```python
import numpy as np
import pandas as pd
import re
import matplotlib.pyplot as plt

# Uploading csv file
df = pd.read_csv("bank.csv")
df
```

```
age;"job";"marital";"education";"default";"balance";"housing";"loan";"
contact";"day";"month";"duration";"campaign";"pdays";"previous";"poutc
ome";"y"
0     30;"unemployed";"married";"primary";"no";1787;...

1     33;"services";"married";"secondary";"no";4789;...

2     35;"management";"single";"tertiary";"no";1350;...

3     30;"management";"married";"tertiary";"no";1476...

4     59;"blue-collar";"married";"secondary";"no";0;...

...                                                 ...

4516  33;"services";"married";"secondary";"no";-333;...

4517  57;"self-employed";"married";"tertiary";"yes";...

4518  57;"technician";"married";"secondary";"no";295...

4519  28;"blue-collar";"married";"secondary";"no";11...

4520  44;"entrepreneur";"single";"tertiary";"no";113...


[4521 rows x 1 columns]
```

## Preprocessing of data

```python
#Converting list of column names into column bariables
colmns = re.sub(";", " ", df.columns[0])
colmns = re.sub("\"", "", colmns)
colmns = colmns.split()

# Data cleaning for each row
def clean_data(row):
    row = re.sub(";", " ", row)
    row = re.sub("\"", "", row)
```

```python
    row = row.split()
    return row

data = df.copy()
data.iloc[:,0] = data.iloc[:,0].map(lambda x: clean_data(x))
data
```

```
age;"job";"marital";"education";"default";"balance";"housing";"loan";"
contact";"day";"month";"duration";"campaign";"pdays";"previous";"poutc
ome";"y"
0     [30, unemployed, married, primary, no, 1787, n...

1     [33, services, married, secondary, no, 4789, y...

2     [35, management, single, tertiary, no, 1350, y...

3     [30, management, married, tertiary, no, 1476, ...

4     [59, blue-collar, married, secondary, no, 0, y...

...                                                 ...

4516  [33, services, married, secondary, no, -333, y...

4517  [57, self-employed, married, tertiary, yes, -3...

4518  [57, technician, married, secondary, no, 295, ...

4519  [28, blue-collar, married, secondary, no, 1137...

4520  [44, entrepreneur, single, tertiary, no, 1136,...


[4521 rows x 1 columns]
```

```python
# Filling the values from row list and creating the column from the
column list which we created earlier
idx = 0
for row in data.iloc[:,0]:
    if len(row) == 17:
        i = 0
        for col in colmns:
            data.loc[idx,col] = row[i]
            i += 1
    idx += 1

data.drop(data.columns[0], axis=1, inplace=True)
```

```
df1 = data.copy()
df1
```

```
     age            job  marital  education default balance housing
loan  \
0     30     unemployed  married    primary      no    1787      no
no
1     33       services  married  secondary      no    4789     yes
yes
2     35     management   single   tertiary      no    1350     yes
no
3     30     management  married   tertiary      no    1476     yes
yes
4     59    blue-collar  married  secondary      no       0     yes
no
...   ..            ...      ...        ...     ...     ...     ...  .
..
4516  33       services  married  secondary      no    -333     yes
no
4517  57  self-employed  married   tertiary     yes   -3313     yes
yes
4518  57     technician  married  secondary      no     295      no
no
4519  28    blue-collar  married  secondary      no    1137      no
no
4520  44   entrepreneur   single   tertiary      no    1136     yes
yes

       contact day month duration campaign pdays previous poutcome   y

0     cellular  19   oct       79        1    -1        0  unknown  no

1     cellular  11   may      220        1   339        4  failure  no

2     cellular  16   apr      185        1   330        1  failure  no

3      unknown   3   jun      199        4    -1        0  unknown  no

4      unknown   5   may      226        1    -1        0  unknown  no

...        ...  ..   ...      ...      ...   ...      ...      ...  ..

4516  cellular  30   jul      329        5    -1        0  unknown  no

4517   unknown   9   may      153        1    -1        0  unknown  no

4518  cellular  19   aug      151       11    -1        0  unknown  no

4519  cellular   6   feb      129        4   211        3    other  no
```

```
4520  cellular   3   apr      345        2   249        7    other  no

[4521 rows x 17 columns]
```

### Converting categorical variables into numerical variables

```
convert_dtype = {"age":int, "balance":int, "day":int, "duration":int,
"campaign":int, "pdays":int, "previous":int}
df1 = df1.astype(convert_dtype)
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4521 entries, 0 to 4520
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        4521 non-null   int32
 1   job        4521 non-null   object
 2   marital    4521 non-null   object
 3   education  4521 non-null   object
 4   default    4521 non-null   object
 5   balance    4521 non-null   int32
 6   housing    4521 non-null   object
 7   loan       4521 non-null   object
 8   contact    4521 non-null   object
 9   day        4521 non-null   int32
 10  month      4521 non-null   object
 11  duration   4521 non-null   int32
 12  campaign   4521 non-null   int32
 13  pdays      4521 non-null   int32
 14  previous   4521 non-null   int32
 15  poutcome   4521 non-null   object
 16  y          4521 non-null   object
dtypes: int32(7), object(10)
memory usage: 476.9+ KB
```

```
df1.head()
```

```
   age          job  marital  education default  balance housing loan
\
0   30   unemployed  married    primary      no     1787      no   no

1   33     services  married  secondary      no     4789     yes  yes

2   35   management   single   tertiary      no     1350     yes   no

3   30   management  married   tertiary      no     1476     yes  yes
```

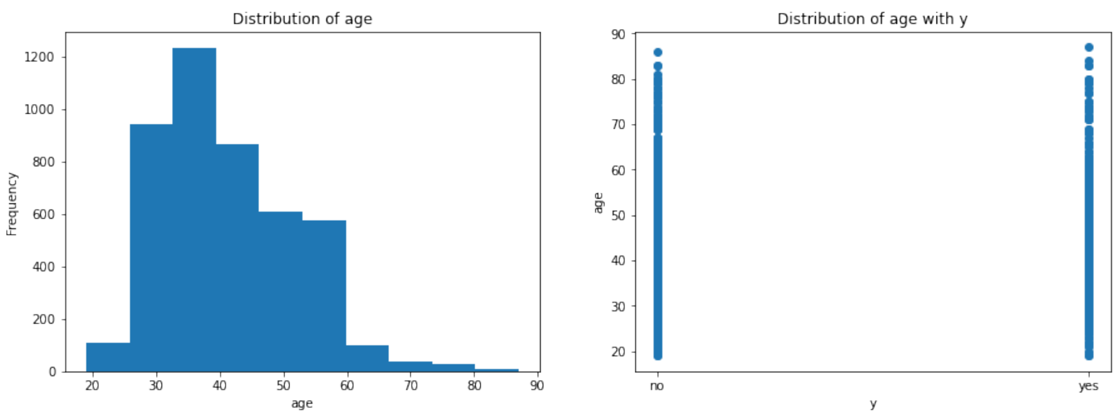
```
4   59  blue-collar  married  secondary      no        0     yes   no


    contact  day month  duration  campaign  pdays  previous poutcome
y
0  cellular   19   oct        79         1     -1         0  unknown
no
1  cellular   11   may       220         1    339         4  failure
no
2  cellular   16   apr       185         1    330         1  failure
no
3   unknown    3   jun       199         4     -1         0  unknown
no
4   unknown    5   may       226         1     -1         0  unknown
no
```

