IDS PROJECT REPORT

on

BANK MARKETING

Semester V - 2019

Team Members:

Aditya Sharma	-	17UCC007
Mohit Sharma	-	17UCC036
Ritvik Singhal	-	17UCC048
Aniket Poddar	-	17UCS022

Submitted to:

Dr. Sakthi Balan Muthiah
Dr. Subrat Kumar Dash

LNMIIT
The LNM Institute of
Information Technology

TABLE OF CONTENTS

1. ABSTRACT

We propose a machine learning(ML) approach to predict the success of telemarketing calls for selling bank long-term deposits. A Portuguese retail bank was addressed, with data collected from 2008 to 2013, thus including the effects of the recent financial crisis. We have studied the dataset, visualized it through various plots and finally classified it to make inferences from the dataset. We compared three ML models: Decision trees (DTs), Random Forest and K-NN classifier. We also performed preprocessing on the data to prepare it for additional handling. After getting our results we analyzed it using various quality measures.

2. INTRODUCTION

2.1 Problem Statement

Apply suitable ML Classification algorithms on the data set and get inferences from the data. Choose the appropriate ML algorithm packages available in R or Python. Do preprocess on the dataset. Do preliminary analysis on the dataset as much as needed and get a summarization of your inferences with proper validation.

2.2 Topic Description

We present a comprehensive analysis of **Bank Marketing** Data taken from the UCI Machine Learning Repository. The marketing campaign was based on phone calls done to people. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed. The data collected is from 2008 to 2013 thus including the effects of the financial crisis. We did an analysis of this data and inferred on whether the client takes a loan subscription or not.

3. LIBRARIES USED: BRIEF OVERVIEW

```
import pandas as pd
import numpy as np
import math
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

3.1 Pandas

The Pandas library provides high-performance, easy-to-use data structures. We used the pandas library to import the dataset in our data frame and the same was used in all the further steps.

3.2 MatplotLib and Seaborn

We used the Seaborn library along with the Matplotlib library to make use of its enhanced visualization features while plotting the required curves.

3.3 SciKit Learn

This is the primary library being used in the project for applying the classification models/families as well as implementing the preprocessing and cross-validation steps.

4. DATASET DESCRIPTION

The dataset has two classes, namely ‘Yes’ and ‘No’. The classes refer to whether the client has subscribed to a term deposit or not.

4.1 Importing the Dataset

We have imported the dataset as a data frame of the pandas library in Python to make further operations fast and easy.

```
bank=pd.read_csv('bank-full.csv',sep=";")
```

4.2 Understanding the dataset

Below attached is an overview of the data frame.

```
bank.head()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no

This gives the count of the number of rows in the dataframe

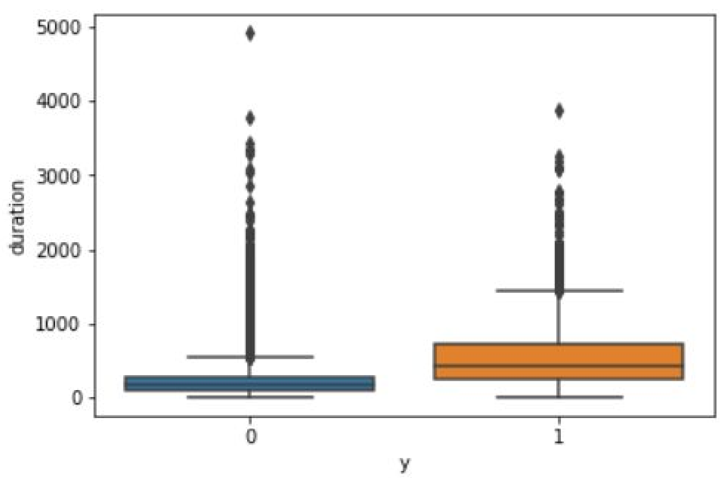
```
print("{rows}".format(rows = len(bank)))
```

```
45211
```

