Import Libraries and load the train and test dataset

```
#Import some libraries to perform some calculations, visualization, plotting and other usage of functions

import pandas as pd
import matplotlib.pyplot as plt
import seaborn  as sns
import sklearn
import numpy as np
import datetime as dt
import math
import scipy as sp
import scipy.stats as stat
```

```
#Load the train dataset of BANK and stored in variable called bank:

bank = pd.read_csv("/content/new_train (1).csv")
bank
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	y
0	49	blue-collar	married	basic.9y	unknown	no	no	cellular	nov	wed	227	4	999	0	nonexistent	no
1	37	entrepreneur	married	university.degree	no	no	no	telephone	nov	wed	202	2	999	1	failure	no
2	78	retired	married	basic.4y	no	no	no	cellular	jul	mon	1148	1	999	0	nonexistent	yes
3	36	admin.	married	university.degree	no	yes	no	telephone	may	mon	120	2	999	0	nonexistent	no
4	59	retired	divorced	university.degree	no	no	no	cellular	jun	tue	368	2	999	0	nonexistent	no
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
32945	28	services	single	high.school	no	yes	no	cellular	jul	tue	192	1	999	0	nonexistent	no
32946	52	technician	married	professional.course	no	yes	no	cellular	nov	fri	64	1	999	1	failure	no
32947	54	admin.	married	basic.9y	no	no	yes	cellular	jul	mon	131	4	999	0	nonexistent	no
32948	29	admin.	married	university.degree	no	no	no	telephone	may	fri	165	1	999	0	nonexistent	no
32949	35	admin.	married	university.degree	no	no	yes	telephone	jun	tue	544	3	999	0	nonexistent	no

32950 rows × 16 columns

```
#Load Test dataset of BANK and stored in variable called bank_t:

bank_t = pd.read_csv("/content/new_test.csv")
bank_t
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	poutcome
0	32	4	0	6	0	0	0	0	3	3	131	5	1
1	37	10	3	6	0	0	0	0	4	3	100	1	1
2	55	5	0	5	1	2	0	0	3	2	131	2	1
3	44	2	1	0	1	0	0	1	4	3	48	2	1
4	28	0	2	3	0	0	0	0	5	0	144	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
8233	48	4	1	2	0	2	0	0	6	3	554	1	1
8234	30	7	2	3	0	2	0	0	6	0	159	1	1
8235	33	7	1	3	0	0	0	0	4	1	472	1	0
8236	44	1	1	1	0	2	2	1	6	1	554	5	1
8237	42	1	1	2	1	2	0	0	6	3	83	5	1

8238 rows × 13 columns

Run Basic Pandas Commands on Bank(Train) dataset:

```
#This command gives the information of train dataset:

bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32950 entries, 0 to 32949
Data columns (total 16 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   age          32950 non-null  int64 
 1   job          32950 non-null  object
 2   marital      32950 non-null  object
 3   education    32950 non-null  object
 4   default      32950 non-null  object
 5   housing      32950 non-null  object
 6   loan         32950 non-null  object
 7   contact      32950 non-null  object
 8   month        32950 non-null  object
 9   day_of_week  32950 non-null  object
 10  duration     32950 non-null  int64 
 11  campaign     32950 non-null  int64 
 12  pdays        32950 non-null  int64 
 13  previous     32950 non-null  int64 
 14  poutcome     32950 non-null  object
 15  y            32950 non-null  object
dtypes: int64(5), object(11)
memory usage: 4.0+ MB
```

```
#This command gives the static information of train dataset:

bank.describe()
```

	age	duration	campaign	pdays	previous
count	32950.000000	32950.000000	32950.000000	32950.000000	32950.000000
mean	40.014112	258.127466	2.560607	962.052413	0.174719
std	10.403636	258.975917	2.752326	187.951096	0.499025
min	17.000000	0.000000	1.000000	0.000000	0.000000

```
#This command shows the order pair of train dataset:

print("Order pair of given dataset is: ")
bank.shape
```

```
Order pair of given dataset is:
(32950, 16)
```

```
#This command shows the columns of train dataset:

bank.columns
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'y'],
      dtype='object')
```

```
#This command gives the missing values of train dataset:

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

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

