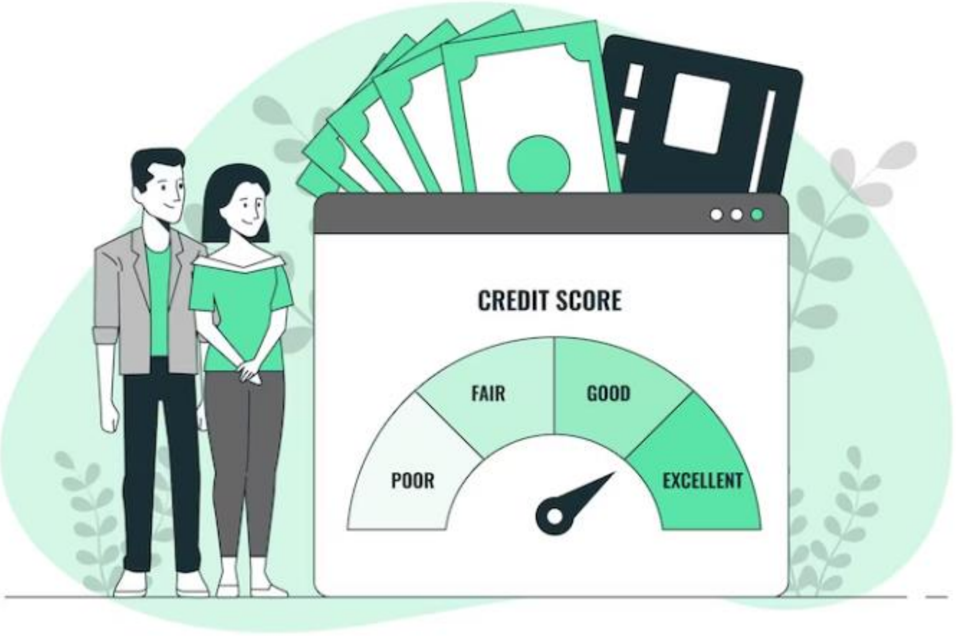
Classification - Credit Score



Agenda

01 Importing the Libraries

02 Loading the Data

03 Data Cleaning

04 One Hot Encoding

05 Feature Selection

06 Implementing ML Algorithms

Problem Statement

You are working as a data scientist in a global finance company. Over the years, the company has collected basic bank details and gathered a lot of credit-related information. The management wants to build an intelligent system to segregate the people into credit score brackets to reduce the manual efforts. Given a person's credit-related information, build a machine learning model that can classify the credit score.

Dataset Information

Credit score dataset contains 1 lac records with 28 features.

Dataset Information

Attributes	Description
ID	unique identification of an entry
Customer_ID	unique identification of a person
Month	month of the year
Name	name of a person
Age	age of the person
SSN	social security number of a person
Occupation	occupation of the person
Annual_Income	annual income of the person
Monthly_Inhand_Salary	monthly base salary of a person
Num_Bank_Accounts	number of bank accounts a person holds
Num_Credit_Card	number of other credit cards held by a person

Dataset Information

Attributes	Description
Interest_Rate	interest rate on credit card
Num_of_Loan	number of loans taken from the bank
Type_of_Loan	types of loan taken by a person
Delay_from_due_date	average number of days delayed from the payment date
Num_of_Delayed_Payment	age of the person
Changed_Credit_Limit	percentage change in credit card limit
Num_Credit_Inquiries	number of credit card inquiries
Credit_Mix	classification of the mix of credits
Outstanding_Debt	remaining debt to be paid (in USD)
Credit_Utilization_Ratio	utilization ratio of credit card
Credit_History_Age	the age of credit history of the person

Dataset Information

Attributes	Description
Payment_of_Min_Amount	the minimum amount was paid by the person
Total_EMI_per_month	monthly EMI payments (in USD)
Amount_Invested_monthly	monthly amount invested by the customer (in USD)
Payment_Behaviour	payment behavior of the customer (in USD)
Monthly_Balance	monthly balance amount of the customer (in USD)
Credit_Score	the bracket of credit score

Importing the Libraries

We start off this project by importing all the necessary libraries that will be required for the process.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

Loading the Data

Loading the data and removing unnecessary column from the dataframe

```
df=pd.read_csv("credit_score.csv")
df=df.drop(columns=["ID","Customer_ID","Name","SSN","Type_of_Loan","Credit_History_Age"])
df.head()
```

	Month	Age	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Num_of_Loan	Delay_from_due_date	...
0	January	23	Scientist	19114.12	1824.843333	3	4	3	4	3	...
1	February	23	Scientist	19114.12	NaN	3	4	3	4	-1	...
2	March	-500	Scientist	19114.12	NaN	3	4	3	4	3	...
3	April	23	Scientist	19114.12	NaN	3	4	3	4	5	...
4	May	23	Scientist	19114.12	1824.843333	3	4	3	4	6	...

