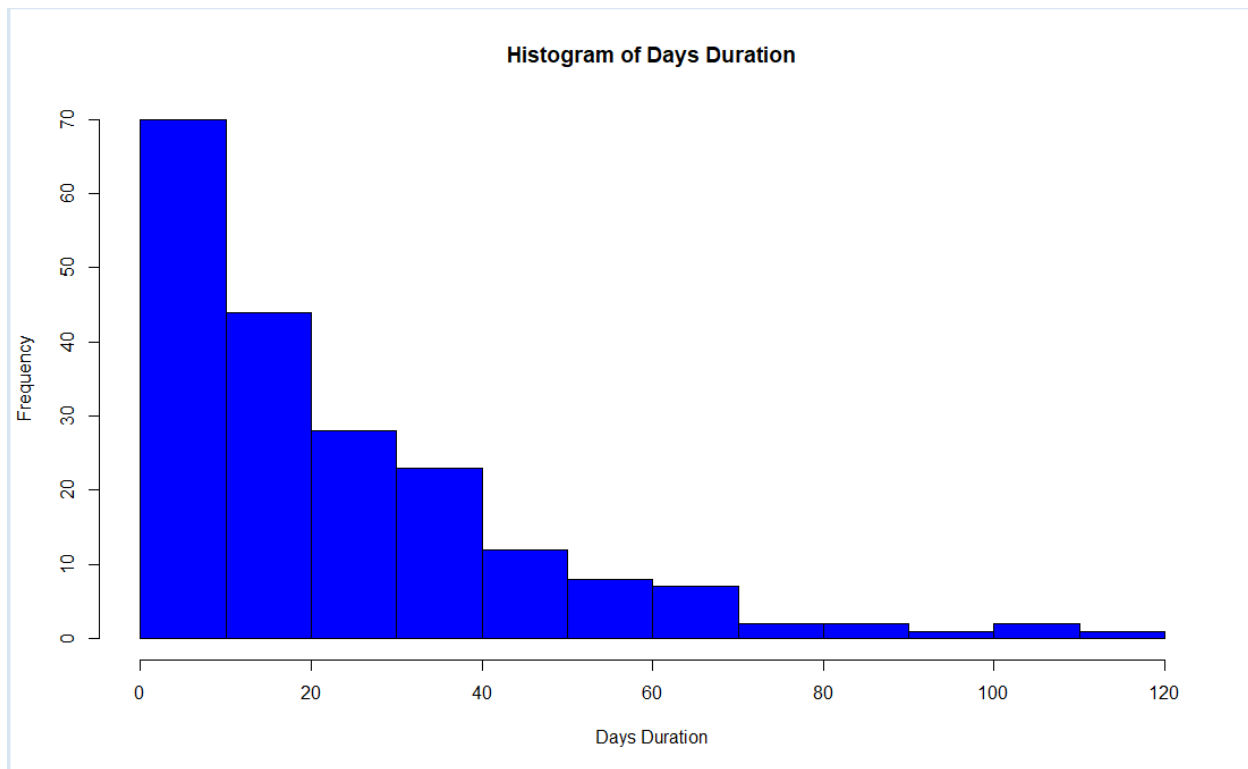
#note run the code to create a single histogram one at a time, or you will only see the last histogram

```
hist(project$project_cost)
```

There are many options to improve the appearance of data visualizations in R. (Often when performing EDA prior to starting a predictive modeling project the visualizations will not be shared and improving the appearance is not a priority). The following code adds labels and recolors the histogram, creating the histogram seen in Figure 2.11:

```
hist(project$days_duration,  
main="Histogram of Days Duration",  
xlab="Days Duration", ylab ="Frequency", col="blue")
```

Figure 2.11 Histogram of Days Duration with Labels and Color Change



Box plots in R

Single box plots may be created as follows (see Figure 2.12):

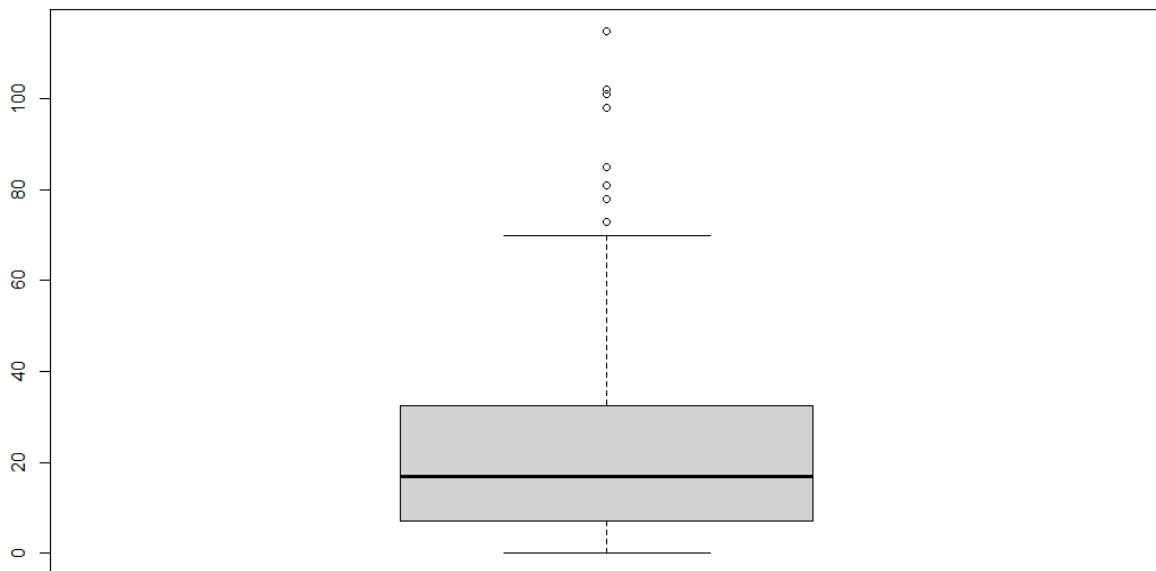
```
boxplot(project$days_duration)
```

```
boxplot(project$project_cost)
```

To create boxplots by emp_id that allows a comparison of days_duration by employee use:

```
boxplot(project$days_duration~project$emp_id)
```

Figure 2.12 Box Plot of “days_duration”



Run Charts in R

The following code creates a run chart and a line at the mean (see Figure 2.13):

```
plot(project$days_duration,type="b")
```

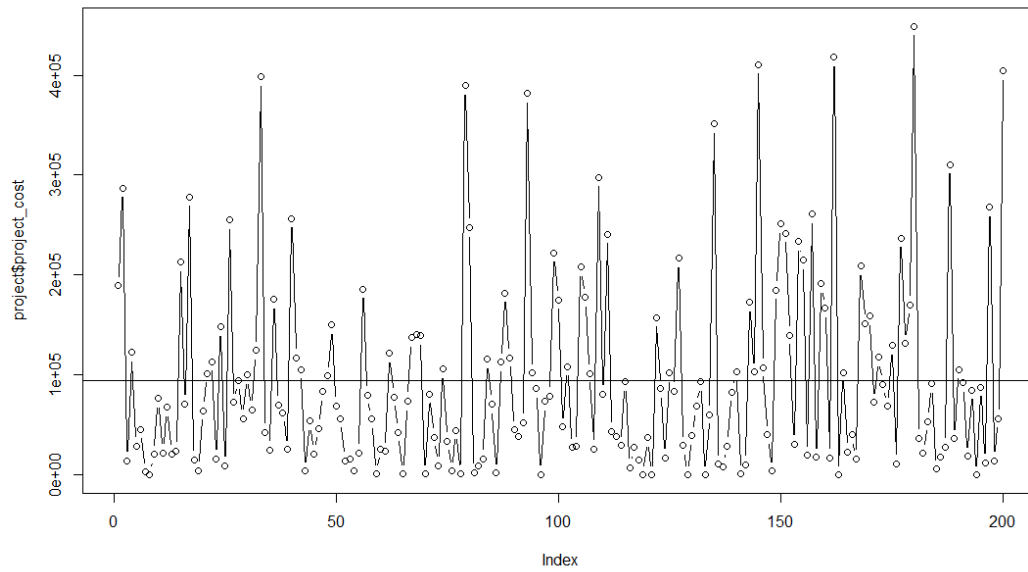
```
abline(h=mean(project$days_duration))
```

For the project cost column:

```
plot(project$project_cost,type="b")
```

```
abline(h=mean(project$project_cost))
```

Figure 2.13 Run Chart of “project_cost”

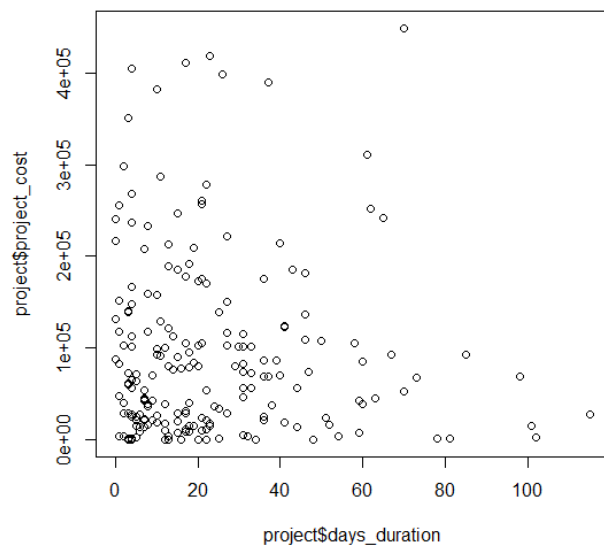


Scatterplot in R

The following code creates a scatterplot (see Figure 2.14).

```
plot(project$days_duration,project$project_cost)
```

Figure 2.14 Scatter Plot of days_duration and project_cost



Full R Code for EDA of “project” Data Set:

```
project <-read.csv("C:/Users/hbrown11/Desktop/project.csv",header=TRUE)
head(project)
summary(project$days_duration)
summary(project$project_cost)
mean(project$days_duration)
sd(project$days_duration)
mean(project$project_cost)
sd(project$project_cost)
cor(project$days_duration,project$project_cost)
hist(project$days_duration)
hist(project$project_cost)
boxplot(project$days_duration)
boxplot(project$project_cost)
plot(project$days_duration,type="b")
abline(h=mean(project$days_duration))
plot(project$project_cost,type="b")
abline(h=mean(project$project_cost))
plot(project$days_duration,project$project_cost)
```

Conclusions from the EDA of the “project” Data Set

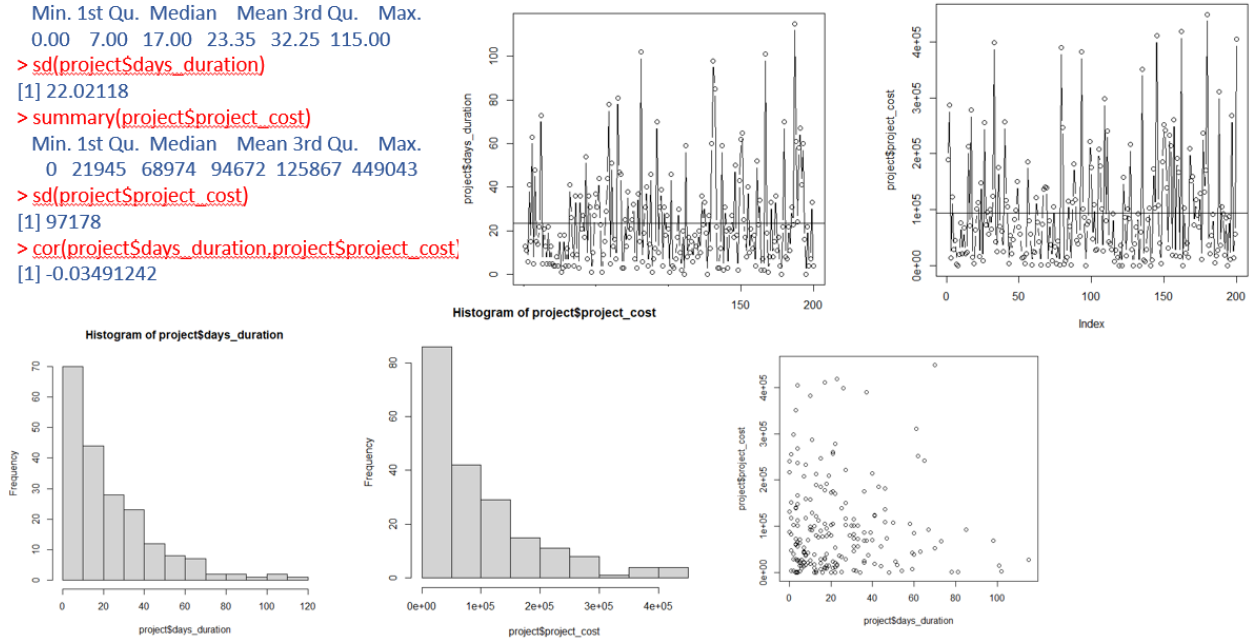
After running the above code in R, the following conclusions can be made:

- Both Days_duration and project_cost appear approximately exponentially distributed (from the histogram)
- No real outliers or datapoints of concern in tdriv.project (from all plots and summaries, considering the data are exponentially distributed)
- Days_duration and project_cost are not correlated and do not seem to be related (from the correlation and scatterplot)

See Figure 2.15 for the output that led to these conclusions.

Figure 2.15 Selected Output from the EDA of the “project” Data Set

```
> summary(project$days_duration)
Min. 1st Qu. Median Mean 3rd Qu. Max.
0.00 7.00 17.00 23.35 32.25 115.00
> sd(project$days_duration)
[1] 22.02118
> summary(project$project_cost)
Min. 1st Qu. Median Mean 3rd Qu. Max.
0 21945 68974 94672 125867 449043
> sd(project$project_cost)
[1] 97178
> cor(project$days_duration,project$project_cost)
[1] -0.03491242
```



Drawing Conclusions from EDA

With a basic understanding of descriptive statistics and data visualization, understanding the nature of the data by looking at graphs and summary statistics is straight forward. For the project data, we can conclude the days_duration and project_cost columns are exponentially distributed, which is not unexpected given the nature of such business data (long expensive projects are less likely to happen, but do occur). Also, interestingly, the length of the project and cost of the project are not correlated.

Are there potential information quality issues in the project data set? Identifying outliers is one indication of data needing investigation. An outlier is not a firmly defined concept, one general rule of thumb is any point falling outside of three standard deviations from the mean (however, this can happen for large sample sizes without being an outlier, or for skewed distributions). The interquartile range (the difference between the 75th percentile and 25th percentile) multiplied by 1.5, as is used in the box and whiskers plot, is also often used—note that this is even more likely to find false outliers than plus or minus three standard deviations. Understanding the underlying data and its context is key to correctly identifying outliers. In addition, graphs (with experience), when possible, are generally better for identifying what does not look typical and should be further investigated. Practice is key to recognizing what doesn't look correct in a set of data. From an experienced data analyst's view, there are no outliers or potential information quality issues in the project data set we just examined.

There are several potential pitfalls involved with trying to identify potential information quality issues. Those new to EDA will often over-identify patterns or points in data as concerning when, in fact, what is observed is expected variation in the data. Depending on where you work and the kinds of data you receive professionally, you may be much more likely to receive data with no issues—dozens of data sets

may be analyzed before an issue is found. In other analyst situations, data may need to be routinely “cleaned” before every analysis (variables recoded, errors addressed, missing values dealt with, etc.). Finally, if questionable data are identified, data points should never be removed from data without an investigation that leads to justifiable reason for removal (such as an identified data entry error).

Example 2 EDA

The csv file dataforEDA contains four simulated columns of data (W, X, Y, and Z) and an id column. Without context to what processes these data represent, we will see what conclusion we can make purely from EDA.

Figure 2.16 Contents of the File “dataforEDA”

```
id      W      X      Y      Z
1  17.4  152.6  59.98  105.6
2  16.1  179.8  52.55  127.7
3  18.3  195.9  46.74  141.0
4  14.9  179.3  54.56  142.9
5  32.4  158.2  41.53   93.2
6  10.4  166.2  47.81  120.1
```

R Code for EDA of “dataforEDA”

```
EDA <-read.csv("C:/Users/hbrown11/Desktop/dataForEDA.csv",header=TRUE)
```

```
#don't forget to change the path to match your file path
```

```
head(EDA)
```

```
summary(EDA)
```

```
mean(EDA$W)
```

```
sd(EDA$W)
```

```
mean(EDA$X)
```

```
sd(EDA$X)
```

```
mean(EDA$Y)
```

```
sd(EDA$Y)
```

```
mean(EDA$Z)
```

```
sd(EDA$Z)
```

```
cor(EDA)
```

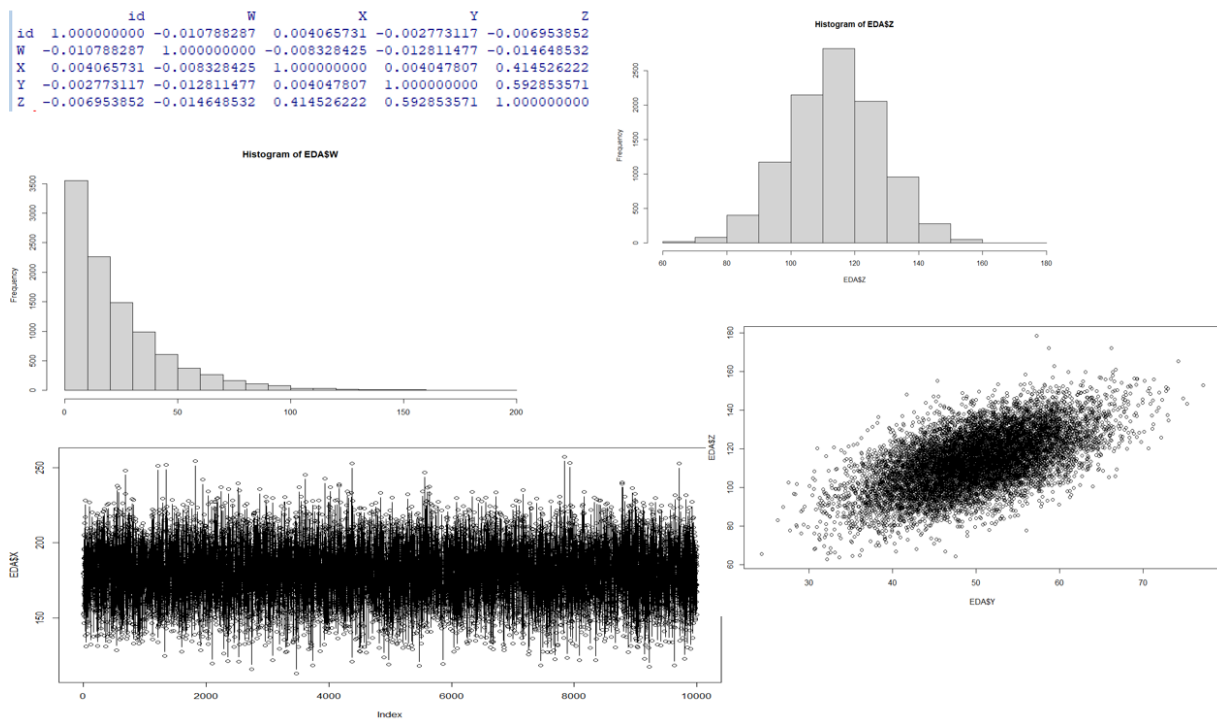
```
hist(EDA$W)
```

```

hist(EDA$X)
hist(EDA$Y)
hist(EDA$Z)
plot(EDA$W,type="b")
abline(h=mean(EDA$W))
plot(EDA$X,type="b")
plot(EDA$Y,type="b")
plot(EDA$Z,type="b")
plot(EDA$W,EDA$X)
plot(EDA$W,EDA$Y)
plot(EDA$W,EDA$Z)
plot(EDA$X,EDA$Y)
plot(EDA$X,EDA$Z)
plot(EDA$Y,EDA$Z)

```

Figure 2.17 Selected Output from the EDA of the “dataforEDA” Data Set



Conclusions EDA of “dataforEDA”

After running the R code and reviewing the output (see Figure 2.17), the following conclusions can be made:

- Z and Y are fairly highly correlated; X and Z somewhat so; no other variables appear correlated
- W is approximately exponentially distributed (from the histogram); X, Y, and Z are normally distributed
- No real outliers or datapoints of concern in dataForEDA (from all plots and summaries)

Exercise Practice EDA

The csv file “retail” contains daily sales values for four items at a particular retail location. Included in the retail data set (Figure 2.18) are the following columns:

Day, sequential identifies the day of sales recorded for this time period (Day =1 is the first recorded day of says, Day=2 is the second, etc.)

The columns Coke2Liter, Advil30CntGel, BountyPaperTowel2pk, BountyPaperTowel2pk, and IvorySoap6pk are daily sales values for the given product.

Figure 2.18 Contents of the File “retail”

Day	Coke2Liter	Advil30CntGel	BountyPaperTowel2pk	IvorySoap6pk
1	32.11	14.58	44.70	40.46
2	33.58	14.83	45.95	36.85
3	29.10	12.25	49.20	45.01
4	33.33	20.60	67.26	43.67
5	23.56	16.69	37.62	53.98
6	31.66	16.66	60.44	52.54

Perform EDA on the retail file on your own (solution will follow) by doing the following:

1. Calculate summary stats: max, min, median, mean, standard deviation, correlation
2. Create Histograms
3. Create Run Charts
4. Create Scatterplots
5. Provide a few sentences that explain what you discovered

Solution to Practice EDA Exercise

R Commands:

```
retail<-read.csv("C:/Users/hbrown11/Desktop/retail EDA.csv",header=TRUE)

head(retail)

summary(retail)

mean(retail$Coke2Liter)

sd(retail$Coke2Liter)

mean(retail$Advil30CntGel)

sd(retail$Advil30CntGel)

mean(retail$BountyPaperTowel2pk)

sd(retail$BountyPaperTowel2pk)

mean(retail$IvorySoap6pk)

sd(retail$IvorySoap6pk)

cor(retail)

hist(retail$Coke2Liter)

hist(retail$Advil30CntGel)

hist(retail$BountyPaperTowel2pk)

hist(retail$IvorySoap6pk)

plot(retail$Coke2Liter,type="b")

plot(retail$Advil30CntGel,type="b")

plot(retail$BountyPaperTowel2pk,type="b")

plot(retail$IvorySoap6pk,type="b")

plot(retail$Coke2Liter,retail$Advil30CntGel)

plot(retail$Coke2Liter,retail$BountyPaperTowel2pk)

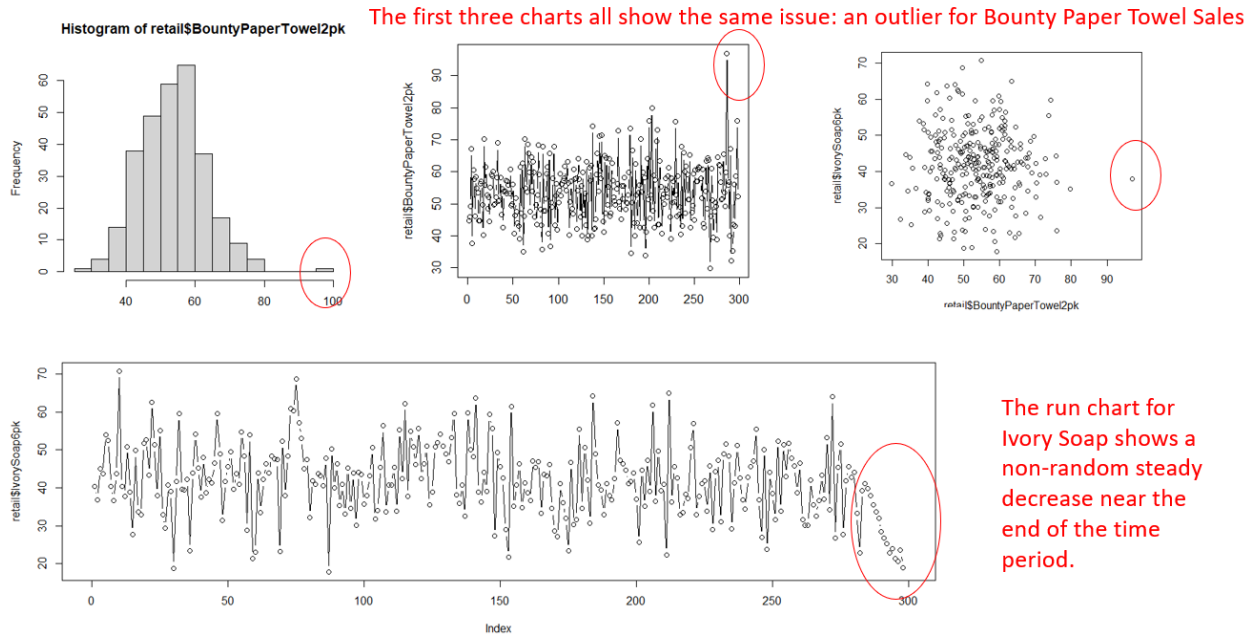
plot(retail$Coke2Liter,retail$IvorySoap6pk)

plot(retail$Advil30CntGel,retail$BountyPaperTowel2pk)

plot(retail$Advil30CntGel,retail$IvorySoap6pk)

plot(retail$BountyPaperTowel2pk,retail$IvorySoap6pk)
```

Figure 2.19 Selected Output from the EDA of the “retail” Data Set



Conclusions EDA of “retail” Data Set

After running the R code and reviewing the output (see Figure 2.19), the following conclusions can be made:

- Bounty Towel has an outlier of 97.04 that should be investigated (to make sure there is not a data entry error; or in the case of better than expected sales, find out what was going on that day...a sale or other promotion? etc.)
- Ivory Soap had an unusual pattern near the end of time period found on the run chart. This also needs investigation (was it a data entry error, was there a stocking or facing issue in the retail store? etc.)
- Otherwise the data were pretty unremarkable, no strong relationships, no other issues

Exploring Categorical Data

Categorical data (or discrete data) are data with columns that contain nonnumeric data such as text data. Categorical data cannot be explored in the same manner as continuous data. Tables and bar plots are quick ways to explore categorical data.

The csv file “SeniorSurveyQuestion” contains results from the following question asked to senior business students in a survey: “Rate your level of satisfaction with your attainment of skills and knowledge that transfer to the workplace”. Included in the data set (Figure 2.20) are only two columns, an “ID” column uniquely identifying the student answering the question and the “Attainment” column with the student response (possible values include “Dissatisfied”, “SomewhatDissatisfied”, “NoOpinion”, “SomewhatSatisfied”, and “Satisfied”).

Figure 2.20 Contents of the File “SeniorSurveyQuestion”

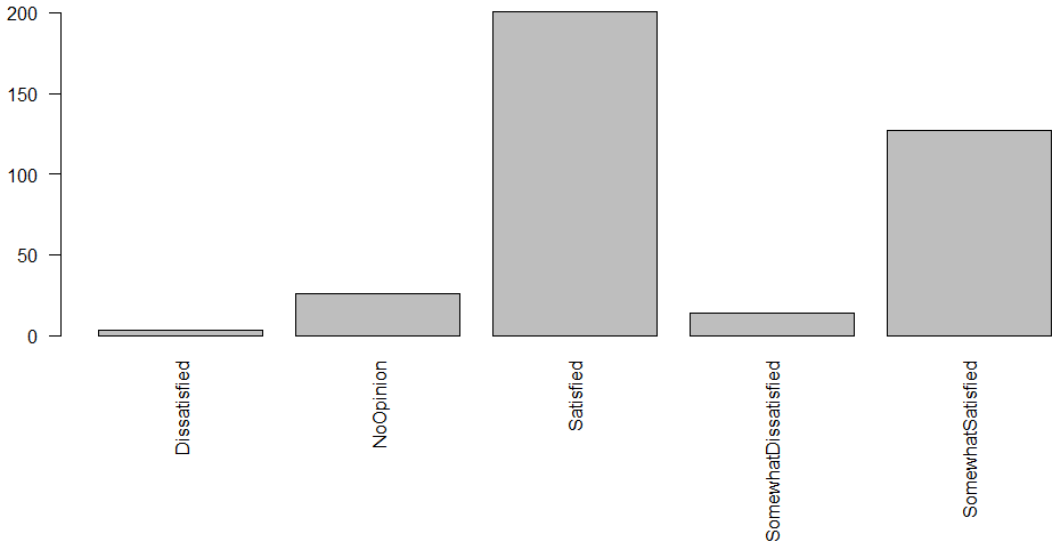
```
ID      Attainment
2 SomewhatSatisfied
3      NoOpinion
4 SomewhatSatisfied
5 SomewhatSatisfied
6      Satisfied
7 SomewhatSatisfied
```

R Code:

```
sr<-read.csv("C:/Users/hbrown11/Desktop/SeniorSurveyQuestion.csv",header=TRUE)
head(sr)
table(sr$Attainment)
t=table(sr$Attainment)
barplot(t)
barplot(t,las=2)
par(mar = c(10, 5, 10, 5))
barplot(t,las=2)
```

Note, the last three lines optionally improve the plot a little. One label is crowded out in the original plot, so the las=2 option, places the labels vertically to fit them all in. Because the names are longer, to see the entire response we need to change the default margins to display the full words with the par(mar=c(10,5,10,5)) statement. In the four numbers, the first number is the bottom margin, the second number is the left margin, third top, fourth right). Figure 2.21 displays the resulting bar chart.

Figure 2.21 Bar Chart of Attainment from "SeniorSurveyQuestion"



Module 2 Assignment

The data set AssignmentdataforEDA.csv are data from a food industry experiment that is seeking to use a UV instrument to predict log₁₀ APC levels in raw meat food samples. The UV instrument is a new method of ultraviolet bacteria detection; log₁₀ APC refers to a log₁₀ transformation of aerobic plate count (a type of bacteria). Raw meat product must be sampled to make sure bacteria levels are below what would indicate spoilage or potential safety concerns. Traditional bacteria testing techniques are destructive to the product and requires a plating and incubation period of approximately 48 hours (a long time when trying to ship food products). The UV instrument attempts to instantly estimate the amount of bacteria on the raw meat. However, a relationship between the UV reading and the tried and trusted log₁₀ APC results must be established via a regression model. Building a good model would potentially speed up and improve the process of determining if there are any food safety issues in raw food. We will build a model for these data later; now we just want to perform exploratory data analysis.

For the data set, "AssignmentdataforEDA.csv", perform the following exploratory data analysis in R:

1. Calculate a median, min, max, mean, standard deviation for each column
2. Calculate correlation
3. Create a run chart and histogram for each column
4. Create a scatterplot
5. In a short paragraph, note anything in the data that seems unusual (outliers, etc.)

Couple of reminders: make sure you include your R commands (R code or program) that you used to generate your graphs and output, make sure you use the correct table.

Turn the assignment in as a single word document (or something equivalent) with your program, any pertinent output or plots you want to include, and your brief explanation of anything you found.

Module 3: Building Predictive Models with Regression and Multiple Regression

Regression for Predictive Modeling

In this module, we finally get to the subject of this course: building predictive models. Note, as we begin predictive modeling and leave EDA behind, we will no longer perform EDA before analyzing data. However, EDA should routinely be done for “real life” data analysis projects. The EDA portion of our predictive analysis examples is left out to focus on the models we are learning in each module.

Regression is a logical first analytics modeling technique on which to begin a course on predictive modeling. It is very likely that you have had exposure to regression numerous times before reaching this course. However, as we will discuss and demonstrate in this module, building a regression model for professional use at a business may be a significantly different experience from how regression is taught in an introduction to statistics course.

Why use Regression for Predictive Modeling?

Regression models offer several advantages. They are well-developed theoretically, with ample documentation and numerous software options available. This ensures that data analysts have access to resources to support their analysis process. In addition, regression models generally enhance the understanding of a problem by providing insights into the relationships between variables. By examining the coefficients and significance levels, one can gain valuable insights into the factors that influence the outcome of interest. Further, when assumptions and inferences are correctly made, regression models allow for the precise quantification of uncertainty through the use of confidence intervals. This provides a measure of the reliability and accuracy of the model's predictions. Moreover, regression models are flexible and can be applied to a wide range of problems across various domains. They offer procedures for variable selection, allowing for the identification of the most relevant features that contribute to the outcome. Last but not least, regression models are often perceived as simpler than other modeling methods such as Artificial Neural Networks (ANN) for similar predictive modeling tasks. This simplicity can make them more accessible to businesses, especially when interpretability and transparency are needed. In summary, regression models provide a robust and versatile approach to understanding and predicting relationships in data.

Disadvantages of Regression Models

Despite their advantages, regression models also come with several disadvantages. As is the case with many analytics techniques, they are easy to use incorrectly. Without a proper understanding of the underlying theory and assumptions, analysts may make serious mistakes in model specification, interpretation, and inference. Regression models have traditionally been developed through careful iterative processes by the analyst, which can be time-consuming and labor-intensive. Another issue is the potential for overfitting. Regression models can be prone to capturing noise and idiosyncrasies in the data, leading to an overly complex model that fails to generalize well to new observations. Moreover, while regression models offer flexibility, they are not suitable for every situation. Different problems may require alternative modeling techniques that better capture nonlinear relationships or handle high-dimensional data. Regression models may be outperformed by other analytics techniques, such as machine learning algorithms or deep learning models, especially in scenarios where complex

patterns and interactions exist within the data. While regression analysis is still widely used and a good choice for many predictive modeling problems, it is necessary to be aware of its limitations and consider alternative approaches when warranted.

Origins of Regression Analysis

The history of regression analysis involves early contributions by Carl Friedrich Gauss. During his career, Gauss worked at an observatory, where he made notable contributions to astronomy and celestial mechanics. In 1801 astronomers discovered Ceres, a dwarf planet located between Mars and Jupiter. However, due to its relatively short observation period, Ceres was subsequently lost and its exact orbit became uncertain. In an effort to locate Ceres again, Gauss employed his method of least squares linear regression. By analyzing the available data points and minimizing the sum of squared errors, Gauss was able to accurately estimate the orbital parameters of Ceres and predict its location. Thus, in 1809 regression analysis was demonstrated as an effective predictive technique. Gauss's work on regression and the method of least squares laid the foundation for modern regression analysis, which has since become a fundamental tool in statistics and data analysis. Adrien Marie Legendre made similar discoveries at about the same time as Gauss. Sir Walter Galton would more thoroughly add contributions, followed by Karl Pearson and others. (For more details on the history of regression see Fahrmeir et al., 2022, among other sources.)

