RStudio Portfolio

Spring 2024

By: Nidhi Ladda

Table of Contents

Introduction

NOTES/ DESCRIPTION	CODE / INSTRUCTIONS
To convert an excel sheet to a csv that can be imported into R	1. Download Excel sheet 2. Enable editing 3. File > Save As 4. Dropdown > CSV (comma delimited)(*.csv) 5. Save to the AQM2000 data folder
Basics: To run a line of Code in R Studios To re – run a line of code from top to bottom - Must sweep before doing this To sweep - Click the broom Two equal signs means equal to - == ! false ! = not equal to Install a package Activate a package	 Ctrl + Enter Ctrl A Ctrl + Enter install.packages (“ggplot2”) library (ggplot2)
#Load Data Frame Need to Set Working Directory when creating a new file Session > Set Working Directory > Choose Directory > AQM2000Spring24 > Data Run 2 R files at the same time	 setwd("C:/Users/nladda1/Desktop/AQM2000spring24/Data") - Once setting the working directory you must copy and paste the above line of code and run it through source("C:/Users/nladda1/Desktop/TheName.R") source("C:/Users/nladda1/Desktop/BabsonAnalyticsR.R")
Import the data frame into R Studios df = data frame To view the entire data frame - It will open in a separate tab all the data points Summary of the entire data frame	df = read.csv (“title.csv”) ex: df=read.csv("ToyotaCorolla.csv") View(df) str(df)

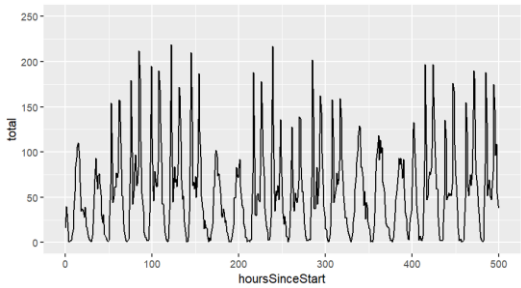
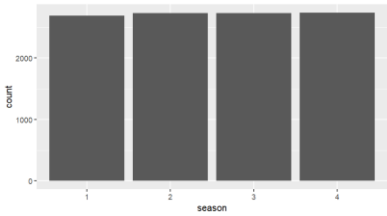
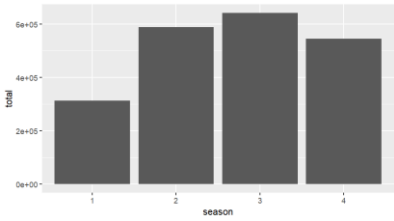
Picking a file that is not in the directory	df = read.csv(file.choose()) > choose the file
Creating a new column	df$newname = df$oldcolumname
If else statement - If value a > 5 then 1, if not 0	df$d = ifelse (df$a>5, 1, 0)
“ # ” symbol is used to make comments and explain the lines of code - It also won't run that line of code as a code - Ex: if you NULL something then need to unNull it	Ex: #Load Data Frame #Manage Data #Partion #Build Model #Key Performance Indicator #Graph
R Studios can do calculations	Ex: 5+3 5-3 8/2 sqrt(9)
To label a line of code with a name	Name of choice = line of code or value Ex: a = 3+5 a #under the values section of the global environment you will see the number 8
Creating a vector	vector name = c (#, #, #) ex: a = c(1,3,5,7) b = c(2,4,6,8) c = c(4,6,8,10)
New data frame	Name of data frame = data.frame (#.#,#) Name of data frame = data.frame (df$columnname, def$column, df$column) Ex: new_df = data.frame (a, b, c)

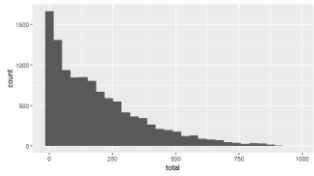
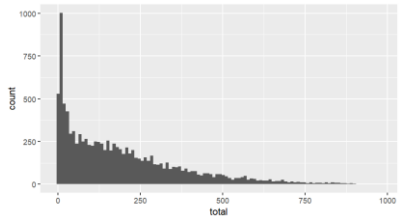
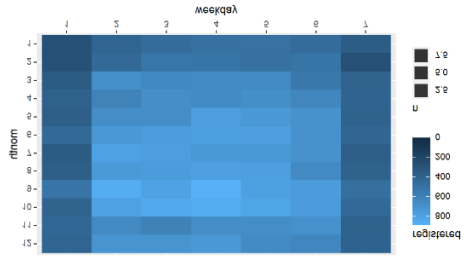
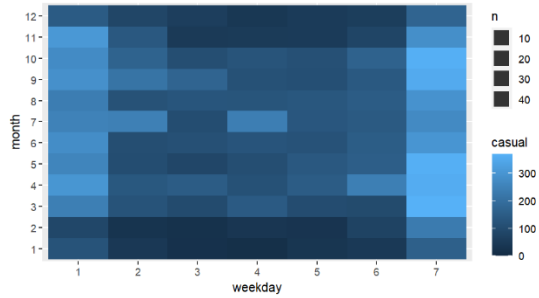
To get a data point in [row , column]	df [1 , 3]
To get every data point in a row	df [, 5]
To get every data point in a column	df [5,]

