CREDIT SCORE CLASSIFICATION 📈

Introduction

🌐 In today's financial landscape, credit scores play a pivotal role in assessing an individual's creditworthiness. Lenders, ranging from traditional banks to online lending platforms, heavily rely on credit scores to make informed decisions about extending credit. As the demand for credit continues to rise, the need for accurate and efficient credit score classification becomes paramount. Machine learning techniques offer a powerful toolset for analyzing vast amounts of financial data to predict and classify credit scores.

Project Statement

🛠 This project aims to explore and implement machine learning algorithms for credit score classification. By leveraging historical credit data, the objective is to train models that can accurately predict credit scores based on various financial and non-financial features.

🤖 The project will delve into feature engineering, model selection, and performance evaluation to create a robust and reliable credit scoring system. The ultimate goal is to contribute to the enhancement of credit risk assessment methodologies, providing financial institutions with more precise tools for evaluating potential borrowers. Through this exploration of machine learning in credit scoring, we aim to contribute to the ongoing evolution of the financial industry and promote more efficient and equitable lending practices 💳

About the Dataset

Dataset Size

1. train.csv - 100000 rows
2. test.csv - 50000 rows

Columns

- ID: Unique identifier for each record in the dataset.
- Customer_ID: Unique identifier for each customer.
- Month: The month for which the financial data is recorded.
- Name: Name of the individual.
- Age: Age of the individual.
- SSN: Social Security Number, a unique identifier for individuals in the U.S.
- Occupation: The occupation or profession of the individual.
- Annual_Income: Annual income of the individual.
- Monthly_Inhand_Salary: Net monthly salary after deductions.
- Num_Bank_Accounts: Number of bank accounts held by the individual.
- Num_Credit_Card: Number of credit cards owned by the individual.
- Interest_Rate: Interest rate associated with financial transactions.
- Num_of_Loan: Number of loans the individual has.
- Type_of_Loan: The type of loan(s) the individual has.
- Delay_from_due_date: Delay in payments from the due date.
- Num_of_Delayed_Payment: Number of delayed payments.
- Changed_Credit_Limit: Whether there has been a change in credit limit.
- Num_Credit_Inquiries: Number of credit inquiries made.
- Credit_Mix: The mix of different types of credit.
- Outstanding_Debt: Amount of outstanding debt.
- Credit_Utilization_Ratio: Ratio of credit used to the total credit available.
- Credit_History_Age: Age of credit history.
- Payment_of_Min_Amount: Payment behavior regarding the minimum amount due.
- Total_EMI_per_month: Total Equated Monthly Installment (EMI) payments.
- Amount_invested_monthly: Amount invested by the individual monthly.
- Payment_Behaviour: Behavior related to payment patterns.
- Monthly_Balance: Monthly balance in the account.
- Credit_Score: The credit score assigned to the individual based on various factors.

Data Pre-Processing

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

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
classification_report, confusion_matrix
```

```
df = pd.read_csv("train.csv",low_memory=False)
```

```
df.head()
```

```
       ID Customer_ID     Month           Name   Age          SSN Occupation  \
0  0x1602   CUS_0xd40   January  Aaron Maashoh    23  821-00-0265  Scientist
1  0x1603   CUS_0xd40  February  Aaron Maashoh    23  821-00-0265  Scientist
2  0x1604   CUS_0xd40     March  Aaron Maashoh  -500  821-00-0265  Scientist
3  0x1605   CUS_0xd40     April  Aaron Maashoh    23  821-00-0265  Scientist
4  0x1606   CUS_0xd40       May  Aaron Maashoh    23  821-00-0265  Scientist

  Annual_Income  Monthly_Inhand_Salary  Num_Bank_Accounts  ...  Credit_Mix  \
0      19114.12            1824.843333                  3  ...           _
1      19114.12                    NaN                  3  ...        Good
2      19114.12                    NaN                  3  ...        Good
3      19114.12                    NaN                  3  ...        Good
4      19114.12            1824.843333                  3  ...        Good

   Outstanding_Debt Credit_Utilization_Ratio     Credit_History_Age  \
0            809.98                26.822620  22 Years and 1 Months
1            809.98                31.944960                    NaN
2            809.98                28.609352  22 Years and 3 Months
3            809.98                31.377862  22 Years and 4 Months
4            809.98                24.797347  22 Years and 5 Months

   Payment_of_Min_Amount Total_EMI_per_month Amount_invested_monthly  \
0                     No           49.574949       80.41529543900253
1                     No           49.574949      118.28022162236736
2                     No           49.574949         81.699521264648
3                     No           49.574949       199.4580743910713
4                     No           49.574949      41.420153086217326

                 Payment_Behaviour     Monthly_Balance Credit_Score
0   High_spent_Small_value_payments  312.49408867943663         Good
1    Low_spent_Large_value_payments  284.62916249607184         Good
2   Low_spent_Medium_value_payments   331.2098628537912         Good
3    Low_spent_Small_value_payments  223.45130972736786         Good
4  High_spent_Medium_value_payments  341.48923103222177         Good

[5 rows x 28 columns]
```

```python
print('Train Data Size : ', df.shape)
```

```
Train Data Size :  (100000, 28)
```

```python
df.columns
```

```
Index(['ID', 'Customer_ID', 'Month', 'Name', 'Age', 'SSN', 'Occupation',
       'Annual_Income', 'Monthly_Inhand_Salary', 'Num_Bank_Accounts',
       'Num_Credit_Card', 'Interest_Rate', 'Num_of_Loan', 'Type_of_Loan',
       'Delay_from_due_date', 'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
       'Num_Credit_Inquiries', 'Credit_Mix', 'Outstanding_Debt',
       'Credit_Utilization_Ratio', 'Credit_History_Age',
       'Payment_of_Min_Amount', 'Total_EMI_per_month',
       'Amount_invested_monthly', 'Payment_Behaviour', 'Monthly_Balance',
       'Credit_Score'],
      dtype='object')
```

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 28 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   ID                        100000 non-null  object 
 1   Customer_ID               100000 non-null  object 
 2   Month                     100000 non-null  object 
 3   Name                      90015 non-null   object 
 4   Age                       100000 non-null  object 
 5   SSN                       100000 non-null  object 
 6   Occupation                100000 non-null  object 
 7   Annual_Income             100000 non-null  object 
 8   Monthly_Inhand_Salary     84998 non-null   float64
 9   Num_Bank_Accounts         100000 non-null  int64  
 10  Num_Credit_Card           100000 non-null  int64  
 11  Interest_Rate             100000 non-null  int64  
 12  Num_of_Loan               100000 non-null  object 
 13  Type_of_Loan              88592 non-null   object 
 14  Delay_from_due_date       100000 non-null  int64  
 15  Num_of_Delayed_Payment    92998 non-null   object 
 16  Changed_Credit_Limit      100000 non-null  object 
 17  Num_Credit_Inquiries      98035 non-null   float64
 18  Credit_Mix                100000 non-null  object 
 19  Outstanding_Debt          100000 non-null  object 
 20  Credit_Utilization_Ratio  100000 non-null  float64
 21  Credit_History_Age        90970 non-null   object 
 22  Payment_of_Min_Amount     100000 non-null  object 
 23  Total_EMI_per_month       100000 non-null  float64
 24  Amount_invested_monthly   95521 non-null   object 
 25  Payment_Behaviour         100000 non-null  object 
 26  Monthly_Balance           98800 non-null   object 
 27  Credit_Score              100000 non-null  object 
dtypes: float64(4), int64(4), object(20)
memory usage: 21.4+ MB
```

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

```
ID                              0
Customer_ID                     0
Month                           0
Name                         9985
Age                             0
SSN                             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
Type_of_Loan                11408
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
Credit_History_Age           9030
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
```

Dataset consists of missing values.

```
# Drop unnecessary columns
df.drop(["ID","Customer_ID","Name","SSN","Type_of_Loan"],axis=1,inplace=True)
```

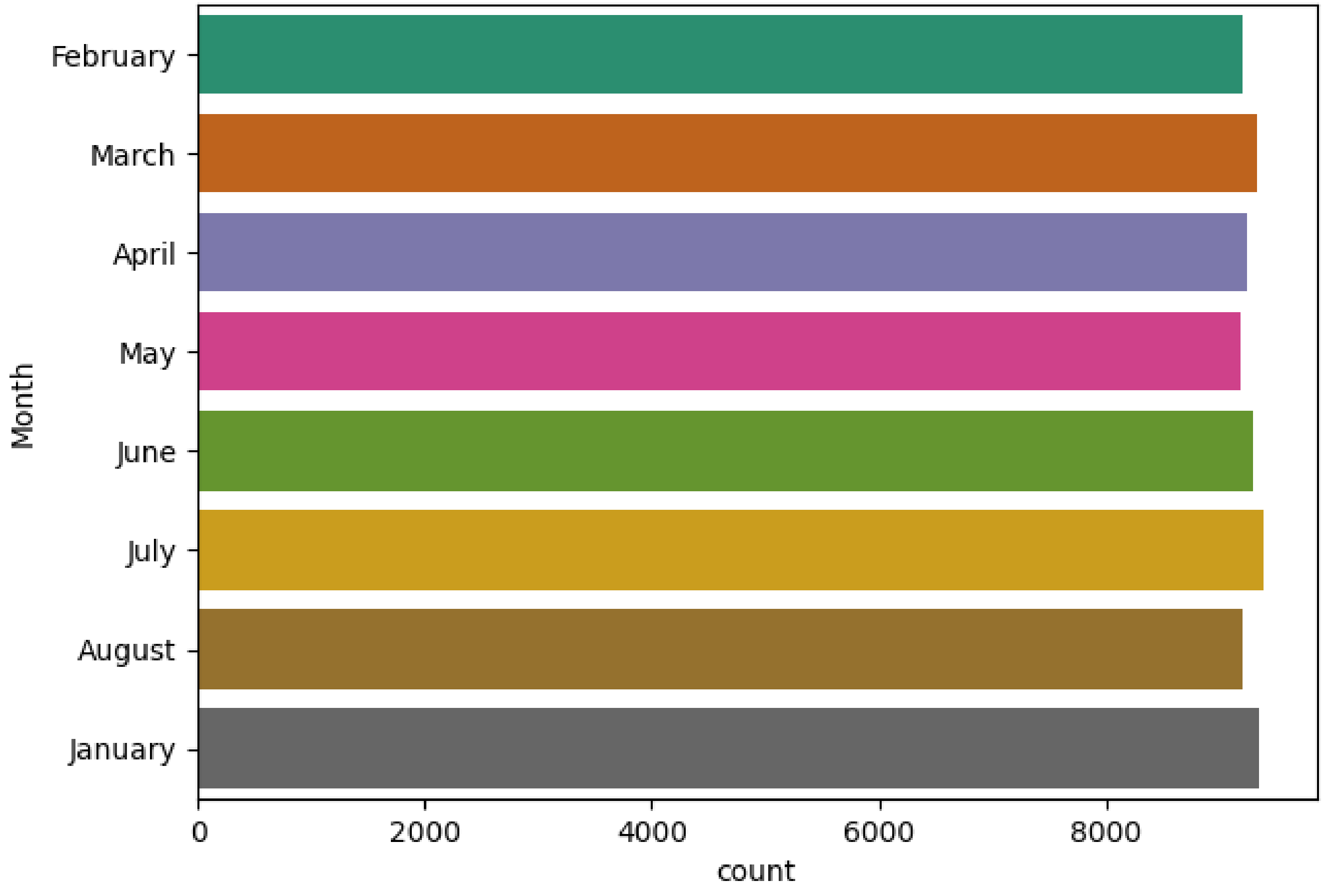
```
# Drop unnecessary row values

df.drop(df[df["Occupation"]=='_______'].index,inplace=True)
df.drop(df[df["Credit_Mix"]=='_'].index,inplace=True)
```

Data Encoding (Categorical ---> Numerical)

```
df["Month"].value_counts()
```

```
July        9377
January     9341
March       9325
June        9295
April       9237
February    9210
August      9198
May         9181
Name: Month, dtype: int64
```

```
plt.figure(figsize=(7,5))
sns.countplot(y="Month",data=df,palette="Dark2")
plt.show()
```