The Many Forms of Regression

The term “regression” refers to numerous modeling techniques that take the basic ideas of Gauss and incorporate them in various types of models. Each form of regression is tailored to different scenarios and data types. The following is an overview of some commonly used types of regression:

- **Linear Regression:** Linear regression is the most basic and widely used form of regression. In mathematics it is described as ordinary least squares linear regression with one predictor. It models the relationship between a dependent variable and a single independent variable using a linear equation. The goal is to find the best-fitting line that minimizes the sum of squared residuals. We will start with this form of regression.
- **Multiple Regression:** Multiple regression extends linear regression to include multiple independent variables. It allows for the analysis of the simultaneous effect of several predictors on the dependent variable. The model estimates the coefficients of each independent variable while considering their relationships with the outcome variable. After completely linear regression, we will cover multiple regression.
- **Polynomial Regression:** Polynomial regression involves fitting a polynomial equation to the data instead of a straight line. It is useful when the relationship between the independent and dependent variables appears to be nonlinear. Polynomial regression can capture curvilinear patterns by including higher-order polynomial terms. More generally, transformations of variables allow various nonlinear relationships to be captured by regression.
- **Logistic Regression:** Logistic regression is employed when the dependent variable is binary or categorical. It models the probability of an event occurring by fitting a logistic function to the data. Logistic regression is commonly used for classification tasks and can handle both binary and multi-class problems.

- **Ridge Regression:** Ridge regression is a regularized form of linear regression that addresses the issue of multicollinearity (high correlation among predictors). It adds a penalty term to the loss function, which shrinks the regression coefficients, thereby reducing their variability.
- **LASSO Regression:** LASSO regression, like ridge regression, is a regularized regression technique. It not only addresses multicollinearity but also performs variable selection by forcing some coefficients to become exactly zero. LASSO regression can be useful when dealing with datasets with a large number of predictors.
- **Time Series Regression:** Time series regression is used when the data exhibits temporal dependence. It models the relationship between the dependent variable and one or more lagged values of itself or other relevant time series variables. Time series regression is commonly employed in forecasting and trend analysis.
- **Spatial Regression:** Spatial regression or geographically weighted regression is used when the data exhibits location dependence. It models the relationship between the dependent variable and one or more spatially lagged values of itself or other relevant geographical variables. Spatial regression is commonly used in regional economics, environmental studies, geology, mining, and epidemiology.

These are just a few examples of regression models, and there are many other specialized variations of regression. The selection of regression modeling technique depends on the nature of the data, the predictive modeling question, and the underlying assumptions of the analysis.

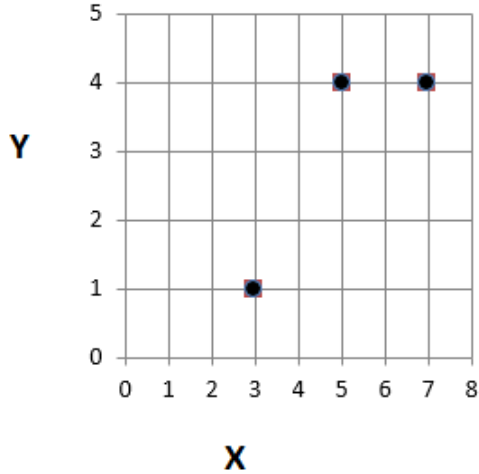
How does Linear Regression Work?

For the purposes of giving a conceptual understanding of how regression works an overly simple example is presented. Be aware, this is an unrealistically small amount of data, regression should never be done based on only three observations (roughly anything less than 30 observations is suspect for statistical purposes; building a regression model for professional purposes, analysts will want much more data than 30 observations). This first example is a demonstration of the calculations of regression where the arithmetic can be kept simple.

Develop a model to predict Y given X using linear regression on the following table (see Figure 3.1 for a plot of these points):

	X	Y
1	3	1
2	5	4
3	7	4

Figure 3.1 A Plot of X and Y in a First Regression Example



Linear regression estimates a slope, β_1 , and an intercept, β_0 to fit the optimal equation to the data:

$$y = \beta_0 + \beta_1 x + \varepsilon$$

Note, ε refers to the error or residuals of the model. The formulas to determine the slope and intercept for linear regression are given as follows:

$$\text{slope} = \beta_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

$$\text{intercept} = \beta_0 = \bar{y} - \beta_1 \bar{x}$$

For our simple example:

$$\text{Mean of } x \text{ (i.e. } \bar{x}) = 5$$

$$\text{Mean of } y \text{ (i.e. } \bar{y}) = 3$$

$$\beta_1 = \frac{(3-5)*(1-3)+(5-5)*(4-3)+(7-5)*(4-3)}{(3-5)^2+(5-5)^2+(7-5)^2} = 6/8 = 0.75$$

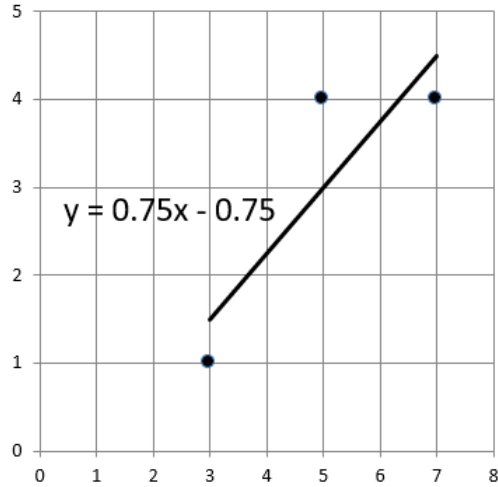
$$\beta_0 = 3 + 0.75 * 5 = -0.75$$

The final regression model can be seen in Figure 3.2. The model can be used to take values of X and predict values of Y. For example, if $x = 4$, the regression line would predict 2.25 as the value of y :

$$y = 0.75 * 4 - 0.75$$

$$y = 2.25$$

Figure 3.2 A Simple Regression Model



Now that we have looked at the basics of the mathematics behind building a regression model, we will look at the process of how to use regression correctly to build a predictive model that could be used by businesses.

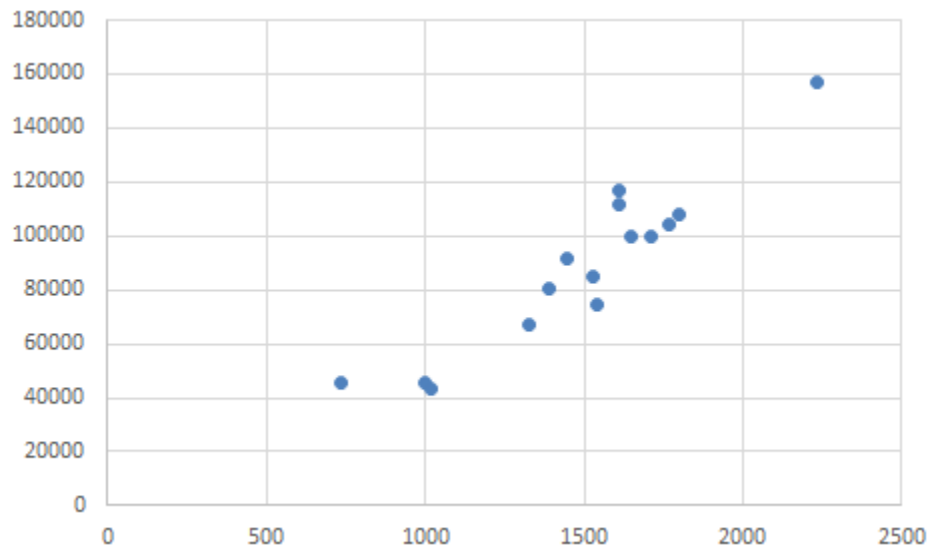
A More Realistic Regression Problem

A real estate agency wants to create a system to help agents estimate house sale prices based on square footage. The table (Figure 3.3) and graph (Figure 3.4) below detail the first 15 data points in a table of recent house sales:

Figure 3.3 The First 15 Rows of a Real Estate Data Set

	Square Footage	Sale Price
1	1326	66960
2	1391	80400
3	1000	45600
4	1542	74400
5	735	45360
6	1444	91920
7	1796	108000
8	1770	104400
9	1708	99600
10	1529	85200
11	2234	157200
12	1607	116520
13	1648	99600
14	1608	111600
15	1020	43200

Figure 3.4 A Scatterplot of Points from the Real Estate Data Set



A linear regression model can be used in this case, because the relationship between square-footage and house price, especially for houses in a particular area, is a linear relationship, i.e. as square footage increases, house price increases. In this case, the dependent variable or response variable (the output of our model, identified as y in our regression formulas, and sometimes called the “target variable”) is house sale price. The independent variable or predictor variable (the input of our model, identified as x in our regression formulas) is square footage. Before completing this example in R, the process of building a predictive regression model will be briefly discussed.

Hypothesis Testing Versus Predictive Modeling

In a first class on statistics, regression is often introduced primarily as a method of hypothesis testing. In regression for hypothesis testing, all of the available data are used to test if the slope is significantly different from zero:

H_0 : slope = 0

H_A : slope \neq 0

In our example, this would be a test to see if square footage significantly impacts the a house sale price (obviously this is a well-known fact and does not need to be tested). If the absolute value of test statistic is large enough (i.e. the probability of the data belonging to the null hypothesis small enough), we reject the null hypothesis and conclude square footage does impact home sale price.

In predictive modeling, we might check the hypothesis test (it should be significant for a good model to exist), but emphasis is on the ability of our model to accurately make predictions for data that was NOT used to build the model. For this reason, we divide our data available for building a model into training and testing data sets. The training data are used to build the model; the testing data are withheld and

only used to test the model. This provides a more realistic measure of how well our model will actually work. Also, generally, we need more data for building predictive models than hypothesis tests (more data improves predictive accuracy). If we only evaluate the model with the data that were used to train the model, the evaluation will be biased since the modeling technique used attempted to optimally fit to the training data (and may have overfit the data).

Dividing your data into a training and testing data sets and trying out your model on the test data is a minimum expectation for all predictive modeling problems (regardless of the technique used, regression, Random Forests, artificial neural networks, etc.). The idea is you only use a portion of the available data to build or “train” your model (or other data mining technique). The other portion of the data is withheld and then used to test the model to provide a non-biased view of how the model performs. The test data set is also sometimes called a “validation” data set. It is also common for multiple training and test data sets to be used when large amounts of data are available.

Generally, we want to divide the data randomly. This can be done by using pseudo-random number generation and then sorting the data based on the random numbers. How we divide the data into testing and training sets can vary, typically you want more data in the training data set than the test data set. A common practice is to use 50 to 80% of the data for building the model and 50 to 20% of the data for training.

Cross-Validation with Folding

An innovative way of dividing data into training and test data that is now commonly used for model building is “cross-validation with folding” or k-fold cross-validation. Using this method, we create multiple training data and test data sets from the data available to build a model (with clever replacement to get the most out of our data). The training data are still used to build the model and the test data are still used to evaluate the model’s performance. In cross-validation, the test data are called “out of sample” data, and the error calculated on the test data will be called “out of sample error” or “OOS error”. The number of “folds” must be selected, typical values for the number of folds are between 5 and 10, as these are believed to balance between bias and variance (see Nti et al., 2021)

Figures 3.5 to Figure 3.8 detail an example of how 5-fold cross-validation with folds works on an unrealistically small data set (so that we can see the details). The data are approximately randomly and evenly divided into 5 groups (or folds). The regression model will be fit 5 times, the first time, folds 1-4 will be used to build the regression model to predict saleprice from sqft and fold 5 will be used as the test data (OOS error will be calculated for the test data). The second time, folds 1,2,3, and 5 will be used to build the regression model and fold 4 will be used as the test data (OOS error again only calculated from fold 4). This process is repeated until each fold has been used as the test data once. The OOS error is then averaged to provide an estimate of how the model will make predictions on data not used to build the model. The regression model parameters can then be estimated with the full set of data.

Figure 3.5 Step 1 of Cross-Validation with 5 Folds:

Data available for regression:

sqft	saleprice
1326	66960
1391	80400
1000	45600
1542	74400
735	45360
1444	91920
1796	108000
1770	104400
1708	99600
1529	85200

Start by dividing the data into 5 folds (this should be done randomly, so in our example we will assume the 10 rows of data have been randomly sorted)

	sqft	saleprice
1.	1326	66960
	1391	80400
2.	1000	45600
	1542	74400
3.	735	45360
	1444	91920
4.	1796	108000
	1770	104400
5.	1708	99600
	1529	85200

Figure 3.6 Step 2 of Cross-Validation with 5 Folds:

	sqft	saleprice
1.	1326	66960
	1391	80400
2.	1000	45600
	1542	74400
3.	735	45360
	1444	91920
4.	1796	108000
	1770	104400
5.	1708	99600
	1529	85200

Use as training data

Use as test data

The regression model will be fit 5 times, the first time, folds 1-4 will be used to build the regression model to predict saleprice from sqft and fold 5 will be used as the test data (OOS error will be calculated for the test data)

Figure 3.7 Step 3 of Cross-Validation with 5 Folds

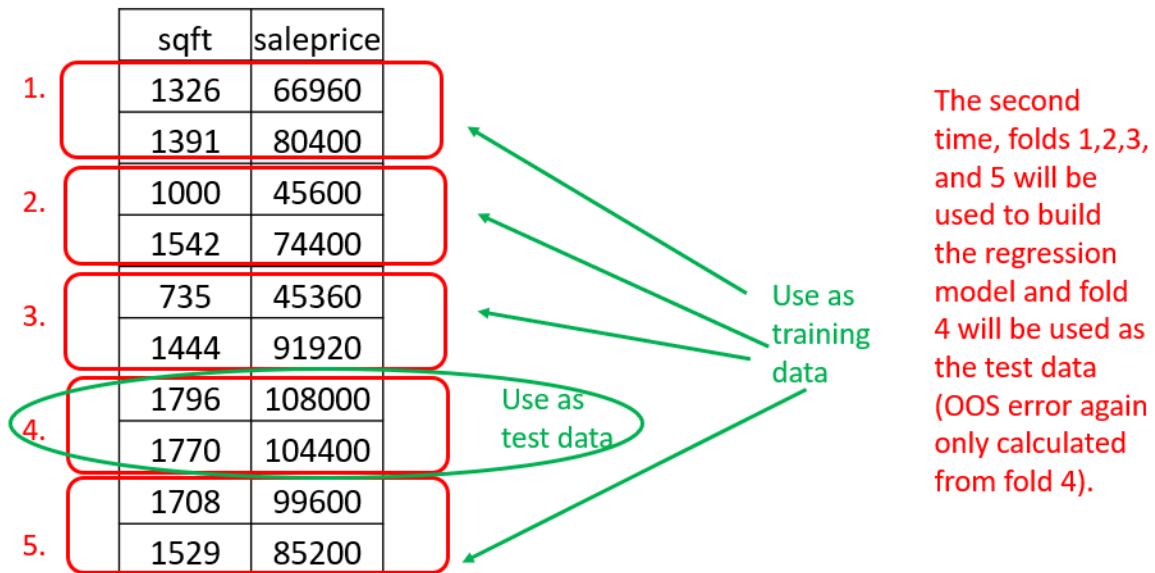
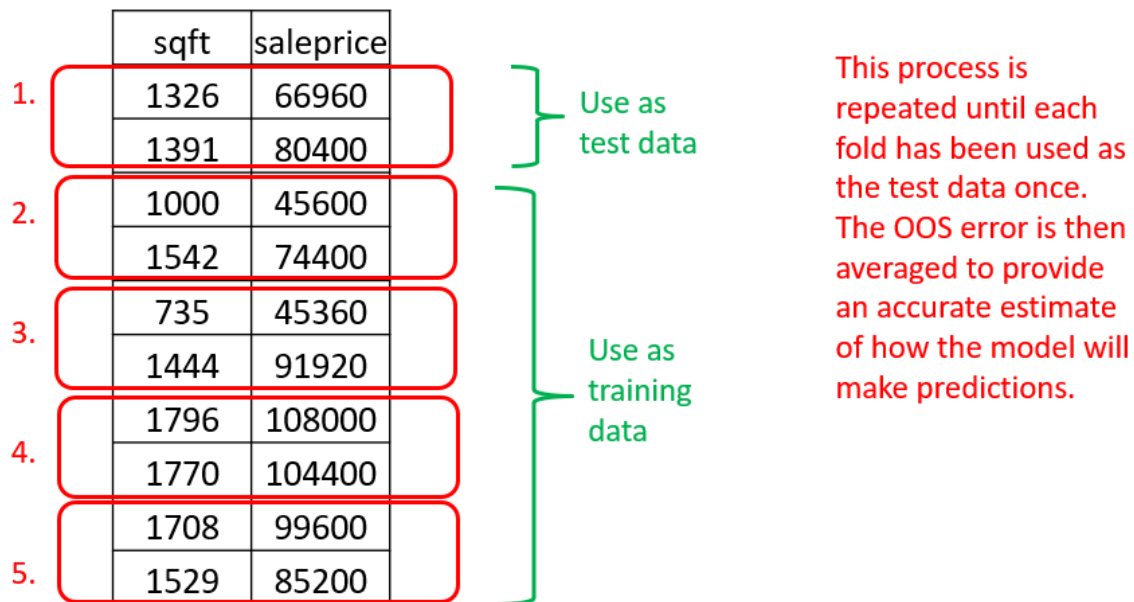


Figure 3.8 Final Steps of Cross-Validation with 5 Folds



Fortunately, the messy details of estimating parameters and dividing data into training and test data sets for using k-fold cross-validation are handled by R. In the next section, we will analyze the real estate data using R.

Regression with R

R is a very good analytics option for building predictive regression models. More than one package is available and can be used to build accurate linear regression models. To demonstrate one way in which R can be used to build a predictive regression model using k-fold cross-validation, we will look at the real-estate example.

The csv file housing contains real estate data from a particular area of a city (see Figure 3.9). It includes the columns “sqft”, the house square footage, and “saleprice”, the sale price of the house. The goal of our regression analysis will be to build a linear model using sqft to predict saleprice. Additionally, we will use cross-validation to accurately determine the expected error when using the model.

Figure 3.9 Contents of the File “housing.csv”

```
sqft saleprice
1326    66960
1391    80400
1000    45600
1542    74400
 735    45360
1444    91920
```

The R code for Building a Linear Regression Model with 5-fold Cross-Validation is as Follows:

```
house <-read.csv("C:/Users/hbrown11/Desktop/housing.csv", header=TRUE)
#reminder the file path will need to be changed to match where you saved the housing file
head(house)
install.packages("caret") #a reminder this only needs to be installed once
library(caret)
#Note we set a seed for the random number generation so that the random sampling will be
repeatable
set.seed(42)
# Set up 5-fold cross-validation
train_control <- trainControl(method = "cv",number = 5)
# fit the model
model <- train(saleprice ~sqft, data = house, method = "lm", trControl = train_control)
# view the model output
summary(model)
model
```

The `summary(model)` statement produces the output in Figure 2.21, which has been annotated. From this output, we have information about residuals, the model's parameter estimates, and hypothesis tests. In a traditional statistical experiment where regression is used to test a hypothesis, the t-tests and F-test play the key role. However, in predictive modeling, while we generally want to see a significant F-test/significant slope, we place more emphasis on evaluating the model with the test data set. From the summary, we can get the parameter estimates used to create the linear model:

house price estimate = $-31246.873 + 78.812 \cdot (\text{square footage})$

To use the model, for example for a 1200 square-foot house, we could estimate a sale price as follows:

House price estimate = $-31246.873 + 78.812 \cdot (1200)$

House price estimate = \$63,327.53

It is important to note that we should only make predictions for values of square-footage that were within the range of square-footages that were used to build the model (you can use `summary(house)` to find the max and min). Making a prediction for a value outside the range used to build the model is extrapolation and will give untested and unreliable results (for example, plugging 300 in for sqft in our model gives a negative selling price),

Figure 2.21 R Output for Viewing the Model and Hypothesis tests

The output from this code should look like:

```
Residuals:
  Min   1Q Median   3Q   Max
-20448 -3536 -2110  2285 21115

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -31246.873  3485.634  -8.964 1.52e-12 ***
sqft         78.812    2.221  35.483 < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8030 on 58 degrees of freedom
Multiple R-squared:  0.956,    Adjusted R-squared:  0.9552
F-statistic: 1259 on 1 and 58 DF, p-value: < 2.2e-16
```

Residual information that we will ignore because we will look at a plot that will tell us more than this

These are the coefficients (or slope and intercept) of our regression model. The values to the right with *** next to it, signify that the slope and intercept are significant

The p-value of the F-statistics also tells us the regression model is significant

The R model output for out of sample error is given in Figure 2.22. R provides three measures of the predictive accuracy using the out of sample data (the OOS data or test data), root mean square error (RMSE), R-squared (R^2), and mean absolute error (MAE). Again, calculating these values using only the data used to build the model would have produced inaccurate error estimates.

Figure 2.22 R Output for Building the Regression Model for “housing”

```
Linear Regression

60 samples
 1 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 48, 48, 48, 48, 48
Resampling results:

    RMSE      Rsquared   MAE
 8162.942  0.960561  6015.2

Tuning parameter 'intercept' was held constant at a value of TRUE
```

RMSE

The RMSE error is analogous to the “standard deviation” of prediction errors. Many professional data analysts will prefer this measure of error because it can be quickly used for a rough estimate of prediction intervals (e.g. multiplying it by 2 and adding it to a prediction and subtracting it from a prediction roughly gives a 95% prediction interval). The RMSE for the test data is calculated as

$$\text{RMSE} = \text{square root of } \Sigma((\text{observed} - \text{predicted})^2/n)$$

where each predicted value from the test data is subtracted from the observed (true) value, squared, averaged and then a square root applied. For our example, the root mean square error is \$8,162, and we could roughly expect our model to predict a housing price within +/- \$16,324 of the true value 95% of the time.

R-squared

The R-squared is the correlation coefficient squared. It is calculated by dividing the sum of the squares of the residuals by the sum of the squares of the observations:

$$\text{R-squared} = 1 - \Sigma((\text{observed} - \text{predicted})^2) / \Sigma(\text{observed} - \text{mean of observed})^2$$

The closer the value of R-squared is to 1 the higher proportion of the variation in the data is explained by the model (a value of 1 means 100% of variation is explained by the model; a value of 0 means 0% of variation is explained by the model). Note again, this is the R-squared value for the test data (not the training data). The R-squared for our housing price example is 0.96, meaning roughly 96% of the variation in the test data was explained by the model.

MAE

The mean absolute error is calculated by subtracting each predicted value of the test data from the observed (true) value, taking the absolute value, and then taking the average:

$$\text{MAE} = \frac{\sum(|\text{observed} - \text{predicted}|)}{n}$$

The MAE can be interpreted as the expected average error for a single prediction from the model. In our example the MAE is \$6,015. On average, a prediction for a house sale price made by our model will miss the true sale price by \$6,015. Note, mean absolute percent error is a closely related metric to MAE this is sometimes used that simply converts the expected error to a percent.

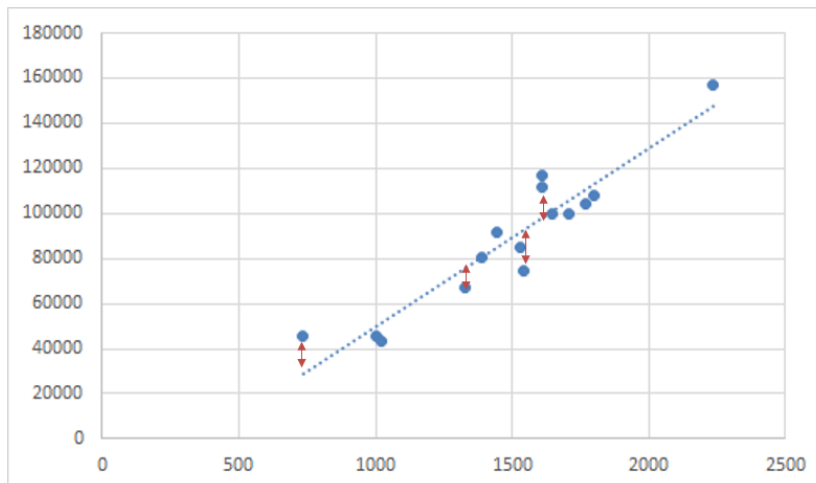
Does Our OOS Error Indicate Our Model Good?

What is considered a “good” model is relative—it depends on the business context. There is nothing in our hypothesis tests or OOS error to indicate this model should not be used, but determining if it is a “good” model depends on questions such as: What is the business currently using to make predictions of home sale prices? Is this better? How good are our competitors at making these kinds of predictions? What kind of error rates can our decision makers live with? Said another way, your model should be better than a random guess, but other than that, there are no “absolutes” when it comes using MSE, R-squared, and MAE to evaluate a model. For one situation an OOS R-square of 0.96 may be too inaccurate to be a useful model; in another completely different situation, using a model with an OOS R-squared of 0.39 might significantly improve profits for a company.

Visualizing Linear Regression and Prediction Intervals

Linear regression with just one predictor is easily visualized. Figure 3.23 visually demonstrates how a regression line is calculated so that the distance from each point to the line is minimized.

Figure 3.23 Visualizing the Process of Linear Regression

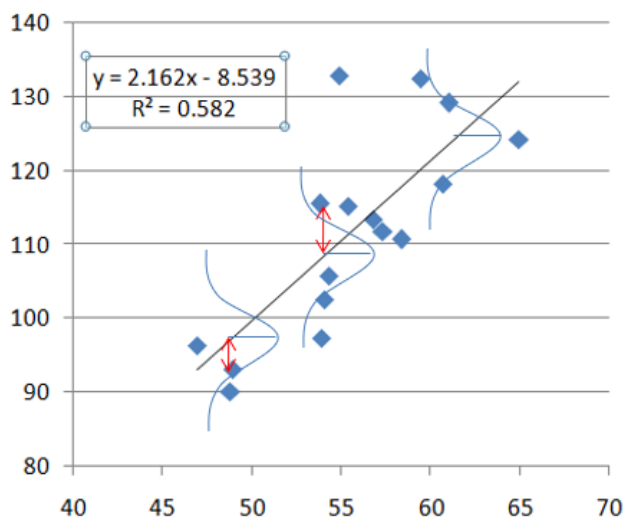


Exact prediction intervals (confidence intervals for individual predicted values), in contrast to using RMSE to roughly estimate a prediction interval, can be calculated for a prediction made with the regression model using the following formula:

$$\text{prediction interval of } \hat{x} = \pm t_{crit}(RMSE) \sqrt{1 + \frac{1}{n} + \frac{(\hat{x} - \bar{x})^2}{\sum_{i=1}^n (x_i - \bar{x})^2}}$$

Conceptually, prediction intervals for individual predictions made by the regression model can be thought of as normal distributions (t distributions because the standard deviation is estimated) with mean at the regression line and RMSE used to estimate the standard deviation (see Figure 3.24)

Figure 3.24 Visualizing How Prediction Intervals are Calculated



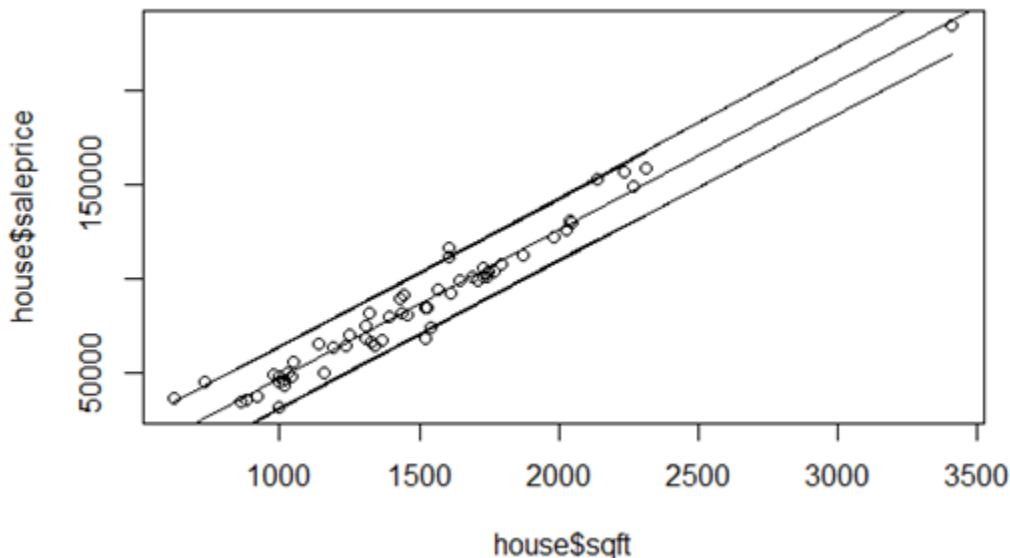
The following R code allows the regression line for the house example to be plotted with 95% prediction intervals. The prediction could be changed to 99% intervals (or any other confidence level) by changing the option on predict to level=0.99.

R Code for Visualizing the Regression Model with 95% Prediction Intervals

```
regmodel<-lm(saleprice~sqft,data=house)
plot(house$sqft,house$saleprice)
abline(model)
pred_interval <- predict(regmodel, newdata=house, interval="prediction",level = 0.95)
lines(house$sqft, pred_interval[,2])
lines(house$sqft, pred_interval[,3])
```

So, for our house sale price/square footage example, the lines surrounding the linear regression model in Figure 3.25 would be the 95% prediction limits. Quantifying uncertainty in this way is always better for supporting decisions. Instead of just saying, “We predict a 1400 square foot house will sale for \$62,900”, we can say “we expect 95% of 1400 square foot houses to sell between \$79,100 and \$95,300”, providing a decision maker with much more accurate information.

Figure 3.25 Regression Line with 95% Prediction Intervals for the “house” Data Set



To use the model we created to make a prediction, we create a data frame with the value of square feet for the house and then use the predict function. A prediction with 95% prediction intervals for a house of 1400 square feet can be found with the following R code (with output shown in Figure 3.26).

R Code for Using Our Model to Make a Prediction

```
newdata=data.frame(sqft=1400)  
predict(regmodel,newdata=newdata,interval="prediction",level=0.95)
```

Figure 3.26 Prediction with 95% Prediction Intervals for a 1400 Square-Foot House

```
fit      lwr      upr  
79090.41 62876.44 95304.38
```

Despite the fact that the prediction gives values down to a cent, in practice rounding the predicted house price would be best practice (to not imply our model is accurate down to the fraction of a cent). A responsible way to report a prediction for a 1400 square-foot house using this model would be: Based on the data used to train the model, the predicted house sale price will be \$79,100. However, we are approximately 95% sure that the house will sell for between \$62,900 and \$95,300. Also, keep in mind predictions are only valid from a range of 620 to 3,410 square-feet (the range of values used to train the model).

Summary for housing.csv Example

Full R Program

```
house <- read.csv("C:/Users/hbrown11/Desktop/housing.csv", header=TRUE)
head(house)
#install.packages("caret") #only needs to be ran once
library(caret)
set.seed(42)
train_control <- trainControl(method = "cv", number = 5)
model <- train(saleprice ~ sqft, data = house, method = "lm", trControl = train_control)
print(model)
summary(model)
regmodel <- lm(saleprice ~ sqft, data = house)
plot(house$sqft, house$saleprice)
abline(regmodel)
pred_interval <- predict(regmodel, newdata = house, interval = "prediction", level = 0.95)
lines(house$sqft, pred_interval[,2])
lines(house$sqft, pred_interval[,3])
newdata = data.frame(sqft = 1400)
predict(regmodel, newdata = newdata, interval = "prediction", level = 0.95)
```

Summary of Results

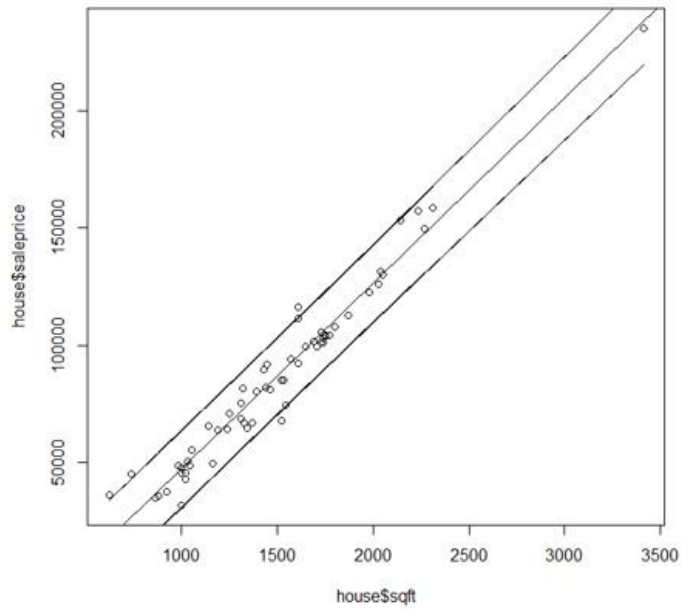
- A predictive model using ordinary least squares linear regression was built to predict house sale price using square footage
- Selected output is given in Figure 3.27
- The model may be written as
$$\text{house price estimate} = -31246.873 + 78.812 * (\text{square footage})$$
- The model is valid for square footages from 620 to 3,410
- The model was highly significant (p-value < 0.0001)
- For out of sample predictions, the RMSE is \$8,081, the R-Square is 0.96, and the MAE is \$6,204. On average, we can expect a prediction made by this model to be wrong by \$6024.

Figure 3.27 Selected Output for the home Regression Example

```
RMSE      Rsquared  MAE
8162.942  0.960561  6015.2

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -31246.873   3485.634  -8.964 1.52e-12 ***
sqft         78.812      2.221   35.483 < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8030 on 58 degrees of freedom
Multiple R-squared:  0.956,    Adjusted R-squared:  0.9552
F-statistic: 1259 on 1 and 58 DF,  p-value: < 2.2e-16
```



Note, the model was highly significant p-value of $<2.2e-16$, which is R scientific notation for 2.2×10^{-16} or 0.000000000000000022.