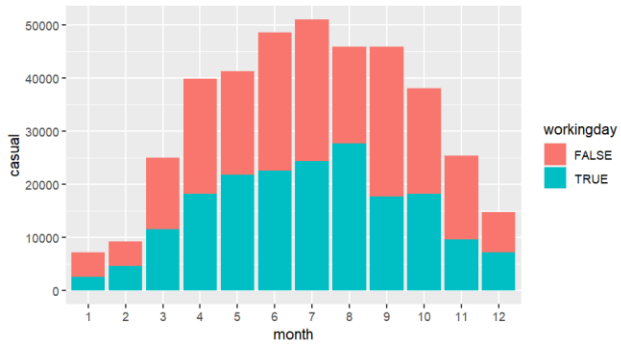
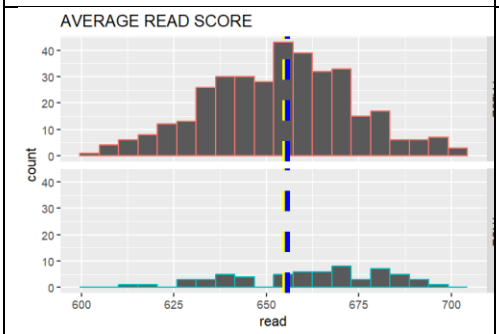
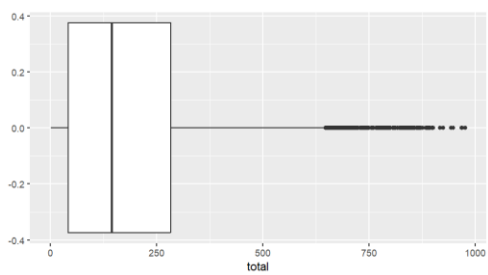
Data Management

NOTES/ DESCRIPTION	CODE / INSTRUCTIONS
Change data structure of variable - Factor: categories, intervals, words - Logical: yes/no, 1/0, true/false (has to be in numbers in order to get it into true false statement) -if it is not already in number form you must create an ifelse statement prior - Integer: any whole number - Number: number w/ decimal	df$rowname = as.factor(df$rowname) = as.logical = as.int = as.num Ex of converting into logic statement if not already in 1/0 - df$grades = ifelse(df$grades=="KK-06",1,0) - df$grades=as.logical(df$grades)
How many missing data points Always do this Cleaning data for missing pieces Always do this Check for any missing data	sum(is.na(df)) df = na.omit(df) anyNA(df)
Clear graph history Always do this	dev.off
To NULL a variable	df$rowname = NULL

Data Visualization

NOTES/ DESCRIPTION	CODE / INSTRUCTIONS
Graphs w/out ggplot 1. Histogram 2. Scatterplot matrix	1. hist(df$rowname) 2. pairs(df)
Install ggplot2	library(ggplot2)
Line Graph	ggplot(df, aes(x = hoursSinceStart, y = total)) +geom_line()+xlim(0,500)+ylim(0,250) 
Bar Chart by Count	ggplot(df, aes(x = season))+geom_bar() 
Bar Chart by Indentity	ggplot(df, aes(x = season, y = total))+geom_bar(stat = "identity") 

Histogram Find Bin width - square room of the total observation Histogram with Bin width	ggplot(df, aes(x = total))+geom_histogram()  sqrt(#of observations) ggplot(df, aes(x = total))+geom_histogram(bins = #) +geom_histogram(bins = round(sqrt(420),0)) 
Heat Mapping - fill is what the shading is based on	ggplot(df, aes(x = weekday, y = month, fill = registered)) + geom_tile(stat = "sum")  ggplot(df, aes(x = weekday, y = month, fill = casual))+geom_tile(stat = "sum") 

