CUSTOMER CHURN PREDICTION

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from imblearn.combine import SMOTETomek
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import
RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import
GradientBoostingClassifier
from sklearn.metrics import accuracy_score,
precision_score, recall_score, f1_score, roc_auc_score,
```

roc_curve,
ConfusionMatrixDisplay,classification_report

import warnings

warnings.filterwarnings("ignore")

df = pd.read_csv("bank.csv")

df

age;"job";"marital";"education";"default";"balance";"housing";"loan";"contact";"day";"month";"duration";"campaign";"pdays";"previous";"poutcome";"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...

df = pd.read_csv("bank.csv", sep = ";")

df.head()

age job marital education default balance housing loan contact day month duration campaign pdays previous poutcome y

0 30 unemployed married primary no 1787 no no cellular 19 oct 79 1 -1 0 unknown no

1 33 services married secondary no 4789 yes yes cellular 11 may 220 1 339 4 failure no

2 35 management single tertiary no 1350 yes no cellular 16 apr 185 1 330 1 failure no

3 30 management married tertiary no 1476 yes yes unknown 3 jun 199 4 -1 0 unknown no

4 59 blue-collar married secondary no 0 yes no unknown 5 may 226 1 -1 0 unknown no

df.tail()

age job marital education default balance housing loan contact day month duration campaign pdays previous poutcome y

4516 33 services married secondary no -333 yes no cellular 30 jul 329 5 -1 0 unknown no

4517 57 self-employed married tertiary yes -3313 yes yes unknown 9 may 153 1 -1 0 unknown no

4518 57 technician married secondary no 295 no no cellular 19 aug 151 11 -1 0 unknown no

4519 28 blue-collar married secondary no 1137 no no cellular 6 feb 129 4 211 3 other no

4520 44 entrepreneur single

df.columns

Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housing',

'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',

'previous', 'poutcome', 'y'],

dtype='object')

df.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 int64

 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 int64

6 housing 4521 non-null object