Regression Practice Exercise

The csv file “tenders” contains data from the food industry. The poultry company wants to be able to predict tender weight (an expensive piece of chicken breast meat) by measuring live weights, which can be done in advance on a farm. This will give the company advance knowledge on how much product they will have between certain weight ranges to meet customer orders. The tenders file has a column “tenders” containing tender weights in grams and a column “live” containing live weights in grams (see Figure 3.28).

Figure 3.28 Contents of the File “tenders.csv”

```
live tenders
3030    120
3540    134
3080    138
3030    127
3750    153
3880    157
```

Use the example program for housing.csv and complete the following (a solution will follow):

- Predict the tender weight from the live weight column
- Set-up 5-fold cross-validation
- Calculate MSE, R-square, and MAE for the out of sample data
- Check statistical hypothesis tests
- Create a plot of the regression line that includes 95% prediction intervals
- Use the model to make a prediction
- Briefly discuss the results

R Program for the “tenders.csv” Regression Exercise

```
tenders <- read.csv("C:/Users/hbrown11/Desktop/tenders.csv", header=TRUE)
head(tenders)
#install.packages("caret") #only needs to be ran once
#library(caret) #still loaded if you did the house ex in the same session
set.seed(42)
train_control <- trainControl(method = "cv", number = 5)
model <- train(tenders ~ live, data = tenders, method = "lm", trControl = train_control)
model
summary(model)
regmodel <- lm(tenders ~ live, data = tenders)
plot(tenders$live, tenders$tenders)
abline(regmodel)
pred_interval <- predict(regmodel, newdata = tenders, interval = "prediction", level = 0.95)
lines(tenders$live, pred_interval[,2])
lines(tenders$live, pred_interval[,3])
newdata = data.frame(live = 3500) #don't make predictions for x outside the range used to build the
model, the summary command can be used to check the max and min for x
predict(regmodel, newdata = newdata, interval = "prediction", level = 0.95)
```

Figure 3.29 What Had to be Changed in the R Code from the “house” Example:

```
tenders <- read.csv("C:/Users/hbrown11/Desktop/tenders.csv", header=TRUE)
head(tenders)
#install.packages("caret") #only needs to be ran once
#library(caret) #still loaded if you did the house ex in the same session
set.seed(42)
train_control <- trainControl(method = "cv", number = 5)
model <- train(tenders ~ live, data = tenders, method = "lm", trControl = train_control)
print(model)
summary(model)
regmodel <- lm(tenders ~ live, data = tenders)
plot(tenders$live, tenders$tenders)
abline(regmodel)
pred_interval <- predict(regmodel, newdata = tenders, interval = "prediction", level = 0.95)
lines(tenders$live, pred_interval[,2])
lines(tenders$live, pred_interval[,3])
newdata = data.frame(live = 3500) #don't make predictions for x outside of y range
#summary(tenders) #this will show the live max and min
predict(regmodel, newdata = newdata, interval = "prediction", level = 0.95)
```

Figure 3.30 Relevant Output for “tenders.csv” Regression Exercise

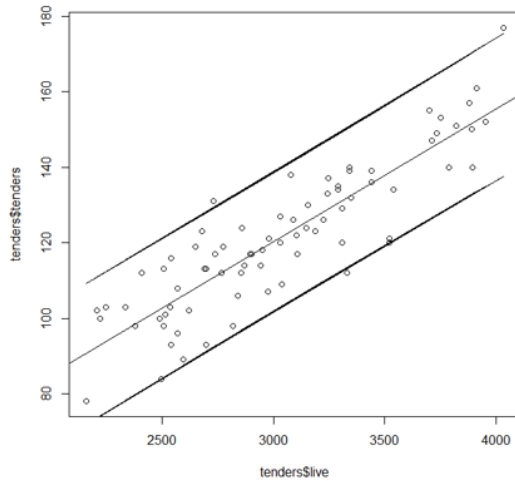
```

RMSE      Rsquared  MAE
9.198435  0.8049075  7.386984

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 14.519579   6.639455   2.187  0.0319 *
live         0.035215   0.002162  16.288 <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.148 on 75 degrees of freedom
Multiple R-squared:  0.7796,    Adjusted R-squared:  0.7767
F-statistic: 265.3 on 1 and 75 DF,  p-value: < 2.2e-16

```



Discussion of Results

- A predictive model using linear regression was built to predict tender weight using live weight
- The model may be written as

$$\text{tender} = 14.519579 + 0.035215 * (\text{live})$$
- The model is valid for live weights from 2160 to 4031 (grams)
- The model was highly significant (p-value <0.0001)
- For a live weight of 3500g, the model predicts a tender weight of 138g (with 95% prediction intervals of 119 and 156, i.e. our best guess of a tender weight for a 3500g live weight is 138g, but we are approximately 95% certain it will be between 119g and 156g)
- For out of sample predictions, the RMSE is 9.2, the R-Squared is 0.80, and the MAE is 7.4. On average, we can expect a prediction made by this model to be wrong by 7.4g.

Reminder, even though the R-Squared is lower and the MAE takes up a larger percentage of the prediction than our previous housing example, it does not necessarily imply this model is a “worse” model. Model efficacy (and profitability) is determined by business context, is this a better method of making a prediction than what is currently being used?

Multiple Regression

Multiple regression is regression with two or more predictor variables (or independent variables). It extends linear regression to a more complex scenario where multiple predictors are considered simultaneously. Since we no longer have just a slope and intercept, the parts of the regression model are called “parameters”. In multiple regression, the relationship between the dependent variable (often denoted as Y) and the independent variables (typically denoted as X_1, X_2, \dots, X_p) is represented by the following equation:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p + \varepsilon$$

In this equation: Y is the dependent variable we want to predict or explain, X_1, X_2, \dots, X_p are the independent variables, $\beta_0, \beta_1, \beta_2, \dots, \beta_p$ are the regression coefficients that represent the relationship between each independent variable and the dependent variable. They indicate the change in the dependent variable for a unit change in the corresponding independent variable holding all other variables constant, ε represents the residual or error term, which captures the unexplained variation in the dependent variable. The multiple regression model estimates the values of the regression coefficients based on a given dataset, aiming to find the best-fitting line or hyperplane that minimizes the sum of squared residuals.

Computer algorithms for multiple regression models work by solving equations in matrix form, the solution (which gives the parameter estimates) can be written as follows:

$$\mathbf{B} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$$

Where \mathbf{B} are the parameter estimates, \mathbf{X} is the matrix of input variables (predictors), \mathbf{Y} is a vector of outputs (responses).

Notice fitting statistical models involves the matrix algebra operations of transpose and inversion. Recall not all matrices can be inverted—however, this turns out to not be a major limitation for most statistical models (we avoid this by not using correlated input variables by feature selection or by dimension reduction, which will be covered later in this course). Also note, matrix inversion will not work on missing values (rows have to be deleted or missing values estimated).

If we are going to allow multiple predictors (inputs), the next question that might arise is how do we decide which inputs should be allowed to predict a particular output? This is called “model selection”, “feature selection”, or “variable selection. There are multiple methods of variable selection related to different techniques we will cover, so we will cover variable selection in Module 8 of this text. Until then, when given multiple variable to use in a model, you can assume all the predictors are needed for the model.

Multiple Regression in R

The csv file `housing2` contains a new set of real estate data (these data are from a larger geographical area than `housing`). See Figure 3.31. It includes the columns “SqFt”, the house square footage, “Age”, the age of the house in years, “Lotsize”, the size of the lot the house sits on in acres, and “SalePrice”, the sale price of the house. The goal of our regression analysis will be to build a multiple regression model

using Age, Lotsize, and SqFt to predict the target variable SalePrice. Additionally, we will use cross-validation to accurately determine the expected error when using the model.

Figure 3.31 “housing2.csv” Contents (First Few Rows)

SqFt	Age	Lotsize	SalePrice
1100	15	0.43	49600
1300	4	0.68	41700
1330	3	0.39	114500
1810	1	0.20	185000
3500	13	0.33	203100
800	4	2.08	73000

The R code for Building the Multiple Regression Model with 5-fold Cross-Validation is as Follows:

```
house2 <- read.csv("C:/Users/hbrown11/Desktop/housing2.csv", header=TRUE)
#reminder the file path will need to be changed to match where you saved the housing file
head(house2)
#install.packages("caret") #a reminder this only needs to be installed once
library(caret)
#Note we set a seed for the random number generation so that the random sampling will be
repeatable
set.seed(42)
set.seed(42)
train_control <- trainControl(method = "cv", number = 5)
model <- train(SalePrice ~., data = house2, method = "lm", trControl = train_control)
#model <- train(SalePrice ~Age+Lotsize+SqFt, data = house2, method = "lm", trControl =
train_control)
summary(model)
model
```

Note there are two ways the model can be specified in R:

SalePrice ~.

The “.” tells R to use all other columns in the table as predictors of SalePrice. Or alternatively,

SalePrice ~Age+Lotsize+SqFt

which tells R exactly which columns to use. The “.” is provided as a method to ease programming typing. However, be careful about using this for every modeling problem...if you have extra columns that should not be used in the model, such as an ID column, you cannot use this shortcut without creating a subset of the data that removes the unnecessary columns.

The performance of the model can be seen in Figure 3.32. The same error metrics are used for multiple regression and may be interpreted in the same manner. As with linear regression, RMSE, R-squared, and MAE are calculated for the out of sample data (not used to build the model). Once again, these measures are relative, and depend on business context to conclude if the model is valuable. These metrics also can be used to compare potential models.

Figure 3.32 Multiple Regression OOS Error for housing2.csv Example

```
RMSE      Rsquared  MAE
26834.47  0.8646122  21467.38
```

As can be seen in Figure 3.33, the model can be written with the following coefficients:

$$\text{SalePrice} = 16629.736 + 77.852 * \text{SqFt} - 2836.042 * \text{Age} + 8661.828 * \text{Lotsize}$$

This equation can be used to plug in values for square footage, age, and lot size, to predict house sale price. The model was highly significant p-value of $< 2.2e-16$, which is R scientific notation for 2.2×10^{-16} or 0.000000000000000022.

Figure 3.33 Coefficient Estimates and Hypothesis Tests for the housing2.csv Example

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 16629.736   8466.889    1.964  0.05244 .
SqFt         77.852     3.455   22.530 < 2e-16 ***
Age        -2836.042   288.472   -9.831 3.79e-16 ***
Lotsize     8661.828   3148.866    2.751  0.00712 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 25540 on 95 degrees of freedom
Multiple R-squared:  0.8757,    Adjusted R-squared:  0.8718
F-statistic: 223.1 on 3 and 95 DF,  p-value: < 2.2e-16
```

Notice a plot of the multiple regression model cannot be made. However, the model can be used to make predictions (for values within the range of the predictors used to the build the model). For example, to make a prediction with 95% prediction intervals for a square footage of 1400, a 4-year-old house, on a 0.5-acre lot, use the following R Code:

```
regmodel<-lm(SalePrice~.,data=house2)
summary(house2) #to see max and min of inputs
newdata=data.frame(SqFt=1400,Age=4,Lotsize=.5)
predict(regmodel,newdata=newdata,interval="prediction",level=0.95)
```


Figure 3.34 A Prediction Using the Multiple Regression Model for “housing2”

```
      fit      lwr      upr
1 118609.1 67347.55 169870.6
```

The prediction may be interpreted as, the predicted house sale price is \$118,609 and we are 95% confident it will be between \$67,3478 to \$169,871.

Summary of “housing2” Example

Full R Program:

```
house2 <- read.csv("C:/Users/hbrown11/Desktop/housing2.csv", header=TRUE)
head(house2)
library(caret)
set.seed(42)
train_control <- trainControl(method = "cv", number = 5)
model <- train(SalePrice ~., data = house2, method = "lm", trControl = train_control)
model
summary(model)
regmodel <- lm(SalePrice ~., data = house2)
summary(house2) #to see max and min of inputs
newdata = data.frame(SqFt=1400, Age=4, Lotsize=.5)
predict(regmodel, newdata=newdata, interval="prediction", level=0.95)
```

Figure 3.35 Selected Output “housing2”:

```
RMSE      Rsquared    MAE
26834.47  0.8646122  21467.38

Coefficients:
      Estimate Std. Error t value Pr(>|t|)
(Intercept) 16629.736   8466.889   1.964  0.05244 .
SqFt         77.852     3.455  22.530 < 2e-16 ***
Age        -2836.042    288.472  -9.831 3.79e-16 ***
Lotsize     8661.828    3148.866   2.751  0.00712 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 25540 on 95 degrees of freedom
Multiple R-squared:  0.8757,    Adjusted R-squared:  0.8718
F-statistic: 223.1 on 3 and 95 DF,  p-value: < 2.2e-16
```

Discussion of Results:

- A predictive model using ordinary least squares linear regression was built to predict house sale price using square footage, house age, and lot size
- The model may be written as
$$\text{SalePrice} = 16629.736 + 77.852 * \text{SqFt} - 2836.042 * \text{Age} + 8661.828 * \text{Lotsize}$$
- It is valid for square footages from 790 to 5800; ages from 1 to 42; and lot sizes from 0.03 to 3.52.
- The model was highly significant (p-value <0.0001)
- For out of sample predictions, the RMSE is 26834, the R-Squared is 0.96, and the MAE is 21467. On average, we can expect a prediction made by this model to be wrong by \$21,467.

Comparing models

The error for out of sample data is useful for comparing different candidate models. For example, if we used only square footage as a predictor for the housing2.csv example, we would have the following program and results:

```
set.seed(42)
```

```
train_control <- trainControl(method = "cv", number = 5)
```

```
model <- train(SalePrice ~ SqFt, data = house2, method = "lm", trControl = train_control)
```

```
model
```

Figure 3.36 OOS Error Using only Square Footage as a Predictor for the housing2 Data

```
RMSE      Rsquared  MAE
37454.59  0.6913571  30760.08
```

Comparing the results from Figure 3.35 to Figure 3.36, the model using all three inputs performs much better than using SqFt alone.

Module 3 Assignment

Part 1:

The data found in the csv file “APCuv” are data from a food industry experiment that is seeking to use a UV instrument to predict log₁₀ APC levels in raw meat food samples (you explored this data last module to address information quality, so you DO NOT need to do EDA). The UV instrument is a new method of ultraviolet bacteria detection; log₁₀ APC refers to a log₁₀ transformation of areobic plate count (a type of bacteria). Raw meat product must be sampled to make sure bacteria levels are below what would indicate spoilage or potential safety concerns. Traditional bacteria testing techniques are destructive to the product and requires a plating and incubation period of approximately 48 hours (a long time when trying to ship food products). The UV instrument attempts to instantly estimate the amount of bacteria on the raw meat. However, a relationship between the UV reading and the tried and trusted log₁₀ APC results must be established via a regression model. Building a good model would potentially speed up and improve the process of determining if there are any food safety issues in raw food.

Using R, build a linear regression model for the attached data to predict Log₁₀APC results using the UV light readings as a predictor. Using 5-fold cross-validation, build a regression model, give the RMSE, R-squared, and MAE, check that the F-test is significant, build 95% prediction intervals and plot the model and prediction intervals, and make a valid prediction for a new UV value (not extrapolation).

Create a document that briefly summarizes what you did, what you found, any key graphs or calculation etc., and includes your full R program (R commands).

Part 2:

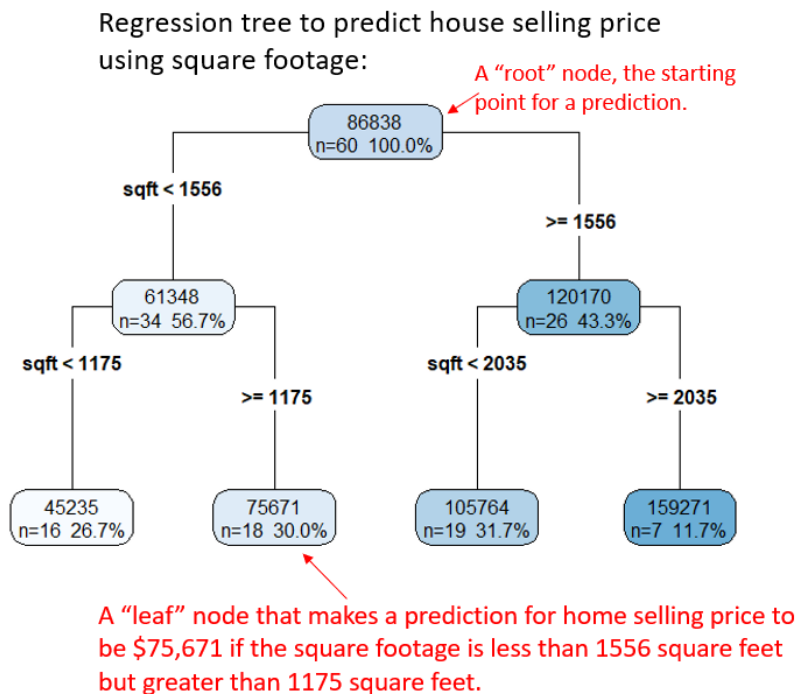
The csv file halloweenCandy contains retail sales data for stores in a large retail chain in the weeks nearing Halloween. Some stores ran out of Halloween candy in a previous year, using the attached data from stores that did not run out of candy, we want to build a model to predict Halloween candy sales based on the sales of other items (so that predictions can be made on how much candy should be ordered for this year for stores that ran out last year). The file halloweenCandy, contains sales data in columns for Candy (Halloween Candy sales), Costume (Halloween costume sales), Decorations (Halloween decoration sales), and Pumpkin (sales of pumpkin). You do NOT need to do exploratory data analysis on these data (in real life you would). Build a model to predict Candy Sales (assume Costume, Decorations, and Pumpkin Sales are all needed--you do NOT have to do variable selection), use 5-fold cross-validation, give the RMSE, R-squared, and MAE, check that the F-test is significant, and to demonstrate the model by making a valid new prediction of Candy Sales for new values of costume, decorations, pumpkin sales. Add the results to your document from Part 1, briefly summarizing what you did, what you found, any key graphs or calculation etc., and include your full R program (R commands).

Module 4: Building Predictive Models with Regression Trees and Random Forests for Regression

Introduction to Regression Trees

Regression trees, also known as "decision trees," are an intuitive approach to predictive modeling. When employed to forecast continuous variables or numeric data, they are referred to as "regression trees." A trained regression tree creates a tree-like structures where the path to making a prediction can be visualized. The construction of regression trees involves algorithms that attempt to optimally partition the predictor variables, forming branches and nodes, by minimizing errors in the response variable at each node. By recursively branching based on different features, these trees can offer a systematic and interpretable framework for analyzing data, enabling accurate predictions while also providing valuable insights into the underlying relationships within the dataset (see Figure 4.1).

Figure 4.1 An Example Regression Tree to Predict House Selling Price Using Square Footage



Regression and classification trees originated in the field of data analysis and machine learning. The concept of decision trees can be traced back to the 1960s, with the work of Arthur Samuel and Edward Feigenbaum on the development of computer programs capable of learning from data. However, the formalization and popularization of decision trees as a standalone technique can be attributed to the work of Leo Breiman, Jerome Friedman, Charles Stone, and Richard Olshen in the early 1980s. In their seminal book "Classification and Regression Trees," published in 1984, they presented a comprehensive framework for constructing decision trees and introduced innovative algorithms for recursive partitioning. This book laid the foundation for the widespread adoption and further advancement of decision trees, leading to the development of more sophisticated tree-based algorithms, such as Random Forests and gradient boosting, in subsequent years.

Advantages of Regression Trees

As a modeling technique, regression trees offer several advantages. Regression trees are more intuitive to use than many techniques. They are relatively computationally inexpensive to create and use. Further, the tree-like structures provide a visual interpretation of the model. While regression trees are known to be one of the less accurate machine learning approaches, their performance can be improved by strategies involving multiple trees (e.g. Random Forests). Finally, tree models handle categorical predictor variables more naturally than other techniques in which categories must often be turned into indicator variables (recoded as 0s and 1s).

Disadvantages of Regression Trees

There are likewise several disadvantages to regression trees as a predictive model technique. Regression trees are prone to overfitting data (for this reason “good” software that builds regression trees will have built in cross-validation, use a complexity parameter, and other techniques, such as a minimum number of values in a leaf node, to prevent overfitting). Further, for large amounts of data and a large number of predictors, a tree would become too large to be visualized. Finally, because of an inability to model complex relationships between predictor variables, unless multiple trees are combined, single regression trees do not perform as well as other techniques.

How a Regression Tree Works

An unrealistically small example is presented to help conceptualize how a regression tree is trained. For the table presented in Figure 4.2, we will build a regression tree to predict Y from X.

Figure 4.2 Small Sample of Data to Demonstrate Regression Tree Training

X	Y
6	16
4	20
8	28
2	10

Begin by sorting by X as seen in Figure 4.3.

Figure 4.3 The Data Sorted by X

X	Y
2	10
4	20
6	16
8	28

Divide the data into all possible splits using X. The first possible split could be made at $X < 3$, $X \geq 3$ (the average of the first two rows of X (Figure 4.4)). The average of Y for each split becomes the predicted value of Y for that split. For example, the average of Y for the top split in Figure 4.4 would be 10; the average of Y for the bottom split would be 21.33. The mean square error (MSE) can then be calculated as the average of $(\text{Actual Y} - \text{Predicted Y})^2$. For the first possible split the MSE can be calculated as:

$$((10-10)^2 + (20-21.33)^2 + (16-21.33)^2 + (28-21.33)^2) / 4$$

Thus, the MSE = 18.67 for possible split 1.

Figure 4.4 The First Possible Split

X	Y
2	10
4	20
6	16
8	28

This process is repeated for each possible split. The second possible split, split 2, is given in Figure 4.5 and would be made at $X < 5$, $X \geq 5$. The average of Y for the top split in Figure 4.5 would be 15; the average of Y for the bottom split would be 22. For the second possible split the MSE can be calculated as:

$$((10-15)^2 + (20-15)^2 + (16-22)^2 + (28-22)^2) / 4$$

MSE = 30.5 for possible split 2.

Figure 4.5 The Second Possible Split

X	Y
2	10
4	20
6	16
8	28

The third possible split, split 3, is given in Figure 4.6 and would be made at $X < 7$, $X \geq 7$. The average of Y for the top split in Figure 4.6 would be 15.33; the average of Y for the bottom split would be 28. For the second possible split the MSE can be calculated as:

$$((10-15.33)^2 + (20-15.33)^2 + (16-15.33)^2 + (28-28)^2) / 4$$

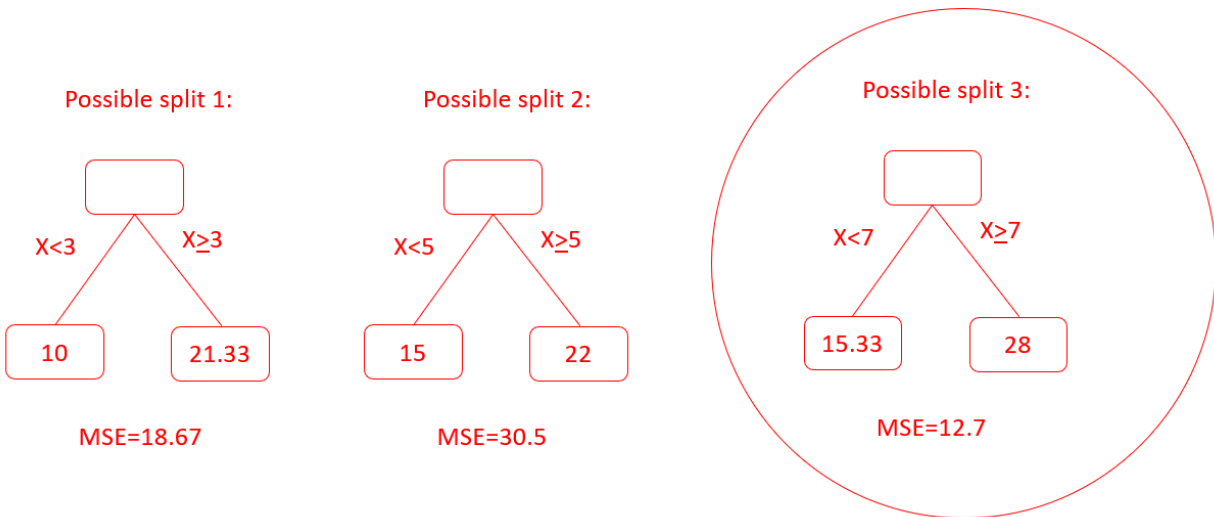
MSE = 12.7 for possible split 3.

Figure 4.6 The Third Possible Split

X	Y
2	10
4	20
6	16
8	28

After considering all possible splits, split three would be chosen (Figure 4.7) because it minimizes MSE.

Figure 4.7 The Selection of Possible Tree Splits



In the selected regression tree, if $X < 7$ then the regression tree predicts Y will be 15.33; if $X \geq 7$, the regression tree predicts Y will be 28.

If we had more rows of data, we would consider a second split for each of our nodes. The process we just completed would be repeated for the subset of data at each node. If we have multiple predictors, the same process is repeated for each predictor, the predictor minimizing MSE is used at each node.

Overfitting is a problem for regression trees. MSE can always be made 0 by continuing the regression tree until each node only has one value, but the resulting tree is likely unrealistically fit to the training data. Good regression tree algorithms (such as the one used in R) will use built-in cross-validation to prevent overfitting. Other settings, such as a minimum number of values in each leaf node, are also used to prevent overfitting.

Regression Trees with R

Regression trees can be easily constructed and visualized in R. To demonstrate a regression tree, we will begin by looking at the house real-estate example.

The csv file housing contains real estate data from a particular area of a city (see Figure 4.8). It includes the columns “sqft”, the house square footage, and “saleprice”, the sale price of the house. The goal of our regression tree will be to build a model using sqft to predict saleprice. Additionally, we will use cross-validation to accurately determine the expected error when using the model.

Figure 4.8 Contents of the File “housing”

```
sqft saleprice
1326    66960
1391    80400
1000    45600
1542    74400
  735    45360
1444    91920
```

As a change from last module, we will use training and testing data sets instead of cross-validation to demonstrate an alternative to cross-validation. Note, both regression trees and Random Forest in R use cross-validation as a part of their algorithms in the model fitting process. Using training and testing sets will allow us to demonstrate another method of evaluating models with out of sample data.

There is more than one way to split data into training and testing data sets. In the following data set, we will do this manually using the features available in base R without installing additional libraries. We will set a random seed, create a column of pseudo-random values between 0 and 1 in our data set, split the data based on the random values into subsets of about 60% for training and 40% for testing, and finally remove the random column. These steps can be seen in the following R code.

The R code for Dividing Data into Training and Testing Data Sets:

```
house <-read.csv("C:/Users/hbrown11/Desktop/housing.csv", header=TRUE)
#reminder the file path will need to be changed to match where you saved the housing file
head(house)
set.seed(19) #make the split repeatable
R=runif(nrow(house)) #create a column of random values between 0 and 1 that is the length of our
table (from the random uniform distribution)
house$R=R #add the random numbers to our table
train<-house[house$R<=.6,] #select roughly 60% for training
test<-house[house$R>.6,] #the other 40% gets placed in test
train<-subset(train,select=-c(R)) #remove the column R
test<-subset(test,select=-c(R)) #remove the column R
```

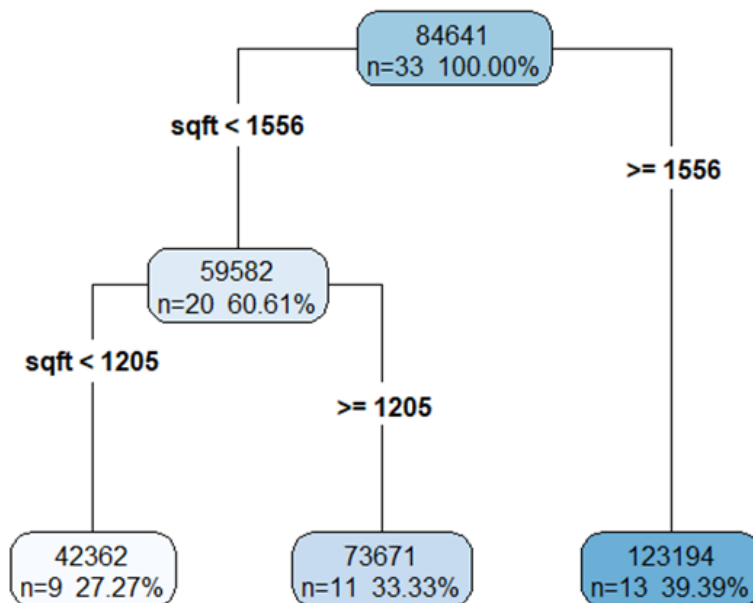
Once the data are divided into training and test data sets, we can build the regression tree.

The R code for Building the Regression Tree:

```
install.packages('rpart', dependencies = TRUE) #once installed, this will not need to be ran again
install.packages('rpart.plot', dependencies = TRUE) #once installed, this will not need to be ran again
library(rpart)
library(rpart.plot)
tree<-rpart(saleprice~sqft,data=train)
#if you want to see details about the tree fitting process use summary and print
#summary(tree)
rpart.plot(tree, type = 4, extra = 101, digits=-4)
```

In the plot command to graphically display the tree, the following options were used: type=4 labels the splits and the nodes in the tree, extra=101 adds extra statistics to each node, digits=-4 prevents R from using scientific notation. The regression tree fit by R is displayed in Figure 4.9.

Figure 4.9 Regression Tree to Predict Sale Price From the “housing” Data



Note: Your tree may look a little different, if you used a different starting seed for dividing the data into training and test datasets, or if you are using a different version of R (or different operating system), our training data may be different (this doesn't mean it is wrong). The tree could be used as follows: to make a prediction for a house that is 1300 square feet, start at the root node and move to the left, since 1300

is less than 1556; then, move to the right, since 1300 is more than 1205. The leaf node predicts the 1300 square-foot house to sell for \$73,671.

Running `summary(tree)` will show, among other things, a behind the scenes statistic, CP, that it used to “prune” the tree. CP is the “Complexity Parameter” which is calculated as the sum of the error and the number of nodes times a tuning parameter; it is used to determine when to stop splitting a tree to avoid over-fitting. CP is not useful for comparing model results (for example, to compare trees to regression, because it is not used in other techniques). Cross-validation is used with CP to reduce over-fitting as a part of the **rpart** algorithm.

To evaluate the regression tree with the test data, the following code is used.

R Code for Evaluating the Regression Tree:

```
predicted <- predict(tree,newdata=test) #apply tree to test data  
mae<-mean(abs(test$saleprice-predicted)) #calculate mae  
cat("MAE",mae,"\n") #just prints the name and the value of mae  
rmse<-((mean((test$saleprice-predicted)**2))**.5  
cat("RMSE",rmse,"\n")  
rsquared=1-( sum((predicted-test$saleprice)^2))/(sum((test$saleprice-mean(test$saleprice))^2))  
cat("Rsquared",rsquared,"\n")
```

The regression tree produces OOS error estimates of a MAE of 13889.34, a RMSE of 16449.89, and an R-squared valued of 0.7411603.

Compared to the Module 3 results for linear regression of a MAE of 6015.2, a RMSE of 8162.942, and an R-squared valued of 0.960561, the regression model is a better technique to use for these data than the decision tree. In general, ordinary least-square regression should perform better than a tree when there is a simple linear relationship between two continuous variables

Summary of Example 1 “housing”

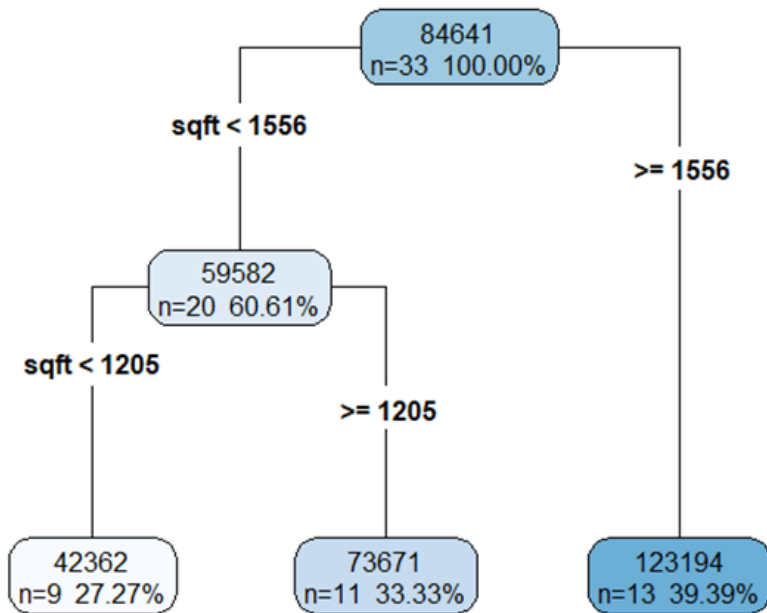
The Full R Program