```
#This command gives the duplicated values  of train dataset:

print("The total duplicated values are: ")
bank.duplicated().sum()
```

```
The total duplicated values are:
8
```

```
#This command drops the duplicated values  of train dataset:

bank.drop_duplicates(inplace = True)
print("After drop dulicate values: ", bank.duplicated().sum())
```

```
After drop dulicate values:  0
```

```
#This command shows the order pair of train dataset:

print("Order pair of given dataset after drop the duplicate value is: ")
bank.shape
```

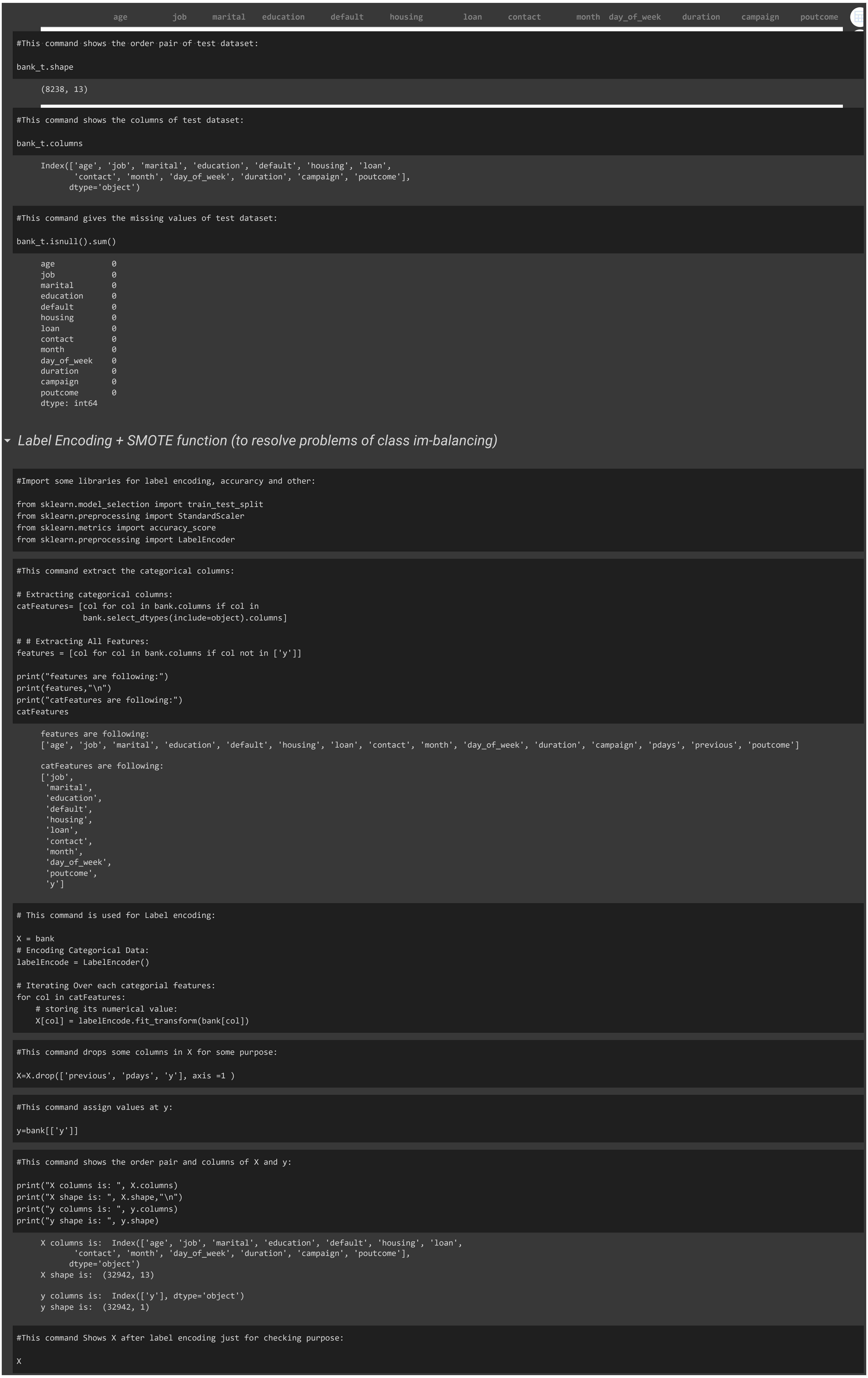
```
Order pair of given dataset after drop the duplicate value is:
(32942, 16)
```

Run Basic Pnadas Commands on Bank(Test) dataset:

```
#This command gives the information of test dataset:

bank_t.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8238 entries, 0 to 8237
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   age          8238 non-null   int64
 1   job          8238 non-null   int64
 2   marital      8238 non-null   int64
 3   education    8238 non-null   int64
 4   default      8238 non-null   int64
 5   housing      8238 non-null   int64
 6   loan         8238 non-null   int64
 7   contact      8238 non-null   int64
 8   month        8238 non-null   int64
 9   day_of_week  8238 non-null   int64
 10  duration     8238 non-null   int64
 11  campaign     8238 non-null   int64
 12  poutcome     8238 non-null   int64
dtypes: int64(13)
memory usage: 836.8 KB
```

```
#This command gives the static information of test dataset:

bank_t.describe()
```



```
        age     job     marital     education     default     housing     loan     contact     month  day_of_week     duration     campaign     poutcome
```

```
#This command shows the order pair of test dataset:

bank_t.shape
```

```
(8238, 13)
```

```
#This command shows the columns of test dataset:

bank_t.columns
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'poutcome'],
      dtype='object')
```

```
#This command gives the missing values of test dataset:

bank_t.isnull().sum()
```

```
age            0
job            0
marital        0
education      0
default        0
housing        0
loan           0
contact        0
month          0
day_of_week    0
duration       0
campaign       0
poutcome       0
dtype: int64
```

Label Encoding + SMOTE function (to resolve problems of class im-balancing)

```
#Import some libraries for label encoding, accurarcy and other:

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
```

```
#This command extract the categorical columns:

# Extracting categorical columns:
catFeatures= [col for col in bank.columns if col in
              bank.select_dtypes(include=object).columns]

# # Extracting All Features:
features = [col for col in bank.columns if col not in ['y']]

print("features are following:")
print(features,"\n")
print("catFeatures are following:")
catFeatures
```

```
features are following:
['age', 'job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays', 'previous', 'poutcome']

catFeatures are following:
['job',
 'marital',
 'education',
 'default',
 'housing',
 'loan',
 'contact',
 'month',
 'day_of_week',
 'poutcome',
 'y']
```

```
# This command is used for Label encoding:

X = bank
# Encoding Categorical Data:
labelEncode = LabelEncoder()

