BIA | BOSTON INSTITUTE OF ANALYTICS ®

SPINNAKER ANALYTICS

CREDIT SCORE CLASSIFICATION

Credit Score Classification

Objective :

The main goal of this project is to develop a robust machine learning model that can accurately predict credit scores of applicants based on various financial and personal attributes.

Data Preprocessing :

Handle Missing Values: Address any missing or incomplete data entries.

Encode Categorical Variables: Convert categorical variables (such as 'gender', 'education') into numerical representations.

Feature scaling (if needed for some models)

Business Impact :

- **Efficiency:** Automates credit scoring, reducing manual effort and decision time.Allows faster processing of loan applications.
- **Consistency:** Ensures uniform application of credit assessment criteria.Minimizes human bias in decision-making.
- **Risk Management:** Improves the accuracy of credit risk assessments.Helps reduce defaults and non-performing loans.
- **Customer Experience:** Enables quicker credit decisions.Enhances customer satisfaction and trust in the financial institution.

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

Introduction

In today’s financial landscape, accurate and efficient credit risk assessment is critical for banks, lenders, and other financial institutions to make informed decisions regarding loan approvals, credit limits, and other financial products. The traditional methods of evaluating creditworthiness involve manual processes, which are often time-consuming, inconsistent, and prone to human error. Therefore, there is a growing need for data-driven approaches to automate the credit scoring process, making it faster, more consistent, and accurate.

This project focuses on developing a machine learning model designed to predict an applicant's credit score based on various financial and personal attributes. The aim is to automate the credit score classification process, helping financial institutions streamline their decision-making and reduce the risk of defaults. By leveraging historical data and advanced machine learning techniques, such as Random Forest and XGBoost, this project seeks to provide more accurate and reliable credit score predictions.

Through exploratory data analysis (EDA), feature engineering, and model evaluation, this project will also explore the most important factors influencing credit scores, offering valuable insights that can inform better credit risk management practices. Ultimately, the model will improve efficiency, consistency, and customer satisfaction by enabling faster and more reliable credit decisions.

By automating the credit risk assessment process and providing deeper insights into credit factors, this model aims to support financial institutions in reducing operational costs, enhancing decision accuracy, and fostering a better customer experience.

THE WORK FLOW OF THE PROJECT IS GIVEN BELOW :

- Data Collection and Understanding
- Data Preprocessing
- Exploratory Data Analysis (EDA)
- Feature Engineering
- Model Selection
- Performance Evaluation
- Visualization
- Reporting
- Deployment

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

Project Goal

The main goal of this project is to build a machine learning model that can accurately classify individuals based on their credit score (e.g., Good, Standard, or Poor), using personal and financial data..

Key Goals:

- Automate credit score evaluation
- Make faster and more reliable lending decisions
- Reduce the risk of loan defaults
- Improve overall efficiency and customer experience in credit assessment processes

Project Plan

Define Objective

- Classify individuals into credit score categories (e.g., Good, Standard, Poor).
- Support financial institutions in making automated, reliable credit decisions.

Exploratory Data Analysis (EDA)

- Analyze class distribution (credit score labels).
- Visualize relationships between variables.
- Identify trends, patterns, and anomalies in the data.

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

Feature Engineering

- Create new relevant features (e.g., debt-to-income ratio).
- Perform feature selection to keep important predictors.
- Remove multicollinearity among features if needed.

Dataset Splitting

Train-Test Split: Divide the data into:

- Training Set (e.g., 70–80%): Used for model learning.
- Test Set (e.g., 20–30%): Used for evaluating model performance.

Model Development

- Train baseline model(s) (e.g., Logistic Regression) for benchmark.
- Train advanced models (Random Forest, XGBoost, etc.).
- Use consistent evaluation metrics across models.

Model Development

- Evaluate models with metrics: Accuracy, Precision, Recall, F1-score, ROC-AUC.
- Create confusion matrices to inspect classification errors.
- Compare performance of all candidate models.