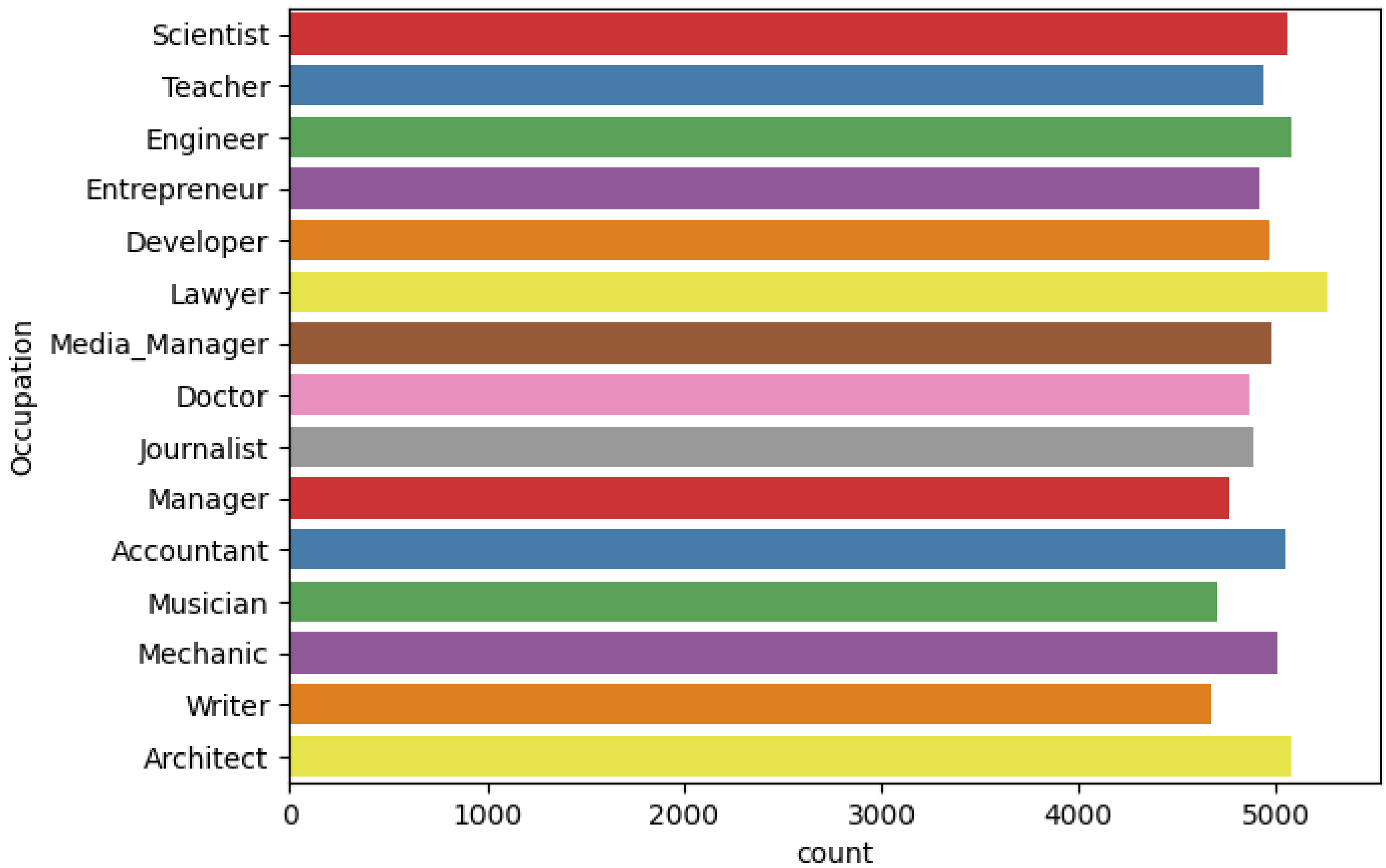



```
month_mapping = {
   'January': 1,
    'February': 2,
    'March': 3,
    "April":4,
    "May":5,
    "June":6,
    "July":7,
    "August":8}
df['Month'] = df['Month'].replace(month_mapping)
```

```
df["Occupation"].value_counts()
```

```
Lawyer           5259
Engineer         5077
Architect        5073
Scientist        5052
Accountant       5042
Mechanic         5001
Media_Manager    4978
Developer        4967
Teacher          4930
Entrepreneur     4911
Journalist       4884
Doctor           4860
Manager          4756
Musician         4702
Writer           4672
Name: Occupation, dtype: int64
```

```
plt.figure(figsize=(7,5))
sns.countplot(y="Occupation",data=df,palette="Set1")
plt.show()
```

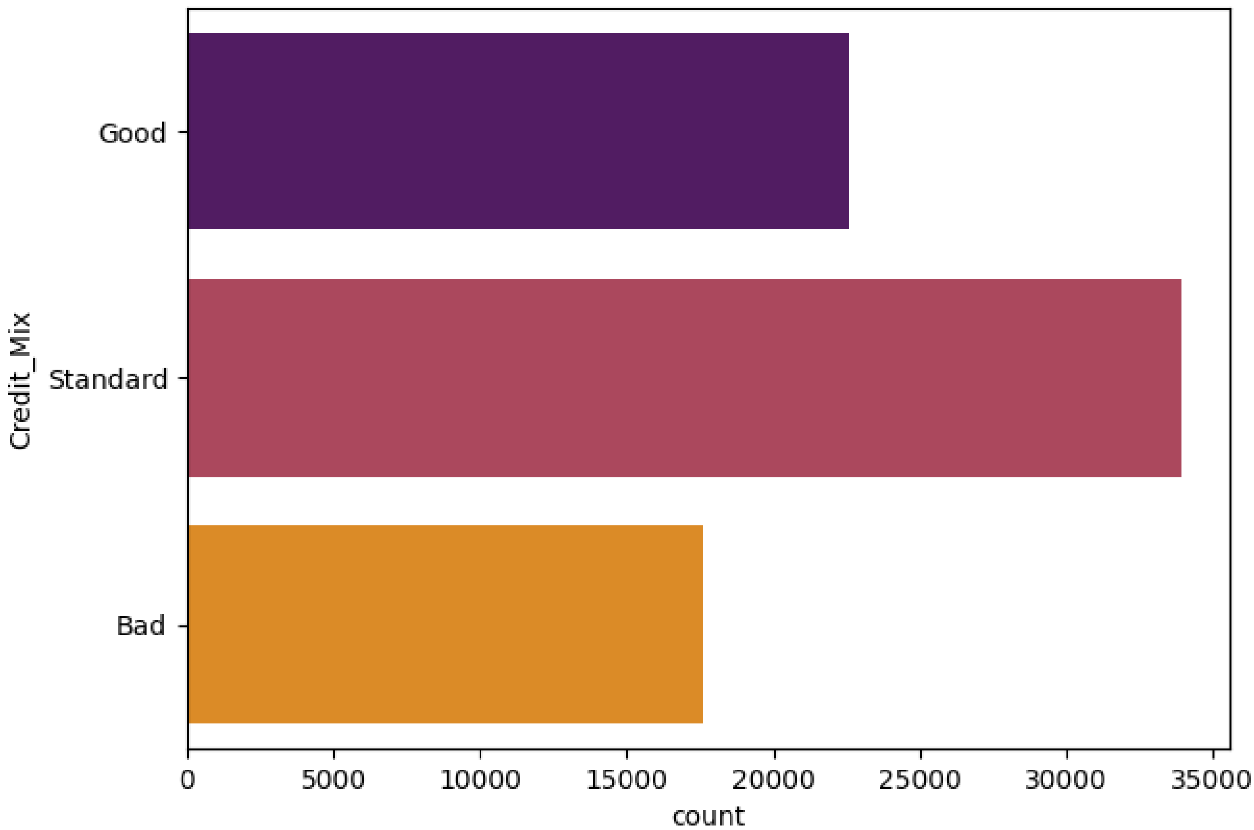


```
occupation_mapping = {
    'Lawyer': 1,
    'Architect': 2,
    'Engineer': 3,
    'Scientist': 4,
    'Mechanic': 5,
    'Accountant': 6,
    'Developer': 7,
    'Media_Manager': 8,
    'Teacher': 9,
    'Entrepreneur': 10,
    'Doctor': 11,
    'Journalist': 12,
    'Manager': 13,
    'Musician': 14,
    'Writer': 15
}
df['Occupation'] = df['Occupation'].replace(occupation_mapping)
```

```
df["Credit_Mix"].value_counts()
```

```
Standard    33916
Good        22618
Bad         17630
Name: Credit_Mix, dtype: int64
```

```
plt.figure(figsize=(7,5))
sns.countplot(y="Credit_Mix",data=df,palette="inferno")
plt.show()
```

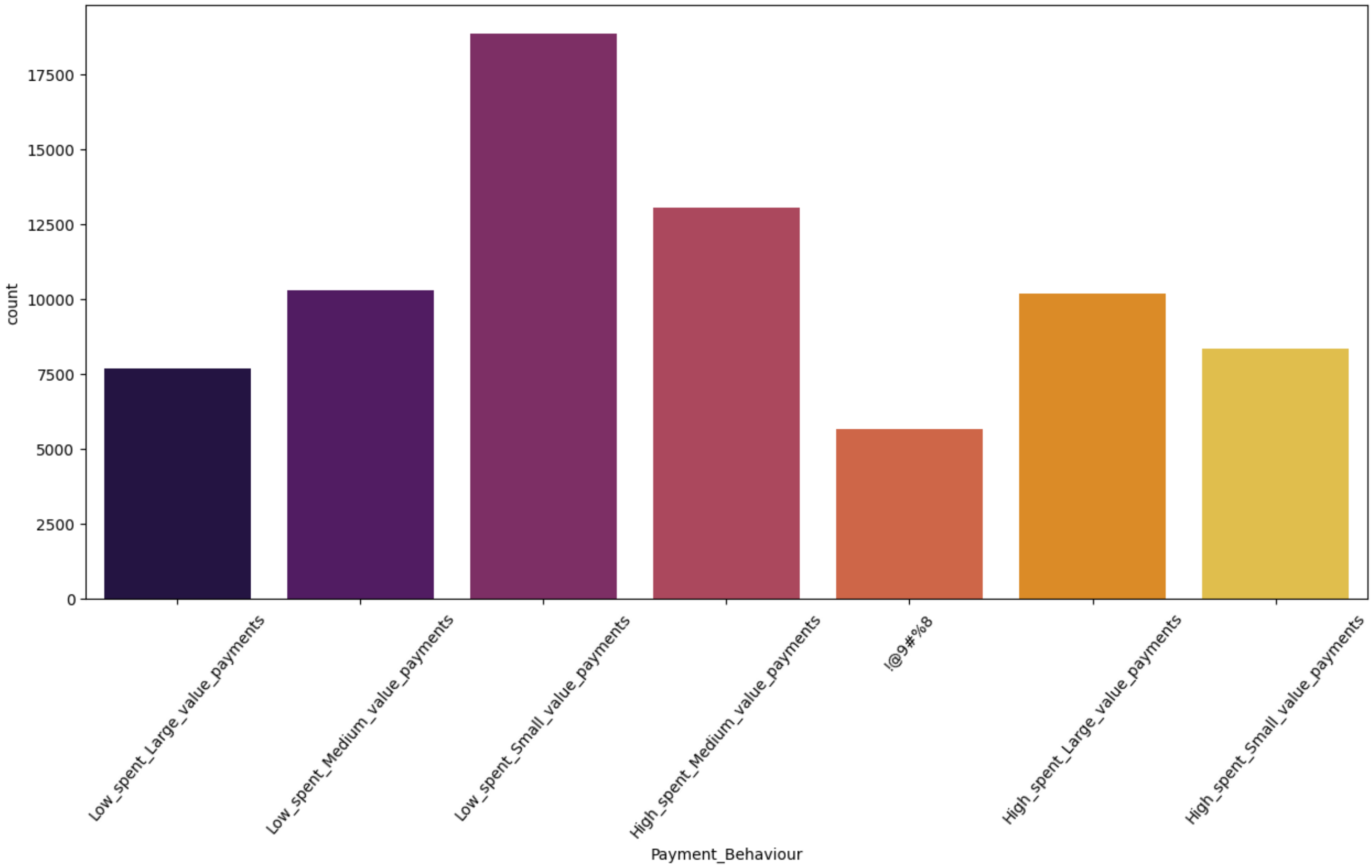



```
credit_map={"Good":1,"Standard":2,"Bad":3}
df['Credit_Mix'] = df['Credit_Mix'].replace(credit_map)
```

```
df["Payment_Behaviour"].value_counts()
```

```
Low_spent_Small_value_payments      18866
High_spent_Medium_value_payments    13075
Low_spent_Medium_value_payments     10304
High_spent_Large_value_payments     10191
High_spent_Small_value_payments      8341
Low_spent_Large_value_payments       7711
!@9#%8                               5676
Name: Payment_Behaviour, dtype: int64
```

```
plt.figure(figsize=(15,7))
sns.countplot(x="Payment_Behaviour",data=df,palette="inferno")
plt.xticks(rotation=50)
plt.show()
```