Stacked Bar Plot	ggplot(df, aes(x=month, y = casual, fill = workingday)) + geom_bar(stat = "identity") 
 	ggplot(df, aes(x=read,color=grades)) +geom_histogram(bins=round(sqrt(420),0))+facet_grid(grades~.)+ geom_vline(aes(xintercept = mean(read)),color = "yellow", linetype ="dashed", linewidth =2)+ geom_vline(aes(xintercept = median(read)),color = "blue", linetype ="dashed" ,linewidth=2)+labs(title="AVERAGE READ SCORE")
Box Plot	ggplot(df, aes(x = total))+geom_boxplot() 
Set limits for the data	+xlim(0,500) + ylim(250,0)
Add title	+labs (x = “name”, y = “name” , title = “name”)
Scale outliers with color	+scale_color_gradient (low = “red”, high = “blue”)
Add mean line	+geom_vline(aes(xintercept = mean(xvariable)), color = “red”, linewidth = 2)
Add median line	+geom_vline(aes(xintercept = median(xvariable)), color = “red”, linewidth = 2)

Make a stacked graph into 2 charts	+facet_grid(thefillvariable~.)
Dashed line	+geom_vline(aes(xintercept=median(read)),color="yellow",linetype="dashed", size=1)
Statistics median mean standard deviation calculate z-score Find Max value Find Min value Correlation for variables	 median(df$rowname) mean(df$rowname) sd(df$rowname) z_score = (df$numericrowname – mean (df$numericrowname)) / sd (df$numericrowname) max(df$rowname) min(df$rowname) cor(df)
How to find outliers 1. calc the z-score 2. create a new data frame for z-score 3. find the z-scores greater than 3 standard deviations labeled as outliers z-score: x - mean/standard deviation	z.scores <- (df$Price - mean(df$Price))/(sd(df$Price)) z.scores <- data.frame(df$Price, z.scores) outliers <- (z.scores$z.scores > 3) outliers

Linear Regression

NOTES/ DESCRIPTION	CODE / INSTRUCTIONS
STEP 1: PARTITION THE DATA	
Setting a Seed - randomization does not exist in computer science - there is a randomization formula which is shown through pathways that determine how the data is chosen	set.seed (###)
Count the number of rows in the data frame and store the value in "N"	N=nrow(df)
TRAINING 1: Provided how many pieces of data is 60% of the observations at rondom in trainingSize	trainingSize = round(N * .6)
TRAINING 2: Actually pulled the individual data observations at random	trainingCases = sample (N, trainingSize)
TRAINING 3: Pulls all of the columns of the training data and stores it as a data frame	training = df [trainingCases ,]
TEST DATA: Pulling all of the columns for the testing data set that are not in the training data set	test = df [- trainingCases,]
BUILDING LINEAR REGRESSION MODEL	
*if you can say it you can code it * Linear Model (lm) This is the equation of a linear model Data = : this the data in which you want to build the model from	model = lm (dependant ~ independant , data = training) model = lm (y ~ x , data = training) ex: model = lm (Price ~ Age , data = training)
Linear Model with all the Variables	model = lm (dependant ~ . , data = training)

Summary of the Model: reads the model - Equation you coded - Coeffiecients: (Intercept) #### Independent Variable ### - "*" indicate significance	summary (model)
Best Subset Model with only the significant variables 1. First create the model with all the variables	model5 = lm (Price ~ . , data = training) step (model5) model5 = step (model 5)
MAKING PREDICTIONS	
Take the formula from the model and run the testing data through it and stores it into predictions	predictions = predict (model ,test) 1. First run the model with all the variables (bench) 2. Second run the model with the significant variables
Observations: it is independent from all the other the data points of the dependent variable from the test data frame is being stored into observation	observations = test$Price
Finds the errors	errors = observations - predictions
Create a histogram of the errors - Histogram of the residuals	hist(errors) library (ggplot2) test$errors = errors test$observations = observations ggplot(test, aes (y=errors, x = observation, color = errors)) + geom_point() + labs (x= "Fits_Test", y = "Residuals_test" , title = "scatterplot") + scale_color_gradient (low = "blue", high = "green")

EVALUATE THE PERFORMANCE (KPI)	
Mean Average Percentage Error - Error in the Model	mape = mean (abs (df$Price – predictions) / (observations)) mape = mean (abs (errors / observations)) mape
RMSE - prediction is wrong by	rmse = sqrt (mean (df$Price – predictions) ^2) rmse = sqrt (mean (errors ^ 2)) rmse
Predictions Bench - average of the dependent data set in the training data set Errors Bench - price column in the data set and taking away the average of the dependent training data set Benchmarks are nonstatistical models	Prediction_Bench = mean (training$Price) errors_bench = observations – Prediction_Bench
RMSE BENCH MAPE BENCH	rmse_bench = sqrt (mean (errors_bench^2)) mape_bench = mean (abs (errors_bench / observations))