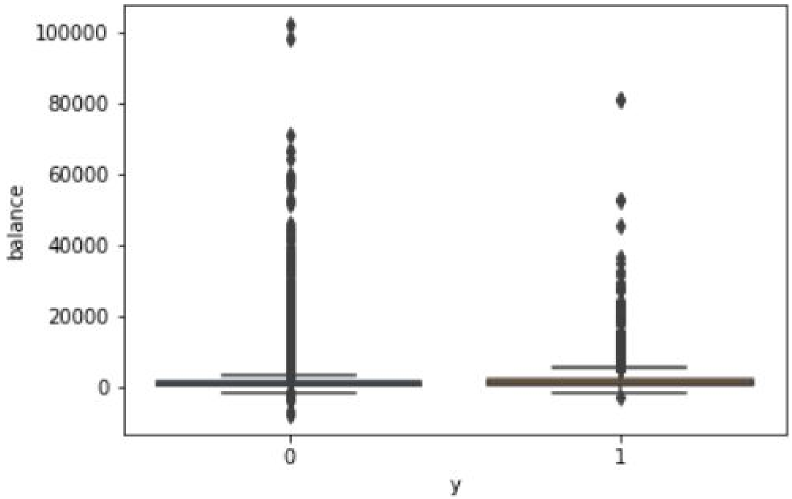
4.3 General Trends In the data

We have created (using Seaborn library) and observed a box plot for the class.
We observed that the attribute named **Duration and balance** has many outliers.

```
ax = sns.boxplot(y=bank["duration"], x = bank['y'])
```



```
ax = sns.boxplot(y=bank["balance"], x = bank['y'])
```

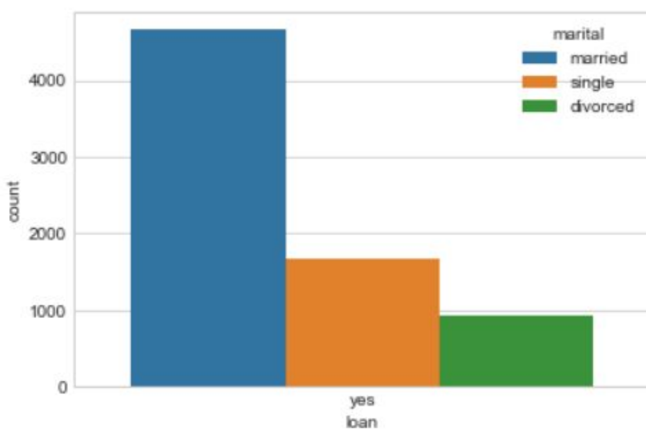


4.3.1 CountPlot

1. **LOAN**
   We have used countplot to plot a graph for loan and compared the results for various marital status namely married, single and divorced.

```
sns.countplot(x=bank[bank['loan']=="yes"]['loan'],hue='marital',data=bank)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1dd98c1c898>
```

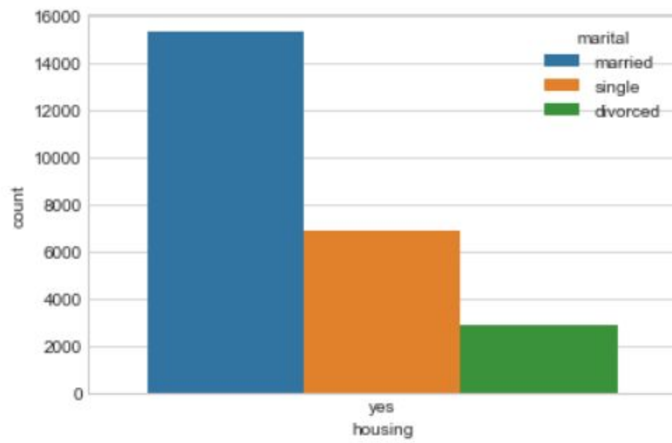


2. HOUSING

We have compared the housing count for various marital status.

```
sns.countplot(x=bank[bank['housing']=="yes"]['housing'],hue='marital',data=bank)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1dd9805e278>
```