# Iterating Over each categorial features:
for col in catFeatures:
    # storing its numerical value:
    X[col] = labelEncode.fit_transform(bank[col])
```

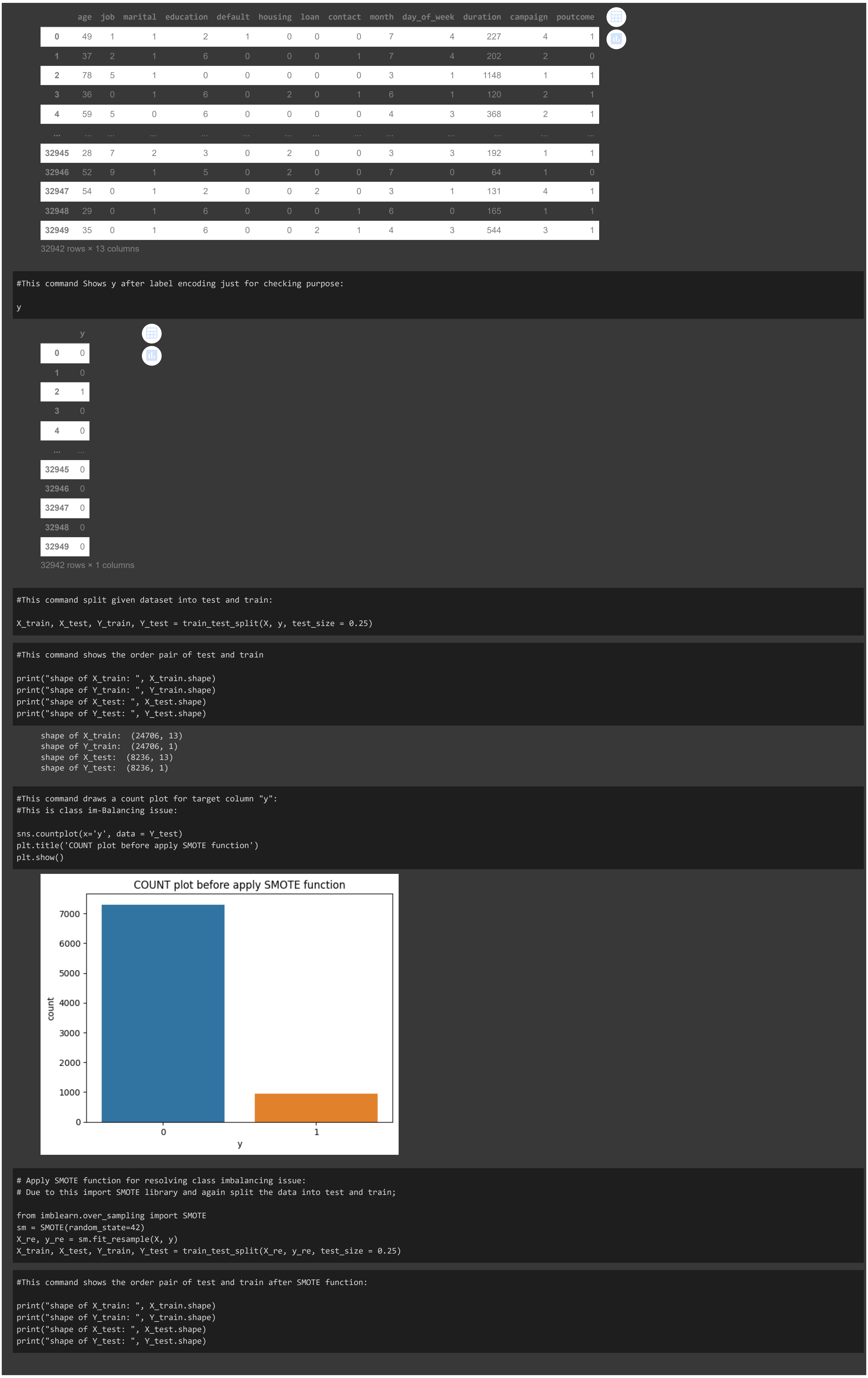
```
#This command drops some columns in X for some purpose:

X=X.drop(['previous', 'pdays', 'y'], axis =1 )
```

```
#This command assign values at y:

y=bank[['y']]
```

```
#This command shows the order pair and columns of X and y:

print("X columns is: ", X.columns)
print("X shape is: ", X.shape,"\n")
print("y columns is: ", y.columns)
print("y shape is: ", y.shape)
```

```
X columns is:  Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'poutcome'],
      dtype='object')
X shape is:  (32942, 13)

y columns is:  Index(['y'], dtype='object')
y shape is:  (32942, 1)
```

```
#This command Shows X after label encoding just for checking purpose:

X
```



	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	poutcome
0	49	1	1	2	1	0	0	0	7	4	227	4	1
1	37	2	1	6	0	0	0	1	7	4	202	2	0
2	78	5	1	0	0	0	0	0	3	1	1148	1	1
3	36	0	1	6	0	2	0	1	6	1	120	2	1
4	59	5	0	6	0	0	0	0	4	3	368	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
32945	28	7	2	3	0	2	0	0	3	3	192	1	1
32946	52	9	1	5	0	2	0	0	7	0	64	1	0
32947	54	0	1	2	0	0	2	0	3	1	131	4	1
32948	29	0	1	6	0	0	0	1	6	0	165	1	1
32949	35	0	1	6	0	0	2	1	4	3	544	3	1

32942 rows × 13 columns