Logistic Regression

Notes/ Description	Code / Instructions
Data Management	
To merge two files of code into one so they run at the same time ADD THIS LINE OF CODE AT THE BEGINNING	source("C:/Users/nladda1/Desktop/BabsonAnalyticsR.R")
On the graph, it needs scaler data points which is the integer value (need to be number) In a logistical regression model you need a logical response in a binary output	ISHIGHVAL : is in integer form 1 , 0 so it can be in the model ISHIGHVAL2 : is logical True , False to make the model df$ISHIGHVAL2 = df$ISHIGHVAL df$ISHIGHVAL2 = as.logical (df$ISHIGHVAL2) ISHIGHVAL = NULL If the Variable is not already in dummy variable form then you must turn it into one 1. df$Income2=ifelse(df$Income>100000, 1, 0) 2. df$Income3=as.logical(df$Income2) 3. df$Income=NULL
Partition the Data	
Same steps as Linear Regression	1. set.seed (###) 2. N=nrow(df) 3. trainingSize = round(N * .6) 4. trainingCases = sample (N, trainingSize) 5. training = df [trainingCases ,] 6. test = df [- trainingCases,]
Build Model	
REMEMBER TO NULL OUT THE integer variable glm = general linear model - can run all types of models To build the model must use logical Family = binomial because it is categorical data	model = glm (ISHIGHVAL2 ~ ., data = training, family = binomial()) model = step (model) #this is best model summary (model)
	model = glm (ISHIGHVAL2 ~ RM, data = training , family = binomial ())
Logistical Equation	$ln\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 x_1$

ODDS = (P / 1 – P)	$$\hat{p} = \frac{e^{\beta_0} + \beta_1 x_1}{1 + e^{\beta_0 + \beta_1 x_1}}$$
KEY PERFORMANCE INDICATOR	
Plugging values into the equation + or – 1 Ex: 5, 6, 7	Five_rooms = (exp (B0 + (B1* X))) / (1 + exp (B0 + (B1*X))) Five_rooms Do the same thing for 6 rooms Then divide the Phat for 6 rooms by 5 rooms to get the e^B1
This will fit it to the s-curve Response = standardizes all the values from 0 to 1	predictions = predict (model2 , test, type = "response") predictions
Observations = has to be the logical True False version of the variable .5 is the center part of the curve	ISHIGHVAL2 1. observations = test$depenantvariable 2. PredictionsTF = predictions1 > .5 3. table (PredictionsTF , observations) observations predictionsTF FALSE TRUE FALSE 43 20 TRUE 39 100 >
Sensitivity = accuracy of the true predictions	sensitivity = sum (predictionsTF = = TRUE & observations = = TRUE) / sum(observations = = TRUE) sensitivity = 100/(100+20)
Specificity = accuracy of the false predictions	specificity = sum(predictionsTF = = FALSE & observations = = FALSE)/ sum(observations = = FALSE) specificity = 43/ (43+39)
Error rate of the model	error_rate = sum (PredictionsTF != observations) /nrow2(test) error_rate
ROCChart	ROCChart (observations, predictions) AUC – area under the curve

	3
Lift chart	liftChart (observations, predictions)
GRAPH	
To build the logistical regression graph	1. Un-NULL the integer version of the variable - Ex: ISHIGHVAL ggplot (test, aes (x = RM, y = predictions, color = ISHIGHVAL)) + geom_point () + geom_point (aes (y = ISHIGHVAL))

KNN Regression

NOTES/ DESCRIPTION	CODE / INSTRUCTIONS
DATA MANAGEMENT	
To merge two files of code into one so they run at the same time ADD THIS LINE OF CODE AT THE BEGINNING	source("C:/Users/nladda1/Desktop/BabsonAnalyticsR.R")
MUST Load package correct Package	library(caret) library(ggplot2)
Load data frame	df=read.csv("BostonHousing.csv")
Remove missing data	df=na.omit(df)
Can only have scaler data points - Remove any categorical data or logistical data	df$CHAS= NULL df$ISHIGHVAL = NULL df$RAD = NULL
standardize df- makes all of the independent variables into z-score values Normalization of df [,column# : column#] Can only do this to the independent variables – NOT dependent ** Make sure dependent variable is at the bottom of the df list**	Processor_s = preProcess(df [, 1:10], c("center", "scale")) Processor_n = preProcess(df [, 1:10], c("range")) If Dependent variable is not at the bottom df$new.name.of.dependant.variable = df$actual.name.of dependant.variable df$acutal.name.of.dependant.variable = NULL
	df1 = predict(Processor_s, df) #standadization first

