CREDIT SCORE CLASSIFICATION

An Internship Project Report submitted to ICT Academy Of Kerala in partial fulfillment of the requirements for the Internship Program

THIRUVANANTHAPURAM, KERALA, INDIA
Nov 2023

submitted by

ARSHAD ARIF M M

Abstract

In response to the evolving landscape of credit assessment, this project, conducted as part of the ICTAK Credit Score Classification Model Development internship, focuses on the development and deployment of an intelligent web application. The application utilizes a predictive model, created through data science and machine learning techniques, to categorize individuals into three creditworthiness levels: Good, Standard, and Poor.

The core objective is to seamlessly integrate the predictive model into a Flask web application, offering a user-friendly interface for banks and financial institutions. This deployment ensures accessibility and ease of use, allowing users to input relevant data and receive instant credit score classifications.

The project unfolds through four distinctive phases: Data Exploration and Preprocessing, Model Selection and Training, Model Evaluation, and Deployment in a Flask Web Application. By leveraging Flask, a lightweight and versatile web framework, the model becomes readily available to end-users, promising to streamline credit assessment processes and enhance decision-making in the finance industry.

Upon completion, thIS project is poised to deliver a fully functional Flask web application, providing an interactive platform for credit score classification. This innovative approach not only optimizes operational efficiency but also paves the way for tailored financial services based on precise creditworthiness categorization. As the finance industry embraces the digital era, this project marks a significant stride in the direction of redefining credit assessment methodologies.

Problem Definition

Overview

The Credit Score Prediction Web Application project aims to address the contemporary challenges within the financial industry, particularly those associated with manual credit assessment processes. In this transformative initiative, our objective is to develop and deploy an intelligent web application that leverages data science and machine learning to predict and categorize individuals into three creditworthiness levels: Good, Standard, and Poor.

Banks and financial institutions currently face inefficiencies and potential inaccuracies in their credit assessment procedures, prompting the need for a more streamlined and automated solution. By integrating a predictive model into a user-friendly Flask web application, we seek to enhance accessibility, reduce manual effort, and improve decision-making in the finance sector.

Problem Statement

In contemporary finance, the manual assessment of creditworthiness poses significant challenges, consuming time and resources while introducing the potential for human error. Banks and financial institutions grapple with the need for a more efficient, accurate, and scalable solution to streamline the credit assessment process. The existing landscape calls for an innovative approach that integrates data science and machine learning into a user-friendly web application, capable of predicting and categorizing individuals into specific creditworthiness levels.

Introduction

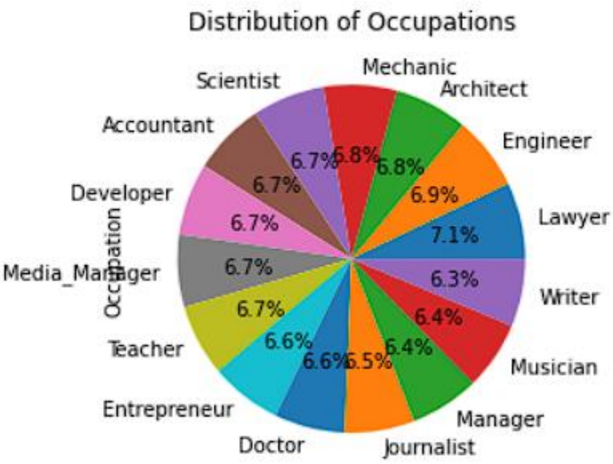
The financial industry, a cornerstone of global economic stability, constantly seeks innovations to enhance efficiency and precision. In response to the persistent challenges associated with manual credit assessment, my project, part of the ICTAK Credit Score Classification Model Development internship, endeavors to revolutionize the conventional methods through an intelligent web application. This application, powered by a predictive model developed using data science and machine learning, is designed to categorize individuals into three distinct creditworthiness levels: Good, Standard, and Poor.

The critical need to expedite credit assessment processes and mitigate human error prompted the integration of this predictive model into a Flask web application. This not only ensures accessibility for banks and financial institutions but also offers an intuitive interface for users to input relevant data and receive instantaneous credit score classifications. The project unfolds through comprehensive phases, ensuring a meticulous approach from data exploration and preprocessing to model training, evaluation, and ultimately, deployment in a Flask web application.

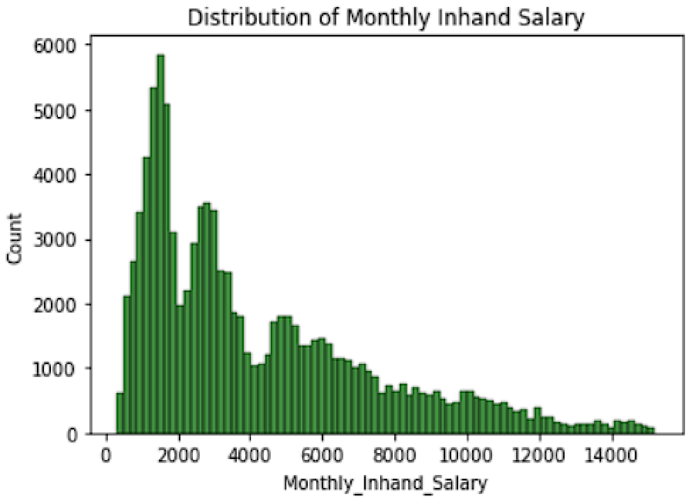
By leveraging Flask, a versatile web framework, the model becomes readily available to end-users, promising to streamline credit assessment processes and elevate decision-making in the finance industry. The ultimate goal is to deliver a fully functional, user-friendly application that optimizes operational efficiency and opens avenues for tailored financial services based on precise creditworthiness categorization. In the digital era, this project represents a significant stride towards redefining credit assessment methodologies, embodying the fusion of data-driven insights and cutting-edge technology in the finance industry.

Exploratory Data Analysis

Graphs - visualizations

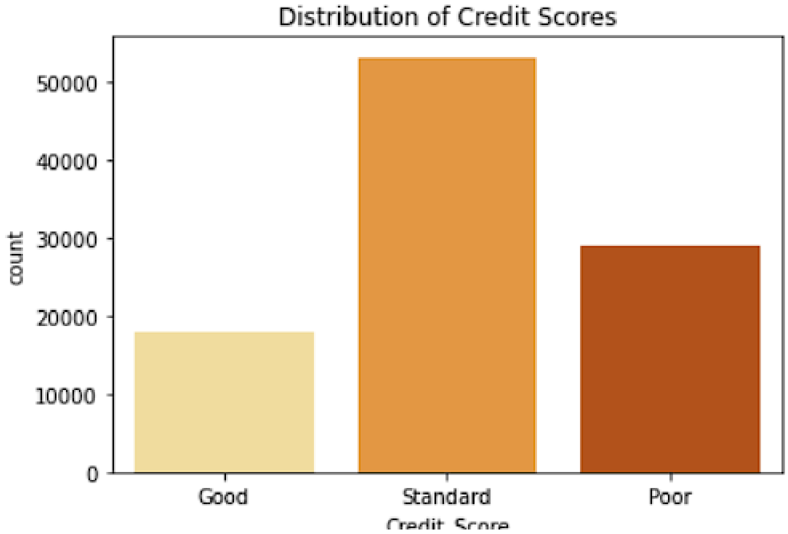


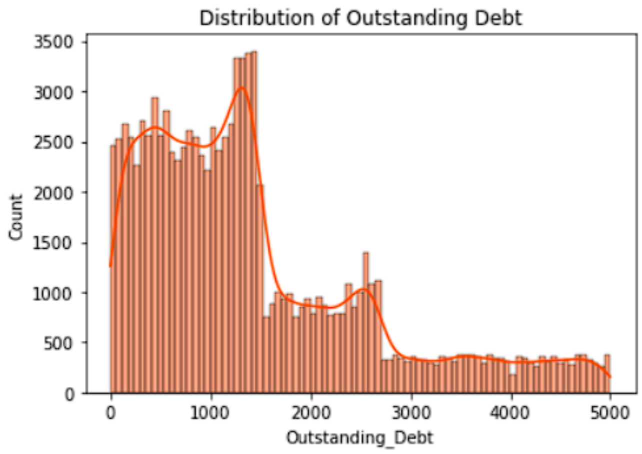
From the above graph, we can see that most of the customers are Journalist(8%).