Importing Important Libraries

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier   # pip install xgboost if not already
```

Data Overview

[3]:
```
df.head()
```

[3]:

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	...	Credit_Mix	Outstanding_Debt	C
0	0x1602	CUS_0xd40	January	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	1824.843333	3	...	_	809.98	
1	0x1603	CUS_0xd40	February	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	NaN	3	...	Good	809.98	
2	0x1604	CUS_0xd40	March	Aaron Maashoh	-500	821-00-0265	Scientist	19114.12	NaN	3	...	Good	809.98	
3	0x1605	CUS_0xd40	April	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	NaN	3	...	Good	809.98	
4	0x1606	CUS_0xd40	May	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	1824.843333	3	...	Good	809.98	

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

```
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
```

Data Summary

- Total records: 100,000
- Total columns: 28

Data types :

- float64: 4 columns
- int64: 4 columns
- object: 20 columns

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

Find null Values .

```
df.isna().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
```

Remove null Values .

```
df.dropna(inplace=True)
```

```
df.isna().sum()
```

```
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
```

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

Exploratory Data Analysis (EDA)

```
[8]: df.describe()
```

[8]:

	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Delay_from_due_date	Num_Credit_Inquiries	Credit_Utilization_Ratio	Total_EMI_per
count	84998.000000	100000.000000	100000.00000	100000.000000	100000.000000	98035.000000	100000.000000	100000
mean	4194.170850	17.091280	22.47443	72.466040	21.068780	27.754251	32.285173	1403
std	3183.686167	117.404834	129.05741	466.422621	14.860104	193.177339	5.116875	8306
min	303.645417	-1.000000	0.00000	1.000000	-5.000000	0.000000	20.000000	0
25%	1625.568229	3.000000	4.00000	8.000000	10.000000	3.000000	28.052567	30
50%	3093.745000	6.000000	5.00000	13.000000	18.000000	6.000000	32.305784	69
75%	5957.448333	7.000000	7.00000	20.000000	28.000000	9.000000	36.496663	161
max	15204.633333	1798.000000	1499.00000	5797.000000	67.000000	2597.000000	50.000000	82331

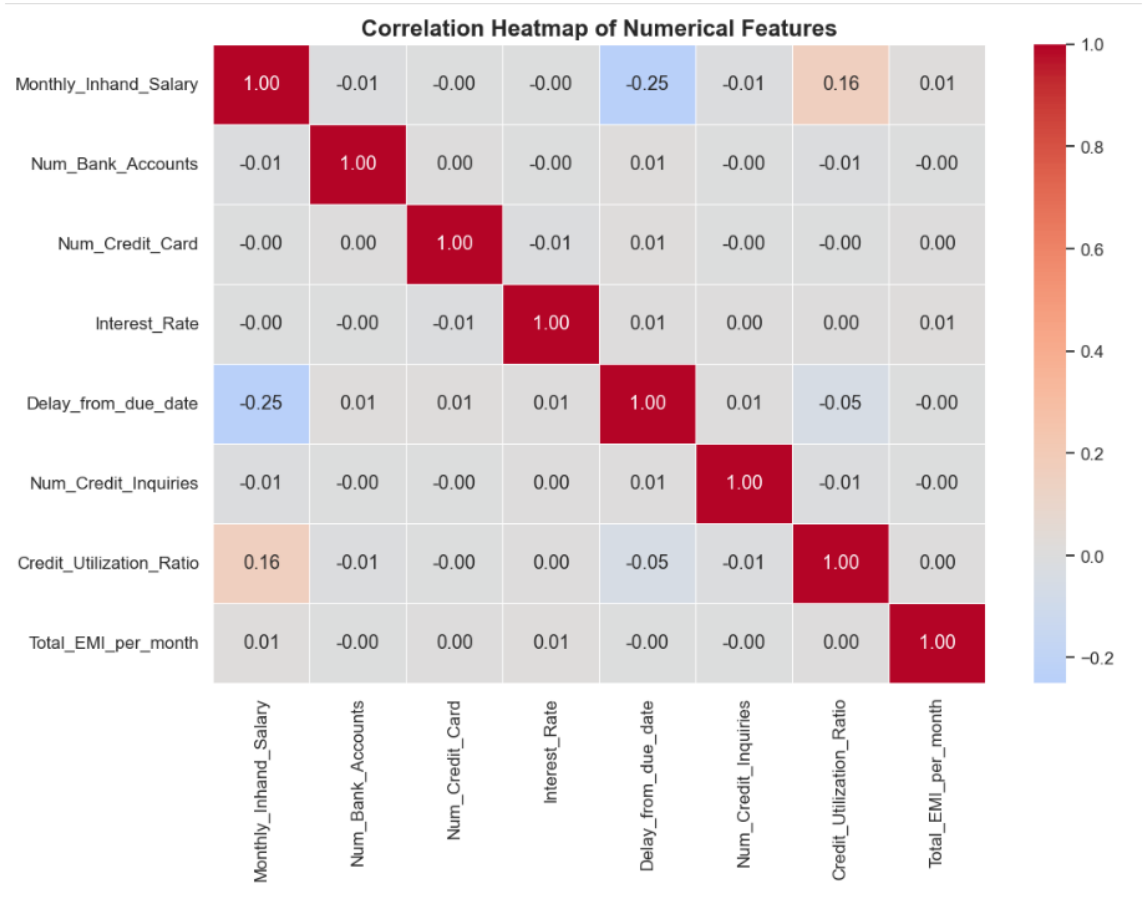
Correlation

