PREDICT CREDIT CARD ACCEPTANCE

Group Members:

SUVAM DAS, GCECT, 161130110072 OF 2016-17
SHAWAN BASU, GCECT, 161130110064 OF 2016-17
PRIYAM MUKHERJEE, GCECT, 161130110057 OF 2016-17
RIYA KARAN, GCECT, 161130110020 OF 2016-17

Table of Contents

- Acknowledgement
- Project Objective
- Project Scope
- Data Description
- Data Preprocessing
- Model Building
- Future Scope of Improvements
- Project Certificate

Acknowledgement

I take this opportunity to express my profound gratitude and deep regards to my faculty, Prof. Arnab Chakraborty for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him/her time to time shall carry me a long way in the journey of life on which I am about to embark.

I am obliged to my project team members for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

Suvam Das

Project Objective

This project aims to predict whether the application of credit card will be approved or not.

The project uses Machine learning concept using Python to implement and apply different algorithms to the provided dataset.

MACHINE LEARNING:

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

Project Scope

- This analysis give a clear picture of application status of Credit Card according to several parameter.
- A Systematic way to approve so many applications for credit card.
- Analysis can deals with various customer’s information.

Different models used are:

- Logistic regression
- Decision tree
- Naïve bayes
- K –NN

Performance evaluation of each model is done.

Data description

- Source of data : a small credit card dataset for simple econometric analysis (taken from www.kaggle.com originally from William Greene’s book Econometric analysis.
- A part of the data set containing all the fields required for the analysis

card	reports	age	income	share	expenditu	owner	selfemp	depender	months	majorcard	active
yes	0	37.66667	4.52	0.03327	124.9833	yes	no	3	54	1	12
yes	0	33.25	2.42	0.005217	9.854167	no	no	3	34	1	13
yes	0	33.66667	4.5	0.004156	15	yes	no	4	58	1	5
yes	0	30.5	2.54	0.065214	137.8692	no	no	0	25	1	7
yes	0	32.16667	9.7867	0.067051	546.5033	yes	no	2	64	1	5

ATTRIBUTE	NON-NULL VALUE	DATA TYPE	DESCRIPTION
Card	1319	Object	Represent card availability
Reports	1319	Int	Describe issues with the bank
Age	1319	Float	Describe age of the customer
Income	1319	Float	Describe income per annum divided by 10000.
Share	1319	Float	Ratio of expenditure to income
Expenditure	1319	float	Average monthly expenditure
Owner	1319	Object	Describe home status
Selfemp	1319	Object	Customer employment status
Dependents	1319	Int	No. of dependents
Major	1319	Int	No. of major credit card held

DATA PREPROCESSING:

Importing Modules and DataSet

```
np.random.seed(0)

df1 = pd.read_csv("Worksheet in Project Topics for SS 2019.csv")
df1 = df1.replace({'yes':1,'no':0})
df1.head()
```

	card	reports	age	income	share	expenditure	owner	selfemp	dependents	months	majorcards	active
0	1	0	37.66667	4.5200	0.033270	124.983300	1	0	3	54	1	12
1	1	0	33.25000	2.4200	0.005217	9.854167	0	0	3	34	1	13
2	1	0	33.66667	4.5000	0.004156	15.000000	1	0	4	58	1	5
3	1	0	30.50000	2.5400	0.065214	137.869200	0	0	0	25	1	7
4	1	0	32.16667	9.7867	0.067051	546.503300	1	0	2	64	1	5

Observing DataSet

This stage allows you to do basic sanity checks about the distribution of the data.
For example, we know that none of the values can be negative.

```
desc = df1.describe()
desc
```

	card	reports	age	income	share	expenditure	owner	selfemp	dependents	months	majorcards	
count	1319.000000	1319.000000	1319.000000	1319.000000	1319.000000	1319.000000	1319.000000	1319.000000	1319.000000	1319.000000	1319.000000	1319.
mean	0.775588	0.456406	33.213103	3.365376	0.068732	185.057071	0.440485	0.068992	0.993935	55.267627	0.817286	6.
std	0.417353	1.345267	10.142783	1.693902	0.094656	272.218917	0.496634	0.253536	1.247745	66.271746	0.386579	6.
min	0.000000	0.000000	0.166667	0.210000	0.000109	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	1.000000	0.000000	25.416670	2.243750	0.002316	4.583333	0.000000	0.000000	0.000000	12.000000	1.000000	2.
50%	1.000000	0.000000	31.250000	2.900000	0.038827	101.298300	0.000000	0.000000	1.000000	30.000000	1.000000	6.
75%	1.000000	0.000000	39.416670	4.000000	0.093617	249.035800	1.000000	0.000000	2.000000	72.000000	1.000000	11.
max	1.000000	14.000000	83.500000	13.500000	0.906320	3099.505000	1.000000	1.000000	6.000000	540.000000	1.000000	46.

INTERPRETING THE DATA:

CARD: This column represents card availability.

```
print('Total No. of accepted cards = ',df1['card'][df1['card']==1].count())
print('Total No. of rejected cards = ',df1['card'][df1['card']==0].count())
print(desc.card)
```

```
Total No. of accepted cards =  1023
Total No. of rejected cards =  296
count    1319.000000
mean        0.775588
std         0.417353
min         0.000000
25%         1.000000
50%         1.000000
75%         1.000000
max         1.000000
Name: card, dtype: float64
```

REPORT: Describe issues about any circumstances with the bank.consist information.

```
print('Value counts:\n',df1['reports'].value_counts())
print(desc.reports)
```

```
Value counts:
 0     1060
1      137
2       50
3       24
4       17
5       11
7        6
6        5
11       4
9        2
14       1
12       1
10       1
Name: reports, dtype: int64
count    1319.000000
mean        0.456406
std         1.345267
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max        14.000000
Name: reports, dtype: float64
```

AGE: Describe age of the customer.consist information.

```
print('Total counts = ',df1['age'].count())
print(desc.age)
```

```
Total counts =  1319
count    1319.000000
mean       33.213103
std        10.142783
min         0.166667
25%        25.416670
50%        31.250000
75%        39.416670
max        83.500000
Name: age, dtype: float64
```

INCOME: Describe income of the customer per annum.

```
print('Total counts = ',df1['income'].count())
print(desc.income)
```

```
Total counts =  1319
count    1319.000000
mean        3.365376
std         1.693902
min         0.210000
25%         2.243750
50%         2.900000
75%         4.000000
max        13.500000
Name: income, dtype: float64
```

EXPENDITURE: Describes average monthly credit card expenditure.

```
print('Total counts = ',df1['expenditure'].count())
print(desc.expenditure)
```

```
Total counts =  1319
count    1319.000000
mean      185.057071
std       272.218917
min         0.000000
25%         4.583333
50%       101.298300
75%       249.035800
max      3099.505000
Name: expenditure, dtype: float64
```

OWNER:-'1'owns their home, '**0**' if rent. Consists information.

```
print('Total No. of applicants who does not live in their own house = ',df1['owner'][df1['owner']==0].count())
print('Total No. of applicants who live in their own house = ',df1['owner'][df1['owner']==1].count())
print(desc.owner)
```

```
Total No. of applicants who does not live in their own house =  581
Total No. of applicants who live in their own house =  738
count    1319.000000
mean        0.440485
std         0.496634
min         0.000000
25%         0.000000
50%         0.000000
75%         1.000000
max         1.000000
Name: owner, dtype: float64
```

SELFEMP: This column represents that the customer is self employeed or not.

```
print('Total No. of self employeed applicants = ',df1['selfemp'][df1['selfemp']==1].count())
print('Total No. of not self employeed applicants = ',df1['selfemp'][df1['selfemp']==0].count())
print(desc.selfemp)
```

```
Total No. of self employeed applicants =  91
Total No. of not self employeed applicants =  1228
count    1319.000000
mean        0.068992
std         0.253536
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000
Name: selfemp, dtype: float64
```

MONTHS: Months living at a current address.

```
print('Total counts = ',df1['months'].count())
print(desc.months)
```

```
Total counts =  1319
count    1319.000000
mean       55.267627
std        66.271746
min         0.000000
25%        12.000000
50%        30.000000
75%        72.000000
max       540.000000
Name: months, dtype: float64
```

MAJOR CARDS:-Number of major credit card held.

```
print('Total No. of major card holder applicants = ',df1['majorcards'][df1['majorcards']==1].count())
print('Total No. of applicants who does not have major card = ',df1['majorcards'][df1['majorcards']==0].count())
print(desc.majorcards)
```

```
Total No. of major card holder applicants =  1078
Total No. of applicants who does not have major card =  241
count    1319.000000
mean        0.817286
std         0.386579
min         0.000000
25%         1.000000
50%         1.000000
75%         1.000000
max         1.000000
Name: majorcards, dtype: float64
```

ACTIVE:-Describe number of active credit accounts.

```
print('Value counts:\n',df1['active'].value_counts())
print(desc.active)
```

```
Value counts:
 0      219
2      92
3      91
5      86
4      84
7      82
1      73
6      72
9      71
11     63
8      62
10     44
13     42
12     38
16     30
14     30
15     27
17     23
19     21
```