Make two separate df's for the standardization and normalization	df2 = predict(Processor_n,df) #then normalization z-score standardization range [
PARTITION DATA	
PARTITION FOR DF 1 AND DF 2 SEPARATELY	df1 partition set.seed(5) N1=nrow(df1) trainingSize1= round(N1*.6) trainingCases1 = sample(N1, trainingSize1) training1 = df1[trainingCases1,] test1 = df1[-trainingCases1,] df2 partition set.seed(5) N2=nrow(df2) trainingSize2= round(N2*.6) trainingCases2 = sample(N2, trainingSize2) training2 = df2[trainingCases2,] test2 = df2[-trainingCases2,]
BUILD MODEL	
Creates a chart using BabsonAnalyticsR.file - graph shows the metric of the neighborhood	knnCrossVal(MEDV ~., data = training1) knnCrossVal(MEDV ~., data = training2)
Creates the model for kNN Regression - knnreg comes from the BabsonAnalyticsR.file - to choose k value: pick the lowest point on the chart after running it a few times	model1 = knnreg(MEDV ~ ., data = training1, k = 2) model2 = knnreg(MEDV ~ ., data = training2, k = 2)
KPI	
	predictions1 = predict(model1, test1) predictions2 = predict(model2, test2) observations1 = test1$MEDV observations2 = test2$MEDV errors1 = observations1 - predictions1 errors2 = observations2 - predictions2
RMSE 1 and RMSE 2 Lower rmse is better	rmse1 = sqrt (mean (errors1 ^ 2)) rmse2 = sqrt (mean (errors2 ^ 2))

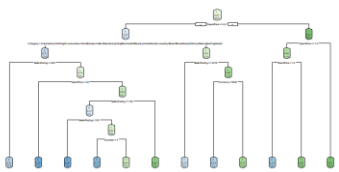
MAPE 1 and MAPE 2 Lower mape is better	mape1 = mean (abs (errors1 / observations1)) mape2 = mean (abs (errors2 / observations2))
Creating a Benchmark - values for benchmark should be the same for benchmark 1 and 2	errors_bench1 = observations1-mean(training1$MEDV) errors_bench2 = observations2-mean(training2$MEDV) mape_bench1 = mean(abs(errors_bench1/observations1)) mape_bench2 = mean(abs(errors_bench2/observations2)) rmse_bench1 = sqrt(mean(errors_bench1^2)) rmse_bench2 = sqrt(mean(errors_bench2^2))
Interpretation of the KPI	Based on the lower/higher RMSE and lower/higher MAPE of model1/model2 we can determine that standardization/normalization is better for this data
GENERAL NOTES	
• kNN : k Nearest Neighbors • neighborhood: measurement of distance around a desirable outcome – it contains the X # of points that will be used to make a prediction • k : is a metric of distance (we use the Euclidean distance formula) – the # of points in neighborhood • need to remove all none numerical variables • need to determine which method is better, standardization or normalization by testing both options out and evaluating the RMSE and MAPE of each • CAN NOT run standardization and normalization method at the same time that is why you must create a two sperate dfs for them • Regression uses numerical values only • MAPE should never be over 1 – mean there is a 100% error that the prediction is wrong by – indicating a poor model • Summary(model) is irrelevant because we are using KPIs to determine if the model is good or bad • If value of K is too large – end up using entire data to predict one specific point not accurate • If value of K is too small – only using 1 point then allows for guessing and randomization to effect the model • Want k to be relatively narrow / appropriate • ** CAN NOT cross check with KPIs to pick the value for K because leakage could occur between training and testing data when we are not using a seed **	

KNN Classification

NOTES/ DESCRIPTION	CODE / INSTRUCTIONS
DATA MANAGEMENT	
	source("C:/Users/nladda1/Desktop/BabsonAnalyticsC (1).R")
Install package	library(ggplot2) library(caret)
Remove missing data always	df3=na.omit(df3)
Dependent Variable – is categorical – make into factor	To turn variable in to a categorical df3$price1 = ifelse(df3$price>540000,1,0)

Independent Variable – is scaler- NULL all the variables that are not scaler	
Partition Data	
	set.seed(12) N3 = nrow(df3) trainingSize3 = round(N3*.6) trainingCases3= sample(N3,trainingSize3) training3 = df3[trainingCases3,] test3 = df3 [-trainingCases3,]
Build Model	
kNNcrossval will give k value	kNNCrossVal(price1 ~ .,training3) predictions3 = kNN(price1~ ., training3,test3, k= 16) observations3 = test3$price1
KPI	
Correctly classifies 0 ((0,0)/(0,0)+(0,1)% of the time Correctly classifies 1 ((1,1)/(1,1)+(1,0)% of the time	table(predictions3, observations3)
Accuracy of the false predictions	error_rate3= sum(predictions3 != observations3)/nrow(test3) error_rate3
Accuracy of the true predictions	error_rate4= 1-sum(predictions3 == observations3)/nrow(test3) error_rate4
	error_bench3 = benchmarkErrorRate(training3$price1, test3$price1) predictions_no_standardization = kNN(price1 ~ ., training3, test3, k=16 , standardize = FALSE) error_rate_no_standardization = sum(predictions_no_standardization != observations3)/nrow(test3)
	error_rate3 #the model is wrong 30.7% of the time error_bench3 #The error bench is 51.2% error_rate_no_standardization #the error without standardization is 28% #The model without standardization is better than the model with standardization
General Notes	
• In regression Knn you are using scaler data points so you can actually calculate the rmse and mape however with KNN classification you are using a categorical dependant variable so you can't calculate the rmse or mape • kNN regression is for scaler data only where as kNN Classification is for predicting categorical data	