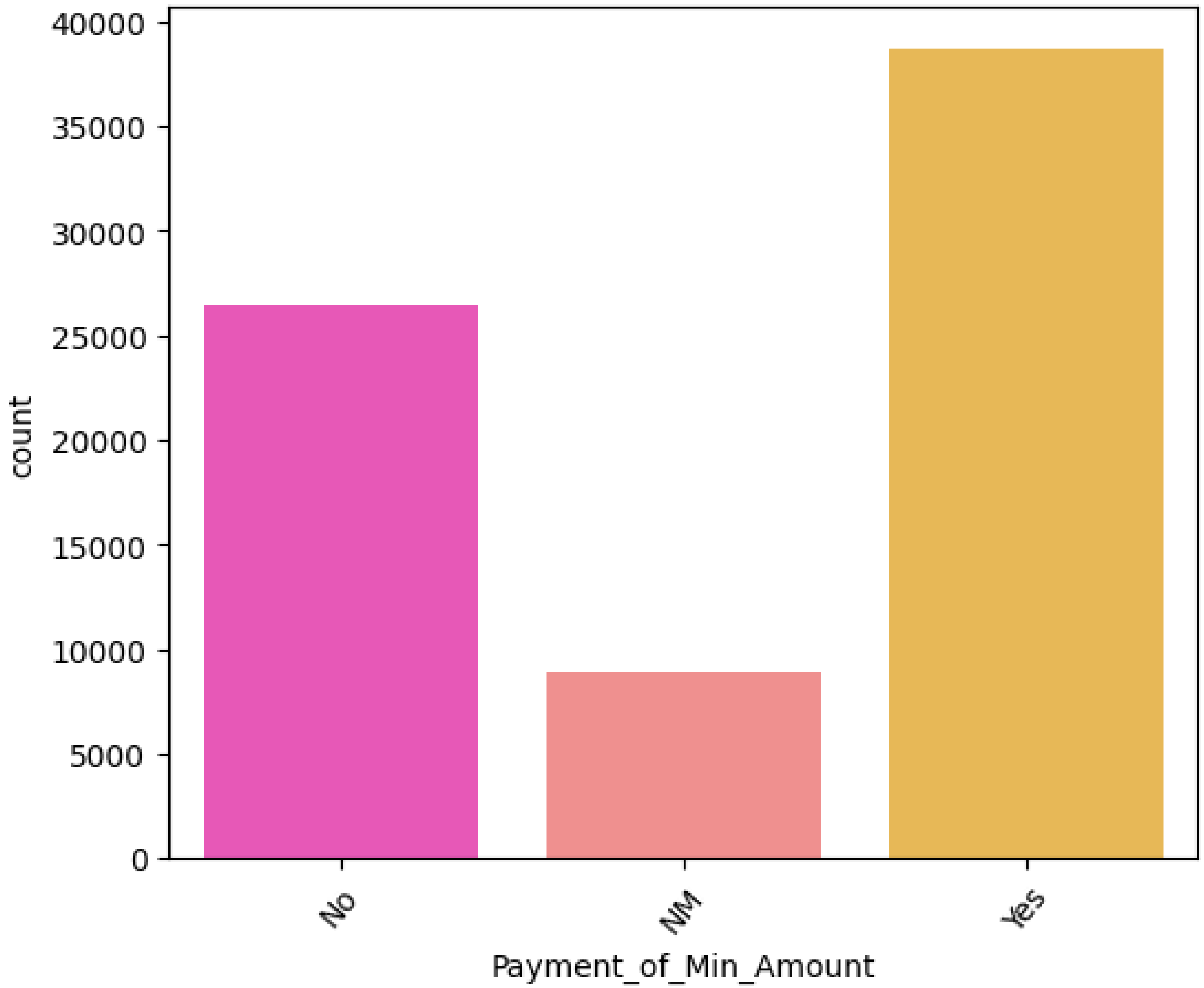



```
df['Payment_Behaviour']= df['Payment_Behaviour'].replace("!@9#%8",np.nan)

category_mapping = {
    'Low_spent_Small_value_payments':1,
    'High_spent_Medium_value_payments':2,
    'Low_spent_Medium_value_payments': 3,
    'High_spent_Large_value_payments': 4,
    'High_spent_Small_value_payments': 5,
    'Low_spent_Large_value_payments': 6
}
df['Payment_Behaviour'] = df['Payment_Behaviour'].replace(category_mapping)
```

```
df["Payment_of_Min_Amount"].value_counts()
```

```
Yes    38737
No     26501
NM      8926
Name: Payment_of_Min_Amount, dtype: int64
```

```
plt.figure(figsize=(6,5))
sns.countplot(x="Payment_of_Min_Amount",data=df,palette="spring")
plt.xticks(rotation=50)
plt.show()
```

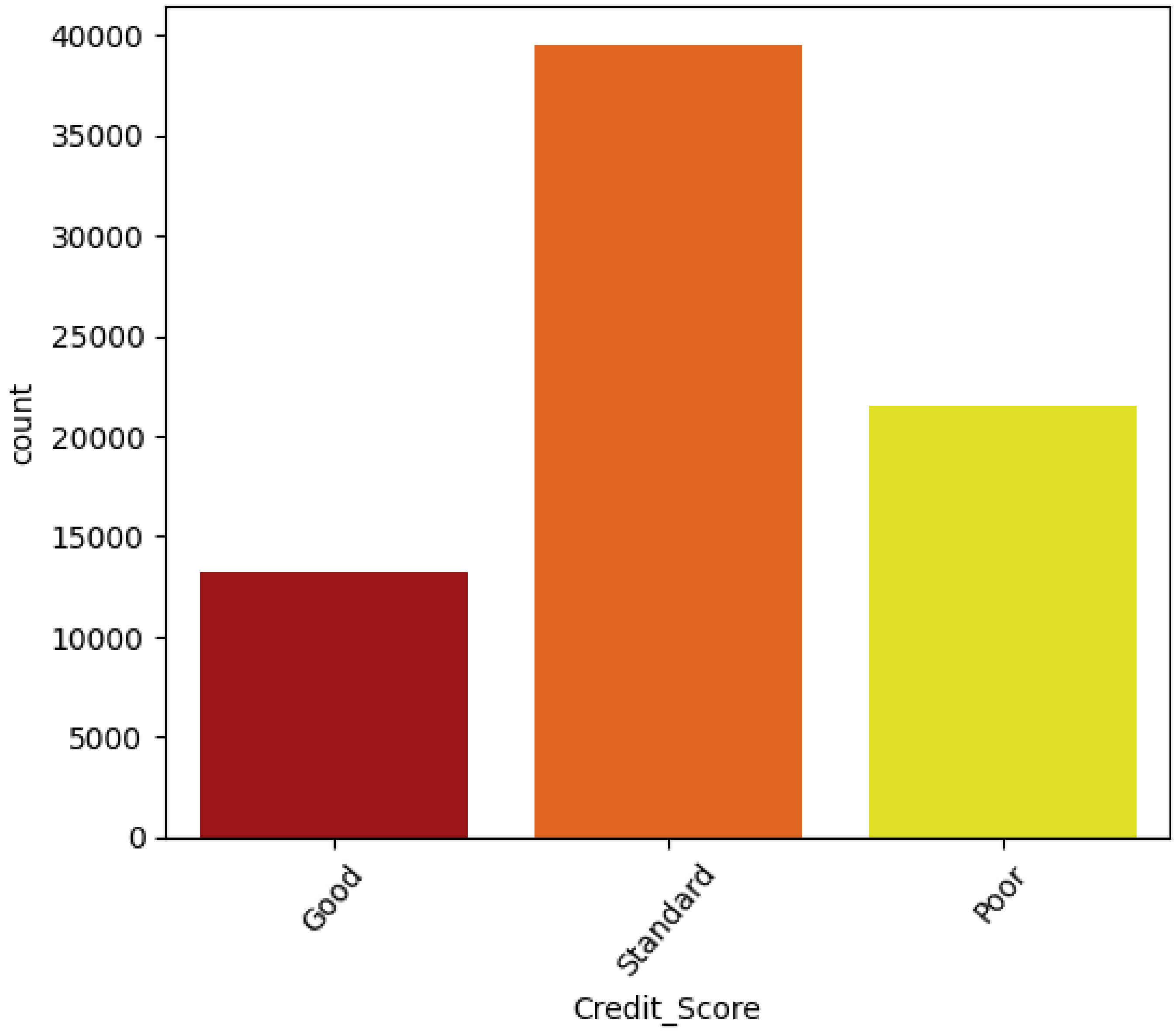



```
pay_map={

      "Yes":1,

      "No":2,

      "NM":3

}

df['Payment_of_Min_Amount'] = df['Payment_of_Min_Amount'].replace(pay_map)
```

```
df["Credit_Score"].value_counts()
```

```
Standard    39441
Poor        21509
Good        13214
Name: Credit_Score, dtype: int64
```

```
plt.figure(figsize=(6,5))
sns.countplot(x="Credit_Score",data=df,palette="hot")
plt.xticks(rotation=50)
plt.show()
```




```
score_map={

      "Standard":0,

      "Poor":1,

      "Good":2

}
df['Credit_Score'] = df['Credit_Score'].replace(score_map)
```

Handling Missing Data

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

```
Month                            0
Age                              0
Occupation                       0
Annual_Income                    0
Monthly_Inhand_Salary        11112
Num_Bank_Accounts                0
Num_Credit_Card                  0
Interest_Rate                    0
Num_of_Loan                      0
Delay_from_due_date              0
Num_of_Delayed_Payment        5195
Changed_Credit_Limit             0
Num_Credit_Inquiries          1472
Credit_Mix                       0
Outstanding_Debt                 0
Credit_Utilization_Ratio         0
Credit_History_Age            6717
Payment_of_Min_Amount            0
Total_EMI_per_month              0
Amount_invested_monthly       3356
Payment_Behaviour             5676
Monthly_Balance                906
Credit_Score                     0
dtype: int64
```

```
mean_salary = df["Monthly_Inhand_Salary"].mean()
df["Monthly_Inhand_Salary"].fillna(mean_salary, inplace=True)

df["Num_of_Delayed_Payment"] = pd.to_numeric(df["Num_of_Delayed_Payment"],
errors="coerce")
n_mean=df["Num_of_Delayed_Payment"].mean()
df["Num_of_Delayed_Payment"].fillna(n_mean, inplace=True)

in_mean=df["Num_Credit_Inquiries"].mean()
df["Num_Credit_Inquiries"].fillna(in_mean, inplace=True)

df['Credit_History_Age'] = df['Credit_History_Age'].str.extract(r'(\d+)')

df["Credit_History_Age"] = pd.to_numeric(df["Credit_History_Age"], errors="coerce")
credit_mean=df["Credit_History_Age"].mean()
df["Credit_History_Age"].fillna(credit_mean, inplace=True)

df["Amount_invested_monthly"] = pd.to_numeric(df["Amount_invested_monthly"],
errors="coerce")
invest_mean=df["Amount_invested_monthly"].mean()
df["Amount_invested_monthly"].fillna(invest_mean, inplace=True)

df.dropna(subset=["Payment_Behaviour"], inplace=True)

df["Monthly_Balance"] = pd.to_numeric(df["Monthly_Balance"], errors="coerce")
month_mean=df["Monthly_Balance"].mean()
df["Monthly_Balance"].fillna(month_mean, inplace=True)
```