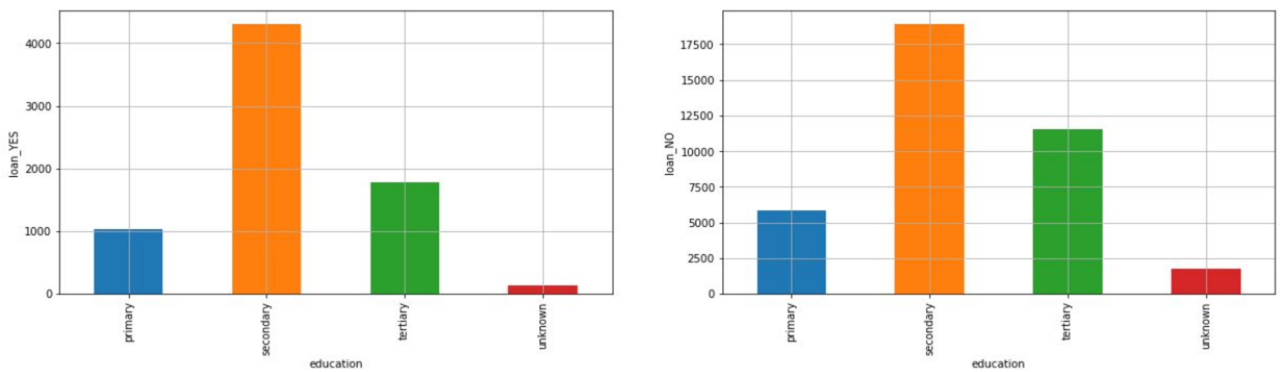


- From the graphs received we realized that both Normal loan and Housing loan are taken the **most by married people** and the **least** amount of housing and normal loans are taken by **Divorcees**.

4.3.2 <u>Barplot</u>

1. **LOAN on the basis of EDUCATION**

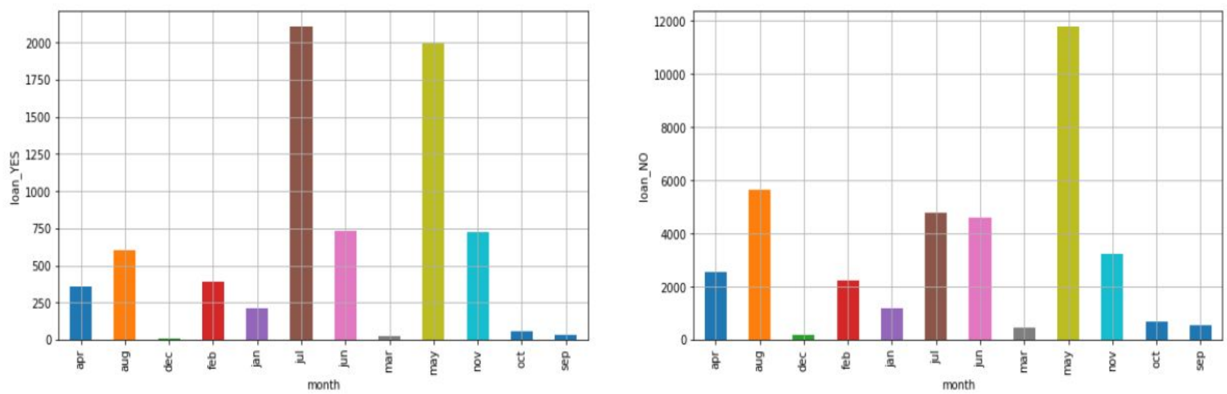
```
data1=bank.groupby('education').apply(lambda x:(x[x['loan']=="yes"]['loan']).count())
plt.subplot(1,2,1)
data1.plot(kind='bar' , figsize= (20,5))
plt.ylabel("loan_YES")
plt.grid()
#print(data1)
data2=bank.groupby('education').apply(lambda x:(x[x['loan']=="no"]['loan']).count())
plt.subplot(1,2,2)
data2.plot(kind='bar')
plt.ylabel("loan_NO")
#print(data1)
plt.grid()
plt.show()
```



- We noticed that a person who had secondary education have applied more for the loan and also got the **most loan approvals**. Following seconding education loans, Tertiary education loans have been accepted the most next.
- When we see the results of the **loans rejected**, secondary education loans have been the most rejected followed by tertiary education loans.