7 loan 4521 non-null object

8 contact 4521 non-null object

9 day 4521 non-null int64

10 month 4521 non-null object

11 duration 4521 non-null int64

12 campaign 4521 non-null int64

13 pdays 4521 non-null int64

14 previous 4521 non-null int64

15 poutcome 4521 non-null object

16 y 4521 non-null object

dtypes: int64(7), object(10)

memory usage: 600.6+ KB

df.describe()

age balance day duration campaign pdays previous

count 4521.000000 4521.000000 4521.000000 4521.000000 4521.000000 4521.000000 4521.000000 mean 41.170095 1422.657819 15.915284 263.961292 2.793630 39.766645 0.542579

std 10.576211 3009.638142 8.247667 259.856633 3.109807 100.121124 1.693562

min 19.000000 -3313.000000 1.000000 4.000000 1.000000 -1.000000 0.000000

25% 33.000000 69.000000 9.000000 104.000000 1.000000 -1.000000 0.000000

50% 39.000000 444.000000 16.000000 185.000000 2.000000 -1.000000 0.000000

75% 49.000000 1480.000000 21.000000 329.000000 3.000000 -1.000000 0.000000

max 87.000000 71188.000000 31.000000 3025.000000 50.000000 871.00000025.000000

df.isnull().any().sum() #no null values

0

df.duplicated().any() #no duplicates

False

#checking of for the balance of data

df['y'].value_counts() #Imabalanced of data

no 4000

yes 521

Name: y, dtype: int64

df.job.value_counts()

management 969

blue-collar 946

technician 768

admin. 478

services 417

retired 230

self-employed 183

entrepreneur 168

unemployed 128

housemaid 112

student 84

unknown 38

Name: job, dtype: int64

df.marital.value_counts()

married 2797

single 1196

divorced 528

Name: marital, dtype: int64

df.education.value_counts()

secondary 2306

tertiary 1350

primary 678

unknown 187

Name: education, dtype: int64

df.default.value_counts()

no 4445

yes 76

Name: default, dtype: int64

df.housing.value_counts()

yes 2559

no 1962

Name: housing, dtype: int64

df.loan.value_counts()

no 3830

yes 691

Name: loan, dtype: int64

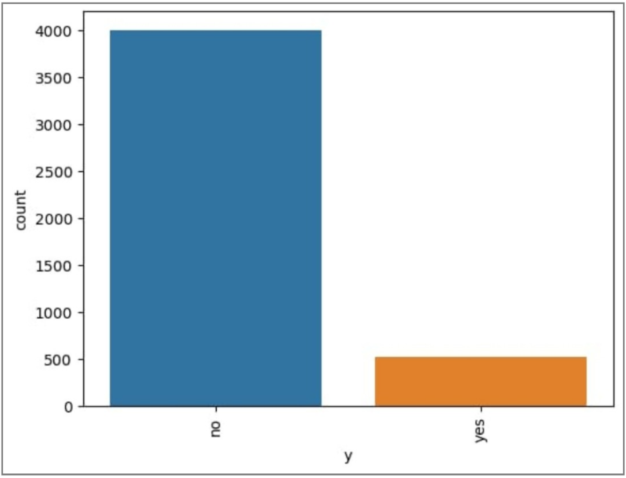
df.contact.value_counts()

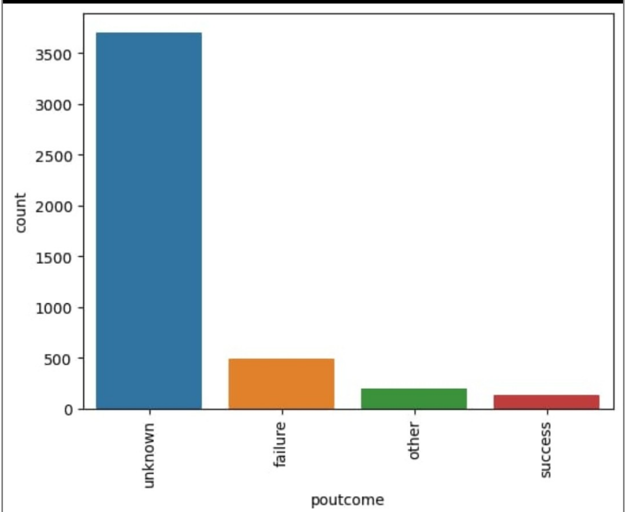
cellular 2896

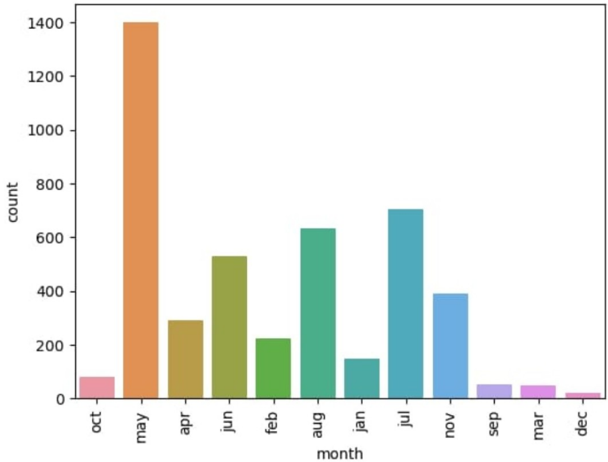
unknown 1324

telephone 301

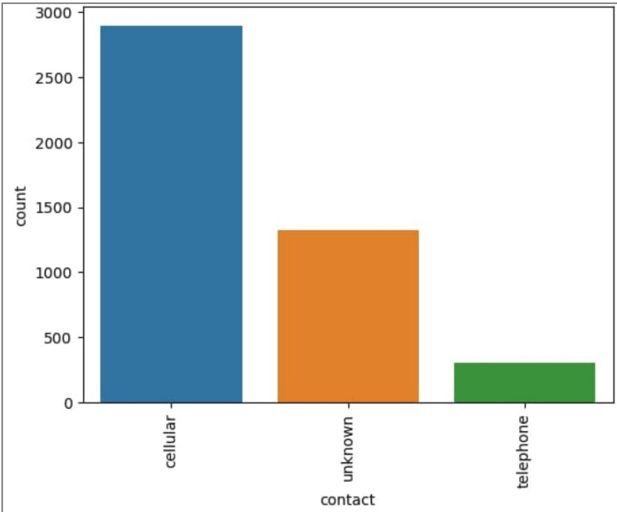
```
Name: contact, dtype: int64