Correlation between the columns:

```
df1.corr()
```

	card	reports	age	income	share	expenditure	owner	selfemp	dependents	months	majorcards	active
card	1.000000	-0.452577	0.000537	0.094308	0.388028	0.365814	0.147826	-0.054340	-0.036126	-0.000268	0.107769	0.080464
reports	-0.452577	1.000000	0.044089	0.011023	-0.159011	-0.136538	-0.053570	0.018835	0.019731	0.048968	-0.007304	0.207755
age	0.000537	0.044089	1.000000	0.324653	-0.115697	0.014948	0.367749	0.100421	0.212146	0.436426	0.009777	0.181070
income	0.094308	0.011023	0.324653	1.000000	-0.054429	0.281104	0.324776	0.112294	0.317601	0.130346	0.107138	0.180540
share	0.388028	-0.159011	-0.115697	-0.054429	1.000000	0.838779	-0.015764	-0.078905	-0.082618	-0.055348	0.051470	-0.023474
expenditure	0.365814	-0.136538	0.014948	0.281104	0.838779	1.000000	0.093180	-0.035638	0.052664	-0.029007	0.077514	0.054724
owner	0.147826	-0.053570	0.367749	0.324776	-0.015764	0.093180	1.000000	0.041673	0.309190	0.238652	0.063851	0.274924
selfemp	-0.054340	0.018835	0.100421	0.112294	-0.078905	-0.035638	0.041673	1.000000	0.042096	0.065912	0.004854	0.029555
dependents	-0.036126	0.019731	0.212146	0.317601	-0.082618	0.052664	0.309190	0.042096	1.000000	0.046512	0.010285	0.107133
months	-0.000268	0.048968	0.436426	0.130346	-0.055348	-0.029007	0.238652	0.065912	0.046512	1.000000	-0.041447	0.100028
majorcards	0.107769	-0.007304	0.009777	0.107138	0.051470	0.077514	0.063851	0.004854	0.010285	-0.041447	1.000000	0.119603
active	0.080464	0.207755	0.181070	0.180540	-0.023474	0.054724	0.274924	0.029555	0.107133	0.100028	0.119603	1.000000

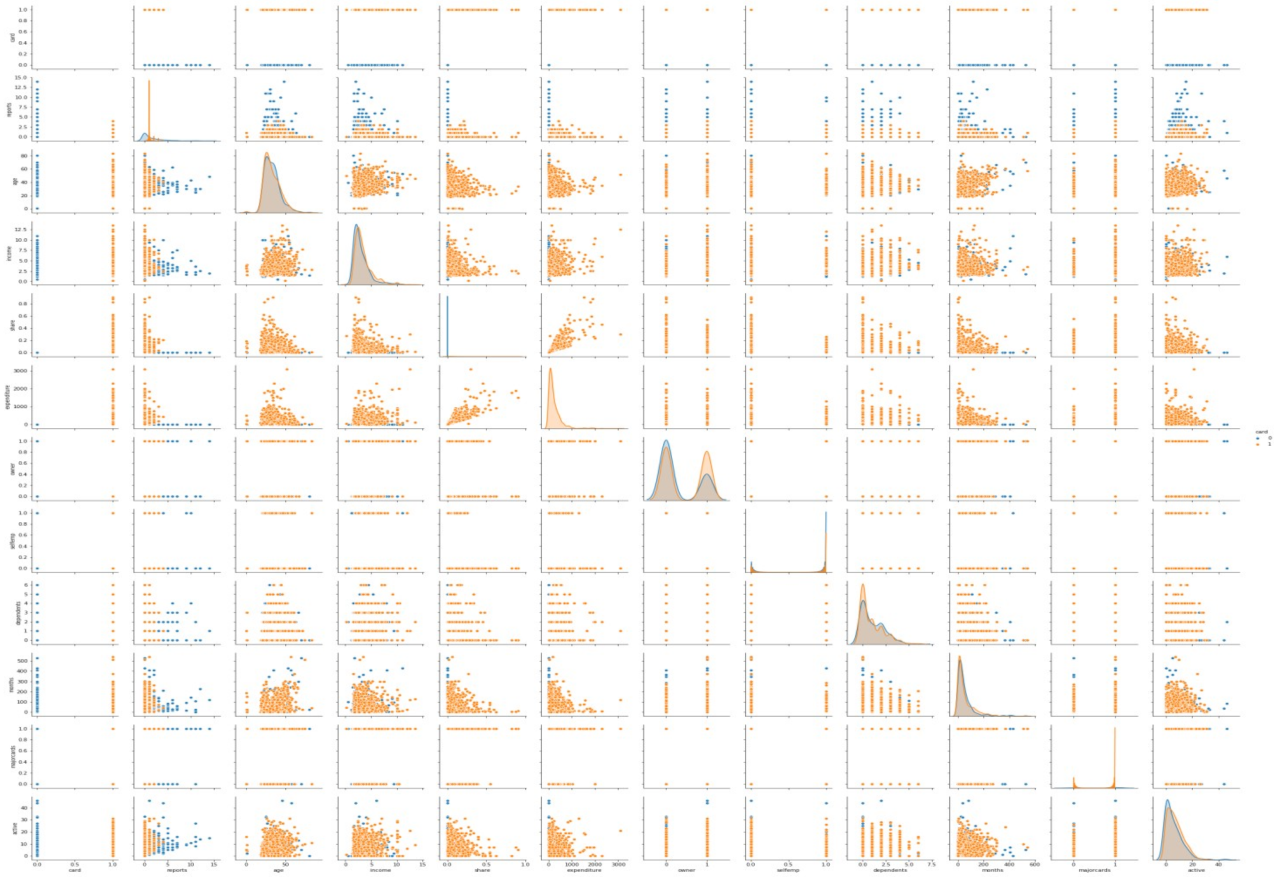
From this table, we can infer there are certain amount of outliers present in the dataset.

PAIRPLOT:

A pairplot allows us to see both distribution of single variables and relationships between two variables.

WITHOUT FEATURE EXTRACTION

```
sns.pairplot(df1,hue='card')
plt.savefig("PAIRPLOT1.png")
```



- **FEATURE SELECTION:**

```python
# view a list of the features and their importance scores
list(zip(X1_train,rnd1.feature_importances_))
```

```
[('reports', 0.08238557190146253),
 ('age', 0.02206311387499043),
 ('income', 0.03288657332268031 5),
 ('share', 0.5272706402701804),
 ('expenditure', 0.26469171570402655),
 ('owner', 0.006404219141486147 5),
 ('selfemp', 0.003500615717335109),
 ('dependents', 0.013451978914425265),
 ('months', 0.017474498207060173),
 ('majorcards', 0.007222305946972882),
 ('active', 0.022648766999380236)]
```

This list consists of different features and their important scores. Depending upon the scores we'll be extracting essential features which will be used in various Machine Learning models.

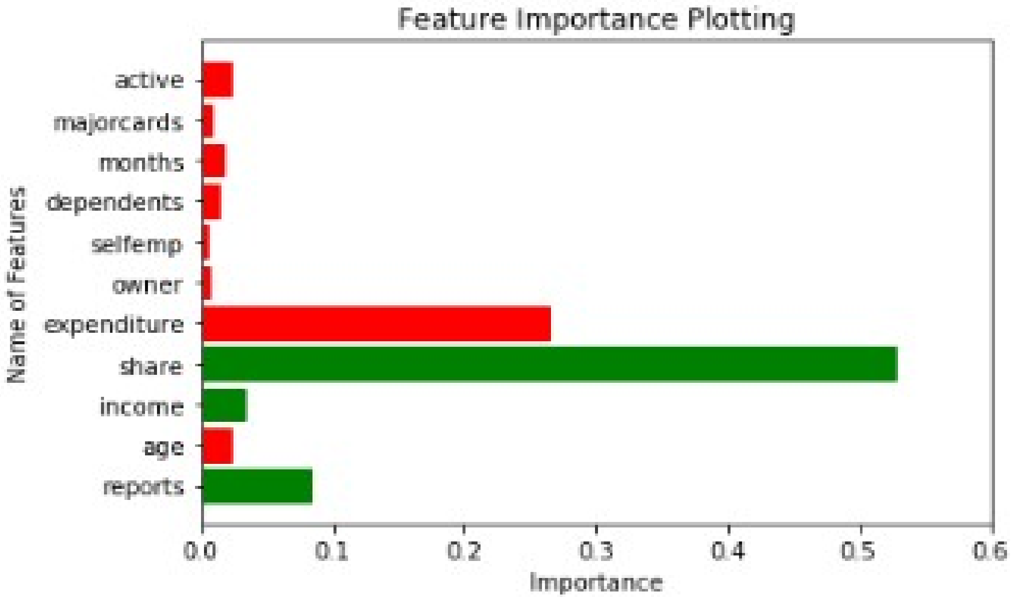
```
feature_list = list(df1.columns)
feature_list.remove('card')
importance_list = [0.08238557190146253,0.02206311387499043,0.03288657332268031 5,0.5272706402701804,0.26469171570402655,
     0.006404219141486 1475,0.003500615717335109,0.013451978914425265,0.017474498207060173,0.007222305946972882,
     0.022648766999380236]

plt.xlabel('Importance')
plt.ylabel('Name of Features')
plt.title("Feature Importance Plotting")

barlist = plt.barh(feature_list, importance_list)
plt.xlim([0,0.6])

barlist[0].set_color('g')
barlist[1].set_color('r')
barlist[2].set_color('g')
barlist[3].set_color('g')
barlist[4].set_color('r')
barlist[5].set_color('r')
barlist[6].set_color('r')
barlist[7].set_color('r')
barlist[8].set_color('r')
barlist[9].set_color('r')
barlist[10].set_color('r')

plt.show()
```



Random forest is the benchmark model model which determines the feature importance.