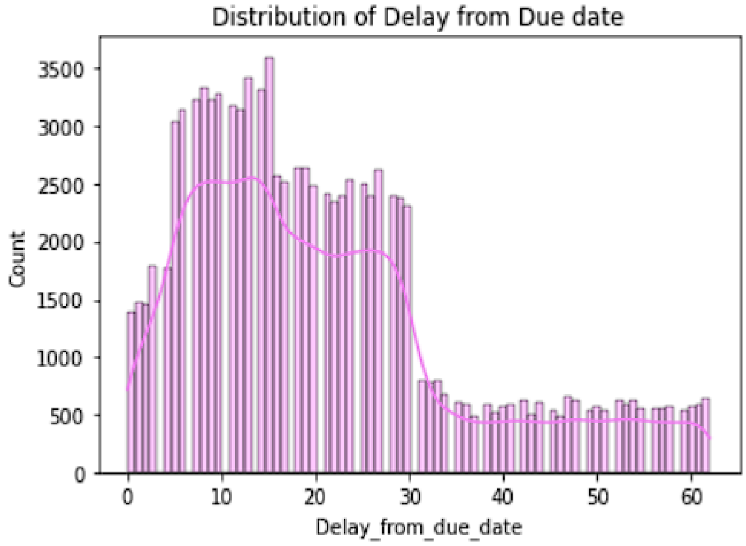
The inhand salary distribution is right-skewed. Mean > median. Most of the customers have inhand salary in the rang

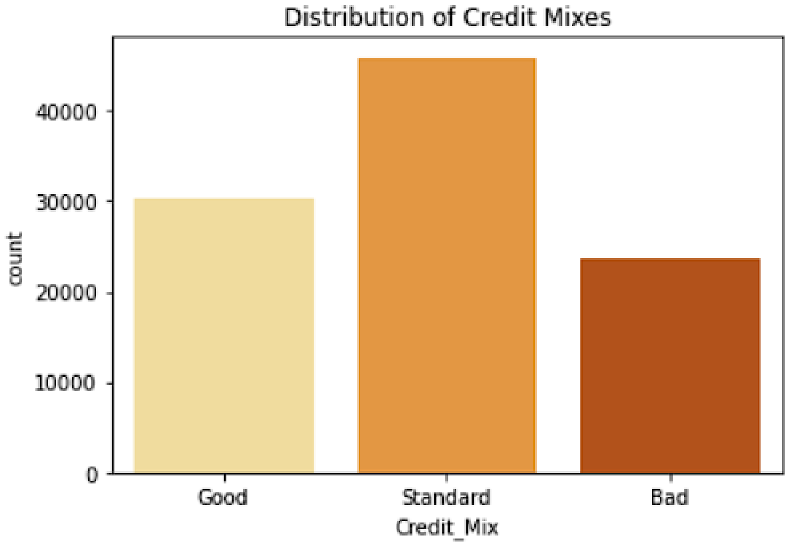
```
#Credit Scores
sns.countplot(x= credit_df1['Credit_Score'],palette="YlOrBr")
plt.title('Distribution of Credit Scores')
plt.show()
```





The distribution is skewed to the right. Most people have less amount of debt left to be paid.

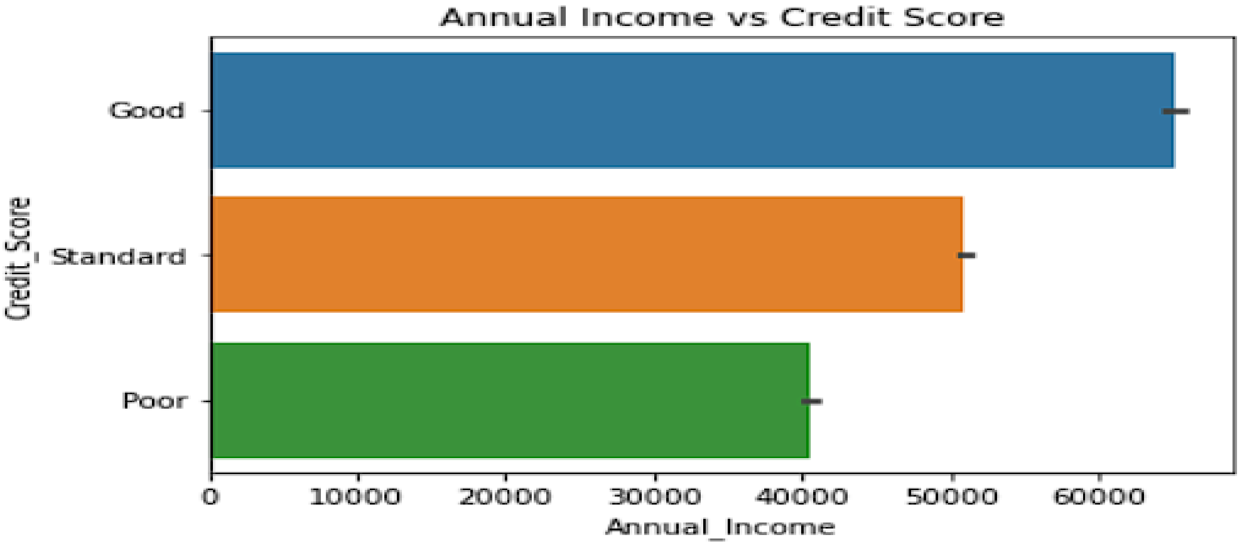




Majority of credit mix type is standard.

Bivariate Analysis

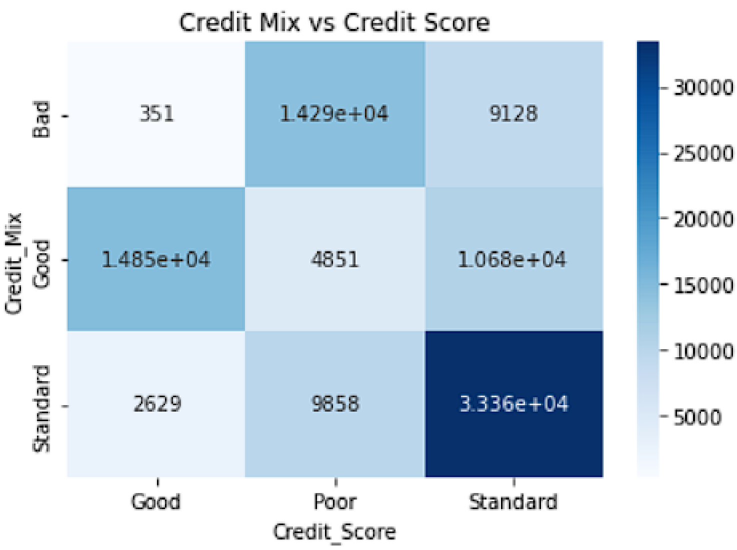
```
# Annual_Income vs credit score
sns.barplot(x=credit_df1['Annual_Income'], y=credit_df1['Credi
plt.title('Annual Income vs Credit Score')
plt.show()
```



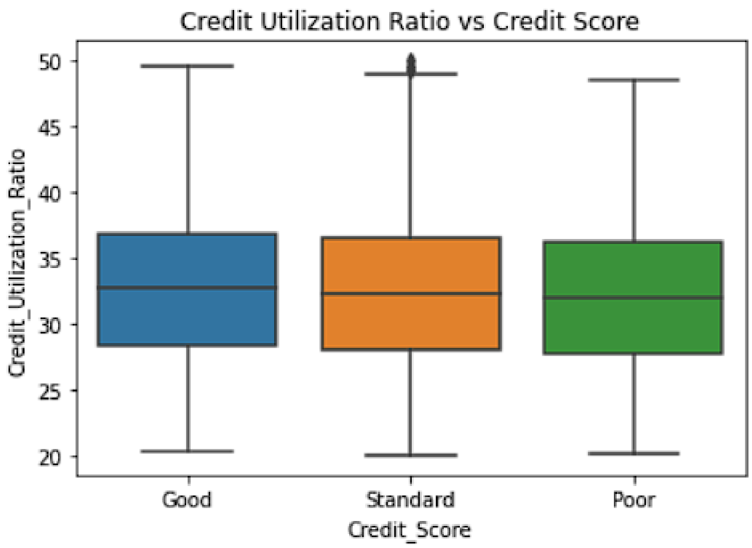
Customers with higher annual income tend to have better credit scores.