```python
# Select only numeric columns for correlation
numeric_df = df.select_dtypes(include=['int64', 'float64'])

# Compute correlation matrix
corr = numeric_df.corr()

# Plot heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", center=0, linewidths=0.5)

plt.title("Correlation Heatmap of Numerical Features", fontsize=16, weight='bold')
plt.show()
```

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

Correlation Heatmap (All Numeric Columns only)



❑ The heatmap shows key correlations between financial features :

1. Negative correlation:

- Monthly Inhand Salary and Delay from Due Date (-0.25) ⟶ Higher salary = fewer delays.

2. Positive correlation:

- Monthly Inhand Salary and Credit Utilization Ratio (0.16) ⟶ Higher salary slightly linked to higher credit use.

❑ Business Uses :

- Credit risk assessment: Use salary and delays for loan approval.
- Customer segmentation: Segment by salary, credit utilization for targeted offers.
- Marketing: High salary and credit use ⟶ Offer credit limit increases or loans.

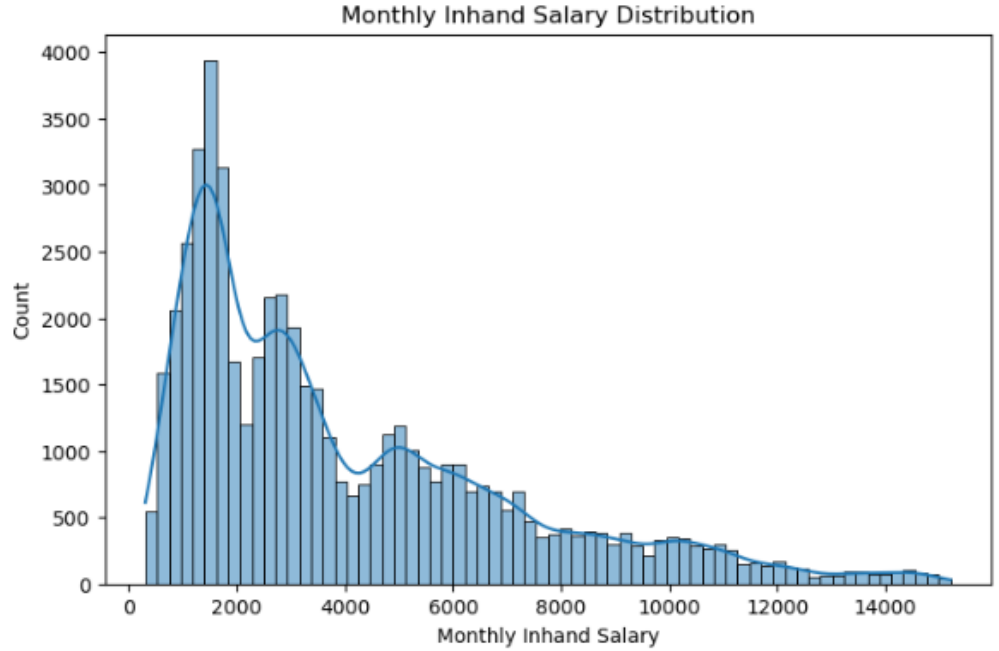
SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

Plots :

- **Histogram plots**