```
#This command Shows y after label encoding just for checking purpose:

y
```

	y
0	0
1	0
2	1
3	0
4	0
...	...
32945	0
32946	0
32947	0
32948	0
32949	0

32942 rows × 1 columns

```
#This command split given dataset into test and train:

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = 0.25)
```

```
#This command shows the order pair of test and train

print("shape of X_train: ", X_train.shape)
print("shape of Y_train: ", Y_train.shape)
print("shape of X_test: ", X_test.shape)
print("shape of Y_test: ", Y_test.shape)
```

```
shape of X_train:  (24706, 13)
shape of Y_train:  (24706, 1)
shape of X_test:  (8236, 13)
shape of Y_test:  (8236, 1)
```

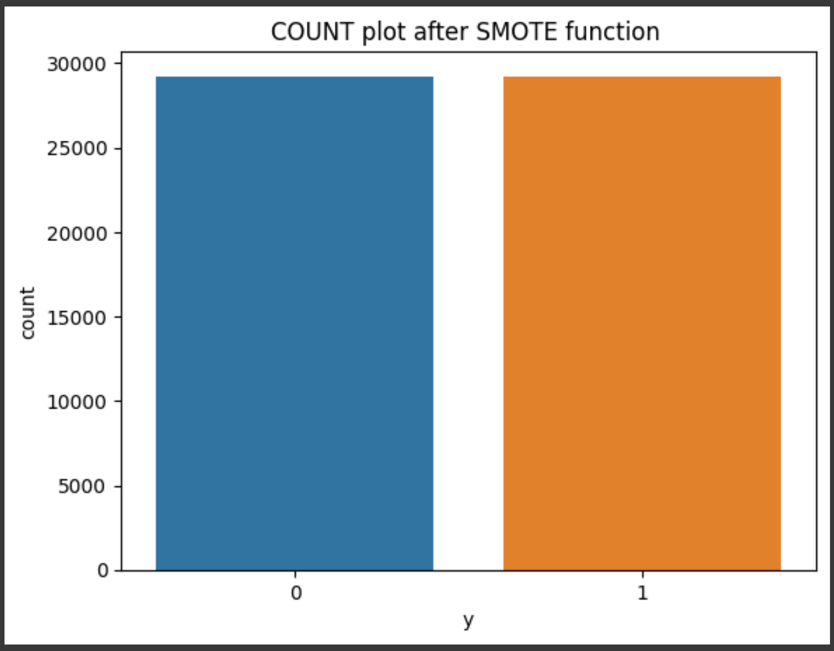
```
#This command draws a count plot for target column "y":
#This is class im-Balancing issue:

sns.countplot(x='y', data = Y_test)
plt.title('COUNT plot before apply SMOTE function')
plt.show()
```

```
# Apply SMOTE function for resolving class imbalancing issue:
# Due to this import SMOTE library and again split the data into test and train;

from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
X_re, y_re = sm.fit_resample(X, y)
X_train, X_test, Y_train, Y_test = train_test_split(X_re, y_re, test_size = 0.25)
```

```
#This command shows the order pair of test and train after SMOTE function:

print("shape of X_train: ", X_train.shape)
print("shape of Y_train: ", Y_train.shape)
print("shape of X_test: ", X_test.shape)
print("shape of Y_test: ", Y_test.shape)
```

```
shape of X_train:  (43845, 13)
shape of Y_train:  (43845, 1)
shape of X_test:  (14615, 13)
shape of Y_test:  (14615, 1)
```

```
#Resolve class im-Balnacing issue, The major difference is gone between yes and no:

sns.countplot(x='y', data = y_re)
plt.title('COUNT plot after SMOTE function ')
plt.show()
```



CLASSIFICATION MODELS

```
#Import some libraries of models and matrices:

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn import metrics
```

```
# import warning libraries for neglecting warning:

import warnings
warnings.filterwarnings("ignore")
```

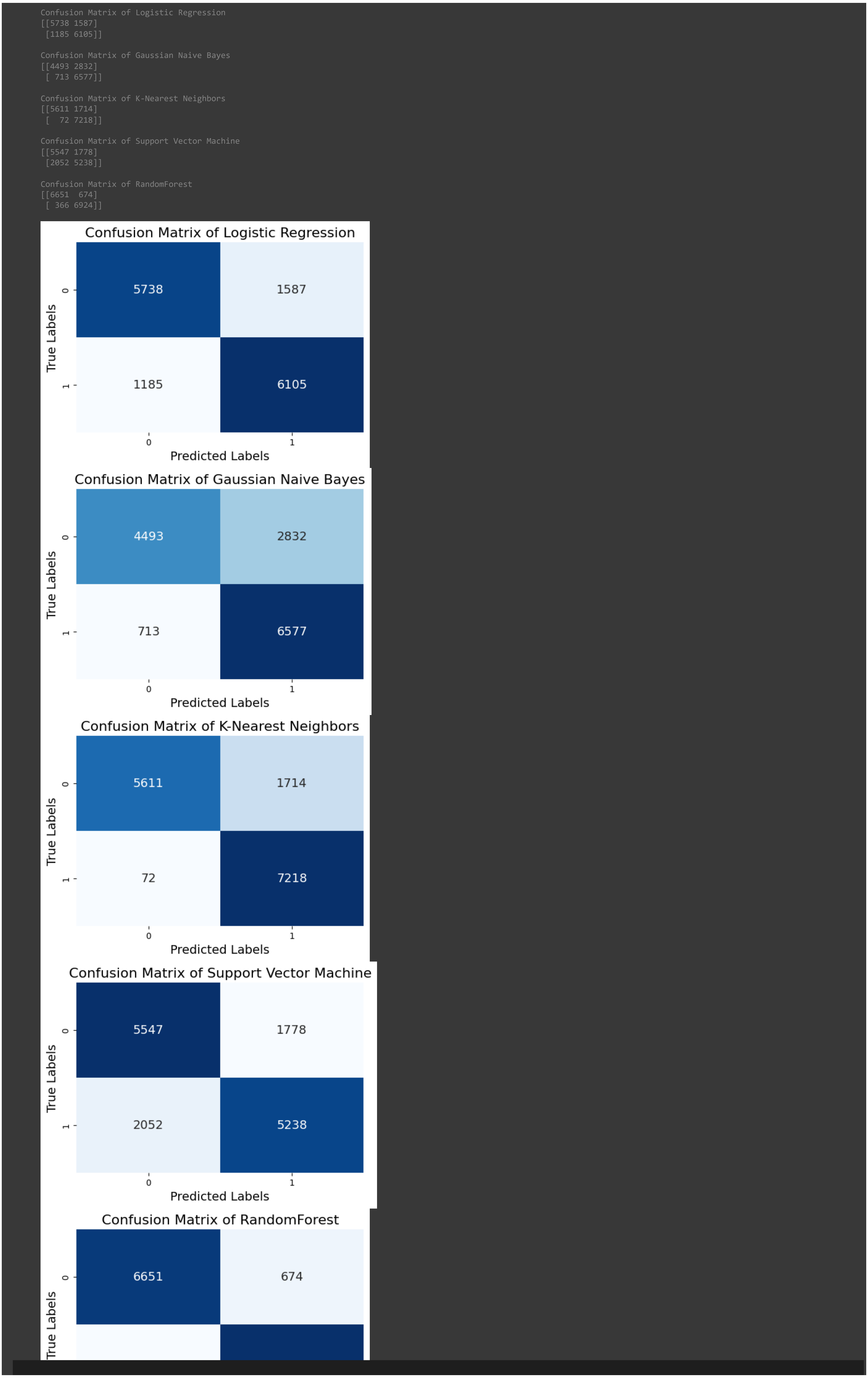
```
#This command loads the multiple model in just one code:

models = {
    "Logistic Regression": LogisticRegression(),
    "Gaussian Naive Bayes": GaussianNB(),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Support Vector Machine": SVC(),
    "RandomForest": RandomForestClassifier()
}

for name, model in models.items():
    model.fit(X_train, Y_train)
    print(name + " trained.")
```