Monthly_Inhand_Salary

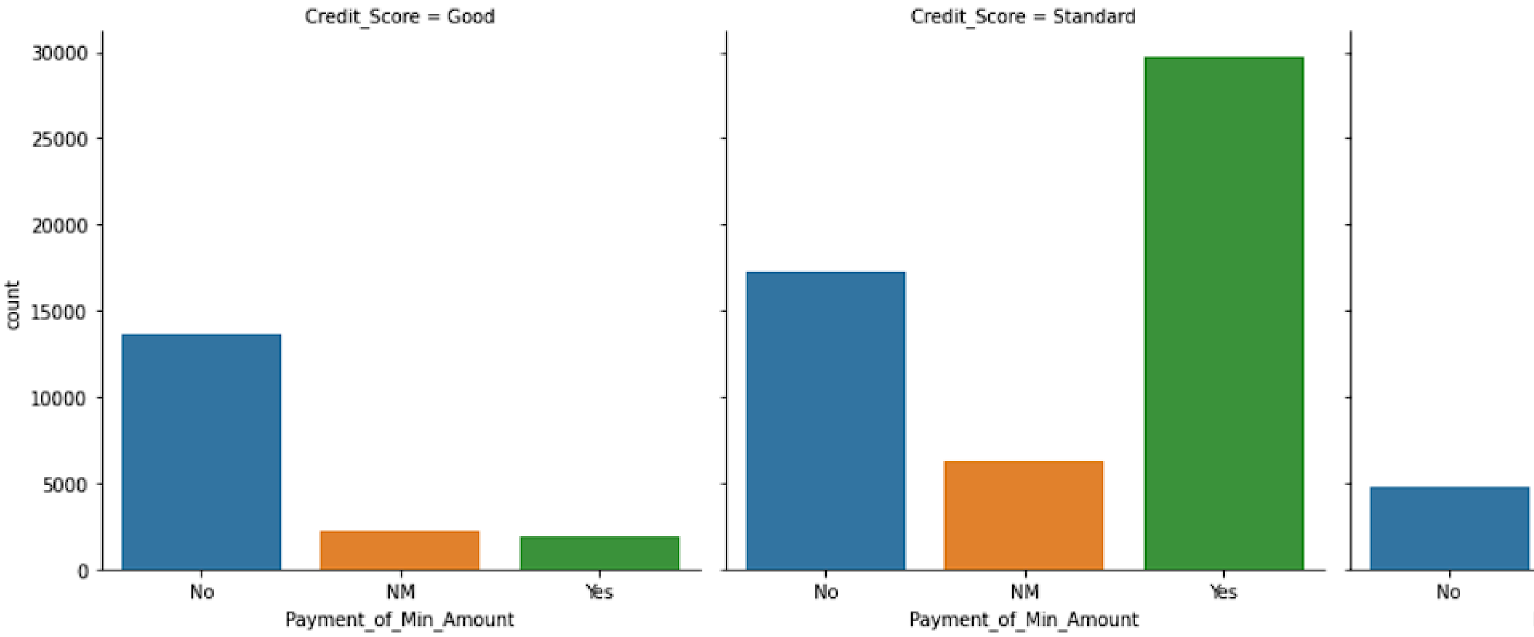
The distribution is right skewed(mean>median) for all three credit scorers.Customers with higher monthly inhand sal scores



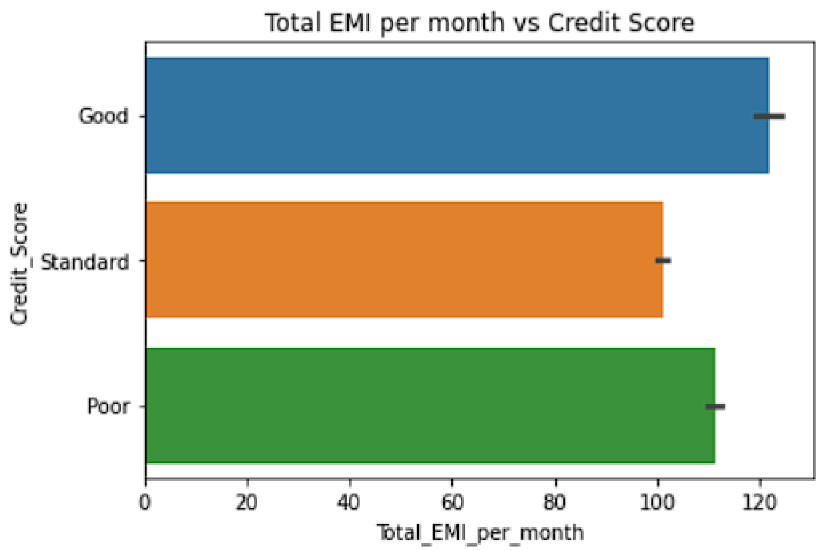
Customers with better credit mix, shows better credit scores.



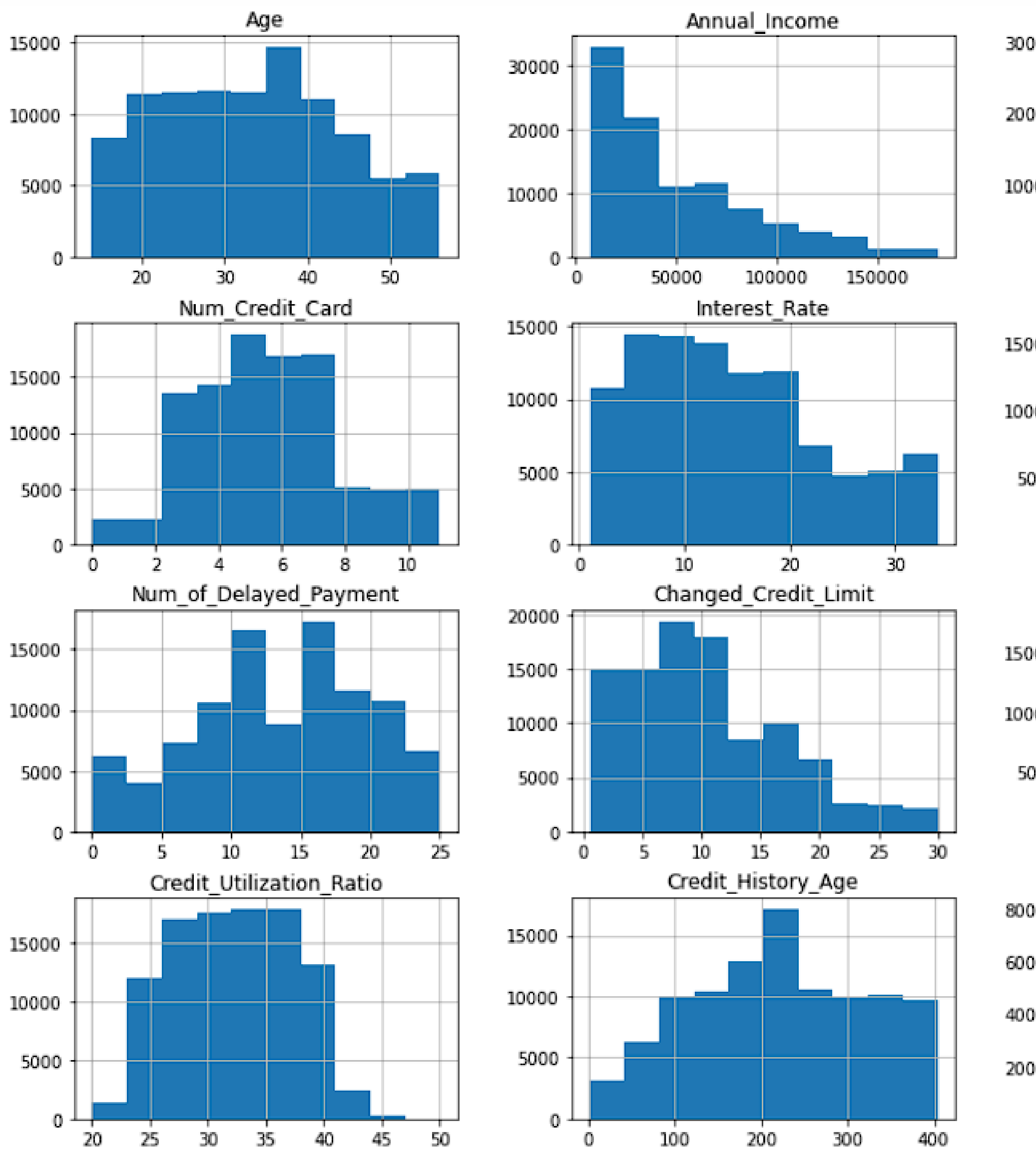
Credit Utilization ratio distribution is almost even for good and standard credit scorers.But, the range is slightly range increase in Credit Utilization ratio is good for better credit score. Beyond limit, credit utilization negatively



