CHICAGO BOOTH

The University of Chicago Booth School of Business

Machine Learning Winter 2020

Midterm

Hikaru Sugimori

PROBLEM #1

#Install and load packages needed

library(tidyverse)
library(caret)
library(ROCR)
library(rpart)
library(dplyr)
library(rpart.plot)
library(ggplot2)
library(class)
UniversalBank <- read_csv('UniversalBank.csv')
set.seed(123)
install.packages('dendextend')
install.packages('factoextra')
library("dendextend")
library(cluster)
suppressPackageStartupMessages(library(dendextend))
library(factoextra)
install.packages("fpc")
install.packages("NbClust")
library(fpc)
library(NbClust)

names(UniversalBank)

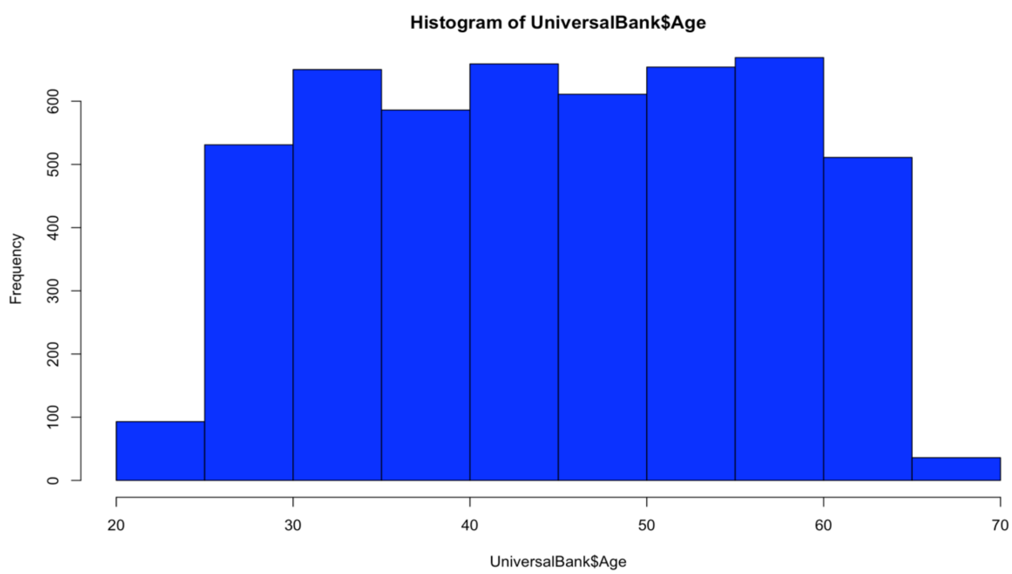
```
[1] "ID"                 "Age"                "Experience"
 [4] "Income"             "ZIP Code"           "Family"
 [7] "CCAvg"              "Education"          "Mortgage"
[10] "Personal Loan"      "Securities Account" "CD Account"
[13] "Online"             "CreditCard"
```

str(UniversalBank)

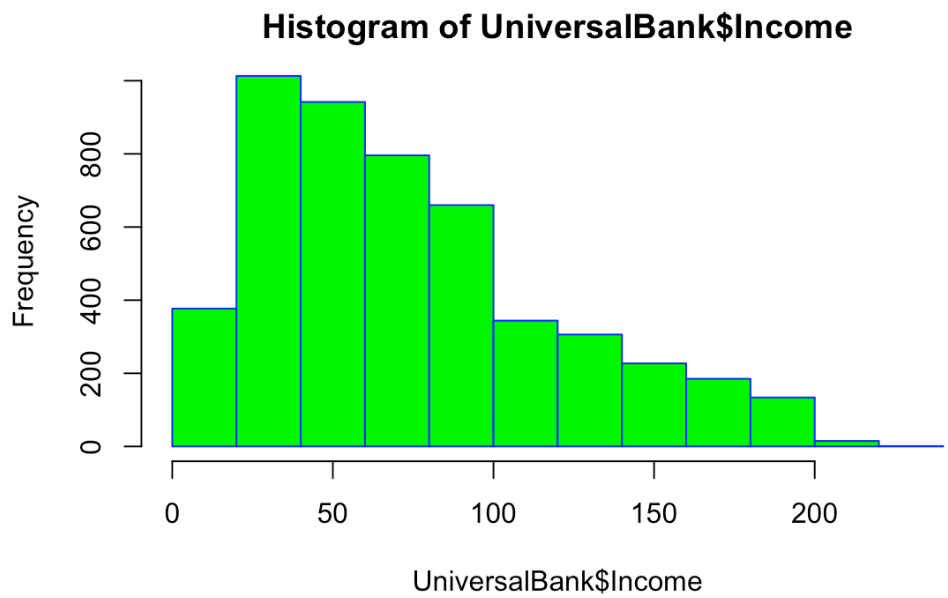
```
Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 5000 obs. of  14 variables:
 $ ID                : num  1 2 3 4 5 6 7 8 9 10 ...
 $ Age               : num  25 45 39 35 35 37 53 50 35 34 ...
 $ Experience        : num  1 19 15 9 8 13 27 24 10 9 ...
 $ Income            : num  49 34 11 100 45 29 72 22 81 180 ...
 $ ZIP Code          : num  91107 90089 94720 94112 91330 ...
 $ Family            : num  4 3 1 1 4 4 2 1 3 1 ...
 $ CCAvg             : num  1.6 1.5 1 2.7 1 0.4 1.5 0.3 0.6 8.9 ...
 $ Education         : num  1 1 1 2 2 2 2 3 2 3 ...
 $ Mortgage          : num  0 0 0 0 0 155 0 0 104 0 ...
 $ Personal Loan     : num  0 0 0 0 0 0 0 0 0 1 ...
 $ Securities Account: num  1 1 0 0 0 0 0 0 0 0 ...
 $ CD Account        : num  0 0 0 0 0 0 0 0 0 0 ...
 $ Online            : num  0 0 0 0 0 1 1 0 1 0 ...
```

```
 $ CreditCard        : num  0 0 0 0 1 0 0 1 0 0 ...
 - attr(*, "spec")=
  .. cols(
  ..   ID = col_double(),
  ..   Age = col_double(),
  ..   Experience = col_double(),
  ..   Income = col_double(),
  ..   `ZIP Code` = col_double(),
  ..   Family = col_double(),
  ..   CCAvg = col_double(),
  ..   Education = col_double(),
  ..   Mortgage = col_double(),
  ..   `Personal Loan` = col_double(),
  ..   `Securities Account` = col_double(),
  ..   `CD Account` = col_double(),
  ..   Online = col_double(),
  ..   CreditCard = col_double()
  .. )
```