```
house <-read.csv("C:/Users/hbrown11/Desktop/housing.csv", header=TRUE)
head(house)
set.seed(19)
R=runif(nrow(house))
house$R=R
train<-house[house$R<=.6,]
test<-house[house$R>.6,]
train<-subset(train,select=-c(R))
test<-subset(test,select=-c(R))
library(rpart)
library(rpart.plot)
tree<-rpart(saleprice~sqft,data=train)
rpart.plot(tree, type = 4, extra = 101, digits=-4)
predicted <- predict(tree,newdata=test)
mae<-mean(abs(test$saleprice-predicted))
cat("MAE",mae,"\n")
rmse<-((mean((test$saleprice-predicted)**2))**.5)
cat("RMSE",rmse,"\n")
rsquared=1-( sum((predicted-test$saleprice)^2))/(sum((test$saleprice-mean(test$saleprice))^2))
cat("Rsquared",rsquared,"\n")
```

The tree and OOS error estimates are the primary output of interest for the regression tree.

Conclusions for the Regression Tree are Given as Follows:

Figure 4.9 Regression Tree to Predict Sale Price From the “housing” Data



- The regression tree error:
 - MAE = \$13,889.34
 - RMSE = \$16,449.89
 - R-squared = 0.7411603
- On average a prediction for house price using this model will be wrong by \$13,889.
- Unless there is a need to have a tree structure, the regression model should be used because it resulted in less error.

Regression Tree Practice Exercise

The csv file “payment” (see Figure 4.10) contains information for patients that have a balance on a procedure not covered by insurance (or the government) that is eventually paid:

DaysUntilPaid – number of days until a procedure is paid for in full

PrivateInsurance – Does the patient have private insurance (yes or no)?

OutPatient – Was the procedure out-patient (yes or no)?

Employed – Is the patient currently employed (yes or no)?

ProcedureCost – cost of the procedure

Figure 4.10 Contents of the File payment (First Few Rows)

DaysUntilPaid	PrivateInsurance	OutPatient	Employed	ProcedureCost
54	yes	no	no	4061.11
89	yes	yes	yes	2234.42
0	no	yes	yes	2375.29
38	yes	yes	yes	2986.60
2	no	yes	yes	2908.53
24	no	no	yes	4071.88

Use the example program for house and complete the following (a solution will follow): Read payment.csv into R, divide the data into training and test data sets, construct a decision tree to predict how long it will take a patient to pay for their procedure (the target variable is DaysUntilPaid), plot the tree, calculate MAE, RMSE, and R-squared.

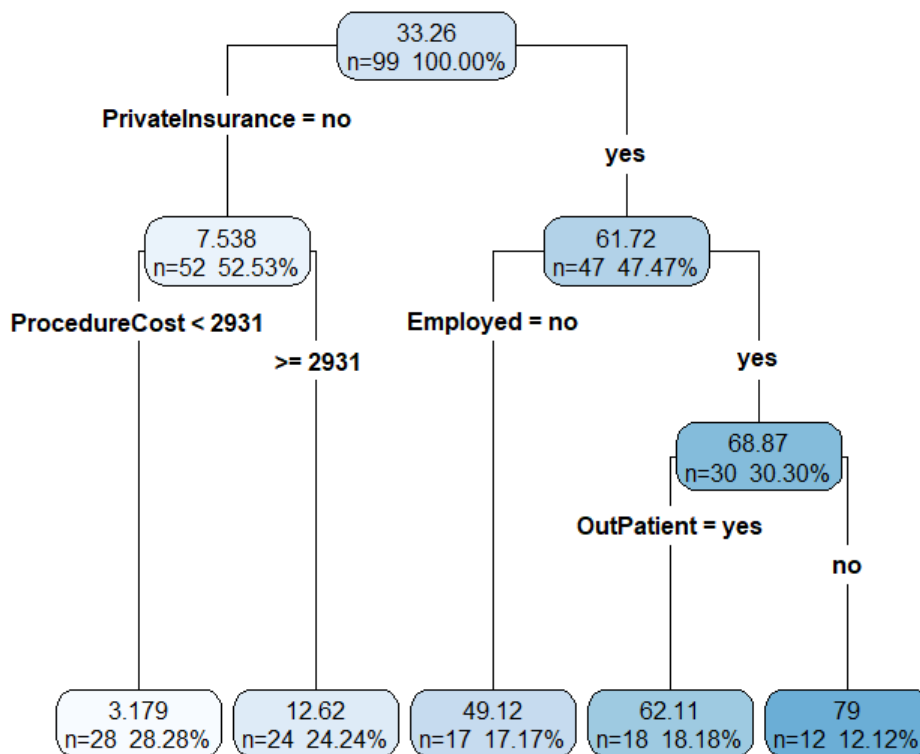
Solution to Practice Exercise

R code:

```
payment<-read.csv("C:/Users/hbrown11/Desktop/payment.csv", header=TRUE)
head(payment)
set.seed(19)
R=runif(nrow(payment))
payment$R=R
train<-payment[payment$R<=.6,]
test<-payment[payment$R>.6,]
train<-subset(train,select=-c(R))
test<-subset(test,select=-c(R))
library(rpart)
library(rpart.plot)
tree<-rpart(DaysUntilPaid~.,data=train)
rpart.plot(tree, type = 4, extra = 101, digits=-4)
predicted <- predict(tree,newdata=test)
mae<-mean(abs(test$DaysUntilPaid-predicted))
cat("MAE",mae,"\n")
rmse<-((mean((test$DaysUntilPaid-predicted)**2))**.5)
cat("RMSE",rmse,"\n")
rsquared=1-( sum((predicted-test$DaysUntilPaid)^2))/(sum((test$DaysUntilPaid-
mean(test$DaysUntilPaid))^2))
cat("Rsquared",rsquared,"\n")
```


The resulting tree:

Figure 4.11 Tree to Predict Payment Days from the payment Example



Using the tree: A patient with private insurance, who is employed, and has an out-patient procedure would be predicted to pay in 62 days.

Conclusions:

The regression tree error:

MAE = 13.73276

RMSE = 18.65054

R-squared = 0.6791615

On average a prediction for number of days until payment will be wrong by approximately 14 days

Is this a good model? Again, it depends. The three error numbers above cannot tell us this by themselves. Is this a better method than what is currently being used? Do we need a simple model where someone can just check the tree to make a prediction?

Random Forests

Random Forests are a high performing ensemble machine learning method that combines the predictions of multiple decision trees to improve the accuracy and robustness of the predictions. (Ensemble methods combine the results from multiple models.) It was first introduced by Leo Breiman in 2001 as an extension of decision trees. Random Forests have gained significant popularity and become a widely used predictive modeling technique.

Random Forests work by creating an ensemble of decision trees, each trained on a random subset of the training data and using a random subset of the features (predictor variables). During the prediction phase, each tree in the forest independently generates a prediction, and the final prediction is determined by aggregating the individual predictions through averaging for regression Random Forests.

Leo Breiman's introduced Random Forests in his paper titled "Random Forests" published in 2001. In this paper, Breiman introduced the concept of Random Forests and highlighted their ability to reduce overfitting, handle high-dimensional data, and provide reliable predictions. The paper demonstrated the effectiveness of Random Forests through various experiments and comparisons with other classification algorithms. Since then, there has been a widespread adoption and diverse applications of Random Forest.

Random Forests address many of the issues of a single regression tree. When constructing a regression tree, there are potentially numerous ways to split trees, and no one way of creating a tree guarantees an optimal tree (small changes can result in drastically different trees being created). Further, determining all possible trees for a large regression problem is unfeasible. In a solution to these issues, Random Forests, create numerous "random" trees (each is potentially a "good" tree by using criteria such as MSE), each tree makes a prediction for a particular set of predictors, the average of the predictions is used as the final prediction.

Advantages of Random Forests

Random Forests have several advantages as a method of predictive modeling. Random Forests are intuitive since they are based on regression trees. However, they can handle much more complexity than a single regression tree. Like regression trees, Random Forests handles categorical inputs more naturally than other techniques. For some problems, Random Forests can outperform many other popular machine learning and statistical techniques. They are considered one of the better modern machine learning techniques. Random Forests have the advantage of producing variable importance measures that can be useful for feature selection or other decisions (we will cover this in Module 8). As we will also see, Random Forests can be used for regression problems or classification problems (causing a little confusion, both are called "Random Forests").

Disadvantages of Random Forests

Random Forests also have several disadvantages as a method of predictive modeling. Random Forests are a "black-box" technique. Since numerous trees are created, there is not a single tree plot that can be visualized and this desirable quality of regression trees is lost. For large amounts of data and large

numbers of predictors, Random Forests become computationally expensive. Although there are suggestions for how many variables to consider, Random Forests can be sensitive to the “mtry” option. Despite these limitations, Random Forests are a popular and effective approach for predictive modeling.

Details of How a Random Forest Algorithm Works

In a Random Forest, a regression tree algorithm (as presented earlier in this module) is used to build a large number of trees, each tree will look at some subset of the predictor variables (all variables are not used at one time). The default values for R are floor (number of predictors/3) predictors per tree (i.e. take the number of predictors divide it by three and truncate it to the nearest integer). This can be changed with the mtry option. 500 trees are created by default in R (this can be changed with the ntrees option). To make a prediction, specific values of the predictors will be input into the 500 trees. A prediction will be made by averaging the result of all 500 trees.

Random Forests in R

We will begin by using the housing data again. Recall, the csv file housing contains real estate data from a particular area of a city (see Figure 4.11). It includes the columns “sqft”, the house square footage, and “saleprice”, the sale price of the house. The goal of our Random Forest will be to build a linear model using sqft to predict saleprice. Additionally, we will use cross-validation to accurately determine the expected error when using the model.

Figure 4.11 Contents of the File “housing”

```
sqft saleprice
1326 66960
1391 80400
1000 45600
1542 74400
 735 45360
1444 91920
```

The following code randomly divides the data into training and test data sets. It creates the same subsets we just used for regression trees, so the following code would not need to be ran twice if both the regression tree and Random Forest were created at the same time.

The R Code for Dividing Data into Training and Testing Data Sets:

```
house <-read.csv("C:/Users/hbrown11/Desktop/housing.csv", header=TRUE)

#reminder the file path will need to be changed to match where you saved the housing file

head(house)

set.seed(19) #make the split repeatable

R=runif(nrow(house)) #create a column of random values between 0 and 1 that is the length of our
table (from the random uniform distribution)

house$R=R #add the random numbers to our table

train<-house[house$R<=.6,] #select roughly 60% for training
test<-house[house$R>.6,] #the other 40% gets placed in test

train<-subset(train,select=-c(R)) #remove the column R
test<-subset(test,select=-c(R)) #remove the column R
```

Once the data is divided into training and testing data sets, we can build the Random Forest.

R Code for Random Forest for "house" Data Set:

```
install.packages('randomForest', dependencies = TRUE)

library(randomForest)

RFM=randomForest(saleprice~sqft,data=train)

#summary(RFM)

predicted <- predict(RFM,newdata=test)

mae<-mean(abs(test$saleprice-predicted))

cat("MAE",mae,"\n")

rmse<-((mean((test$saleprice-predicted)**2))**.5

cat("RMSE",rmse,"\n")

rsquared=1-( sum((predicted-test$saleprice)^2))/(sum((test$saleprice-mean(test$saleprice))^2))

cat("Rsquared",rsquared,"\n")
```

Results

The Random Forest model results in the following error for the test data:

MAE 7770.567

RMSE 18060.23

R-squared 0.823324

Compare these to the single regression tree and we can see the improvement of the forest:

The regression tree error:

MAE 13889.34

RMSE 16449.89

R-squared 0.7411603

Note, the regression model from the previous module still performs best for these data.

Full R Code for "house" Random Forest Example

```
house <-read.csv("C:/Users/hbrown11/Desktop/housing.csv", header=TRUE)
```

```
head(house)
```

```
set.seed(19
```

```
R=runif(nrow(house))
```

```
house$R=R
```

```
train<-house[house$R<=.6,]
```

```
test<-house[house$R>.6,]
```

```
train<-subset(train,select=-c(R))
```

```
test<-subset(test,select=-c(R))
```

```
install.packages('randomForest', dependencies = TRUE)
```

```
library(randomForest)
```

```
RFM=randomForest(saleprice~sqft,data=train)
```

```
predicted <- predict(RFM,newdata=test)
```

```
mae<-mean(abs(test$saleprice-predicted))
```

```
cat("MAE",mae,"\n")
```

```
rmse<-((mean((test$saleprice-predicted)**2))**.5
```

```
cat("RMSE",rmse,"\n")
```

```
rsquared=1-( sum((predicted-test$saleprice)^2))/(sum((test$saleprice-mean(test$saleprice))^2))
```

```
cat("Rsquared",rsquared,"\n")
```

Conclusions for Random Forest “house” Example

- The regression tree error:
 - MAE 7770.567
 - RMSE 18060.23
 - R-squared 0.823324
- On average a prediction for house price using the Random Forest model will be wrong by \$7,770.
- The regression model should still be used because it resulted in less error.

Random Forest for Regression Practice Exercise

The csv file “payment” (see Figure 4.12) contains information for patients that have a balance on a procedure not covered by insurance (or the government) that is eventually paid:

DaysUntilPaid – number of days until a procedure is paid for in full

PrivateInsurance – Does the patient have private insurance (yes or no)?

OutPatient – Was the procedure out-patient (yes or no)?

Employed – Is the patient currently employed (yes or no)?

ProcedureCost – cost of the procedure

Figure 4.12 Contents of the File “payment” (First Few Rows)

DaysUntilPaid	PrivateInsurance	OutPatient	Employed	ProcedureCost
54	yes	no	no	4061.11
89	yes	yes	yes	2234.42
0	no	yes	yes	2375.29
38	yes	yes	yes	2986.60
2	no	yes	yes	2908.53
24	no	no	yes	4071.88

Use the example program for house and complete the following (a solution will follow): Read payment.csv into R, divide the data into training and test data sets, construct a Random Forest to predict how long it will take a patient to pay for their procedure (the target variable is DaysUntilPaid), plot the tree, calculate MAE, RMSE, and R-squared.

Solution Random Forest Exercise

R code for payment Random Forest example (default Random Forest settings):

```
payment<-read.csv("C:/Users/hbrown11/Desktop/payment.csv", header=TRUE)
head(payment)
set.seed(19)
R=runif(nrow(payment))
payment$R=R
train<-payment[payment$R<=.6,]
test<-payment[payment$R>.6,]
train<-subset(train,select=-c(R))
test<-subset(test,select=-c(R))
library(randomForest)
RFM=randomForest(DaysUntilPaid~.,data=train)
predicted <- predict(RFM,newdata=test)
mae<-mean(abs(test$DaysUntilPaid-predicted))
cat("MAE",mae,"\n")
rmse<-((mean((test$DaysUntilPaid-predicted)**2))**.5)
cat("RMSE",rmse,"\n")
rsquared=1-(sum((predicted-test$DaysUntilPaid)^2))/(sum((test$DaysUntilPaid-
mean(test$DaysUntilPaid))^2))
cat("Rsquared",rsquared,"\n")
```

Results

The Random Forest OOS errors were, MAE=17.77911, RMSE=21.219, and R-squared=0.5847079. Compared to the regression tree OOS errors of MAE=13.73276, RMSE=18.65054, R-squared=0.6791615. Because the regression tree out-performed the Random Forest, it is a clear sign the parameters of the Random Forest need to be adjusted. Trying different values of mtry (to force each tree to try a different number of columns as predictors) and a larger than 500 number of trees are typical actions to improve a Random Forest. mtry can be anything between 1 and the number of predictor columns minus 1. Usually more than 1000 trees are not needed.

Fine Tuning a Random Forest Model:

```
RFM=randomForest(DaysUntilPaid~.,data=train,mtry=2, ntrees=1000)
predicted <- predict(RFM,newdata=test)
mae<-mean(abs(test$DaysUntilPaid-predicted))
cat("MAE",mae,"\n")
rmse<-(mean((test$DaysUntilPaid-predicted)**2))**.5
cat("RMSE",rmse,"\n")
rsquared=1-( sum((predicted-test$DaysUntilPaid)^2))/(sum((test$DaysUntilPaid-
mean(test$DaysUntilPaid))^2))
cat("Rsquared",rsquared,"\n")
```

Conclusions

Trying out different values for mtry and increasing the number of trees to 1000, results in better results: MAE=13.91291, RMSE=18.34712, R-squared=0.6895159. On average, a prediction for the number of days until payment will be wrong by approximately 14 days.

The Dangers of Not Using Test/Validation Data

Now that we have tried a few different methods of predictive analytics, we will demonstrate why data must always be split into test/validation data to develop a predictive model. As we have already stated, when fitting models for the purpose of predication, we should always use some form of model validation that examines the effectiveness of the model on data that was not used to build the model. The simplest way of doing this is to randomly divide data into training and test data sets. Cross-validation with folds is another popular way. Regardless of which particular method is used, the key idea is that data that are used to build the model are not used to evaluate the model. If we only use the data that were used to construct the model to evaluate the model, we will get an unrealistic view of our model's performance (the model was optimally fit to those data).

Overfit example

The csv dataset entitled "exOverfit" contains six predictor variables (x1 to x6) and the target variable of y. The data have already been partitioned into train and test sets (Figure 4.13).

Figure 4.13 The First Few Rows of the Data Set exOverfit

```
type x1  x2    x3      x4      x5      x6    y
train 66 4356 287496 18974736 1252332576 82653950016 89.4
train 48 2304 110592 5308416 254803968 12230590464 70.2
train 12 144 1728 20736 248832 2985984 7.8
train 95 9025 857375 81450625 7737809375 735092000000 92.5
train 41 1681 68921 2825761 115856201 4750104241 31.9
train 23 529 12167 279841 6436343 148035889 18.7
```

The following R program constructs a multiple regression model from training data, evaluates the model with only the training data, and then evaluates the model with the test data to illustrate the importance of using test data or cross-validation

R Program for the Training Data:

```
overfit <- read.csv("C:/Users/hbrown11/Desktop/exOverfit.csv", header=TRUE)
head(overfit)
train <- overfit[overfit$type == 'train',]
test <- overfit[overfit$type == 'test',]
train <- subset(train, select = -c(type)) #remove the column type
test <- subset(test, select = -c(type)) #remove the column type
head(train)
head(test)
regmodel <- lm(y ~ ., data = train)
summary(regmodel)
#calculate MAE, RMSE, and Rsquare for the training data
predicted <- predict.lm(regmodel, newdata = train)
mae <- mean(abs(train$y - predicted))
cat("MAE", mae, "\n")
rmse <- (mean((train$y - predicted)**2))**.5
cat("RMSE", rmse, "\n")
rsquared = 1 - (sum((predicted - train$y)^2)) / (sum((train$y - mean(train$y))^2))
cat("Rsquared", rsquared, "\n")
```

Just to reiterate, what we are doing is to demonstrate the incorrect notion of evaluating a model only using the data that was used to build the model. (So far this semester we have been calculating MAE, RMSE, and R-squared for the test data or out of sample data, which is the correct thing to do; in this example, we are calculating MAE, RMSE, and R-squared for the training data.)

Results of Evaluating the Model Only with the Training Data

As can be seen in Figure 4.14, using only the training data, everything looks very promising for the model. All predictors are significant. The MAE is 8.1, the RMSE is 10.4, and the R-squared is 0.88.

Figure 4.14 Results of Evaluating the Model Only with the Training Data

```
Call:
lm(formula = y ~ ., data = train)

Residuals:
    Min       1Q   Median       3Q      Max
-19.555  -9.799   2.045   2.962  17.077

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.666e+02  1.867e+02  -1.964  0.0697 .
x1           7.797e+01  3.503e+01   2.226  0.0430 *
x2          -5.597e+00  2.394e+00  -2.338  0.0348 *
x3           1.868e-01  7.720e-02   2.420  0.0297 *
x4          -3.121e-03  1.270e-03  -2.458  0.0276 *
x5           2.539e-05  1.031e-05   2.462  0.0274 *
x6          -8.013e-08  3.280e-08  -2.443  0.0284 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12.75 on 14 degrees of freedom
Multiple R-squared:  0.8836,    Adjusted R-squared:  0.8337
F-statistic: 17.71 on 6 and 14 DF,  p-value: 8.404e-06

> predicted <- predict.lm(regmodel,newdata=train)
> mae<-mean(abs(train$y-predicted))
> cat("MAE",mae,"\n")
MAE 8.153177
> rmse<-(mean((train$y-predicted)**2))**.5
> cat("RMSE",rmse,"\n")
RMSE 10.41289
> rsquared=1-( sum((predicted-train$y)^2))/(sum((train$y-mean(train$y))^2))
> cat("Rsquared",rsquared,"\n")
Rsquared 0.8836104
```

Now let's try evaluating the model with the test data:

R Program for Correctly Evaluating with the Test Data:

```
predicted <- predict.lm(regmodel,newdata=test)
mae<-mean(abs(test$y-predicted))
cat("MAE",mae,"\n")
rmse<-(mean((test$y-predicted)**2))**.5
cat("RMSE",rmse,"\n")
rsquared=1-( sum((predicted-test$y)^2))/(sum((test$y-mean(test$y))^2))
cat("Rsquared",rsquared,"\n")
```

Results of Evaluating the Model with the Test Data

As can be seen in Figure 4.15, from evaluating the test data, the MAE is 25.3, the RMSE is 45.4, and the R-squared is -6.569 (Note, R-square can be negative for test data, if the model fits very poorly). When compared to the MAE of 8.1, the RMSE of 10.4, and the R-squared of 0.88 for the test data; the test data tell us the reality of our model...it is much much worse than the training data would lead us to believe.

Figure 4.14 Results of Evaluating the Model with the Test Data

```
> predicted <- predict.lm(regmodel,newdata=test)
> mae<-mean(abs(test$y-predicted))
> cat("MAE",mae,"\n")
MAE 25.27939
> rmse<-(mean((test$y-predicted)**2)**.5)
> cat("RMSE",rmse,"\n")
RMSE 45.39913
> rsquared=1-(sum((predicted-test$y)^2))/(sum((test$y-mean(test$y))^2))
> cat("Rsquared",rsquared,"\n")
Rsquared -6.569628
> |
```

Conclusions

A model's evaluation can go from promising using only the data that the model was trained on to evaluate the model, to unusable based on the correct application of test data (or cross-validation). Professionally, predictive models should not be used or trusted if they have not been validated with test data or cross-validation out of sample data. Professional data analysts must diligently report uncertainty and limitations of their analysis—actions that have even a chance of misleading can lead to horrendous decision-making mistakes (and new career paths for the offending data analysts).

Model Selection and the Rule of Parsimony

A widely accepted heuristic in selecting between modeling techniques is the Rule of Parsimony, sometimes also called Occam's Razor Rule. The Rule of Parsimony for models says use the model or technique that produces comparable results that uses the simplest technique or has the fewest inputs. For example, if deciding between a regression tree model and a Random Forest model that give approximately the same error rates, use the regression tree model because it is a simpler technique. There are a couple of reasons for this. First, simpler models have less chance for errors (fewer inputs and fewer steps means less chance for errors or impact from an information quality issue). Moreover, simpler models are easier to explain. In a business or organization, it is often easier to get buy in to using a model if details of how the model work are understood.

Module 4 Assignment

The csv file housing3 data contains SalePrice (house sale price, what we are trying to predict), and the following predictor variables: Sqft (square footage), Location (is the location “good”, yes or no), Condition (is the condition of the house “good”, yes or no), Yard (size of the yard), and NbrAvg (the average selling price for the neighborhood the house is in).

- Read the housing3.csv file into R
- Divide the data into training and testing data sets.
- Build a regression tree to predict house Sale Price from the other variables
- Plot the regression tree.
- Calculate MAE, RMSE, and R-squared for the regression tree.
- For the same training and test data sets, build a Random Forest to predict house Sales
- Calculate MAE, RMSE, and R-squared for the Random Forest.
- Discuss the performance of your two models, including which model performed the best.

Group Review Project 1

The csv file “US University Data” is a subset of the College Scorecard data found at data.gov from a few years ago. As a group, prepare a presentation that presents the most interesting discoveries from the US University data using exploratory data analysis and predictive models (regression, multiple regression, regression trees, and Random Forests). It can be assumed that there are no information quality issues in the data. However, exploratory data analysis can be used to find unexpected observations or relationships. Different predictive models can be built using different combinations of variables as predictor and dependent variables. Note, there are potentially many “correct” answers to this project, it is an open ended “what can you find” assignment. You can take different points of view in your analysis, e.g. discoveries can be of interest to universities from a marketing or retention standpoint, from a university consumer standpoint (students), from a state or region’s perspective, etc.

Included in the table are the following columns:

- Id = a unique identifier for each university
- University = name of the university
- CITY = city location of the university
- State = state location of the university
- AdmissionRate = admission rate (proportion accepted)
- PercentBusinessDegreesAwarded = proportion of degrees awarded that are business degrees
- NbrofDegreeSeekingStudents = degree seeking student enrollment
- AvgCostPerYear = average cost of attendance per year
- AvgFacultySalaryPerMonth = average faculty salary per month
- PercentFullTimeFaculty = proportion of faculty that are full time
- RetentionRate = proportion of students retained until graduation
- AvgAgeEntry = average age of entering students
- PercentFEMALE = proportion of students that are female
- PercentFirstGenStudents = proportion of students that are first generation students
- MedianHouseHoldIncome = median household income of students

Your group’s final presentation must conform to the following:

- It must be a single, cohesive and professional presentation
- At least one graph or table that presents a discovery from the data.
- At least one predictive model that analyzes the data.
- At least one meaningful contribution from data analysis using R for each member of the group.
- Assume you are making the presentations for a non-data analyst audience. Include an interpretation of what each analysis means and why it is interesting in terms of practical information in terms that are easily understood.
- An appendix containing the R programs used, labeling each member’s R program contribution.

Module 5: Introduction to Classification and Logistic Regression

Introduction to Classification

Classification involves the construction of models to predict categories or groups, as opposed to the models discussed in previous modules that focus on predicting continuous or numeric outcomes. It serves as a fundamental technique in predictive analytics, specifically within the subfields of artificial intelligence and data mining. There are a wide range of techniques available for classification tasks, all with the common objective of assigning unknown labeled data to one or more predefined classes. In classification, the response variable (also known as the dependent variable, target variable, or "y") is a categorical variable that lacks numerical interpretation. The values this variable assumes are referred to as its "levels." In contrast, the predictor variables (also known as independent variables, input variables, or "x") can encompass both numeric and categorical types.

Similar to regression models, a classification model may appear as follows:

$$y = x_1 + x_2 + \dots + x_i$$

As an example, we may build a model to help the IRS predict if someone is likely to cheat on their taxes (so that audit resources are applied more efficiently):

$$\text{Cheated} = \text{maritalStatus} + \text{refund} + \text{income} + \text{fileDate}$$

Cheated has levels "yes" or "no" and is based on the results of historical tax audits, maritalStatus and refund are categorical predictors, income and fileDate are numerical predictors.

Just as when building regression models, data must be validated by cross-validation or dividing data into test and training data sets. Error is measured on out of sample or test data. However, different metrics for error are used for classification, error rate / accuracy, confusion matrices, and receiver operating characteristic (ROC) curves are common methods

Examples of techniques used for classification include, classification trees, Random Forests, nearest neighbor method, logistic regression, naïve Bayes classifiers, artificial neural networks, and support vector machines. These techniques are used for a variety of business and science applications, such as classifying a new retail store into an existing group of stores; natural language processing; entity resolution (information quality); predicting tumor cells as benign or malignant; and categorizing news stories as finance, weather, entertainment, sports, etc.

A confusion matrix is a standard metric for evaluating the performance of a classification predictive model. Figure 5.1 shows an example confusion matrix where a model was trained to predict "yes" or "no" (for example, to predict tax fraud). In the example, 50 times the model correctly predicted "yes", 12 times the model incorrectly predicted "yes"; 13 times the model incorrectly predicted "no"; 25 times the model correctly predicted "no". The structure of a confusion matrix is labeled in Figure 5.2.

Figure 5.1 An Example Confusion Matrix

		Predicted Class	
		Class = Yes	Class = No
Actual Class	Class = Yes	50	13
	Class = No	12	25

Figure 5.2 Confusion Matrix Structure

		Predicted Class	
		Class = Yes	Class = No
Actual Class	Class = Yes	a = True Positives (TP)	b = False Negatives (FN)
	Class = No	c = False Positives (FP)	d = True Negatives (TN)

Several widely used metrics can be directly calculated from the confusion matrix.

$$Accuracy = \frac{a + d}{a + b + c + d} = \frac{TP + TN}{TP + FN + FP + TN} = \frac{50 + 25}{50 + 13 + 12 + 25} = 0.75$$

$$Error = (1 - Accuracy) = \frac{b + c}{a + b + c + d} = \frac{FN + FP}{TP + FN + FP + TN} = \frac{13 + 12}{50 + 13 + 12 + 25} = 0.25$$

$$True\ Positive\ Rate = Sensitivity = \frac{a}{a + c} = \frac{TP}{TP + FP} = \frac{50}{50 + 12} = 0.81$$

$$True\ Negative\ Rate = Specificity = \frac{d}{b + d} = \frac{TN}{FN + TN} = \frac{25}{13 + 25} = 0.66$$

The Problem of Class Imbalance

Class imbalance is the situation when a class we are attempting to predict occurs very rarely. There are numerous examples of important classification applications where class imbalance occurs, for example, identifying network attacks, detecting rare diseases, and anomaly detection. A confusion matrix and the subsequent calculation made from the confusion matrix can be ineffective in the case of class imbalance. In Figure 5.3, an elementary model that always picks “yes” no matter the input would have an accuracy of 99%, seemingly very high.

$$Accuracy = \frac{99 + 0}{99 + 0 + 1 + 0} = 0.99$$

Figure 5.3 A Confusion Matrix for a Sample of Data Exhibiting Class Imbalance

		Predicted Class	
		Class = Yes	Class = No
Actual Class	Class = Yes	99	0
	Class = No	1	0

For classification problems exhibiting class imbalance, the receiver operating characteristic (ROC) curve provides an alternative metric of model performance. The curve is a plot of one minus the specificity (false positive rate) versus sensitivity (true positive rate) using different classification thresholds for a particular model. As the ROC curve approaches the top of the graph the model is more accurate. A model that follows the diagonal line shows no improvement over a “random guess”. The performance of the ROC can be summarized with the area under the curve (AUC). An AUC of 1 indicates perfect model performance; a value of 0.5 has no improvement over random guessing. Figure 5.4 shows an example ROC curve for a promising model. Figure 5.5 shows an example ROC curve for a poorly performing model that should not be used.

In general, the confusion matrix is preferred over the ROC curve because it is a more easily understood practical explanation of model performance. However, in cases of extreme class imbalance, the ROC curve is necessary to examine true performance. ROC curves can also help choose a decision level for a classification model. For example, the typical threshold to change classes for logistic regression is at the 0.5 threshold; the ROC curve allows other thresholds to be considered to see if performance could be improved.

Figure 5.4 Example ROC Curve

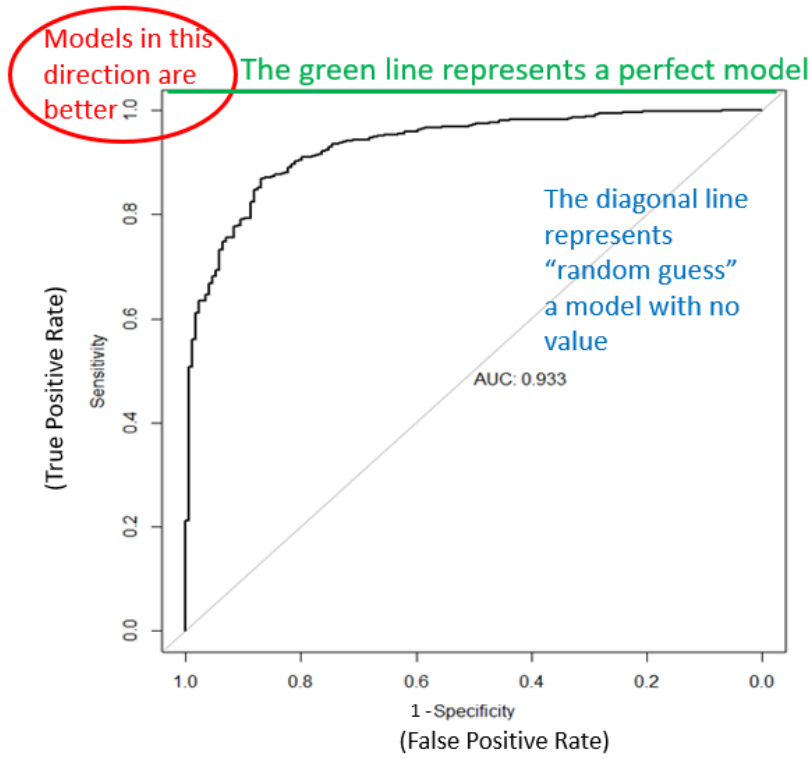
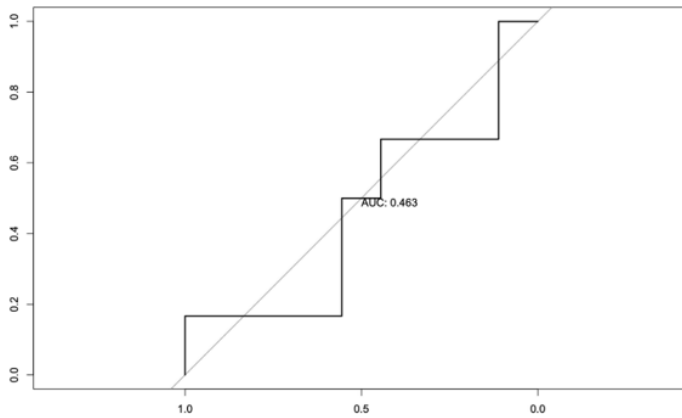


Figure 5.5 Example of a ROC Curve Showing a Poorly Performing Model



Introduction to Logistic Regression for Classification

Logistic regression is a statistical modeling technique used to analyze the relationship between a binary dependent variable and one or more independent variables. It is widely employed for classification tasks, where the goal is to predict the probability of an event or the likelihood of belonging to a specific category. Logistic regression has its historical origins in the field of statistics and was developed as an extension of linear regression. The technique was first introduced by statistician David Cox in 1958, building on the work of Karl Pearson and Ronald Fisher. The name "logistic regression" is attributed to David Cox, and it refers to the underlying mathematical function used in this regression model, which is the logistic function. (See Hosmer, Lemeshow & Sturdivant, 2013, among others, for a more detailed introduction to logistic regression and its history.)

Advantages of Logistic Regression

The advantages of logistic regression are much the same as linear and multiple regression. Logistic regression has been around for decades and thus is well-developed theoretically with plentiful documentation and many software options. Logistic regression generally increases the understanding of a problem by creating a model showing the nature of the relationship of the predictor variables to the target variable. Further, if assumptions and inferences are correctly made, uncertainty is exactly quantified through confidence intervals and tests for significance can be conducted. In addition, logistic regression is flexible for a wide range of problems. It is generally easier to explain than other classification methods such as Artificial Neural Networks (ANN) or Support Vector Machines (SVM), which may be used for similar predictive analytics problems.

Disadvantages of Logistic Regression

There are likewise disadvantages to using logistic regression as a classification technique. Although a global optimum solution is found—the optimum is for one set of assumptions about the problem, other ways of looking at the problem may result in better performance. Further, although they are flexible, they aren't appropriate for every situation. In addition, they are traditionally "supervised" techniques that require careful iterative development by the person creating the regression model. Finally, in practice they are computationally more complex than linear and multiple regression models due to a need to use approaches other than ordinary least squares.

Details of How Logistic Regression Works

Conceptually, logistic regression is the same approach as linear regression covered in Module 3. In statistical models the "link" function is a transformation on the response variable that extends statistical regression to additional types (distributions) of target data. Ordinary least squares linear regression has a link function of "1" (essentially there is no special link function needed for the linear and multiple regression models we covered in Module 3). Logistic regression assumes the response variable is binomially distributed instead of the normal distribution and as a result the logit link function is used, $\log(p/(1-p))$, this allows us to model the probability of an observation belonging to one of two

categories. After logit transformation, the method of ordinary least square linear regression could conceptually then be applied to the data. However, difficulty with this approach to logistic regression occurs when you have estimated proportions of zero. The logit function is undefined in those situations (which ends of being often in practice), so we change methods and use maximum likelihood probability estimates for the parameters. There is more than one way to do this and the mathematics is more complex (the algorithms used in analytics software such as R make numerical approximations for the maximum likelihood estimates). Methods such as the Fisher scoring method or Newton-Raphson technique are used for the estimation of logistic regression (see Schworer & Hovey, 2004, among others for more details).

For predictive modeling, logistic regression can be extended to more than binary problems (more than two categories). The multiple categories can be recoded into binary predictors and multiple logistic regression models created for each binary split. For example, if we have a target variable with levels of “RED”, “GREEN”, “BLUE”, and “YELLOW”, we would fit a logistic regression model with a target of “RED” and “not RED” (grouping the other three categories together). Then we would fit a logistic regression model for “GREEN” and “not GREEN” and so on (note, the last category would not need to be explicitly fit as it would already be accounted for in all other groupings).

R for Logistic Regression

To demonstrate logistic regression for classification in R we will begin with a classification problem to predict student success. The file “student success.csv” contains records for 2,000 students enrolled in a university’s computer science department. “FinalGrade1403” is the final grade in a first class in computing. “Transfer” is whether the student transferred into the university or not. “ACT_Score” is the student’s composite ACT (college entrance exam) score. “ACT_Math” is the student’s score on the Math portion of the ACT. “HS_GPA” is the student’s high school grade point average. “success” is the target (or response) variable, what we are trying to predict—if the student successfully completed a computer science degree within 6 years. Figure 5.6 shows the first few rows of the student success data.

Figure 5.6 First Few Rows of the “student success.csv” Data