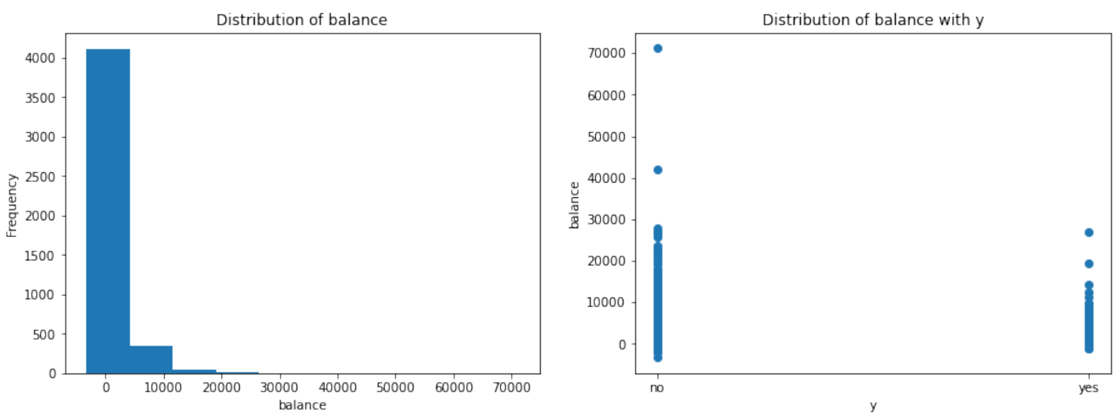
```
#Disintegrating list of categorical column and numerical column
categorical_cols = [col for col in df1.columns if df1[col].dtype ==
"O"]
numerical_cols = [col for col in df1.columns if df1[col].dtype != "O"]

# EDA of numerical columns
df1.loc[:,numerical_cols].describe()
```

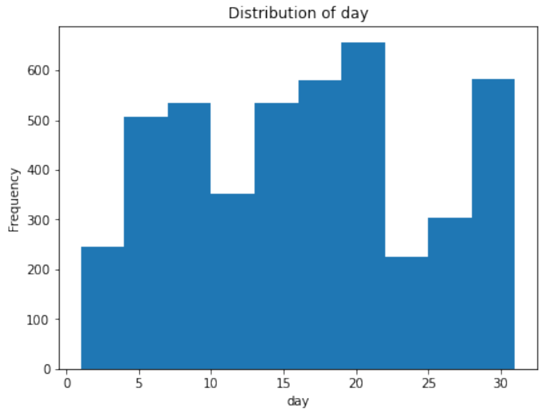
```
               age       balance          day     duration
campaign  \
count  4521.000000   4521.000000  4521.000000  4521.000000
4521.000000
mean     41.170095   1422.657819    15.915284   263.961292
2.793630
std      10.576211   3009.638142     8.247667   259.856633
3.109807
min      19.000000  -3313.000000     1.000000     4.000000
1.000000
25%      33.000000     69.000000     9.000000   104.000000
1.000000
50%      39.000000    444.000000    16.000000   185.000000
2.000000
75%      49.000000   1480.000000    21.000000   329.000000
3.000000
max      87.000000  71188.000000    31.000000  3025.000000
50.000000

             pdays     previous
count  4521.000000  4521.000000
mean     39.766645     0.542579
std     100.121124     1.693562
min      -1.000000     0.000000
25%      -1.000000     0.000000
50%      -1.000000     0.000000
```

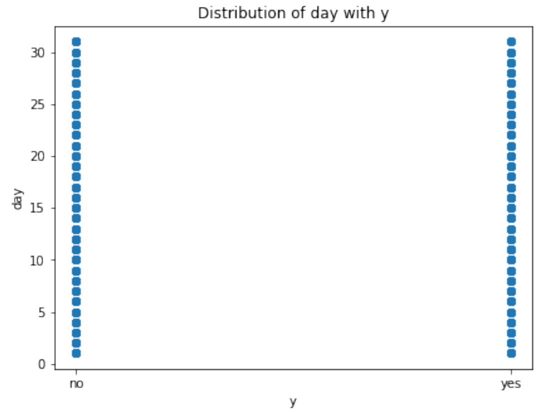
```
75%      -1.000000     0.000000
max     871.000000    25.000000
```

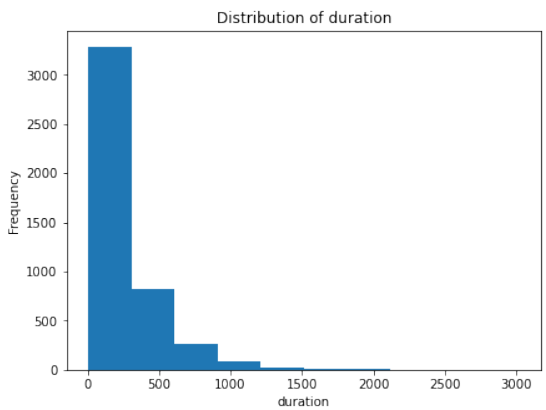
```python
# Value distribution in numeric column
for col in numerical_cols:
    plt.figure(figsize=(15,5))
    right = plt.subplot(1,2,1)
    plt.hist(df1[col])
    right.set_ylabel("Frequency")
    right.set_xlabel(col)
    right.set_title(f"Distribution of {col}")
    left = plt.subplot(1,2,2)
    plt.scatter(df1["y"],df1[col])
    left.set_ylabel(col)
    left.set_xlabel("y")
    left.set_title(f"Distribution of {col} with y")
    plt.show()
```