#Let's explore this data visually first.
hist(UniversalBank$Age, border = "black", col = "blue")



hist(UniversalBank$Income, border = "blue", col = "green")




```
# As instructed, we will use a 60/40 split for training and validation
trainIndex <- createDataPartition(UniversalBank$`Personal Loan`, p=.6,
                                  list = FALSE,
                                  times = 1)
#Cleaning dataset
#Exclude some columns we don't need and normalize numeric data
UniversalBank.train <- UniversalBank[trainIndex,c(-1,-5)]
UniversalBank.valid <- UniversalBank[-trainIndex,c(-1,-5)]
UniversalBank.trainnorm <- UniversalBank.train[, c(1:3,5,7)]
UniversalBank.trainnorm.z <- as.data.frame(scale(UniversalBank.trainnorm))
UniversalBank.validnorm <- UniversalBank.valid[, c(1:3,5,7)]
UniversalBank.validnorm.z <- as.data.frame(scale(UniversalBank.validnorm))
train.knn <- cbind(UniversalBank.trainnorm.z, UniversalBank.train$`Personal Loan`)
names(train.knn)
```

```
[1] "Age"                                    "Experience"
[3] "Income"                                 "CCAvg"
[5] "Mortgage"                               "UniversalBank.train$`Personal Loan`"
```

```
summary(train.knn)
```

```
      Age              Experience            Income             CCAvg
 Min.   :-1.96925   Min.   :-2.03259   Min.   :-1.4286   Min.   :-1.1157
 1st Qu.:-0.82784   1st Qu.:-0.89392   1st Qu.:-0.7514   1st Qu.:-0.7098
 Median : 0.05016   Median :-0.01801   Median :-0.2053   Median :-0.2459
 Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.0000
 3rd Qu.: 0.84037   3rd Qu.: 0.85789   3rd Qu.: 0.4720   3rd Qu.: 0.3921
 Max.   : 1.89398   Max.   : 1.99656   Max.   : 3.2901   Max.   : 4.6835
    Mortgage        UniversalBank.train$`Personal Loan`
 Min.   :-0.5477   Min.   :0.00000
 1st Qu.:-0.5477   1st Qu.:0.00000
 Median :-0.5477   Median :0.00000
 Mean   : 0.0000   Mean   :0.09267
```

```
 3rd Qu.: 0.4231   3rd Qu.:0.00000
 Max.   : 5.7429   Max.   :1.00000
```

valid.knn <- cbind(UniversalBank.validnorm.z, UniversalBank.valid$`Personal Loan`)
names(valid.knn)

```
[1] "Age"                                    "Experience"
[3] "Income"                                 "CCAvg"
[5] "Mortgage"                               "UniversalBank.valid$`Personal Loan`"
```

summary(valid.knn)

```
Age             Experience            Income            CCAvg
 Min.   :-1.91828   Min.   :-1.988048   Min.   :-1.4290   Min.   :-1.0991
 1st Qu.:-0.88151   1st Qu.:-0.862068   1st Qu.:-0.7613   1st Qu.:-0.7064
 Median :-0.01754   Median : 0.004071   Median :-0.2443   Median :-0.2014
 Mean   : 0.00000   Mean   : 0.000000   Mean   : 0.0000   Mean   : 0.0000
 3rd Qu.: 0.84643   3rd Qu.: 0.870210   3rd Qu.: 0.5527   3rd Qu.: 0.3036
 Max.   : 1.88320   Max.   : 1.996190   Max.   : 2.7929   Max.   : 4.5118
    Mortgage        UniversalBank.valid$`Personal Loan`
 Min.   :-0.5670   Min.   :0.000
 1st Qu.:-0.5670   1st Qu.:0.000
 Median :-0.5670   Median :0.000
 Mean   : 0.0000   Mean   :0.101
 3rd Qu.: 0.4344   3rd Qu.:0.000
 Max.   : 5.2762   Max.   :1.000
```

#Now we will create a KNN model
#0 means No and 1 means Yes
train.knn.predictors <- train.knn[, 1:5]
train.knn.target <- train.knn[,6]

valid.knn.predictors <- valid.knn[, 1:5]
valid.knn.target <- valid.knn[,6]

set.seed(123)
preds.k.1 <- knn (train=train.knn.predictors, test=valid.knn.predictors, cl=train.knn.target, k=1, prob=TRUE)

#Now let's evaluate the KNN model.
#The KNN model's performance is subtoptimal, it commits many Type 1 and Type 2 errors.
#The results for 1 fold, 3 fold, and 5 fold cross validation all had similar results.
confusionMatrix(table(preds.k.3, valid.knn.target))

```
Confusion Matrix and Statistics

         valid.knn.target
preds.k.3    0    1
        0 1736  118
        1   62   84
```

```
               Accuracy : 0.91
                 95% CI : (0.8966, 0.9222)
    No Information Rate : 0.899
    P-Value [Acc > NIR] : 0.05351

                  Kappa : 0.4349

 Mcnemar's Test P-Value : 4.141e-05

            Sensitivity : 0.9655
            Specificity : 0.4158
         Pos Pred Value : 0.9364
         Neg Pred Value : 0.5753
             Prevalence : 0.8990
         Detection Rate : 0.8680
   Detection Prevalence : 0.9270
      Balanced Accuracy : 0.6907

       'Positive' Class : 0
```

options(scipen = 999)
logit.reg <- glm(UniversalBank.train$`Personal Loan` ~., data = UniversalBank.train, family = "binomial")
summary(logit.reg)

```
Call:
glm(formula = UniversalBank.train$`Personal Loan` ~ ., family = "binomial",
    data = UniversalBank.train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.0507  -0.2071  -0.0848  -0.0313   3.5932

Coefficients:
                        Estimate  Std. Error z value            Pr(>|z|)
(Intercept)          -11.4366626   2.0840548  -5.488       0.00000004072 ***
Age                   -0.0615230   0.0777453  -0.791             0.42875
Experience             0.0604845   0.0770791   0.785             0.43263
Income                 0.0539786   0.0033673  16.030 < 0.0000000000000002 ***
Family                 0.5707496   0.0944417   6.043       0.00000000151 ***
CCAvg                  0.0635946   0.0526481   1.208             0.22708
Education              1.8306206   0.1550091  11.810 < 0.0000000000000002 ***
Mortgage               0.0005397   0.0007251   0.744             0.45666
`Securities Account`  -0.6583435   0.3447701  -1.910             0.05620 .
`CD Account`           4.0049605   0.4067795   9.846 < 0.0000000000000002 ***
Online                -0.6421412   0.2039473  -3.149             0.00164 **
CreditCard            -1.2683118   0.2725820  -4.653       0.00000327212 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1851.99  on 2999  degrees of freedom
Residual deviance:  775.53  on 2988  degrees of freedom
AIC: 799.53

Number of Fisher Scoring iterations: 8
```

exp(cbind(Odds=coef(logit.reg)))

```
  Odds
(Intercept)           0.00001079246
Age                   0.94033131384
Experience            1.06235113021
Income                1.05546202211
Family                1.76959307547
CCAvg                 1.06566029998
Education             6.23775690561
Mortgage              1.00053988374
`Securities Account`  0.51770819935
`CD Account`         54.86965434975
Online                0.52616459246
CreditCard            0.28130613331
```

#The logistic regression model does a lot better than the KNN model.
logit.reg.pred <- predict(logit.reg, UniversalBank.valid, type="response")
logit.reg.pred.cat <- ifelse(logit.reg.pred>0.5, 1,0)
logit.reg.pred.cat <- as.factor(logit.reg.pred.cat)
table(pred)
confusionMatrix(table(logit.reg.pred.cat, valid.knn.target))

```
Confusion Matrix and Statistics

                  valid.knn.target
logit.reg.pred.cat    0    1
                 0 1778   71
                 1   20  131

               Accuracy : 0.9545
                 95% CI : (0.9444, 0.9632)
    No Information Rate : 0.899
    P-Value [Acc > NIR] : < 0.00000000000000022

                  Kappa : 0.7178

 Mcnemar's Test P-Value : 0.0000001593

            Sensitivity : 0.9889
            Specificity : 0.6485
         Pos Pred Value : 0.9616
         Neg Pred Value : 0.8675
             Prevalence : 0.8990
         Detection Rate : 0.8890
   Detection Prevalence : 0.9245
      Balanced Accuracy : 0.8187

       'Positive' Class : 0
```