df.poutcome.value_counts()

unknown 3705

failure 490

other 197

success 129

Name: poutcome, dtype: int64

#EDA

cat_val = df.iloc[:,[1,2,3,4,6,7,8,10,15,16]]


for i, col in enumerate(cat_val.columns):

    plt.figure(i)

    sns.countplot(cat_val[col])

    plt.xticks(rotation=90)
```


4000
3500
3000
2500
2000
1500
1000
500
0
count
no
yes
y

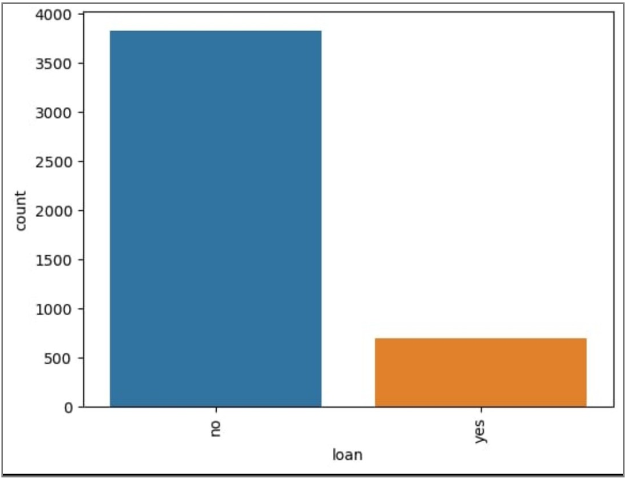

3500
3000
2500
2000
1500
1000
500
0
count
unknown
failure
other
success
poutcome

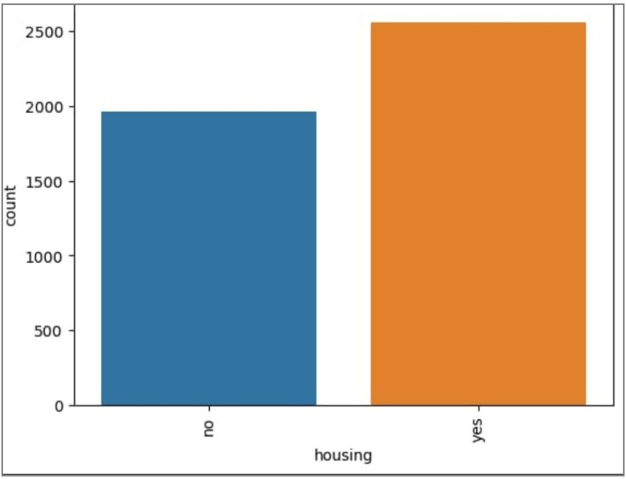


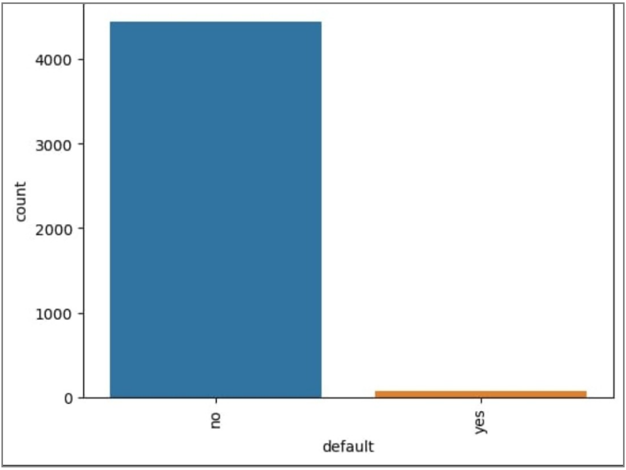
count
1400
1200
1000
800
600
400
200
0
oct
may
apr
jun
feb
aug
jan
jul
nov
sep
mar
dec
month

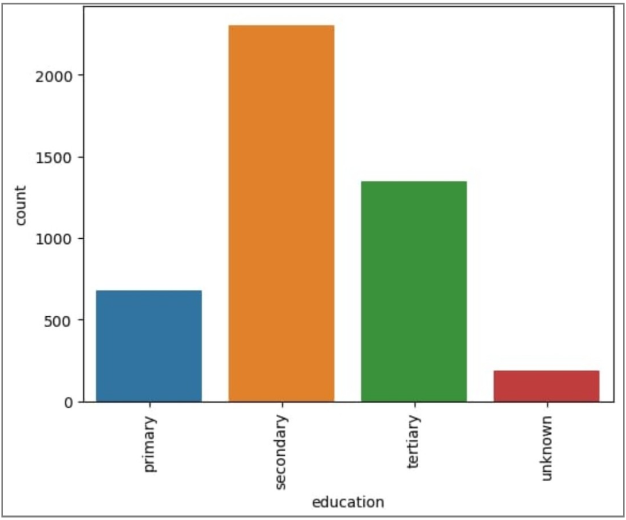


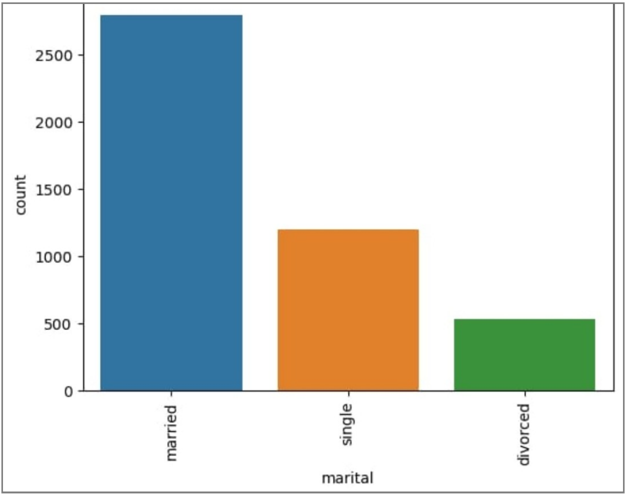
count
3000
2500
2000
1500
1000
500
0
cellular
unknown
telephone
contact

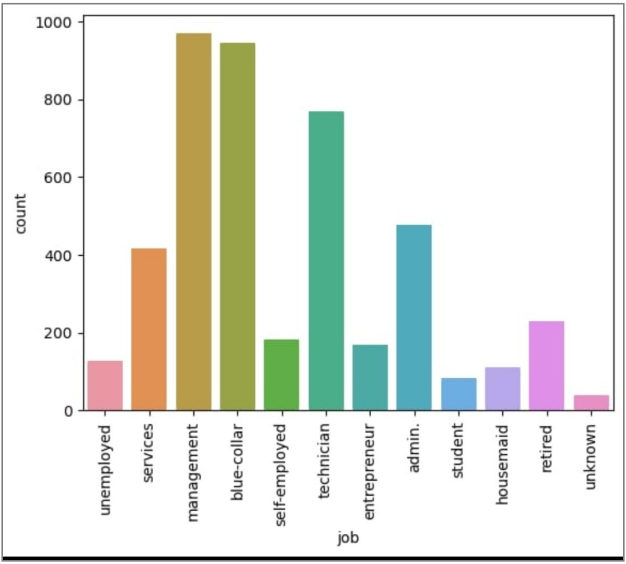

4000
3500
3000
2500
2000
1500
1000
500
0
count
no
yes
loan


2500
2000
1500
1000
500
0
count
no
yes
housing

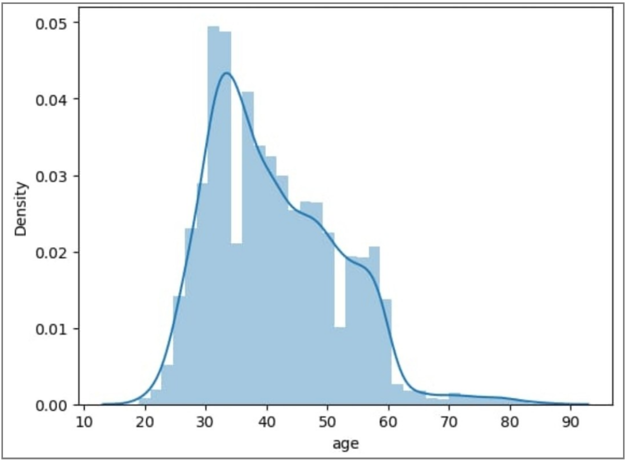

4000
3000
2000
1000
0
count
no
yes
default

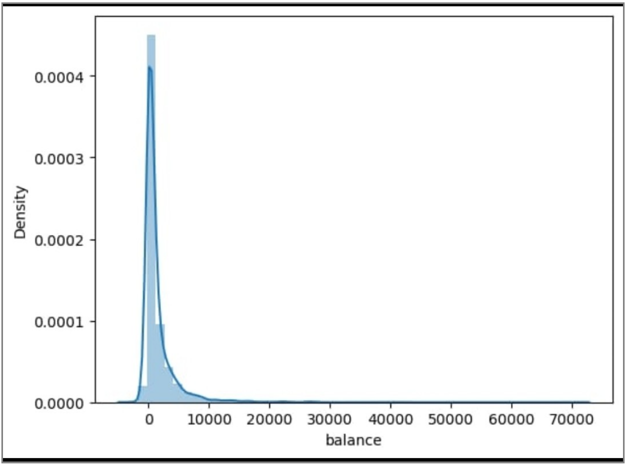

2000
1500
1000
500
0
count
primary
secondary
tertiary
unknown
education

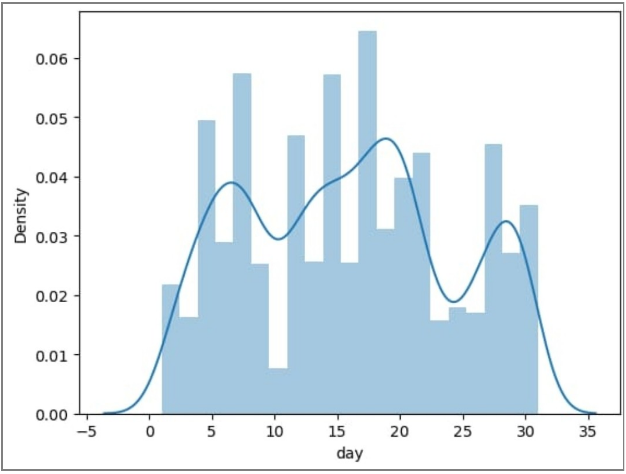

2500
2000
1500
1000
500
0
count
married
single
divorced
marital