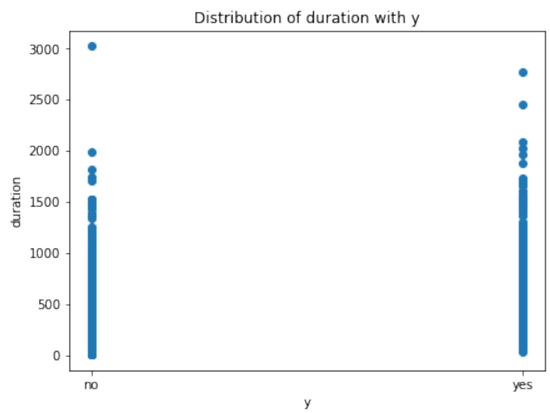


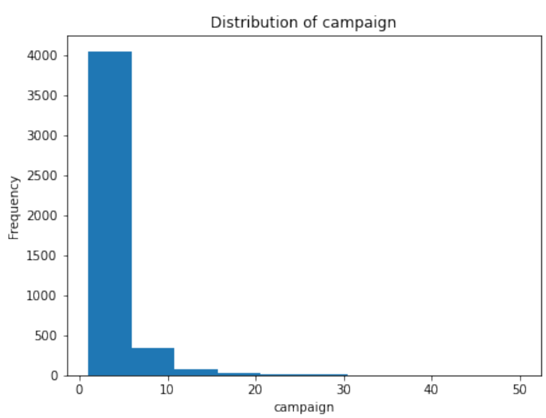



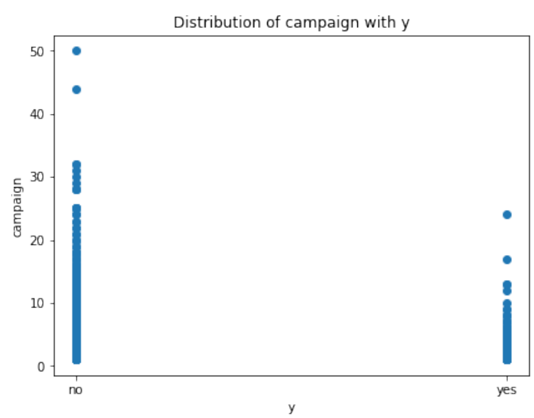

Distribution of day
Frequency
day
0
100
200
300
400
500
600
5
10
15
20
25
30

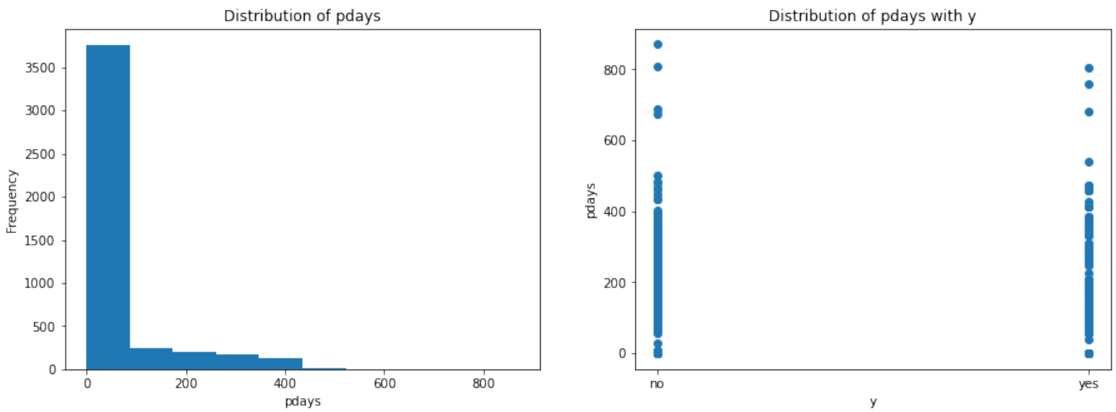

Distribution of day with y
day
y
no
yes
0
5
10
15
20
25
30

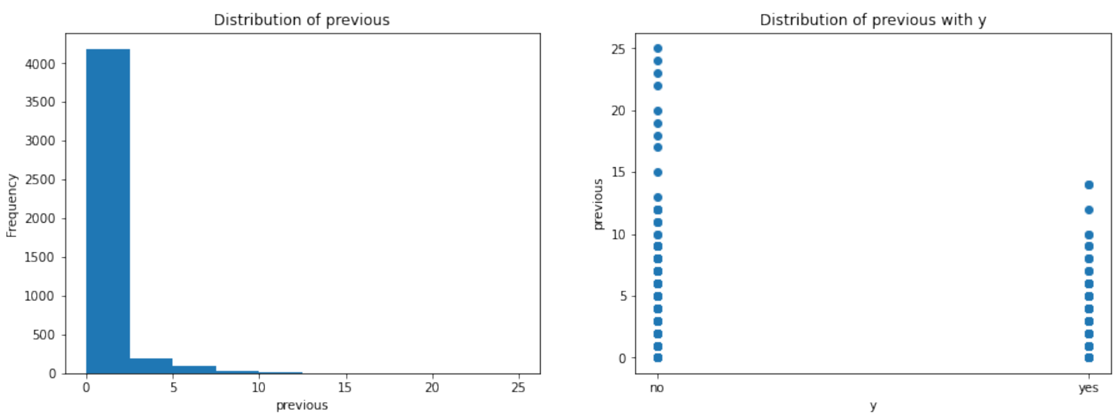

Distribution of duration
Frequency
duration
0
500
1000
1500
2000
2500
3000


Distribution of duration with y
duration
y
no
yes
0
500
1000
1500
2000
2500
3000

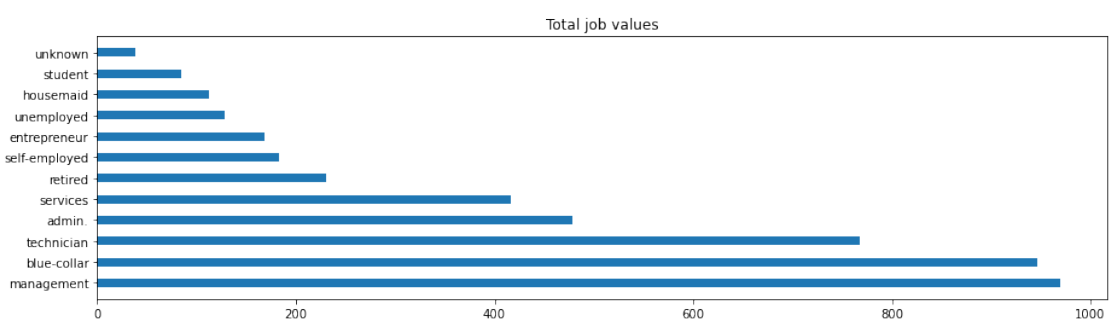

Distribution of campaign
Frequency
campaign
0
500
1000
1500
2000
2500
3000
3500
4000
10
20
30
40
50