Most of the Customers with poor and standard credit scores did only the minimum payment. Most of the customers more than the minimum payment.From the above graphs, we can see that the most of the customers with a good cre minimum amount for the loan.

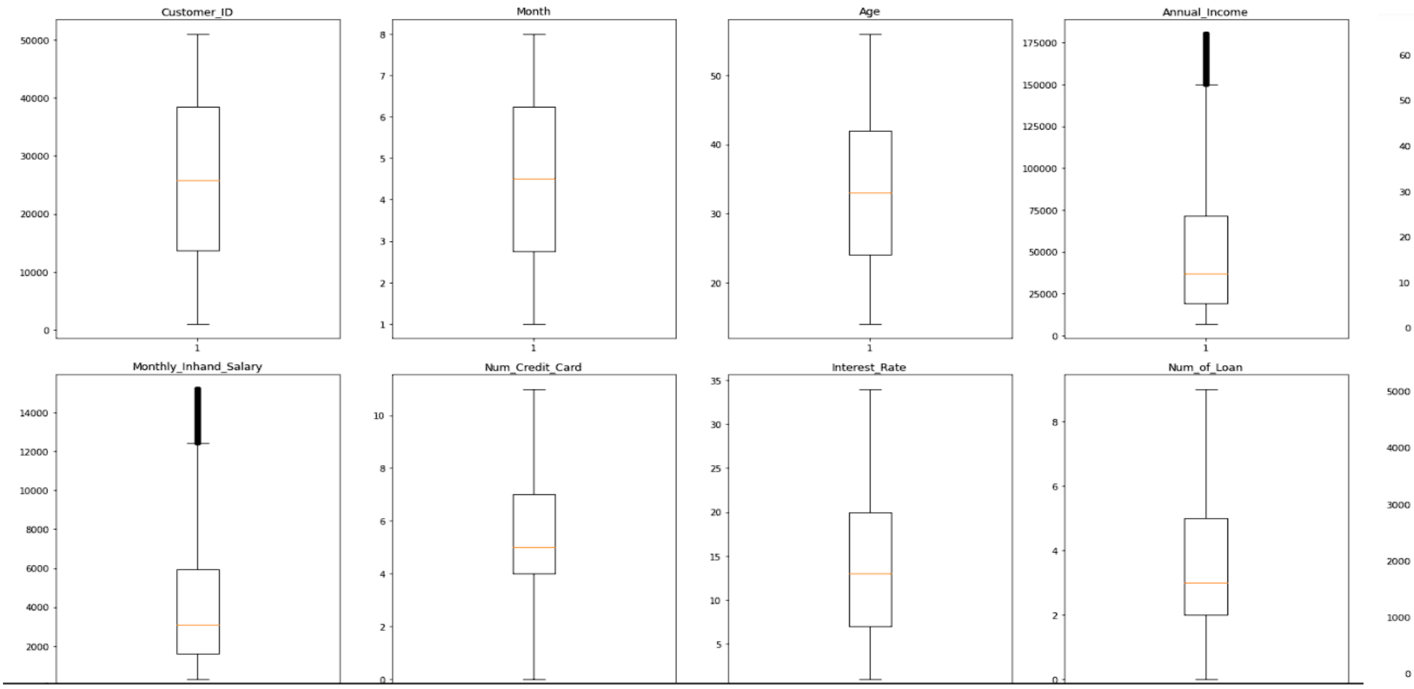


Customers that paid higher EMI tend to have better credit scores.



BOX PLOTS

A box plot, also known as a box-and-whisker plot, is a graphical representation of the distribution of a dataset. It provides a visual summary of key statistical measures, such as the median, quartiles, and potential outliers, in a concise and informative manner. Box plots are commonly used to display the spread and central tendency of a dataset, making it easier to understand its overall characteristics.



Data Preprocessing

1. FINDING MSSING VALUES

- **No missing values are present in this dataset**

In [78]:
```
#check for null values
```

In [79]:
```
credit_df.isna().sum()
```

Out[79]:
```
ID                          0
Customer_ID                 0
Month                       0
Name                        0
Age                         0
SSN                         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
Type_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
```

2. SCALING AND ENCODING

STANDARD SCALING

In [141]:
```
scale_cols = ['Age',  'Annual_Income', 'Monthly_Inhand_Salary',
        'Num_Credit_Card', 'Interest_Rate', 'Num_of_Loan',
        'Delay_from_due_date', 'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
        'Num_Credit_Inquiries', 'Outstanding_Debt', 'Credit_Utilization_Ratio','Credit_History_Age',
        'Total_EMI_per_month', 'Amount_invested_monthly', 'Monthly_Balance']

from sklearn.preprocessing import StandardScaler

std = StandardScaler()

credit_df1[scale_cols]= std.fit_transform(credit_df1[scale_cols])
```

In [142]:
```
credit_df1.columns
```

Out[142]:
```
Index(['Month', 'Age', 'Occupation', 'Annual_Income', 'Monthly_Inhand_Salary',
       'Num_Credit_Card', 'Interest_Rate', 'Num_of_Loan',
       'Delay_from_due_date', 'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
       'Num_Credit_Inquiries', 'Outstanding_Debt', 'Credit_Utilization_Ratio',
       'Credit_History_Age', 'Payment_of_Min_Amount', 'Total_EMI_per_month',
       'Amount_invested_monthly', 'Monthly_Balance', 'Credit_Score'],
      dtype='object')
```

In [143]:
```
credit_df1
```

LABEL ENCODING

```
In [132]:
from sklearn.preprocessing import LabelEncoder
label_encoder=LabelEncoder()

for i in ['Occupation']:
    credit_df1[i]=label_encoder.fit_transform(credit_df1[i])
    le_name_mapping =dict((zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_))))
    print(le_name_mapping)
```

```
{'Accountant': 0, 'Architect': 1, 'Developer': 2, 'Doctor': 3, 'Engineer': 4, 'Entrepreneur': 5, 'Journalist': 6, 'Lawyer
'Manager': 8, 'Mechanic': 9, 'Media_Manager': 10, 'Musician': 11, 'Scientist': 12, 'Teacher': 13, 'Writer': 14}
```

```
In [133]:
# It also prints a dictionary that maps the original categorical values to their corresponding numerical codes
#It creates a le_name_mapping dictionary that contains the mapping of original categorical values to their correspond

#It prints the le_name_mapping dictionary, which shows the mapping for each variable separately.
```

```
In [ ]:
```

```
In [134]:
# drop Credit Score
credit_df1=credit_df1.drop('Credit_Mix',axis=1)
```

```
In [135]:
# Mapping Credit score

replace_map = {'Credit_Score': {'Poor': 0, 'Good': 2, 'Standard': 1 }}
```

```
In [302]:
print(credit_df1['Credit_Score'].unique())
```

```
[2 1 0]
```