CART

NOTES/ DESCRIPTION	CODE / INSTRUCTIONS
DATA MANAGEMENT	
Install Babson Analytics C file	source("C:/Users/nladda1/Desktop/BabsonAnalyticsC (1).R")
Turn on Library Packages	library(rpart) library(rpart.plot)
Partition Data	
	set.seed(12) n = nrow(df) trainingsize = round(n*.6) trainingcases = sample(n, trainingsize) training = df[trainingcases,] test = df[-trainingcases,]
Build Model	
 #Leaf node is the last node in the diagram # the right most leaf node is the strongest because 91% of this leaf node is competitive out of 21% of the data set # the left most leaf node is also strong because 24% of the leaf node is competitive out of 34% of the data - the majority however is not competitive # this first diagram is not good because there are some leaf nodes with only 1% of the data set -- ideally we would want it to have 3% or more	model = rpart(Competitive ~., data=training) rpart.plot(model)
Makes the model a bit easier to read	rpart.plot(model, tweak = 1.1, varlen = 4 , faclen = 4)
type = class - says don't give me the numbers give me the factor level back	predictions = predict(model, test, type = "class") observations = test$Competitive
# Athough our model has a much lower error rate than the benchmark it is important to note that overcomplextiy is present becasue we have leaf nodes at 1% , 2%, 3% - menaing th model has to many splits for this respected data set	error_rate = sum(predictions != observations)/nrow(test) error_bench = benchmarkErrorRate(training$Competitive, test$Competitive)
# Stopping rules # model complexity is the number of splits in the model -- the more splits in the model the highr the chances are for overfitting # minplit: how data is left in the leaf node -- is the node big enough to split, typically	stopping_rules = rpart.control(minsplit=2, minbucket=1, cp=0) # overfitted model: this mean only 2 observations needed in each leaf node and only 1 observation needed to be different # kinda have to guess and check here but try to get the nodes to 3% atleast

something like 2% of the total number of records --- # minbucket: comparison of the split itself -- when you split, the split should be big enough (on both side) of the split. something like 1% of the total number of records # Complexity parameter: CP : does the split improve the model (many factors go into this) (in this class always 0)	model = rpart(Competitive ~., data=training, control = stopping_rules) rpart.plot (model)
	predictions_overfit = predict(model, test, type="class") error_rate_overfit = sum(predictions_overfit != observations)/nrow(test)
	model = easyPrune(model) rpart.plot(model)
	predictions_pruned = predict(model, test, type="class") error_rate_pruned = sum(predictions_pruned!= observations)/nrow(test)

General Notes

- The strongest lead node is the one where the middle # is the closest to 0 or 1
  - Close to 0 means you should not do the action
  - Close to 1 means you should do the action
- The minsplit option specifies the minimum number of observations that must exist in a node in order for a split to be attempted.
- The minbucket option specifies the minimum number of observations allowed in any terminal (leaf) node.

Naïve Bayes

NOTES/ DESCRIPTION	CODE / INSTRUCTIONS
DATA MANAGEMENT	
MUST ADD THE SOURCE	source("C:/Users/nladda1/Desktop/BabsonAnalyticsC (1).R") ** there is a space between C and (1) ** df2= read.csv ("MovieReviews.csv")
Every variable needs to be a factor so that it is in the form of 0 and 1	every_column = colnames(df) df[every_column] = lapply(df[every_column], as.factor)
Turn on the needed library package	library(e1071)
PARTITION DATA	
Partition the data into testing and training	set.seed(33) n2=nrow(df2) trainingsize2 = round(n2*.6) trainingcases2= sample(n2, trainingsize2)