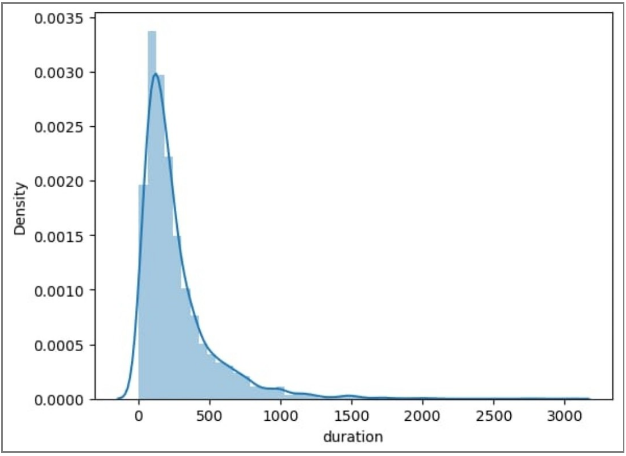

1000
800
600
400
200
0
count
unemployed
services
management
blue-collar
self-employed
technician
entrepreneur
admin.
student
housemaid
retired
unknown
job

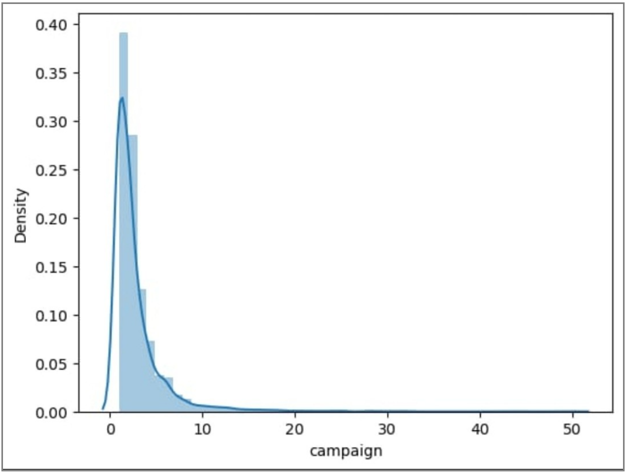
```
df_num = df.iloc[:,[0,5,9,11,12,13,14]]
for i, col in enumerate(df_num.columns):
    plt.figure(i)
    sns.distplot(df_num[col]
```

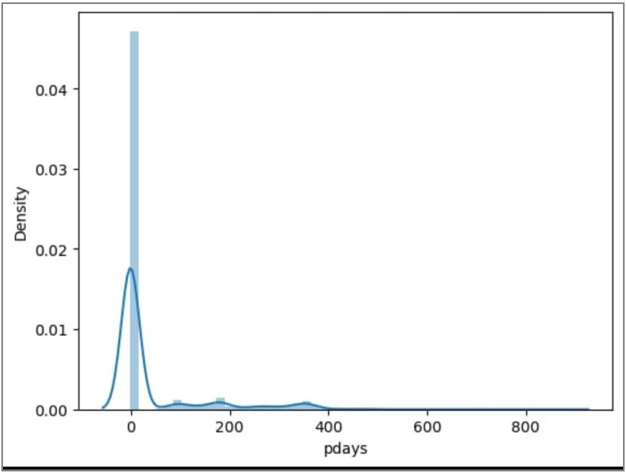

0.05
0.04
0.03
0.02
0.01
0.00
Density
10
20
30
40
50
60
70
80
90
age

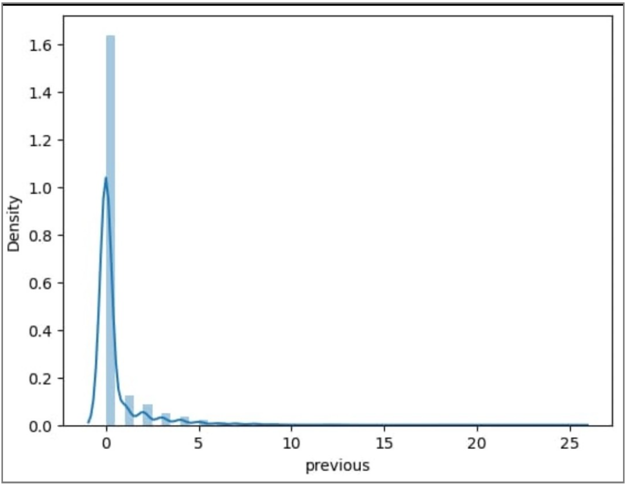

0.0004
0.0003
0.0002
0.0001
0.0000
Density
0
10000
20000
30000
40000
50000
60000
70000
balance


0.06
0.05
0.04
0.03
0.02
0.01
0.00
Density
−5
0
5
10
15
20
25
30
35
day


0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
Density
0
500
1000
1500
2000
2500
3000
duration


0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Density
0
10
20
30
40
50
campaign


0.04
0.03
0.02
0.01
0.00
Density
0
200
400
600
800
pdays



creating label encoder

labelencoder = LabelEncoder()

cat_val = df.iloc[:,[1,2,3,4,6,7,8,10,15,16]]

cat_val['job']=labelencoder.fit_transform(cat_val['job']
)

cat_val['marital']=labelencoder.fit_transform(cat_val['
marital'])

cat_val['education']=labelencoder.fit_transform(cat_v
al['education'])