3. FEATURE SELECTION

```
In [145]:
credit_df1['Credit_Score'].value_counts()
```

```
Out[145]:
1    53174
0    28998
2    17828
Name: Credit_Score, dtype: int64
```

```
In [ ]:
```

```
In [148]:
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

```
In [149]:
# Select features and target
X = credit_df1.drop('Credit_Score', axis=1)  # Features
y = credit_df1['Credit_Score']  # Target
```

```
In [150]:
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
In [151]:
print(len(X_train))
print(len(X_test))
print(len(y_train))
print(len(y_test))
```

```
80000
20000
80000
20000
```

4.MODEL TRAINING

LOGISTIC REGRESSION

In [221]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()

model = LogisticRegression(max_iter=1000)  # Increase max_iter value
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

```

In [222]:
```
# Calculate accuracy
accuracy_lr = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_lr)
```

```
Accuracy: 0.6432
```

In [223]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.67      0.53      0.60      5874
           1       0.65      0.77      0.70     10599
           2       0.57      0.46      0.51      3527

    accuracy                           0.64     20000
   macro avg       0.63      0.59      0.60     20000
weighted avg       0.64      0.64      0.64     20000
```

I got accuracy of 0.64 in logistic Regression

KNN

In [224]:
```
# Initialize the KNN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)
```

In [225]:
```
# Train the KNN classifier
knn_classifier.fit(X_train,y_train)
```

Out[225]:
```
KNeighborsClassifier()
```

In [226]:
```
# Predictions on the test set
y_pred = knn_classifier.predict(X_test)
```

In [227]:
```
# Calculate accuracy
accuracy_knn = accuracy_score(y_test, y_pred)
```

In [228]:
```
print("Accuracy:", accuracy_knn)
```

```
Accuracy: 0.7044
```

In [229]:
```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
cm
```

Out[229]:
```
array([[4157, 1545,  172],
       [1562, 8167,  870],
       [ 278, 1485, 1764]], dtype=int64)
```

Got accuracy of 0.70 in KNN

DECISION TREE

```
In [231]: from sklearn.tree import DecisionTreeClassifier
          from sklearn.metrics import accuracy_score

In [232]: decision_tree = DecisionTreeClassifier(random_state=21)

In [233]: decision_tree.fit(X_train, y_train)

Out[233]: DecisionTreeClassifier(random_state=21)

In [234]: y_pred = decision_tree.predict(X_test)

In [235]: accuracy_dt = accuracy_score(y_test, y_pred)
          print("Accuracy:", accuracy_dt)

Accuracy: 0.75635

In [236]: from sklearn.metrics import classification_report
          from sklearn.metrics import confusion_matrix

          print(classification_report(y_test, y_pred))

              precision    recall  f1-score   support

           0       0.75      0.74      0.75      5874
           1       0.78      0.79      0.78     10599
           2       0.71      0.69      0.70      3527

    accuracy                           0.76     20000
   macro avg       0.74      0.74      0.74     20000
weighted avg       0.76      0.76      0.76     20000
```

got the accuracy of 0.75 in Decision Tree

RANDOM FOREST

In [237]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

In [238]:
```
random_forest = RandomForestClassifier(random_state=21)
random_forest.fit(X_train, y_train)
```

Out[238]:
```
RandomForestClassifier(random_state=21)
```

In [239]:
```
y_pred = random_forest.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred)
```

In [242]:
```
accuracy_rf
```

Out[242]:
```
0.8335
```

In [243]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.87      0.84      5874
           1       0.85      0.83      0.84     10599
           2       0.80      0.79      0.79      3527

    accuracy                           0.83     20000
   macro avg       0.82      0.83      0.83     20000
weighted avg       0.83      0.83      0.83     20000
```

Got accuracy of 0.83 in Random Forest

In [244]:
```
from sklearn.svm import SVC # "Support vector classifier"
classifier = SVC(kernel='rbf', random_state=0)
classifier.fit(X_train, y_train)
```

Out[244]:
```
SVC(random_state=0)
```

In [245]:
```
#Predicting the test set result
y_pred= classifier.predict(X_test)
```

In [246]:
```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix ,classification_report
cm= confusion_matrix(y_test, y_pred)
clr = classification_report(y_test, y_pred)
print(clr)
```

```
              precision    recall  f1-score   support

           0       0.72      0.60      0.66      5874
           1       0.67      0.80      0.73     10599
           2       0.61      0.42      0.50      3527

    accuracy                           0.67     20000
   macro avg       0.67      0.61      0.63     20000
weighted avg       0.67      0.67      0.67     20000
```

In [247]:
```
accuracy_svm = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_svm)
```

```
Accuracy: 0.6733
```

Got accuracy of 0.67 in SVM

In [264]:
```python
import xgboost as xgb
```

In [265]:
```python
xgboost = xgb.XGBClassifier(learning_rate=0.1,max_depth=5,n_estimators=100, num_cl
xgboost.fit(X_train, y_train)
```

Out[265]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, 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=5, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=100, n_jobs=None, num_class=3,
              num_parallel_tree=None, ...)
```

In [266]:
```python
xgb_pred = xgboost.predict(X_test)
```

In [267]:
```python
print(classification_report(y_test, xgb_pred))
```

```
              precision    recall  f1-score   support

           0       0.75      0.64      0.69      5874
           1       0.72      0.78      0.75     10599
           2       0.60      0.58      0.59      3527

    accuracy                           0.71     20000
   macro avg       0.69      0.67      0.68     20000
weighted avg       0.71      0.71      0.70     20000
```

In [268]:
```python
accuracy_xgb = accuracy_score(y_test, xgb_pred)
print("Accuracy:", accuracy_xgb)
```

```
Accuracy: 0.7054
```

Got Accuracy of 0.70 in XGBoost

CATBOOST

In [269]:
```
from catboost import CatBoostClassifier
```

In [270]:
```
catboost = CatBoostClassifier(random_state=42,classes_count=3, verbose=False)
catboost.fit(X_train, y_train)
cat_pred = catboost.predict(X_test)
cat_score = catboost.score(X_test, y_test)
```

In [271]:
```
print(classification_report(y_test, cat_pred))
```

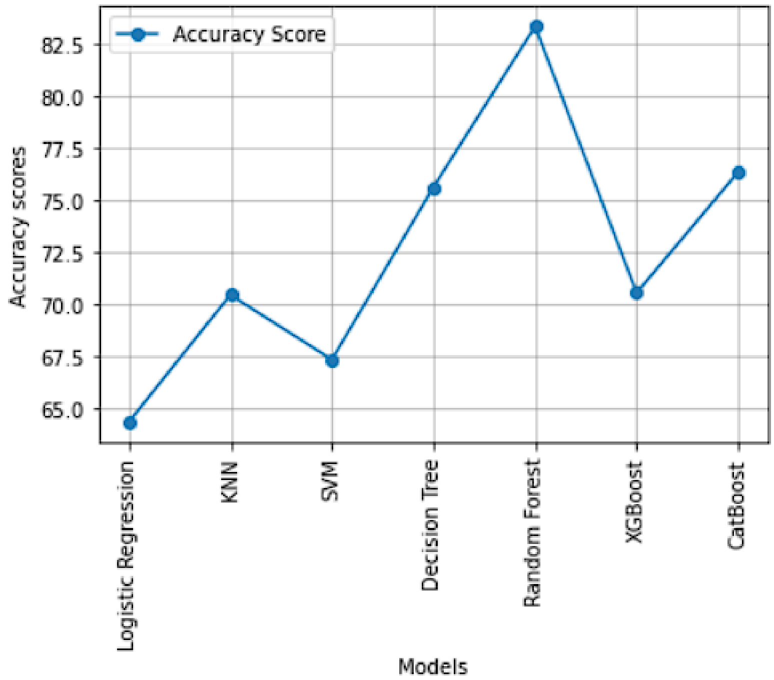
```
              precision    recall  f1-score   support