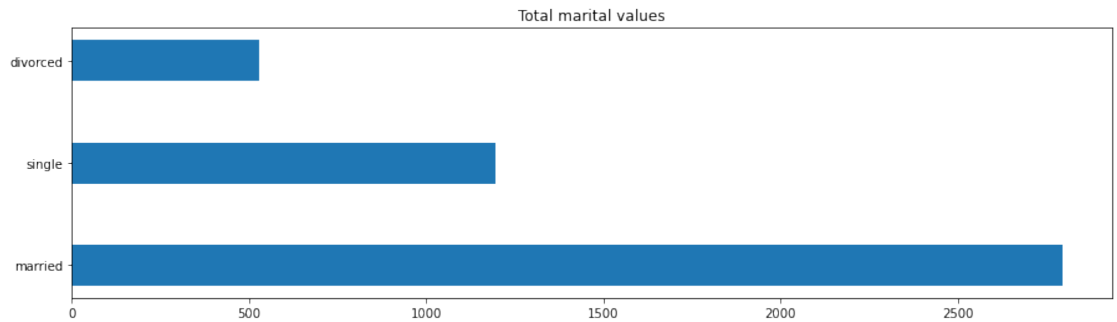

Distribution of campaign with y
campaign
y
no
yes
0
10
20
30
40
50

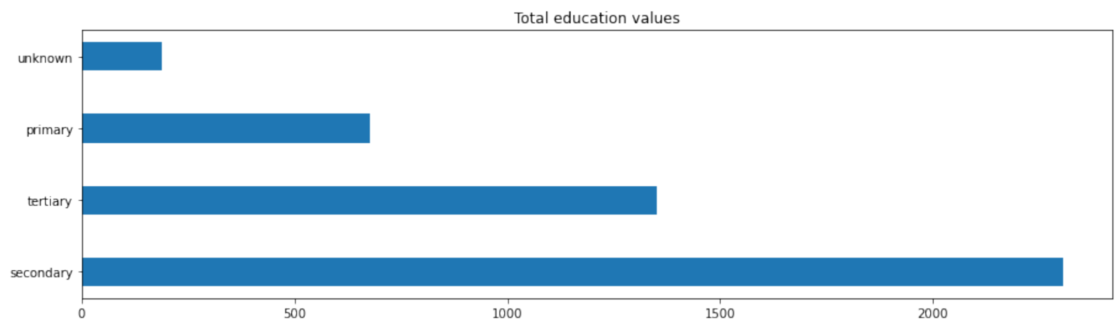


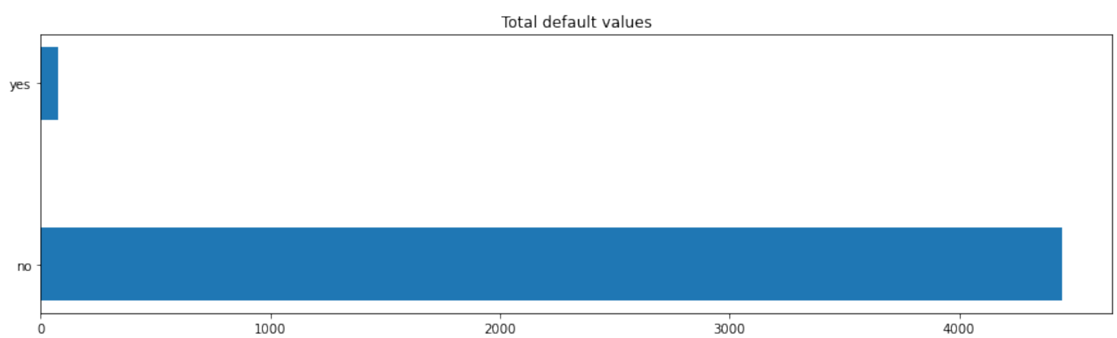



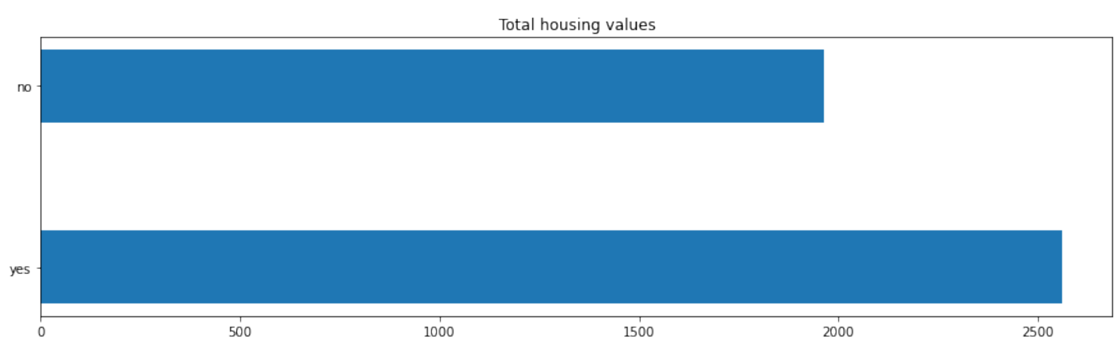

```python
#Categorical column-Unique value count
for col in categorical_cols:
    uni_values = df1[col].value_counts()
    plt.figure(figsize=(15, 4))
    plt.barh(uni_values.index , uni_values, height=0.4)
    plt.title(f"Total {col} values")
    plt.show()
```