5 rows × 22 columns

Loading the Data

Checking the shape of a dataframe and datatypes of all columns along with calculating the statistical data.

```
df.shape
df.info()
df.describe()
```

```
(100000, 22)
```

	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Delay_from_due_date	Num_Credit_Inquiries
count	84998.000000	100000.000000	100000.00000	100000.000000	100000.000000	98035.000000
mean	4194.170850	17.091280	22.47443	72.466040	21.068780	27.754251
std	3183.686167	117.404834	129.05741	466.422621	14.860104	193.177339
min	303.645417	-1.000000	0.00000	1.000000	-5.000000	0.000000
25%	1625.568229	3.000000	4.00000	8.000000	10.000000	3.000000
50%	3093.745000	6.000000	5.00000	13.000000	18.000000	6.000000
75%	5957.448333	7.000000	7.00000	20.000000	28.000000	9.000000
max	15204.633333	1798.000000	1499.00000	5797.000000	67.000000	2597.000000

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 22 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   Month                     100000 non-null  object 
 1   Age                       100000 non-null  object 
 2   Occupation                100000 non-null  object 
 3   Annual_Income             100000 non-null  object 
 4   Monthly_Inhand_Salary     84998 non-null   float64
 5   Num_Bank_Accounts         100000 non-null  int64  
 6   Num_Credit_Card           100000 non-null  int64  
 7   Interest_Rate             100000 non-null  int64  
 8   Num_of_Loan               100000 non-null  object 
 9   Delay_from_due_date       100000 non-null  int64  
 10  Num_of_Delayed_Payment    92998 non-null   object 
 11  Changed_Credit_Limit      100000 non-null  object 
 12  Num_Credit_Inquiries      98035 non-null   float64
 13  Credit_Mix                100000 non-null  object 
 14  Outstanding_Debt          100000 non-null  object 
 15  Credit_Utilization_Ratio  100000 non-null  float64
 16  Payment_of_Min_Amount     100000 non-null  object 
 17  Total_EMI_per_month       100000 non-null  float64
 18  Amount_invested_monthly   95521 non-null   object 
 19  Payment_Behaviour         100000 non-null  object 
 20  Monthly_Balance           98800 non-null   object 
 21  Credit_Score              100000 non-null  object 
dtypes: float64(4), int64(4), object(14)
memory usage: 16.8+ MB
```

Missing Values

Checking out the missing values in a dataframe

```
df.isnull().sum()
```

```
Month                          0
Age                            0
Occupation                     0
Annual_Income                  0
Monthly_Inhand_Salary      15002
Num_Bank_Accounts              0
Num_Credit_Card                0
Interest_Rate                  0
Num_of_Loan                    0
Delay_from_due_date            0
Num_of_Delayed_Payment      7002
Changed_Credit_Limit           0
Num_Credit_Inquiries        1965
Credit_Mix                     0
Outstanding_Debt               0
Credit_Utilization_Ratio       0
Payment_of_Min_Amount          0
Total_EMI_per_month            0
Amount_invested_monthly     4479
Payment_Behaviour              0
Monthly_Balance             1200
Credit_Score                   0
dtype: int64
```

Data Cleaning