	training2 = df2[trainingcases2,] test2 = df2[-trainingcases2,]
BUILD MODEL	
From the Babson Analytics C file use naiveBayes code to build model	model2 = naiveBayes(PositiveTweet ~ ., data = training2)
Predictions	predictions2 = predict(model2, test2)
Observations	observations2 = test2$PositiveTweet
Error rate	error_rate2 = sum(predictions2 != observations2)/nrow(test2) error_rate2
Error Bench	error_bench2 = benchmarkErrorRate(training2$PositiveTweet, test2$PositiveTweet) error_bench2
INTERPRETATIONS	
Build Table Observation / predictions table	table(predictions2, observations2) ``` > table(predictions2, observations2) observations2 predictions2 0 1 0 1122 25 1 62 1625 ``` >
Table to test the Likelihood	model2$tables$awesome ``` > model2$tables$awesome awesome Y 0 1 0 1.0000000 0.0000000 1 0.7292111 0.2707889 ``` (1:1) : given that the statement is positive there is a 27% chance the word awesome is present (1: 0) : given that the statement is positive there is a 72% chance the word awesome is not present (0,1) : given that the statement is not positive there is a 0% chance the word awesome is present (0,0) given that the statement is not positive there is a 100% chance the word awesome is not present Y axis : conditional statement is not present (0) conditional statement is present (1) Awesome: Not Present (0) Present (1) GIVEN is going to be from the model of what the dependent variable/ conditional statement is (ex: positive statement or not)
Create a new data frame that pulls all the statements that have the word in it	limited2 = df2[df2$awesome == "1",] ** note the number of observations that have the word Ex: the word awesome appears 1123 times

The odds of the condition in the scenario	odds2 = sum(limited2$PositiveTweet == "1")/ sum(limited2$PositiveTweet == "0") odds2 for every 1122 positive tweets that contains the word awesome there is 1 negative tweet that contains the word awesome it is a ratio 1122 : 1 inf : means condition is always in that scenario zero : means conditions is never in that scenario

GENERAL NOTES

- NB works directly with conditional probability → something already exists we are seeing the likely hood of the action occurring
  - Ex: given that it is a rainy day what are the odds you have a test
- Issues with NB when measuring the sensitivity of a string of sentences
  - In theory we would need to build out the entire English language including slang, abbreviations, dif variations of words, special characters, emojis (how would the algorithm know the difference between aww and awwww)
  - Every single piece of data needs to be in 1's and 0's → all conditions → 0 = not present 1= present
- The dependent variable (aka conditional probability statement) does not need to be first in the data set → make sure it is in the appropriate location of the model section of code
- STEPS
  - Change all the columns to factor → NB algorithm will not work with numbers
  - Summary table is not useful because it is not a model with a graph associated because it is all dealing with conditional probabilities and table associations
  - Benchmarks comparing training and test data to see when they match up without any sort of statistical model
    - Training data is observable observations
    - Testing data is the model you created for the benchmark

Clustering

NOTES/ DESCRIPTION	CODE / INSTRUCTIONS
DATA MANAGEMENT	
MUST ADD THE SOURCE	source("C:/Users/nladda1/Desktop/BabsonAnalyticsC (1).R")
#This type a of data is ordinal - it is an opinion-based rating system - not reliable #normal curve of distribution - symmetric, asymtotic, normal - 50% of data on either side	library (caret) df = read.csv("utilities.csv") df$Company = NULL – null because it is a chr / has words df= na.omit(df) elbowChart(df)

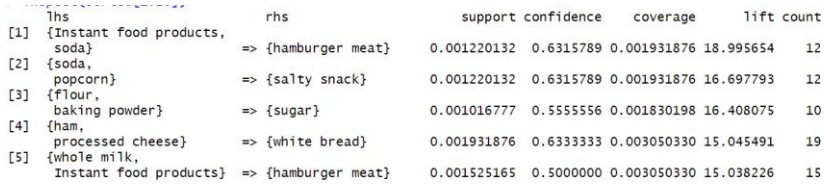
#read the cusp of the curve and that would tell you what the cluster size should be #cluster of 1 is the whole data set is in it so it is not useful which is not useful #too many clusters means every data set is its on cluster #y axis - is the sum of sqares which is the variance between the groups #knn has a target value and clustering analysis does not - supervised vs unsupervised #this elbow chart should come first so you can determine how many there should be	standardizer = preProcess(df, method = c("center", "scale")) df2= predict (standardizer, df) model = kmeans(df2, 4) model$centers model$size #If there is an outlier in the model it would just create its own cluster d = dist(df2) d model = hclust(d, method= "average") plot(model, labels = df2$Company) model = hclust(d, method= "single") plot(model, labels = df2$Company) model = hclust(d, method= "complete") plot(model, labels = df2$Company)