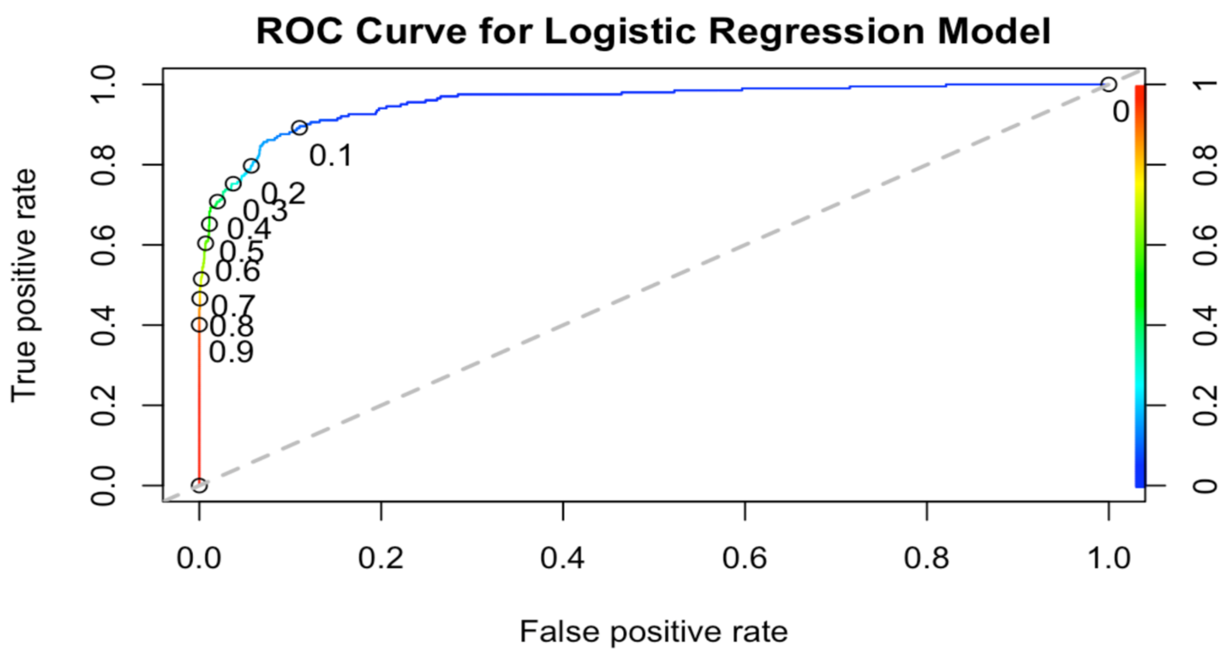
#Our ROC curve for the lostic regression also shows that our model is pretty accurate.

#Receiver Operating Characteristic Curve (ROC) is a standard technique for summarizing classifier performance over a range of trade-offs between true positive (TP) and false positive (FP) error rates
#It shows the tradeoff between sensitivity and specificity (any increase in sensitivity will be accompanied by a decrease in specificity).
The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test.

The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

#Let's visualize the ROC curve of this logistic regression

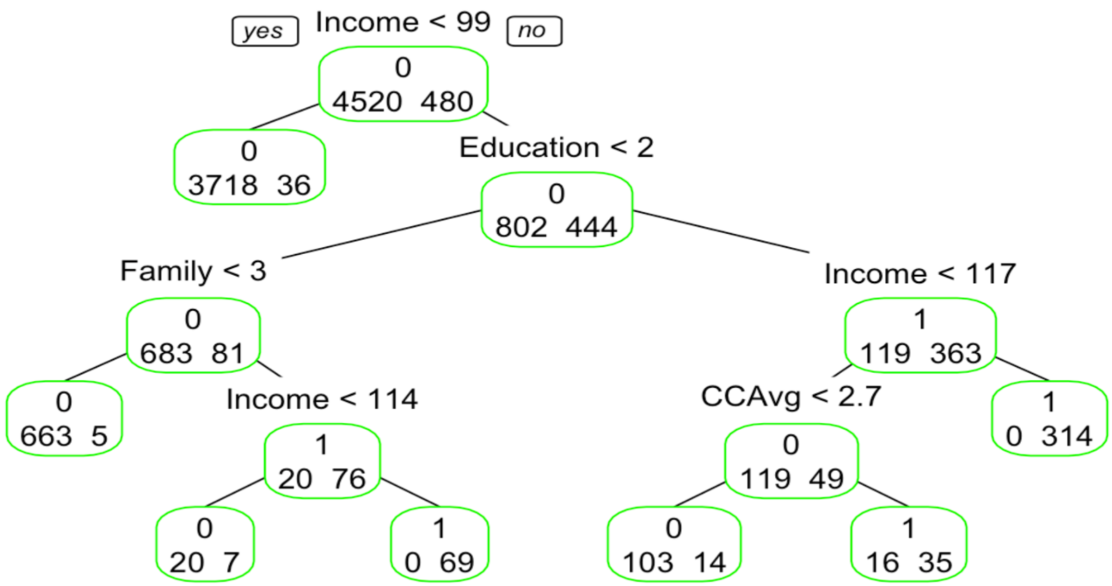
```
pred_logit <- prediction(logit.reg.pred, UniversalBank.valid$`Personal Loan`)
perf_logit <- performance(pred_logit, "tpr", "fpr")
plot(perf_logit, colorize=TRUE, print.cutoffs.at=seq(0,1,by=0.1), text.adj=c(-0.2,1.7),
    main = "ROC Curve for Logistic Regression Model")
abline(a=0,b=1,lwd=2,lty=2,col="gray")
```




```
data.train <- UniversalBank[1:3500,]
data.valid <- UniversalBank[3501:5000,]
prop.table(table(data.train$`Personal Loan`))
```

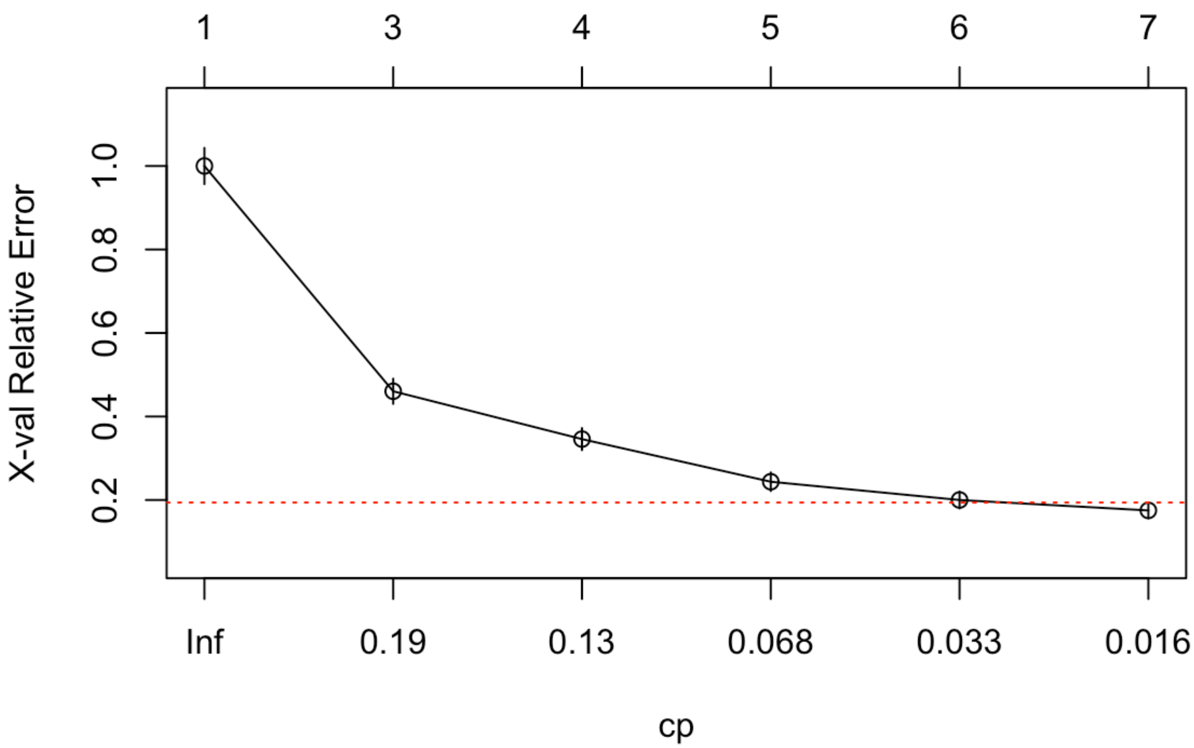
```
        0         1 
0.8988571 0.1011429 
```

```
#Now I will will build a decision tree model
data.rpart <- rpart(`Personal Loan` ~., data =  UniversalBank, method="class",
                parms=list(split="information"),
                control=rpart.control(minsplit = 1))

prp(data.rpart, type=1, extra=1, split.font=1, varlen = -10, border = "green")
```