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

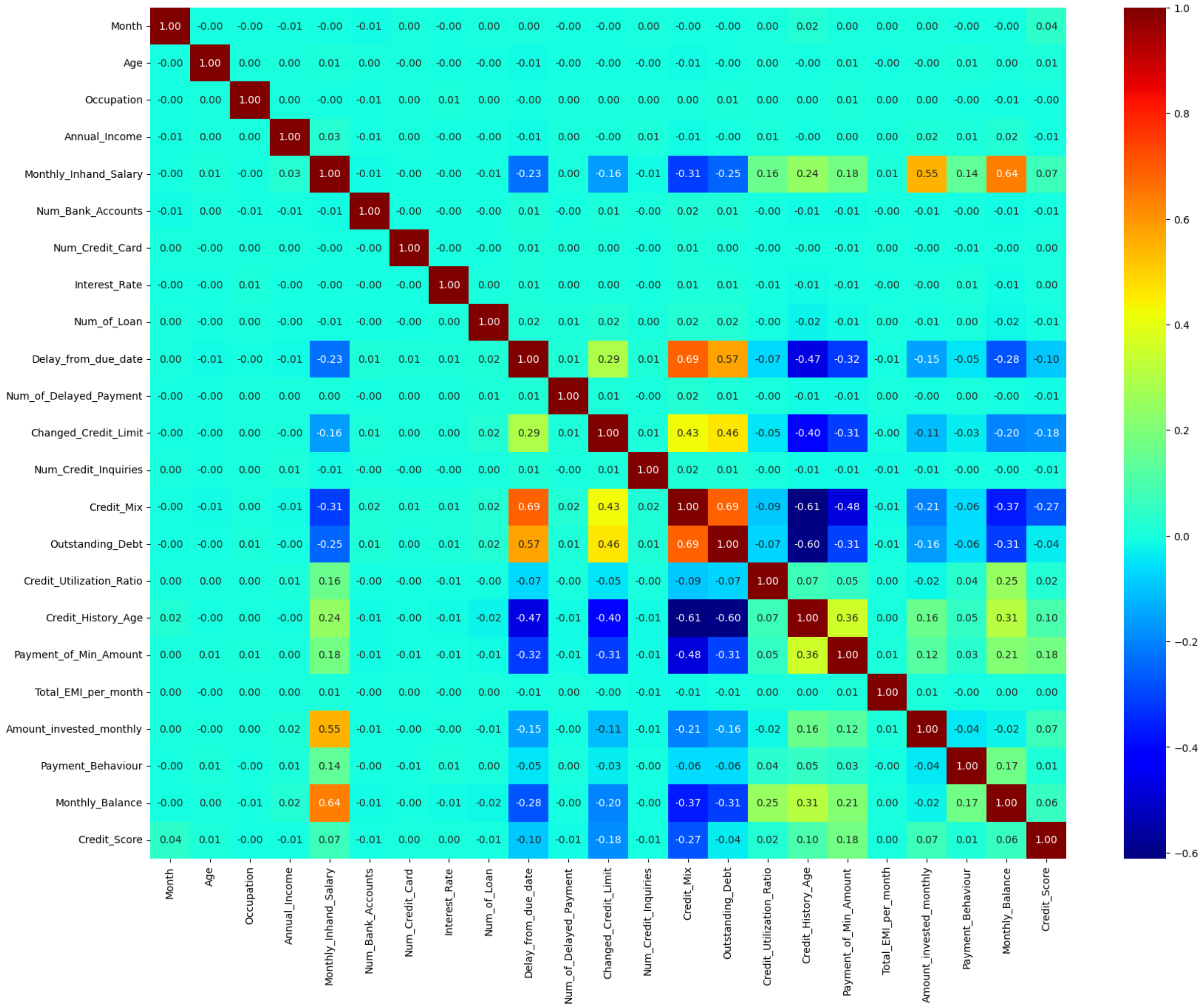
```
Month                       0
Age                         0
Occupation                  0
Annual_Income               0
Monthly_Inhand_Salary       0
Num_Bank_Accounts           0
Num_Credit_Card             0
Interest_Rate               0
Num_of_Loan                 0
Delay_from_due_date         0
Num_of_Delayed_Payment      0
Changed_Credit_Limit        0
Num_Credit_Inquiries        0
Credit_Mix                  0
Outstanding_Debt            0
Credit_Utilization_Ratio    0
Credit_History_Age          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
```

All missing values have been handled

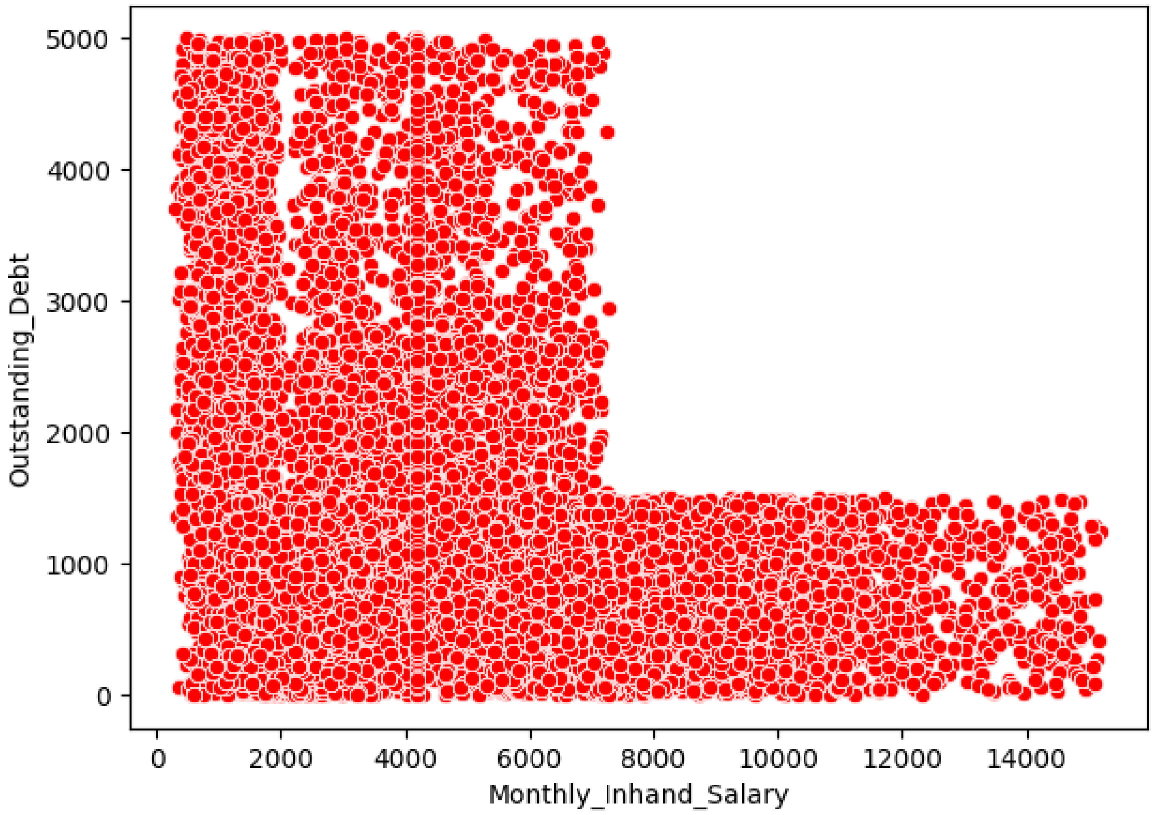
```
# Pre-processing the rest of the columns

df["Annual_Income"] = pd.to_numeric(df["Annual_Income"], errors="coerce")
an_mean=df["Annual_Income"].mean()
df["Annual_Income"].fillna(an_mean, inplace=True)

df['Outstanding_Debt'] = pd.to_numeric(df['Outstanding_Debt'].str.replace(r'[^0-9.]',
'', regex=True), errors='coerce')

df['Changed_Credit_Limit'] = df['Changed_Credit_Limit'].replace('_',np.nan)  # Replace
'_' with 0
df["Changed_Credit_Limit"] = pd.to_numeric(df["Changed_Credit_Limit"],
errors="coerce")
c_mean=df["Changed_Credit_Limit"].mean()
df["Changed_Credit_Limit"].fillna(c_mean, inplace=True)

df['Age'] = df['Age'].replace('-500',np.nan)
df["Age"] = pd.to_numeric(df["Age"], errors="coerce")
age_mean=df["Age"].mean()
df["Age"].fillna(age_mean, inplace=True)

df["Num_of_Loan"] = pd.to_numeric(df["Num_of_Loan"], errors="coerce")
num_mean=df["Num_of_Loan"].mean()
df["Num_of_Loan"].fillna(num_mean, inplace=True)

df['Delay_from_due_date'] = df['Delay_from_due_date'].abs()
```

Data Visualization

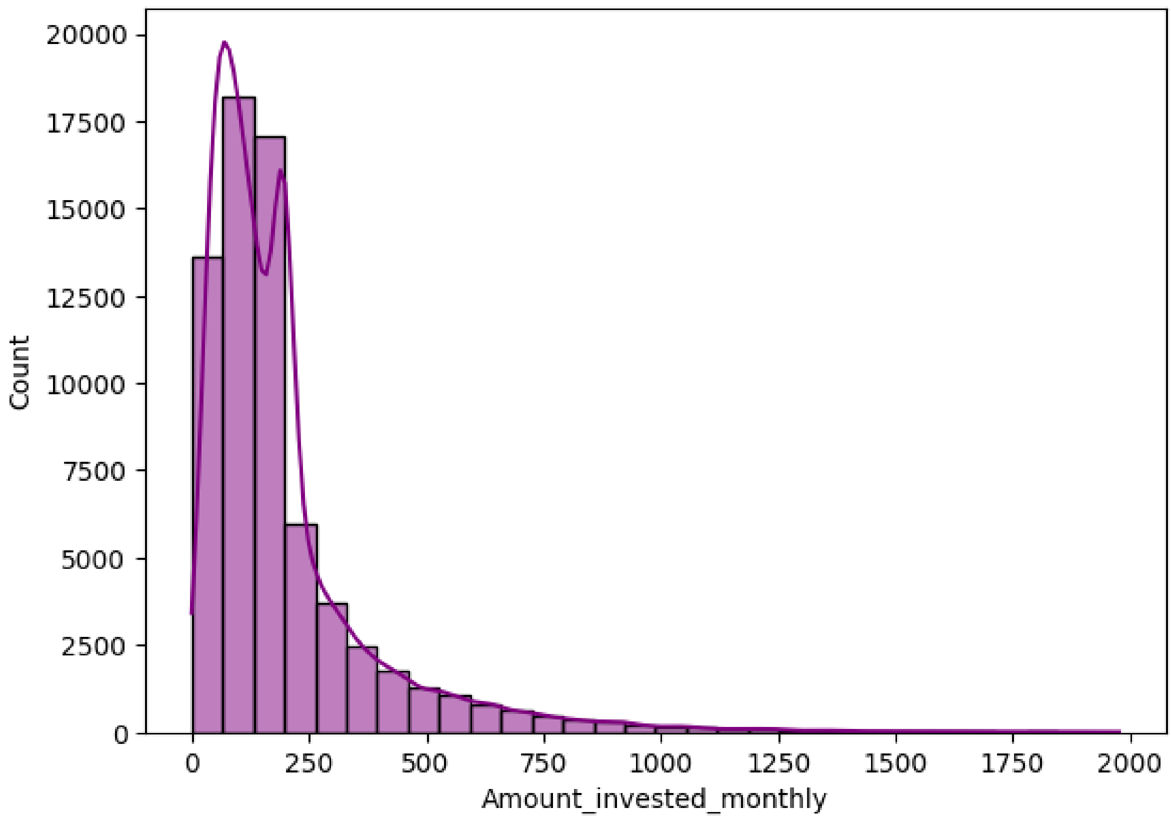
```
cr=df.corr()
plt.figure(figsize=(20,15))
sns.heatmap(cr,annot=True,fmt=".2f",cmap="jet")
plt.show()
```