```
Logistic Regression trained.
Gaussian Naive Bayes trained.
K-Nearest Neighbors trained.
Support Vector Machine trained.
RandomForest trained.
```

```
#This command gives multiple model score in just one code:

for name, model in models.items():
  print(name +": {:.2f}%".format(model.score(X_test, Y_test)*100))
```

```
Logistic Regression: 81.03%
Gaussian Naive Bayes: 75.74%
K-Nearest Neighbors: 87.78%
Support Vector Machine: 73.79%
RandomForest: 92.88%
```

```
#This command gives the confusion metrix of multiple models:

for name, model in models.items():
  y_pred = model.predict(X_test)
  # PRINT THE CONFUSION MATRIX
  print("Confusion Matrix of "+name)
  cm = confusion_matrix(Y_test, y_pred)
  print(cm, "\n")
  plt.figure(figsize = (6, 4))
  sns.heatmap(cm, annot = True, fmt = 'd', cmap = 'Blues', cbar = False, annot_kws = {'size' : 14})
  plt.xlabel('Predicted Labels', fontsize = 14)
  plt.ylabel('True Labels', fontsize = 14)
  plt.title('Confusion Matrix of '+name, fontsize = 16)
plt.show()
```



```
Confusion Matrix of Logistic Regression
[[5738 1587]
 [1185 6105]]

Confusion Matrix of Gaussian Naive Bayes
[[4493 2832]
 [ 713 6577]]

Confusion Matrix of K-Nearest Neighbors
[[5611 1714]
 [  72 7218]]

Confusion Matrix of Support Vector Machine
[[5547 1778]
 [2052 5238]]

Confusion Matrix of RandomForest
[[6651  674]
 [ 366 6924]]
```

```
#This command is used to predict the target column ("y") of Test dataset depends on multiple model which we have trained:

for name, model in models.items():
  y_predicted = model.predict(bank_t)
  print(name ,"Prediction :", y_predicted)
```

```
Logistic Regression Prediction : [0 0 0 ... 1 0 0]
Gaussian Naive Bayes Prediction : [0 1 0 ... 1 0 0]
K-Nearest Neighbors Prediction : [0 0 0 ... 1 0 0]
Support Vector Machine Prediction : [0 0 0 ... 1 1 0]
RandomForest Prediction : [0 0 0 ... 1 0 0]
```

```
#This command extract target column "y" of RandomForest
#because this model has high accuracy

if name == 'RandomForest':
  print(y_predicted)
y_predicted.shape
```

```
[0 0 0 ... 1 0 0]
(8238,)
```

```
#Put target column ("y") in test dataset for prediction:

prediction= pd.DataFrame(y_predicted, columns=["y_predicted"])

prediction_dataset = pd.concat([bank_t, prediction], axis=1)
prediction_dataset
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	poutcome	y_predicted
0	32	4	0	6	0	0	0	0	3	3	131	5	1	0
1	37	10	3	6	0	0	0	0	4	3	100	1	1	0
2	55	5	0	5	1	2	0	0	3	2	131	2	1	0
3	44	2	1	0	1	0	0	1	4	3	48	2	1	0
4	28	0	2	3	0	0	0	0	5	0	144	2	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8233	48	4	1	2	0	2	0	0	6	3	554	1	1	1
8234	30	7	2	3	0	2	0	0	6	0	159	1	1	0
8235	33	7	1	3	0	0	0	0	4	1	472	1	0	1
8236	44	1	1	1	0	2	2	1	6	1	554	5	1	0
8237	42	1	1	2	1	2	0	0	6	3	83	5	1	0

8238 rows × 14 columns

```
#This command shows the order pair of test dataset before adding prediction column:

bank_t.shape
```

```
(8238, 13)
```

```
#This command shows the order pair of test dataset after adding prediction column:

prediction_dataset.shape
```

```
(8238, 14)
```