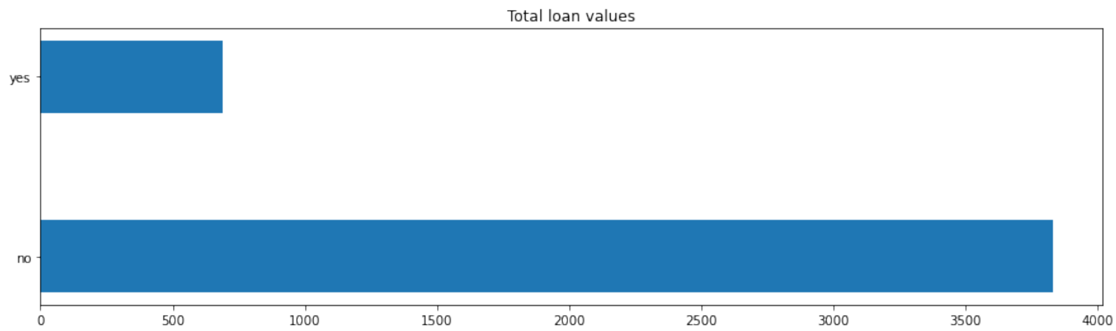


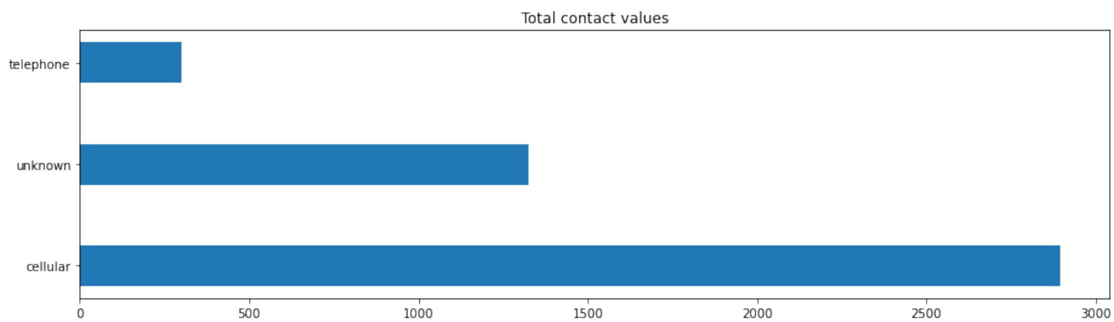

Total marital values
divorced
single
married
0
500
1000
1500
2000
2500

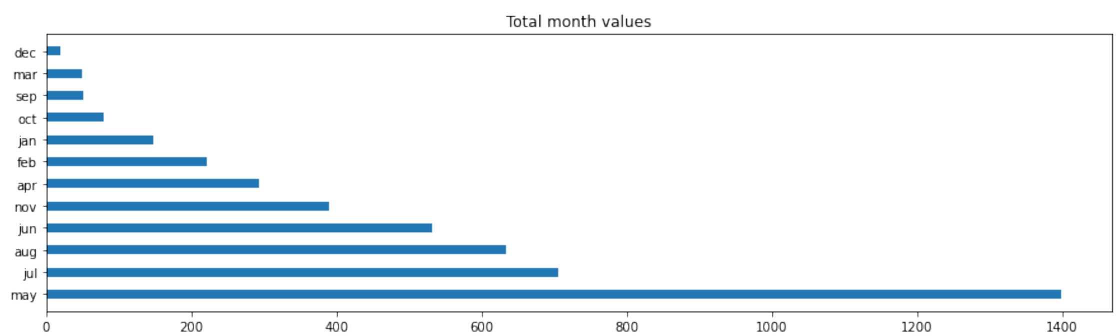

Total education values
unknown
primary
tertiary
secondary
0
500
1000
1500
2000

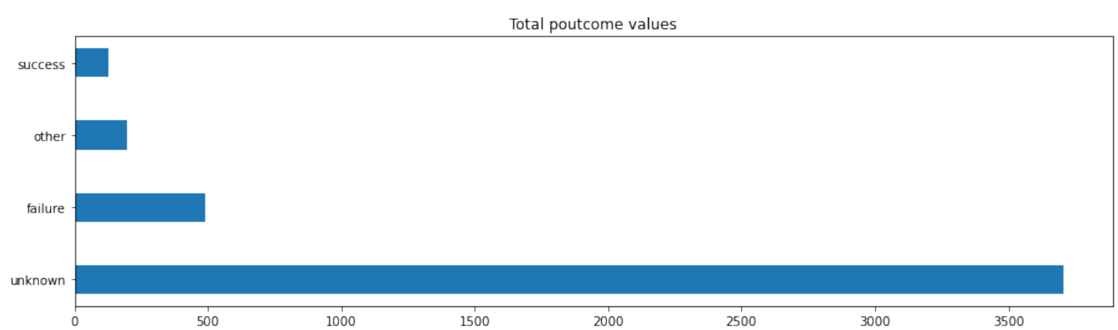

Total default values
yes
no
0
1000
2000
3000
4000

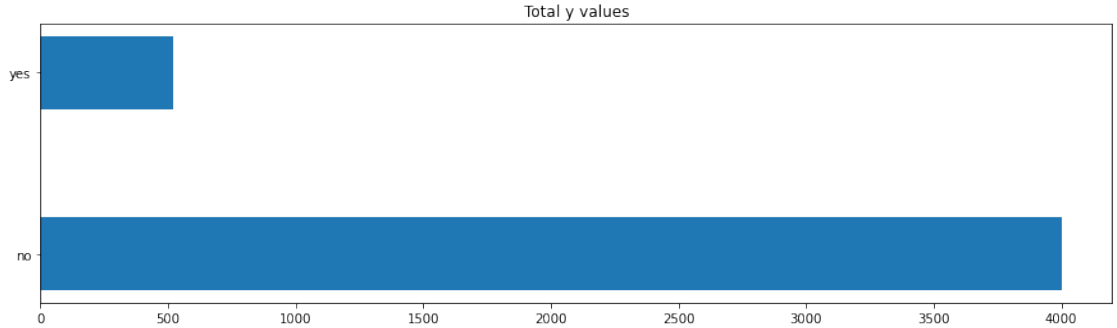

Total housing values
no
yes
0
500
1000
1500
2000
2500


Total loan values
yes
no
0
500
1000
1500
2000
2500
3000
3500
4000


Total contact values
telephone
unknown
cellular
0
500
1000
1500
2000
2500
3000


Total month values
dec
mar
sep
oct
jan
feb
apr
nov
jun
aug
jul
may
0
200
400
600
800
1000
1200
1400


Total poutcome values
success
other
failure
unknown
0
500
1000
1500
2000
2500
3000
3500




```python
# Convert y (target value) to numeric - as one hot encoding can't be used
df1["y"] = df1["y"].map(lambda x: 1 if x=="yes" else 0)

# Day and Month columns combined to show day of the year as both represent last contact
# Previous month days combined together with current month as we are combining days with month
months = {"jan":0, "feb":31, "mar":59, "apr":90, "may":120, "jun":151, "jul":181, "aug":212, "sep":243, "oct":273, "nov":304, "dec":334}
df1["month"] = df1["month"].map(lambda x: months[x])
df1["day_of_year"] = df1["day"] + df1["month"]
df1.drop(["month","day"], axis=1, inplace=True)

# Converting the duration column from second to minutes
df1["duration"] = df1["duration"] / 60

# Change the order of columns
df1 = df1.iloc[:,[0,1,2,3,4,5,6,7,8,9,10,11,12,13,15,14]]
df1.head()
```

```
   age          job  marital  education default  balance housing loan  \
0   30   unemployed  married    primary      no     1787      no   no

1   33     services  married  secondary      no     4789     yes  yes

2   35   management   single   tertiary      no     1350     yes   no

3   30   management  married   tertiary      no     1476     yes  yes

4   59  blue-collar  married  secondary      no        0     yes   no


    contact  duration  campaign  pdays  previous poutcome  day_of_year  y
0  cellular  1.316667         1     -1         0  unknown          292  0
1  cellular  3.666667         1    339         4  failure          131
```

```
0
2  cellular  3.083333         1    330         1  failure         106
0
3   unknown  3.316667         4     -1         0  unknown         154
0
4   unknown  3.766667         1     -1         0  unknown         125
0
```

```
#!pip install category_encoders
```

```
Requirement already satisfied: category_encoders in c:\users\hp\
anaconda3\lib\site-packages (2.6.0)
Requirement already satisfied: pandas>=1.0.5 in c:\users\hp\anaconda3\
lib\site-packages (from category_encoders) (1.3.4)
Requirement already satisfied: patsy>=0.5.1 in c:\users\hp\anaconda3\
lib\site-packages (from category_encoders) (0.5.2)
Requirement already satisfied: numpy>=1.14.0 in c:\users\hp\anaconda3\
lib\site-packages (from category_encoders) (1.20.3)
Requirement already satisfied: scipy>=1.0.0 in c:\users\hp\anaconda3\
lib\site-packages (from category_encoders) (1.7.1)
Requirement already satisfied: statsmodels>=0.9.0 in c:\users\hp\
anaconda3\lib\site-packages (from category_encoders) (0.12.2)
Requirement already satisfied: scikit-learn>=0.20.0 in c:\users\hp\
anaconda3\lib\site-packages (from category_encoders) (0.24.2)
Requirement already satisfied: pytz>=2017.3 in c:\users\hp\anaconda3\
lib\site-packages (from pandas>=1.0.5->category_encoders) (2021.3)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\hp\
anaconda3\lib\site-packages (from pandas>=1.0.5->category_encoders)
(2.8.2)
Requirement already satisfied: six in c:\users\hp\anaconda3\lib\site-
packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=0.11 in c:\users\hp\anaconda3\
lib\site-packages (from scikit-learn>=0.20.0->category_encoders)
(1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\hp\
anaconda3\lib\site-packages (from scikit-learn>=0.20.0-
>category_encoders) (2.2.0)
```

```
# We will use count encoder to transform the categorical variable into
frequency of subcategory
from category_encoders.count import CountEncoder

categorical_cols.remove("month")
categorical_cols.remove("y")
col_to_transform = [col for col in categorical_cols if col !=
"day_of_year" and col != "y"]

CEnc = CountEncoder(cols=col_to_transform, normalize=True)
trans_data = CEnc.fit_transform(X=df1)
```

trans_data

```
      age       job   marital  education  default  balance  
housing  \
0      30  0.028312  0.618668   0.149967  0.98319     1787  0.433975   

1      33  0.092236  0.618668   0.510064  0.98319     4789  0.566025   

2      35  0.214333  0.264543   0.298607  0.98319     1350  0.566025   

3      30  0.214333  0.618668   0.298607  0.98319     1476  0.566025   

4      59  0.209246  0.618668   0.510064  0.98319        0  0.566025   

...   ...       ...       ...        ...      ...      ...       ...   

4516   33  0.092236  0.618668   0.510064  0.98319     -333  0.566025   

4517   57  0.040478  0.618668   0.298607  0.01681    -3313  0.566025   

4518   57  0.169874  0.618668   0.510064  0.98319      295  0.433975   

4519   28  0.209246  0.618668   0.510064  0.98319     1137  0.433975   

4520   44  0.037160  0.264543   0.298607  0.98319     1136  0.566025   


          loan   contact  duration  campaign  pdays  previous  
poutcome  \
0     0.847158  0.640566  1.316667         1     -1         0  
0.819509   
1     0.152842  0.640566  3.666667         1    339         4  
0.108383   
2     0.847158  0.640566  3.083333         1    330         1  
0.108383   
3     0.152842  0.292856  3.316667         4     -1         0  
0.819509   
4     0.847158  0.292856  3.766667         1     -1         0  
0.819509   
...        ...       ...       ...       ...    ...       ...       ..
.   
4516  0.847158  0.640566  5.483333         5     -1         0  
0.819509   
4517  0.152842  0.292856  2.550000         1     -1         0  
0.819509   
4518  0.847158  0.640566  2.516667        11     -1         0  
0.819509   
4519  0.847158  0.640566  2.150000         4    211         3  
```

```
0.043574
4520  0.152842  0.640566  5.750000         2    249          7
0.043574

      day_of_year  y
0             292  0
1             131  0
2             106  0
3             154  0
4             125  0
...           ... ..
4516          211  0
4517          129  0
4518          231  0
4519           37  0
4520           93  0

[4521 rows x 16 columns]
```