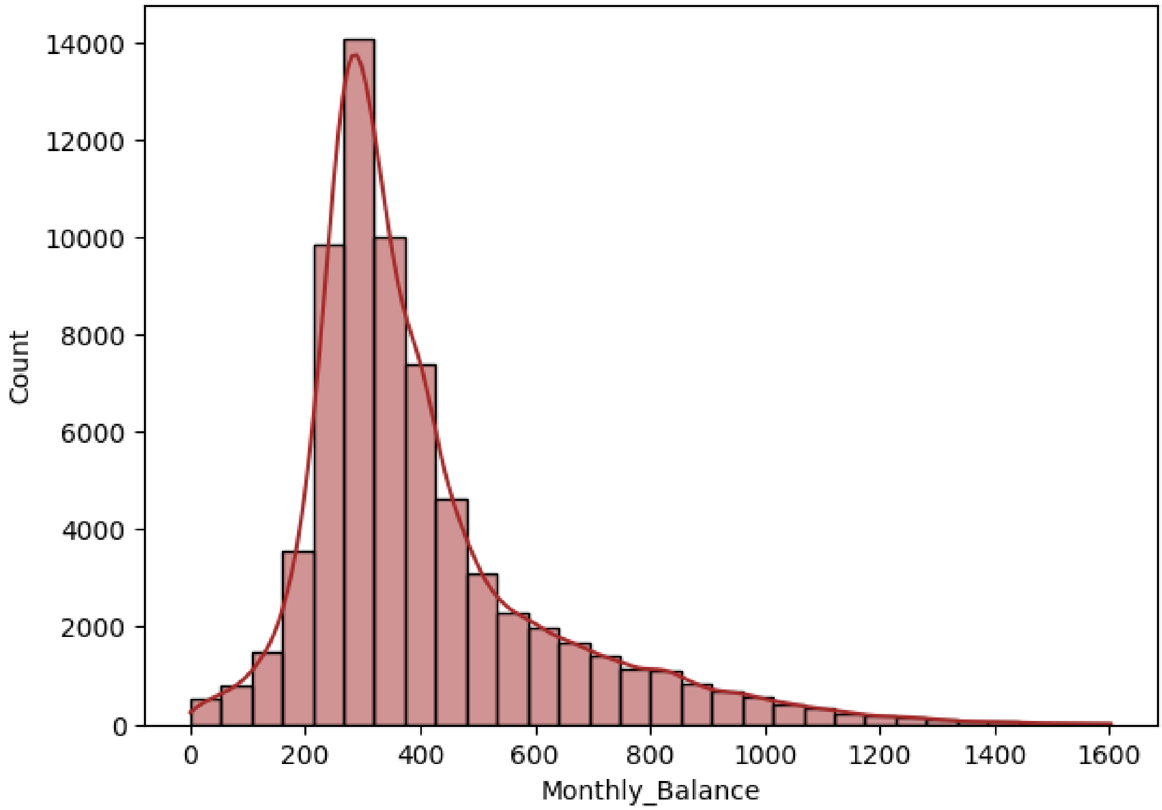
```
plt.figure(figsize=(7,5))
sns.scatterplot(data=df, x="Monthly_Inhand_Salary", y="Outstanding_Debt",color="red")
plt.show()
```



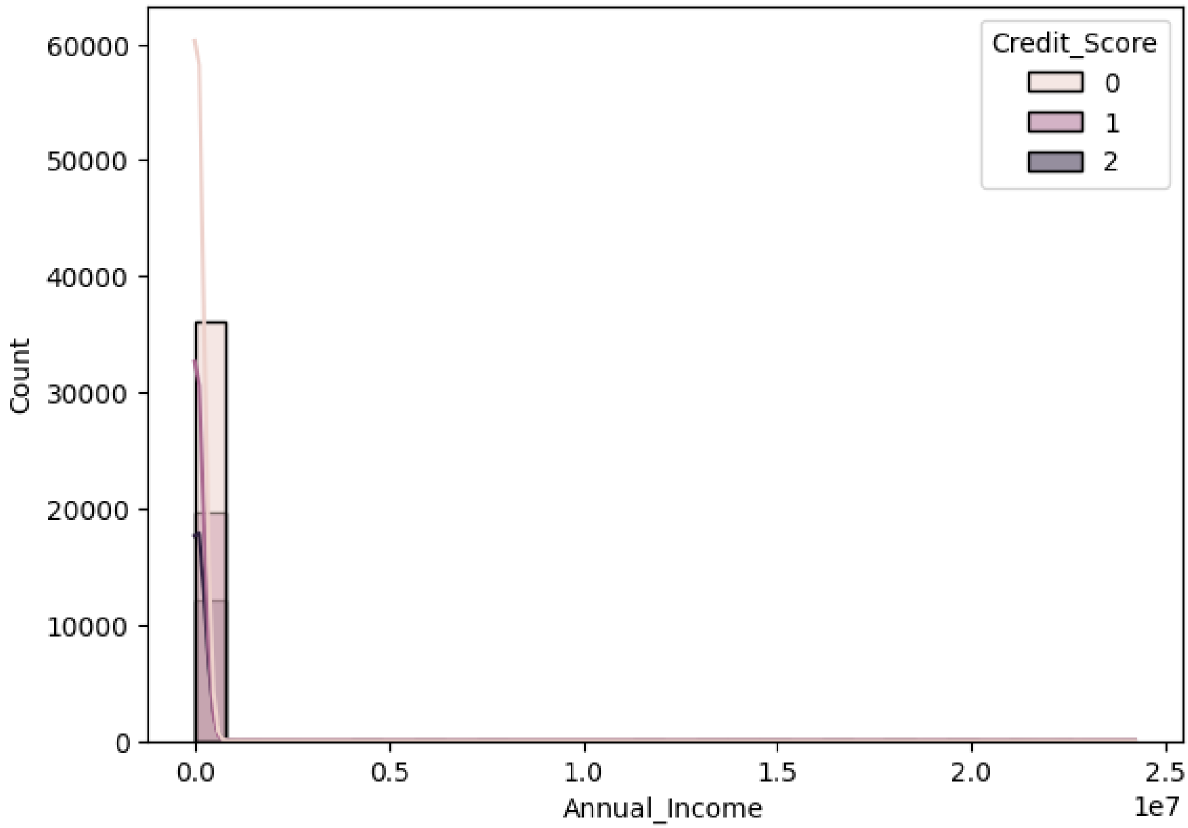

```
plt.figure(figsize=(7,5))
sns.histplot(data=df, x="Amount_invested_monthly", kde=True,bins=30,color="purple")
plt.show()
```



```
plt.figure(figsize=(7,5))
sns.histplot(data=df, x="Monthly_Balance", kde=True,bins=30,color="brown")
plt.show()
```




```
plt.figure(figsize=(7,5))
sns.histplot(data=df, x="Annual_Income", kde=True,bins=30,hue="Credit_Score")
plt.show()
```



Data Scaling

```
columns_to_scale = ['Age', 'Annual_Income', 'Monthly_Inhand_Salary',
'Outstanding_Debt','Credit_Utilization_Ratio', 'Credit_History_Age',
'Total_EMI_per_month','Amount_invested_monthly', 'Monthly_Balance']

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()


df[columns_to_scale] = scaler.fit_transform(df[columns_to_scale])

x=df.drop("Credit_Score",axis=1)
y=df["Credit_Score"]

from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=42)
```

Model Training

We will be training the dataset on 2 different models:

1. Extreme Gradient Boosting Classifier
2. Light Gradient Boosting Machine

```
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report,log_loss
from sklearn.metrics import roc_curve, auc
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
```

Extreme Gradient Boosting Classifier

```
xgb_classifier = XGBClassifier(max_depth=3, learning_rate=0.1,
n_estimators=100,eval_metric='logloss', objective='binary:logistic', booster='gbtree')
xgb_classifier.fit(xtrain, ytrain)
```

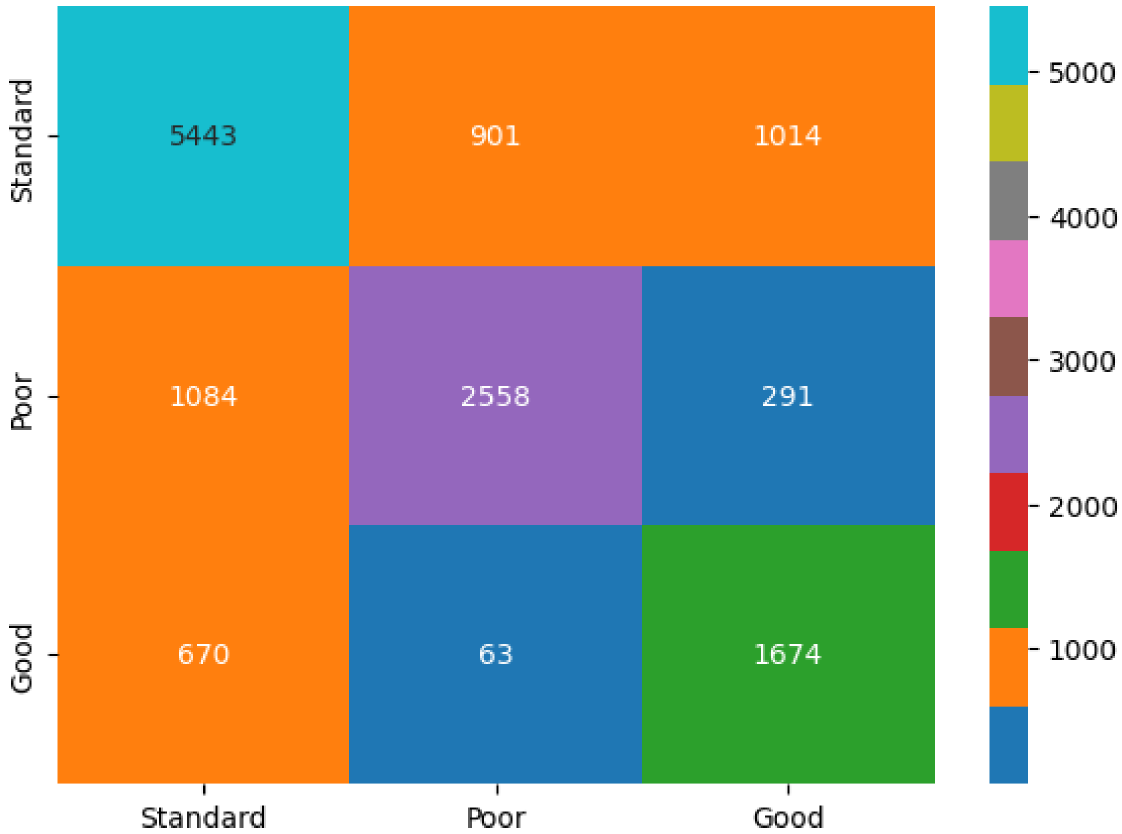
```
XGBClassifier(base_score=None, booster='gbtree', callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.1, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=3,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=100,
              n_jobs=None, num_parallel_tree=None, objective='multi:softprob', ...)
```

```
pred=xgb_classifier.predict(xtest)
xgb_ac=accuracy_score(ytest,pred)
print("XGB Accuracy Score :",xgb_ac)
```

```
XGB Accuracy Score : 0.7063074901445466
```

```
cf_mat=confusion_matrix(ytest, pred)
label_name=["Standard","Poor","Good"]
plt.figure(figsize=(7,5))
```

```
sns.heatmap(cf_mat,annot=True,fmt="d",xticklabels=label_name,yticklabels=label_name,cm
ap="tab10")
plt.show()
```