```
# Histogram for a numerical column
plt.figure(figsize=(8, 5))
sns.histplot(df['Monthly_Inhand_Salary'], kde=True)
plt.title("Monthly Inhand Salary Distribution")
plt.xlabel("Monthly Inhand Salary")
plt.ylabel("Count")
plt.show()
```



What Understand from It

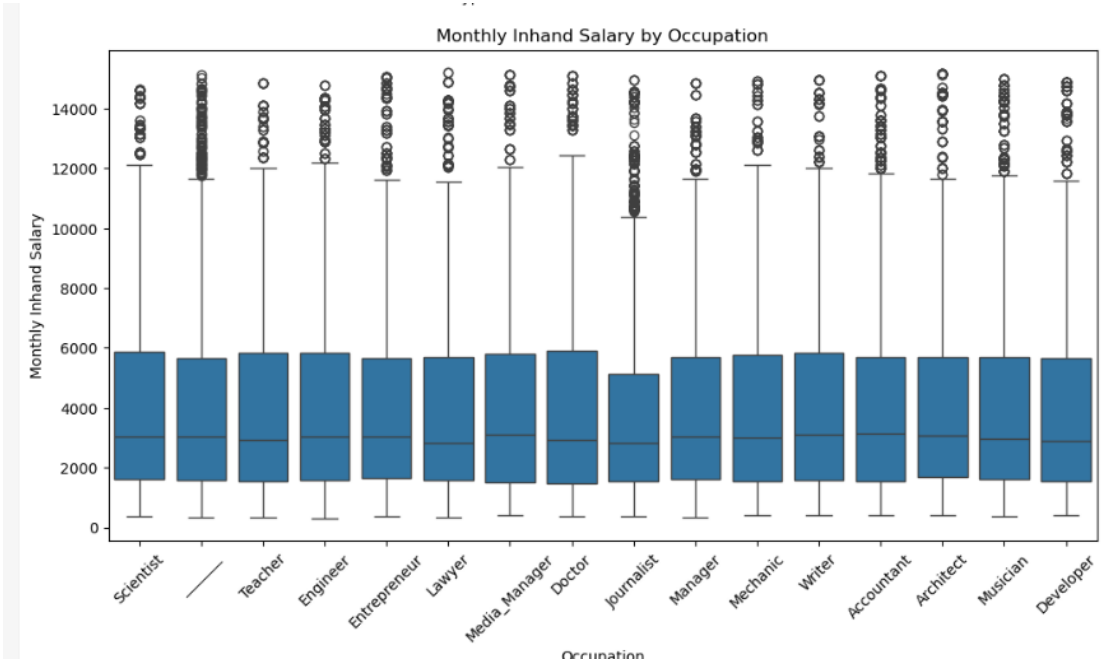
- Most people earn a lower inhand salary: The bulk of the population earns between ₹1,000 and ₹4,000 monthly.
- Fewer people earn high salaries: As the salary increases, the number of people earning that much decreases.
- Right-skewed distribution: This means there's a small group of people with very high salaries pulling the tail of the graph to the right.
- Income disparity: Indicates uneven salary distribution—more people earn less, and only a few earn a lot.
- Use in analysis: This kind of distribution helps understand customer segments, affordability, or even loan eligibility in a financial dataset.

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

- **Box plots**