## Train test split

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test =
train_test_split(trans_data.iloc[:,:-1], trans_data.iloc[:,-1],
                                                     test_size=0.2,
stratify=trans_data.iloc[:,-1])
```

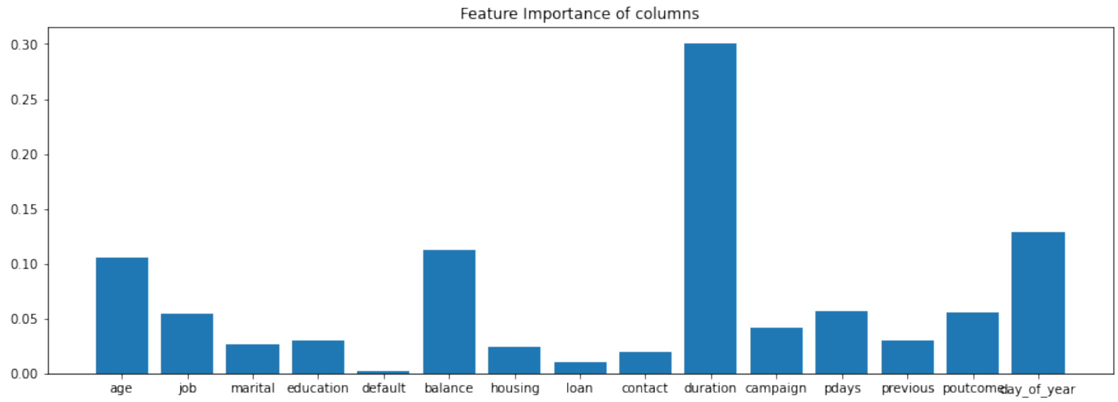
## Fine tuning the model by different parameters or technique to improve performance

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

RF = RandomForestClassifier()
RF.fit(X_train, Y_train)
res = RF.predict(X_test)
print(classification_report(Y_test, res))

              precision    recall  f1-score   support

           0       0.91      0.96      0.94       801
           1       0.49      0.29      0.36       104

    accuracy                           0.88       905
   macro avg       0.70      0.62      0.65       905
weighted avg       0.86      0.88      0.87       905
```

## Feature Engineering

```
plt.figure(figsize=(15,5))
plt.bar(X_train.columns, RF.feature_importances_)
plt.title("Feature Importance of columns")
```

```
Text(0.5, 1.0, 'Feature Importance of columns')
```



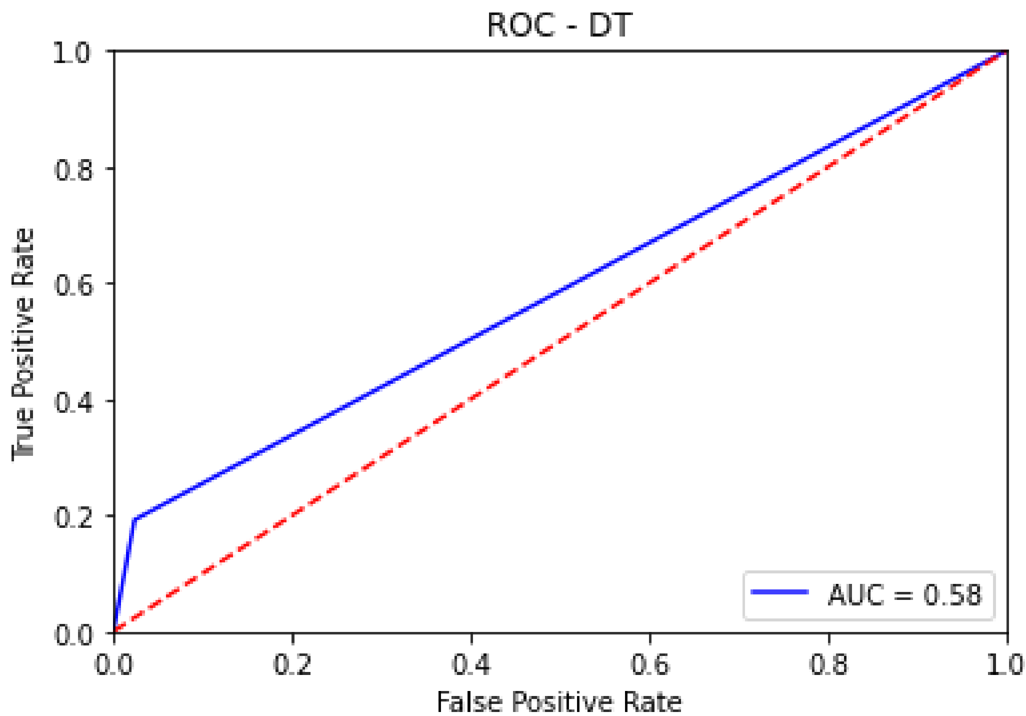

```
X_train.drop(["default"], axis=1, inplace=True)
X_train.head()
```

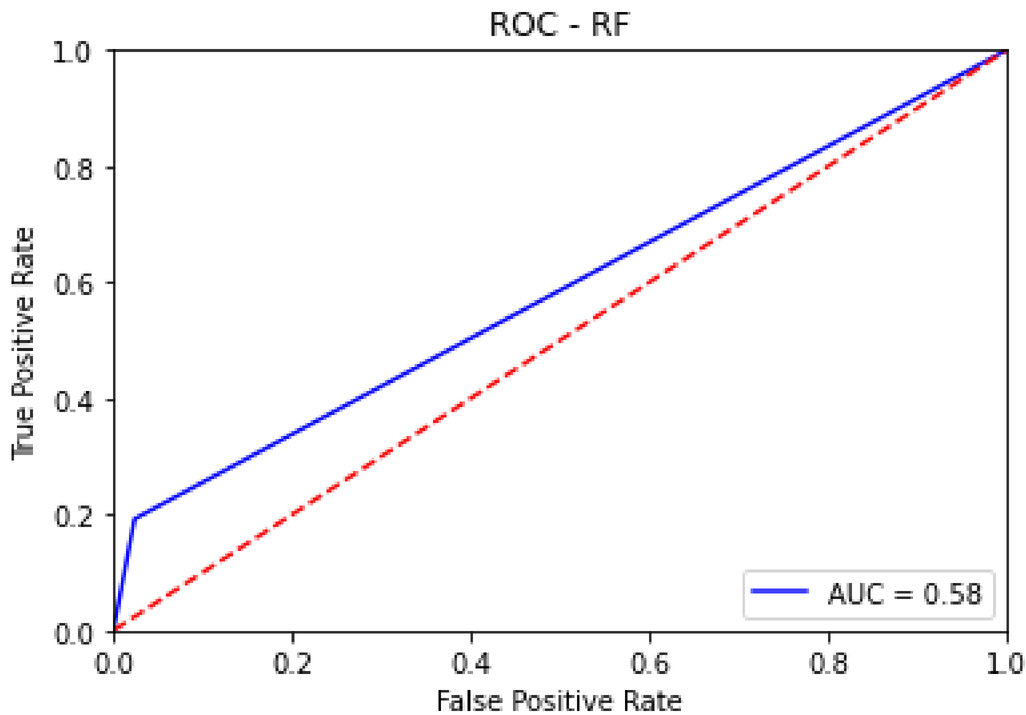
```
      age       job   marital  education  balance   housing      loan  \
786    51  0.214333  0.116788   0.298607        0  0.433975  0.847158   

1753   56  0.092236  0.618668   0.510064       83  0.433975  0.847158   

1695   43  0.105729  0.618668   0.510064      132  0.433975  0.847158   

2367   40  0.214333  0.264543   0.041363      838  0.566025  0.847158   

200    34  0.169874  0.264543   0.298607      992  0.566025  0.847158   


       contact   duration  campaign  pdays  previous  poutcome  day_of_year  
786   0.640566   1.000000         2     -1         0  0.819509          225  
1753  0.640566   0.433333        11     -1         0  0.819509          239  
1695  0.640566   9.566667         1     84         3  0.028534          231  
2367  0.292856  10.316667         3     -1         0  0.819509          132  
200   0.640566   5.016667         1     88         2  0.028534          124  
```