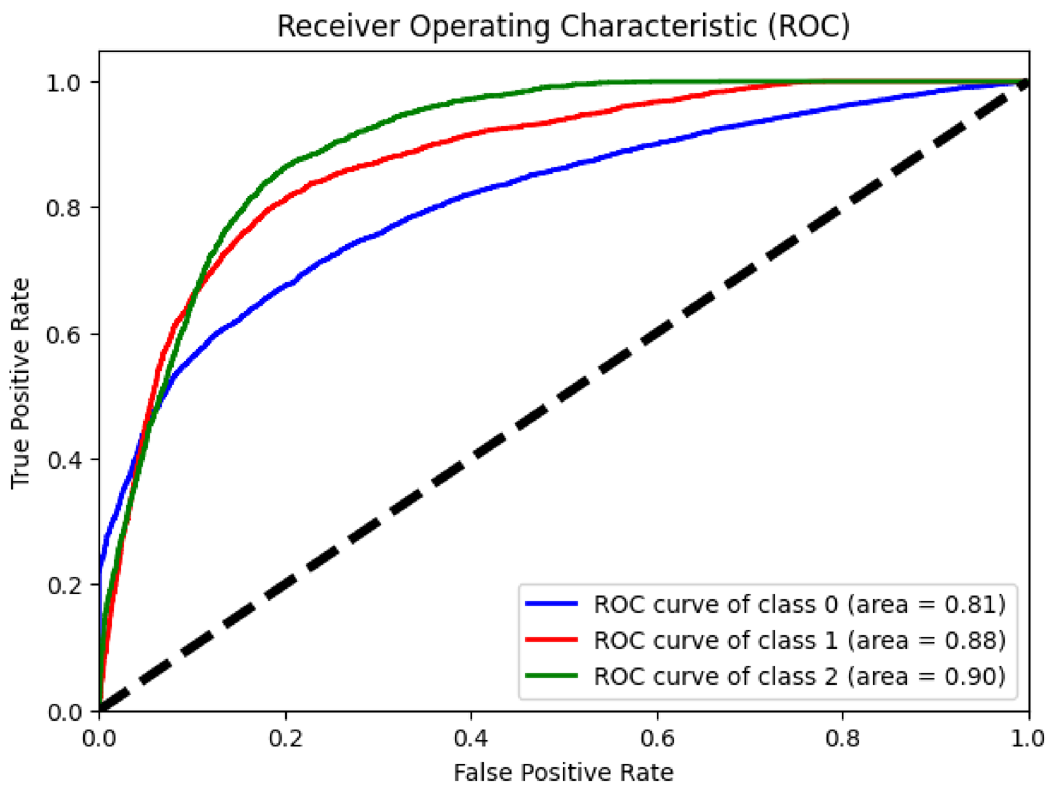

```
print(classification_report(ytest,pred,target_names=label_name))
```

```
              precision    recall  f1-score   support

    Standard       0.76      0.74      0.75      7358
        Poor       0.73      0.65      0.69      3933
        Good       0.56      0.70      0.62      2407

    accuracy                           0.71     13698
   macro avg       0.68      0.70      0.69     13698
weighted avg       0.71      0.71      0.71     13698
```

```python
x_loss=xgb_classifier.predict_proba(xtest)
logloss = log_loss(ytest,x_loss)
print("Log Loss:", logloss)
```

```
Log Loss: 0.6492392568955992
```

```python
from sklearn.metrics import roc_curve, auc

fpr = dict()
tpr = dict()
roc_auc = dict()

n_classes = 3  # Number of classes

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(ytest,x_loss[:, i], pos_label=i)
    roc_auc[i] = auc(fpr[i], tpr[i])


plt.figure(figsize=(7,5))
colors = ['blue', 'red', 'green']
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,label='ROC curve of class {0} (area =
{1:0.2f})'.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], color='black', linestyle='--',lw=4)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```



LightGBM (Light Gradient Boosting Machine)

```
lgb_classifier = LGBMClassifier(boosting_type='gbdt', num_leaves=31,max_depth=-1,learning_rate=0.1,
                                n_estimators=100,

                                random_state=42,
                                objective='multiclass',  # Multi-class objective
                                metric='multi_logloss')
```

```
lgb_classifier.fit(xtrain, ytrain, eval_set=[(xtest, ytest)])
```

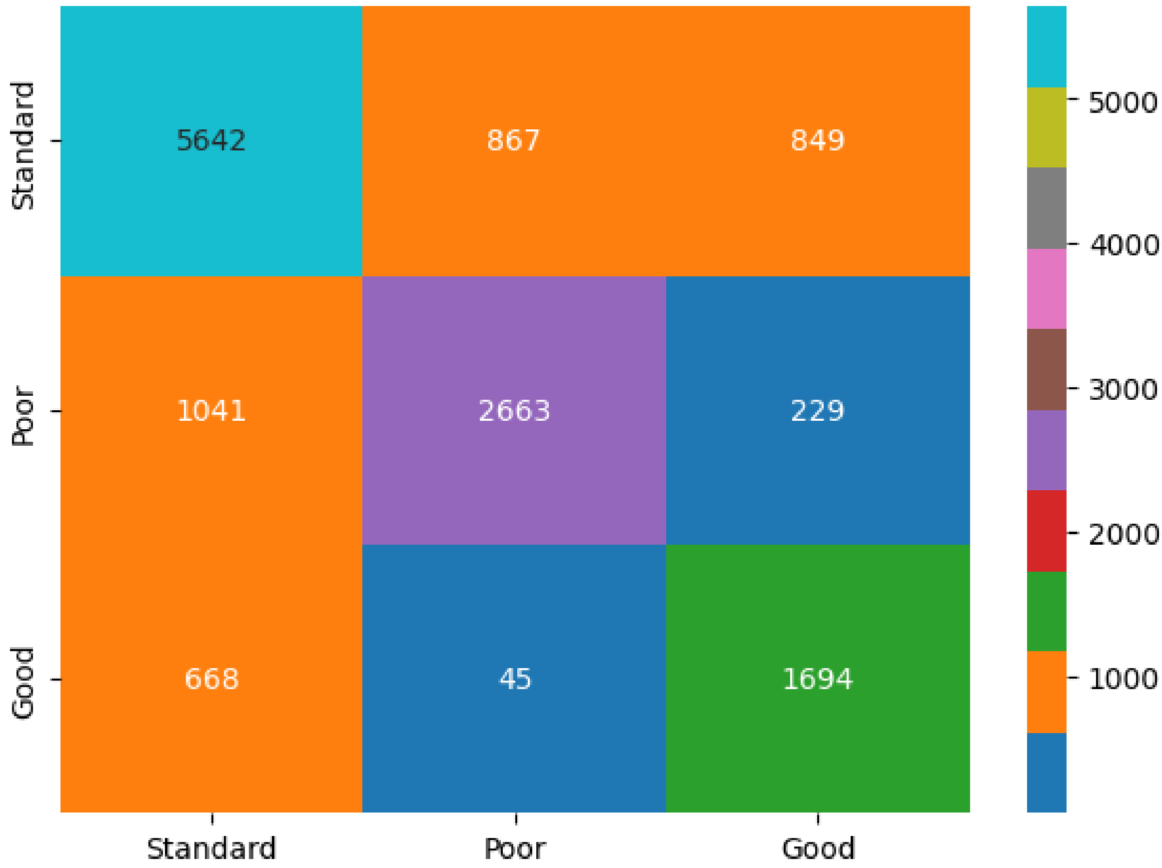
```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.004016 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3794
[LightGBM] [Info] Number of data points in the train set: 54790, number of used features: 22
[LightGBM] [Info] Start training from score -0.633113
[LightGBM] [Info] Start training from score -1.235680
[LightGBM] [Info] Start training from score -1.723577
```

```
LGBMClassifier(metric='multi_logloss', objective='multiclass', random_state=42)
```

```
pred0=lgb_classifier.predict(xtest)
acc0=accuracy_score(ytest,pred0)
print("accuracy score :",acc0)
```

```
accuracy score : 0.7299605781865965
```

```
cf_mat=confusion_matrix(ytest, pred0)
label_name=["Standard","Poor","Good"]
plt.figure(figsize=(7,5))
sns.heatmap(cf_mat,annot=True,fmt="d",xticklabels=label_name,yticklabels=label_name,cm
ap="tab10")
plt.show()
```



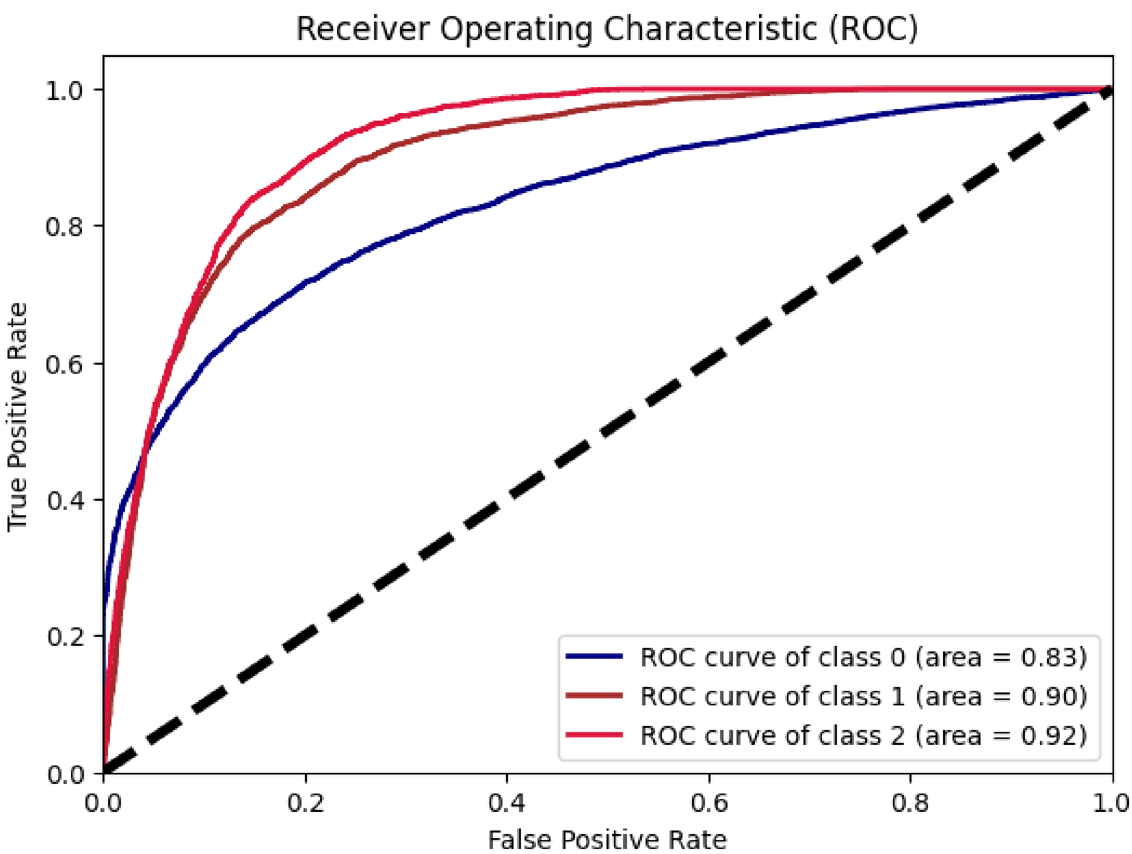

```
print(classification_report(ytest,pred0,target_names=label_name))
```

```
              precision    recall  f1-score   support

    Standard       0.77      0.77      0.77      7358
        Poor       0.74      0.68      0.71      3933
        Good       0.61      0.70      0.65      2407

    accuracy                           0.73     13698
   macro avg       0.71      0.72      0.71     13698
weighted avg       0.73      0.73      0.73     13698
```