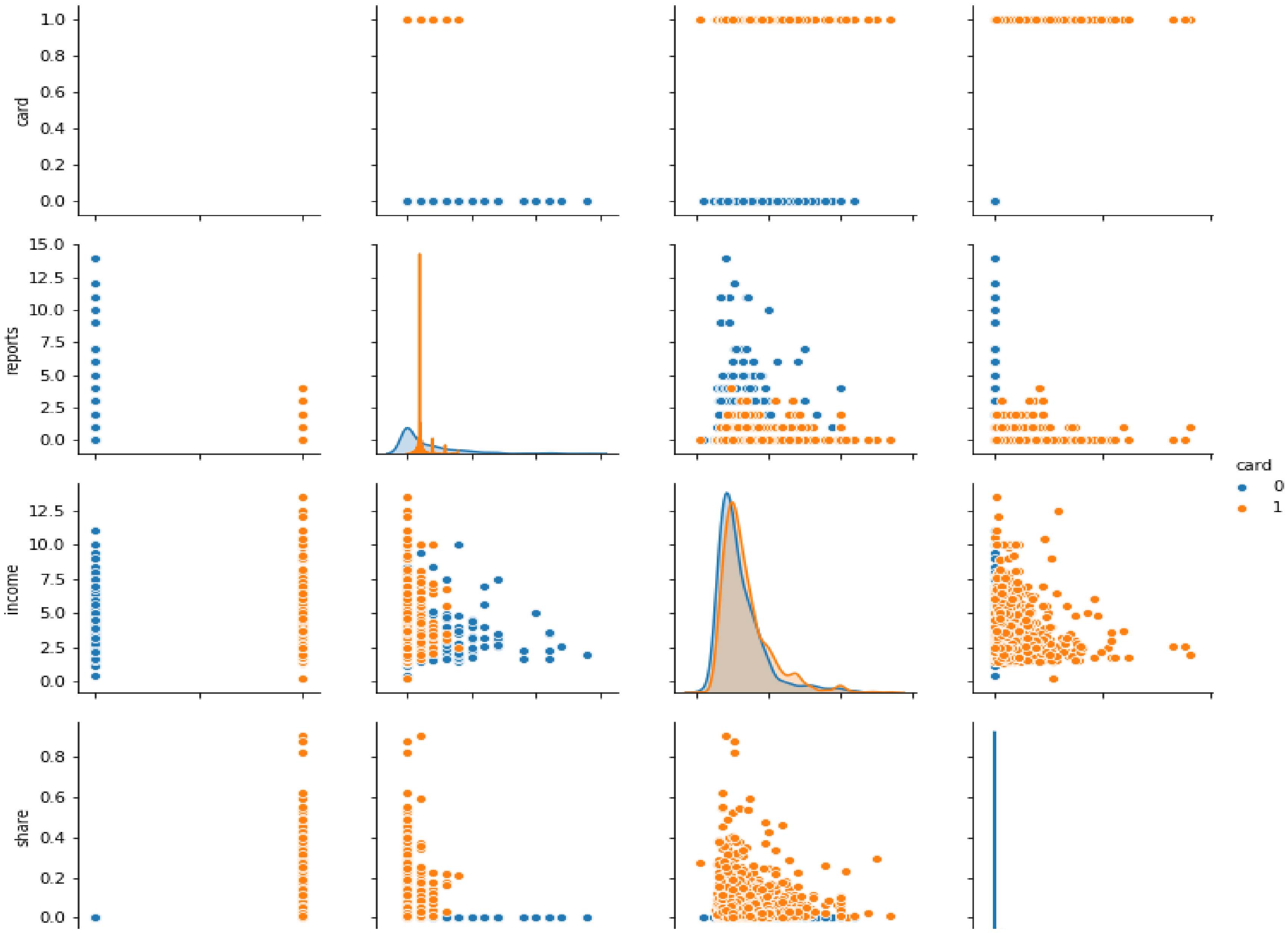
1. The red bar denotes the attributes of least importance so all of these attributes are not taken into account.
   The Expenditure attribute is a dummy variable and hence omitted for future Machine Learning operations.
2. The green bar denotes the attributes of highest importance and hence they are extracted for further Machine Learning operations.
   (Share,Income,Reports)

WITH FEATURE EXTRACTION:

```
leaks = ['expenditure','dependents','months','majorcards','owner','selfemp','active','age']
df2 = df1.drop(potential_leaks, axis=1)
df2.head()
```

	card	reports	income	share
0	1	0	4.5200	0.033270
1	1	0	2.4200	0.005217
2	1	0	4.5000	0.004156
3	1	0	2.5400	0.065214
4	1	0	9.7867	0.067051

```
sns.pairplot(df2,hue='card')
plt.savefig("PAIRPLOT2.png")
```


card
reports
income
share
1.0
0.8
0.6
0.4
0.2
0.0
15.0
12.5
10.0
7.5
5.0
2.5
0.0
12.5
10.0
7.5
5.0
2.5
0.0
0.8
0.6
0.4
0.2
0.0
card
0
1

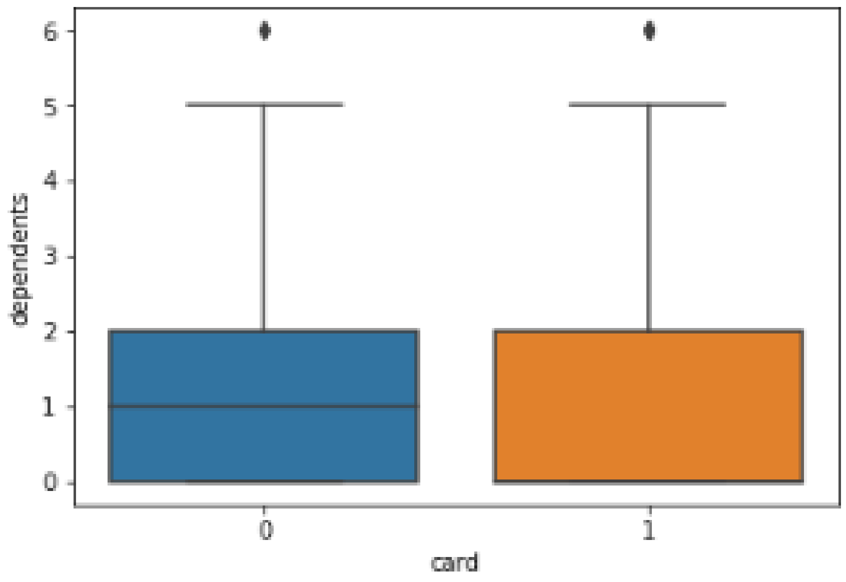
Box-plots for determining the outliers:

1. Boxplot of x='card' and y='dependents' to see the outliers:

```
# Boxplot of x='card' and y='dependents' to see the outliers
```

```
sns.boxplot(x='card',y='dependents',data=df1)
```

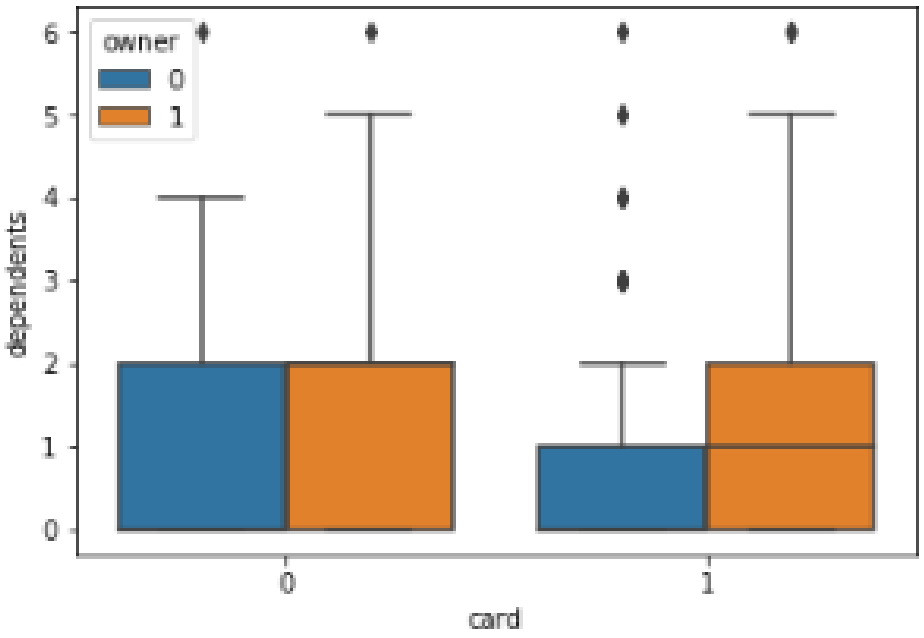
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f491f20d160>
```



2. **Boxplot of x='card' and y='dependents' to see the outliers depending on 'owner'.**