## Random Forest

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

RFC = RandomForestClassifier()
parameters = {"n_estimators":[70,80,90,100,110,120],
              "max_depth":[4,5,6,7]}

GB_RF = GridSearchCV(estimator=RFC, param_grid=parameters)
GB_RF.fit(X_train, Y_train)

print(f"Best Score for Random forest is {GB_RF.best_score_}")

Best Score for Random forest is 0.9026581233809404

# Model Evaluation
X_test.drop(["default"], axis=1, inplace=True)
ypred_RF = GB_RF.predict(X_test)
print(f"Classification Report of Random forest \
n{classification_report(Y_test, ypred_RF)}")

Classification Report of Random forest
              precision    recall  f1-score   support

           0       0.90      0.98      0.94       801
           1       0.51      0.19      0.28       104

    accuracy                           0.89       905
   macro avg       0.71      0.58      0.61       905
weighted avg       0.86      0.89      0.86       905


import sklearn.metrics as metrics
from sklearn.metrics import classification_report, confusion_matrix,
ConfusionMatrixDisplay

# AUC ROC - DT
# calculate the fpr and tpr for all thresholds of the classification
fpr, tpr, threshold = metrics.roc_curve(Y_test, ypred_RF)
roc_auc = metrics.auc(fpr, tpr)
plt.title('ROC - DT')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```python
# AUC ROC - RF
# calculate the fpr and tpr for all thresholds of the classification
fpr, tpr, threshold = metrics.roc_curve(Y_test, ypred_RF)
roc_auc = metrics.auc(fpr, tpr)
plt.title('ROC - RF')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