2. **<u>LOAN on the basis of MONTH</u>**

```
data1=bank.groupby('month').apply(lambda x:(x[x['loan']=="yes"]['loan']).count())
plt.subplot(1,2,1)
data1.plot(kind='bar' , figsize=(20,5))
plt.ylabel("loan_YES")
plt.grid()
data2=bank.groupby('month').apply(lambda x:(x[x['loan']=="no"]['loan']).count())
plt.subplot(1,2,2)
data2.plot(kind='bar')
plt.ylabel("loan_NO")
#print(data2)
plt.grid()
plt.show()
```

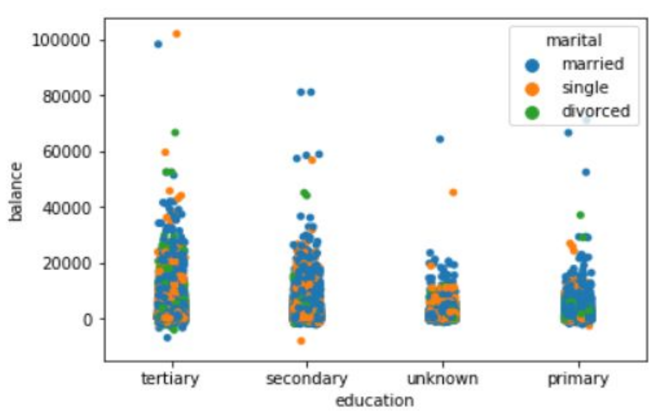


- We noticed that the **maximum loans** were **accepted** in the month of **July** followed by May.
- When we see the results of the **loans rejected**, the **maximum loans** where rejected in the month of **May** followed by August..

4.3.3 <u>Striplot</u>

1. **Balance VS Education(on the basis of GENDER)**

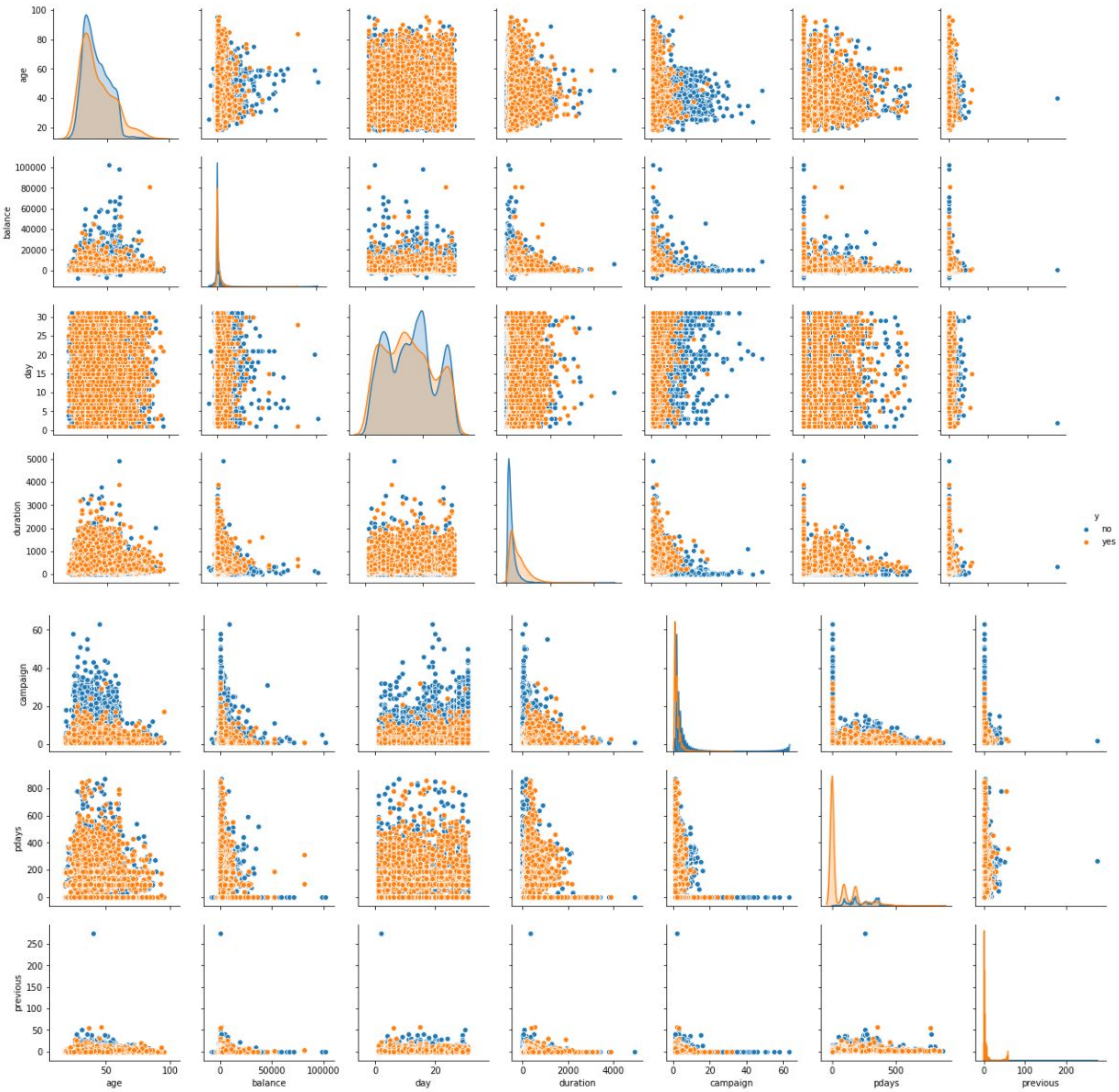
```
ax=sns.stripplot(x="education",y="balance",hue="marital",data=bank)
```



