Import necessory libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Load train and test file

In [2]:
```
trainD = pd.read_csv("epsilon_train.csv")
trainD = pd.DataFrame(trainD)
testD = pd.read_csv("epsilon_test.csv")
testD = pd.DataFrame(testD)
trainD.head()
```

Out[2]:

	ID	Default	Checking_amount	Term	Credit_score	Gender	Marital_status	Car_loan	Personal_loan	Home_loan	Educatio
0	101	No		988.0	15.0	796.0	Female	Single	Yes	No	No
1	102	No		458.0	15.0	813.0	Female	Single	Yes	No	No
2	103	No		158.0	14.0	756.0	Female	Single	No	Yes	No
3	104	Yes		300.0	25.0	737.0	Female	Single	No	No	No
4	105	Yes		63.0	24.0	662.0	Female	Single	No	No	No

In [3]:
```
# Train datatype info
trainD.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 804 entries, 0 to 803
Data columns (total 17 columns):
ID                  804 non-null int64
Default             803 non-null object
Checking_amount     803 non-null float64
Term                803 non-null float64
Credit_score        802 non-null float64
Gender              804 non-null object
Marital_status      804 non-null object
Car_loan            803 non-null object
Personal_loan       801 non-null object
Home_loan           802 non-null object
Education_loan      803 non-null object
Emp_status 804 non-null object
Amount              803 non-null float64
Saving_amount       804 non-null int64
Emp_duration        801 non-null float64
Age                 803 non-null float64
No_of_credit_acc 803 non-null float64
dtypes: float64(7), int64(2), object(8)
memory usage: 106.9+ KB
```

In [4]:
```
# Check columnwise null values
print(trainD.isnull().sum())
```

```
ID                  0
Default             1
Checking_amount     1
Term                1
Credit_score        2
Gender              0
Marital_status      0
Car_loan            1
Personal_loan       3
Home_loan           2
Education_loan      1
Emp_status          0
Amount              1
Saving_amount       0
Emp_duration        3
Age                 1
No_of_credit_acc    1
dtype: int64
```

Handling the missing values (Fill with appropriate mean, mode or median value)

In [5]:
```
# Default column
trainD.Default.value_counts(sort=True)
trainD.Default.fillna('Yes',inplace=True)

#Checking amount column
trainD.Checking_amount.value_counts(sort=True)
trainD.Checking_amount.fillna(trainD["Checking_amount"].mean(),inplace=True)

 #Term column
trainD.Term.value_counts(sort=True)
trainD.Term.fillna(trainD["Term"].mean(),inplace=True)

# credit score column
trainD.Credit_score.value_counts(sort=True)
trainD.Credit_score.fillna(trainD["Credit_score"].mean(),inplace=True)

# Car_loan column
trainD.Car_loan.value_counts(sort=True)
trainD.Car_loan.fillna("Yes",inplace=True)

# personal loan column
trainD.Personal_loan.value_counts(sort=True)
trainD.Personal_loan.fillna('Yes',inplace=True)

# home loan column
trainD.Home_loan.value_counts(sort=True)
trainD.Home_loan.fillna('Yes',inplace=True)

# Education loan column
trainD.Education_loan.value_counts(sort=True)
trainD.Education_loan.fillna('Yes',inplace=True)

# amount
trainD.Amount.value_counts(sort=True)
trainD.Amount.fillna(trainD["Amount"].mean(),inplace=True)

# Emp_duration
trainD.iloc[:,14].value_counts(sort=True)
trainD.iloc[:,14].fillna(trainD.iloc[:,14].mean(),inplace=True)

#Age
trainD.Age.value_counts(sort=True)
trainD.Age.fillna(trainD["Age"].mean(),inplace=True)

#No_of_credit_acc column
trainD.No_of_credit_acc.value_counts(sort=True)
trainD.No_of_credit_acc.fillna(1,inplace=True)
```

In [6]:
```
# Drop all the rows which contains null values
# trainD = trainD.dropna(axis = 0, how ='any')
```

In [7]:
```
# check whether all null values are removed or not
print(trainD.isnull().sum())
```

```
ID                  0
Default             0
Checking_amount     0
Term                0
Credit_score        0
Gender              0
Marital_status      0
Car_loan            0
Personal_loan       0
Home_loan           0
Education_loan      0
Emp_status          0
Amount              0
Saving_amount       0
Emp_duration        0
Age                 0
No_of_credit_acc    0
dtype: int64
```

In [8]:
```
testD.head()
```

Out[8]:

	ID	Checking_amount	Term	Credit_score	Gender	Marital_status	Car_loan	Personal_loan	Home_loan	Education_loan
0	1001	111.0	20	668.0	Male	Married	No	Yes	No	No
1	1002	270.0	19	612.0	Male	Married	No	No	No	Yes
2	1003	211.0	19	751.0	Male	Married	Yes	No	No	No
3	1004	325.0 16		722.0	Male	Married	Yes	No	No	No
4	1005	1237.0 21		841.0	Male	Married	Yes	No	No	No