```
lgb=lgb_classifier.predict_proba(xtest)
logloss2 = log_loss(ytest,lgb)
print("Log Loss:", logloss2)
```

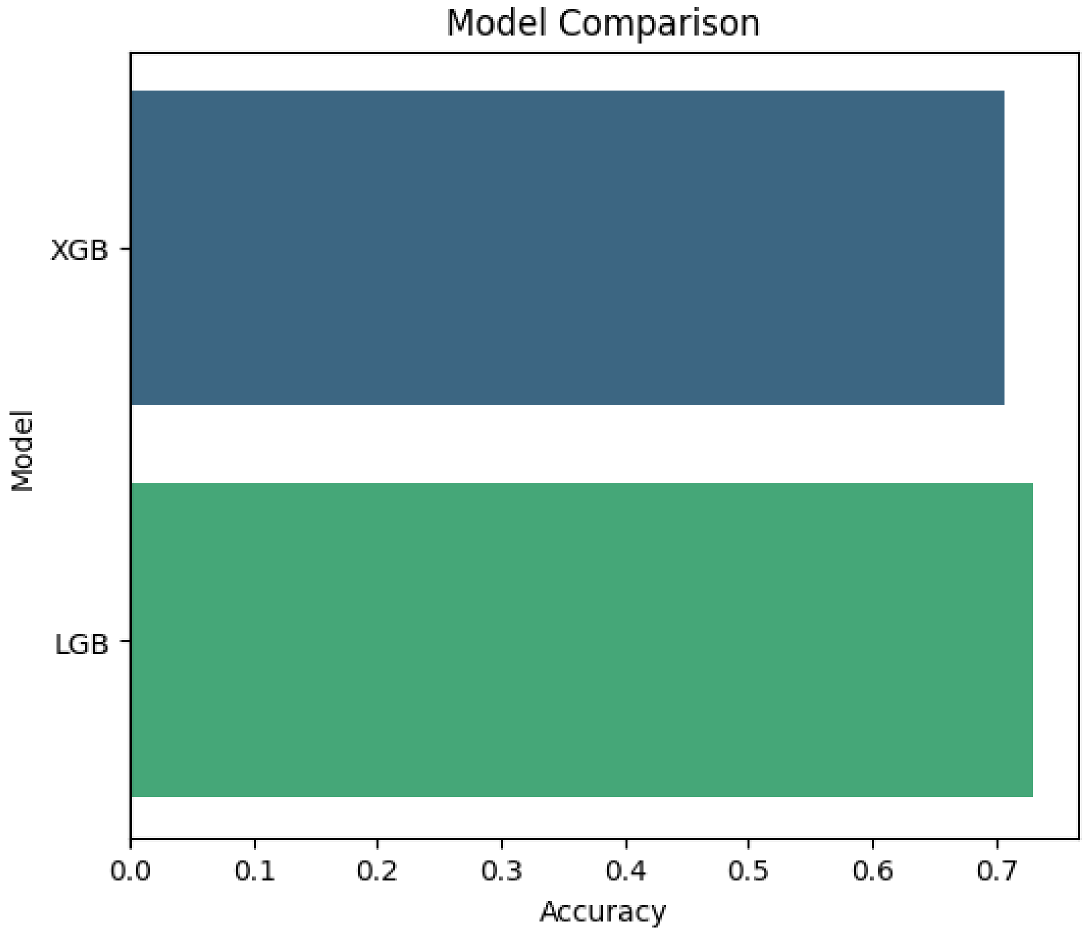
```
Log Loss: 0.5968068761627431
```

```
from sklearn.metrics import roc_curve, auc

fpr = dict()
tpr = dict()
roc_auc = dict()

n_classes = 3  # Number of classes

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(ytest,lgb[:, i], pos_label=i)
    roc_auc[i] = auc(fpr[i], tpr[i])


plt.figure(figsize=(7,5))
colors = ['navy', 'brown', 'crimson']
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,label='ROC curve of class {0} (area =
{1:0.2f})'.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], color='black', linestyle='--',lw=4)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```

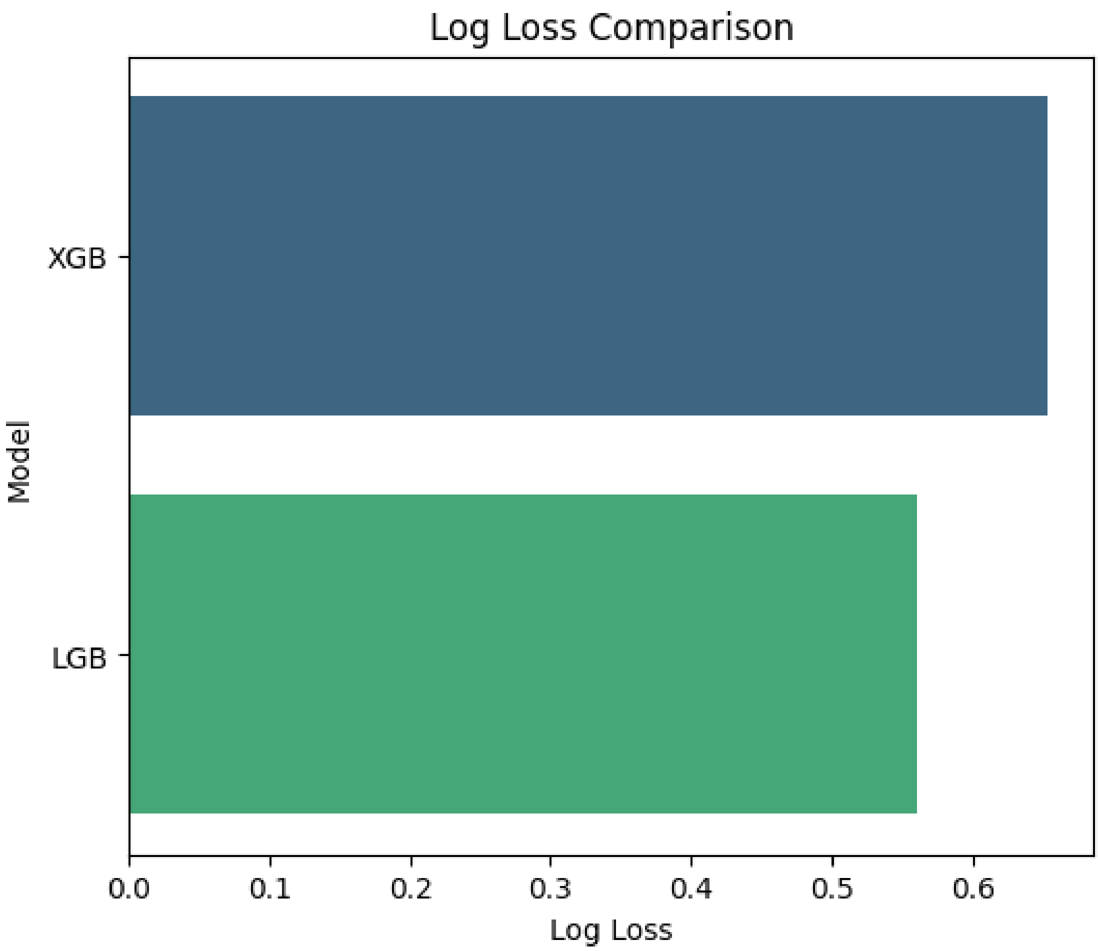


Model Evaluation

```
results = {'XGB': xgb_ac,'LGB': acc0}
results_df = pd.DataFrame(list(results.items()), columns=['Model', 'Accuracy'])
plt.figure(figsize=(6,5))
sns.barplot(y='Model', x='Accuracy', data=results_df, palette='viridis')
plt.title('Model Comparison')
plt.show()
```



```
log_loss_results = {'LGB': logloss2,'XGB': logloss}
log_loss_df = pd.DataFrame(list(log_loss_results.items()), columns=['Model', 'Log
Loss'])
plt.figure(figsize=(6,5))
sns.barplot(y='Model', x='Log Loss', data=log_loss_df, palette='viridis')
plt.title('Log Loss Comparison')
plt.show()
```



<u>Hence, LGBM is the most feasible model to train this dataset.</u>

Model Testing

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

# Load the test data
df_test = pd.read_csv("test.csv")



# Preprocess the test data
# Drop unnecessary columns
df_test.drop(["ID","Customer_ID","Name","SSN","Type_of_Loan"],axis=1,inplace=True)

# Encode categorical features
month_mapping = {
   'January': 1,
    'February': 2,
    'March': 3,
    "April":4,
    "May":5,
    "June":6,
    "July":7,
    "August":8,
    "September":9,
    "October":10,
    "November":11,
    "December":12}
df_test['Month'] = df_test['Month'].replace(month_mapping)


df_test.drop(df_test[df_test["Occupation"]=='_______'].index,inplace=True)
df_test.drop(df_test[df_test["Credit_Mix"]=='_'].index,inplace=True)


occupation_mapping = {
    'Lawyer': 1,
    'Architect': 2,
    'Engineer': 3,
    'Scientist': 4,
    'Mechanic': 5,
    'Accountant': 6,
    'Developer': 7,
    'Media_Manager': 8,
    'Teacher': 9,
    'Entrepreneur': 10,
    'Doctor': 11,
    'Journalist': 12,
    'Manager': 13,
    'Musician': 14,
    'Writer': 15,
    'Scientist': 16
}
df_test['Occupation'] = df_test['Occupation'].replace(occupation_mapping)
```

```python
credit_map={"Good":1,"Standard":2,"Bad":3}
df_test['Credit_Mix'] = df_test['Credit_Mix'].replace(credit_map)

df_test['Payment_Behaviour']= df_test['Payment_Behaviour'].replace("!@9#%8",np.nan)
category_mapping = {
    'Low_spent_Small_value_payments':1,
    'High_spent_Medium_value_payments':2,
    'Low_spent_Medium_value_payments': 3,
    'High_spent_Large_value_payments': 4,
    'High_spent_Small_value_payments': 5,
    'Low_spent_Large_value_payments': 6
}
df_test['Payment_Behaviour'] = df_test['Payment_Behaviour'].replace(category_mapping)

pay_map={"Yes":1,"No":2,"NM":3}
df_test['Payment_of_Min_Amount'] = df_test['Payment_of_Min_Amount'].replace(pay_map)



# Handle missing values


mean_salary = df_test["Monthly_Inhand_Salary"].mean()
df_test["Monthly_Inhand_Salary"].fillna(mean_salary, inplace=True)


df_test["Num_of_Delayed_Payment"] = pd.to_numeric(df_test["Num_of_Delayed_Payment"],
errors="coerce")
n_mean=df_test["Num_of_Delayed_Payment"].mean()
df_test["Num_of_Delayed_Payment"].fillna(n_mean, inplace=True)


in_mean=df_test["Num_Credit_Inquiries"].mean()
df_test["Num_Credit_Inquiries"].fillna(in_mean, inplace=True)


df_test['Credit_History_Age'] = df_test['Credit_History_Age'].str.extract(r'(\d+)')
df_test["Credit_History_Age"] = pd.to_numeric(df_test["Credit_History_Age"],
errors="coerce")
credit_mean=df_test["Credit_History_Age"].mean()
df_test["Credit_History_Age"].fillna(credit_mean, inplace=True)


df_test["Amount_invested_monthly"] = pd.to_numeric(df_test["Amount_invested_monthly"],
errors="coerce")
invest_mean=df_test["Amount_invested_monthly"].mean()
df_test["Amount_invested_monthly"].fillna(invest_mean, inplace=True)


df_test.dropna(subset=["Payment_Behaviour"], inplace=True)
df_test["Monthly_Balance"] = pd.to_numeric(df_test["Monthly_Balance"],

errors="coerce")
month_mean=df_test["Monthly_Balance"].mean()
df_test["Monthly_Balance"].fillna(month_mean, inplace=True)
```

```python
# Pre-process all features


df_test["Annual_Income"] = pd.to_numeric(df_test["Annual_Income"], errors="coerce")
an_mean=df_test["Annual_Income"].mean()
df_test["Annual_Income"].fillna(an_mean, inplace=True)


df_test['Outstanding_Debt'] =
pd.to_numeric(df_test['Outstanding_Debt'].str.replace(r'[^0-9.]', '', regex=True),
errors='coerce')


df_test['Changed_Credit_Limit'] = df_test['Changed_Credit_Limit'].replace('_',np.nan)
df_test["Changed_Credit_Limit"] = pd.to_numeric(df_test["Changed_Credit_Limit"],
errors="coerce")
c_mean=df_test["Changed_Credit_Limit"].mean()
df_test["Changed_Credit_Limit"].fillna(c_mean, inplace=True)


df_test['Age'] = df_test['Age'].replace('-500',np.nan)
df_test["Age"] = pd.to_numeric(df_test["Age"], errors="coerce")
age_mean=df_test["Age"].mean()
df_test["Age"].fillna(age_mean, inplace=True)


df_test["Num_of_Loan"] = pd.to_numeric(df_test["Num_of_Loan"], errors="coerce")
num_mean=df_test["Num_of_Loan"].mean()
df_test["Num_of_Loan"].fillna(num_mean, inplace=True)


df_test['Delay_from_due_date'] = df_test['Delay_from_due_date'].abs()



# Scale numerical features


columns_to_scale = ['Age', 'Annual_Income', 'Monthly_Inhand_Salary',
'Outstanding_Debt',
                    'Credit_Utilization_Ratio', 'Credit_History_Age',
'Total_EMI_per_month',
                    'Amount_invested_monthly', 'Monthly_Balance']
scaler = StandardScaler()
df_test[columns_to_scale] = scaler.fit_transform(df_test[columns_to_scale])