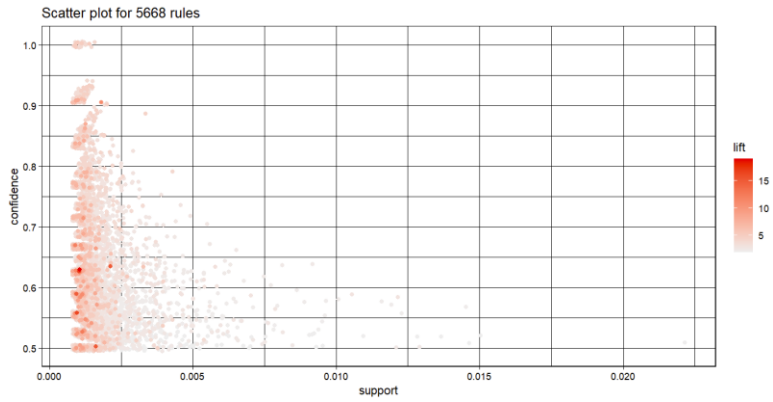
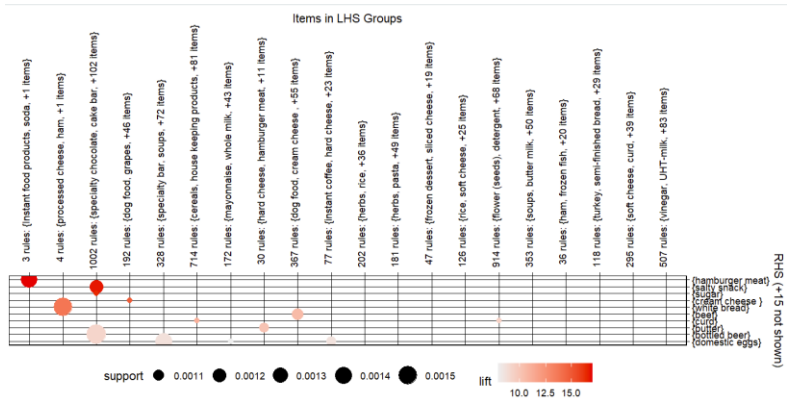
Association Rules

BUILD MODEL	
NOTES/ DESCRIPTION	CODE / INSTRUCTIONS
	install.packages("arules") install.packages("arulesViz") library(arules) library(arulesViz) data("Groceries") itemFrequencyPlot(Groceries) itemFrequencyPlot(Groceries, topN=5) ##33 rules = apriori(Groceries) #The default of 10% and 80% or what's being used for support and confidence summary(rules)#note, no rules shown ##

```
rules = apriori(Groceries, parameter =
list(support=0.001,confidence=0.5))
summary(rules)# now we have 5668 rules



#rule length distribution (lhs + rhs):sizes
#2    3    4    5    6      2 means Item 1 --> item 2, 3 means item
1&2 --> item 3, 4 means item 1&2&3 --> item 4 ...
#11 1461 3211  939   46     how many are in each grouping
#####################################################

#Once the support and confidence thresholds are met choose the
one with
#the strongest lift.  Lift is how often we see the given scenario of
the
#conditional probability and the confidence together relative to
how often
#we would see them together if they were independent

rules = apriori(Groceries, parameter =
list(support=0.001,confidence=0.5))
sorted = sort(rules, by="lift")
inspect(sorted[1:10])
summary(rules)

#####################################################
###########part 2 ################################

?apriori # look at the bottom for the rules threshold
rules = apriori(Groceries, parameter = list(support=0.1))
summary(rules)

rules = apriori(Groceries, parameter = list(support=0.01))
summary(rules)

rules = apriori(Groceries, parameter = list(support=0.001))
summary(rules)

rules = apriori(Groceries, parameter =
list(support=0.001,confidence=0.5))
summary(rules)

sorted = sort(rules, by="lift")
inspect(sorted[1:20])

plot(rules)
```

	plot(rules, method="grouped") plot(sorted[1:10], method = "graph")

Sort new rules by highest lift values	
	lhs rhs support confidence coverage lift count
	[1] {Instant food products, soda} => {hamburger meat} 0.001220132 0.6315789 0.001931876 18.995654 12
	[2] {soda, popcorn} => {salty snack} 0.001220132 0.6315789 0.001931876 16.697793 12
	[3] {flour, baking powder} => {sugar} 0.001016777 0.5555556 0.001830198 16.408075 10
	[4] {ham, processed cheese} => {white bread} 0.001931876 0.6333333 0.003050330 15.045491 19
	[5] {whole milk, Instant food products} => {hamburger meat} 0.001525165 0.5000000 0.003050330 15.038226 15
	
	rules= apriori(Groceries, parameter = list(support = 0.001, confidence = 0.5))
	sorted = sort(rules, by = "lift") #sorting by lift
	inspect(sorted[1:10]) #inspect top 10
	summary(rules)

CREATE GRAPHS	
Graph of strong lift values	plot(rules) 
View rules	plot(rules, method = "grouped") #rules 

Graph that shows how strong lift items are purchased together	plot(sorted[1:10], method = "graph") #shows the diagram with arrows, double ended arrows means purchased together, listing out the strong lifts