In [9]:
```
testD.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 203 entries, 0 to 202
Data columns (total 16 columns):
ID                  203 non-null int64
Checking_amount  202 non-null float64
Term                203 non-null int64
Credit_score        202 non-null float64
Gender              203 non-null object
Marital_status   202  non-null  object
Car_loan            202 non-null object
Personal_loan    203  non-null  object
Home_loan           201 non-null object
Education_loan   201   non-null   object
Emp_status 203 non-null object
Amount              202 non-null  float64
Saving_amount   202   non-null   float64
Emp_duration 201 non-null float64
Age                 203 non-null int64
No_of_credit_acc 202 non-null float64
dtypes: float64(6), int64(3), object(7)
memory usage: 25.5+ KB
```

In [10]:
```
print(testD.isnull().sum())
```

```
ID                  0
Checking_amount     1
Term                0
Credit_score        1
Gender              0
Marital_status      1
Car_loan            1
Personal_loan       0
Home_loan           2
Education_loan      2
Emp_status          0
Amount              1
Saving_amount       1
Emp_duration        2
Age                 0
No_of_credit_acc    1
dtype: int64
```

In [12]:
```
testD=testD.dropna(axis=0,how='any')
```

In [45]:
```
print(testD.isnull().sum())
```

```
ID                  0
Checking_amount     0
Term                0
Credit_score        0
Gender              0
Marital_status      0
Car_loan            0
Personal_loan       0
Home_loan           0
Education_loan      0
Emp_status          0
Amount              0
Saving_amount       0
Emp_duration        0
Age                 0
No_of_credit_acc    0
dtype: int64
```

Label Encoding of Categorical Variables

In [14]:
```
from sklearn.preprocessing import LabelEncoder
```

In [15]:
```
number=LabelEncoder()
trainD['Gender']=number.fit_transform(trainD['Gender'].astype('str'))
testD['Gender']=number.fit_transform(testD['Gender'].astype('str'))
```

In [16]:
```
trainD.head()
```

Out[16]:

	ID	Default	Checking_amount	Term	Credit_score	Gender	Marital_status	Car_loan	Personal_loan	Home_loan	Educatio
0	101	No		988.0	15.0	796.0	0	Single	Yes	No	No
1	102	No		458.0	15.0	813.0	0	Single	Yes	No	No
2	103	No		158.0	14.0	756.0	0	Single	No	Yes	No
3	104	Yes		300.0	25.0	737.0	0	Single	No	No	No
4	105	Yes		63.0	24.0	662.0	0	Single	No	No	No

In [17]:
```
testD.head()
```

Out[17]:

	ID	Checking_amount	Term	Credit_score	Gender	Marital_status	Car_loan	Personal_loan	Home_loan	Education_loan
0	1001	111.0	20	668.0	1	Married	No	Yes	No	No
1	1002	270.0	19	612.0	1	Married	No	No	No	Yes
2	1003	211.0	19	751.0	1	Married	Yes	No	No	No
3	1004	325.0	16	722.0	1	Married	Yes	No	No	No
4	1005	1237.0	21	841.0	1	Married	Yes	No	No	No

In [18]:
```
#LabelEncoding for on more variables
number=LabelEncoder()
trainD['Personal_loan']=number.fit_transform(trainD['Personal_loan'].astype('str'))
testD['Personal_loan']=number.fit_transform(testD['Personal_loan'].astype('str'))
```

In [19]:
```
trainD.head()
```

Out[19]:

	ID	Default	Checking_amount	Term	Credit_score	Gender	Marital_status	Car_loan	Personal_loan	Home_loan	Educatio
0	101	No		988.0	15.0	796.0	0	Single	Yes	0	No
1	102	No		458.0	15.0	813.0	0	Single	Yes	0	No
2	103	No		158.0	14.0	756.0	0	Single	No	1	No
3	104	Yes		300.0	25.0	737.0	0	Single	No	0	No
4	105	Yes		63.0	24.0	662.0	0	Single	No	0	No

In [20]:
```
trainD['Emp_status']=number.fit_transform(trainD['Emp_status'].astype('str'))
testD['Emp_status']=number.fit_transform(testD['Emp_status'].astype('str'))
```

In [21]:
```
trainD.head()
```

Out[21]:

	ID	Default	Checking_amount	Term	Credit_score	Gender	Marital_status	Car_loan	Personal_loan	Home_loan	Educatio
0	101	No		988.0	15.0	796.0	0	Single	Yes	0	No
1	102	No		458.0	15.0	813.0	0	Single	Yes	0	No
2	103	No		158.0	14.0	756.0	0	Single	No	1	No
3	104	Yes		300.0	25.0	737.0	0	Single	No	0	No
4	105	Yes		63.0	24.0	662.0	0	Single	No	0	No

In [22]:
```
trainD['Default']=number.fit_transform(trainD['Default'].astype('str'))
```