```
# Box plot for salary by occupation
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x='Occupation', y='Monthly_Inhand_Salary')
plt.title("Monthly Inhand Salary by Occupation")
plt.xlabel("Occupation")
plt.ylabel("Monthly Inhand Salary")
plt.xticks(rotation=45)
plt.show()
```



What Understand from It

- The median monthly in-hand salary (middle line in each box) is fairly similar across all listed occupations.
- The spread (range between upper and lower quartiles) of salaries is also similar for each occupation, indicating a comparable distribution pattern.
- There are many outliers above the upper quartile, especially at higher salary levels, suggesting that while most people in each occupation earn in a similar range, a smaller group in each field earns significantly more.
- The overall salary ranges overlap considerably among occupations.

BIA BOSTON INSTITUTE OF ANALYTICS ®

Why This is Important:

Business Importance

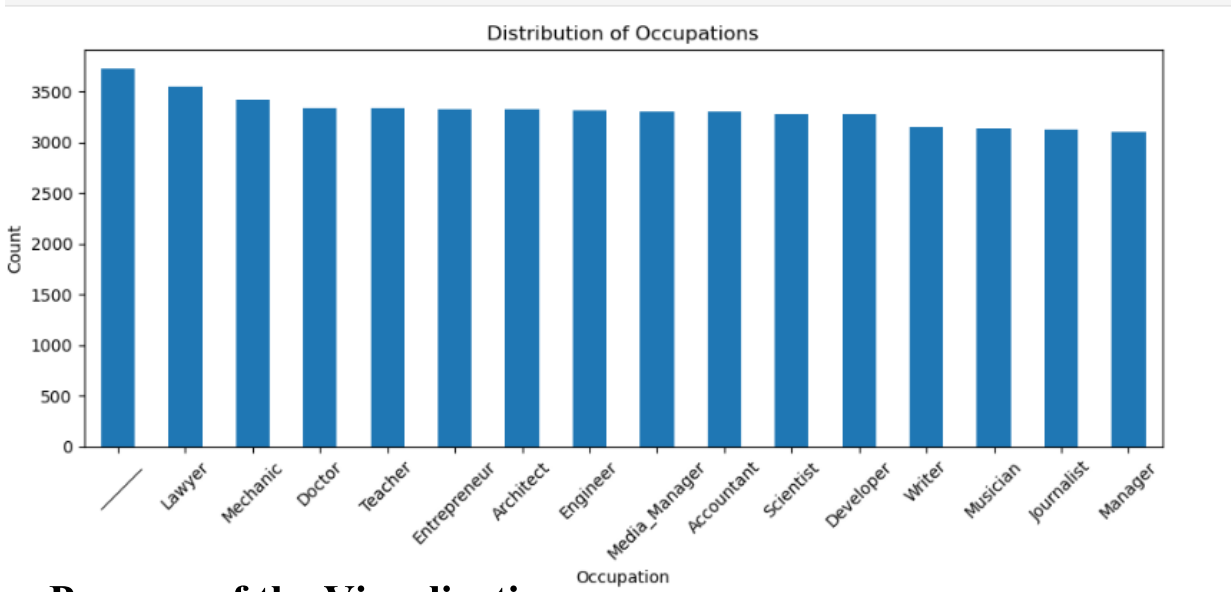
- **Targeted retention strategies: If online bookings have high cancellations, hotels can focus on stricter policies or better incentives for online customers.**
- **Revenue forecasting: Understanding which segments cancel more helps in more accurate occupancy predictions and pricing.**
- **Marketing focus: Low-cancellation segments (e.g., corporate) might deserve more sales efforts because they're reliable revenue sources.**

What I Can Gather from It:

- Booking patterns vary by market segment – Online and Offline segments dominate bookings, while Corporate, Aviation, and Complementary are much smaller.
- Cancellation trends differ by segment –Online bookings: High volume, but also a significant cancellation rate.Offline bookings: Fewer cancellations compared to online.Corporate bookings: Very low cancellation rate.Aviation & Complementary: Small volumes, minimal cancellations.
- Why this is important –Helps identify which customer segments are most likely to cancel.Guides targeted strategies, like offering flexible policies for high-risk groups (e.g., online customers) or incentives for lower-risk ones.Useful for resource planning and revenue forecasting in the hotel industry.

- **Bar Plot**