# Separate features and target


X_test = df_test.copy()

from lightgbm import LGBMClassifier
import joblib
```

```python
# Load the trained LightGBM model


lgb_classifier = LGBMClassifier(
    boosting_type='gbdt',
    num_leaves=31,
    max_depth=-1,
    learning_rate=0.1,
    n_estimators=100,
    random_state=42,
    objective='multiclass',
    metric='multi_logloss'
)



# Load the saved model from the specified path
model_path = '/content/lgb_model.pkl'
lgb_classifier = joblib.load(model_path)



# Make predictions on the test set
predicted_credit_scores = lgb_classifier.predict(X_test)



# Print the first 100 predicted credit scores in the form of an array


print("First 100 Predicted Credit Scores:")
print("[", end="")

for i in range(min(100, len(predicted_credit_scores))):
    if i > 0:
        print(", ", end="")
    print(predicted_credit_scores[i], end="")

    # Add line break after every 10 scores
    if (i + 1) % 10 == 0:
        print("\n", end="")

print("]")
```

```
First 100 Predicted Credit Scores:
[0.0, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0, 0.0, 2.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 2.0, 2.0, 2.0, 2.0, 2.0
, 2.0, 2.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 2.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.0, 2.0, 2.0
, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 2.0
, 2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0
, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0
]
```

```python
# Add the predicted credit scores to the test data
df_test["Credit_Score"] = predicted_credit_scores



# Save the updated test data to the same CSV file
df_test.to_csv("test.csv", index=False)

print("Predicted Credit Scores added to test.csv")
```

```
Predicted Credit Scores added to test.csv
```

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	W
1	Month	Age	Occupatio	Annual_In	Monthly_I	Num_Bank	Num_Cred	Interest_R	Num_of_L	Delay_fror	Num_of_D	Changed_(	Num_Cred	Credit_Mi	Outstandir	Credit_Util	Credit_Hist	Payment_(	Total_EMI_per_	Amount_invest	Payment_Beha	Credit_Score
2	9	-0.14056	16	-0.11279	-0.80938	3	4	3	4	3	7	11.27	2022	1	-0.53269	0.538351	0.441874	2	-0.169330331	0.218586386	1	0
3	10	-0.13906	16	-0.11279	-0.80938	3	4	3	4	3	9	13.27	4	1	-0.53269	0.151383	0.441874	2	-0.169330331	-0.926354538	2	0
4	11	-0.13906	16	-0.11279	-0.80938	3	4	3	4	1	4	12.27	4	1	-0.53269	0.299881	0.001651	2	-0.169330331	-0.251829352	3	0
5	12	-2.13E-17	16	-0.11279	-0.00016	3	4	3	4	4	5	11.27	4	1	-0.53269	0.029545	0.567991	2	-0.169330331	-0.832615219	2	0
6	10	-0.13306	9	-0.10078	-0.3924	2	4	6	1	3	3	5.42	5	1	-0.71056	-0.42331	1.072459	2	-0.172835206	0.298318686	6	2
7	9	-0.12256	3	-0.01805	-0.00016	1	5	8	3	8	1942	7.1	3	1	-0.10482	0.577356	-0.06259	2	-0.146835152	1.074514535	3	0
8	10	-0.12256	3	-0.01805	2.75235	1	5	8	3	6	3	2.1	3	1	-0.10482	0.666624	-0.06259	2	-0.146835152	1.373079141	6	0
9	11	-0.12256	3	-0.01805	2.75235	1	5	8	1381	8	5	7.1	5	1	-0.10482	-0.08998	-0.06259	2	-0.146835152	3.435558238	3	0
10	12	-0.12256	3	-0.01805	2.75235	1	5	8	3	8	6	7.1	5	1	-0.10482	0.279386	0.001651	2	-0.146835152	1.653452442	2	2
11	9	-0.09255	10	-0.10395	-0.53865	2	5	4	1	5	6	1.99	4	1	-0.68675	1.389052	-0.18871	2	-0.173108767	-0.672857323	2	0
12	11	-0.09255	10	-0.10395	-0.53865	2	5	4	1	5	6	1.99	4	1	-0.68675	0.889461	-0.06259	2	-0.173108767	-0.019558983	3	0
13	12	-0.09255	10	3.041312	-0.53865	2	5	4	1	6	6	1.99	7	1	-0.68675	0.645767	-0.06259	2	-0.173108767	0.204170608	1	0
14	10	-0.14206	7	-0.10024	-0.45587	7	5	5	0	5	18	2.58	5	2	-0.41651	-0.12276	1.576926	1	-0.174979263	-0.436897723	2	0
15	11	-0.14206	7	-0.10024	-0.45587	7	5	5	-100	5	15	2.58	5	2	-0.41651	0.811251	0.001651	1	-0.174979263	-0.792803763	2	0
16	12	-0.14206	7	-0.10024	-0.45587	7	5	5	0	5	18	2.58	5	2	-0.41651	-0.53104	1.576926	1	-0.174979263	-0.077445398	1	0
17	9	-0.12856	1	-0.07093	0.621815	4	5	8	0	8	7	14.14	4	1	-0.75988	1.146024	1.703043	2	-0.174979263	-0.741954674	4	2
18	10	-0.12706	1	-0.07093	0.621815	4	5	8	0	9	31.27265	10.14	4	1	-0.75988	1.163843	1.703043	2	-0.174979263	-0.187923124	5	2
19	12	-0.12706	1	-0.07093	0.621815	4	5	8	0	8	7	10.14	1552	1	-0.75988	-0.61229	1.703043	2	-0.174979263	-0.344200364	4	2
20	9	-0.12406	1	-0.0271	2.16184	0	1	8	2	0	31.27265	9.34	4	1	-0.93001	-0.75906	0.001651	2	-0.071148171	0.684624143	2	2
21	10	-0.12406	1	-0.0271	2.16184	0	1	8	2	0	31.27265	9.34	4	1	-0.93001	-0.24542	1.576926	2	-0.071148171	4.0998329	6	2
22	11	-0.12406	1	-0.0271	2.16184	0	1	8	2	0	1	9.34	4	1	-0.93001	-1.21592	1.576926	3	-0.071148171	5.031770399	3	2
23	12	-0.12406	1	-0.0271	-0.00016	0	1	8	2	0	0	10.34	4	1	-0.93001	0.466955	1.576926	3	-0.071148171	1.91705186	3	2
24	10	-0.13006	8	-0.10136	-0.00016	8	7	15	3	30	17	17.13	9	2	0.243335	1.191674	-0.44094	1	-0.166948447	-0.439804559	5	1
25	11	-0.12856	8	-0.10136	-0.53912	8	7	15	3	29	15	17.13	9	2	0.243335	0.693373	-0.44094	3	-0.166948447	-0.773827318	2	1
26	12	-0.12856	8	-0.10136	-0.53912	8	7	15	3	30	14	17.13	9	2	0.243335	0.48327	-0.44094	3	-0.166948447	-0.510147292	2	1
27	9	-0.13906	11	-0.03968	1.9469	2	5	7	3	11	12	8.24	8	1	-0.03996	1.796473	0.441874	3	-0.149125439	0.095320948	4	0
28	10	-0.13906	11	-0.03968	1.9469	2	5	7	3	11	9	10.37048	8	1	-0.03996	-0.60732	0.441874	2	-0.149125439	3.83845421	3	0
29	11	1.19606	11	-0.03968	-0.00016	2	5	7	3	11	8	8.24	8	1	-0.03996	-0.64303	0.441874	2	-0.149125439	1.233491855	6	0
30	10	-0.10756	12	-0.10343	-0.46552	1	6	12	2	2	3	5.76	5	1	-0.8699	0.30814	1.072459	2	-0.169667481	-0.458351102	2	0
31	12	-0.10756	12	-0.10343	-0.00016	1	6	12	2	1	0	5.76	5	1	-0.8699	1.570474	1.072459	2	-0.169667481	-0.921923932	4	0
32	9	-2.13E-17	9	-0.10161	-0.42312	5	5	20	3	16	19	11	8	2	-0.08232	-0.84085	0.063524	1	-0.167571757	0.402506295	1	0
33	12	-2.13E-17	9	-0.10161	-0.42312	5	5	20	3	16	20	11	8	2	-0.08232	1.234141	0.001651	1	-0.167571757	-0.491980214	2	0
34	9	-0.12556	3	-0.05969	1.061079	3	6	1	2	4	2	3.51	3	1	-0.41087	0.731023	0.001651	2	-0.159576621	-0.626382624	4	2
35	12	-0.12556	3	-0.05969	-0.00016	3	6	1766	2	5	2	3.51	5	1	-0.41087	1.678047	0.946342	2	-0.159576621	0.903264287	5	0
36	9	-0.12106	10	-0.08585	0.201784	6	4	14	3	10	8	5.54	7	2	-1.0801	0.239323	1.072459	1	-0.16080512	0.712954598	6	0
37	10	-0.12106	10	-0.08585	0.201784	6	4	14	3	10	10	5.54	7	2	-1.0801	0.341156	1.072459	1	-0.16080512	-0.267277889	3	0
38	9	-0.11656	13	-0.12075	-1.25817	6	5	32	7	24	9	8.86	9	2	1.023104	1.214409	-1.19764	1	-0.170814695	0.000321741	5	1
39	12	-2.13E-17	13	-0.12075	-1.25817	6	5	32	7	23	31.27265	8.86	9	2	1.023104	-0.52476	-1.19764	1	-0.170814695	-0.808116909	1	1
40	9	-0.11956	10	-0.10788	-0.60623	8	7	14	5	12	31.27265	7.83	5	2	-0.57742	-1.23262	-0.06259	1	-0.163433137	-0.93757727	4	0
41	10	-0.11956	10	-0.10788	-0.60623	8	7	14	5	16	14	7.83	7	2	-0.57742	1.244397	-0.06259	1	-0.163433137	-0.477416178	3	0
42	12	-0.11956	10	-0.10788	-0.60623	8	7	14	5	16	15	1.83	7	2	-0.57742	1.214487	0.063524	1	-0.163433137	-0.290055762	3	0
43	9	-0.12856	16	-0.10296	-0.42534	6	6	3097	2	8	14	6.28	1	2	-0.52554	0.961036	-0.18871	1	-0.169835534	-0.916746864	4	0
44	10	-0.12856	16	-0.10296	-0.42534	163	6	7	2	8	31.27265	6.28	1	2	-0.52554	-1.01117	-0.18871	3	-0.169835534	-0.147277301	6	0
45	10	-0.14356	6	2.22E-17	-0.00016	6	7	16	0	16	11	9.13	4	2	-0.11035	0.465682	1.324692	3	-0.174979263	-0.378220067	4	0
46	11	-0.14356	6	-0.05709	1.172754	6	7	16	0	18	31.27265	16.13	30.0921	2	-0.11035	1.018075	1.450809	1	-0.174979263	1.640981121	1	0
47	9	-0.10605	9	-0.10273	-0.49146	6	7	17	6	9	1150	9.22	10	2	-0.12186	-0.66897	-1.44988	1	-0.146303275	0.244807759	1	0
48	12	-0.10605	9	-0.10273	-0.49146	6	7	17	6	11	31.27265	9.22	10	2	-0.12186	-1.74312	0.001651	1	-0.146303275	-0.734741252	4	0
49	10	-0.13456	14	-0.0527	1.123905	6	6	12	0	18	8	17.92	1	2	-1.14242	0.040097	1.072459	3	-0.103213461	2.676395935	6	2
50	11	-0.13456	14	-0.0527	1.123905	6	6	12	0	18	6	10.92	3	2	-1.14242	0.811724	1.198575	1	-0.103213461	-0.191464675	2	2

```python
# Create a count plot with custom x-tick labels
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(6, 5))
sns.countplot(x=predicted_credit_scores, data=df_test, palette="hot")
plt.title('Count of Predicted Credit Scores')
plt.xlabel('Credit Score')
plt.ylabel('Count')
plt.xticks(ticks=[0, 1, 2], labels=['Standard', 'Poor', 'Good'])
plt.show()
```