In [23]:
```
trainD['Car_loan']=number.fit_transform(trainD['Car_loan'].astype('str'))
testD['Car_loan']=number.fit_transform(testD['Car_loan'].astype('str'))
```

In [24]:
```
trainD['Home_loan']=number.fit_transform(trainD['Home_loan'].astype('str'))
testD['Home_loan']=number.fit_transform(testD['Home_loan'].astype('str'))
```

In [25]:
```
trainD['Education_loan']=number.fit_transform(trainD['Education_loan'].astype('str'))
testD['Education_loan']=number.fit_transform(testD['Education_loan'].astype('str'))
```

In [26]:
```
trainD['Marital_status']=number.fit_transform(trainD['Marital_status'].astype('str'))
testD['Marital_status']=number.fit_transform(testD['Marital_status'].astype('str'))
```

In [27]:
```
trainD.head()
```

Out[27]:

	ID	Default	Checking_amount	Term	Credit_score	Gender	Marital_status	Car_loan	Personal_loan	Home_loan	Educatio
0	101	0		988.0	15.0	796.0	0	1	1	0	0
1	102	0		458.0	15.0	813.0	0	1	1	0	0
2	103	0		158.0	14.0	756.0	0	1	0	1	0
3	104	1		300.0	25.0	737.0	0	1	0	0	0
4	105	1		63.0	24.0	662.0	0	1	0	0	0

Separating the Feature Variables and Target Variable

In [28]:
```
X=trainD[['ID','Checking_amount','Term','Credit_score','Gender','Marital_status','Car_loan',
'Personal_loan','Home_loan','Education_loan','Emp_status','Amount','Saving_amount','Emp_dura
tion','Age','No_of_credit_acc']]
```

In [29]:
```
y=trainD['Default']
```

Split the train data into train-validation data

In [30]:
```
#split X and y into training and testing datasets
from sklearn.model_selection import train_test_split
X_train,X_val,y_train,y_val=train_test_split(X,y,test_size=0.25,random_state=0)
```

1. Logistic Regression

In [31]:
```
#import classes
from sklearn.linear_model import LogisticRegression
model1 = LogisticRegression()
```

Train the model

In [32]:
```
lr = model1.fit(X_train,y_train)
```

```
E:\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default so
lver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

predict on validation data to check accuracy

In [33]:
```
pred = lr.predict(X_val)
```

Results of LR

In [34]:
```
from sklearn import metrics

print("Accuracy:",metrics.accuracy_score(y_val,pred))
print("Precision:",metrics.precision_score(y_val,pred))
print("Recall:",metrics.recall_score(y_val,pred))
```

```
Accuracy: 0.8855721393034826
Precision: 0.7719298245614035
Recall: 0.8148148148148148
```

In [35]:
```
# Confusion Matrix
cnf_matrix=metrics.confusion_matrix(y_val,pred)
cnf_matrix
```

Out[35]:
```
array([[134, 13],
       [ 10, 44]], dtype=int64)
```

Prediction on Test Data(predict the values of default column) using LR

In [36]:
```
# Create a submission file
submission1 = pd.DataFrame()
testD1 = testD.copy()

testD1['Default'] = lr.predict(testD1)

submission1['ID'] = testD1['ID']
submission1['Default'] = testD1['Default']


# Submission file to csv
submission1.to_csv('LR_Predictions.csv', index=False)
submission1.head()
```

Out[36]:

	ID	Default
0	1001	1
1	1002	1
2	1003	0
3	1004	1
4	1005	0

In [ ]:
```
```

2. Random Forest

In [39]:
```
from sklearn.ensemble import RandomForestClassifier

model2 = RandomForestClassifier(n_estimators=1000,random_state=20)

rf = model2.fit(X_train,y_train)
```

In [40]:
```
prediction_RF = rf.predict(X_val)
```

In [41]:
```
from sklearn import metrics

print("Accuracy=",metrics.accuracy_score(y_val, prediction_RF))
print("Precision:",metrics.precision_score(y_val,prediction_RF))
print("Recall:",metrics.recall_score(y_val,prediction_RF))
```

```
Accuracy= 0.9203980099502488
Precision: 0.8392857142857143
Recall: 0.8703703703703703
```

In [42]:
```
# Confusion Matrix
cnf_matrix_RF =metrics.confusion_matrix(y_val,prediction_RF)
cnf_matrix_RF
```

Out[42]:
```
array([[138,  9],
       [ 7, 47]], dtype=int64)
```

Prediction on test data using Random Forest

In [44]:
```
# Create a submission file
submission2 = pd.DataFrame()
testD2 = testD.copy()

testD2['Default'] = rf.predict(testD2)


submission2['ID'] = testD2['ID']
submission2['Default'] = testD2['Default']


# Submission file to csv
submission2.to_csv('RF_Predictions.csv', index=False)
submission2.head()
```

Out[44]:

	ID	Default
0	1001	1
1	1002	1
2	1003	0
3	1004	0
4	1005	0

Happy Learning !!