```

Student FinalGrade1403 Transfer ACT_Score ACT_Math HS_GPA success
1      1             83      no         23        14    3.58    yes
2      2             98      yes         30        32    3.60    yes
3      3             89      no         24        21    2.77    yes
4      4             47      no         21        24    2.89    no
5      5             91      no         24        18    2.85    yes
6      6             78      no         31        31    3.52    yes
~ |

```

The data may be read in as usual in R.

R Code for Reading in Data

```

stu <- read.csv("C:/Users/hbrown11/Desktop/student success.csv", header=TRUE)
head(stu)

```

Note for classification problems we will need to identify our target variable as a factor (recall this process from Module 1).

R Code for Setting Success as a Factor

```
stu$success<-factor(stu$success)
```

The data are now ready to be divided into training and testing data sets. This process is identical to what was done in Module 4. Note, cross-validation could also alternatively be used.

R Code for Dividing Data into Training and Testing Data Sets

```
set.seed(23)
```

```
R=runif(nrow(stu)) #create a random uniform column the length of our table
```

```
stu$R=R #add the random numbers to our table
```

```
train<-stu[stu$R<=.6,] #select roughly 60% for training
```

```
test<-stu[stu$R>.6,] #the other 40% gets placed in test
```

```
train<-subset(train,select=-c(R)) #remove the column R
```

```
test<-subset(test,select=-c(R)) #remove the column R
```

The `glm` function in R is now used to estimate the logistic regression model. In this case, we cannot specify the model with “`success~.`” because there is an id column, “Student”. Addition of this student id as a predictor to the model would be nonsense (as it is an arbitrary label uniquely identifying rows in the table). For this reason, the five predictors are listed out, separated by a “+”.

R Code for Building the Logistic Regression Model

```
lreg<-glm(success~FinalGrade1403+Transfer+ACT_Score+ACT_Math+HS_GPA, data=train,  
family=binomial)
```

```
summary(lreg)
```

The fit logistic regression model with significance tests is given in Figure 5.7. We can see that the majority of model inputs are significant. Note, we will NOT remove variables that are not significant; there are much more appropriate approaches to variable selection that we will cover in Module 8. The final model can be expressed as:

$$P(\text{success}=\text{yes}) = -20.9 + 0.150*\text{FinalGrade1403} + 0.339*\text{TransferYes} + 0.228*\text{ACT_Score} + 0.244*\text{ACT_Math} + 0.638*\text{HS_GPA}$$

The model can be used to make a prediction (just as was done with a multiple regression equation). However, the result is a probability, in this example the probability that student success = “yes” (the student successfully completed a computing degree within six years). The typical classification threshold is 0.5 for logistic regression, i.e. any predicted probability greater than 0.5 would be classified as “yes”. (Not to be confused with an alpha level of 0.05 used for 95% statistical confidence.)

Figure 5.7 Parameter Estimates and Hypothesis Tests for the “student success” Logistic Regression Example

```

Coefficients:
      Estimate Std. Error z value Pr(>|z|)
(Intercept) -20.90574    1.58691 -13.174 < 2e-16 ***
FinalGrade1403  0.14982    0.01128  13.286 < 2e-16 ***
Transferyes     0.33876    0.36383   0.931  0.3518
ACT_Score       0.22768    0.03584   6.352 2.12e-10 ***
ACT_Math        0.24438    0.02672   9.147 < 2e-16 ***
HS_GPA          0.63779    0.22222   2.870  0.0041 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

The model can then be fit to the testing data and evaluated with a confusion matrix, Figure 5.8.

R Code for Evaluating the Logistic Regression Model with the Test Data

```

pmodel<- predict(lreg,newdata=test, type = "response")
model<- ifelse(pmodel > 0.5, "yes", "no") #turn the probability into a pred
known<-test["success"] #extract the actual value of success for comparing
knownt<-t(known) #transpose the data for the confusion matrix
table(knownt,model) #create the confusion matrix

```

Figure 5.8 Confusion Matrix “student success” for OOS Data

```

      model
knownt no yes
no     111  58
yes    31 633

```

Metrics that can be calculated from the confusion matrix are given in Figure 5.9.

Figure 5.9 Interpreting the Confusion Matrix

		Model	
		no	yes
Known	no	111	58
	yes	31	633

111 times the model correctly predicted a student would not be successful
633 times the model correctly predicted a student would be successful
31 times the model incorrectly predicted a student would not be successful
58 times the model incorrectly predicted a student would be successful

From a confusion matrix, we can calculate many different commonly used metrics:

Accuracy = $(111+633)/(111+58+31+633) = 0.89$ (89% accuracy rate)

Error Rate = $(31+58)/(111+58+31+633) = 0.11$ (11% error rate)

True “no” rate = $111/(111+31) = 0.78$ (78% true “positive” rate, this is also called the “sensitivity”)

True “yes” rate = $633/(58+633) = 0.92$ (92% true “negative” rate, this is also called the “specificity”)

Is this a good model? Similar to regression models, the answer depends on business context. It is a viable model (certainly better than guessing or not using a model), but whether it is “good” depends on the context, and the question, is there something better that could be used?

The model can be used to make a prediction as is demonstrated in the R code below. Note, extrapolation (using values for the predictor variables outside the range used to train the model), should still be avoided.

R Code to Make a Prediction

```
newdata=data.frame(FinalGrade1403=87,Transfer="yes",ACT_Score=26,ACT_Math=25,HS_GPA=3.9)  
predict(lreg,newdata=newdata , type = "response")
```

For this student, the model output is 0.9990741, which is greater than 0.5 and indicates the student would be predicted to be successful (success = “yes”).

Full R Program

```
stu <-read.csv("C:/Users/hbrown11/Desktop/student success.csv", header=TRUE)  
head(stu)  
stu$success<-factor(stu$success)  
set.seed(23)  
R=runif(nrow(stu)) #create a random uniform column the length of our table  
stu$R=R #add the random numbers to our table  
train<-stu[stu$R<=.6,] #select roughly 60% for training  
test<-stu[stu$R>.6,] #the other 40% gets placed in test  
train<-subset(train,select=-c(R)) #remove the column R
```

```

test<-subset(test,select=-c(R)) #remove the column R

lreg<-glm(success~FinalGrade1403+Transfer+ACT_Score+ACT_Math+HS_GPA, data=train,
family=binomial)

summary(lreg)

pmodel<- predict(lreg,newdata=test, type = "response")

model<- ifelse(pmodel > 0.5, "yes", "no") #turn the probability into a pred

known<-test["success"]

knownt<-t(known)

table(knownt,model)

newdata=data.frame(FinalGrade1403=87,Transfer="yes",ACT_Score=26,
ACT_Math=25,HS_GPA=3.9)

predict(lreg,newdata=newdata, type = "response")

```

Additional Information on Logistic Regression in R

As we have introduced in this module, logistic regression makes a binary prediction into one of two categories. In our first example, our target variable was “success”, that distinguishes if a student was successful, denoted with “yes” if the row of data belongs to a student successfully completing the computer science program within six years; or “no” for a student unsuccessful. We would say the factor “customer” has two levels, “no” and “yes”. The logistic regression algorithm in R must choose one of the two levels to be a reference level. By default, R assigns the reference level alphabetically. The level alphabetically first becomes the reference level. So, for a factor with levels, “no” and “yes”, “no” would be the reference level. Logistic regression always works by predicting the probability of the non-reference level. So, for a factor with levels “no” and “yes”, the logistic regression model would make a prediction on the probability the target variable is “yes”. Then, to turn a logistic regression probability into a category prediction, we use the probability threshold of 0.5, meaning any value greater than 0.5 would be classified as “yes”. If you wanted to change the reference level in R, the following code could be executed (this is not necessary, just a demonstration of how to change the level being modeled, if you wanted the model to predict the probability of “no” instead of “yes” in this example):

R Code for Changing the Reference Level (This Code is Optional and not a Part of the Example)

```

stu$success <- factor(stu$success, levels = c("yes", "no"))

str(stu$success)

summary(stu$success)

```

A ROC and AUC can be created in R using the following code. The resulting plot is given in Figure 5.10.

R Code for ROC and AUC for Logistic Regression

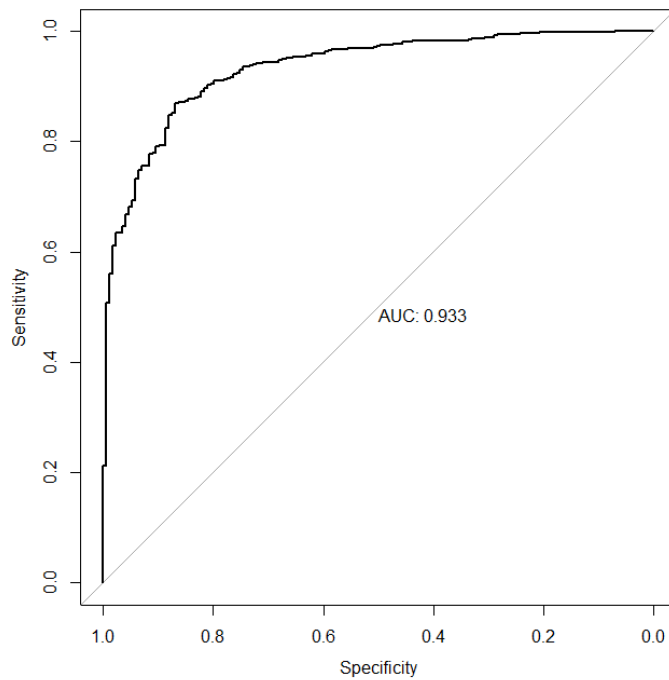
```
install.packages("pROC",dependencies=TRUE) #only needs to be ran once
```

```
library(pROC)
```

```
test_prob = predict(lreg, newdata = test, type = "response")
```

```
test_roc = roc(test$success ~ test_prob, plot = TRUE, print.auc = TRUE)
```

Figure 5.10 ROC Curve for “student success” Example



Logistic Regression Practice Exercise

The file "appt.csv" contains information regarding patients missing appointments in the healthcare setting. Patient no-show leads to inefficient resource allocation and limits access to care. A way to counteract no-show is over-booking; however, poor over-booking can lead to equally negative outcomes. To aid in the most efficient use of resources, can we accurately predict if a patient is likely to be a no-show for their appointment?

In the data set, each row represents a patient, the column " MissedAppt" indicates which patient missed an appointment ("yes"=missed). PrevMissedAppt=if the patient missed a previous appointment, SameDayAppt=was the appointment made on the same day, Age=age of patient, Male=gender of patient, White =nationality of patient, WaitTime=average wait time the day of the appointment, Monday etc. = day of week of appointment.

Figure 5.11 Contents of the File appt (First Few Rows)

```
MissedAppt PrevMissedAppt SameDayAppt Age Male White WaitTime Monday Tuesday
no          no             no      42  yes   no    46    no    no
yes         no             no      37  no    no     6    no    yes
no          no             no      32  yes   no    98    no    no
yes         no             no      53  yes   no    14    no    yes
no          no             no      15  no    yes   22    no    no
yes         no             yes     52  no    no    25    no    no
Thursday   Friday Saturday
no         yes       no
no         no        no
no         no        yes
no         no        no
no         yes       no
no         no        no
```

Use the example program for student success and complete the following (a solution will follow): Build a logistic regression model to predict MissedApt from the other columns. (Divide into training and test data sets, created a confusion matrix, calculate accuracy, create a ROC curve and AUC calculation, make a prediction that is not extrapolation).

R Code Solution

```
appt <- read.csv("C:/Users/hbrown11/Desktop/appt.csv", header=TRUE)
head(appt)
appt$MissedAppt <- factor(appt$MissedAppt)
set.seed(23)
R = runif(nrow(appt))
appt$R = R
train <- appt[appt$R <= .6,]
test <- appt[appt$R > .6,]
train <- subset(train, select = -c(R))
test <- subset(test, select = -c(R))
lreg <- glm(MissedAppt ~ ., data = train, family = binomial)
summary(lreg)
pmodel <- predict(lreg, newdata = test, type = "response")
model <- ifelse(pmodel > 0.5, "yes", "no")
known <- test["MissedAppt"]
knownt <- t(known)
table(knownt, model)
library(pROC)
test_prob = predict(lreg, newdata = test, type = "response")
test_roc = roc(test$MissedAppt ~ test_prob, plot = TRUE, print.auc = TRUE)
newdata = data.frame(PrevMissedAppt = "no", SameDayAppt = "yes", Age = 47, Male = "yes",
White = "yes", WaitTime = 30, Monday = "yes", Tuesday = "no", Thursday = "no",
Friday = "no", Saturday = "no")
predict(lreg, newdata = newdata, type = "response")
```

Figure 5.12 highlights the changes that need to be made from one logistic regression problem to another.

Figure 5.12 What Changed from Example 1 to Example 2

```
appt <- read.csv("C:/Users/hbrown11/Desktop/appt.csv", header=TRUE)
head(appt)
appt$MissedAppt <- factor(appt$MissedAppt)
set.seed(23)
R = runif(nrow(appt))
appt$R = R
train <- appt[appt$R <= .6,]
test <- appt[appt$R > .6,]
train <- subset(train, select = -c(R))
test <- subset(test, select = -c(R))
lreg <- glm(MissedAppt ~ ., data = train, family = binomial)
summary(lreg)
pmodel <- predict(lreg, newdata = test, type = "response")
model <- ifelse(pmodel > 0.5, "yes", "no")
known <- test["MissedAppt"]
knownt <- t(known)
table(knownt, model)
library(pROC)
test_prob = predict(lreg, newdata = test, type = "response")
test_roc = roc(test$MissedAppt ~ test_prob, plot = TRUE, print.auc = TRUE)
newdata = data.frame(PrevMissedAppt = "no", SameDayAppt = "yes", Age = 47, Male = "yes",
White = "yes", WaitTime = 30, Monday = "yes", Tuesday = "no", Thursday = "no",
Friday = "no", Saturday = "no")
predict(lreg, newdata = newdata, type = "response")
```

Output and Conclusions for the “appt” Example

The fit model can be seen in Figure 5.13. The final model can be written as:

$$P(\text{MissedAppt}=\text{yes}) = -0.16 + 0.096*\text{PrevMissedAppt} + 0.38*\text{SameDayAppt} + 0.030*\text{Age} + 0.38*\text{Male} + 0.16*\text{White} - 0.053*\text{WaitTime} - 0.28*\text{Monday} + 0.18*\text{Tuesday} + 0.28*\text{Thursday} - 0.037*\text{Friday} + 0.013*\text{Saturday}$$

The confusion matrix and accuracy are given in Figure 5.14; The ROC Curve and AUC are given in Figure 5.15. Based on our OOS calculations, we would expect our model to correctly predict if a patient will keep or miss their appointment 77% of the time.

Figure 5.13 The Fit Logistic Regression Model for “appt”

```

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.159874   0.306710  -0.521  0.60219
PrevMissedApptyes  0.096238   0.170333   0.565  0.57207
SameDayApptyes    0.383641   0.144954   2.647  0.00813 **
Age              0.029989   0.005051   5.938 2.89e-09 ***
Maleyes         0.377097   0.132818   2.839  0.00452 **
Whiteyes        0.160702   0.132711   1.211  0.22593
WaitTime        -0.053230   0.004133 -12.878 < 2e-16 ***
Mondayyes       -0.282056   0.230720  -1.223  0.22152
Tuesdayyes      0.176542   0.233940   0.755  0.45046
Thursdayyes     0.275984   0.230591   1.197  0.23136
Fridayyes       -0.037168   0.231035  -0.161  0.87219
Saturdayyes     0.012708   0.227605   0.056  0.95547
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Figure 5.14 The Confusion Matrix and Accuracy for the “appt” Example

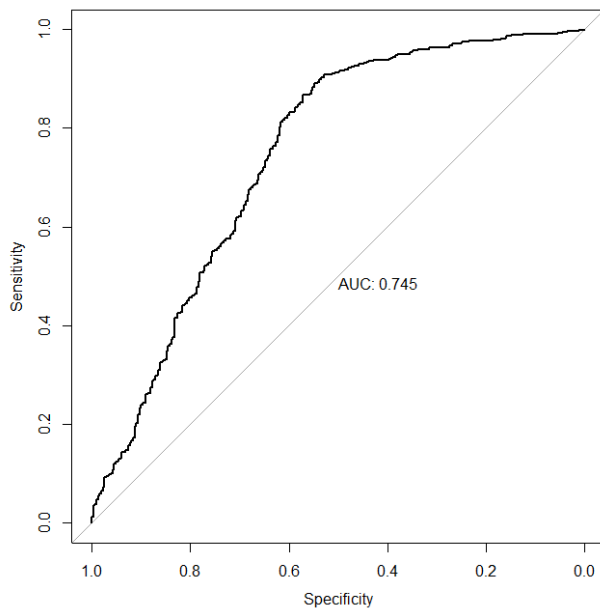
```

      model
knownt no yes
no     156 154
yes    51 535

```

Accuracy = (156+535)/(156+51+154+535)
Accuracy = 0.7712

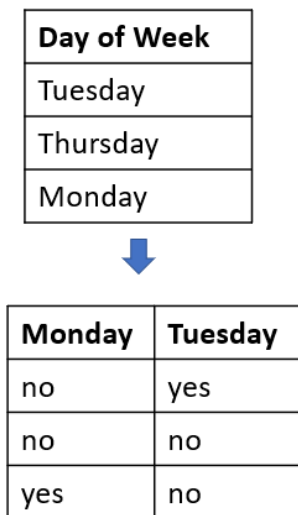
Figure 5.15 The ROC Curve and AUC for the “appt” Example



Creating Indicator Variables for Models

In modeling it is often necessary to recode categorical data (both predictor and response variables) into indicator variables. In the appt example, the original data had a column with day of week of the appointment (Monday-Saturday, the clinic was closed on Sundays). In order to be used by a model, columns with multiple categories often have to be recoded as indicator variables (instead of “yes” and “no” many software packages will require 1 and 0 for indicator variables), see Figure 5.16. Recoded there is a column for each possible day of the week, except for one day (because if all the other values are “no”s we know it must be the day that is not included); in these appointment data, Wednesday is not included, if all other days are “no”s, we know the appointment happened on a Wednesday.

Figure 5.16 Recoding Data into Indicator Variables for Models



Module 5 Assignment

The csv file titanic.csv contains data on the survivors and casualties of the 1912 titanic disaster. The data can be found online at Kaggle.com (2021).

Class refers to the type ticket that was purchased, 1, 2 or 3 (1st being the most expensive).

Age is the age in years of the passenger.

Gender is the gender of the passenger.

Survived is the target (response) variable, it is “yes” if the passenger survived and “no” if they did not.

Build a logistic regression model to predict if a passenger would have survived the titanic based on their age, gender, and class of ticket. Divide the data into training and testing data sets, construct a confusion matrix from the out of sample (test) data, calculate the accuracy of the model. Give the coefficients for final model that was built. Create a ROC curve and include a calculation of the area under the curve.

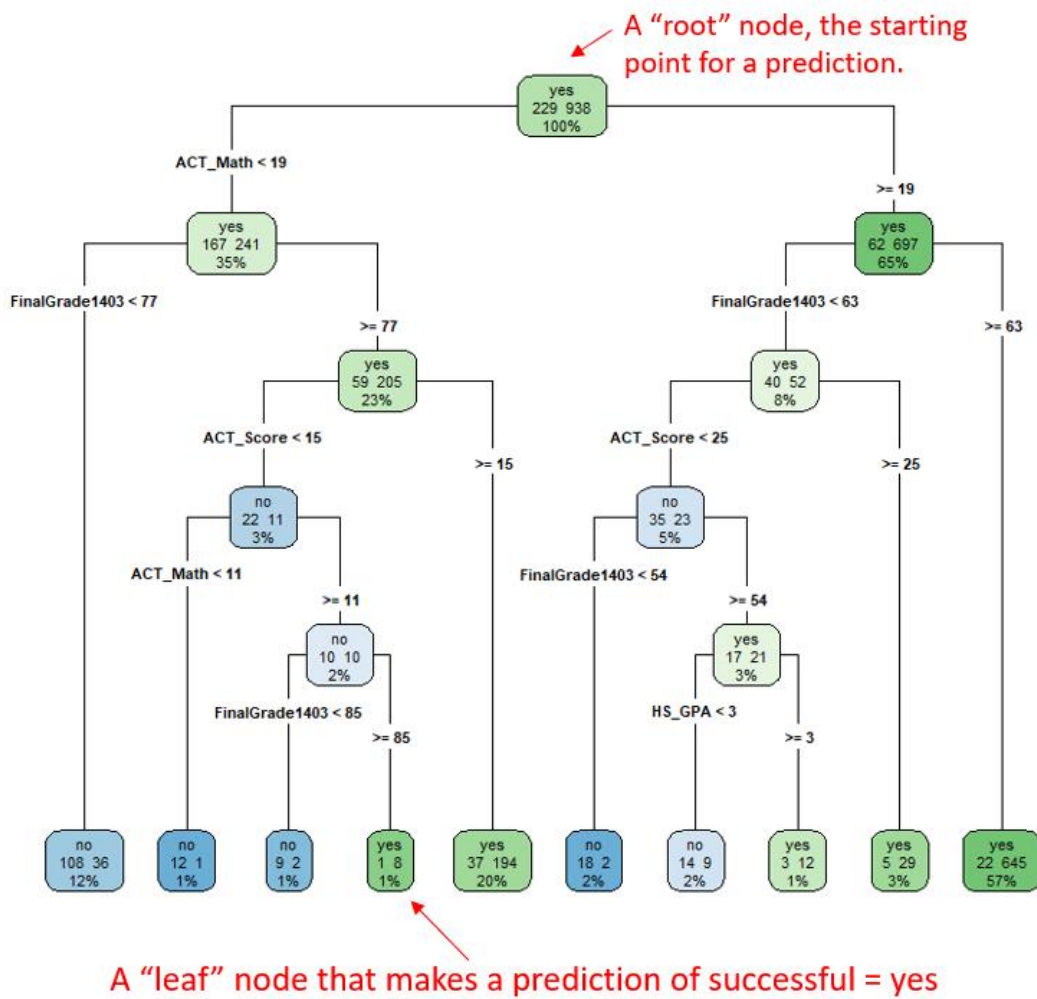
Make a prediction using your data to predict your hypothetical survival on the titanic (you can decide if you would have purchased a 1st, 2nd, or 3rd class ticket; and if you don't want to use your information, just make some up for a hypothetical person). Briefly summarize what you did in this assignment, include any key graphs, etc. As always, include your full R program.

Module 6: Introduction to Classification Trees and Random Forests for Classification

Introduction to Classification Trees

Classification trees, also known as "decision trees," are an intuitive approach to classification analogous to regression trees. When decision trees are used for classification they are called "classification trees". Similar to regression trees, classification trees create tree-like structures that can be used to make a prediction. The trees are constructed by making splits in the predictor variables that minimize error in the response variable at each node. Like regression trees, classification trees originated with Breiman, Friedman, Stone, and Olshen early 1980s.

Figure 6.1 Example Classification Tree



Advantages of Regression Trees

As a modeling technique, classification trees offer several advantages that mirror regression trees. Classification trees are more intuitive to use than many techniques. They are relatively computationally inexpensive to create and use. Further, the tree-like structures provide a visual interpretation of the model. While classification trees are known to be one of the more inaccurate approaches, to improve performance, strategies involving multiple trees can be used (e.g. Random Forests). Also, tree models handle categorical predictor variables more naturally than other techniques where categories must often be turned into indicator variables (recoded as 0s and 1s). Finally, it should be noted that classification trees are generally a better predictive modeling technique than regression trees because root nodes in classification are discrete (providing a better match for the task of classification).

Disadvantages of Regression Trees

Disadvantages to classification trees as a predictive model technique include being prone to overfitting data (for this reason “good” software that builds classification trees will have built in cross-validation, use a complexity parameter, and other techniques, such as a minimum number of values in a leaf node, to prevent overfitting). Further, for large amounts of data and a large number of predictors, a tree would become too large to be visualized. Finally, because of an inability to model complex relationships between predictor variables, unless multiple trees are combined, single classification trees do not perform as well as other techniques.

How a Classification Tree Works

An unrealistically small example is presented to help conceptualize how a classification tree is trained. For the table presented in Figure 6.2, we will build a regression tree to predict Y from X. Instead of MSE, a common measure for best split is the Gini index. The Gini is a measure of “impurity”, the effectiveness of a split in separating classes.

$$Gini = 1 - \sum_{i=1}^c (p_i)^2$$

Where C is the number of classes, and p is the proportion of target classes made by the split. The classification tree algorithm works to minimize the Gini at each split.

Figure 6.2 Small Sample of Data to Demonstrate Classification Tree Training

X	Y
6	No
4	No
8	Yes
2	No
9	Yes

Begin by sorting by X as seen in Figure 6.3.

Figure 6.3 The Data Sorted by X

X	Y
2	No
4	No
6	No
8	Yes
9	Yes

Divide the data into all possible splits using X. The first possible split could be made at $X < 3$, $X \geq 3$ (the average of the first two rows of X) (Figure 6.4). The Gini can then be calculated as:

top split, 0/1 yeses, 1/1 no's: $Gini = 1 - ((0)^2 + (1)^2) = 0$,

bottom split, 2/4 yeses, 2/4 no's: $Gini = 1 - ((0.5)^2 + (0.5)^2) = 0.5$.

The combined weighted Gini = $1/5*(0) + 4/5*(.5) = 0.4$

Thus, the Gini = 0.4 for possible split 1.

Figure 6.4 The First Possible Split

X	Y
2	No
4	No
6	No
8	Yes
9	Yes

This process is repeated for each possible split. The second possible split, split 2, is given in Figure 6.5 and would be made at $X < 5$, $X \geq 5$. The Gini can then be calculated as:

top split, 0/2 yeses, 2/2 no's: $Gini = 1 - ((0)^2 + (1)^2) = 0$,

bottom split, 2/3 yeses, 1/3 no's: $Gini = 1 - ((0.67)^2 + (0.33)^2) = 0.444$.

The combined weighted Gini = $2/5 * (0) + 3/5 * (.444) = 0.27$

Thus, the Gini = 0.27 for possible split 2.

Figure 6.5 The Second Possible Split

X	Y
2	No
4	No
6	No
8	Yes
9	Yes

The third possible split, split 3, is given in Figure 6.6 and would be made at $X < 7$, $X \geq 7$. The Gini can then be calculated as:

top split, 0/3 yeses, 3/3 no's: $Gini = 1 - ((0)^2 + (1)^2) = 0$,

bottom split, 2/2 yeses, 0/2 no's: $Gini = 1 - ((0)^2 + (1)^2) = 0$.

The combined weighted Gini = $3/5 * (0) + 2/5 * (0) = 0$

Thus, the Gini = 0 for possible split 3.

Figure 6.6 The Third Possible Split

X	Y
2	No
4	No
6	No
8	Yes
9	Yes

The fourth and final possible split, split 4, is given in Figure 6.7 and would be made at $X < 8.5$, $X \geq 8.5$. The Gini can then be calculated as:

top split, 1/4 yeses, 3/4 no's: $Gini = 1 - ((.25)^2 + (.75)^2) = 0.375$,

bottom split, 1/1 yeses, 0/0 no's: $Gini = 1 - ((0)^2 + (1)^2) = 0$.

The combined weighted Gini = $4/5 * (0.375) + 1/5 * (0) = 0.3$

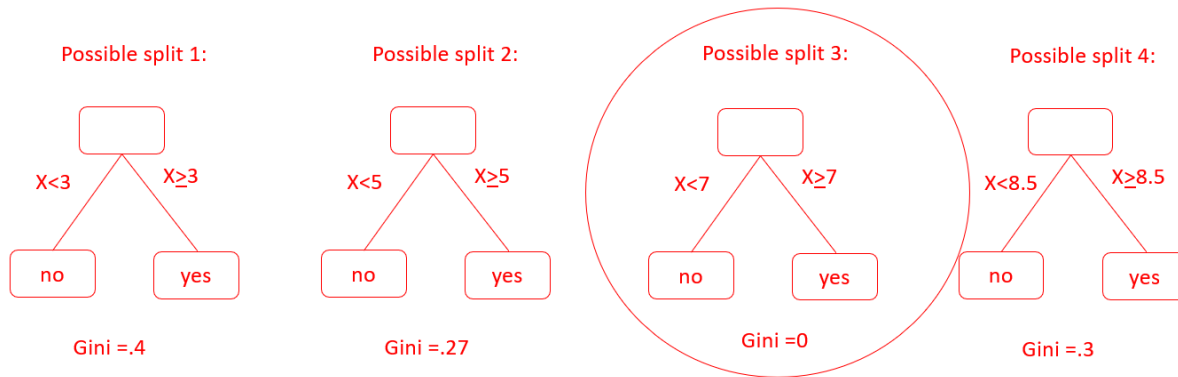
Thus, the Gini = 0.3 for possible split 3.

Figure 6.7 The Fourth Possible Split

X	Y
2	No
4	No
6	No
8	Yes
9	Yes

Split 3 is chosen, as seen in Figure 6.8, because it minimizes the Gini index.

Figure 6.8 Selecting a Split



In the selected tree if $X < 7$, then the classification tree predicts Y will be “no”; if $X \geq 7$, the regression tree predicts Y will be “yes”.

If we had more rows of data, we would consider a second split for each of our nodes. The process we just completed would be repeated for just the subset of data at each node. If we have multiple predictors, the same process is repeated for each predictor, the predictor minimizing the combined Gini index is used at each node.

Overfitting is also problem for classification trees. The Gini can be made 0 by continuing the tree until each node only has one value, but the resulting tree is likely unrealistically fit to the training data. Good tree algorithms (such as the one used in R) will use built-in cross-validation to prevent overfitting. Similar to regression trees, other settings, such as a minimum number of values in each leaf node, are also used to prevent overfitting.

Classification Trees in R

The student success and appointment examples will be repeated so that the results of the models can be compared. The file “student success.csv” contains records for 2,000 students enrolled in a university’s computer science department. “FinalGrade1403” is the final grade in a first class in computing. “Transfer” is whether the student transferred into the university or not. “ACT_Score” is the student’s composite ACT (college entrance exam) score. “ACT_Math” is the student’s score on the Math portion of the ACT. “HS_GPA” is the student’s high school grade point average. “success” is the target (or response) variable, what we are trying to predict—if the student successfully completed a computer science degree within 6 years. Figure 6.9 shows the first few rows of the student success data.

Figure 6.9 First Few Rows of the “student success” Data

```
Student FinalGrade1403 Transfer ACT_Score ACT_Math HS_GPA success
1      1             83      no         23      14    3.58    yes
2      2             98      yes        30      32    3.60    yes
3      3             89      no         24      21    2.77    yes
4      4             47      no         21      24    2.89    no
5      5             91      no         24      18    2.85    yes
6      6             78      no         31      31    3.52    yes
```

The data may be read in as usual in R.

R Code for Reading in Data

```
stu <-read.csv("C:/Users/hbrown11/Desktop/student success.csv", header=TRUE)
```

```
head(stu)
```

Note for classification problems we will need to identify our target variable as a factor (recall this process from Module 5).

R Code for Setting “success” as a Factor

```
stu$success<-factor(stu$success)
```

The data are now ready to be divided into training and testing data sets. This process is identical to what was done in Module 5. Note, cross-validation could also alternatively be used.

R Code for Dividing Data into Training and Testing Data Sets

```
set.seed(23)