cat_val['default']=labelencoder.fit_transform(cat_val['
default'])

cat_val['housing']=labelencoder.fit_transform(cat_val['housing'])

cat_val['loan']=labelencoder.fit_transform(cat_val['loan'])

cat_val['contact']=labelencoder.fit_transform(cat_val['contact'])

cat_val['month']=labelencoder.fit_transform(cat_val['month'])

cat_val['poutcome']=labelencoder.fit_transform(cat_val['poutcome'])

cat_val['y']=labelencoder.fit_transform(cat_val['y'])

df1 = pd.concat([df_num, cat_val], axis = 1)

df1

age balance day duration campaign pdays previous job marital education default housing loan contact month poutcome y

0 30 1787 19 79 1 -1 0 10 1 0 0 0 0 0 10 3 0

1 33 4789 11 220 1 339 4 7 1 1 0 1 1 0 8 0 0

2 35 1350 16 185 1 330 1 4 2 2 0 1 0 0 0 0 0

3 30 1476 3 199 4 -1 0 4 1 2 0 1 1 2 6 3 0

4 59 0 5 226 1 -1 0 1 1 1 0 1 0 2 8 3 0

... ...

4516 33 -333 30 329 5 -1 0 7 1 1 0 1 0 0 5 3 0

4517 57 -3313 9 153 1 -1 0 6 1 2 1 1 1 2 8 3 0

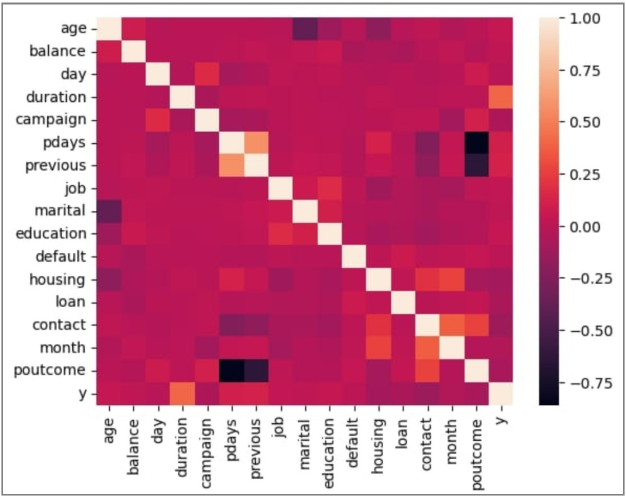
4518 57 295 19 151 11 -1 0 9 1 1 0 0 0 0 1 3 0

4519 28 1137 6 129 4 211 3 1 1 1 0 0 0 0 3 1 0

4520 44 1136 3 345 2 249 7 2 2 2 0 1 1 0 0 1 0

sns.heatmap(df1.corr())

<AxesSubplot:>



Its highly imbalanced data so lets use SMOTE technique to balance the data

X =df1.drop("y", axis = 1)

y = df1["y"]