           0       0.78      0.74      0.76      5874
           1       0.77      0.81      0.79     10599
           2       0.73      0.65      0.69      3527

    accuracy                           0.76     20000
   macro avg       0.76      0.74      0.75     20000
weighted avg       0.76      0.76      0.76     20000
```

In [205]:
```
accuracy = accuracy_score(y_test, cat_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7636
```

Got accuracy of 0.76

	Accuracy Score
Random Forest	83.350
CatBoost	76.360
Decision Tree	75.635
XGBoost	70.540
KNN	70.440
SVM	67.330
Logistic Regression	64.320



Stratified K-Fold Cross Validation

```

from sklearn.model_selection import StratifiedKFold, cross_val_score

# Create a list of models to evaluate
model_names = ['Logistic Regression','KNN','SVM','Decision Tree','Random Forest', 'XGBoost','CatBoost']
models = [model, knn_classifier, classifier, decision_tree, random_forest, xgboost,catboost]
mean_scores = []

# Split the data into K folds, ensuring that the distribution of classes is the same in each fold
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Use a for loop to iterate through the models
for model in models:
    # Calculate the cross-validated accuracy using the `cross_val_score` function
    scores = cross_val_score(model, X_pca, y, cv=kfold, scoring='accuracy')
    mean_scores.append(scores.mean())

    # Print the mean and standard deviation of the scores for the current model
    print(f'{model.__class__.__name__}: {scores.mean():.2f} ')
```

```
LogisticRegression: 0.64
KNeighborsClassifier: 0.68
SVC: 0.67
DecisionTreeClassifier: 0.61
RandomForestClassifier: 0.72
XGBClassifier: 0.68
CatBoostClassifier: 0.70
```

```
# RandomForest: 72 %
```

Hyperparameter Tuning

Grid search

Random Forest

```
#grid search on Random Forest

from sklearn.model_selection import GridSearchCV


param_grid = {'n_estimators': [50, 100],
              'max_depth': [4, 8],
              'min_samples_leaf': [1, 2]}

rf = RandomForestClassifier()

grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy')

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
best_model = grid_search.best_estimator_
rf_score = best_model.score(X_test , y_test)
print(f"Best model:{best_model},accuracy is :{rf_score*100:.2f}%")
```

```
Best model:RandomForestClassifier(max_depth=8, min_samples_leaf=2, n_estimators=50),accuracy is :69.58%
```

XGBoost

[293]:

```
# Define the hyperparameters
param_grid = {'learning_rate': [0.1, 0.5, 1],
              'max_depth': [2, 4, 6, 8]}

# Create the XGBoost classifier
xgboost = xgb.XGBClassifier(num_class=3)

# Set up the grid search
grid_search = GridSearchCV(xgboost, param_grid, cv=5, scoring='accuracy')

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
best_model = grid_search.best_estimator_
xgb_score = best_model.score(X_test , y_test)
print(f"Best model:{best_model},accuracy is :{xgb_score*100:.2f}%")
```

```
Best model:XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.5, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=8, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None, num_class=3,
              num_parallel_tree=None, ...),accuracy is :81.89%
```

CatBoost

[294]:

```
# Define the hyperparameters
param_grid = {'depth': [3, 6, 9],'n_estimators':[100, 200, 300]}

# Create the CatBoost classifier
catboost = CatBoostClassifier(verbose=False)

# Set up the grid search
grid_search = GridSearchCV(catboost, param_grid, cv=5, scoring='accuracy')

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
best_model = grid_search.best_estimator_
cat_score = best_model.score(X_test , y_test)
print(f"Best model:{best_model},accuracy is :{cat_score*100:.2f}%")
print(f"Best hyperparameters: {grid_search.best_params_}")
```

```
Best model:<catboost.core.CatBoostClassifier object at 0x000002BBA41DBA00>,accuracy is :79.98%
Best hyperparameters: {'depth': 9, 'n_estimators': 300}
```

```python
# evaluate the models
models = [ 'Random Forest', 'XGBoost','CatBoost']
data = [rf_score*100, xgb_score*100,cat_score*100]
cols = ['Accuracy Score']
pd.DataFrame(data=data , index= models , columns= cols).sort_values(by=['Accuracy Sc
```

	Accuracy Score
XGBoost	81.895
CatBoost	79.985
Random Forest	69.580

XGBoost have high accuracy of 82%

XGBoost:

Achieved the highest accuracy after hyperparameter tuning.

Considered a strong candidate for the final model due to its top performance

Random Forest model might have been overfitting the training data before hyperparameter tuning

DEPLOYING FLASK APP

app.py

```python
"""
Credit Score Classification-Web Application

@author: Arshad Arif
"""
# import necessary libraries
import numpy as np
import pandas as pd
from flask import Flask,request,render_template
import pickle


# create an object app taking current module as argument
app = Flask(__name__)
# load the pickled file
model = pickle.load(open('xgboost.pkl','rb'))
X_train = pickle.load(open('X_train.pkl','rb'))

#decorator to route to main page
@app.route("/")
def home():
    return render_template('index.html')#returns the home page

# decorator to route to prediction page
@app.route("/predict", methods=['POST'])
def predict():
    input_features = [float(x) for x in request.form.values()]
    final_features = pd.DataFrame(data=[input_features], columns=X_train.columns)
    prediction = model.predict(final_features)
    if prediction == 0:
            result = 'Bad'
    elif prediction == 1:
            result = 'Standard'
    elif prediction == 2:
            result = 'Good'

    #returns home page with the prediction
    return render_template('index.html', prediction_text='Your Credit Score is {}!'.format(result))

# to run
if __name__ == "__main__":
    app.run(debug=True)
```

Model.py

Welcome | app.py | xgboost.pkl | index.html | model.py | X_train.pkl

model.py > ...

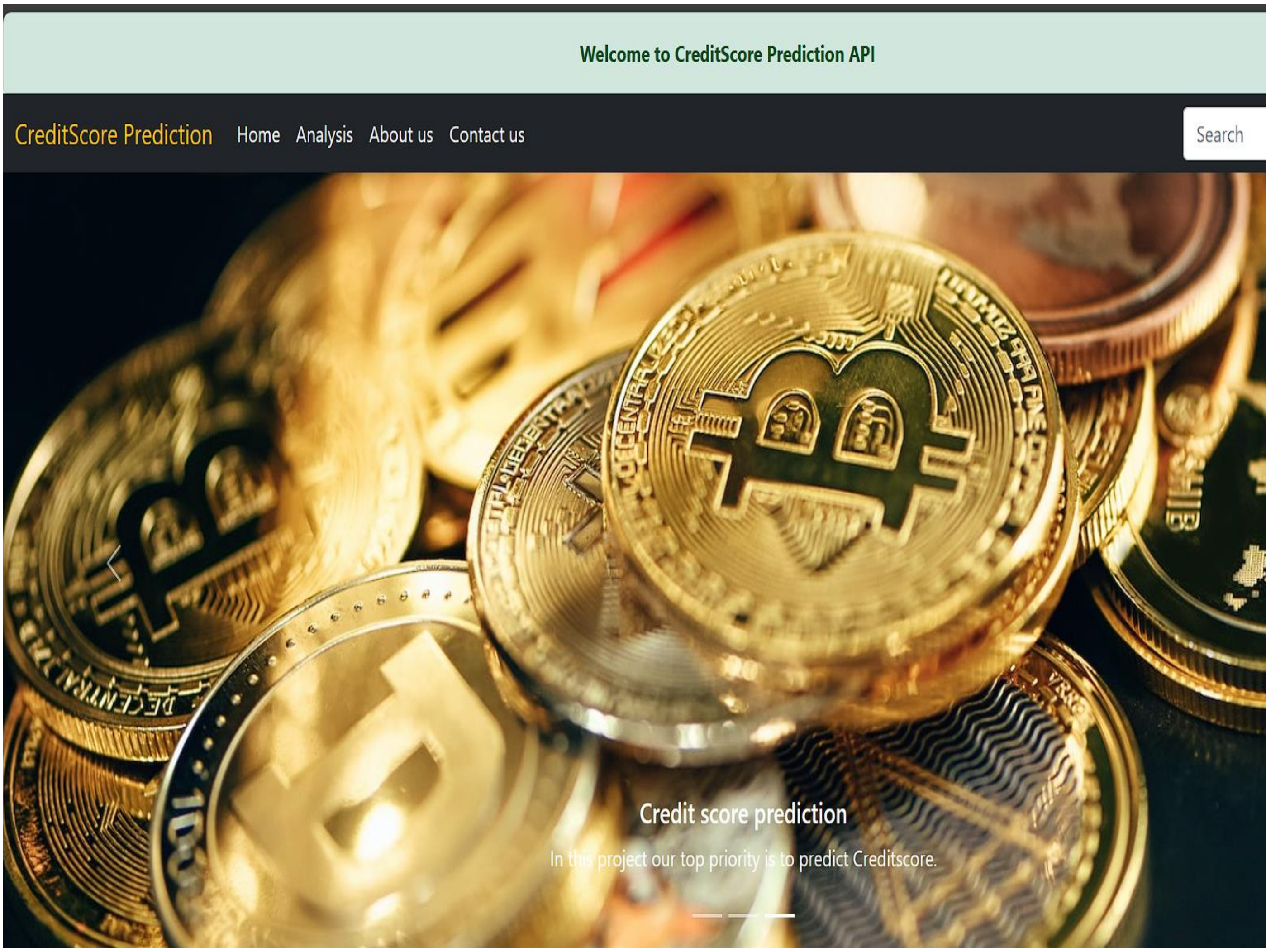
```python
# Importing the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