R=runif(nrow(stu)) #create a random uniform column the length of our table

stu$R=R #add the random numbers to our table

train<-stu[stu$R<=.6,] #select roughly 60% for training

test<-stu[stu$R>.6,] #the other 40% gets placed in test

train<-subset(train,select=-c(R)) #remove the column R

test<-subset(test,select=-c(R)) #remove the column R
```

We are now ready to train the classification tree from the training data. Note, the packages `rpart` and `rpart.plot` may already be installed from working with regression trees in Module 4. In this case, we cannot specify the model with “`success~.`” because there is an id column, “Student”. Addition of this student id as a predictor to the model would be nonsense, as it is an arbitrary label uniquely identifying rows in the table. For this reason, the five predictors are listed out, separated by a “+”.

R Code for Building the Classification Tree

```
#install.packages('rpart', dependencies = TRUE) #this should be done from earlier this semester

#install.packages('rpart.plot', dependencies = TRUE) #this should be done from earlier this semester

library(rpart)

library(rpart.plot)

ctree<-rpart(success~FinalGrade1403+Transfer+ACT_Score+ACT_Math+HS_GPA,data=train)

#summary(ctree)

rpart.plot(ctree, type = 4, extra = 101)
```

The fit tree is given in Figure 6.10

Figure 6.10 Classification Tree for “student success” Example

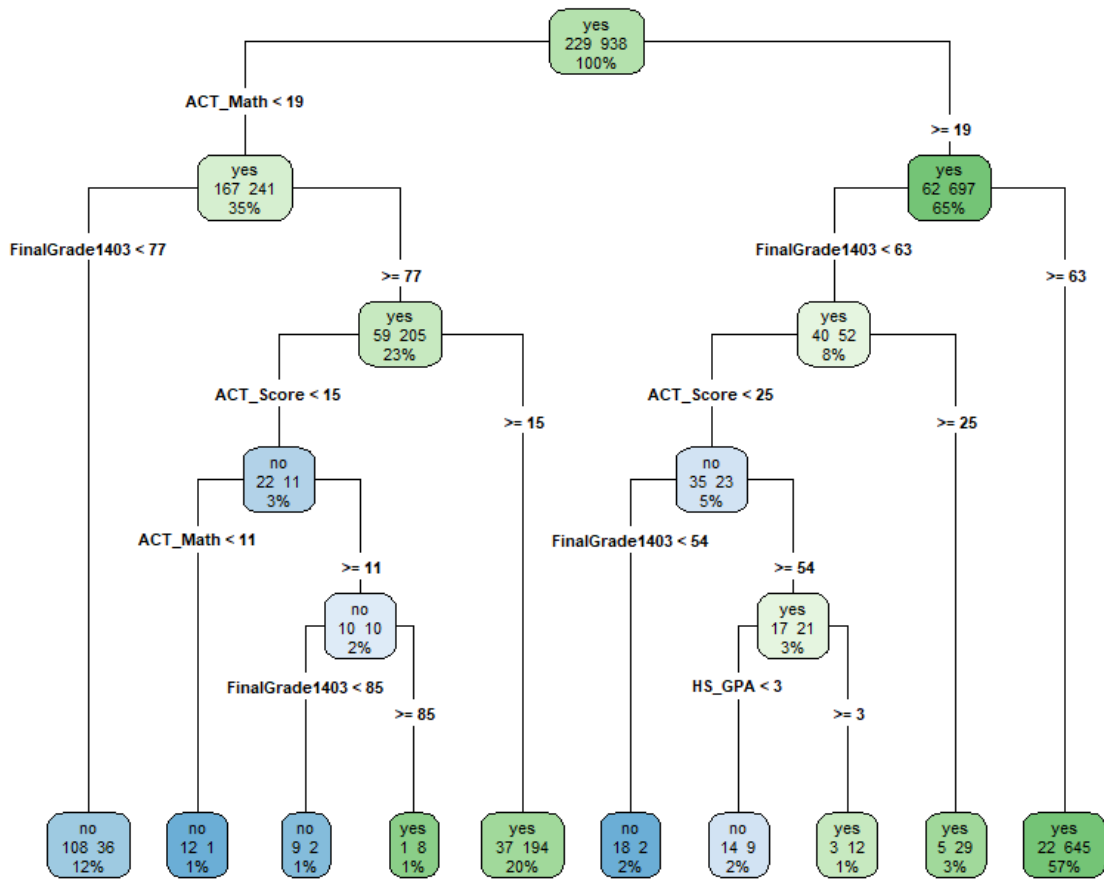
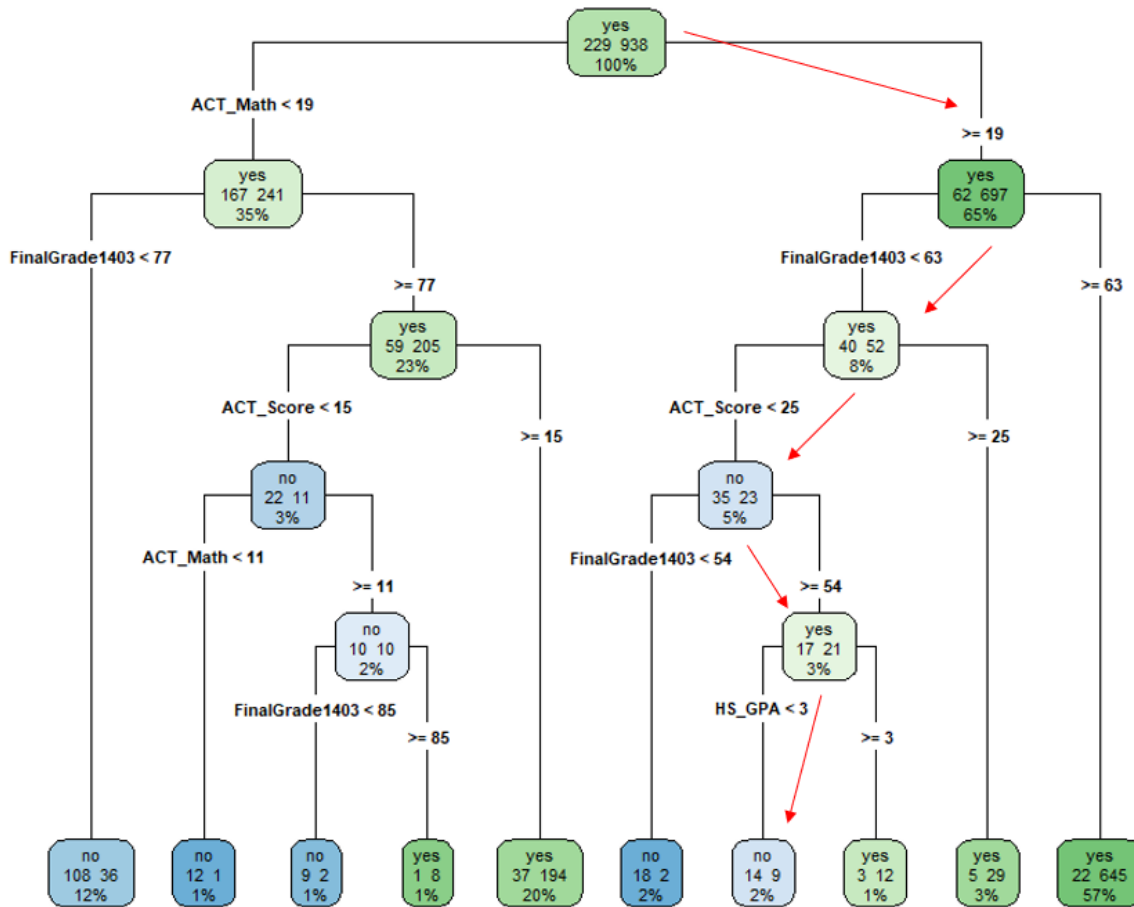


Figure 6.11 demonstrates how to use the model for a student with an ACT math score of 24, Final Grade in COMS1403 of 59, overall ACT composite score of 23, and high school GPA of 2.7. This student would be predicted to not successfully complete a computer science degree.

Figure 6.11 Example Using A Classification Tree to Make a Prediction



The classification tree can be applied to the test data to create the confusion matrix in Figure 6.12.

R Code for Applying the Model to the Test Data

```

model<- predict(ctree, newdata=test, type = "class")
known<-test["success"] #extract the actual value of success for comparing
knownt<-t(known) #transpose the data for the confusion matrix
table(knownt,model) #create the confusion matrix
  
```

Figure 6.12 Confusion Matrix for Classification Tree to Predict Student Success

```

model
knownt no yes
no 102 67
yes 56 608
  
```

Metrics that can be calculated from the confusion matrix are given in Figure 6.13.

Figure 6.13 Interpreting the Confusion Matrix

		Model	
		no	yes
Known	no	102	67
	yes	56	608

102 times the model correctly predicted a student would not be successful
608 times the model correctly predicted a student would be successful
56 times the model incorrectly predicted a student would not be successful
67 times the model incorrectly predicted a student would be successful

From a confusion matrix, we can calculate many different commonly used metrics:

Accuracy = $(102+608)/(102+56+67+608) = 0.85$ (85% accuracy rate)

Error Rate = $(56+67)/(102+56+67+608) = 0.15$ (15% error rate)

True “no” rate = $102/(102+56) = 0.65$ (65% true “positive” rate, this is also called the “sensitivity”)

True “yes” rate = $608/(67+608) = 0.90$ (90% true “negative” rate, this is also called the “specificity”)

Is this a good model? Again, the answer depends on business context. However, we now have the context of the logistic regression model to better decide; the logistic regression model with a 89% accuracy rate would appear to be a better choice for these data, unless the ease of using tree for prediction was needed.

Full R Program for Classification Tree Example 1

```
stu <- read.csv("C:/Users/hbrown11/Desktop/student success.csv", header=TRUE)
head(stu)
stu$success <- factor(stu$success)
set.seed(23)
R = runif(nrow(stu))
stu$R = R
train <- stu[stu$R <= .6,]
test <- stu[stu$R > .6,]
train <- subset(train, select = -c(R))
test <- subset(test, select = -c(R))
#install.packages('rpart', dependencies = TRUE) #this should be done from earlier this semester
#install.packages('rpart.plot', dependencies = TRUE) #this should be done from earlier this semester
library(rpart)
library(rpart.plot)
```

```
ctree<-rpart(success~FinalGrade1403+Transfer+ACT_Score+ACT_Math+HS_GPA,data=train)
rpart.plot(ctree, type = 4, extra = 101)
model<- predict(ctree, newdata=test, type = "class")
known<-test["success"]
knownt<-t(known)
table(knownt,model)
```

Random Forests

Random Forests for classification are very similar to Random Forests for regression. Recall, Random Forests are a high performing ensemble machine learning method that combines the predictions of multiple decision trees to improve the accuracy and robustness of the predictions. (Ensemble methods combine the results from multiple models.) It was first introduced by Leo Breiman in 2001 as an extension of decision trees. Random Forests have gained significant popularity and become a widely used predictive modeling technique.

Random Forests work by creating an ensemble of decision trees, each trained on a random subset of the training data and using a random subset of the features (predictor variables). During the prediction phase, each tree in the forest independently generates a prediction, and the final prediction is determined by aggregating the individual predictions through consensus voting for classification Random Forests. (Each classification tree gets a “vote” on the category, the category with the most votes is the category predicted by the Random Forest.)

Leo Breiman's introduced Random Forests in his paper titled "Random Forests" published in 2001. In this paper, Breiman introduced the concept of Random Forests and highlighted their ability to reduce overfitting, handle high-dimensional data, and provide reliable predictions. The paper demonstrated the effectiveness of Random Forests through various experiments and comparisons with other classification algorithms. Since then, there has been a widespread adoption and diverse applications of Random Forest.

Random Forests address many of the issues of a single classification tree. When constructing a classification tree, there are potentially numerous ways to split trees, and no one way of creating a tree guarantees an optimal tree (small changes can result in drastically different trees being created). Further, determining all possible trees for a large classification problem is unfeasible. In a solution to these issues, Random Forests, create numerous “random” trees (each is potentially a “good” tree by using criteria such as the Gini index), each tree makes a prediction for a particular set of predictors (that prediction is treated like a vote), the category that was voted for the most by the trees is used as the final prediction.

Advantages of Random Forests

Random Forests have several advantages as a method of predictive modeling. Random Forests are intuitive since they are based on classification trees. However, they can handle much more complexity than a single classification tree. Like classification trees, Random Forest handles categorical inputs more naturally than other techniques. For some problems, Random Forests can outperform many other popular machine learning and statistical techniques. They are considered one of the better modern machine learning techniques. Random Forests have the advantage of producing variable importance measures that can be useful for feature selection or other decisions (see Module 8). As we saw in Module 4, Random Forests can be used for regression problems or classification problems (both are called “Random Forests”).

Disadvantages of Random Forests

Random Forests also have several disadvantages as a method of predictive modeling. Random Forests are a “black-box” technique. Since numerous trees are created, there is not a single tree plot that can be visualized, and this desirable quality of classification trees is lost. For large amounts of data and large numbers of predictors, Random Forest become computationally expensive. Although there are suggestion for how many variables to consider, Random Forests can be sensitive to the “mtry” option. Despite these limitations, Random Forests are a popular and effective approach for predictive modeling.

Details of How a Random Forest Algorithm Works

In a Random Forest, the classification tree algorithm (as presented earlier in this module) is used to build a large number of trees, each tree will look at some subset of the predictor variables (all variables are not used at one time). The default value for R is to use the floor of the square root of the total number of predictors as the number of predictors to include per tree (i.e. take the square root of the number of predictors and truncate it to the nearest integer). Notice this is different than the default values for Random Forests for regression. This can be changed with the `mtry` option. 500 trees are created by default in R (this can be changed with the `ntrees` option). To make a prediction, specific values of the predictors will be input into the 500 trees. A final prediction will be made by using each prediction made by the individual trees as a “vote”, the category receiving the majority vote is used as the final prediction for the Random Forest.

Random Forests in R

We will use the student success data again. The file “student success.csv” contains records for 2,000 students enrolled in a university’s computer science department. “FinalGrade1403” is the final grade in a first class in computing. “Transfer” is whether the student transferred into the university or not. “ACT_Score” is the student’s composite ACT (college entrance exam) score. “ACT_Math” is the student’s score on the Math portion of the ACT. “HS_GPA” is the student’s high school grade point average. “success” is the target (or response) variable, what we are trying to predict—if the student

successfully completed a computer science degree within 6 years. Figure 6.14 shows the first few rows of the student success data.

Figure 6.14 First Few Rows of the “student success.csv” Data

```
Student FinalGrade1403 Transfer ACT_Score ACT_Math HS_GPA success
1      1             83      no         23      14    3.58    yes
2      2             98      yes        30      32    3.60    yes
3      3             89      no         24      21    2.77    yes
4      4             47      no         21      24    2.89    no
5      5             91      no         24      18    2.85    yes
6      6             78      no         31      31    3.52    yes
```

The data may be read in as usual in R.

R Code for Reading in Data

```
stu <- read.csv("C:/Users/hbrown11/Desktop/student success.csv", header=TRUE)
```

```
head(stu)
```

Again, for classification problems we will need to identify our target variable as a factor.

R Code for Setting “success” as a Factor

```
stu$success <- factor(stu$success)
```

The data are now ready to be divided into training and testing data sets. This process is identical to what was done in previous examples. Note, cross-validation could also alternatively be used.

R Code for Dividing Data into Training and Testing Data Sets

```
set.seed(23)
```

```
R = runif(nrow(stu)) #create a random uniform column the length of our table
```

```
stu$R = R #add the random numbers to our table
```

```
train <- stu[stu$R <= .6,] #select roughly 60% for training
```

```
test <- stu[stu$R > .6,] #the other 40% gets placed in test
```

```
train <- subset(train, select = -c(R)) #remove the column R
```

```
test <- subset(test, select = -c(R)) #remove the column R
```

We are now ready to build the Random Forest from the training data. The **randomForest** package may already be installed from working with regression trees in Module 4. In this case, we cannot specify the model with “**success**~.” because there is an id column, “Student”. Adding this student id as a predictor

to the model would nonsense (as it is an arbitrary label uniquely identifying rows in the table). For this reason, the five predictors are listed out, separated by a “+”.

R Code for Creating the Random Forest

```
#install.packages('randomForest', dependencies = TRUE)

library(randomForest)

set.seed(23) #needed to make the RF repeatable

RFM=randomForest(success~FinalGrade1403+Transfer+ACT_Score+ACT_Math+HS_GPA,data=train)

#summary(RFM)

model<- predict(RFM, newdata=test, type = "class")

known<-test["success"] #extract the actual value of success for comparing

knownt<-t(known) #transpose the data for the confusion matrix

table(knownt,model) #create the confusion matrix
```

Figure 6.15 Resulting Confusion Matrix

```
      model
knownt no  yes
no     105  64
yes    47  617
```

Efforts can again be made to try and fine tune the model for different values of mtry and ntrees. After a few attempts, mtry of 4 and ntrees=1000, resulted in slightly better results.

R Code After Fine Tuning the Random Forest

```
set.seed(23)

RFM=randomForest(success~FinalGrade1403+Transfer+ACT_Score+ACT_Math+HS_GPA,data=train,
mtry=4,ntrees=1000)

model<- predict(RFM, newdata=test, type = "class")

known<-test["success"] #extract the actual value of success for comparing

knownt<-t(known) #transpose the data for the confusion matrix

table(knownt,model) #create the confusion matrix

table(knownt,model) #create the confusion matrix
```


Figure 6.16 Resulting Confusion Matrix

```
      model
knownt no yes
no     110 59
yes    51 613
```

Figure 6.17 Interpreting the Confusion Matrix

		Model	
		no	yes
Known	no	110	59
	yes	51	613

110 times the model correctly predicted a student would not be successful
613 times the model correctly predicted a student would be successful
51 times the model incorrectly predicted a student would not be successful
59 times the model incorrectly predicted a student would be successful

From a confusion matrix, we can calculate many different commonly used metrics:

Accuracy = $(110+613)/(110+51+59+613) = 0.87$ (87% accuracy rate)

Error Rate = $(51+59)/(102+56+67+608) = 0.13$ (13% error rate)

True “no” rate = $110/(110+51) = 0.64$ (64% true “positive” rate, this is also called the “sensitivity”)

True “yes” rate = $613/(59+613) = 0.91$ (91% true “negative” rate, this is also called the “specificity”)

We now have three models we can compare (Figure 6.18). Based on accuracy (and the overall confusion matrix error calculations), logistic regression would be preferred. However, all three models are not far apart in performance if there was a particular reason to use one of the others.

Figure 6.18 Comparing the Results of the Logistic Regression, Classification, and Random Forest Models

Logistic Regression

Accuracy=89%

```
      model
knownt no yes
no     111 58
yes    31 633
```

Classification tree

Accuracy=85%

```
      model
knownt no yes
no     102 67
yes    56 608
```

Random forest

Accuracy=87%

```
      model
knownt no yes
no     110 59
yes    51 613
```

Full Random Forest Code

```
stu <-read.csv("C:/Users/hbrown11/Desktop/student success.csv", header=TRUE)
stu$success<-factor(stu$success)
set.seed(23)
R=runif(nrow(stu)) #create a random uniform column the length of our table
stu$R=R
train<-stu[stu$R<=.6,]
test<-stu[stu$R>.6,]
train<-subset(train,select=-c(R))
test<-subset(test,select=-c(R))
#install.packages('randomForest', dependencies = TRUE)
library(randomForest)
set.seed(23)
RFM=randomForest(success~FinalGrade1403+Transfer+ACT_Score+ACT_Math+HS_GPA,data=train,
mtry=4,ntrees=1000)
model<- predict(RFM, newdata=test, type = "class")
known<-test["success"]
knownt<-t(known)
table(knownt,model)
```

Classification Tree and Random Forest Practice Exercise

The file "appt.csv" contains information regarding patients missing appointments in the healthcare setting. Patient no-show leads to inefficient resource allocation and limits access to care. A way to counteract no-show is over-booking; however, poor over-booking can lead to equally negative outcomes. To aid in the most efficient use of resources, can we accurately predict if a patient is likely to be a no-show for their appointment?

In the data set, each row represents a patient, the column " MissedAppt" indicates which patient missed an appointment ("yes"=missed). PrevMissedAppt=if the patient missed a previous appointment, SameDayAppt=was the appointment made on the same day, Age=age of patient, Male=gender of patient, White =nationality of patient, WaitTime=average wait time the day of the appointment, Monday etc. = day of week of appointment.

Figure 6.19 Contents of the File appt (First Few Rows)

```
MissedAppt PrevMissedAppt SameDayAppt Age Male White WaitTime Monday Tuesday
no          no             no      42  yes   no    46    no    no
yes         no             no      37  no    no     6    no    yes
no          no             no      32  yes   no    98    no    no
yes         no             no      53  yes   no    14    no    yes
no          no             no      15  no    yes   22    no    no
yes         no             yes     52  no    no    25    no    no
Thursday   Friday Saturday
no          yes       no
no          no        no
no          no        yes
no          no        no
no          yes       no
no          no        no
```

Build a classification tree model and a Random Forest to predict MissedAppt from the other columns. (Divide into training and test data sets, created a confusion matrix, calculate accuracy, and compare the results of these two models to the logistic regression model).

R code for the Classification Tree

```
appt <- read.csv("C:/Users/hbrown11/Desktop/appt.csv", header=TRUE)
head(appt)
appt$MissedAppt <- factor(appt$MissedAppt)
set.seed(23)
R = runif(nrow(appt))
appt$R = R
train <- appt[appt$R <= .6,]
test <- appt[appt$R > .6,]
train <- subset(train, select = -c(R))
test <- subset(test, select = -c(R))
#install.packages('rpart', dependencies = TRUE) #install.packages('rpart.plot', dependencies = TRUE)
library(rpart)
library(rpart.plot)
ctree <- rpart(MissedAppt ~ ., data = train)
rpart.plot(ctree, type = 4, extra = 101)
model <- predict(ctree, newdata = test, type = "class")
known <- test["MissedAppt"]
knownt <- t(known)
table(knownt, model)
```

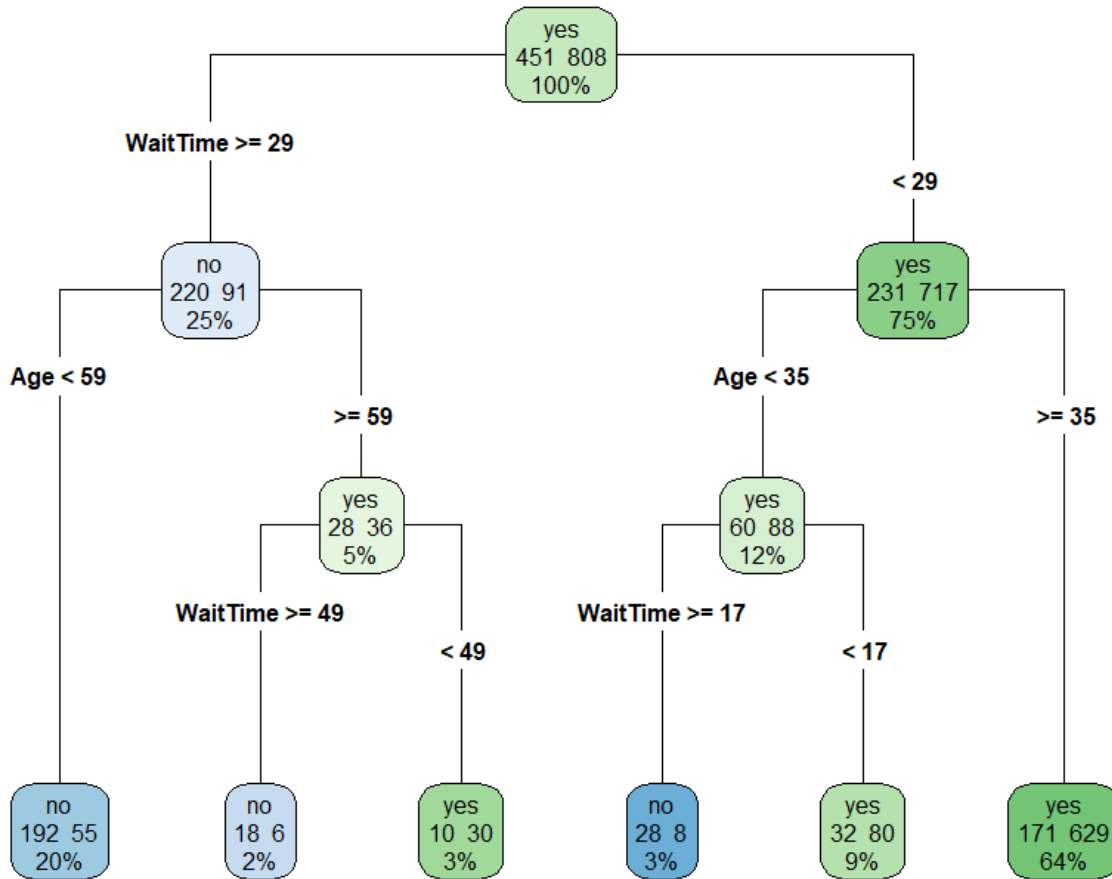
6.20 Confusion Matrix for the Classification Tree (“appt” Example)

```
      model
knownt no yes
no     156 154
yes    63 523
```

Accuracy of the classification tree for this example was 76%. Using the tree in Figure 6.21, we can predict the following patient: PrevMissedAppt=no, SameDayApp=no, Age=42, Male=yes, White=no, WaitTime=46, Monday=no, Tuesday=no, Thursday=no, Friday=yes, Saturday=no would be predicted to

not miss the appointment. (We would follow the tree to the left twice, using the wait time and age columns, to predict “no”, the appointment would not be missed)

Figure 6.21 The Classification Tree



Assuming we just completed the above code to create a classification tree, we can add the following code to add the Random Forest.

R code to Add the Random Forest

```

#install.packages('randomForest', dependencies = TRUE)

library(randomForest)

set.seed(23)

RFM=randomForest(MissedAppt~.,data=train)

model<- predict(RFM, newdata=test, type = "class")
  
```

```
known<-test["MissedAppt"]
knownt<-t(known)
table(knownt,model)
```

Adjusting the values of mtry and ntrees did not result in improvement this time. The resulting confusion matrix is given in Figure 6.22.

6.22 Confusion Matrix for the Random Forest ("appt" Example)

```
      model
knownt no yes
no    157 153
yes   50 536
```

Accuracy of Random Forest for this example was 77%. The results of the logistic regression (Module 5), classification tree, and Random Forest can now be compared (see Figure 6.23)

Figure 6.23 Comparing the Results of the Logistic Regression, Classification, and Random Forest Models ("app" Example)

Logistic Regression

Accuracy=77%

```
      model
knownt no yes
no    156 154
yes   51 535
```

Classification tree

Accuracy=76%

```
      model
knownt no yes
no    156 154
yes   63 523
```

Random forest

Accuracy=77%

```
      model
knownt no yes
no    157 153
yes   50 536
```

Based on accuracy (and the overall confusion matrix error calculations), all three models were very similar. Any of the three could be legitimately used.

R code for Making a Prediction

#create a new row of data to predict:

```
newdata=data.frame(PrevMissedAppt="no",SameDayAppt="yes", Age=47, Male="yes", White="yes",
WaitTime=30, Monday="yes", Tuesday="no", Thursday="no",Friday="no", Saturday="no")
```