cptable <- printcp(data.rpart)

```
Classification tree:
rpart(formula = `Personal Loan` ~ ., data = UniversalBank, method = "class",
    parms = list(split = "information"), control = rpart.control(minsplit = 1))

Variables actually used in tree construction:
[1] CCAvg     Education Family    Income

Root node error: 480/5000 = 0.096

n= 5000

        CP nsplit rel error  xerror     xstd
1 0.254167      0   1.00000 1.00000 0.043397
2 0.145833      2   0.49167 0.46042 0.030279
3 0.116667      3   0.34583 0.34583 0.026393
4 0.039583      4   0.22917 0.24375 0.022269
5 0.027083      5   0.18958 0.20000 0.020216
6 0.010000      6   0.16250 0.17500 0.018933


plotcp(data.rpart, minline=TRUE, col="red")
```



#The confusion matrix shows that the decisions tree model is the most accurate of all of our models.
rpart.pred <- predict(data.rpart, data.valid, type="class")
confusionMatrix(table(rpart.pred, data.valid$`Personal Loan`))

```
Confusion Matrix and Statistics


rpart.pred    0    1
         0 1368   15
         1    6  111

               Accuracy : 0.986
                 95% CI : (0.9787, 0.9913)
    No Information Rate : 0.916
    P-Value [Acc > NIR] : < 0.0000000000000002

                  Kappa : 0.906

 Mcnemar's Test P-Value : 0.08086

            Sensitivity : 0.9956
            Specificity : 0.8810
         Pos Pred Value : 0.9892
         Neg Pred Value : 0.9487
             Prevalence : 0.9160
         Detection Rate : 0.9120
   Detection Prevalence : 0.9220
      Balanced Accuracy : 0.9383
```

```
       'Positive' Class : 0
```

```
I apologize, I had some technical difficulties saving my predictions into a dataset.
```

PROBLEM #2

a) describe the confusion matrix to your boss;
#A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.
#It's also a good way to see how many Type 1 and Type 2 errors the classifier makes.

b) describe how you will fill out the confusion matrix for the consultant’s model;
#I will fill out the confusion matrix by creting a table with 2 rows and columns. The rows will say "true negative" and "true positive" and the columns will say, "predicted negative" and predicted "positive". Then I will fill in where each predicition fell in the matrix. Ideally there will be no deviance from predicted results and actual results.

c) describe the cost/benefit matrix for this problem;
#The cost benefit matrix will be created by taking into account how costly Type 1 and Type 2 errors are that the model makes. very low.

d) explain briefly why the confusion matrix and the cost/benefit matrix are important for this problem (1-2sentences);
#This is important because we need to make sure that the model is minimizing the kind of errors that will be very costly to us. If this is not the case, the added value of the consultant's model will be

#e) show the proper evaluation function (equation) for the consultant’s model
#The consultant's model should be evaluated in the following way:

#Added value of the model = Benefit of being right* X likelihood of being right* - cost of being wrong*X likelihood of being wrong

#f) how do the confusion and cost matrices come into play in this function.
#Both are taken into account in my function. The evaluation function is the sum of the products of the confusion and cost matrices.

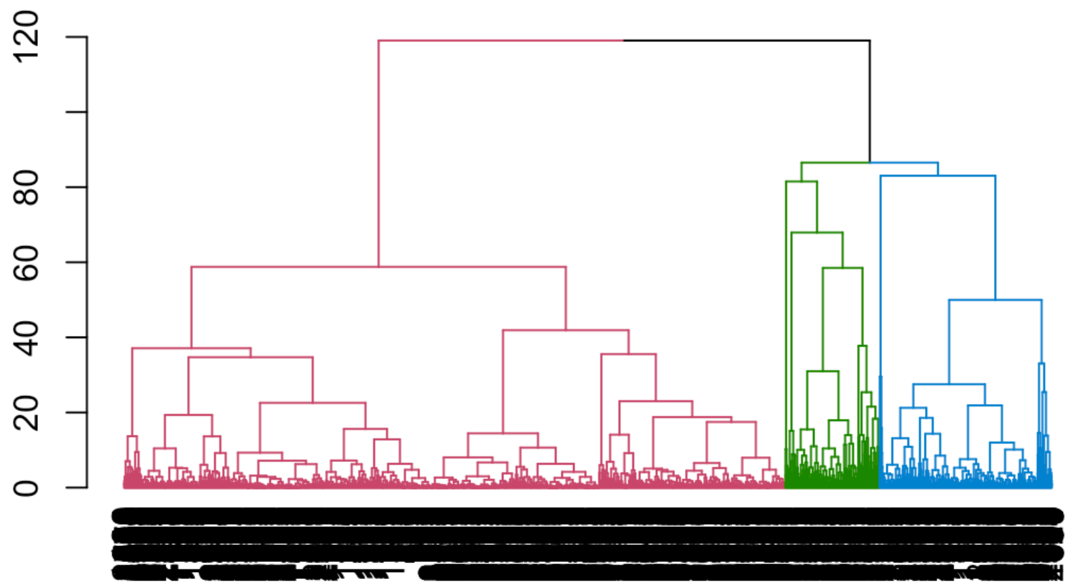
PROBLEM #3

AirLine_DF = read.csv("EastWestAirlines.csv")
str(AirLine_DF)

```
'data.frame':      3999 obs. of  12 variables:
 $ ID.             : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Balance         : int  28143 19244 41354 14776 97752 16420 84914 20856 443003 104860 ...
 $ Qual_miles      : int  0 0 0 0 0 0 0 0 0 0 ...
 $ cc1_miles       : int  1 1 1 1 4 1 3 1 3 3 ...
 $ cc2_miles       : int  1 1 1 1 1 1 1 1 2 1 ...
 $ cc3_miles       : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Bonus_miles     : int  174 215 4123 500 43300 0 27482 5250 1753 28426 ...
```

```
 $ Bonus_trans      : int  1 2 4 1 26 0 25 4 43 28 ...
 $ Flight_miles_12mo: int  0 0 0 0 2077 0 0 250 3850 1150 ...
 $ Flight_trans_12  : int  0 0 0 0 4 0 0 1 12 3 ...
 $ Days_since_enroll: int  7000 6968 7034 6952 6935 6942 6994 6938 6948 6931 ...
 $ Award.           : int  0 0 0 0 1 0 0 1 1 1 ...
```