### Remove unnecessary warnings
import warnings
warnings.filterwarnings('ignore')

# load  data
credit_df = pd.read_csv("credit.csv",low_memory=False)

# can drop ID, Name and SSN as they are identifiers and not useful for visualization
credit_df1 = credit_df.drop(['ID', 'Name', 'SSN'], axis=1)

#  drop, Num_Bank_Accounts, Type_of_Loan as they do not contribute to target
credit_df1.drop(['Num_Bank_Accounts','Type_of_Loan'],axis =1 , inplace = True )

# replace NM with nan
credit_df1.loc[credit_df1['Payment_of_Min_Amount'] == 'NM', 'Payment_of_Min_Amount'] = np.nan

# fill nan using mode within each group
credit_df1['Payment_of_Min_Amount'] = credit_df1.groupby('Customer_ID')['Payment_of_Min_Amount'].transform(lambda x: x.fillna(x.mode()[0]))



#Box plot of numerical columns

num_col = credit_df1.select_dtypes(include=np.number).columns.tolist()

plt.figure(figsize=(20,30))

for i, variable in enumerate(num_col):
                    plt.subplot(5,4,i+1)
                    plt.boxplot(credit_df1[variable],whis=1.5)
                    plt.tight_layout()
                    plt.title(variable)

# drop the other columns
num_col1=credit_df1.drop(['Customer_ID','Month','Age','Num_Credit_Card','Interest_Rate','Num_of_Loan','Num_of_Delayed_Payment','Credit_History_Age','Credit_Score','Payment_
```

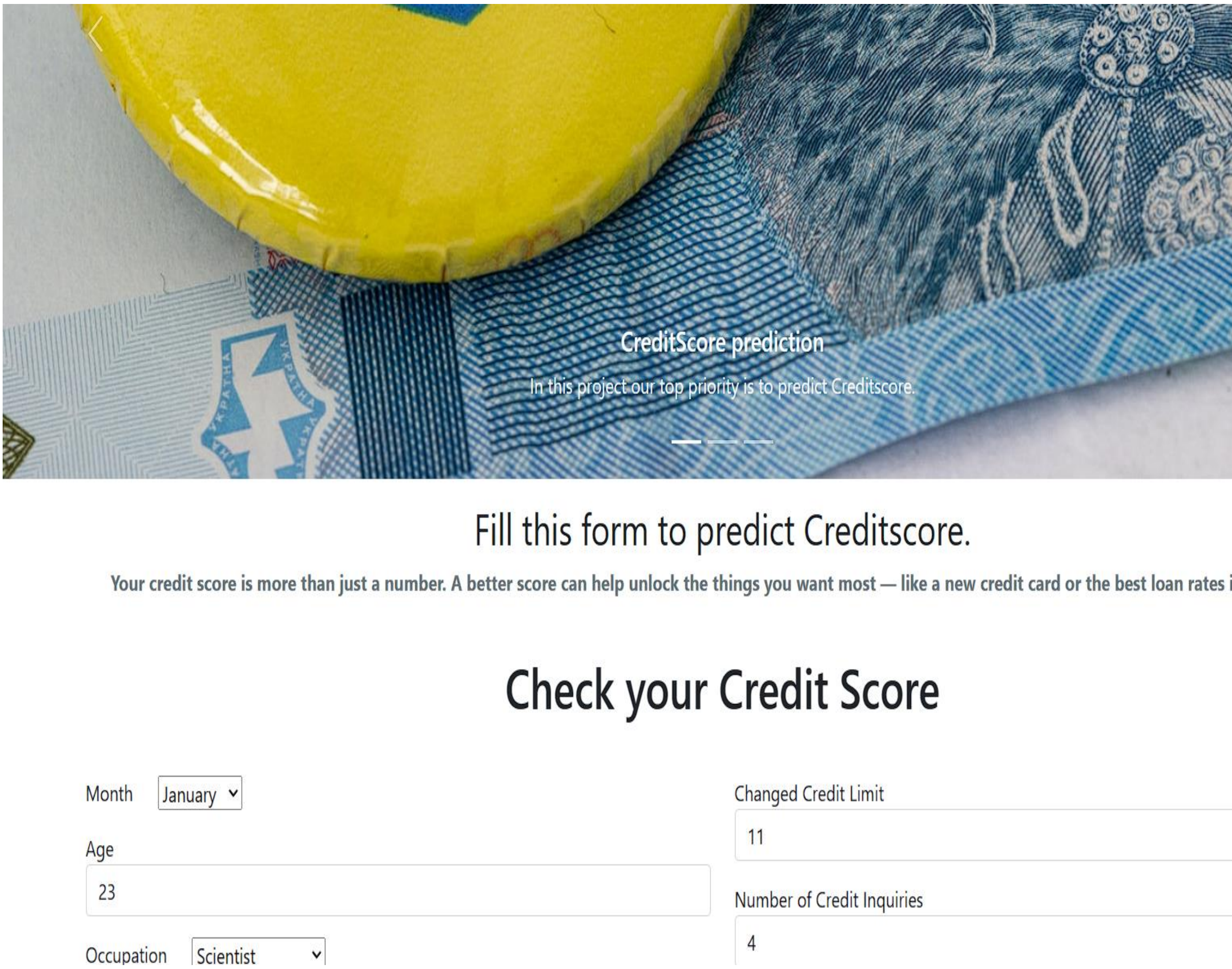
```python
# Identify the outliers and remove

for i in num_col1:
    Q1=credit_df1[i].quantile(0.25) # 25th quantile
    Q3=credit_df1[i].quantile(0.75) # 75th quantile
    IQR = Q3-Q1
    Lower_Whisker = Q1 - 1.5*IQR
    Upper_Whisker = Q3 + 1.5*IQR
    credit_df1[i] = np.clip(credit_df1[i], Lower_Whisker, Upper_Whisker)

# drop Customer ID
credit_df1 = credit_df1.drop('Customer_ID', axis=1)

# drop payment behaviour
credit_df1 = credit_df1.drop('Payment_Behaviour', axis=1)

# ONE HOT ENCODING
replace_map = {'Payment_of_Min_Amount': {'Yes': 1, 'No': 0} }

credit_df1.replace(replace_map, inplace=True)

   # It's replacing 'Yes' with 1 and 'No' with 0 in that specific column

# LABEL ENCODING
from sklearn.preprocessing import LabelEncoder
label_encoder=LabelEncoder()

for i in ['Occupation']:
    credit_df1[i]=label_encoder.fit_transform(credit_df1[i])
    le_name_mapping =dict((zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_))))
    print(le_name_mapping)


credit_df1=credit_df1.drop('Credit_Mix',axis=1)

# Mapping Credit score
replace_map = {'Credit_Score': {'Poor': 0, 'Good': 2, 'Standard': 1 }}
credit_df1.replace(replace_map, inplace=True)
```

Welcome | app.py | xgboost.pkl | index.html | model.py | X_train.pkl

model.py > ...