```
#predict with the classification tree  
predict(ctree,newdata=newdata, type = "class")  
#predict with the Random Forest  
predict(RFM,newdata=newdata, type = "class")
```

Both models predict “no”, the patient will not miss the appointment, for this input.

Module 6 Assignment

The csv file titanic.csv contains data on the survivors and casualties of the 1912 titanic disaster. The data can be found online at Kaggle.com (2021).

Class refers to the type ticket that was purchased, 1, 2 or 3 (1st being the most expensive).

Age is the age in years of the passenger.

Gender is the gender of the passenger.

Survived is the target (response) variable, it is “yes” if the passenger survived and “no” if they did not.

Build a classification tree model and a Random Forest model to predict if a passenger would have survived the titanic based on their age, gender, and class of ticket. Divide the data into training and testing data sets, construct a confusion matrix from the out of sample (test) data, calculate the accuracy of the model. Include an image of the classification tree for the final model. Compare the confusion matrix and accuracy between the logistic regression model from last week, the classification tree, and the Random Forest model.

Using either the tree or Random Forest model, make a prediction using your data to predict your hypothetical survival on the titanic (you can decide if you would have purchased a 1st, 2nd, or 3rd class ticket; and if you don't want to use your information, just make some up for a hypothetical person). Briefly summarize what you did in this assignment, include any key graphs, etc. As always, include your full R program.

Module 7: Introduction to Artificial Neural Networks for Classification

Introduction to Artificial Neural Networks

Artificial Neural Networks (ANN) are an artificial intelligence machine learning technique widely used for classification. As with all predictive modeling techniques, to create an ANN we must have training data with inputs and known outputs to construct the model and test or validation data are important to avoid over-fitting. Once an ANN is constructed, predictions for outputs based on a new set of inputs can be made. There are many different types of ANN that implement different topologies or strategies to improve the predictive ability of the ANN or adjust it to a particular type of problem. Examples of different types of ANN include, but are not limited to:

- Feedforward neural networks: a single input layer, a single output layer, and one or more hidden layers, used for classification and regression tasks.
- Recurrent neural networks: process sequential data, used for natural language processing and speech recognition
- Transformer neural networks: used for natural language processing, e.g. ChatGPT
- Convolutional neural networks: designed for image processing, used for tasks such as image classification and object detection.
- Autoencoders: dimensionality reduction and feature extraction, used for tasks such as image compression and anomaly detection.
- Generative adversarial networks: generate new data, used for tasks such as image generation and text generation.
- Deep learning: ANN with many hidden layers and parameters

ANN are based on biological neural networks. A biological neural network is a model of reasoning of the physical human brain. The brain consists of a densely interconnected set of nerve cells, or basic information-processing units, called neurons. The human brain incorporates nearly 10 billion neurons and 60 trillion connections, synapses, between them. Each neuron has a very simple structure, but collectively the elements constitute an enormous processing power, referred to as a neural network.

Artificial neural networks (ANNs) have a long history that traces back to the 1940s when the concept of a computational model inspired by the human brain first emerged. In 1943, Warren McCulloch and Walter Pitts proposed a model of artificial neurons, laying the foundation for ANNs. However, progress was slow until the 1950s and 1960s when researchers like Frank Rosenblatt introduced the perceptron, an early form of a neural network capable of learning. Despite initial enthusiasm, interest waned as perceptrons faced limitations in solving complex problems. The ANN approach experienced a resurgence in the 1980s with the development of backpropagation, a technique for training multilayer neural networks. This breakthrough paved the way for more sophisticated architectures, such as convolutional neural networks (CNNs) in the 1990s and recurrent neural networks (RNNs) in the 2000s. Recent advancements in computing power, big data, and algorithmic improvements have led to significant breakthroughs in deep learning, enabling ANNs to achieve a high level of performance in various domains, including large language models such as ChatGPT in 2023. (For more general information on ANNs and their history see Goodfellow, Bengio & Courville, 2016.)

Advantages of ANN

Artificial neural networks have several advantages as a method of predictive modeling. ANN have been demonstrated to be useful for a variety of problems. They can handle complex problems, require little supervision, and can be combined with other techniques (for example, genetic algorithms). Further, currently ANN significantly outperform other modeling approaches for certain types of problems, such as natural language processing.

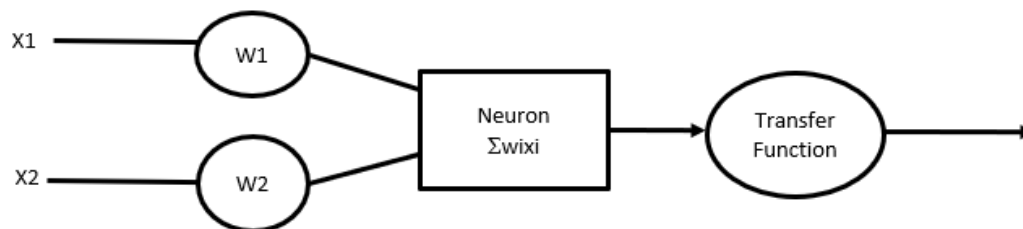
Disadvantages of ANN

ANN also have several disadvantages as a method of predictive modeling. Like Random Forests, ANN are a “black-box” technique that can be difficult to explain to business decision makers. For a large ANN, specifically how a prediction is made can be difficult to ascertain. In addition, ANN do not guarantee optimal solutions and require fine tuning for which there is often little guidance. Further, because of the enthusiasm over new ANN techniques, artificial neural networks can be overused when simpler (and perhaps equally or more effective) techniques could be used. Despite these limitations, modern ANN are arguably one of the most promising approaches to building models for classification.

Details of How Artificial Neural Networks Work

A neural network is a computing model that has many units working together. The connections between units have weights that are trained by a learning algorithm. The behavior of an ANN is determined by its topology and the nature of the individual units. To conceptually understand how ANN are trained, we will look at how an ANN works for an unrealistically small and simple example. In this example, there will be two inputs, X1 and X2; weights, W1 and W2, that are trained for these inputs, a neuron that combines the weights, and a transfer function that scales the output of the neuron to give a predicted value of the output (Y) (see Figure 7.1). A common transfer function for classification is the Sigmoid transfer function, $Y_T = 1/(1+e^{-y})$. However, in our first example we will simply use rounding as our transfer function to give our output as a 0 or 1.

Figure 7.1 Processing Information in an ANN



Training an ANN to Recognize the OR Operator

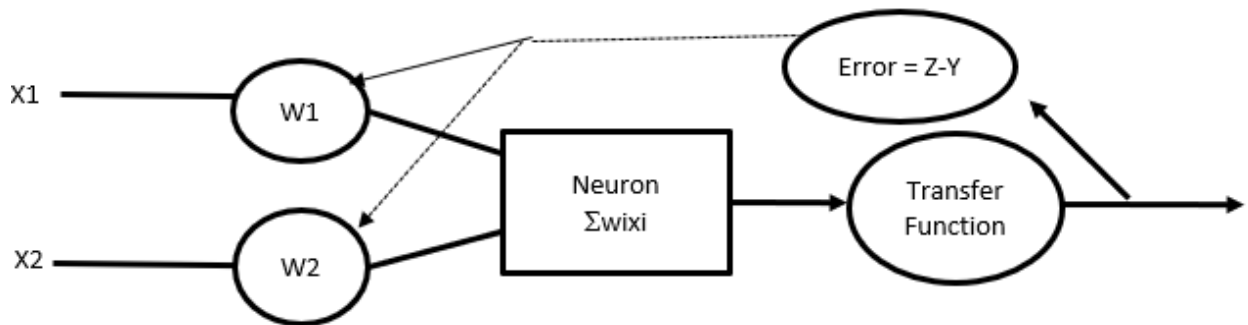
The OR binary operator has the input and output depicted in Figure 7.2. For example, if X1=1 and X2=0, the output would be 1; if the input is X1=0 and X2=0 the output would be 0. Using these four rows (again this is an unrealistically small set of data to use for training), we will train a simple ANN.

Figure 7.2 Input and Output for the Binary OR Operator

INPUTS		OUTPUT
X1	X2	Z
0	0	0
0	1	1
1	0	1
1	1	1

The training algorithm will begin with initial random guesses for W1 and W2 using pseudorandom number generation. The initial weights will be processed by the neuron, rounded by the transfer function, then the error will be calculated and used to adjust the weights with a learning parameter, α . In our example we will use $\alpha = 0.2$. This process will repeat until the weights are trained to produce the proper output (Figure 7.3).

Figure 7.3 Example Training Process for an ANN



Suppose the initial random values of W1 and W2 in Figure 7.4 were chosen:

Figure 7.4 Initial Random Weights Are Chosen for the ANN

input	input	actual output	W1	W2
X1	X2	Z		
0	0	0	0.30	0.17
0	1	1	0.75	0.03
1	0	1	0.43	0.94
1	1	1	0.23	0.82

Based on the random values of W1 and W2, the output of the ANN for each row of X1 and X2 would be determined as follows: Y (unrounded) = $X1 * W1 + X2 * W2$ and then rounded to apply the transfer function (as depicted in figure 7.5).

Figure 7.5 The Neuron and Transfer Function Are Used to Make a Prediction from the Initial Values of W1 and W2

input	input	actual output				predicted (ANN output)
X1	X2	Z	W1	W2	Y (unrounded) = X1*W1+X2*W2	Y (rounded)
0	0	0	0.30	0.17	0	0
0	1	1	0.75	0.03	0.03	0
1	0	1	0.43	0.94	0.43	0
1	1	1	0.23	0.82	1.05	1

An error is calculated by subtracting Z-Y. The error is multiplied by the learning parameter (0.2) and by the value of X1 then added to the old value of W1: $NewW1 = OldW1 + \alpha * \delta * X1$. The process is repeated for W2: $NewW2 = OldW2 + \alpha * \delta * X2$ as depicted in Figure 7.6.

Figure 7.6 Adjusting the Values of W1 and W2 Based on the Error and Learning Parameter ($\alpha = 0.2$).

input	input	actual output	Weights		Neuron	predicted (ANN output)	Apply Learning Parameter ($\alpha = 0.2$) with Error to Adjust Weights		
X1	X2	Z	W1	W2	Y (unrounded) = X1*W1+X2*W2	Y (rounded)	Error (δ)	NewW1 = OldW1 + $\alpha * \delta * X1$	NewW2 = OldW2 + $\alpha * \delta * X2$
0	0	0	0.30	0.17	0	0	0	0.30	0.17
0	1	1	0.75	0.03	0.03	0	1	0.75	0.23
1	0	1	0.43	0.94	0.43	0	1	0.63	0.94
1	1	1	0.23	0.82	1.05	1	0	0.23	0.82

The second iteration of learning then begins with the new values of W1 and W2 as depicted in Figure 7.7. Notice only the weights that resulted in an error are adjusted.

Figure 7.7 ANN Weights After One Iteration

input	input	actual output		
X1	X2	Z	W1	W2
0	0	0	0.30	0.17
0	1	1	0.75	0.23
1	0	1	0.63	0.94
1	1	1	0.23	0.82

The same process is repeated as depicted in Figure 7.8

Figure 7.8 The Second Iteration of the ANN Training Algorithm

input	input	actual output	Weights		Neuron	predicted (ANN output)	Apply Learning Parameter ($\alpha = 0.2$) with Error to Adjust Weights		
X1	X2	Z	W1	W2	Y (unrounded) = $X1*W1+X2*W2$	Y (rounded)	Error (δ)	NewW1 = $OldW1+\alpha*\delta*X1$	NewW2 = $OldW2+\alpha*\delta*X2$
0	0	0	0.30	0.17	0	0	0	0.30	0.17
0	1	1	0.75	0.23	0.23	0	1	0.75	0.43
1	0	1	0.63	0.94	0.63	1	0	0.63	0.94
1	1	1	0.23	0.82	1.05	1	0	0.23	0.82

The process would continue to repeat until the weights give the correct output. In this case, it would take two more iterations (Figures 7.9 and 7.10).

Figure 7.9 The Third Iteration of the ANN Training Algorithm

input	input	actual output	Weights		Neuron	predicted (ANN output)	Apply Learning Parameter ($\alpha = 0.2$) with Error to Adjust Weights		
X1	X2	Z	W1	W2	Y (unrounded) = $X1*W1+X2*W2$	Y (rounded)	Error (δ)	NewW1 = $OldW1+\alpha*\delta*X1$	NewW2 = $OldW2+\alpha*\delta*X2$
0	0	0	0.30	0.17	0	0	0	0.30	0.17
0	1	1	0.75	0.43	0.43	0	1	0.75	0.63
1	0	1	0.63	0.94	0.63	1	0	0.63	0.94
1	1	1	0.23	0.82	1.05	1	0	0.23	0.82

Figure 7.10 The Fourth and Final Iteration of the ANN Training Algorithm

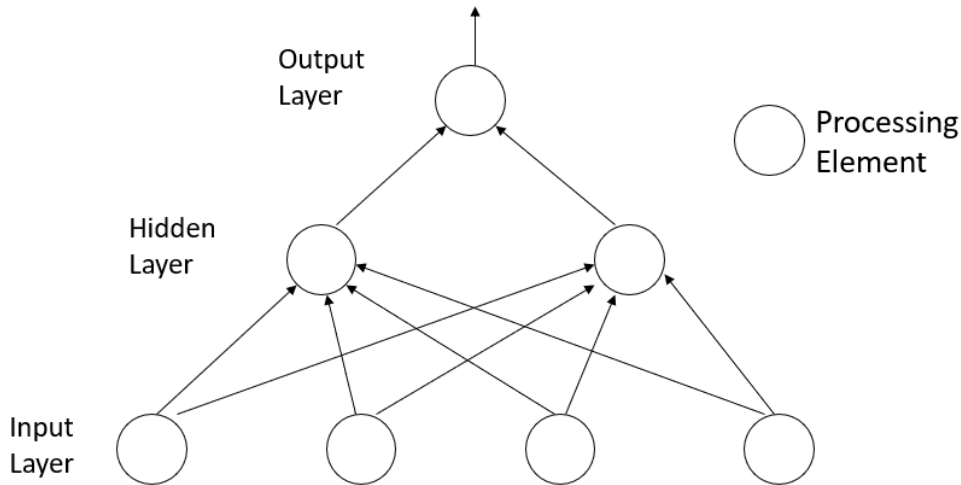
input	input	actual output	Weights		Neuron	predicted (ANN output)	Apply Learning Parameter ($\alpha = 0.2$) with Error to Adjust Weights		
X1	X2	Z	W1	W2	Y (unrounded) = $X1*W1+X2*W2$	Y (rounded)	Error (δ)	NewW1 = $OldW1+\alpha*\delta*X1$	NewW2 = $OldW2+\alpha*\delta*X2$
0	0	0	0.30	0.17	0	0	0	0.30	0.17
0	1	1	0.75	0.63	0.63	1	0	0.75	0.63
1	0	1	0.63	0.94	0.63	1	0	0.63	0.94
1	1	1	0.23	0.82	1.05	1	0	0.23	0.82

If the initial weights had been different, the final weights would certainly also be different and a different number of steps may have been needed based on the initial weights. The trained weights used in a neuron are therefore not unique (many different weights can result in a properly trained ANN that gives the correct outputs). For this reason, the weights are not reported as a part of the ANN and the ANN is considered a “black box”.

The ANN used for classification and available in analytics software such as R will generally be feed-forward neural networks with hidden layers. For example, Figure 7.11 depicts a feed-forward ANN with one hidden layer. The hidden layers allow for more complex output (such as interactions between variables) to be modeled. In the examples we will complete in R, we will use a similar neural network as

that depicted in Figure 7.11 (one hidden layer with several processing elements). Note, this is far from the most advanced commercial ANN used in industry. For example, ChatGPT3 uses an ANN with a reported 96 hidden layers and over 175 billion parameters.

Figure 7.11 Feed-Forward Network with One Hidden Layer



Artificial Neural Networks in R

We will use the student success data again. The file “student success.csv” contains records for 2,000 students enrolled in a university’s computer science department. “FinalGrade1403” is the final grade in a first class in computing. “Transfer” is whether the student transferred into the university or not. “ACT_Score” is the student’s composite ACT (college entrance exam) score. “ACT_Math” is the student’s score on the Math portion of the ACT. “HS_GPA” is the student’s high school grade point average. “success” is the target (or response) variable, what we are trying to predict—if the student successfully completed a computer science degree within 6 years. Figure 7.12 shows the first few rows of the student success data.

Figure 7.12 First Few Rows of the Student Success Data

	Student	FinalGrade1403	Transfer	ACT_Score	ACT_Math	HS_GPA	success
1	1	83	no	23	14	3.58	yes
2	2	98	yes	30	32	3.60	yes
3	3	89	no	24	21	2.77	yes
4	4	47	no	21	24	2.89	no
5	5	91	no	24	18	2.85	yes
6	6	78	no	31	31	3.52	yes

The data may be read in as usual in R.

R Code for Reading in Data

```
stu <- read.csv("C:/Users/hbrown11/Desktop/student success.csv", header=TRUE)
head(stu)
```

Again, for classification problems we will need to identify our target variable as a factor.

R Code for Setting Success as a Factor

```
stu$success <- factor(stu$success)
```

The data are now ready to be divided into training and test data sets. This process is identical to what was done in previous examples. Note, cross-validation could also alternatively be used.

R Code for Dividing Data into Training and Testing Data Sets

```
set.seed(23)
R = runif(nrow(stu)) #create a random uniform column the length of our table
stu$R = R #add the random numbers to our table
train <- stu[stu$R <= .6,] #select roughly 60% for training
test <- stu[stu$R > .6,] #the other 40% gets placed in test
train <- subset(train, select = -c(R)) #remove the column R
test <- subset(test, select = -c(R)) #remove the column R
```

We are now ready to build the ANN from the training data. There are multiple packages available for ANN in R, we will use the **nnet** package. Again, we cannot specify the model with “**success~.**” because there is an id column, “Student”. Additionally, using the student id as a predictor to the model would be nonsense (as it is an arbitrary label uniquely identifying rows in the table). For this reason, the five predictors are listed out, separated by a “+”. Notice the formula stays the same regardless of whether we are using a logistic regression model, a decision tree, a Random Forest or an ANN.

The `set.seed` controls the random weights that are used as initial guesses (makes the program repeatable). `Size=4` refers to the number of neurons to use in the hidden layer of the ANN. There is no clear consensus on the number of neurons to use in a hidden layer, one rule of thumb is two use the $2/3$ the number of inputs plus the number of outputs: 5 input columns + 1 output column = 6. Then, $0.667 * 6$ is approximately 4, which is why `size=4` was selected. Much like the `mtry` parameter in the Random Forest, changing the `size` parameter can potentially create improved performance for the ANN.

R Code for Creating the Artificial Neural Network

```
install.packages("nnet")  
library(nnet)  
set.seed(17)  
ANN<-nnet(success~FinalGrade1403+Transfer+ACT_Score+ACT_Math+HS_GPA,data=train, size=4)
```

Once the ANN is trained, the model can be fit to the test data.

R Code for Fitting the Model to the Test Data

```
model<- predict(ANN, newdata=test, type = "class")  
known<-test["success"] #extract the actual value of success for comparing  
knownt<-t(known) #transpose the data for the confusion matrix  
table(knownt,model) #create the confusion matrix
```

The confusion matrix is given in Figure 7.13.

Figure 7.13 ANN Confusion Matrix (“student success” Example)

```
      model  
knownt  no  yes  
no      116  53  
yes     35  629
```

Figure 7.14 compares the ANN results to Logistic Regression, Classification, and Random Forests. The accuracy of the ANN model is 89% and it very slightly outperforms the Logistic Regression model for this example. Either the ANN or Logistic Regression would be good choices for this problem.

Figure 7.14 Comparing the Results of the Logistic Regression, Classification, and Random Forest Models (“student success” Example)

Logistic Regression

Accuracy=89%

```

      model
knownt no yes
no     111 58
yes    31 633

```

Classification tree

Accuracy=85%

```

      model
knownt no yes
no     102 67
yes    56 608

```

Random forest

Accuracy=87%

```

      model
knownt no yes
no     110 59
yes    51 613

```

ANN

Accuracy=89%

```

      model
knownt no yes
no     116 53
yes    35 629

```

The Full R Program for the ANN

```

stu <- read.csv("C:/Users/hbrown11/Desktop/student success.csv", header=TRUE)
head(stu)
stu$success <- factor(stu$success)
set.seed(23)
R = runif(nrow(stu)) #create a random uniform column the length of our table
stu$R = R #add the random numbers to our table
train <- stu[stu$R <= .6,] #select roughly 60% for training
test <- stu[stu$R > .6,] #the other 40% gets placed in test
train <- subset(train, select = -c(R)) #remove the column R
test <- subset(test, select = -c(R)) #remove the column R
library(nnet)
set.seed(17)
ANN <- nnet(success ~ FinalGrade1403 + Transfer + ACT_Score + ACT_Math + HS_GPA, data = train, size = 4)

```

```
model<- predict(ANN, newdata=test, type = "class")
known<-test["success"] #extract the actual value of success for comparing
knownt<-t(known) #transpose the data for the confusion matrix
table(knownt,model) #create the confusion matrix
```

Module 7 Assignment

The csv file `irs.csv` contains data from past US Internal Revenue Service audits. In an effort to more efficiently audit only those returns most likely to have violated tax laws, you have been asked to build a model to predicted if cheating has likely occurred for a particular tax return.

The columns in the file `irs.csv` are:

- `cheated`: "yes" if the audit found the return violated tax laws, otherwise "no" -- this is the target variable we are building the model to predict
- `married`: "yes" if the return is for a married individual; "no" for single
- `incomeOver100K`: "yes" if the income of the individual exceeds \$100,000
- `return`: "yes" if the individual received a return ("no" if the individual did not receive a return or had to pay in)
- `filedLate`: "yes" if the return was filed late, "no" if filed early or on time
- `Over4DeductClaimed`: "yes" if over 4 deductions were claimed, "no" if 4 or less

Using the other 5 columns, use R to build an artificial neural network to predict if the audit resulted in `cheated="yes"` or `"no"`. Create a confusion matrix and calculate accuracy.

A return comes in with the following attributes: not married, income of \$120,000, received a return, filed late, and claimed over 4 deductions. Using your ANN, make a recommendation if the tax return should be audited.

Module 8: Variable (Model) Selection

Introduction to Variable/Model Selection

When building predictive models with potentially multiple predictor variables it is important to determine which subset of variables will result in the best model. Specifically, it must be determined if any or all variables can be used to make valid predictions. Further, if at least some predictors can be used, it must be determined which subset of inputs provides the greatest predictive ability. Variable selection or model selection (also called “feature extraction”) is the process of determining which inputs to use for a particular model. Thus, in variable selection we want to choose the subset of independent variables (input columns) that will create the best predictive model

The need for variable selection is somewhat counterintuitive; it may seem a model should always improve with the addition of legitimate predictors. However, there are numerous issues with including too many predictor variables or the wrong subset of predictor variables in a model. As was demonstrated at the end of module 4, using too many variables can lead to overfitting a model. Further, too many independent variables can prevent a model from being created by having too few rows to uniquely estimate the impact of each variable or due to the time required to execute the algorithm. Also, independent variables that are highly correlated with other independent variables in the same model can create issues in the algorithms used to build models, conceptually if two predictor variables say the same thing about the response variable—both are not needed. Moreover, variables can interact with other variables (a variable by itself may not be useful, but in the presence of another variable it becomes a useful predictor). Finally, fewer input variables can result in an easier to interpret model. For these reasons, selecting variables has become a routine step in the construction of predictive models.

There are many incorrect approaches to variable selection. Things NOT to do in variable selection include:

- Do not simply run a regression (or logistic regression) model and remove variables that are not significant.
- Do not repetitively run hypothesis tests to select variables.
- Do not forget that selected variables should be cross-validated or be evaluated with test data.
- Do not rely entirely on manual selection.
- Do not include a predictor variable that does not have a justifiable relationship to the response variable.

Conversely, there are several valid approaches to variable selection that can be used. In this module, we will look at three commonly used approaches, step-wise variable selection, Random Forest importance, and LASSO regression. Prior to using an algorithm for variable selection, we will also discuss certain preselection considerations that should also be made.

Preselection

Before using variable selection methods, exploratory data analysis and common sense should be applied to the process of determining candidate predictor variables. Only independent variables that can be logically justified as predictors of the dependent variable should be considered for models (e.g. when

trying to predict store sales, store manager shoe size should not be considered). Further, some columns may not be suitable as predictors due to the nature of the available data (for example, a column of categorical data with too many levels may lead to errors in algorithm implementation). In addition, simple correlation coefficient calculations can be useful for identifying highly correlated input variables (if two inputs are highly correlated, only one of the inputs should be chosen). Finally, in a particular modeling problem, you should try to learn as much about the context of the model as possible and look for those independent variables that are commonly thought to impact the dependent variable.

Stepwise Selection

Stepwise selection is one of the oldest and most commonly used methods of selecting the best subset of predictors. In stepwise selection, an algorithm is used to repetitively fit a regression model with different combinations of potential input variables, each candidate model is evaluated for its predictive ability and the model with the best performing subset of variables is selected. A commonly used metric for stepwise selection is Akaike's Information Criterion (AIC), $AIC = 2k - 2\ln(L)$, where k is the number of model inputs and L is the maximized value of the likelihood function. AIC is penalized for adding extra terms and attempts to estimate out of sample performance (corrected AIC, AICC, is better in some situations). Note that not just any measure can be used in stepwise selection! Common metrics, like R-squared, always increase if more variables are added to a model.

Random Forests for Variable Importance

Random Forests that we examined in Modules 4 and 6 can be used to aid in variable selection. Recall that Random Forests construct numerous decision trees using a subset of variables for each tree. As a result, tree performance with and without each variable can be measured across the forest. The average impact the variable has on improving decision trees when that variable is included is called its "importance". After fitting a Random Forest model, we can request the variable importance from R in terms of MSE or purity for regression and accuracy or Gini index for classification.

LASSO Regression for Variable Selection

LASSO regression, also known as least absolute shrinkage and selection operator, is a popular technique used in statistical modeling and machine learning to perform variable selection. LASSO regression imposes a penalty on the absolute values of the regression coefficients, shrinking some of them to zero, effectively performing feature selection. In ordinary least square regression the algorithm estimates parameters to minimize the "deviance" (difference between dependent and independent variables):

$$\frac{1}{n} \sum (y_i - x'_i \beta)^2$$

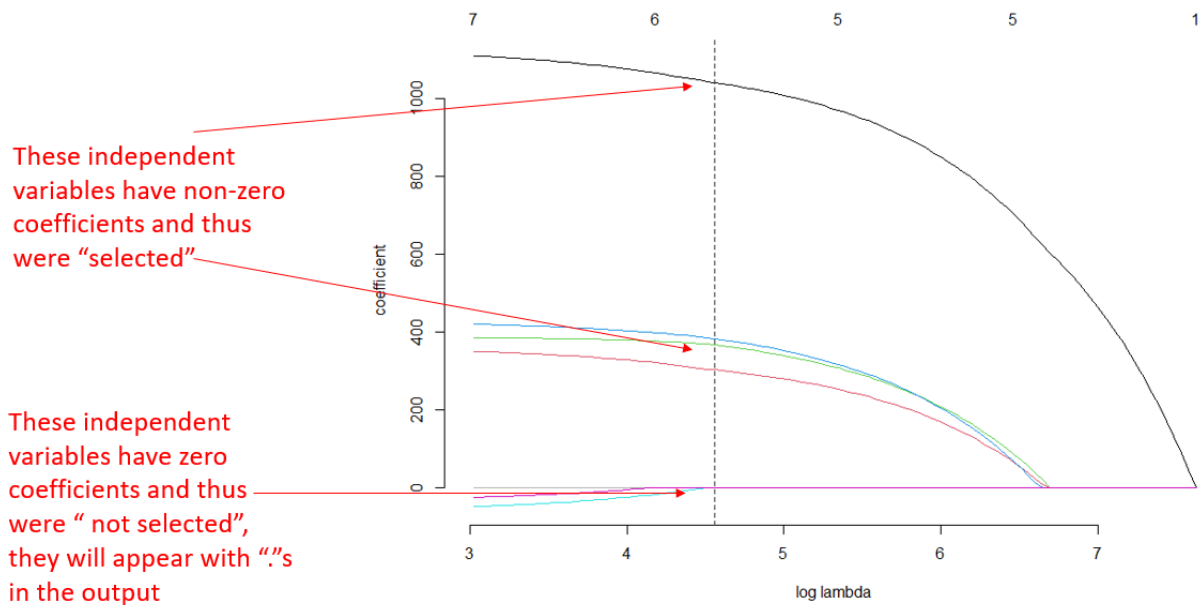
In LASSO regression the algorithm minimizes a cost function (sometimes called a penalty):

$$\frac{1}{n} \sum (y_i - x'_i \beta)^2 + \lambda \sum c(\beta_j)$$

Methods such as LASSO regression are called “regularization” (penalizing complexity), and have been shown to be an effective way to choose an optimal model. For a more detailed understanding of LASSO regression and its implementation, see Hastie, Tibshirani, and Friedman (2009) among others.

In the LASSO regression algorithm, we begin with penalties (λ) that make each parameter (β) zero, the algorithm then works to find values that minimize the fit and penalty. Parameters (β) that do not improve fit more than increasing complexity cost will stay zero. The process can be visualized with regularization paths. Variables that were not selected will have a “.” next to them in the R output. Figure 8.1 shows an example LASSO regularization path.

Figure 8.1 Example LASSO Regularization Path in R



Variables need to be on the same scale in order for LASSO regression to be effective. If variables are not on the same scale this can be accomplished by “standardization”. To standardize a column, find the mean and standard deviation for that column, then take each value in the column subtract the mean and divide by the standard deviation as follows:

$$\text{Standardized of } x_i = \frac{(x_i - \bar{x})}{s}$$

This will place all values roughly on the same scale (centered at 0, from usually ranging from around -3 to 3) while preserving the nature of the data. This can be done prior to loading the data into R or with the R function scale().

Variable Selection in R

The csv file “product survey.csv” includes data from 40 different products from the same product category for a large online retail company (Figure 8.2). The column “sales” is the average weekly sales of each item. Columns, “color”, “size”, “features”, “reliability”, “warranty”, and “cost” are the average scores from product surveys that ask about the customers’ satisfaction with each aspect of the product. A score closer to 10 indicates a higher average customer satisfaction for each item; a score closer to 1 indicates greater dissatisfaction. Determine the best subset of variables to predict sales from the customer satisfaction measures.

Figure 8.2 Contents of the “product survey.csv” File (First Few Rows)

Sales	Color	Size	Features	Reliability	Warranty	Cost
21245.16	5.4	8.9	9.0	3.4	6.5	9.0
15120.88	7.2	1.6	6.3	9.4	2.3	5.7
12351.06	8.9	5.9	7.3	8.2	9.0	2.9
18771.43	8.3	9.3	7.7	9.9	6.0	5.0
13092.65	8.0	6.5	6.6	8.5	2.1	1.6
19282.52	9.0	6.4	9.0	6.9	8.8	8.3

Stepwise variable selection with AIC for the product survey data can be completed as follows:

Stepwise Variable Select Using AIC in R

```
ps <- read.csv("C:/Users/hbrown11/Desktop/product survey.csv", header=TRUE)
head(ps)
install.packages("MASS")
library(MASS)
step_model <- stepAIC(lm(Sales ~ ., data = ps), direction = "both", trace = FALSE)
# Print the selected variables
print(step_model$anova)
```

The output for the stepwise selection is given in Figure 8.3. Notice the final variables selected were Size, Features, Reliability, and Cost; Color and Warranty were not selected. Once the variables are selected, we would proceed with building our model as we have done in previous modules, using only the selected variables.

Figure 8.3 Stepwise Selection Output (“product survey” Example)

```
Stepwise Model Path  
Analysis of Deviance Table
```

```
Initial Model:  
Sales ~ Color + Size + Features + Reliability + Warranty + Cost
```

```
Final Model:  
Sales ~ Size + Features + Reliability + Cost
```

The final variables selected were:
Size, Features, Reliability, and Cost

	Step	Df	Deviance	Resid. Df	Resid. Dev	AIC
	1			33	94345556	600.9438
	2	- Color	1 235310.4	34	94580867	599.0435
	3	- Warranty	1 652013.5	35	95232880	597.3183

Color and Warranty were not selected

Ranking variable importance using Random Forests for the product survey data can be completed as follows:

Random Forest Variable Importance in R

```
library(randomForest)
```

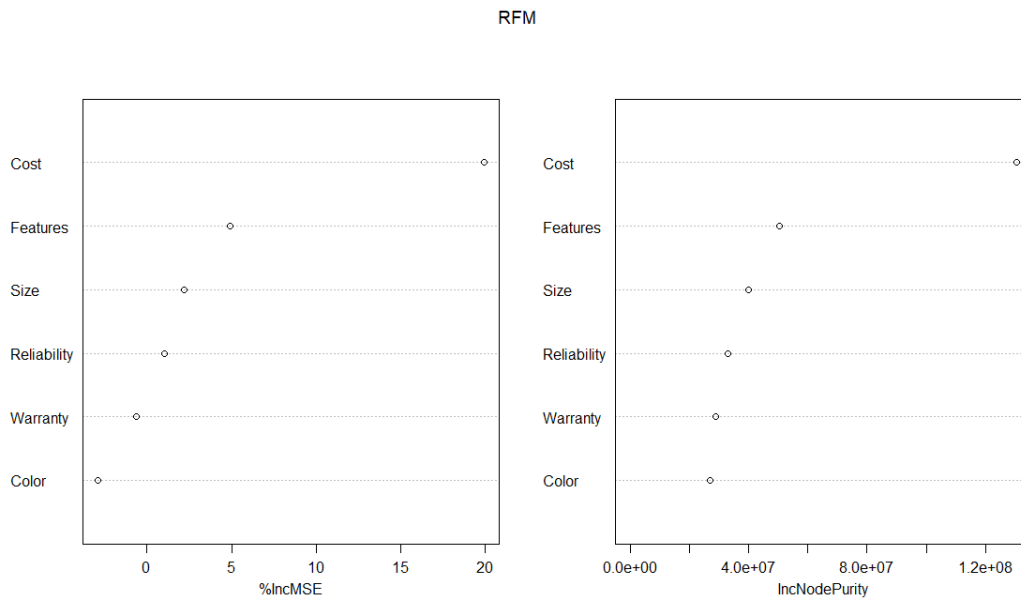
```
RFM=randomForest(Sales~.,data=ps, importance=TRUE)
```

```
importance(RFM)
```

```
varImpPlot(RFM)
```

The output for variable importance is given in Figure 8.4. Importance using MSE is given in the first graph; importance using Node Purity is given in the second—either could be used to determine which variables are most useful for building a model. Unlike stepwise and LASSO regression, when using Random Forests for variable selection, while variables are ranked by importance, there is no clear “cut-off” for where to stop including variables (it is left up to analyst to say where to stop including variables). However, in this case, warrant and color are less than 0 on %IncMSE, clearly indicating they should not be used.

Figure 8.4 Variable Importance from the Random Forest Approach to Variable Selection (“product survey” Data)



LASSO regression for the product survey data can be completed as follows (note these variables are all on the same scale, 1-10, so no standardization is needed):

LASSO Regression in R

```
install.packages("gamlr")
```

```
library(gamlr)
```

```
modMat<-model.matrix(~Color+Size+Features+Reliability+Warranty+Cost, data=ps)
```

```
fit<-gamlr(x=modMat,y=ps$Sales)
```

```
#the plot shows the coeff moving from zero to values
```

```
plot(fit)
```

```
#selected model
```

```
coef(fit)
```

The output for LASSO regression is given in Figures 8.5 and 8.6. From the output (both the regularization path and table output), we can see Size, Features, Reliability, and Cost were selected; Color and Warranty were not selected.

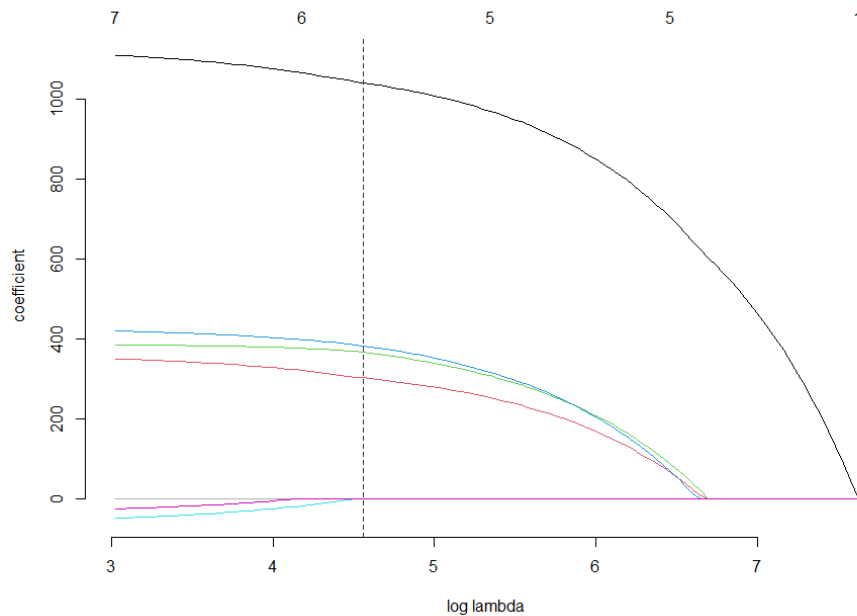
Figure 8.5 Table Output of LASSO Regression (“product survey” Data)

```

8 x 1 sparse Matrix of class "dgCMatrix"
      seg67
intercept  3338.6351
(Intercept)  .
Color      . ← Dot indicates not selected
Size      302.2088
Features  365.7851
Reliability 382.6246
Warranty   . ← Dot indicates not selected
Cost      1041.1788

```

Figure 8.6 LASSO Regularization Path in R (“product survey” Data)



Conclusions for Example 1 “product survey” Data

All three methods of variable selection agreed that Color and Warranty should not be used in the model. However, for some sets of data the different methods of variable selection will not agree on the predictors that should be used. In such instances the analyst must evaluate the recommendations made by each technique and make the final choice (perhaps trying a few of the different models that were proposed). For the product survey data, the process for building the final model would now begin using just the selected columns: Size, Features, Reliability, and Cost.

A Second Variable Selection Example

The csv file "irs2.csv" contains data from past US Internal Revenue Service audits (Figure 8.7). In an effort to more efficiently audit only those returns most likely to have violated tax laws, the columns in the file irs2.csv are:

- cheated: "yes" if the audit found the return violated tax laws, otherwise "no" -- this is the target variable we are building the model to predict
- married: "yes" if the return is for a married individual; "no" for single
- dependents: "yes" if the return is for an individual claiming dependents; "no" otherwise
- incomeOver100K: "yes" if the income of the individual exceeds \$100,000
- return: "yes" if the individual received a return ("no" if a the individual did not receive a return or had to pay in)
- largedonation: "yes" if the individual made a donation in excess of 10% of their annual income, no otherwise
- mathErrors: "yes": if there were math errors in the return, "no" otherwise
- filedLate: "yes" if the return was filed late, "no" if filed early or on time
- Over4DeductClaimed: "yes" if over 4 deductions were claimed, "no" if 4 or less

Perform variable selection for the IRS audit data using stepwise selection with AIC, Random Forest Importance, and LASSO regression.

Figure 8.7 The "irs2.csv" Data Set (First Few Rows)

```
cheated married dependents incomeOver100K return largedonation mathErrors
yes no no yes no no yes
no yes yes yes no no yes
no yes no yes no yes no
yes no no yes no no yes
no yes no yes yes yes no
no yes yes yes yes yes yes
filedLate Over4DeductClaimed
no no
no no
no no
no no
no yes
yes no
```

Stepwise variable selection with AIC for the irs2 data requires an extra step for categorical data, to turn the "yes" and "no"s into 1's and 0's (called dummy variables or indicator variables):

Preparing the Data for Stepwise Variable Selection by Creating Indicator Variables

```
irs <-read.csv("C:/Users/hbrown11/Desktop/irs2.csv", header=TRUE)
```

```
head(irs)
```

```
# Dummy code categorical predictor variables
```

```

irsdum <- model.matrix(~., irs)[-1]
irsdum<-data.frame(irsdum)
head(irsdum)
library(MASS)
step_model <- stepAIC(lm(Sales ~ ., data = ps), direction = "both", trace = FALSE)
# Print the selected variables
print(step_model$anova)
Stepwise selection with AIC then proceeds as before:
#install.packages("MASS")
library(MASS)
step_model <- stepAIC(lm(cheatedyes ~ ., data = irsdum, family=binomial), direction = "both", trace =
FALSE)
# Print the selected variables
print(step_model$anova)

```

The output for the stepwise selection is given in Figure 8.8. The variables married, incomeOver100, return, filedLate, and Over4DeductClaim were selected. (Depedents, mathErrors, and largedonation were not selected)

Figure 8.8 Stepwise Output (“irs2” Data)

```

Initial Model:
cheatedyes ~ marriedyes + dependentsyes + incomeOver100Kyes +
  returnyes + largedonationyes + mathErrorsyes + filedLateyes +
  Over4DeductClaimedyes