4.3.4 Pairplot

```
sns.pairplot(bank,hue='y')
```

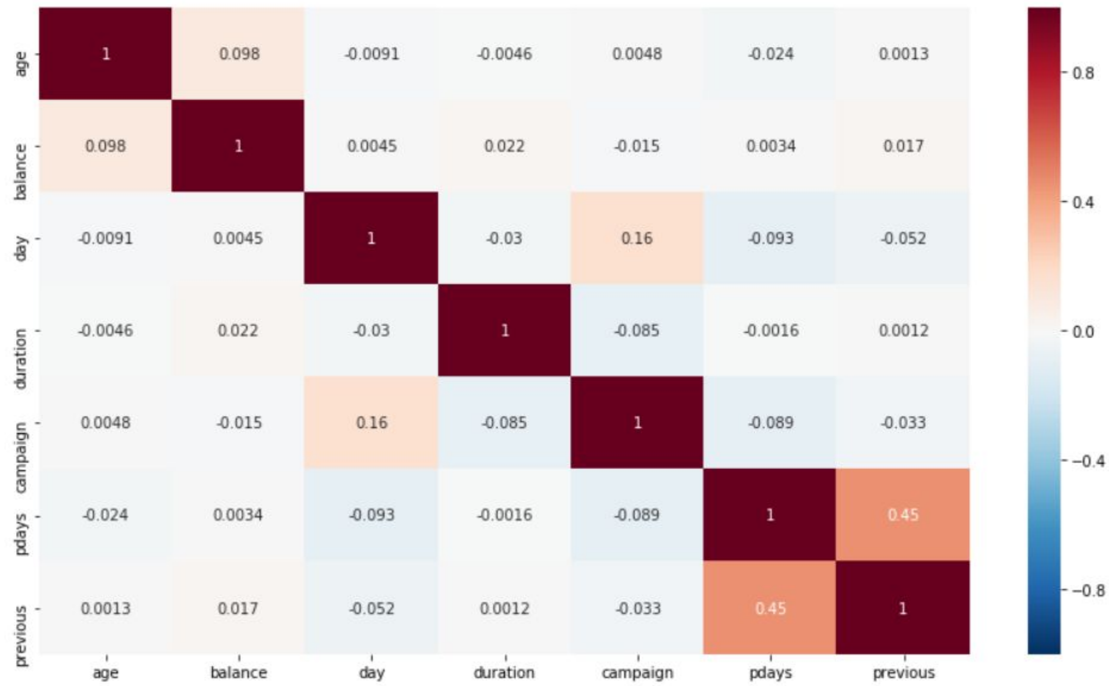
<seaborn.axisgrid.PairGrid at 0x1dd91521908>



4.3.5 Heatmap

```
correlation=bank.corr()
plt.figure(figsize=(14,8))
sns.heatmap(correlation,annot=True,linewidth=0,vmin=-1,cmap="RdBu_r")
```