```
sns.boxplot(x='card',y='dependents',data=df1,hue='owner')
```

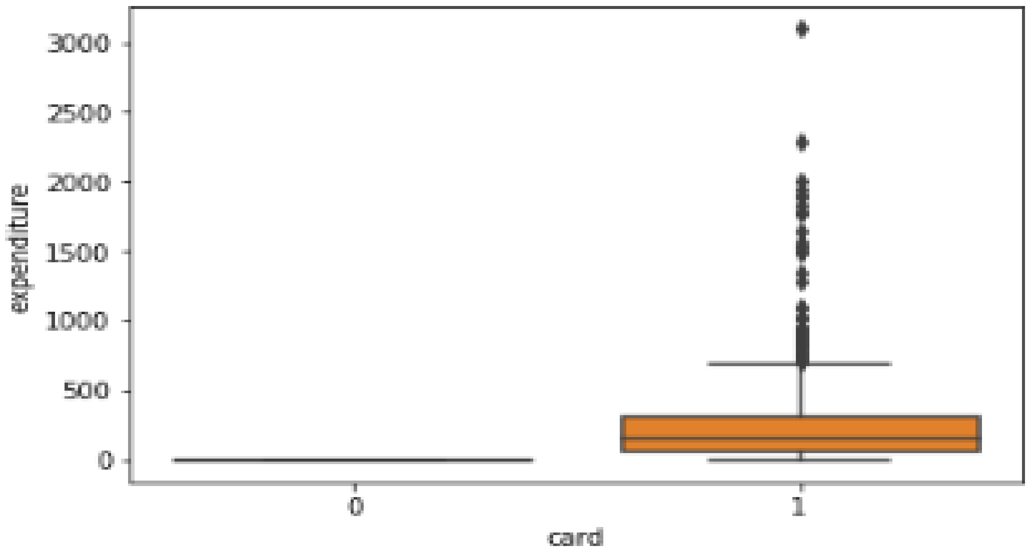
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f491f113a90>
```



3. Boxplot of x='card' and y='expenditure' to see the outliers.

```
sns.boxplot(x='card',y='expenditure',data=df1)
```

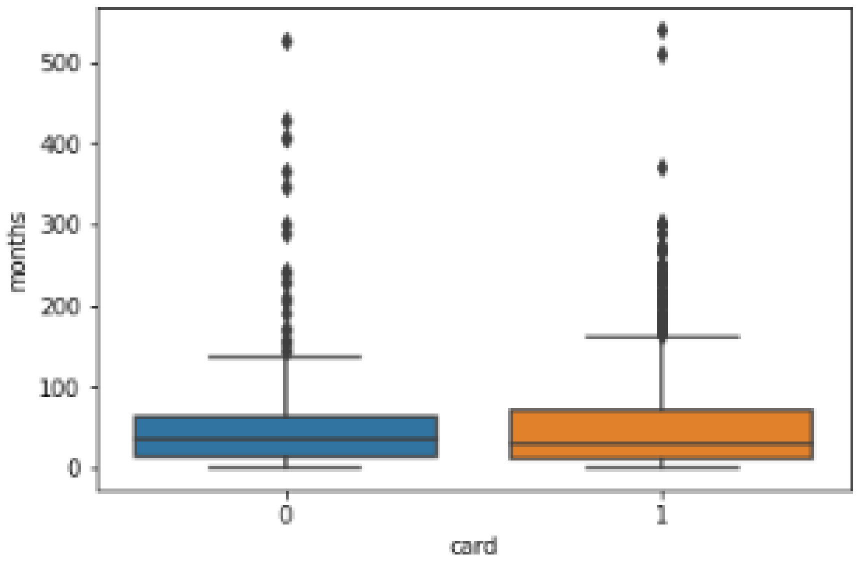
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f491efb12e8>
```



4. Boxplot of x='card' and y='months' to see the outliers.

```
sns.boxplot(x='card',y='months',data=df1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f491eed2a90>
```



Model Building

Short description of each model:

- **LOGISTIC REGRESSION:**

  Logistic regression is a technique borrowed by machine learning from the field of statistics. It is the go-to method for binary classification problems (problems with two class values). Techniques used to learn the coefficients of a logistic regression model from data.

Logistic Regression Without Feature Extraction

- Creating Independent Variables and Target Variable:

```
X1 = df1.drop('card',axis=1)
X1.head()

# DataFrame of Independent Variables
```

	reports	age	income	share	expenditure	owner	selfemp	dependents	months	majorcards	active
0	0	37.66667	4.5200	0.033270	124.983300	1	0	3	54	1	12
1	0	33.25000	2.4200	0.005217	9.854167	0	0	3	34	1	13
2	0	33.66667	4.5000	0.004156	15.000000	1	0	4	58	1	5
3	0	30.50000	2.5400	0.065214	137.869200	0	0	0	25	1	7
4	0	32.16667	9.7867	0.067051	546.503300	1	0	2	64	1	5

```
y1 = df1.card
y1.head()

# Target Variable
```

```
0    1
1    1
2    1
3    1
4    1
Name: card, dtype: int64
```

- Creating Test and Train DataSet:

```
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.25, random_state=101)
```

```
# Training The Machine Learning Model and showing the 'Coefficients', 'Intercept' and 'Score' of the model
```

```
log1 = LogisticRegression()
log1.fit(X1_train,y1_train)

print ("\nNumber of coefficients:",len(log1.coef_))
b1 = log1.coef_.reshape(11,1)
coeff1 = pd.DataFrame(data=b1,index=X1_train.columns,columns=['Coefficients'])

print ("\nCoefficients are:\n",coeff1)

print ("\nIntercept:",log1.intercept_)

score1 = log1.score(X1_train,y1_train)*100
print ("\nScore:",score1)
```

```
Number of coefficients: 1

Coefficients are:
              Coefficients
reports          -1.269563
age              -0.014531
income           -0.286945
share            -0.000651
expenditure       1.429178
owner             0.687162
selfemp           0.394337
dependents       -0.396870
months            0.001328
majorcards       -0.009204
active            0.043922

Intercept: [-1.0654274]

Score: 98.58442871587462
```

- Predicting the test Data:

```
pred1 = log1.predict(X1_test)
pred1[:5]
```

```
array([1, 1, 1, 1, 1])
```

```
pred_proba1 = log1.predict_proba(X1_test)
pred_proba1[:5]
```

```
array([[0.00000000e+00, 1.00000000e+00],
       [0.00000000e+00, 1.00000000e+00],
       [0.00000000e+00, 1.00000000e+00],
       [0.00000000e+00, 1.00000000e+00],
       [1.73295224e-08, 9.99999983e-01]])
```

- Creating Confusion Matrix:

```
pd.crosstab(index=y1_test,columns=pred1,rownames = ["Actual Value"],
            colnames = ["Predicted Value"])
```

Predicted Value	0	1
Actual Value		
0	64	0
1	9	257

```python
# Precision = TP/(TP + FP)
# Precision measures how many of the samples
# predicted as positive are actually positive
# Precision is also known as positive predictive value (PPV)

# Recall = TP/(TP + FN)
# measures how many of the positive samples are captured
# by the positive predictions:
# Other names for recall are sensitivity, hit rate,
# or true positive rate (TPR).

# F1-score = 2 x (precision x recall)/(precision + recall)
# f-score or f-measure, which is with the harmonic mean of
# precision and recall

print(metrics.classification_report(y1_test,pred1))
```

```
              precision    recall  f1-score   support

           0       0.88      1.00      0.93        64
           1       1.00      0.97      0.98       266

   micro avg       0.97      0.97      0.97       330
   macro avg       0.94      0.98      0.96       330
weighted avg       0.98      0.97      0.97       330
```

- For regression:

```python
print("MAE: ", metrics.mean_absolute_error(y1_test,pred1))
print("MSE: ",metrics.mean_squared_error(y1_test,pred1))
print("RMSE: ",np.sqrt(metrics.mean_squared_error(y1_test,pred1)))
```

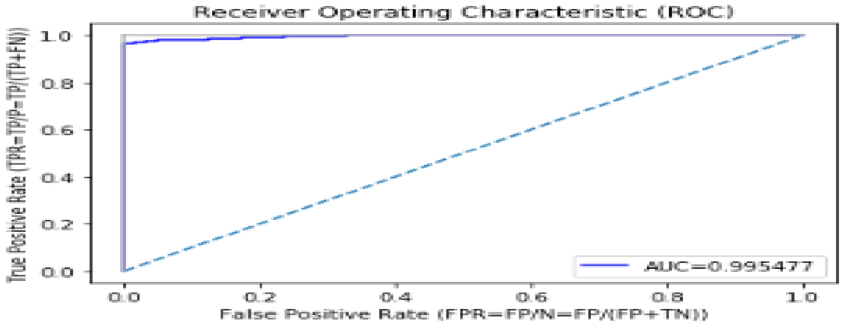
```
MAE:  0.02727272727272727
MSE:  0.02727272727272727
RMSE:  0.1651445647689541
```

- Plotting ROC curve:

```
# Plot Receiving Operating Characteristic Curve
# Create true and false positive rates

y1_score = log1.predict_proba(X1_test)[:,1]
false_positive_rate, true_positive_rate, threshold = metrics.roc_curve(y1_test, y1_score)

# Plot ROC curve
plt.title('Receiver Operating Characteristic (ROC)')
roc_auc = metrics.auc(false_positive_rate,true_positive_rate)
plt.plot(false_positive_rate, true_positive_rate,'b',label="AUC=%f"%roc_auc)
plt.legend(loc='lower right')
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7")
plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate (TPR=TP/P=TP/(TP+FN))')
plt.xlabel('False Positive Rate (FPR=FP/N=FP/(FP+TN))')
plt.show()
```



Linear Regression With Feature Extraction¶

- **Importing Pre-processed DataSet:**