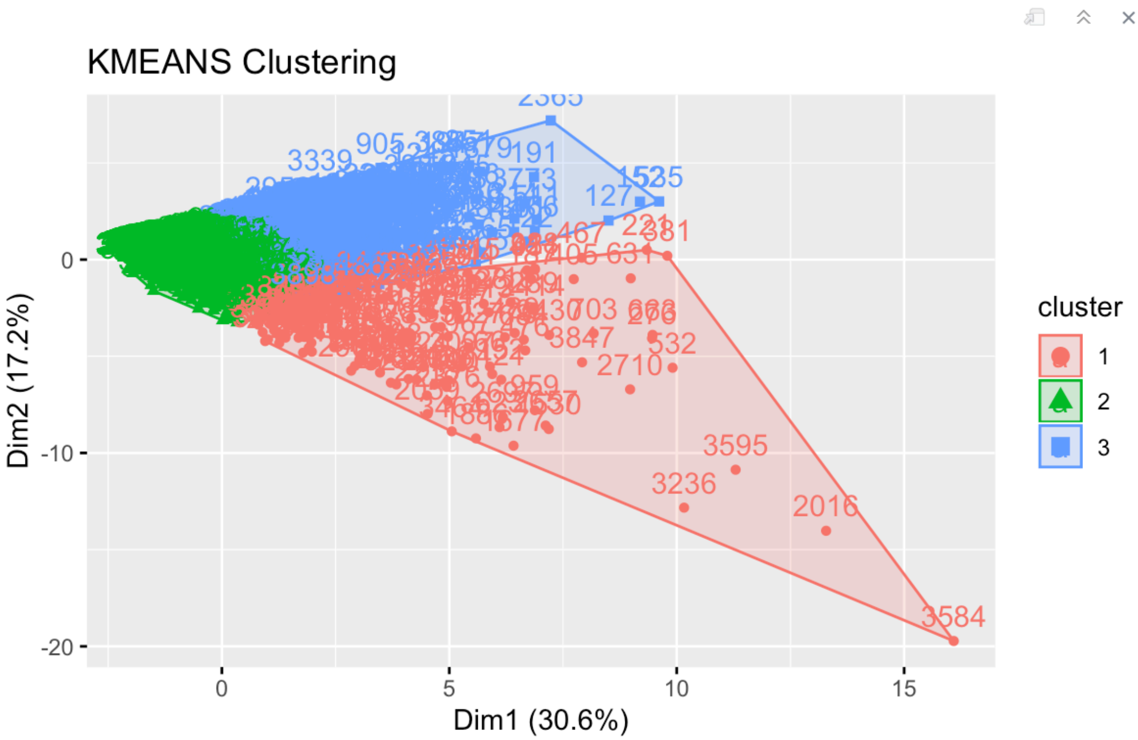
```
#There are several columns in are data that are categorical not numeric.
#We need to convert these to numeric because classifiers calculate the distance between two points by the Euclidean distance

AirLine_DF$cc1_miles = ifelse(AirLine_DF$cc1_miles==1,2500,
                          ifelse(AirLine_DF$cc1_miles==2,7500,
                              ifelse(AirLine_DF$cc1_miles==3,17500,
                                  ifelse(AirLine_DF$cc1_miles==4,32500,
                                      ifelse(AirLine_DF$cc1_miles==5,50000,0)))))

AirLine_DF$cc2_miles = ifelse(AirLine_DF$cc2_miles==1,2500,
                          ifelse(AirLine_DF$cc2_miles==2,7500,
                              ifelse(AirLine_DF$cc2_miles==3,17500,
                                  ifelse(AirLine_DF$cc2_miles==4,32500,
                                      ifelse(AirLine_DF$cc2_miles==5,50000,0)))))

AirLine_DF$cc3_miles = ifelse(AirLine_DF$cc3_miles==1,2500,
                          ifelse(AirLine_DF$cc3_miles==2,7500,
                              ifelse(AirLine_DF$cc3_miles==3,17500,
                                  ifelse(AirLine_DF$cc3_miles==4,32500,
ifelse(AirLine_DF$cc3_miles==5,50000,0)))))


#Normalize data  with Mean=0 and SD=1
#Normalization is done to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values.
#Basically we normalize because if scales for different features are wildly different, this can distort our model. Ensuring standardised feature values implicitly weights all features equally in their representation.
data = scale(AirLine_DF)
d <- dist(data[,2:11], method = "euclidean")
#Now let's visualize the data to find patterns.
fit <- hclust(d, method="ward.D2")
fit <- as.dendrogram(fit)
cd = color_branches(fit,k=3) #Coloured dendrogram branches
plot(cd)
```




```
groups <- cutree(fit, k=3)
g1 = aggregate(AirLine_DF[,2:11],list(groups),median)
data.frame(Cluster=g1[,1],Freq=as.vector(table(groups)),g1[,-1])
#Based on  this preliminary exploratory analysis we can already characterize these 3 clusters.
# Cluster1 has the most observations. These are new customers with low balance and bonus miles.
# Cluster3 has fewer observations. These are old customers with both high balance and bonus miles.
#Cluster2 is in between cluster 1 and 2.
```

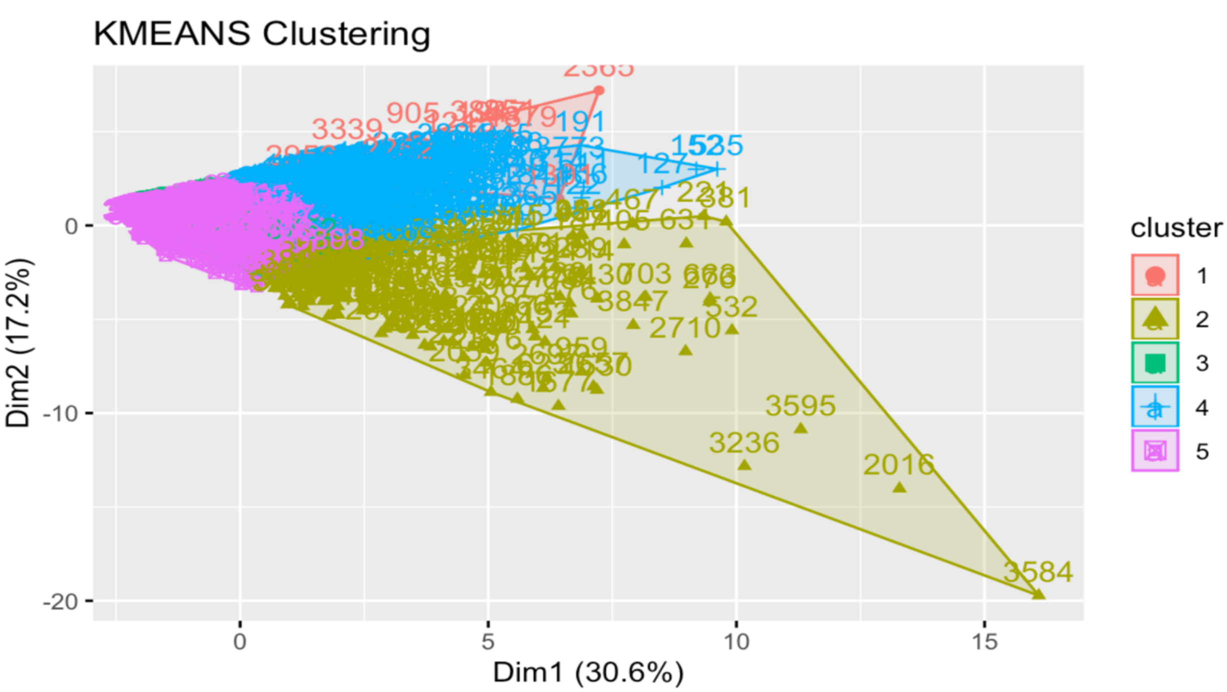
Cluster <int>	Freq <int>	Balance <dbl>	Qual_miles <dbl>	cc1_miles <dbl>	cc2_miles <dbl>	cc3_miles <dbl>
1	2850	31419.0	0	2500	2500	2500
2	405	79333.0	0	2500	2500	2500
3	744	97990.5	0	32500	2500	2500

```
#Per the midterm's instructions, we will now use K-means clustering
#The 5 cluster visualization is below.
km.3 <- eclust(data[,2:11], "kmeans", k = 3, nstart = 25, graph = TRUE)
```



3 clusters is a better that 5 clusters because there is a lot of overlap between clusters when you go over 3.

km.5 <- eclust(data[,2:11], "kmeans", k = 5, nstart = 25, graph = TRUE)



For the infrequent flyers who are new customers with low balance and bonus miles, I would offer discounts to increase Sales. Most of the customers in this cluster did not fly in last 12 months and this is the biggest cluster no matter what clustering method you use. There is a lot of untapped potential here.