```
smt = SMOTETomek(sampling_strategy = "not majority", random_state=100)

X1, y1 = smt.fit_resample(X, y)

X1.shape , y1.shape
```

((7748, 16), (7748,))

```
#standarization #mean=0 , std=1

def norm_func(i):
    x = (i-i.mean())/(i.std())
    return(x)

df2 = norm_func(X1)

final = pd.concat([df2,y1], axis = 1)

final
```

age balance day duration campaign pdays previous job marital education default housing loan contact month poutcome y

0 -1.074135 0.124214 0.422748 -0.884989 -0.549595 -0.487783 -0.398512 1.927765 -0.050335 -1.618657 -0.098192 -0.819477 -0.313642 -0.564921 1.640509 0.582795 0

1 -0.792707 1.242287 -0.617160 -0.486186 -0.549595 2.813377 1.973425 0.906292 -0.050335 -0.192206 -0.098192 1.220133 3.187939 -0.564921 0.957910 -2.320437 0

2 -0.605088 -0.038544 0.032782 -0.585180 -0.549595 2.725993 0.194472 -0.115182 1.616327 1.234244 -0.098192 1.220133 -0.313642 -0.564921 -1.772482 -2.320437 0

3 -1.074135 0.008384 -1.657069 -0.545582 0.592744 -0.487783 -0.398512 -0.115182 -0.050335 1.234244 -0.098192 1.220133 3.187939 2.022729 0.275312 0.582795 0

4 1.646339 -0.541342 -1.397092 -0.469216 -0.549595 -0.487783 -0.398512 -1.136655 -0.050335 -0.192206 -0.098192 1.220133 -0.313642 2.022729 0.957910 0.582795 0

… … … … … … … … … … … … … … … … … …

7743 -0.886516 -0.491062 -1.267103 -0.517299 -0.549595 -0.487783 -0.398512 1.246783 -0.050335 -0.192206 -0.098192 -0.819477 -0.313642 -0.564921 0.957910 0.582795 1

7744 -0.136041 -0.540970 0.812714 -0.350424 -0.549595 1.619134 1.380441 0.225310 -0.050335 -0.192206 -0.098192 -0.819477 -0.313642 -0.564921 -1.089884 -1.352693 1

7745 -0.511278 1.364821 0.422748 -0.452246 1.354304 1.026867 1.973425 0.225310 -1.716998 1.234244 -0.098192 -0.819477 -0.313642 -0.564921 -1.089884 -2.320437 1

7746 -0.792707 -0.486593 -1.007126 -0.537097 -0.549595 -0.487783 -0.398512 0.225310 -1.716998

-0.192206 -0.098192 -0.819477 -0.313642 -0.564921 0.616611 0.582795 1

7747 -0.698897 0.898895 -1.137114 0.981746 -0.549595 2.007506 0.194472 0.225310 -0.050335 -0.192206 -0.098192 -0.819477 -0.313642 -0.564921 0.275312 -2.320437 1

7748 rows × 17 columns

final.drop(["y"], axis = 1)

y2 = final['y']

X2.shape , y2.shape

((7748, 16), (7748,))

#Splitting the data into test and train

X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.2)

Logistic Regression

Fitting on Logistic Regression model

clf = LogisticRegression()

clf.fit(X_train, y_train)

predictions on test data

y_pred = clf.predict(X_test)

Evaluation metrics

print("Accuracy:", accuracy_score(y_test, y_pred))

print("Precision:", precision_score(y_test, y_pred)) print("Recall:", recall_score(y_test, y_pred))

print("F1_score:", precision_score(y_test, y_pred))

print("AUC-ROC:", roc_auc_score(y_test, y_pred))

Accuracy: 0.8548387096774194

Precision: 0.8435374149659864

Recall: 0.8493150684931506

F1_score: 0.8435374149659864

AUC-ROC: 0.854535583027063

K Nearest Neighbours

#Fitting on KNN

clf1 = KNeighborsClassifier(n_neighbors=5)

clf1.fit(X_train, y_train)

#predictions on test data

y_pred1 = clf1.predict(X_test)

Evaluation metrics

print("Accuracy:", accuracy_score(y_test, y_pred1))

print("Precision:", precision_score(y_test, y_pred1))

print("Recall:", recall_score(y_test, y_pred1))

print("F1_score:", precision_score(y_test, y_pred1))

print("AUC-ROC:", roc_auc_score(y_test, y_pred1))

Accuracy: 0.8877419354838709

Precision: 0.8518987341772152

Recall: 0.9219178082191781

F1_score: 0.8518987341772152

AUC-ROC: 0.8896174406949549

Decision Tree