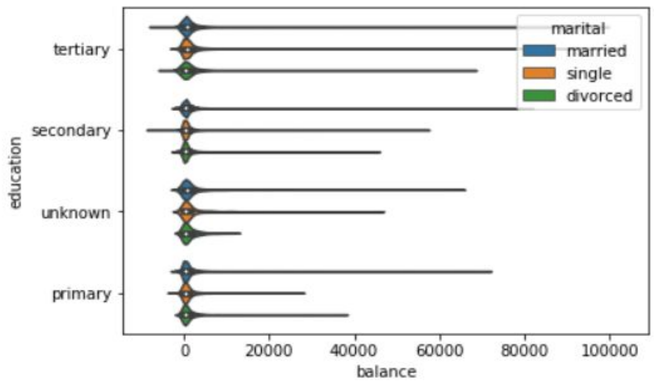
```
<matplotlib.axes._subplots.AxesSubplot at 0x1dd91b22e10>
```



- It is based on the similarity between attributes(normalized) the more similar its is the closer it is to 1 and vice-versa.
- Here we can observe that the more the person is older the more balance is there in there account.

4.3.6 ViolinPlot

```
ax=sns.violinplot(x="balance",y="education",hue="marital",data=bank)
```



- Here we can see some unexpected rise in the balance those can be considered as outliers and can be removed when needed. Here we didn't consider removing it as it didn't interfere in the analysing or classifier.

5. DATA PREPROCESSING

5.1. Checking for Null Values

```
bank.isnull().sum()
```

```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
dtype: int64
```

NULL values generally creep into a dataset because of reasons such as incomplete extraction of data, corrupt data, failure to load information etc. This figure above shows that there are **NO NULL values** present in our dataset.

Here attached is a statistical view of the dataset.

```
bank.describe()
```

	age	balance	day	duration	campaign	pdays	previous
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828	0.580323
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746	2.303441
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

5.2. Handling Nominal Attributes

```
In [61]: label_encoder = LabelEncoder()
bank['job'] = label_encoder.fit_transform(bank['job'])
bank['marital'] =label_encoder.fit_transform(bank['marital'])
bank['education'] = label_encoder.fit_transform(bank['education'])
bank['default'] = label_encoder.fit_transform(bank['default'])
bank['housing'] = label_encoder.fit_transform(bank['housing'])
bank['loan'] = label_encoder.fit_transform(bank['loan'])
bank['month'] = label_encoder.fit_transform(bank['month'])
#bank['day_of_week'] = label_encoder.fit_transform(bank['day_of_week'])
bank['poutcome'] = label_encoder.fit_transform(bank['poutcome'])
bank['y'] = label_encoder.fit_transform(bank['y'])
```

The target variable in the dataset is a categorical variable having two non-numeric values 'y' and 'n. To handle this, we used label_encoder to change the categorical data to numeric data

5.3. Binning

```
def Age(ag):
    if ag > 60 : return 1
    elif 60 >ag >=45:
        return 2
    elif 45 > ag >=30:
        return 3
    elif 30 > ag >=15 :
        return 4
    else :
        return 5
bank['age']=bank['age'].map(Age)
```

```
def Pdays(pd):
    if pd == 871 : return 1

    else :
        return 0
bank['pdays']=bank['pdays'].map(Pdays)
```

We have done Binning so that the value in any particular range can only be represented by a single value. We have used Binning by Width here since for a particular range of width only one value is assigned to all the data points in that range.

5.4. Dropping Similar Attributes

```
In [92]: bank = bank.drop('poutcome', axis=1)
```

. The outcome has similar values throughout the dataset and hence we have dropped these values since they have no contribution in model training.

6. CLASSIFIERS

A classifier utilizes some training data to understand how given input variables relate to the class. There are a lot of classification algorithms to choose from and it is difficult to say which is better than the other as it depends on the application and the nature of the available dataset. Thus, in this section we'll be looking at how we went about this process of classifying.

6.1. Splitting training and Test data

30% of the data was chosen as test data and the rest 70% was chosen as training data. We randomly selected rows for dividing the data. We chose the partition percentage in a random manner to avoid any bias.

```
x = bank.drop('y', axis=1)
Y = bank['y']

X_train, X_test, y_train, y_test = train_test_split(x,Y,test_size=0.30,random_state=1)
```

6.2. Parameter tuning of classifier

For parameter tuning, we use 3-fold cross validation method in which 2/3 of the data is taken as the training data and the rest as cross validation data. This improves the hyper-parameters that need to be tuned in order to get more accurate predictions.