PROBLEM #4

#Problem 4 was done in Python

```
import pandas as pd
import numpy as np

df_data = pd.read_csv('marketing.csv')
df_data.head()
```

recency	history	used_discount	used_bogo	zip_code	is_referral	channel	offer	conversion	
0	10	142.44	1	0	Surburban	0	Phone	Buy One Get One	0
1	6	329.08	1	1	Rural	1	Web	No Offer	0
2	7	180.65	0	1	Surburban	1	Web	Buy One Get One	0
3	9	675.83	1	0	Rural	1	Web	Discount	0
4	2	45.34	1	0	Urban	0	Web	Buy One Get One	0

```
df_data.isnull().sum().sum()
```

```
0
```

```
df_data['conversion'].value_counts()
```

```
0    54606
1     9394
Name: conversion, dtype: int64

df_data.dtypes

recency            int64
history          float64
used_discount      int64
used_bogo          int64
zip_code          object
is_referral        int64
channel           object
offer             object
conversion         int64
dtype: object

def calc_uplift(df):
    avg_order_value = 25
    base_conv = df[df.offer == 'No Offer']['conversion'].mean()
    disc_conv = df[df.offer == 'Discount']['conversion'].mean()
```

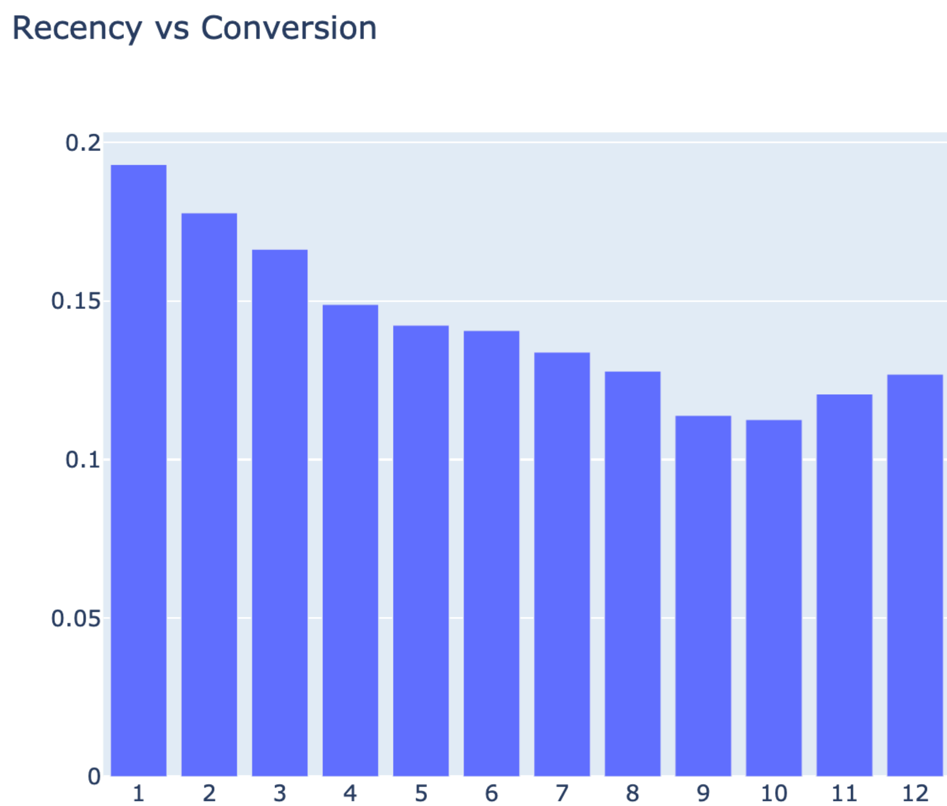
```
    bogo_conv = df[df.offer == 'Buy One Get One']['conversion'].mean()
    disc_conv_uplift = disc_conv - base_conv
    bogo_conv_uplift = bogo_conv - base_conv
    disc_order_uplift = disc_conv_uplift * len(df[df.offer ==
'Discount']['conversion'])
    bogo_order_uplift = bogo_conv_uplift * len(df[df.offer == 'Buy One Get
One']['conversion'])
    disc_rev_uplift = disc_order_uplift * avg_order_value
    bogo_rev_uplift = bogo_order_uplift * avg_order_value
    print('Discount Conversion Uplift: {0}%'.format(np.round(disc_conv_uplift*100,2)))
    print('Discount Order Uplift: {0}'.format(np.round(disc_order_uplift,2)))
    print('Discount Revenue Uplift: ${0}\n'.format(np.round(disc_rev_uplift,2)))
    print('-------------- \n')
    print('BOGO Conversion Uplift: {0}%'.format(np.round(bogo_conv_uplift*100,2)))
    print('BOGO Order Uplift: {0}'.format(np.round(bogo_order_uplift,2)))
    print('BOGO Revenue Uplift: ${0}'.format(np.round(bogo_rev_uplift,2)))
```

```
#Looks like offers do increase conversion. Discounts work better than By one get oe free.
calc_uplift(df_data)
```

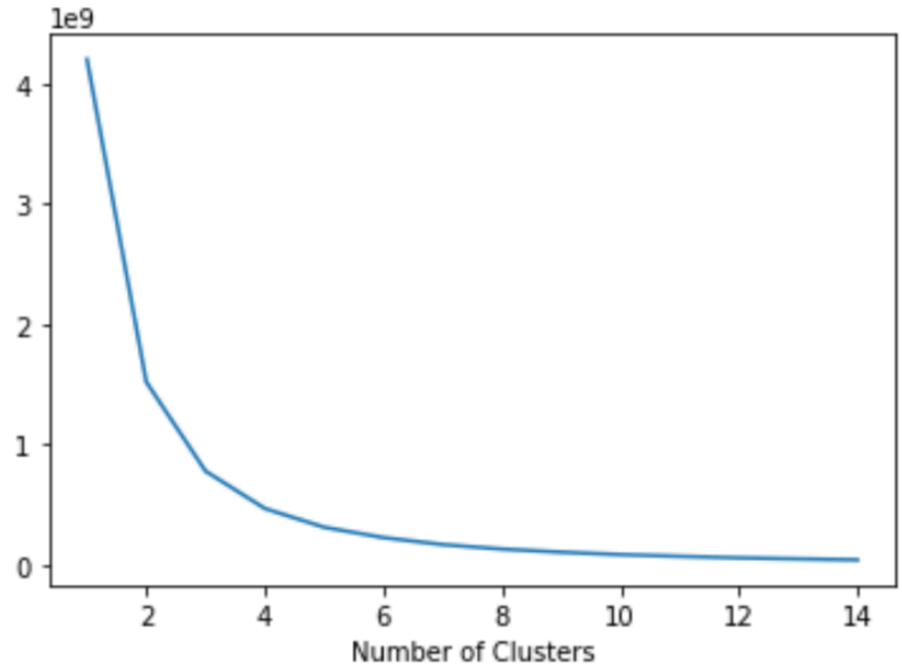
```
Discount Conversion Uplift: 7.66%
Discount Order Uplift: 1631.89
Discount Revenue Uplift: $40797.35

--------------

BOGO Conversion Uplift: 4.52%
BOGO Order Uplift: 967.4
BOGO Revenue Uplift: $24185.01
```

```
#Next I will looks at individual-level characteristics.
#We will consider all parameters to figure out which ones are worth including in our model.

#$ pip install chart_studio
import chart_studio.plotly.plotly as py
import plotly.offline as pyoff
import plotly.graph_objs as go

df_plot = df_data.groupby('recency')['conversion'].mean().reset_index()
plot_data = [
  go.Bar(
    x=df_plot['recency'],
    y=df_plot['conversion'],
  )
]
plot_layout = go.Layout(
    xaxis={"type": "category"},
    title='Recency vs Conversion',
  )
fig = go.Figure(data=plot_data, layout=plot_layout)
pyoff.iplot(fig)
```