```
#Fitting on decision tree
clf2 = DecisionTreeClassifier()
clf2.fit(X_train, y_train)
#predictions on test data
y_pred2 = clf2.predict(X_test)
# Evaluation metrics
print("Accuracy:", accuracy_score(y_test, y_pred2))
print("Precision:", precision_score(y_test, y_pred2))
print("Recall:", recall_score(y_test, y_pred2))
print("F1_score:", precision_score(y_test, y_pred2))
print("AUC-ROC:", roc_auc_score(y_test, y_pred2))
```

Accuracy: 0.8870967741935484

Precision: 0.8627450980392157

Recall: 0.9041095890410958

F1_score: 0.8627450980392157

AUC-ROC: 0.8880304042766456

Random Forest Tree

```
#Fitting on random forest tree
clf3 = RandomForestClassifier(n_estimators=100)
clf3.fit(X_train, y_train)
#predictions on test data
y_pred3 = clf3.predict(X_test)
# Evaluation metrics
print("Accuracy:", accuracy_score(y_test, y_pred3))
print("Precision:", precision_score(y_test, y_pred3))
print("Recall:", recall_score(y_test, y_pred3))
print("F1_score:", precision_score(y_test, y_pred3))
print("AUC-ROC:", roc_auc_score(y_test, y_pred3))
```

```
Accuracy: 0.9329032258064516
Precision: 0.9129287598944591
Recall: 0.947945205479452
F1_score: 0.9129287598944591
AUC-ROC: 0.9337287003007017
```

```
#Fitting on random forest tree
clf3 = RandomForestClassifier(n_estimators=100)
clf3.fit(X_train, y_train)
```

```
#predictions on test data
y_pred3 = clf3.predict(X_test)
# Evaluation metrics
print("Accuracy:", accuracy_score(y_test, y_pred3))
print("Precision:", precision_score(y_test, y_pred3))
print("Recall:", recall_score(y_test, y_pred3))
print("F1_score:", precision_score(y_test, y_pred3))
print("AUC-ROC:", roc_auc_score(y_test, y_pred3))
```

```
Accuracy: 0.9329032258064516
Precision: 0.9129287598944591
Recall: 0.947945205479452
F1_score: 0.9129287598944591
AUC-ROC: 0.9337287003007017
```

Support vector machine

```
#Fitting on Support vector machine
clf4 = SVC()
clf4.fit(X_train, y_train)
#predictions on test data
y_pred4 = clf4.predict(X_test)
# Evaluation metrics
print("Accuracy:", accuracy_score(y_test, y_pred4))
```

print("Precision:", precision_score(y_test, y_pred4))

print("Recall:", recall_score(y_test, y_pred4))

print("F1_score:", precision_score(y_test, y_pred4))

print("AUC-ROC:", roc_auc_score(y_test, y_pred4))

Accuracy: 0.9

Precision: 0.891156462585034

Recall: 0.8972602739726028

F1_score: 0.891156462585034

AUC-ROC: 0.8998496491814234

Naive Bayes

#Fitting Naive Bayes model

clf5 = GaussianNB()

clf5.fit(X_train, y_train)

#predictions on test data

y_pred5 = clf5.predict(X_test)

Evaluation metrics

print("Accuracy:", accuracy_score(y_test, y_pred5))

print("Precision:", precision_score(y_test, y_pred5))

print("Recall:", recall_score(y_test, y_pred5))

print("F1_score:", precision_score(y_test, y_pred5))

print("AUC-ROC:", roc_auc_score(y_test, y_pred5))

Accuracy: 0.7296774193548388

Precision: 0.6468366383380547

Recall: 0.9383561643835616

F1_score: 0.6468366383380547

AUC-ROC: 0.741129301703976

Gradient Boosting

```
# Fitting Gradient Boosting Classifier
clf6 = GradientBoostingClassifier()
clf6.fit(X_train, y_train)
#predictions on test data
y_pred6 = clf6.predict(X_test)
# Evaluation metrics
print("Accuracy:", accuracy_score(y_test, y_pred5))
print("Precision:", precision_score(y_test, y_pred5))
print("Recall:", recall_score(y_test, y_pred5))
print("F1_score:", precision_score(y_test, y_pred5))
print("AUC-ROC:", roc_auc_score(y_test, y_pred5))
```

Accuracy: 0.7296774193548388

Precision: 0.6468366383380547

Recall: 0.9383561643835616

F1_score: 0.6468366383380547

AUC-ROC: 0.741129301703976

Accuracy: 0.9348387096774193

Precision: 0.918774966711052

Recall: 0.9452054794520548

F1_score: 0.918774966711052

AUC-ROC: 0.935407617774808

#Fitting Naive Bayes model

param5 = GaussianNB()

param5.fit(X_train, y_train)

#predictions on test data

y_pred_param5 = param5.predict(X_test)

Evaluation metrics

print("Accuracy:", accuracy_score(y_test, y_pred_param5))

print("Precision:", precision_score(y_test, y_pred_param5))

print("Recall:", recall_score(y_test, y_pred_param5))

print("F1_score:", precision_score(y_test, y_pred_param5))

print("AUC-ROC:", roc_auc_score(y_test, y_pred_param5))

Accuracy: 0.7296774193548388

Precision: 0.6468366383380547

Recall: 0.9383561643835616

F1_score: 0.6468366383380547

AUC-ROC: 0.741129301703976

```
# Fitting Gradient Boosting Classifier
param6 = GradientBoostingClassifier(loss='deviance',
                                    learning_rate=0.5,
                                    n_estimators=1000,
                                    criterion='friedman_mse',
                                    min_samples_split=5,
                                    min_samples_leaf=1,