```
df["Age"]=df["Age"].str.replace("_","")
df["Age"]=df["Age"].astype(int)
df["Occupation"]=df["Occupation"].replace("_______",np.nan)
df["Annual_Income"]=df["Annual_Income"].str.replace("_","")
df["Annual_Income"]=df["Annual_Income"].astype(float)
df["Num_of_Loan"]=df["Num_of_Loan"].str.replace("_","")
df["Num_of_Loan"]=df["Num_of_Loan"].astype(int)
df["Num_of_Delayed_Payment"]=df["Num_of_Delayed_Payment"].str.replace("_","")
df["Num_of_Delayed_Payment"]=df["Num_of_Delayed_Payment"].astype(float)
df["Credit_Score"]=df["Credit_Score"].replace(["Poor","Standard","Good"],[0,1,2])
df["Monthly_Balance"]=df["Monthly_Balance"].str.replace("_","")
df["Monthly_Balance"]=df["Monthly_Balance"].astype(float)
df["Payment_Behaviour"]=df["Payment_Behaviour"].replace("!@9#%8",np.nan)
df["Amount_invested_monthly"]=df["Amount_invested_monthly"].str.replace("_","")
df["Amount_invested_monthly"]=df["Amount_invested_monthly"].astype(float)
df["Payment_of_Min_Amount"]=df["Payment_of_Min_Amount"].replace("NM","No")
df["Payment_of_Min_Amount"]=df["Payment_of_Min_Amount"].replace(["Yes","No"],[1,0])
df["Outstanding_Debt"]= df["Outstanding_Debt"].str.replace("_","")
df["Outstanding_Debt"]=df["Outstanding_Debt"].astype(float)
df["Credit_Mix"]=df["Credit_Mix"].replace("_",np.nan)
df["Credit_Mix"]=df["Credit_Mix"].replace(["Standard",'Good',"Bad"],[1,2,0])
df["Changed_Credit_Limit"]= df["Changed_Credit_Limit"].replace("_",np.nan)
df["Changed_Credit_Limit"]=df["Changed_Credit_Limit"].astype(float)
```

Replacing the special characters with empty string or with null values according to the data and converting it into int or float datatype. Also, Converting the categorical values of some columns into integer values.

Data Cleaning

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 22 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   Month                     100000 non-null  object 
 1   Age                       100000 non-null  int64  
 2   Occupation                92938 non-null   object 
 3   Annual_Income             100000 non-null  float64
 4   Monthly_Inhand_Salary     84998 non-null   float64
 5   Num_Bank_Accounts         100000 non-null  int64  
 6   Num_Credit_Card           100000 non-null  int64  
 7   Interest_Rate             100000 non-null  int64  
 8   Num_of_Loan               100000 non-null  int64  
 9   Delay_from_due_date       100000 non-null  int64  
 10  Num_of_Delayed_Payment    92998 non-null   float64
 11  Changed_Credit_Limit      97909 non-null   float64
 12  Num_Credit_Inquiries      98035 non-null   float64
 13  Credit_Mix                79805 non-null   float64
 14  Outstanding_Debt          100000 non-null  float64
 15  Credit_Utilization_Ratio  100000 non-null  float64
 16  Payment_of_Min_Amount     100000 non-null  int64  
 17  Total_EMI_per_month       100000 non-null  float64
 18  Amount_invested_monthly   95521 non-null   float64
 19  Payment_Behaviour         92400 non-null   object 
 20  Monthly_Balance           97132 non-null   float64
 21  Credit_Score              100000 non-null  int64  
dtypes: float64(11), int64(8), object(3)
memory usage: 16.8+ MB
```

Clearly, The datatype of the columns have been changed after performing the operation

Data Cleaning

```
df.isnull().sum()
df=df.fillna(method="ffill")
df=df.fillna(method="bfill")
df.isnull().sum()
```

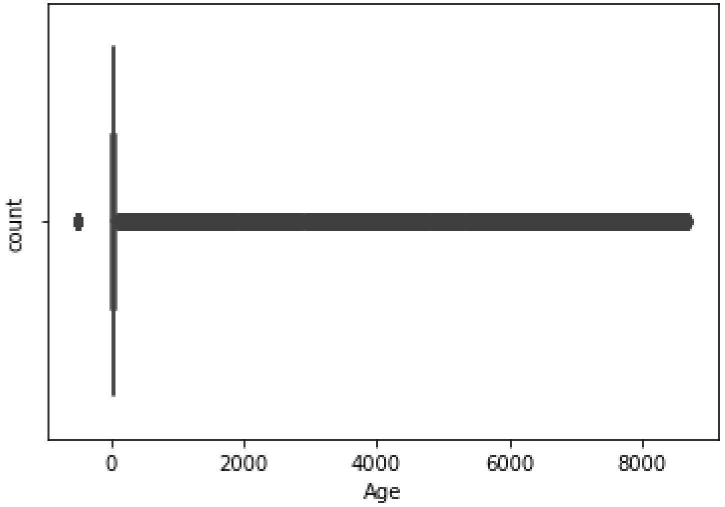
```
Month                        0
Age                          0
Occupation                7062
Annual_Income                0
Monthly_Inhand_Salary    15002
Num_Bank_Accounts            0
Num_Credit_Card              0
Interest_Rate                0
Num_of_Loan                  0
Delay_from_due_date          0
Num_of_Delayed_Payment    7002
Changed_Credit_Limit      2091
Num_Credit_Inquiries      1965
Credit_Mix               20195
Outstanding_Debt             0
Credit_Utilization_Ratio     0
Payment_of_Min_Amount        0
Total_EMI_per_month          0
Amount_invested_monthly   4479
Payment_Behaviour         7600
Monthly_Balance           2868
Credit_Score                 0
dtype: int64
```

```
Month                       0
Age                         0
Occupation                  0
Annual_Income               0
Monthly_Inhand_Salary       0
Num_Bank_Accounts           0
Num_Credit_Card             0
Interest_Rate               0
Num_of_Loan                 0
Delay_from_due_date         0
Num_of_Delayed_Payment      0
Changed_Credit_Limit        0
Num_Credit_Inquiries        0
Credit_Mix                  0
Outstanding_Debt            0
Credit_Utilization_Ratio    0
Payment_of_Min_Amount       0
Total_EMI_per_month         0
Amount_invested_monthly     0
Payment_Behaviour           0
Monthly_Balance             0
Credit_Score                0
dtype: int64
```