#Higher recency is associated with higher conversion.




```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

sse = {}
tx_history = df_data[['history']]
for k in range(1,15):
    kmeans = KMeans(n_clusters = k, max_iter= 1_000).fit(tx_history)
    tx_history['clusters'] = kmeans.labels_
    sse[k] = kmeans.inertia_

plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel('Number of Clusters')
plt.show()
```

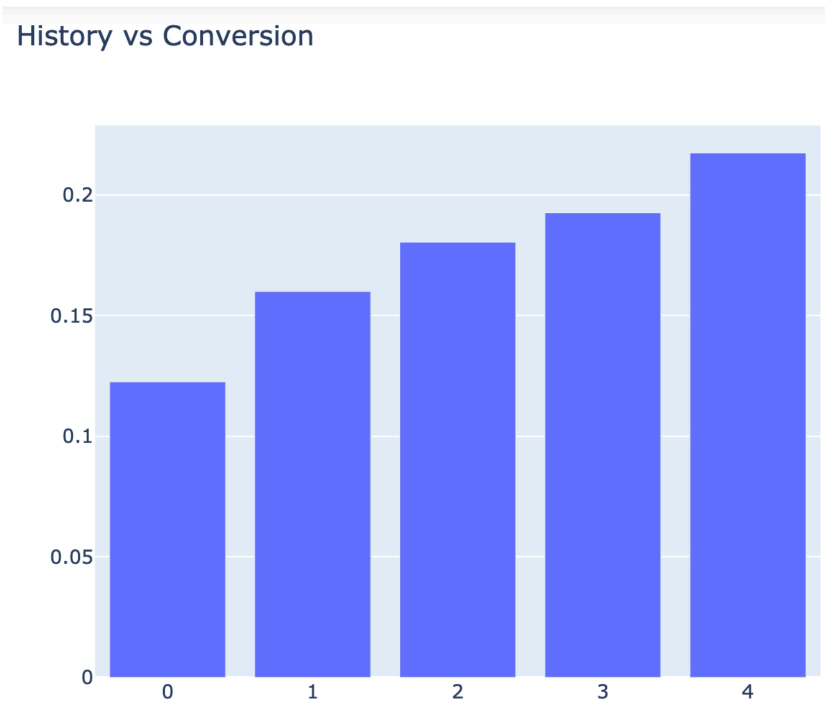



```
kmeans = KMeans(n_clusters=5)
kmeans.fit(df_data[['history']])
df_data['history_cluster'] = kmeans.predict(df_data[['history']])

#order the cluster numbers
df_data = order_cluster('history_cluster', 'history',df_data,True)

#print how the clusters look like
df_data.groupby('history_cluster').agg({'history':['mean','min','max'], 'conversion':['count', 'mean']})
```

	history			conversion	
	mean	**min**	**max**	**count**	**mean**
history_cluster					
0	73.907381	29.99	160.28	32278	0.122560
1	246.434560	160.30	362.49	17955	0.160067
2	478.085526	362.51	644.62	9105	0.180450
3	810.504639	644.66	1110.09	3742	0.192678
4	1410.097750	1111.09	3345.93	920	0.217391

```
#Customers with higher $ value of history are more liely to convert
df_plot = df_data.groupby('history_cluster').conversion.mean().reset_index()
plot_data = [
    go.Bar(
        x=df_plot['history_cluster'],
        y=df_plot['conversion'],
    )
]
```

```
plot_layout = go.Layout(
        xaxis={"type": "category"},
        title='History vs Conversion',
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
pyoff.iplot(fig)
```

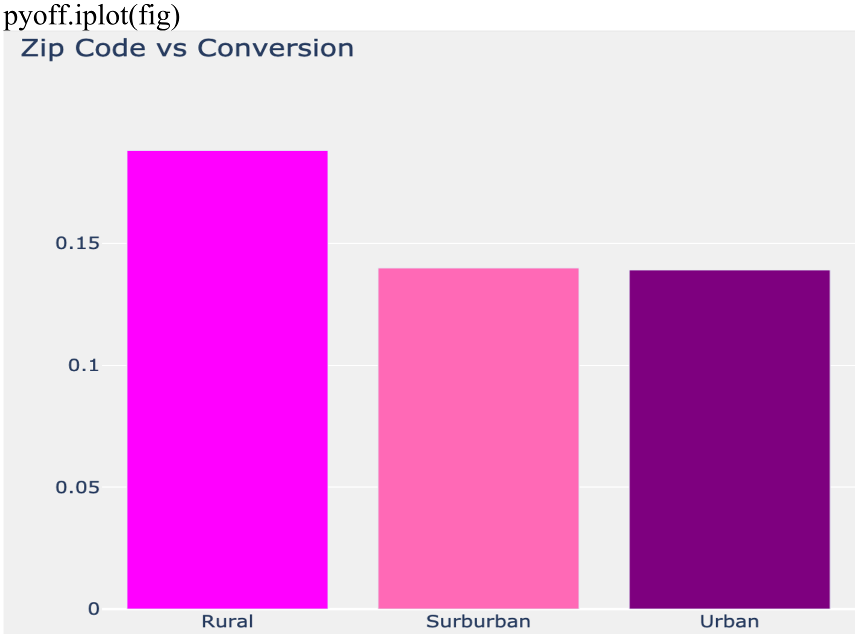



```
#People who used both discounts and BOGO have the highest conversion rate
df_data.groupby(['used_discount','used_bogo','offer']).agg({'conversion':'mean'})
```