6.3. Classifying using K-NN Classifier

K-NN classifier can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. In K-NN algorithm we decide the number of nearest neighbours (given by K) and then classifies the data based on the majority of the labels these neighbours have. Higher the number of nearest neighbours, higher is the accuracy of our classification. .

6.3.1 Why K-NN is used?

On analysing the data it was seen that the data points were close to each other. Because of this, forming groups was easier and classifying could be done nicely. Here we used K=3 i.e. 3 nearest neighbours were used for classifying the data.

6.3.2 Results

```
from sklearn.svm import SVC
```

```
model =  KNeighborsClassifier(n_neighbors=3)
model.fit(X_train , y_train )
predicted2 = model.predict(X_test)
```

```
print('Accuracy:',round(metrics.accuracy_score(y_test,predicted2),5))
```

```
Accuracy: 0.86531
```

The **accuracy** of K-NN Classification is **0.86531 = 86.53%**

6.3.3 Issues faced while using K-NN

We faced the problem of Overfitting of the data points. We now try classifying using Decision Tree. 2 of our attributes- Balance and Duration have many outliers in the data. Due to this fact K-NN couldn’t handle the noise points in the data and thus our classifying couldn't be done perfectly.

6.4. Classifying using Decision Tree

Decision tree builds classification or regression models in the form of a tree structure. It utilizes an if-then rule set which is mutually exclusive and exhaustive for classification. The rules are learned sequentially using the training data one at a time. Each time a rule is learned, the tuples covered by the rules are removed. This process is continued on the training set until meeting a termination condition

6.4.1 Why Decision Tree is used?

The data we had were majorly of the Binary Form type and Ordinal Type. 2 attributes were of continuous type.This made splitting the data and making the decision based on the split easier. We used GINI index to determine the best splitting point.

6.4.2.Results

```
from sklearn.tree import DecisionTreeClassifier
model =  DecisionTreeClassifier()
model.fit(X_train , y_train )
predicted3 = model.predict(X_test)
```

```
print('Accuracy:',round(metrics.accuracy_score(y_test,predicted3),5))
```

```
Accuracy: 0.86523
```

The **accuracy** of Decision Tree Classification is **0.86523= 86.523%**

6.3.3 Issues faced while using Decision Tree

We faced the problem of overfitting in decision tree classification. A decision tree can be easily over-fitted generating too many branches. Decision trees are prone to overfitting, especially when a tree is particularly deep. We finally use Random Forest.

6.5. Classifying using Random Forest

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction. Uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions

6.5.1 Why Random Forest is used?

Random Forest can easily handle noisy data which our K-NN classifier failed to do. As observed in the other two methods, overfitting was happening. In Random Forest, overfitting is avoided. This makes our result more generalized and gives better predictions.

6.5.2.Results

```
model =RandomForestClassifier(n_estimators = 1000 , criterion = 'entropy' , random_state=3)
model.fit(X_train , y_train)
predicted = model.predict(X_test)
```

```
from sklearn import metrics

print('Accuracy:',round(metrics.accuracy_score(y_test,predicted),5))
```

```
Accuracy: 0.90077
```

The accuracy of Random Forest Classification is 0.90077= 90.07%

7. EVALUATIONS AND INFERENCES

A classifier must not predict Yes('Y') to be No('n') and it is equally important for vice-versa. The reason for the former being, a high false positive score implies interpreting false information for loan prediction. We don't want this as this may result in misinterpretation of source of sensing data. Similarly we also don't want that we miss out on information on the loans not approved. For these reasons we can't rely solely on accuracy for quality metric of classification. So we use precision, recall and F1 score for the same which also takes false negative and false positive into account. We also plotted the ROC Curve to find the accuracy.

```
results = confusion_matrix(y_test, predicted3)
print(results)
print( classification_report(y_test, predicted3))
```

```
[[11014   967]
 [  861   722]]
              precision    recall  f1-score   support

           0       0.93      0.92      0.92     11981
           1       0.43      0.46      0.44      1583

   micro avg       0.87      0.87      0.87     13564
   macro avg       0.68      0.69      0.68     13564
weighted avg       0.87      0.87      0.87     13564
```

