**Jupyter Notebook**

**Classification model based on Loan status, keeping amount duration and payments**

**Independent variables**

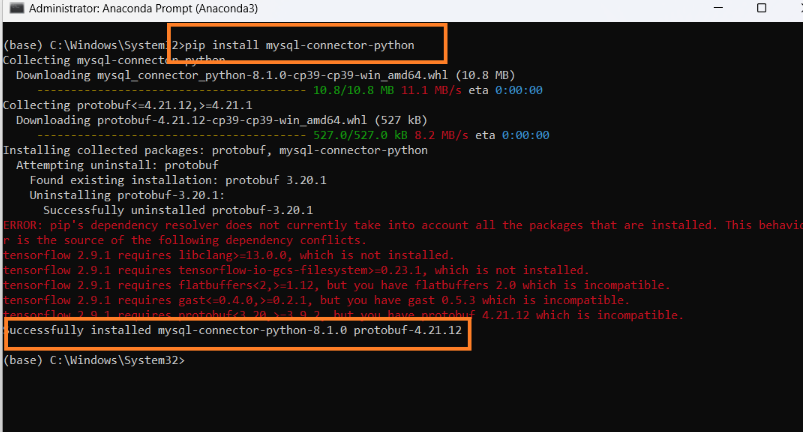
* loan\_amount
* duration
* payments

**Dependent variables**

* status

**Install MySQL Connector**

pip install mysql-connector-python



**Important Libraries**

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

import numpy as np

from matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 20, 10

**Create SQL**

sql= "select \* from acc\_ord\_card\_disp\_client\_dist aocdcd join loan\_trans lt on lt.account\_id= aocdcd.account\_id"

**Create Connection and Read Data as data Frame**

import mysql.connector as connection

try:

mydb = connection.connect(host="localhost", database = 'capstone\_prj',user="root", passwd="Bangalore",use\_pure=True)

df = pd.read\_sql(sql,mydb)

mydb.close() #close the connection

except Exception as e:

mydb.close()

print(str(e))

**Display Data**

display (df)

**Display Shape**

display (df.shape)

**Display all columns**

df.columns

**Display Null values**

print (df.isnull().sum())

**Display Unique values**

display (df.status.unique())

**Number of records in each category**

display (df.status.value\_counts())

**Label Encoding**

df.status=pd.DataFrame(df.status.map({'A':0,'B':1,'C':2,'D':3}),

columns=['status'])

display (df.status)

**Display Unique Values**

display (df.status.value\_counts())

**Create x – independent variable**

x=df[['loan\_amount','duration','payments']].values

display(x)

**Create Y - Dependent variable**

y=df['status'].values

display (y)

**Min Max Scaler**

from sklearn.preprocessing import MinMaxScaler

sc\_x= MinMaxScaler()

x = sc\_x.fit\_transform(x)

print (x)

**Train Test Split**

from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3,random\_state=143)

display (x\_train.shape)

display (x\_test.shape)

display (y\_train.shape)

display (y\_test.shape)

**Identify the best K Values**

accuracy=[]

for i in range(1,15):

knn=KNeighborsClassifier(n\_neighbors=i)

knn=knn.fit(x\_train,y\_train)

y\_pred =knn.predict(x\_train)

acc= accuracy\_score (y\_train,y\_pred)

accuracy.append (acc)

**Display the accuracy**

display (accuracy)

**Plot the Graph**

plt.plot(range(1,15),accuracy)

plt.show()

**Create KNN with N Neighbors = 3**

knn=KNeighborsClassifier(n\_neighbors=3)

knn=knn.fit(x\_train,y\_train)

**Prediction and check Accuracy**

y\_pred = knn.predict (x\_test)

display (accuracy\_score(y\_pred , y\_test))

**Create KNN with N Neighbors = 10**

knn=KNeighborsClassifier(n\_neighbors=10)

knn=knn.fit(x\_train,y\_train)

y\_pred = knn.predict (x\_test)

display (accuracy\_score(y\_pred , y\_test))

**Classification Report**

from sklearn.metrics import classification\_report

print(classification\_report (y\_test,y\_pred))

**Confusion Matrix**

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix (y\_test,y\_pred)

print (cm)

**Decision Tree with Criterion = Entropy**

from sklearn.tree import DecisionTreeClassifier

TREE = DecisionTreeClassifier (criterion = 'entropy')

TREE.fit(x\_train,y\_train)

TREE\_pred = TREE.predict(x\_test)

print (TREE\_pred)

print (pd.DataFrame(TREE\_pred).groupby(0).agg({0:np.size}))

print('\n Accuracy Score')

print (accuracy\_score (y\_test,TREE\_pred))

print('\nClassification Report')

print(classification\_report (y\_test,TREE\_pred))

print('Confusion Matrix')

print (confusion\_matrix (y\_test,TREE\_pred))

**Decision Tree with Criterion = Gini**

from sklearn.tree import DecisionTreeClassifier

TREE = DecisionTreeClassifier (criterion = 'gini')

TREE.fit(x\_train,y\_train)

TREE\_pred = TREE.predict(x\_test)

print (TREE\_pred)

print (pd.DataFrame(TREE\_pred).groupby(0).agg({0:np.size}))

print('\n Accuracy Score')

print (accuracy\_score (y\_test,TREE\_pred))

print('\nClassification Report')

print(classification\_report (y\_test,TREE\_pred))

print('Confusion Matrix')

print (confusion\_matrix (y\_test,TREE\_pred))

**Random Forest**

from sklearn.ensemble import RandomForestClassifier

RF = RandomForestClassifier(n\_estimators=50, criterion='entropy')

RF.fit(x\_train,y\_train)

RF\_pred = RF.predict(x\_test)

print (RF\_pred)

print (pd.DataFrame(RF\_pred).groupby(0).agg({0:np.size}))

print('\n Accuracy Score')

print (accuracy\_score (y\_test,RF\_pred))

print('\nClassification Report')

print(classification\_report (y\_test,RF\_pred))

print('Confusion Matrix')

print (confusion\_matrix (y\_test,RF\_pred))

**Logistic Regression**

from sklearn.linear\_model import LogisticRegression

logmodel = LogisticRegression()

logmodel.fit(x\_train, y\_train)

log\_pred = logmodel.predict(x\_test)

print (log\_pred)

print (pd.DataFrame(log\_pred).groupby(0).agg({0: np.size}))

print('\n Accuracy Score')

print (accuracy\_score (y\_test,log\_pred))

print('\nClassification Report')

print(classification\_report (y\_test,log\_pred))

print('Confusion Matrix')

print (confusion\_matrix (y\_test,log\_pred))