**Report**

**Introduction**

A credit scoring model is designed to assess the creditworthiness of individuals or entities, predicting their likelihood of defaulting on financial obligations. This report outlines the development of a credit scoring model, covering setup, data preparation, feature engineering, model development, evaluation, and tuning.

**Prerequisite –**

* Basics of Phyton language
* Installation of python on the system
* Dataset (to work upon)
* Installation of following packages:-

1. Pandas
2. Numpy
3. Scikit

Note- install the packages in the windows terminal by using the following syntax :-

py -m pip install package\_name

and in macos use the following syntax :-

python3 pip install package\_name

I have made the project using 9 basic steps:-

Step 1- importing necessary libraries

import pandas as pd

import numpy as np

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.preprocessing import LabelEncoder

the libraries used in the project and their uses are-

* Pandas – for data manipulation and analysis
* Numpy – for numerical operation
* Train\_test\_split – to split dataset into training and testing datasets
* RandomForestClassifier – ensemble method for classification
* Accuracy\_score, confusion\_matrix, classification\_report – evaluating model’s performance
* lableEncoder – convert categorical data into numerical format

Step 2- loading and inspecting the dataset

data = pd.read\_csv('credit\_data.csv')

print(data.head())

if data.empty:

    print("dataset is empty")

else:

    print(data.isnull().sum())

* pd.read\_csv – loads the dataset from the csv file (provided in the reposisty with the name ‘cedit\_data.csv’)
* data.head() – display the first few rows of the dataset
* data.isnull().sum() – checks for the missing value in the dataset

Step 3 – handling missing values

numeric\_columns = data.select\_dtypes(include= [np.number]).columns

data[numeric\_columns] = data[numeric\_columns].fillna(data[numeric\_columns].mean())

* select\_dtypes – selects columns with numeric datatype
* fillna(data[numeric\_columns].mean()) – fill the missing value with the mean of those columns

Step 4 – encoding categorical variables

categorical\_columns = data.select\_dtypes(include=[object]).columns

label\_encoders = {}

for col in categorical\_columns:

    le = LabelEncoder()

    data[col] = le.fit\_transform(data[col])

    label\_encoders[col] = le

* select\_dtypes(include=[object]) – select columns with categorical (object) data type
* LabelEncoder – encodes categorical data into numerical data
* Fit\_transfrom – fits the encoder to transform the categorical value to numerical value

Step 5 – defining features and target variable

x = data.drop('credit\_history', axis=1)

y = data['credit\_history']

* X = features and all columns except the target column
* Y = target column

Step 6- splitting the dataset

try:

    x\_train, x\_test, y\_train, y\_test=train\_test\_split(x,y,test\_size=0.3, random\_state=42)

    print("data split successfully!")

except Exception as e:

    print(f"error during data split: {e}")

* Train\_test\_split- splits data into training and testing sets. Here ‘test\_size=0.3’ indicates that the 30%data will be used for testing and 70% for training.

Step 7 – initializing and training the random forest classifier

clf = RandomForestClassifier(n\_estimators=100, random\_state=42)

try:

    clf.fit(x\_train, y\_train)

except ValueError as e:

    print(f"error during model training:{e}")

* RandomForestClassifier- initialise the classifier with 100 trees (n\_estimators=100)
* Fit- train the model on the training data

Step 8 – making predictions

try:

    y\_pred = clf.predict(x\_test)

except  ValueError as e:

    print(f"error during prediction:{e}")

* Predict- uses the train model to make prediction on the test data

Step 9 – evaluating the model

try:

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

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

    class\_report = classification\_report(y\_test, y\_pred)

    print(f'accuracy: {accuracy}')

    print('confusion matrix:')

    print(conf\_matrix)

    print('classification report:')

    print(class\_report)

except ValueError as e:

    print(f"error during evaluation:{e}")

* Accuracy\_score- calculates the accuracy of the model
* Confusion\_matrix – visualize the performance of classification
* Classification\_report- generate a report with prediction

This is all that I used in the code and their respective functions.