After replacing the special characters with null value. The new missing value is shown in the figure. Here Forward and backward filling method is used to fill the missing values.

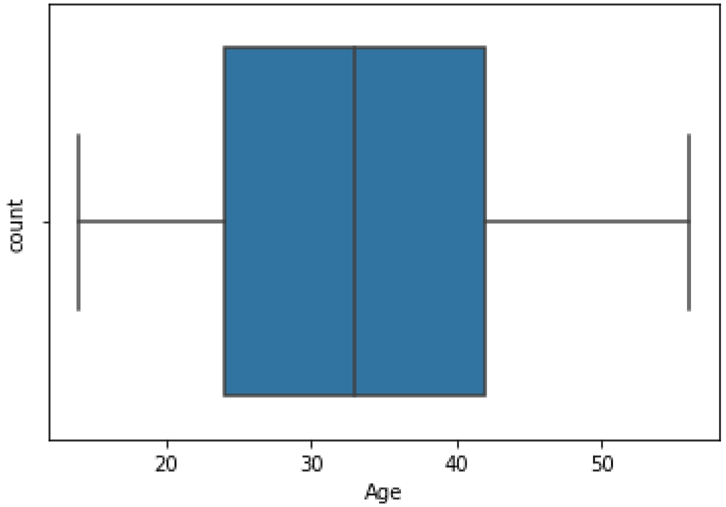
Data Cleaning

```
sns.boxplot(df["Age"])
plt.xlabel("Age")
plt.ylabel('count')
plt.show()
```



removing outliers from age since all other columns values are relevant

```
col_names=["Age"]
Q1 = df.Age.quantile(0.25)
Q3 = df.Age.quantile(0.75)
IQR = Q3 - Q1
data = df[(df.Age >= Q1 - 1.5*IQR) & (df.Age <= Q3 + 1.5*IQR)]
sns.boxplot(data["Age"])
plt.xlabel("Age")
plt.ylabel('count')
plt.show()
```



Data Cleaning

```
from sklearn.preprocessing import LabelEncoder
le= LabelEncoder()
df["Month"]=le.fit_transform(df["Month"])
df["Occupation"]=le.fit_transform(df["Occupation"])
df["Payment_Behaviour"]=le.fit_transform(df["Payment_Behaviour"])
df.info()
```

Performing One Hot Encoding for categorical features of a dataframe

```
Column                    Non-Null Count   Dtype
------                    --------------   -----
Month                     100000 non-null  int64
Age                       100000 non-null  int64
Occupation                100000 non-null  int64
Annual_Income             100000 non-null  float64
Monthly_Inhand_Salary     100000 non-null  float64
Num_Bank_Accounts         100000 non-null  int64
Num_Credit_Card           100000 non-null  int64
Interest_Rate             100000 non-null  int64
Num_of_Loan               100000 non-null  int64
Delay_from_due_date       100000 non-null  int64
Num_of_Delayed_Payment    100000 non-null  float64
Changed_Credit_Limit      100000 non-null  float64
Num_Credit_Inquiries      100000 non-null  float64
Credit_Mix                100000 non-null  float64
Outstanding_Debt          100000 non-null  float64
Credit_Utilization_Ratio  100000 non-null  float64
Payment_of_Min_Amount     100000 non-null  int64
Total_EMI_per_month       100000 non-null  float64
Amount_invested_monthly   100000 non-null  float64
Payment_Behaviour         100000 non-null  int64
Monthly_Balance           100000 non-null  float64
Credit_Score              100000 non-null  int64
```