Fig : Confusion Matrix and Precision,Recall and F1-score for K-NN Classifier

```
results = confusion_matrix(y_test, predicted2)
print(results)
print( classification_report(y_test, predicted2))
```

```
[[11367   614]
 [ 1213   370]]
              precision    recall  f1-score   support

           0       0.90      0.95      0.93     11981
           1       0.38      0.23      0.29      1583

   micro avg       0.87      0.87      0.87     13564
   macro avg       0.64      0.59      0.61     13564
weighted avg       0.84      0.87      0.85     13564
```

Fig : Confusion Matrix and Precision,Recall and F1-score for Decision Tree

```
results = confusion_matrix(y_test, predicted)
print(results)
print( classification_report(y_test, predicted))
```

```
[[11642   339]
 [  940   643]]
              precision    recall  f1-score   support

           0       0.93      0.97      0.95     11981
           1       0.65      0.41      0.50      1583

   micro avg       0.91      0.91      0.91     13564
   macro avg       0.79      0.69      0.72     13564
weighted avg       0.89      0.91      0.90     13564
```

Fig : Confusion Matrix and Precision,Recall and F1-score for Random Forest

```
kn_fpr , kn_tpr,_=roc_curve(y_test,kn_probs)
dt_fpr , dt_tpr,_=roc_curve(y_test,dt_probs)
rf_fpr , rf_tpr,_=roc_curve(y_test,rf_probs)

plt.plot(kn_fpr, kn_tpr, label='KNN')
plt.plot(dt_fpr, dt_tpr, label='Decision Tree')
plt.plot(rf_fpr, rf_tpr, label='Random Forest')

plt.title("Roc Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")


plt.legend()
#plt.show()
```

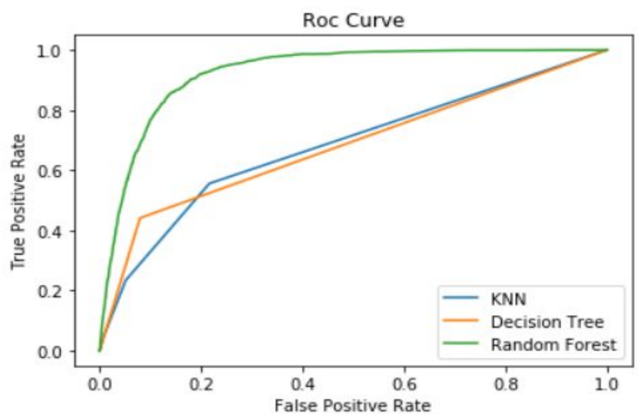


FIG: ROC Curve for K-NN, Decision Tree and Random Forest

```
kn_au = roc_auc_score(y_test,kn_probs)
rf_au = roc_auc_score(y_test,rf_probs)
dt_au = roc_auc_score(y_test,dt_probs)
```

```
print("Area under the curve:\nKNN:{0:}\tDecision Tree:{1:}\tRandom Forest:{2:}".format(kn_au ,
dt_au ,
rf_au))
```

```
Area under the curve:
KNN:0.6811431745241189	Decision Tree:0.6807378950130716	Random Forest:0.9252332723274265
```

FIG: The area under the curve for all classifiers

- We can clearly see that the precision ,recall and F1- score for decision tree and K-NN Classifiers are almost similar and are less than the values obtained for Random Forest. It can be clearly inferred that Random Forest Classification outperformers the other two.

- On comparing the area under the curve in the ROC Curve for KNN,Decision Tree and Random Forest it is clearly visible that KNN and decision have similar accuracy but Random Forest has a far greater accuracy of 92.52% and outperforms the other two.

- Random Forest is able to outperform the other two because of its ability to use many trees to classify the data. Thus random forest becomes more generalized than the other two and thus is performing better than the other two classifying algorithms.

8. CONCLUSIONS

In our study we found out that:

- The data we have obtained is a classification related dataset. We got to know this by using various visualization techniques.

- We used various classification techniques and compared them. On evaluating the results we can conclude that Random Forest is the best classification technique according to the dataset.