#Fitting on random forest tree
param3 = RandomForestClassifier(n_estimators=1000,
criterion='entropy', max_depth=None,
                                min_samples_split=2,
min_samples_leaf=1, max_features='auto',
                                bootstrap=True, oob_score=False)
param3.fit(X_train, y_train)
#predictions on test data
y_pred_param3 = param3.predict(X_test)
```

```
# Evaluation metrics
print("Accuracy:", accuracy_score(y_test,
y_pred_param3))
print("Precision:", precision_score(y_test,
y_pred_param3))
print("Recall:", recall_score(y_test, y_pred_param3))
print("F1_score:", precision_score(y_test,
y_pred_param3))
print("AUC-ROC:", roc_auc_score(y_test,
y_pred_param3))
```

```
Accuracy: 0.9335483870967742
Precision: 0.9163346613545816
Recall: 0.9452054794520548
F1_score: 0.9163346613545816
AUC-ROC: 0.934188105579686
```

```
# predicting the train label
y_pred_train = param3.predict(X_train)
print(classification_report(y_train,y_pred_train))
```

```
          precision recall f1-score support

        0 1.00 1.00 1.00 3054
        1 1.00 1.00 1.00 3144
```

```
    accuracy 1.00 6198
   macro avg 1.00 1.00 1.00 6198
weighted avg 1.00 1.00 1.00 6198
```

```
print(f1_score(y_train,y_pred_train,average='weighted'))
```

1.0

```
train_fpr,train_tpr,train_threshold = roc_curve(y_train,y_pred_train)
print(train_fpr)
print(train_tpr)
print(train_threshold)
```

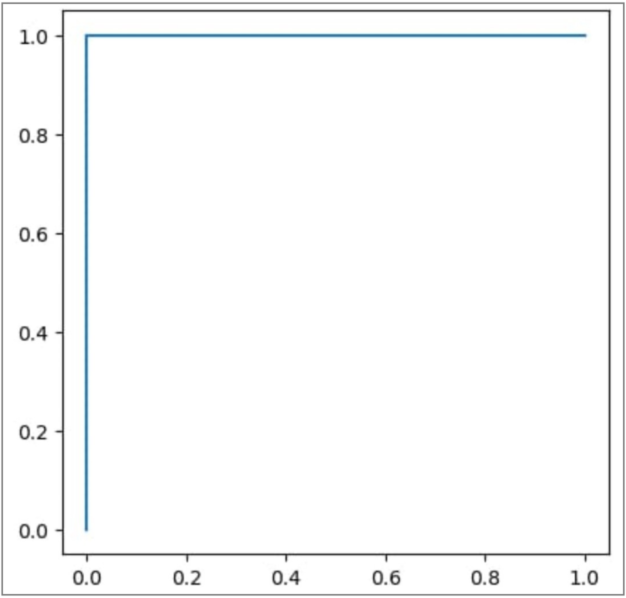
[0. 0. 1.]

[0. 1. 1.]

[2 1 0]

```
plt.figure(figsize=(5,5))
plt.plot(train_fpr,train_tpr)
```

```
[<matplotlib.lines.Line2D at 0x23252a77a30>]
```



Now predicting on test data

y_pred_test = param3.predict(X_test)

print(classification_report(y_test,y_pred_test))

precision recall f1-score support

0 0.95 0.92 0.94 820

1 0.92 0.95 0.93 730

```
    accuracy 0.93 1550

   macro avg 0.93 0.93 0.93 1550

weighted avg 0.93 0.93 0.93 1550
```

```
print(f1_score(y_test,y_pred_test,average='weighted'))
```

0.9335910899598903

```
train_fpr2,train_tpr2,train_threshold2 =
roc_curve(y_test,y_pred_test)

print(train_fpr2)

print(train_tpr2)

print(train_threshold2)
```

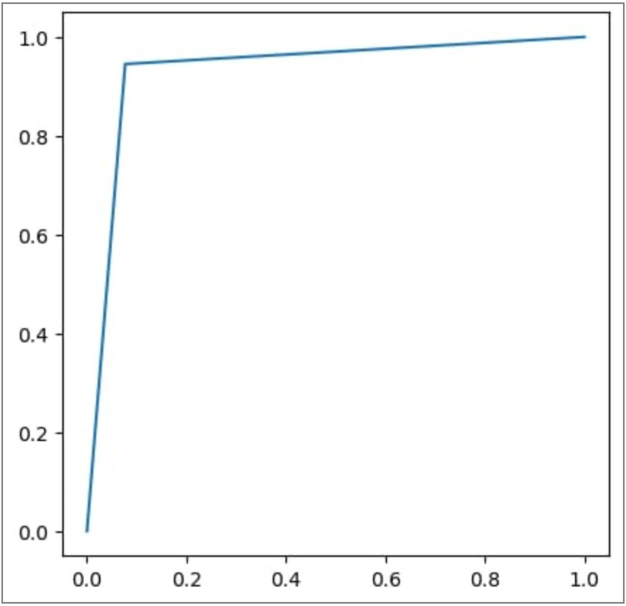
[0. 0.07682927 1.]

[0. 0.94520548 1.]

[2 1 0]

```
plt.figure(figsize=(5,5))

plt.plot(train_fpr2,train_tpr2)
```

[<matplotlib.lines.Line2D at 0x232527a34c0>]


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0