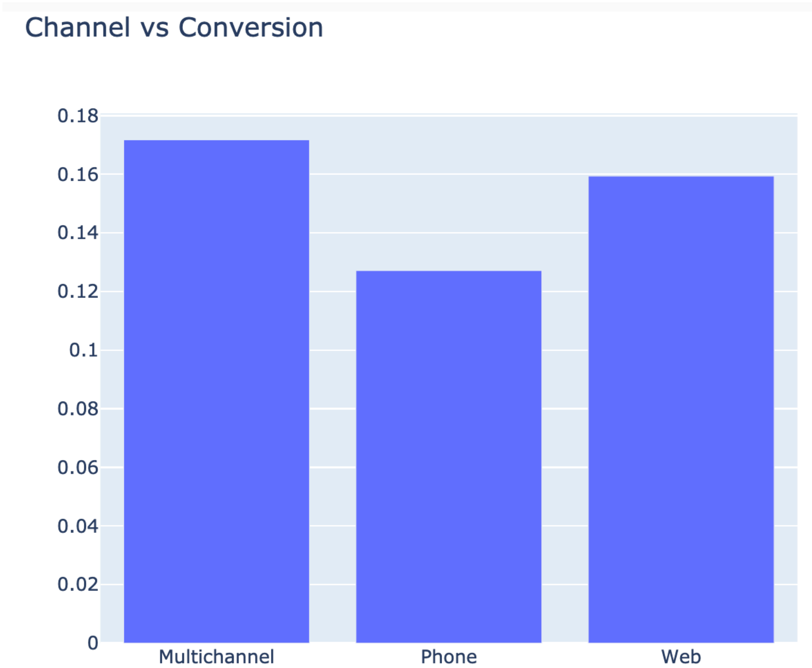
			conversion
used_discount	**used_bogo**	**offer**	
0	1	Buy One Get One	0.169794
		Discount	0.166388
		No Offer	0.095808
1	0	Buy One Get One	0.110892
		Discount	0.168968
		No Offer	0.099813
	1	Buy One Get One	0.251653
		Discount	0.314993
		No Offer	0.180549

```
df_plot = df_data.groupby('zip_code').conversion.mean().reset_index()
plot_data = [
```

```
    go.Bar(
        x=df_plot['zip_code'],
        y=df_plot['conversion'],
        marker=dict(
        color=['magenta', 'hotpink', 'purple'])
    )
]
plot_layout = go.Layout(
        xaxis={"type": "category"},
        title='Zip Code vs Conversion',
        plot_bgcolor  = 'rgb(243,243,243)',
        paper_bgcolor  = 'rgb(243,243,243)',
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
pyoff.iplot(fig)
```

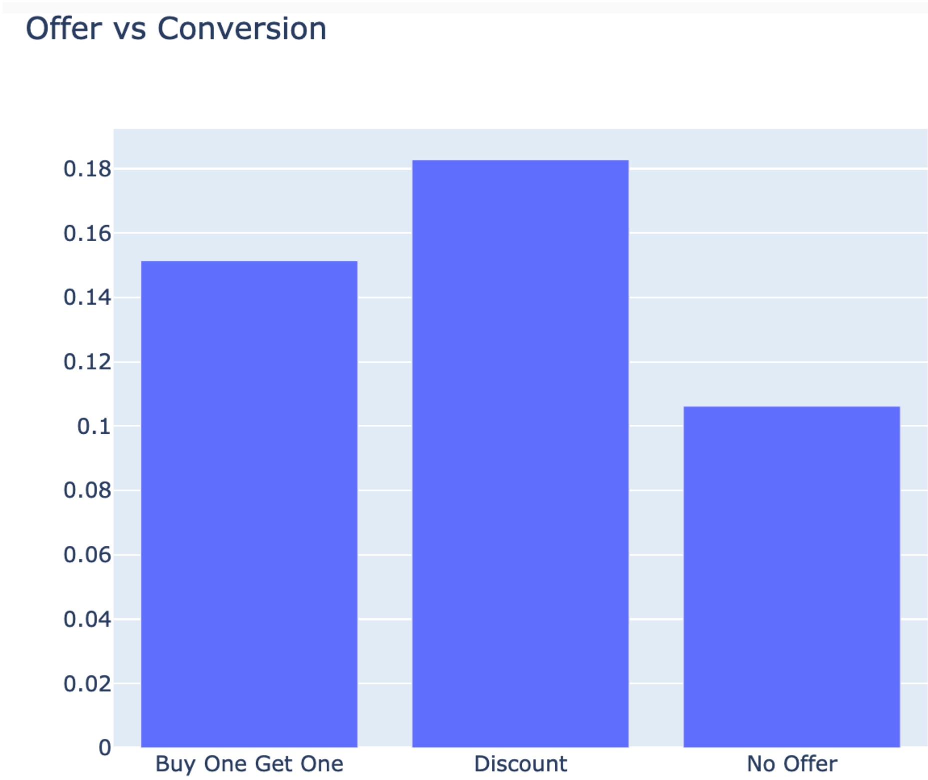



```
#Multichannel has higher conversion rate.
df_plot = df_data.groupby('channel').conversion.mean().reset_index()
plot_data = [
    go.Bar(
        x=df_plot['channel'],
        y=df_plot['conversion'],
    )
]
plot_layout = go.Layout(
        xaxis={"type": "category"},
        title='Channel vs Conversion',
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
pyoff.iplot(fig)
```



#Discounts really have a big impact on conversion.
#Customers who get discount offers have an 18% conversion rate.
#As discounts are very predictive of conversion we will definitely include this characteristic in our model.
#BOGO does well also.

```
df_plot = df_data.groupby('offer').conversion.mean().reset_index()
plot_data = [
    go.Bar(
        x=df_plot['offer'],
        y=df_plot['conversion'],
        marker=dict()
    )
]
plot_layout = go.Layout(
        xaxis={"type": "category"},
        title='Offer vs Conversion',
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
pyoff.iplot(fig)
```




```
df_model = df_data.copy()
df_model = pd.get_dummies(df_model)


X = df_model.drop(['conversion'], axis = 1)
y = df_model[['conversion']]

from sklearn.model_selection import train_test_split

#Partition the data into 60% train and 40% validation set per the midterm's directions.
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                     test_size = 0.4,
                                                    random_state = 56,
                                                    stratify = y)

#For macx homebrew has to be downloaded to run XGBoost
mkdir homebrew && curl -L https://github.com/Homebrew/brew/tarball/master | tar xz --strip 1 -C
homebrew

#XGBoost is an open-source software library which provides a gradient boosting framework
#We will use the XGBClassifier to do boosting.
from xgboost import XGBClassifier

conda install -c anaconda py-xgboost

X_test['proba'] = xgb_model.predict_proba(X_test)[:,1]
```

```
X_test['conversion'] = y_test

#Uplift shows how much better would we do with specific targeting relative to random targeting
#Below I calculate the uplift from offering discounts and BOGO
#We will find that discounts are much more effective

real_disc_uptick = int(len(X_test)*(X_test[X_test['offer_Discount'] == 1].conversion.mean() -
X_test[X_test['offer_No Offer'] == 1].conversion.mean()))
pred_disc_uptick = int(len(X_test)*(X_test[X_test['offer_Discount'] == 1].proba.mean() -
X_test[X_test['offer_No Offer'] == 1].proba.mean()))

#Uplift cutoff for discounts should be based on the output below.
print('Real Discount Uptick - Order: {}, Revenue: {}'.format(real_disc_uptick, real_disc_uptick*25))
print('Predicted Discount Uptick - Order: {}, Revenue: {}'.format(pred_disc_uptick,
pred_disc_uptick*25))
```

```
Real Discount Uptick - Order: 1985, Revenue: 49625
Predicted Discount Uptick - Order: 1840, Revenue: 46000
```

```
real_bogo_uptick = int(len(X_test)*(X_test[X_test['offer_Buy One Get One'] == 1].conversion.mean() -
X_test[X_test['offer_No Offer'] == 1].conversion.mean()))
pred_bogo_uptick = int(len(X_test)*(X_test[X_test['offer_Buy One Get One'] == 1].proba.mean() -
X_test[X_test['offer_No Offer'] == 1].proba.mean()))

#Uplift cutoff for BOGO should be based on the output below.
print('Real Bogo Uptick - Order: {}, Revenue: {}'.format(real_bogo_uptick, real_bogo_uptick*25))
print('Predicted Bogo Uptick - Order: {}, Revenue: {}'.format(pred_bogo_uptick, pred_bogo_uptick*25))
```

```
Real Bogo Uptick - Order: 1006, Revenue: 25150
Predicted Bogo Uptick - Order: 1223, Revenue: 30575
```