```python

#credit_df1[scale_cols]= std.fit_transform(credit_df1[scale_cols])


from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Select features and target
X = credit_df1.drop('Credit_Score', axis=1)  # Features
y = credit_df1['Credit_Score']  # Target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,stratify=y)

print(len(X_train))
print(len(X_test))
print(len(y_train))
print(len(y_test))

from sklearn.metrics import classification_report

import xgboost as xgb
xgboost = xgb.XGBClassifier(learning_rate=0.1,max_depth=5,n_estimators=100, num_class=3)
xgboost.fit(X_train, y_train)
xgb_pred = xgboost.predict(X_test)
print(classification_report(y_test, xgb_pred))



#Serialize the python object using pickle
import pickle
pickle.dump(xgboost, open('xgboost.pkl', 'wb'))

pickle.dump(X_train, open('X_train.pkl', 'wb'))

print(xgboost.predict(X_test))


print(X_train.columns)
```


Welcome to CreditScore Prediction API
CreditScore Prediction
Home
Analysis
About us
Contact us
Search
Credit score prediction
In this project our top priority is to predict Creditscore.


CreditScore prediction
In this project our top priority is to predict Creditscore.
Fill this form to predict Creditscore.
Your credit score is more than just a number. A better score can help unlock the things you want most — like a new credit card or the best loan rates
Check your Credit Score
Month
January
Age
23
Occupation
Scientist
Changed Credit Limit
11
Number of Credit Inquiries
4

Annual Income

19114

Monthly Inhand Salary

1824

Number of Credit Cards

4

Interest Rate

3

Number of Loans

4

Delay from due date

3

Number of Delayed Payments

7

Outstanding Debt

809

Credit Utilization ratio

26

Credit History Age

265

Payment of Minimum Amount

○ Yes

◉ No

Total EMI per month

45

Amount Invested monthly

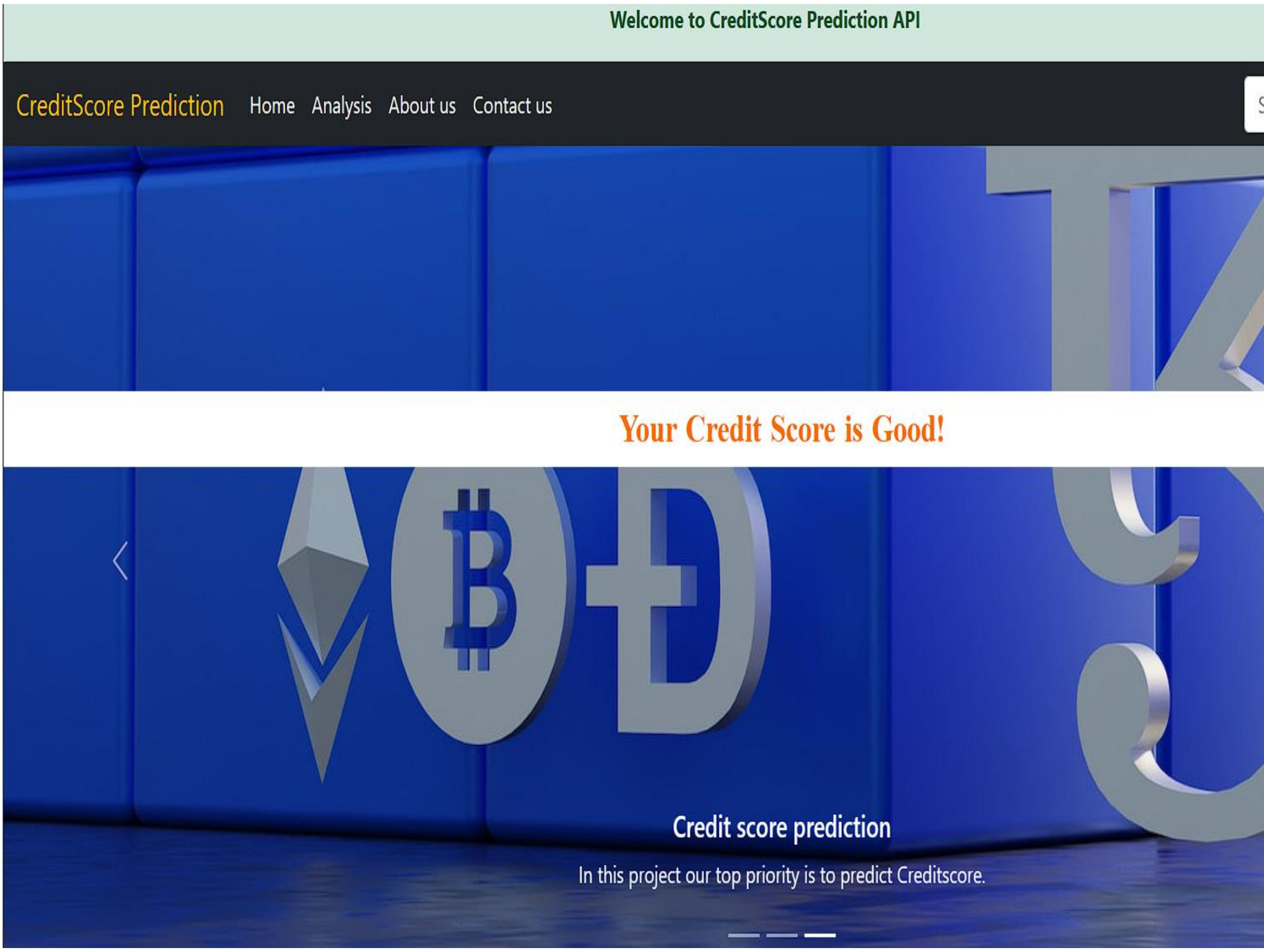
21

Monthly Balance

312

Check

RESULT


Welcome to CreditScore Prediction API
CreditScore Prediction
Home
Analysis
About us
Contact us
Your Credit Score is Good!
Credit score prediction
In this project our top priority is to predict Creditscore.

Conclusion

In culmination, the Credit Score Prediction Web Application project has successfully navigated the intricacies of data science and machine learning to deliver a robust and intelligent system. The primary goal was to automate the categorization of individuals into specific creditworthiness levels—Good, Standard, and Poor—providing a valuable tool for banks and financial institutions.

The journey began with a comprehensive exploration of the dataset, encompassing essential banking details and extensive credit-related information. Through meticulous data preprocessing and feature engineering, the project laid a solid foundation for the subsequent phases.

Several machine learning models were considered, and after thorough evaluation and Stratified K-Fold Cross Validation, XGBoost emerged as a frontrunner, exhibiting an accuracy of 70.54%. Subsequent hyperparameter tuning through grid search propelled XGBoost to an impressive accuracy of 81.895%, solidifying its position as the top-performing model.

In parallel, the project leveraged Flask to deploy the predictive model into an intuitive and user-friendly web application. This application, accessible to banks and financial institutions, enables users to input relevant data and receive instant credit score classifications. The integration of Flask provides a seamless and efficient platform for real-world applications.

Despite Random Forest experiencing a reduction in accuracy post-hyperparameter tuning, the iterative nature of the project allowed for valuable insights into model performance and the impact of tuning decisions.

In essence, this project represents a pivotal stride towards redefining credit assessment methodologies in the finance industry. The fusion of data-driven insights, cutting-edge technology, and an interactive web application marks a significant contribution to the digitization of creditworthiness evaluation. As the finance industry embraces the digital era, this project serves as a testament to the transformative power of data science in optimizing operational processes and enabling more tailored financial services. The journey doesn't conclude here but rather opens avenues for further enhancements, continuous improvement, and a future where intelligent systems redefine the landscape of credit assessment.