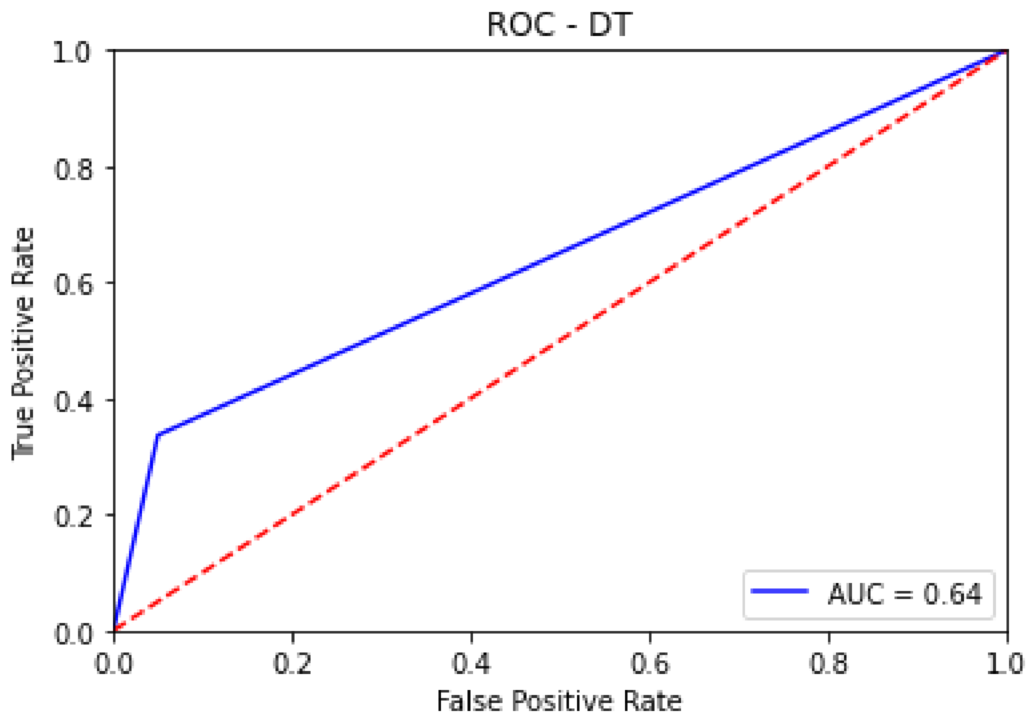


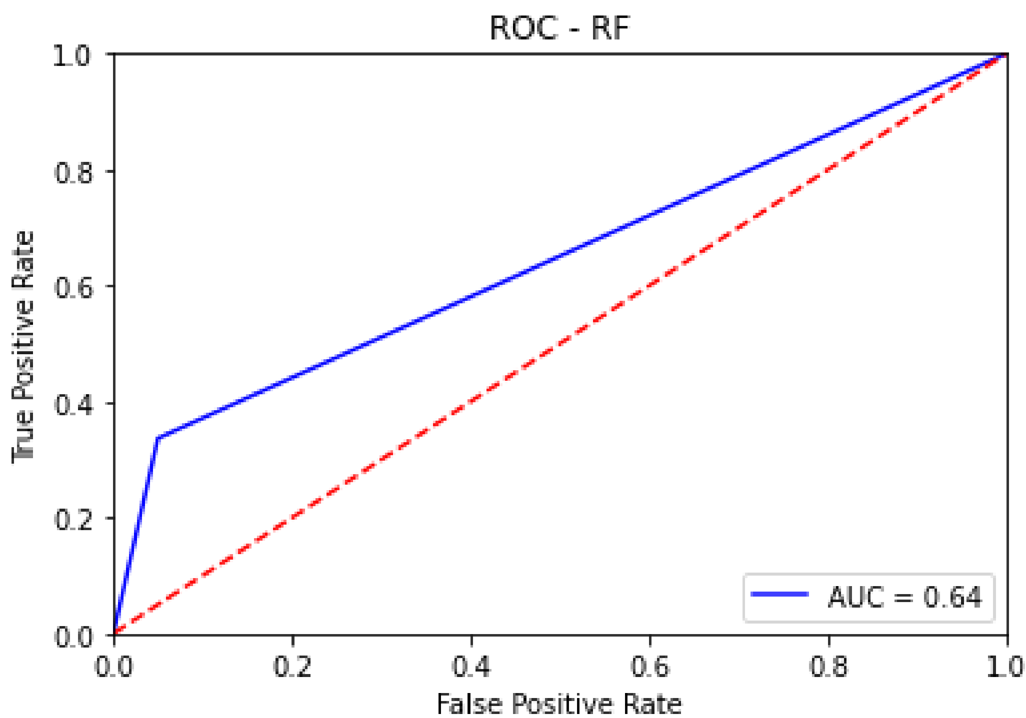

## Gradient Boosting

```
from sklearn.ensemble import AdaBoostClassifier,
GradientBoostingClassifier

parameters = {"n_estimators":[70,80,90,100,110,120],
              "learning_rate":[0.05,0.07,0.1,0.13,0.15,0.2],
              "max_depth":[4,5,6,7]}

GBC = GradientBoostingClassifier()
Grid_GB = GridSearchCV(estimator=GBC, param_grid=parameters, cv=4)
Grid_GB.fit(X_train, Y_train)

print(f"Best Score for Random forest is {Grid_GB.best_score_}")

Best Score for Random forest is 0.9065265486725664

ypred_GB = Grid_GB.predict(X_test)
print(f"Classification Report of Gradient Boosting Classifier \n\
n{classification_report(Y_test, ypred_GB)}")

Classification Report of Gradient Boosting Classifier

              precision    recall  f1-score   support

           0       0.92      0.95      0.93       801
```

```
           1       0.47      0.34      0.39       104

    accuracy                           0.88       905
   macro avg       0.69      0.64      0.66       905
weighted avg       0.87      0.88      0.87       905
```

```
# AUC ROC - DT
# calculate the fpr and tpr for all thresholds of the classification
fpr, tpr, threshold = metrics.roc_curve(Y_test,ypred_GB)
roc_auc = metrics.auc(fpr, tpr)
plt.title('ROC - DT')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
# AUC ROC - RF
# calculate the fpr and tpr for all thresholds of the classification
fpr, tpr, threshold = metrics.roc_curve(Y_test, ypred_GB)
roc_auc = metrics.auc(fpr, tpr)
plt.title('ROC - RF')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



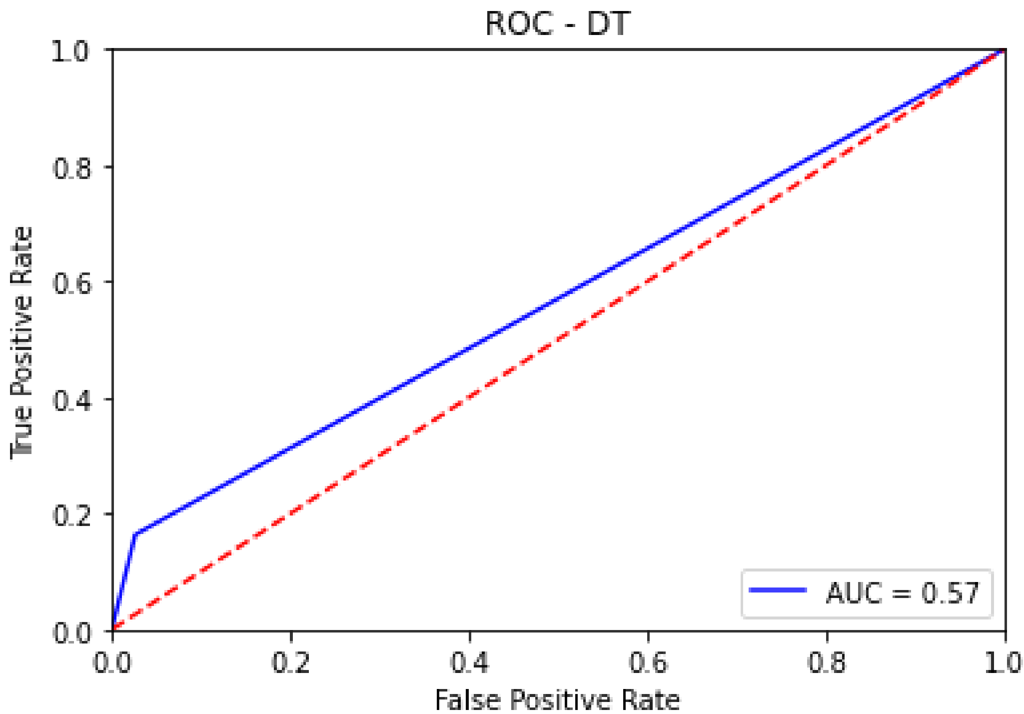



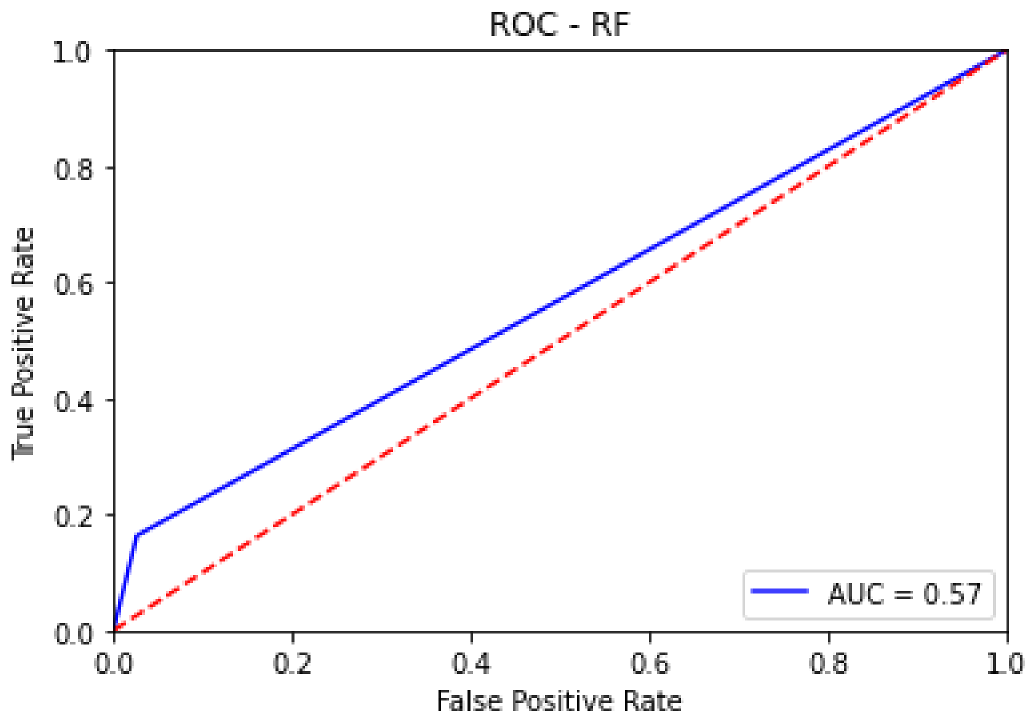

## Logistic Regression

```
from sklearn.linear_model import LogisticRegression

LR = LogisticRegression()
```

```
LR.fit(X_train, Y_train)
ypred_LR = LR.predict(X_test)
print(f"Classification Report of Logistic Regression Classifier \n\
n{classification_report(Y_test, ypred_LR)}")

Classification Report of Logistic Regression Classifier

              precision    recall  f1-score   support

           0       0.90      0.97      0.94       801
           1       0.45      0.16      0.24       104

    accuracy                           0.88       905
   macro avg       0.67      0.57      0.59       905
weighted avg       0.85      0.88      0.86       905


C:\Users\hp\anaconda3\lib\site-packages\sklearn\linear_model\
_logistic.py:763: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

# AUC ROC - DT
# calculate the fpr and tpr for all thresholds of the classification
fpr, tpr, threshold = metrics.roc_curve(Y_test, ypred_LR)
roc_auc = metrics.auc(fpr, tpr)
plt.title('ROC - DT')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
# AUC ROC - RF
# calculate the fpr and tpr for all thresholds of the classification
fpr, tpr, threshold = metrics.roc_curve(Y_test, ypred_LR)
roc_auc = metrics.auc(fpr, tpr)
plt.title('ROC - RF')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
```

```
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```






Limitation -

1.Logistic regression is cannot be used with continous value 2.Gradient boost is prone to over fitting 3.Large number of trees can make the algorithm too slow.

Future work-

1. Logistic regression is best used for prediction and classification problem
2. Gradient boost is used in classification and regression task
3. Random forest is used to solve regression and classification problem.