Final Model:
cheatedyes ~ marriedyes + incomeOver100Kyes + returnyes + filedLateyes +
  Over4DeductClaimedyes

```

	Step	Df	Deviance	Resid. Df	Resid. Dev	AIC
	1			477	41.48767	-1177.955
2	- dependentsyes	1	0.006368431	478	41.49404	-1179.880
3	- mathErrorsyes	1	0.010191336	479	41.50423	-1181.761
4	- largedonationyes	1	0.078835478	480	41.58306	-1182.839

Ranking variable importance using Random Forests for the irs2 data can be completed as follows (note, there is no need to recode the data into indicator variables for Random Forests):

Random Forest Variable Importance in R

```
irs$cheated<-factor(irs$cheated)
```

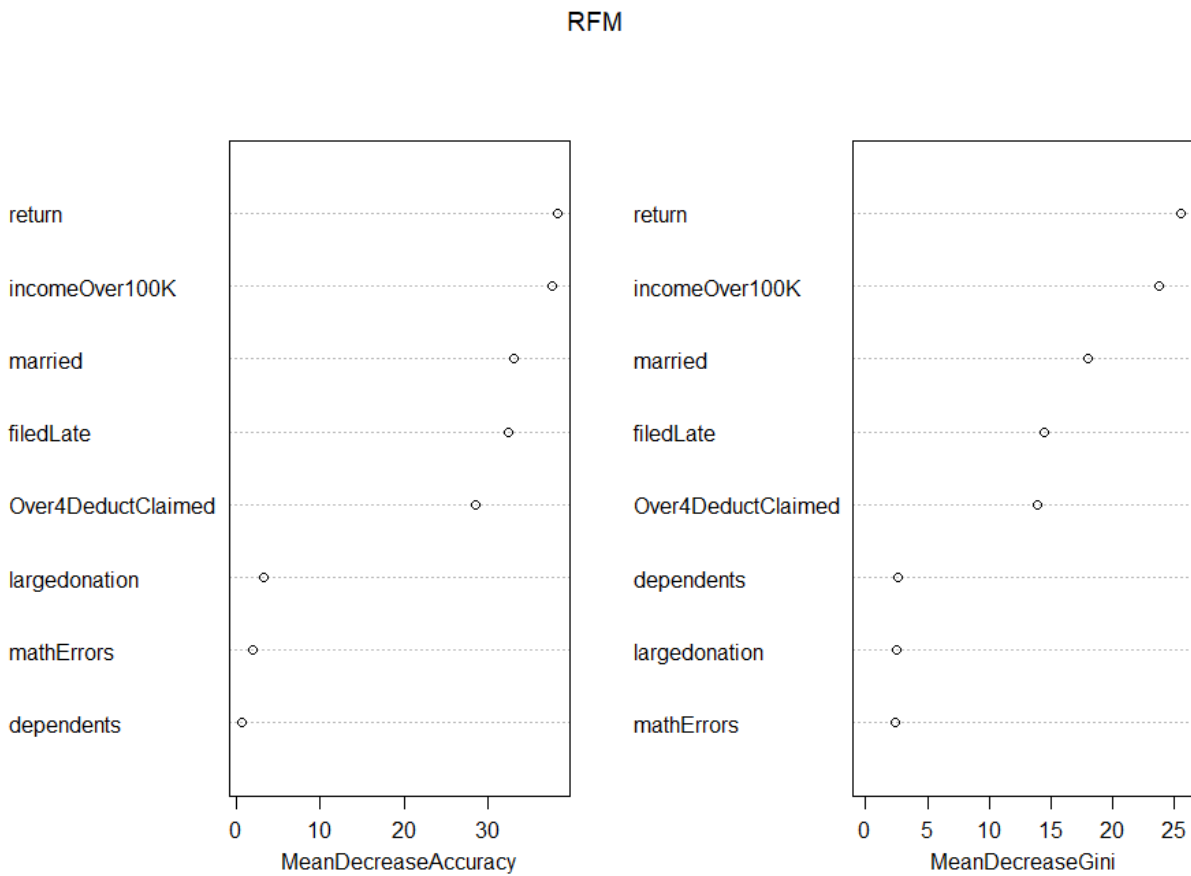
```
RFM=randomForest(cheated~.,data=irs, importance=TRUE)
```

```
importance(RFM)
```

```
varImpPlot(RFM)
```

The output for variable importance is given in Figure 8.9. Importance using accuracy is given in the first graph; importance using the Gini criterion is given in the second—either could be used to determine which variables are most useful for building a model. In this case, there is a clear drop-off for largedonation, mathErrors, and dependents, indicating they should not be used.

Figure 8.9 Random Forest Importance (“irs2” data)



LASSO logistic regression for the irs2 data again requires the creation of an indicator variables. Note however, all the variables are again on the same scale, so standardization is not needed. The R code for LASSO logistic regression is given as follows:

LASSO Logistic Regression in R

```
irs <-read.csv("C:/Users/hbrown11/Desktop/irs2.csv", header=TRUE)  
head(irs)  
# Dummy code categorical predictor variables  
irsdum <- model.matrix(~., irs)[-1]  
irsdum<-data.frame(irsdum)  
head(irsdum)  
#LASSO regression  
#install.packages("gamlr")  
library(gamlr)  
modMat<-model.matrix(~marriedyes + dependentsyes + incomeOver100Kyes +  
  returnyes + largedonationyes + mathErrorsyes + filedLateyes +  
  Over4DeductClaimedyes, data=irsdum)  
fit<-gamlr(x=modMat,y=irsdum$cheatedyes, family = "binomial")  
#the plot shows the coeff moving from zero to values  
plot(fit)  
#selected model  
coef(fit)
```

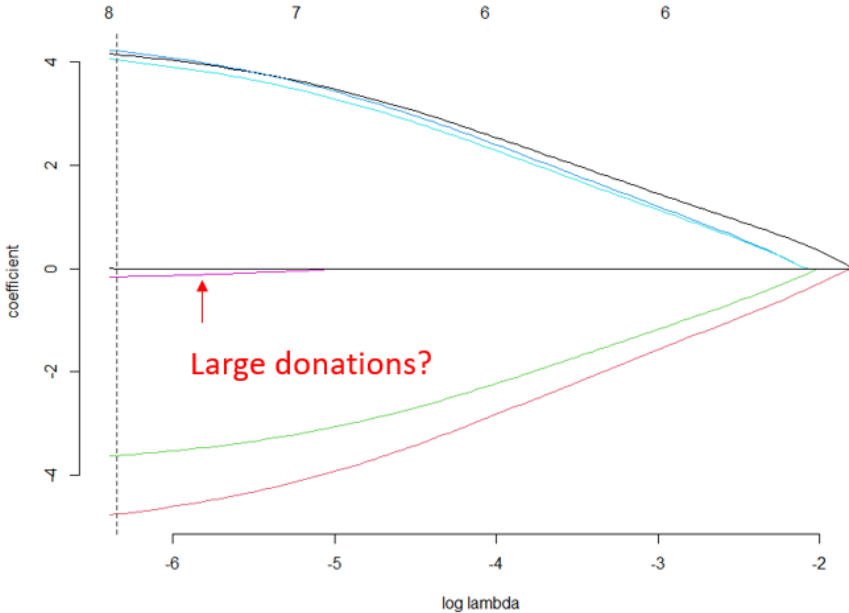
The output for LASSO regression is given in Figures 8.10 and 8.11. From the output (both the regularization path and table output), only dependents and mathErrors were removed; however, largedonation departed from zero late in the process and may not be needed; in this case both stepwise and Random Forests can help clarify whether to include largedonations as a predictor.

Figure 8.10 Table Output of LASSO Logistic Regression (“irs2” Data)

```

10 x 1 sparse Matrix of class "dgCMatrix"
      seg99
intercept      -2.5741270
(Intercept)    .
marriedyes     -3.6222103
dependentsyes  .
incomeOver100Kyes  4.1429629
returnyes     -4.7533481
largedonationyes -0.1616757
mathErrorsyes  .
filedLateyes   4.2112906
Over4DeductClaimedyes  4.0355239
    
```

Figure 8.11 Regularization Lasso Logistic Regression (“irs2” Data)



Conclusions for Example 2

For the “irs2” example, we can make the following conclusions. All three methods of variable selection agreed that dependents and mathErrors should not be used in the model. Stepwise and Random Forests indicated largedonation should also be removed. Further, largedonation was selected late in the regularization path from the LASSO logistic regression model. The final model that is built should therefore not use these three columns and instead be based on married, incomeOver100K, return, filedLate, and Over4DeductClaimedyes.

Module 8 Assignment

The file "nflcombine.csv" contains NFL combine data. The NFL combine is a yearly event where college football players that are NFL prospects are put through a series of tests. The NFL combine data (performance and measurement data) are provided for 267 college football players. The columns are as follows: Ht is the player height, Wt is the player weight, yd40 is the player's 40 yard dash time, Vertical is the player's vertical jump, Bench is the number of times 225 pounds can be bench pressed, BroadJump is the player's broad jump, Cone3 is the time it takes the player to complete a particular drill, Shuttle is the time it takes a player to complete a different drill, and "Draftedyes" indicates if the player was drafted (picked up by a NFL team). The file "znflcombine.csv" is the same data, only the eight potential predictor variables have been standardized for use with LASSO regression. Using the stepwise, Random Forest importance, and LASSO methods of variable selection, determine which of the other eight columns, if any, would be good predictors of Draftedyes. Note: for stepwise and Random Forests, "nflcombine.csv" can be used. For LASSO regression, "znflcombine.csv" must be used. (There is no need to create dummy variables or to preprocess the data in any other way as long the standardized file is used for LASSO regression.)

After variable selection analysis there may not be a perfect consensus between the three methods on variables selected; however, clearly state which variables each method selected and give suggestions for how you would proceed to construct a model (you do not have to build a particular model, just choose the variables that would be used).

Module 9: Principal Component Analysis for Predictive Modeling

Introduction to Principal Component Analysis

Principal Component Analysis (PCA) is a dimensionality reduction technique commonly used in regression analysis to handle multicollinearity and extract meaningful information from high-dimensional datasets. By transforming the original set of variables into a new set of uncorrelated variables called principal components, PCA helps identify the most important features that explain the maximum amount of variance in the data. These principal components are ordered based on their respective eigenvalues, with the first few components capturing the majority of the variance. PCA can be particularly useful in regression when dealing with a large number of correlated predictors, as it reduces the dimensionality of the data while retaining as much information as possible. PCA can be used in combination with any modeling technique to prepare the set of predictor variables. When PCA is used in combination with regression, it is often called principal component regression (PCR).

PCA finds orthogonal transformations of correlated data, creating a new smaller set of uncorrelated factors. Eigenvectors and eigenvalues are found for the covariance matrix, and their relative significance is measured and a few vectors are selected to represent the data. (Eigenvectors and eigenvalues are a linear algebra topic and numerous tutorials are available online if you are interested, we will not go through the mathematics.) See Johnson & Wichern (2002) among others for a more detailed introduction to principal component analysis.

Advantages of PCA for Predictive Modeling

There are several advantages to using PCA for predictive modeling. PCA combines columns into a smaller subset of columns that can be used for predictive modeling. Further, PCA provides an alternative to variable selection to reduce the number of inputs, without losing information. Moreover, PCA can increase the understand of a set of data by discovering underlying or “latent” factors.

Disadvantages of PCA for Predictive Modeling

Principal component analysis also has disadvantages for predictive modeling. PCA adds extra complexity to the model building process. It can be difficult to explain to a nontechnical audience and using variable selection when possible can be a simpler solution. In addition, the new factors discovered by PCA lose the practical meaning of their numeric values (PCA produces numbers that are not easily interpreted in the original scale of the predictors).

PCA Process

Principal component analysis process starts as we typically begin a modeling process by dividing the data into training and testing data sets. We determine the number of principal components to use by examining the eigenvalues. A typical rule of thumb is to only use factors with eigenvalues greater than one. During this phase you may also want to examine the underlying factors—each principal component is a latent variable that can often be “named” by how the original variables load to that

factor. Using the selected principal components, a predictive model is then constructed. As always, we evaluate the final model performance on the test data in our typical manner.

An Example of PCA in R for Regression

The file “empRet.csv” contains data from 100 large companies on retention and the results of employee surveys on job satisfaction (Figure 9.1). The target column is retention, which is the proportion of employees retained at the company after two years. The following 20 columns are predictor variables with the average results from survey items where the employees were asked to rate the company on each of the following characteristics (1 being the employee is very dissatisfied with the company’s performance in this area, 5 being neutral, 10 being the employee is very satisfied with the company’s performance in this area): Trust, Stable, Diverse, Ethical, Opportunity, Pay, Communication, Benefits, HomeWorkBalance, Respected, Rewarded, Engaged, Environment, Leadership, Recognition, Development, Culture, Hours, TimeOff, and ProfitFocus.

The goal is to create a model predicting retention using the other 20 columns. However, many of the 20 columns are highly correlated. All columns potentially impact retention, so, rather than use variable selection, principal components will be used to combine the 20 columns into a suitable number of factors that will then be used to predict retention.

Figure 9.1 “empRet.csv” Data (First Few Rows)

Retention	Trust	Stable	Diverse	Ethical	Opportunity	Pay	Communication	Benefits
0.822	6.8	7.0	7.9	8.1	6.4	6.2	4.4	5.8
0.897	8.1	6.5	8.5	8.5	6.7	6.9	5.7	6.6
0.808	6.2	6.2	5.7	5.4	6.1	6.1	6.0	5.9
0.735	6.8	6.0	5.9	5.7	6.4	6.5	4.4	6.9
0.774	4.0	5.9	3.8	4.1	6.3	6.6	7.7	6.9
0.888	6.9	7.9	6.8	6.9	7.8	7.9	4.2	8.2
HomeWorkBalance	Respected	Rewarded	Engaged	Environment	Leadership	Recognition		
5.8	6.1	6.1	6.5	6.3	5.9	6.0		
3.9	7.4	7.1	6.8	7.3	7.6	7.0		
6.3	7.9	6.3	7.8	7.8	4.9	8.1		
3.6	6.3	6.3	6.2	6.2	4.7	6.3		
4.7	9.7	6.7	9.4	9.8	1.6	9.6		
6.8	6.4	7.9	6.6	6.3	6.3	6.2		
Development	Culture	Hours	TimeOff	ProfitFocus				
6.3	6.3	5.7	5.7	6.7				
6.8	7.8	4.1	4.0	5.2				
6.0	4.9	6.4	6.2	7.3				
6.4	4.8	3.8	3.4	8.3				
6.2	2.0	5.0	4.7	7.1				
7.8	7.2	7.1	6.9	9.1				

The following R code performs PCA for the empRet data. The data are first divided into training and testing data sets as usual. Note, the predictor columns are extracted from the data frame for PCA analysis into the data frame entitled forpca. Data must be standardized before PCA. However, standardization is built into the PCA procedure in R.

R Code for Principal Component Analysis

```
ret <-read.csv("C:/Users/hbrown11/Desktop/empRet.csv",header=TRUE)

head(ret)

set.seed(42)

R=runif(nrow(ret))

ret$R=R

train<-ret[ret$R<=.6,]

test<-ret[ret$R>.6,]

train<-subset(train,select=-c(R))

test<-subset(test,select=-c(R))

#install and load needed packages, installation only needs to happen once

install.packages("FactoMineR")

library("FactoMineR")

install.packages("factoextra")

library("factoextra")

#extract the 20 predictor variables to create principal component factors (the response variable
should never be included in the principal component analysis for principal component regression):

forpca<-subset(train,select=-c(Retention))

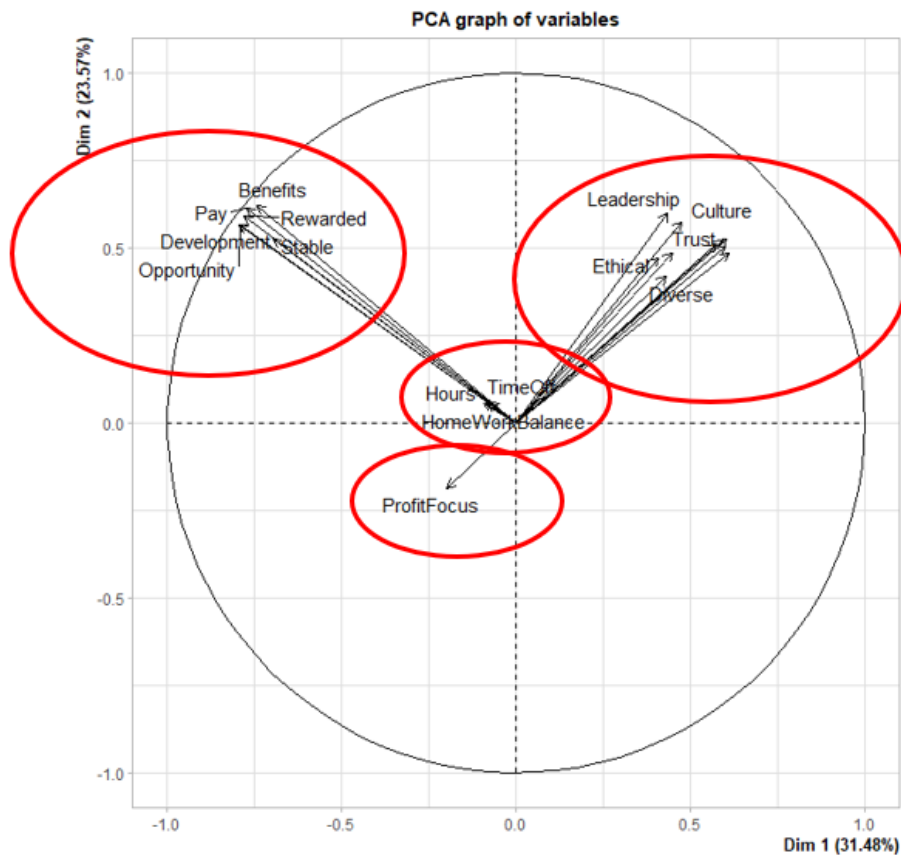
#head(forpca)

#use PCA to create underlying factors from the 20 predictor variables, note scale.unit=TRUE should
always be used to tell R to standardize the variables before PCA

pca<-PCA(forpca, scale.unit = TRUE)
```

The PCA graph of variables (Figure 9.2) produced by R can be helpful for understanding how the original 20 predictor variables are being combined into a handful of new latent variables. For example, responses to the survey items Opportunity, Development, Stable, Reward, Pay, and Benefits all seem to be related and may be captured through a single latent variable. This can more closely be examined in the factor pattern, see Figure 9.4.

Figure 9.2 PCA Graph of Variables ("empRet" data)



The following code can be used to display the eigenvalues to determine how many principal component factors to use.

R Code for Principal Component Analysis

#examine the eigenvalues and the variation explained

```
ev<-get_eigenvalue(pca)
```

```
ev<-data.frame(ev)
```

```
ev
```

#create a scree plot (same information, visually displayed)

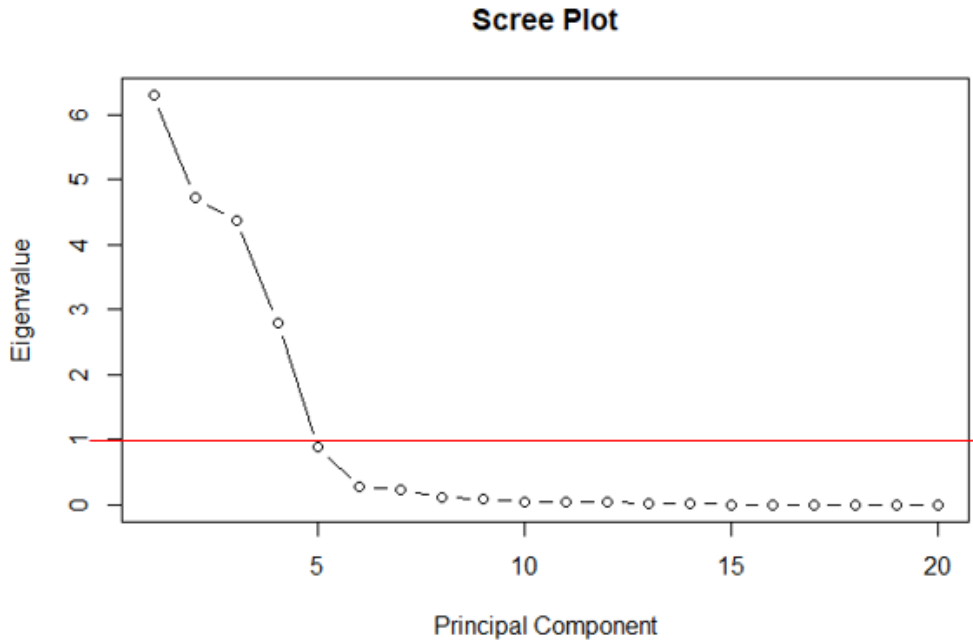
```
plot(ev$eigenvalue, type="b", main="Scree Plot", xlab="Principal Component", ylab="Eigenvalue")
```

The R output is displayed in Figure 9.3. The magnitude of the eigenvalue and the cumulative percent of the variance explained are used to determine how many factors (dimensions) to use. The rule of thumb is to only use eigenvalues over 1. The scree plot shows the same information in a plot as the table

provides numerically (Figure 9.4). For these data, the first four principal components should be used (all have eigenvalues above 1).

Figure 9.3 Table of Eigenvalues and Scree Plot (“empRet” Data)

	eigenvalue	variance.percent	cumulative.variance.percent
Dim.1	6.296689459	31.483447293	31.48345
Dim.2	4.714368160	23.571840798	55.05529
Dim.3	4.379145590	21.895727948	76.95102
Dim.4	2.794828331	13.974141653	90.92516
Dim.5	0.887647162	4.438235809	95.36339
Dim.6	0.271258448	1.356292240	96.71969
Dim.7	0.239445132	1.197225661	97.91691
Dim.8	0.118806798	0.594033991	98.51095
Dim.9	0.088234580	0.441172900	98.95212
Dim.10	0.051443008	0.257215038	99.20933
Dim.11	0.040002608	0.200458038	99.40978



The following code can be used to display the factor pattern to see how the four principal components relate to the original 20 columns.

R Code for Principal Component Factor Pattern

```
fp<-get_pca_var(pca)
```

```
fp$cor
```

The factor pattern is displayed in Figure 9.4. The factor pattern shows the correlation between each principal component and the original 20 predictor variables, it tells a similar story to the “PCA graph of

variables". Note: Dim. 1 (or dimension 1) is also called principal component 1 or PC1, Dim.2 = PC2, etc. Correlation close to 1 or -1 tells you about the composition of the principal component. For example, Dim. 4 (PC4) is highly correlated with HomeWorkBalance, Hours, and TimeOff. This factor pattern can be used to "name" a latent factor. These three survey items are measuring the same underlying variable that could be named something like "employee satisfaction with workload". Dim. 1 is relatively highly correlated with Communication, Respected, Engaged, Environment, and Recognition and could be named something like "employees perception of being valued".

Figure 9.4 Factor Pattern ("empRet" Data)

	Dim.1	Dim.2	Dim.3	Dim.4
Trust	0.44723307	0.48580465	-0.66752606	0.17210424
Stable	-0.69753314	0.52492441	-0.04246610	0.01097424
Diverse	0.42877328	0.41855707	-0.71229157	0.20598112
Ethical	0.40621007	0.47279516	-0.71268918	0.20871992
Opportunity	-0.79173958	0.56720996	0.09497901	-0.09321861
Pay	-0.77264701	0.61537945	0.03180480	-0.07704519
Communication	0.57415277	0.51334871	0.59403583	-0.17495977
Benefits	-0.74254405	0.62267344	0.02630049	-0.04816483
HomeWorkBalance	-0.08003229	0.05325693	0.42770024	0.89728389
Respected	0.59057080	0.52357963	0.57951621	-0.18981138
Rewarded	-0.77958026	0.59285242	0.01220098	-0.07876104
Engaged	0.59827922	0.50590529	0.57529254	-0.12992252
Environment	0.60411357	0.52663211	0.56538951	-0.16175668
Leadership	0.43523125	0.59954823	-0.56864866	0.20177570
Recognition	0.61081503	0.48581294	0.57128830	-0.18305930
Development	-0.79427430	0.56859008	0.07758231	-0.08644351
Culture	0.47404306	0.57479279	-0.52919211	0.18215359
Hours	-0.09105259	0.05644912	0.41352712	0.90107627
TimeOff	-0.07138517	0.06025391	0.41486018	0.90232435
ProfitFocus	-0.19772861	-0.18634569	0.32312909	-0.01183239

A regression model can be built using the four principal components with the following R code.

The R Code for PCR

```
install.packages("pls")

library("pls")

#The options say to use 4 principal components from the training data, scale the data

pca_reg<-pca(Retention~.,data=train,ncomp=4,scale=TRUE)

#apply the model to the test data and calculate MAE, RMSE, and RSQUARE (as we have done with
other models)

predicted<-predict(pca_reg,ncomp=4,newdata=test)
```

```

mae<-mean(abs(test$Retention-predicted))
cat("MAE",mae,"\n")
rmse<-(mean((test$Retention-predicted)**2))**.5
cat("RMSE",rmse,"\n")
rsquared=1-( sum((predicted-test$Retention)^2))/(sum((test$Retention-mean(test$Retention))^2))
cat("Rsquared",rsquared,"\n")

```

Applying the PCR model to the test data results in a MAE of 0.018, a RMSE of 0.023, and a R-squared value of 0.9088. These results are all promising, again context would be needed to say if the model was useful to the business. We can, however, compare the PCR model to regression with all 20 predictors.

R Code for Comparing Regular Multiple Regression to the PCR Model

```

reg<-lm(Retention~.,data=train)
predicted<-predict(reg,newdata=test)
mae<-mean(abs(test$Retention-predicted))
cat("MAE",mae,"\n")
rmse<-(mean((test$Retention-predicted)**2))**.5
cat("RMSE",rmse,"\n")
rsquared=1-( sum((predicted-test$Retention)^2))/(sum((test$Retention-mean(test$Retention))^2))
cat("Rsquared",rsquared,"\n")

```

Applying the multiple regression model with all 20 predictors to the test data results in a MAE of 0.025, a RMSE of 0.031, and a R-squared value of 0.8414. The PCR model clearly performs better in this instance.

Full R Code for PCA

```

ret <-read.csv("C:/Users/hbrown11/Desktop/empRet.csv",header=TRUE)
set.seed(42)
R=runif(nrow(ret))
ret$R=R
train<-ret[ret$R<=.6,]

```

```

test<-ret[ret$R>.6,]
train<-subset(train,select=-c(R))
test<-subset(test,select=-c(R))
#install.packages("FactoMineR")
library("FactoMineR")
#install.packages("factoextra")
library("factoextra")
forpca<-subset(train,select=-c(Retention))
pca<-PCA(forpca, scale.unit = TRUE)
ev<-get_eigenvalue(pca)
ev<-data.frame(ev)
ev
plot(ev$eigenvalue, type="b", main="Scree Plot", xlab="Principal Component", ylab="Eigenvalue")
fp<-get_pca_var(pca)
fp$cor
#install.packages("pls")
library("pls")
pca_reg<-pca(Retention~.,data=train,ncomp=4,scale=TRUE)
predicted<-predict(pca_reg,ncomp=4,newdata=test)
mae<-mean(abs(test$Retention-predicted))
cat("MAE",mae,"\n")
rmse<-((mean((test$Retention-predicted)**2))**.5)
cat("RMSE",rmse,"\n")
rsquared=1-( sum((predicted-test$Retention)^2))/(sum((test$Retention-mean(test$Retention))^2))
cat("Rsquared",rsquared,"\n")

```

Guidelines for Variable Selection Versus Principal Component Analysis

In many instances either variable selection or PCA may be used to reduce the number of predictors before creating a predictive model. However, here are some guidelines for consideration when variable selection or PCA could be used. If some of the predictors are correlated and it is reasonable to assume they may be combined to form latent variables, consider using PCA. If there is a high number of potential predictors and all can be justified as important to the response variable, consider using PCA. If there are very few potential predictors, PCA should not be used. If there is a potential some of the predictors may not predict the response variable well, and it would be beneficial to know this, variable selection should be used. If there is a need to keep the model in terms of the original variables, PCA should not be used.

Module 9 Assignment

A large retail company wants to predict the use of their store credit card at their different retail locations using sales in different departments. Sales in all departments can be expected to contribute to StoreCardUse and sales between departments are correlated. The file "storeCard.csv" contains average daily department sales and average store card use for 200 stores. (StoreCardUse=average amount placed on the store credit card, the remaining 13 columns are average daily sales for the given department: HealthandBeauty, PaperGoods, Toys, Pets, SportingGoods, Automotive, Hardware, HouseholdChemicals, KitchenandDining, LawnandGarden, HomeDecor, BooksandMagazines, and Electronics).

Use principal component regression to predict "StoreCardUse" using an appropriate number of principal components. Divide the data into training and testing data sets, include the PCA graph of variables, the scree plot, the number of principal components you selected, a brief description of the how the factors are correlated to the original predictor variables, the MAE, RMSE, and R-squared for the test data, and a brief conclusion of how the model performed. Include your R program.

Group Review Project 2

As a group, prepare a presentation that presents the most interesting discoveries from the Somerville, MA Happiness Survey using exploratory data analysis and predictive models. There are potentially information quality issues in the data; exploratory data analysis will need to be used. Different predictive models can be built using different combinations of variables as predictor and dependent variables. Note, there are potentially many “correct” answers to this project, it is an open ended “what can you find” assignment. You can take different points of view in your analysis and can choose to only use subsets of the data. You are limited to only the predictive models we have covered in this course: regression, multiple regression, logistic regression, regression and classification trees, Random Forests, artificial neural networks, variable selection, and principal components.

Each person in the group must contribute slides to the presentation that include output from the R code they constructed.

The attached data are from <https://data.somervillema.gov/Happiness/Somerville-Happiness-Survey-Responses/bi8e-5vw8>

Included are three files, the raw data as an Excel file—if you want to use some of the data that was removed during the cleaning process, a csv file where the data are cleaned, and a pdf of 2019 survey (so you can see how the data were collected).

According to the documentation:

Every two years, the City of Somerville sends out a happiness survey to a random sample of Somerville residents. The survey asks residents to rate their personal happiness, wellbeing, and satisfaction with City services. This combined dataset includes the survey responses from 2011 to 2019. The surveys are attached as PDFs.

Within the data, an NA or blank indicates that the question was not asked during the identified year while a "999" indicates a nonresponse to a question which was asked.

happinessSurvey.csv has been cleaned to remove missing values, to use only more current data, and to remove some problematic columns

Included in the file are the following columns:

- ID = Combined ID
- Year = Year
- HappyNow = How happy do you feel right now
- GeneralLifeSatisfaction = How satisfied are you with your life in general
- SvilleSatisfaction = How satisfied are you with Somerville as a place to live
- NbrHoodSatisfaction = How satisfied are you with your neighborhood
- AvailabilityInfoCity = How would you rate the following: The availability of information about city services
- RateCostofHousing = How would you rate the following: The cost of housing
- RateTrustinPolice = How would you rate the following: Your trust in the local police

- RateMaintenanceStreets = How would you rate the following: The maintenance of streets and sidewalks
- RateSocialEvents = How would you rate the following: The availability of social community events
- SafeWalkingatNight = How safe do you feel walking in your neighborhood at night
- SatisfiedwithBeautyNbrhood = How satisfied are you with the beauty or physical setting of your neighborhood
- SatisfiedParks = How satisfied are you with the appearance of parks and squares in your neighborhood
- Gender = What is your gender
- Age = Age
- Ethnicity = What is your race or ethnicity
- Children = Do you have children age 18 or younger who live with you
- HousingStatus = Describe your housing status in Somerville
- YearsResidence = How long have you lived here
- Income = What is your annual household income
- Student = Are you a student
- CityDirection = Do you feel the City is headed in the right direction or is it on the wrong track
- SafeCrossingStreet = How safe do you feel crossing a busy street in Somerville
- ConvenientTravel = How convenient is it for you to get where you want to go
- SatisfiedHousingCond = How satisfied are you with the condition of your housing

Your group's final presentation must include the following:

- It must be a single, cohesive and professional presentation.
- At least one graph or table that presents a discovery from the data.
- At least one predictive model that analyzes the data.
- At least one meaningful contribution from data analysis using R for each member of the group.
- Assume you are making the presentations for a non-data analyst audience. Include an interpretation of what each analysis means and why it is interesting in terms of practical information in terms that are easily understood.
- An appendix containing the R programs used, labeling each member's R program contribution.

References

(In order of Appearance)

- R Project. (n.d.). The Comprehensive R Archive Network (CRAN). <http://cran.r-project.org/>
- Posit Cloud. (n.d.). Home. <https://posit.cloud/>
- W3 Schools. (n.d.). R Tutorial. Retrieved May 17, 2023 from <https://www.w3schools.com/r/>
- Nti, I. K., Nyarko-Boateng, O., & Aning, J. (2021). Performance of Machine Learning Algorithms with Different K Values in K-fold Cross-Validation. *International Journal of Information Technology and Computer Science*, 13(6), 61-71.
- Fahrmeir, L., Kneib, T., Lang, S., & Marx, B. D. (2022). Regression models. In *Regression: Models, methods and applications* (pp. 23-84). Berlin, Heidelberg: Springer Berlin Heidelberg.
- Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. CRC Press.
- Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5-32.
- Department of Education. (2019) College Scorecard [Data set]. data.gov. <https://catalog.data.gov/dataset/college-scorecard>
- Hosmer Jr., D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied Logistic Regression*. Wiley.
- Schworer, A., & Hovey, P. (2004). Newton-raphson versus fisher scoring algorithms in calculating maximum likelihood estimates.
- Kaggle. (2021) Titanic: Machine Learning from Disaster [Data set]. Kaggle.com <https://www.kaggle.com/c/titanic/data>
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.
- Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction* (Vol. 2, pp. 1-758). New York: springer.
- Johnson, R. A., & Wichern, D. W. (2002). *Applied multivariate statistical analysis*.
- City of Somerville. (2023) Somerville Happiness Survey Responses [Data set]. data.gov. <https://data.somervillema.gov/Happiness/Somerville-Happiness-Survey-Responses/bi8e-5vw8>