Feature Selection

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
col_list = []
for col in df.columns:
    if ((df[col].dtype != 'object') & (col != 'Credit_Score') ):
        col_list.append(col)

X = df[col_list]
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                          for i in range(len(X.columns))]
print(vif_data)
```

Selecting the features using VIF. VIF should be less than 5. Here, all features have VIF value less than 5, So we will select all the features.

feature	VIF
Month	0.300012
Age	0.974661
Occupation	0.277722
Annual_Income	0.985001
Monthly_Inhand_Salary	0.365970
Num_Bank_Accounts	0.979247
Num_Credit_Card	0.970567
Interest_Rate	0.976430
Num_of_Loan	0.997697
Delay_from_due_date	0.332213
Num_of_Delayed_Payment	0.981707
Changed_Credit_Limit	0.299307
Num_Credit_Inquiries	0.979793
Credit_Mix	0.321474
Outstanding_Debt	0.396141
Credit_Utilization_Ratio	0.024506
Payment_of_Min_Amount	0.476749
Total_EMI_per_month	0.972258
Amount_invested_monthly	0.911321
Payment_Behaviour	0.310525
Monthly_Balance	1.000207

Logistic Regression

The accuracy of the logistic regression model is 61.8 percentage

```
X=df.drop(columns=["Credit_Score"])
y=df["Credit_Score"]
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =train_test_split(X,y,test_size=0.2,random_state=42)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train= sc.fit_transform(x_train)
x_test= sc.transform(x_test)
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(x_train,y_train)
y_pred=lr.predict(x_test)
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
pd.DataFrame({"actual_value":y_test,"predicted_value":y_pred})
```

```
0.61855
```

	actual_value	predicted_value
75721	2	2
80184	0	0
19864	2	2
76699	0	0
92991	2	2
...	...	...
32595	1	1
29313	1	1
37862	0	1
53421	1	1
42410	1	1

20000 rows × 2 columns

Decision Tree

The accuracy of the decision tree model is 69.7 percentage

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
y_pred= dt.predict(x_test)
accuracy_score(y_test,y_pred)
pd.DataFrame({"actual_value":y_test,'predicted_value':y_pred})
```

```
0.6977
```

	actual_value	predicted_value
75721	2	2
80184	0	0
19864	2	2
76699	0	0
92991	2	2
...	...	...
32595	1	1
29313	1	1
37862	0	1
53421	1	1
42410	1	0

20000 rows × 2 columns

Hyperparameter Tuning on Decision Tree

```
from sklearn.model_selection import GridSearchCV
parameters = {'max_features': ['log2', 'sqrt','auto'],
              'criterion': ['entropy', 'gini'],
              'max_depth': [2, 3, 5, 10, 50],
              'min_samples_split': [2, 3, 50, 100],
              'min_samples_leaf': [1, 5, 8, 10]
             }

grid_obj = GridSearchCV(dt, parameters)
grid_obj = grid_obj.fit(x_train, y_train)
dt = grid_obj.best_estimator_
dt.fit(x_train,y_train)
y_pred = dt.predict(x_test)
acc_dt = round(accuracy_score(y_test, y_pred) * 100, 2 )
print( 'Accuracy of Decision Tree model : ', acc_dt )
```

```
Accuracy of Decision Tree model :  70.93
```

Here, We are using GridSearch CV technique which is used to identify the optimal hyperparameters for a model and the accuracy obtained from Decision Tree is 70.93

Random Forest

The accuracy of the random forest model is
79.7 percentage

```
from sklearn.ensemble import RandomForestClassifier
rfc= RandomForestClassifier()
rfc.fit(x_train,y_train)
y_pred=rfc.predict(x_test)
accuracy_score(y_test,y_pred)
pd.DataFrame({"Actual_Value":y_test,"Predicted_Value":y_pred})
```

```
0.7974
```

	Actual_Value	Predicted_Value
75721	2	2
80184	0	0
19864	2	2
76699	0	0
92991	2	2
...	...	...
32595	1	1
29313	1	1
37862	0	1
53421	1	1
42410	1	0

20000 rows × 2 columns