```
df2 = pd.read_csv('Pre-processed Data.csv')
df2.head()
```

	card	reports	income	share
0	1	0	4.5200	0.033270
1	1	0	2.4200	0.005217
2	1	0	4.5000	0.004156
3	1	0	2.5400	0.065214
4	1	0	9.7867	0.067051

- **Creating Independent Variabales and Target Variable**

```
x2 = df2.drop('card',axis=1)
x2.head()

# Indpendent Variabales
```

	reports	income	share
0	0	4.5200	0.033270
1	0	2.4200	0.005217
2	0	4.5000	0.004156
3	0	2.5400	0.065214
4	0	9.7867	0.067051

```
y2 = df2.card
y2.head()

# Target Variable
```

```
0    1
1    1
2    1
3    1
4    1
Name: card, dtype: int64
```

- Creating Test and Train DataSet:

```
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.25, random_state=101)
```

```
# Training The Machine Learning Model and showing the 'Coefficients', 'Intercept' and 'Score' of the model
```

```
log2 = LogisticRegression()
log2.fit(X2_train,y2_train)

print ("\nNumber of coefficients:",len(log2.coef_))
b2 = log2.coef_.reshape(3,1)
coeff2 = pd.DataFrame(data=b2,index=X2_train.columns,columns=['Coefficients'])
print ("\nCoefficients are:\n",coeff2)

print ("\nIntercept:",log2.intercept_)

score2 = log2.score(X2_train,y2_train)*100
print ("\nScore:",score2)
```

```
Number of coefficients: 1

Coefficients are:
          Coefficients
reports     -1.223600
income       0.259417
share        7.245085

Intercept: [0.53297186]

Score: 85.9453993933266
```

- Predicting the test Data:

```
pred2 = log2.predict(X2_test)
pred2[0:5]
```

```
array([1, 1, 1, 1, 1])
```

```
pred_proba2 = log2.predict_proba(X2_test)
pred_proba2[:5]
```

```
array([[0.46140387, 0.53859613],
       [0.00264755, 0.99735245],
       [0.16378919, 0.83621081],
       [0.14125941, 0.85874059],
       [0.26320839, 0.73679161]])
```

```
# Creating Confussion Matrix
```

```
pd.crosstab(index=y2_test,columns=pred2,rownames = ["Actual Value"],
            colnames = ["Predicted Value"])
```

Predicted Value	0	1
Actual Value		
0	25	39
1	4	262

```python
# Precision = TP/(TP + FP)
# Precision measures how many of the samples
# predicted as positive are actually positive
# Precision is also known as positive predictive value (PPV)

# Recall = TP/(TP + FN)
# measures how many of the positive samples are captured
# by the positive predictions:
# Other names for recall are sensitivity, hit rate,
# or true positive rate (TPR).

# F1-score = 2 x (precision x recall)/(precision + recall)
# f-score or f-measure, which is with the harmonic mean of
# precision and recall

print(metrics.classification_report(y2_test,pred2))
```

```
              precision    recall  f1-score   support

           0       0.86      0.39      0.54        64
           1       0.87      0.98      0.92       266

   micro avg       0.87      0.87      0.87       330
   macro avg       0.87      0.69      0.73       330
weighted avg       0.87      0.87      0.85       330
```

- For regression:

```python
print("MAE: ", metrics.mean_absolute_error(y2_test,pred2))
print("MSE: ",metrics.mean_squared_error(y2_test,pred2))
print("RMSE: ",np.sqrt(metrics.mean_squared_error(y2_test,pred2)))
```

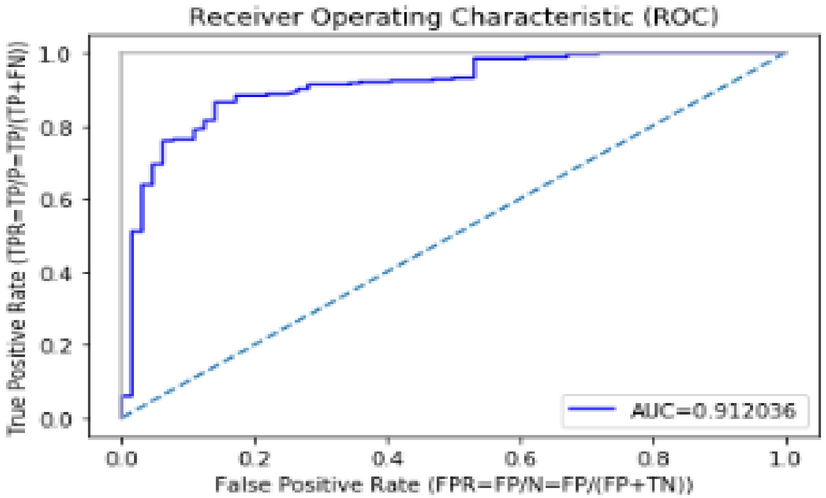
```
MAE:  0.13030303030303
MSE:  0.13030303030303
RMSE:  0.36097511036500884
```

- For ROC curve:

```python
# Plot Receiving Operating Characteristic Curve
# Create true and false positive rates

y2_score = log2.predict_proba(X2_test)[:,1]
false_positive_rate, true_positive_rate, threshold = metrics.roc_curve(y2_test, y2_score)

# Plot ROC curve
plt.title('Receiver Operating Characteristic (ROC)')
roc_auc = metrics.auc(false_positive_rate,true_positive_rate)
plt.plot(false_positive_rate, true_positive_rate,'b',label="AUC=%f"%roc_auc)
plt.legend(loc='lower right')
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7")
plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate (TPR=TP/P=TP/(TP+FN))')
plt.xlabel('False Positive Rate (FPR=FP/N=FP/(FP+TN))')
plt.show()
```



2. DECISION TREE:

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves.

- Creating Indpendent Variabales and Target Variable:

```python
X2 = df2.drop('card',axis=1)
X2.head()

# Indpendent Variabales
```

	reports	income	share
0	0	4.5200	0.033270
1	0	2.4200	0.005217
2	0	4.5000	0.004156
3	0	2.5400	0.065214
4	0	9.7867	0.067051

```python
y2 = df2.card
y2.head()

# Target Variable
```

```
0    1
1    1
2    1
3    1
4    1
Name: card, dtype: int64
```

- Creating Test and Train DataSet:

```
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.25, random_state=101)
```

```
# Training The Machine Learning Model and showing the 'Coefficients', 'Intercept' and 'Score' of the model
```

```
dtree2 = DecisionTreeClassifier(random_state = 100,max_depth=3,min_samples_leaf=3)
dtree2.fit(X2_train,y2_train)

score2 = dtree2.score(X2_train,y2_train)*100
print ("\nScore:",score2)
```

```
Score: 98.48331648129424
```

- Predicting the test Data:

```
pred2 = dtree2.predict(X2_test)
pred2[0:5]
```

```
array([1, 1, 1, 1, 1])
```

```
pred_proba2 = dtree2.predict_proba(X2_test)
pred_proba2[:5]
```

```
array([[0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.]])
```

```
# Creating Confussion Matrix
```

```
pd.crosstab(index=y2_test,columns=pred2,rownames = ["Actual Value"],
            colnames = ["Predicted Value"])
```

Predicted Value	0	1
Actual Value		
0	64	0
1	0	357

```python
# Precision = TP/(TP + FP)
# Precision measures how many of the samples
# predicted as positive are actually positive
# Precision is also known as positive predictive value (PPV)

# Recall = TP/(TP + FN)
# measures how many of the positive samples are captured
# by the positive predictions:
# Other names for recall are sensitivity, hit rate,
# or true positive rate (TPR).

# F1-score = 2 x (precision x recall)/(precision + recall)
# f-score or f-measure, which is with the harmonic mean of
# precision and recall

print(metrics.classification_report(y2_test,pred2))
```

```
              precision    recall  f1-score   support

           0       0.88      1.00      0.93        64
           1       1.00      0.97      0.98       266

   micro avg       0.97      0.97      0.97       330
   macro avg       0.94      0.98      0.96       330
weighted avg       0.98      0.97      0.97       330
```

•

```
print("MAE: ", metrics.mean_absolute_error(y2_test,pred2))
print("MSE: ",metrics.mean_squared_error(y2_test,pred2))
print("RMSE: ",np.sqrt(metrics.mean_squared_error(y2_test,pred2)))
```

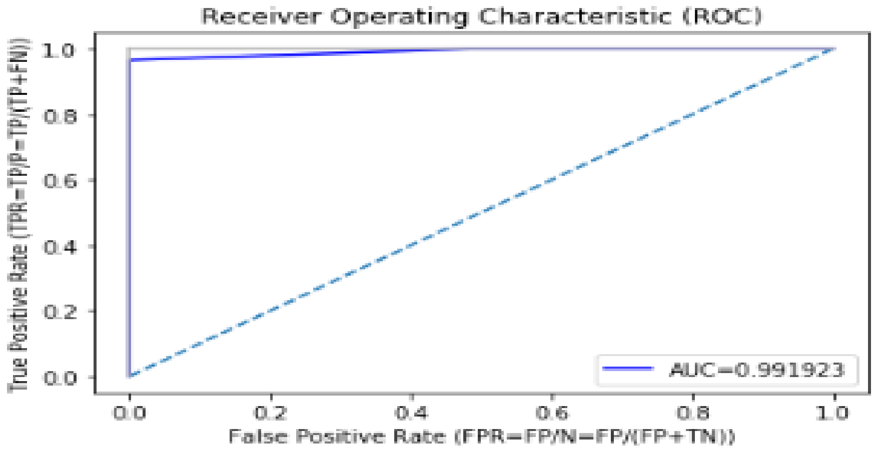
```
MAE:  0.02727272727272727
MSE:  0.02727272727272727
RMSE:  0.1651445647689541
```

- ROC curve:

```
# Plot Receiving Operating Characteristic Curve
# Create true and false positive rates

y2_score = dtree2.predict_proba(X2_test)[:,1]
false_positive_rate, true_positive_rate, threshold = metrics.roc_curve(y2_test, y2_score)

# Plot ROC curve
plt.title('Receiver Operating Characteristic (ROC)')
roc_auc = metrics.auc(false_positive_rate,true_positive_rate)
plt.plot(false_positive_rate, true_positive_rate,'b',label="AUC=%f"%roc_auc)
plt.legend(loc='lower right')
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7")
plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate (TPR=TP/P=TP/(TP+FN))')
plt.xlabel('False Positive Rate (FPR=FP/N=FP/(FP+TN))')
plt.show()
```



3. NAÏVE BAYES:

In machine learning, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features. Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

Gaussian Naive Bayes With Feature Extraction

- Creating Independent Variables and Target Variable:

```
X2 = df2.drop('card',axis=1)
X2.head()

# Indpendent Variabales
```

	reports	income	share
0	0	4.5200	0.033270
1	0	2.4200	0.005217
2	0	4.5000	0.004156
3	0	2.5400	0.065214
4	0	9.7867	0.067051

```
y2 = df2.card
y2.head()

# Target Variable
```

```
0    1
1    1
2    1
3    1
4    1
Name: card, dtype: int64
```

- Creating Test and Train DataSet and training the model :

```
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.25, random_state=101)
```

```
# Training The Machine Learning Model and showing 'Score' of the model
```

```
gnb2 = GaussianNB()
gnb2.fit(X2_train,y2_train)
score2 = gnb2.score(X2_train,y2_train)*100
print ("\nScore:", score2)
```

```
Score: 98.38220424671385
```

- Predicting the test Data:

```python
pred2 = gnb2.predict(X2_test)
pred2[0:5]
```

```
array([1, 1, 1, 1, 1])
```

```python
pred_proba2 = gnb2.predict_proba(X2_test)
pred_proba2[:5]
```

```
array([[0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.]])
```

```python
pd.crosstab(index=y2_test,columns=pred2,rownames = ["Actual Value"],
            colnames = ["Predicted Value"])
```

Predicted Value	0	1
Actual Value		
0	63	1
1	9	257

```python
# Precision = TP/(TP + FP)
# Precision measures how many of the samples
# predicted as positive are actually positive
# Precision is also known as positive predictive value (PPV)

# Recall = TP/(TP + FN)
# measures how many of the positive samples are captured
# by the positive predictions:
# Other names for recall are sensitivity, hit rate,
# or true positive rate (TPR).

# F1-score = 2 x (precision x recall)/(precision + recall)
# f-score or f-measure, which is with the harmonic mean of
# precision and recall

print(metrics.classification_report(y2_test,pred2))
```

```
             precision    recall  f1-score   support

          0       0.88      0.98      0.93        64
          1       1.00      0.97      0.98       266

  micro avg       0.97      0.97      0.97       330
  macro avg       0.94      0.98      0.95       330
weighted avg      0.97      0.97      0.97       330
```

- Errors:

```python
# Errors
print("MAE: ", metrics.mean_absolute_error(y2_test,pred2))
print("MSE: ",metrics.mean_squared_error(y2_test,pred2))
print("RMSE: ",np.sqrt(metrics.mean_squared_error(y2_test,pred2)))
```

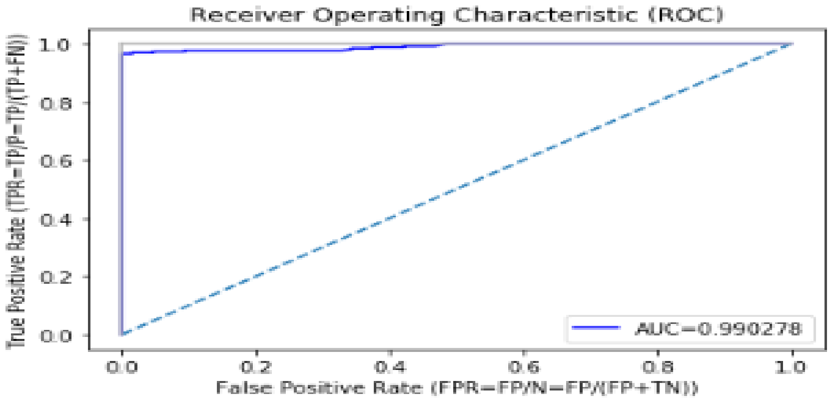
```
MAE:  0.030303030303030304
MSE:  0.030303030303030304
RMSE:  0.17407765595569785
```

- ROC Curve:

```
# Plot Receiving Operating Characteristic Curve
# Create true and false positive rates

y2_score = gnb2.predict_proba(X2_test)[:,1]
false_positive_rate, true_positive_rate, threshold = metrics.roc_curve(y2_test, y2_score)

# Plot ROC curve
plt.title('Receiver Operating Characteristic (ROC)')
roc_auc = metrics.auc(false_positive_rate,true_positive_rate)
plt.plot(false_positive_rate, true_positive_rate,'b',label="AUC=%f"%roc_auc)
plt.legend(loc='lower right')
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7")
plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate (TPR=TP/P=TP/(TP+FN))')
plt.xlabel('False Positive Rate (FPR=FP/N=FP/(FP+TN))')
plt.show()
```



4. K-NN:

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression.[1] In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

1. In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

2. In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors.

KNN With Feature Extraction

- Creating Independent Variables and Target Variable**:**

```
X2 = df2.drop('card',axis=1)
X2.head()

# Indpendent Variabales
```

	reports	income	share
0	0	4.5200	0.033270
1	0	2.4200	0.005217
2	0	4.5000	0.004156
3	0	2.5400	0.065214
4	0	9.7867	0.067051

```
y2 = df2.card
y2.head()

# Target Variable
```

```
0    1
1    1
2    1
3    1
4    1
Name: card, dtype: int64
```

- Identifying Best Values for K:

```
# Standardize Data
# Create standardizer
standardizer = StandardScaler()
```

```
# Standardize features
X_std = standardizer.fit_transform(X2)
print (X_std[:5])
```

```
[[-0.3393968   0.68189427 -0.3747874 ]
 [-0.3393968  -0.55831728 -0.67126874]
 [-0.3393968   0.67008274 -0.68248613]
 [-0.3393968  -0.48744805 -0.03718458]
 [-0.3393968   3.7922858  -0.01777202]]
```

- Creating Test and Train DataSet and training the model:

```
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.25, random_state=101)
```

```
# Training The Machine Learning Model and showing 'Score' of the model
```

```
knn2 = KNeighborsClassifier(n_neighbors=5,metric='euclidean',n_jobs=2)
knn2.fit(X2_train,y2_train)
score2 = knn2.score(X2_train,y2_train)*100
print ("\nScore:", score2)
```

```
Score: 91.10212335692619
```

- Predicting the test Data:

```
pred2 = knn2.predict(X2_test)
pred2[0:5]
```

```
array([1, 1, 1, 1, 0])
```

```
pred_proba2 = knn2.predict_proba(X2_test)
pred_proba2[:5]
```

```
array([[0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.]])
```

```
pd.crosstab(index=y2_test,columns=pred2,rownames = ["Actual Value"],
            colnames = ["Predicted Value"])
```

Predicted Value	0	1
Actual Value		
0	43	21
1	24	242

```python
# Precision = TP/(TP + FP)
# Precision measures how many of the samples
# predicted as positive are actually positive
# Precision is also known as positive predictive value (PPV)

# Recall = TP/(TP + FN)
# measures how many of the positive samples are captured
# by the positive predictions:
# Other names for recall are sensitivity, hit rate,
# or true positive rate (TPR).

# F1-score = 2 x (precision x recall)/(precision + recall)
# f-score or f-measure, which is with the harmonic mean of
# precision and recall

print(metrics.classification_report(y2_test,pred2))
```

```
              precision    recall  f1-score   support

           0       0.64      0.67      0.66        64
           1       0.92      0.91      0.91       266

   micro avg       0.86      0.86      0.86       330
   macro avg       0.78      0.79      0.79       330
weighted avg       0.87      0.86      0.86       330
```

- Errors:

```python
print("MAE: ", metrics.mean_absolute_error(y2_test,pred2))
print("MSE: ",metrics.mean_squared_error(y2_test,pred2))
print("RMSE: ",np.sqrt(metrics.mean_squared_error(y2_test,pred2)))
```

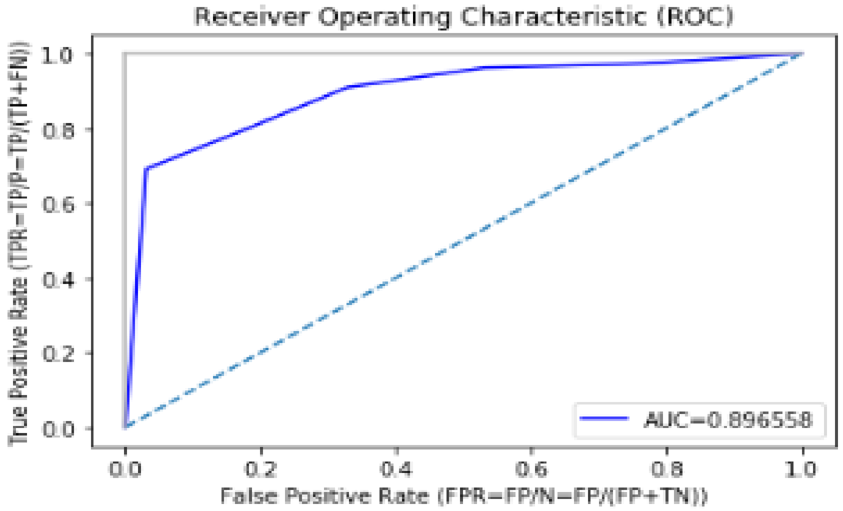
```
MAE:  0.13636363636363635
MSE:  0.13636363636363635
RMSE:  0.3692744729379982
```

- ROC Curve:

```
# Plot Receiving Operating Characteristic Curve
# Create true and false positive rates

y2_score = knn2.predict_proba(X2_test)[:,1]
false_positive_rate, true_positive_rate, threshold = metrics.roc_curve(y2_test, y2_score)

# Plot ROC curve
plt.title('Receiver Operating Characteristic (ROC)')
roc_auc = metrics.auc(false_positive_rate,true_positive_rate)
plt.plot(false_positive_rate, true_positive_rate,'b',label="AUC=%f"%roc_auc)
plt.legend(loc='lower right')
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7")
plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate (TPR=TP/P=TP/(TP+FN))')
plt.xlabel('False Positive Rate (FPR=FP/N=FP/(FP+TN))')
plt.show()
```



5. RANDOM FOREST MODEL:

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.[1][2] Random decision forests correct for decision trees' habit of overfitting to their training set.

Random Forest With Feature Extraction

- Creating Indpendent Variabales and Target Variable:

```
X2 = df2.drop('card',axis=1)
X2.head()

# Indpendent Variabales
```

	reports	income	share
0	0	4.5200	0.033270
1	0	2.4200	0.005217
2	0	4.5000	0.004156
3	0	2.5400	0.065214
4	0	9.7867	0.067051

```
y2 = df2.card
y2.head()

# Target Variable
```

```
0    1
1    1
2    1
3    1
4    1
Name: card, dtype: int64
```

- Creating Test and Train DataSet:

```
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.25, random_state=101)
```

```
# Training The Machine Learning Model and showing the 'Coefficients', 'Intercept' and 'Score' of the model
```

```
rnd2 = RandomForestClassifier(n_jobs=2,random_state=0)
rnd2.fit(X2_train,y2_train)

score2 = rnd2.score(X2_train,y2_train)*100
print ("\nScore:",score2)
```

```
Score: 99.19110212335693
```

- Predicting the test Data:

```
pred2 = rnd2.predict(X2_test)
pred2[0:5]
```

```
array([1, 1, 1, 1, 1])
```

```
pred_proba2 = rnd2.predict_proba(X2_test)
pred_proba2[:5]
```

```
array([[0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0.1, 0.9]])
```

```
# Creating Confussion Matrix
```

```
pd.crosstab(index=y2_test,columns=pred2,rownames = ["Actual Value"],
            colnames = ["Predicted Value"])
```

Predicted Value	0	1
Actual Value		
0	60	4
1	9	257

```
# Precision = TP/(TP + FP)
# Precision measures how many of the samples
# predicted as positive are actually positive
# Precision is also known as positive predictive value (PPV)

# Recall = TP/(TP + FN)
# measures how many of the positive samples are captured
# by the positive predictions:
# Other names for recall are sensitivity, hit rate,
# or true positive rate (TPR).

# F1-score = 2 x (precision x recall)/(precision + recall)
# f-score or f-measure, which is with the harmonic mean of
# precision and recall

print(metrics.classification_report(y2_test,pred2))
```

```
              precision    recall  f1-score   support

           0       0.87      0.94      0.90        64
           1       0.98      0.97      0.98       266

   micro avg       0.96      0.96      0.96       330
   macro avg       0.93      0.95      0.94       330
weighted avg       0.96      0.96      0.96       330
```

- Error:

```
print("MAE: ", metrics.mean_absolute_error(y2_test,pred2))
print("MSE: ",metrics.mean_squared_error(y2_test,pred2))
print("RMSE: ",np.sqrt(metrics.mean_squared_error(y2_test,pred2)))
```

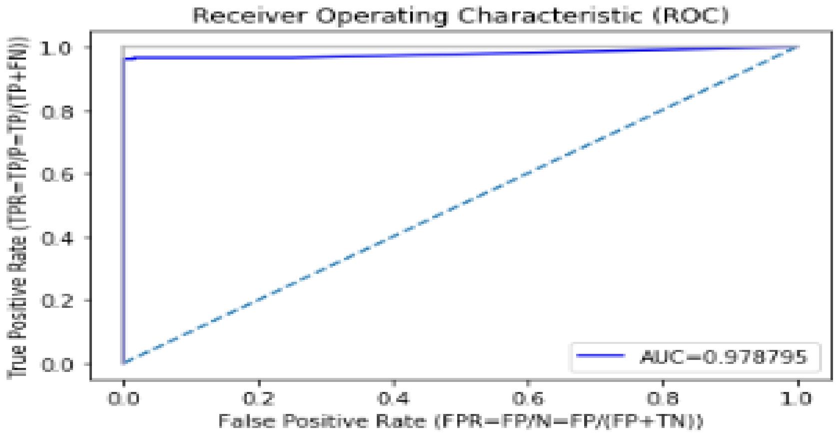
```
MAE:  0.03939393939393939
MSE:  0.03939393939393939
RMSE:  0.19847906537954926
```

- ROC Curve:

```
# Plot Receiving Operating Characteristic Curve
# Create true and false positive rates

y2_score = rnd2.predict_proba(X2_test)[:,1]
false_positive_rate, true_positive_rate, threshold = metrics.roc_curve(y2_test, y2_score)

# Plot ROC curve
plt.title('Receiver Operating Characteristic (ROC)')
roc_auc = metrics.auc(false_positive_rate,true_positive_rate)
plt.plot(false_positive_rate, true_positive_rate,'b',label="AUC=%f"%roc_auc)
plt.legend(loc='lower right')
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7")
plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate (TPR=TP/P=TP/(TP+FN))')
plt.xlabel('False Positive Rate (FPR=FP/N=FP/(FP+TN))')
plt.show()
```



Accuracy Plotting of Classifiers:

```
# Importing Libraries
```

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Gathering Data
```

```
accuracy_dtr1 = 97.27273 # Accuracy of Decision Tree Classification Model 1
accuracy_dtr2 = 97.27273 # Accuracy of Decision Tree Classification Model 2

accuracy_gnb1 = 97.27273 # Accuracy of Gaussian Naive Bayes Classification Model 1
accuracy_gnb2 = 96.96970 # Accuracy of Gaussian Naive Bayes Classification Model 2

accuracy_knn1 = 94.84848 # Accuracy of K-Nearest Neabour Classification Model 1
accuracy_knn2 = 86.36364 # Accuracy of K-Nearest Neabour Classification Model 2

accuracy_log1 = 97.27273 # Accuracy of Logistic Regression Classification Model 1
accuracy_log2 = 86.96970 # Accuracy of Logistic Regression Classification Model 2

accuracy_rnd1 = 97.27273 # Accuracy of Random Forest Classification Model 1
accuracy_rnd2 = 96.06061 # Accuracy of Random Forest Classification Model 1
```

```
# Creating DataFrame of Accuracies
```

```
accuracy_set1 = np.array([[accuracy_dtr1, accuracy_gnb1, accuracy_knn1, accuracy_log1, accuracy_rnd1],
                          [accuracy_dtr2, accuracy_gnb2, accuracy_knn2, accuracy_log2, accuracy_rnd2]])

columns_list = ['Decision Tree','Gaussian Naive Bayes','K-Nearest Neabour','Logistic Regression','Random Forest']

accuracy = pd.DataFrame(data=accuracy_set1, index=['Accuracy 1','Accuracy 2'], columns=columns_list)
accuracy
```

	Decision Tree	Gaussian Naive Bayes	K-Nearest Neabour	Logistic Regression	Random Forest
Accuracy 1	97.27273	97.27273	94.84848	97.27273	97.27273
Accuracy 2	97.27273	96.96970	86.36364	86.96970	96.06061

```
# Saving the output as CSV file
```

```
accuracy.to_csv('Accuracies of Models.csv',index_label=['Accuracy 1','Accuracy 2'],index=False)
```

This table consists of two types of accuracies.

Accuracy1: This accuracy is calculated without feature extraction.

Accuracy2:This accuracy is calculated with feature extraction.

And output is stored as a .csv file

Plotting Accuracies

Model1 : Without feature extraction

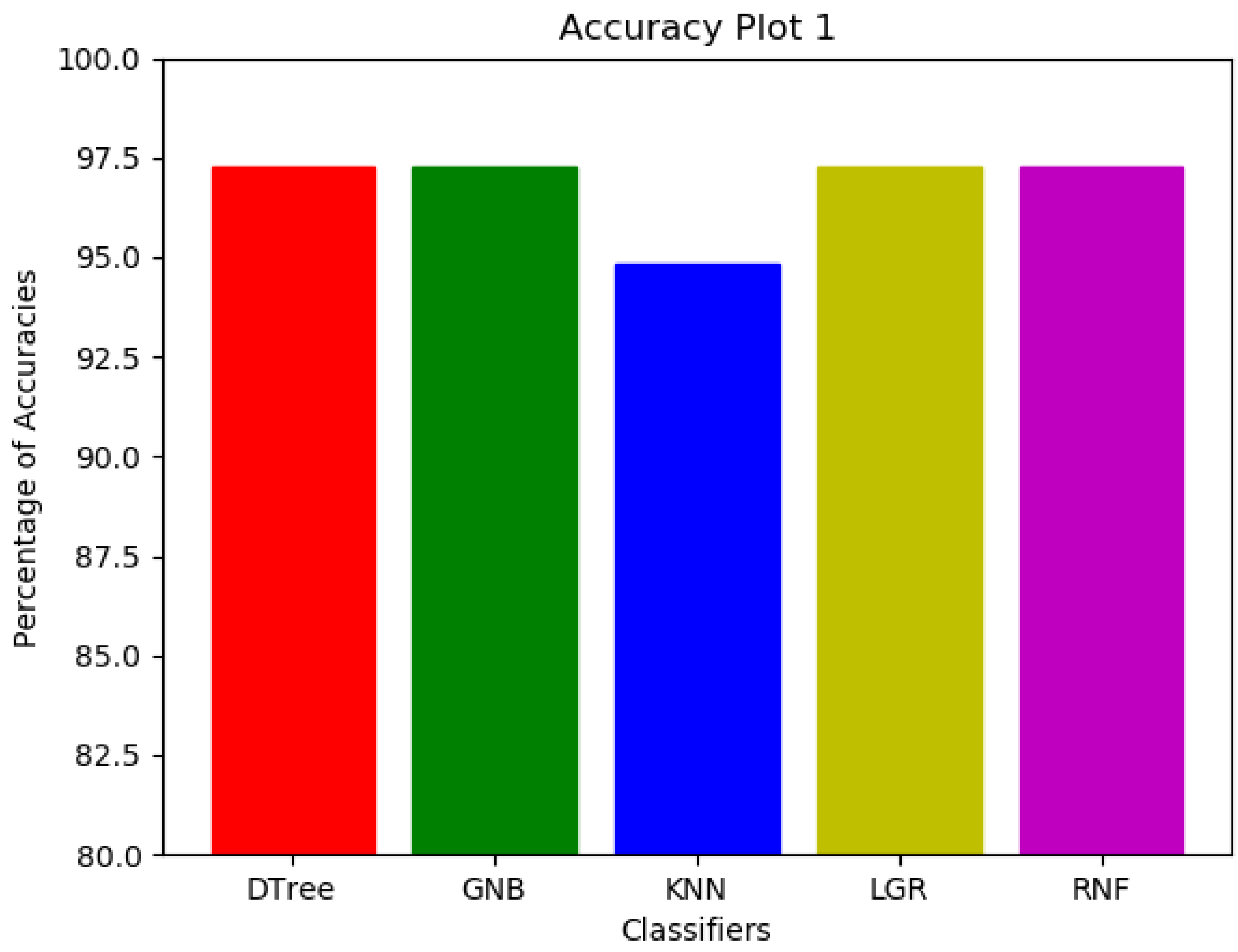
```
accuracy_set1 = [accuracy_dtr1, accuracy_gnb1, accuracy_knn1, accuracy_log1, accuracy_rnd1]
class_list = ['DTree','GNB','KNN','LGR','RNF']

plt.xlabel('Classifiers')
plt.ylabel('Percentage of Accuracies')
plt.title("Accuracy Plot 1")

barlist = plt.bar(class_list, accuracy_set1, width=0.8)
plt.ylim([80,100])

barlist[0].set_color('r')
barlist[1].set_color('g')
barlist[2].set_color('b')
barlist[3].set_color('y')
barlist[4].set_color('m')

plt.show()
# plt.savefig('ACCURACY PLOT 1.png')
```


Accuracy Plot 1
100.0
97.5
95.0
92.5
90.0
87.5
85.0
82.5
80.0
Percentage of Accuracies
DTree
GNB
KNN
LGR
RNF
Classifiers

Model2: With feature extraction

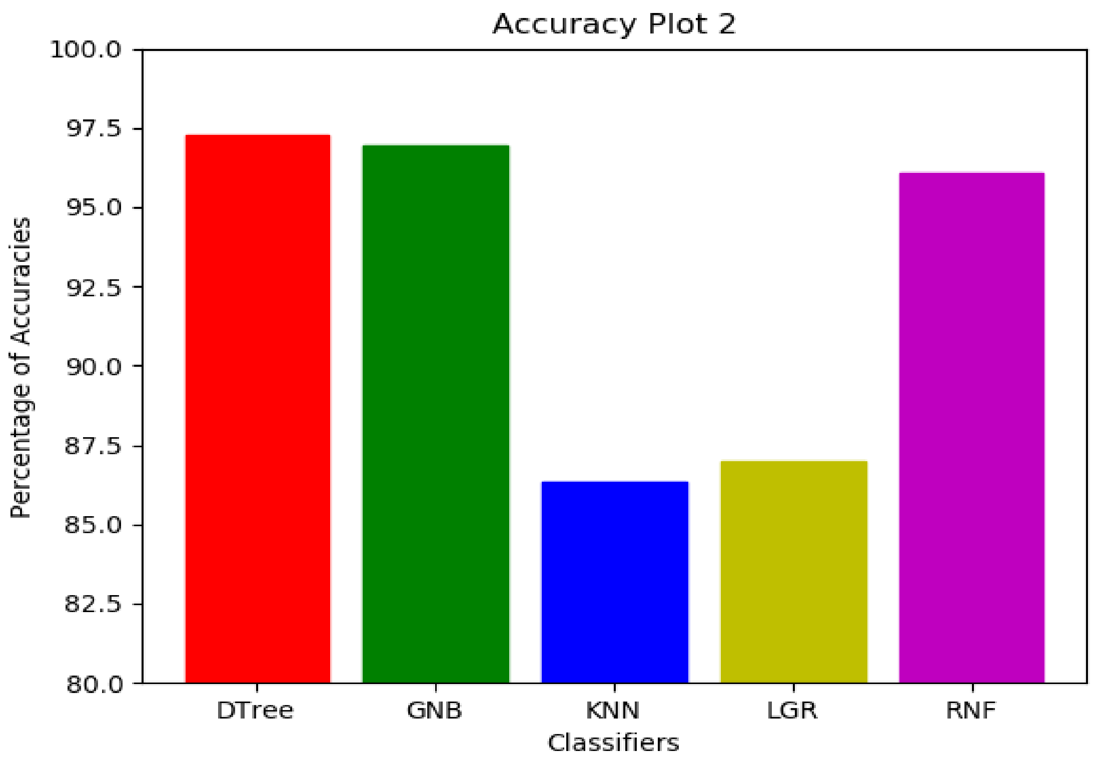
```
class_list = ['DTree','GNB','KNN','LGR','RNF']

plt.xlabel('Classifiers')
plt.ylabel('Percentage of Accuracies')
plt.title("Accuracy Plot 2")

barlist = plt.bar(class_list, accuracy.loc['Accuracy 2',:], width=0.8)
plt.ylim([80,100])

barlist[0].set_color('r')
barlist[1].set_color('g')
barlist[2].set_color('b')
barlist[3].set_color('y')
barlist[4].set_color('m')

plt.show()
# plt.savefig('ACCURACY PLOT 1.png')
```



CONCLUSION:

The best model for the prediction analysis is Decision Tree with an accuracy percentage of **97.27273%**

Future Scope of Improvements

- Data is a single most important asset.Essentially we need to know some details about some customers and the given information.

- For a data set which is containing various information will be helpful for the user to choose the best way to justify application

Certificate

This is to certify that Mr. SUVAM DAS of Government College of Engineering and Ceramic Technology, registration number: 161130110072, has successfully completed a project on Predict Credit Card Acceptance using Machine Learning with Python under the guidance of Mr. Arnab Chakraborty.

Mr. Arnab Chakrabotry

Globsyn Finishing School

Certificate

This is to certify that Mr. PRIYAM MUKHERJEE of Government College of Engineering and Ceramic Technology, registration number: 161130110057 of 2016-17, has successfully completed a project on PREDICT CREDIT CARD ACCEPTANCE using Machine Learning with Python under the guidance of Mr. Arnab Chakraborty .

Mr. Arnab Chakrabotry

Globsyn Finishing School

Certificate

Certificate

This is to certify that Mr SHAWAN BASU of Government College of Engineering and Ceramic Technology, registration number: 161130110064 of 2016-17, successfully completed a project on PREDICT CREDIT CARD ACCEPTANCE using Machine Learning with Python under the guidance of Mr. Arnab Chakraborty.

Mr. Arnab Chakraborty

Globsyn Finishing School

Certificate

This is to certify that Ms Riya Karan registration number: 161130110020 OF 2016-2017, has successfully completed a project on PREDICT CREDIT CARD ACCEPTANCE using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty.

Globsyn Finishing School