```
# Bar plot for Occupation
plt.figure(figsize=(10, 5))
df['Occupation'].value_counts().plot(kind='bar')
plt.title("Distribution of Occupations")
plt.xlabel("Occupation")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



What the Chart Shows

- The x-axis lists different occupations such as Lawyer, Mechanic, Doctor, Teacher, Entrepreneur, Architect, Engineer, Media Manager, Accountant, Scientist, Developer, Writer, Musician, Journalist, and Manager.
- The y-axis indicates the count or frequency of individuals in each occupation.
- The height of each bar represents how many people belong to each occupation

Observations : >

- Lawyer has the highest representation among the listed occupations, while Manager has the lowest.
- Most occupations have a fairly similar count, with only a slight decline from the highest (Lawyer) to the lowest (Manager).
- The occupations are distributed quite evenly, with no extreme dominance or negligible presence among the categories.The x-axis lists different occupations such as Lawyer, Mechanic, Doctor, Teacher, Entrepreneur, Architect, Engineer, Media Manager, Accountant, Scientist, Developer, Writer, Musician, Journalist, and Manager.
- The y-axis indicates the count or frequency of individuals in each occupation.
- The height of each bar represents how many people belong to each occupation

Purpose of the Visualization

This visualization helps understand the relative number of individuals in each occupation.

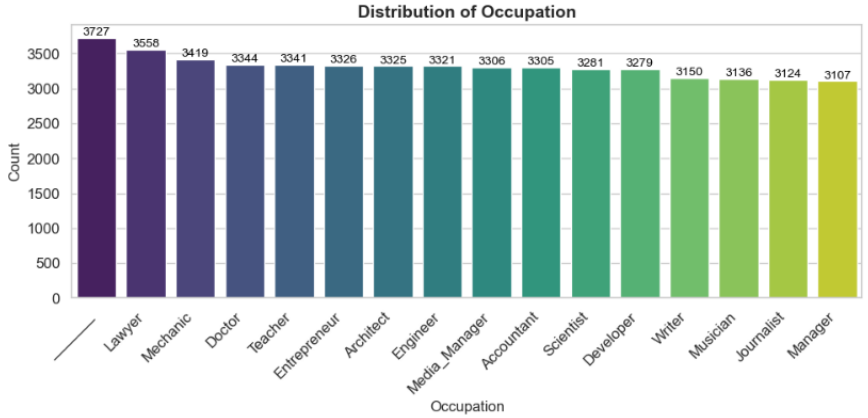
• Count Plot

Annual Income

```
[22]:
# Set style
sns.set(style="whitegrid", palette="muted", font_scale=1.1)

# List of categorical columns
categorical_cols = ['Occupation', 'Type_of_Loan',
                    'Credit_Mix', 'Payment_of_Min_Amount',
                    'Payment_Behaviour']

# Plot count plots for each categorical column
for col in categorical_cols:
    plt.figure(figsize=(10, 5))
    ax = sns.countplot(data=df, x=col, order=df[col].value_counts().index, palette="viridis")
    plt.title(f"Distribution of {col}", fontsize=14, weight='bold')
    plt.xticks(rotation=45, ha="right")
    plt.xlabel(col, fontsize=12)
    plt.ylabel("Count", fontsize=12)

    # Show exact counts on top of bars
    for p in ax.patches:
        ax.annotate(f'{int(p.get_height())}',
                    (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='bottom', fontsize=10, color='black', rotation=0)

    plt.tight_layout()
    plt.show()
```

```
plt.show()
```

```
C:\Users\nayan\AppData\Local\Temp\ipykernel_4476\4023166164.py:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  ax = sns.countplot(data=df, x=col, order=df[col].value_counts().index, palette="viridis")
```



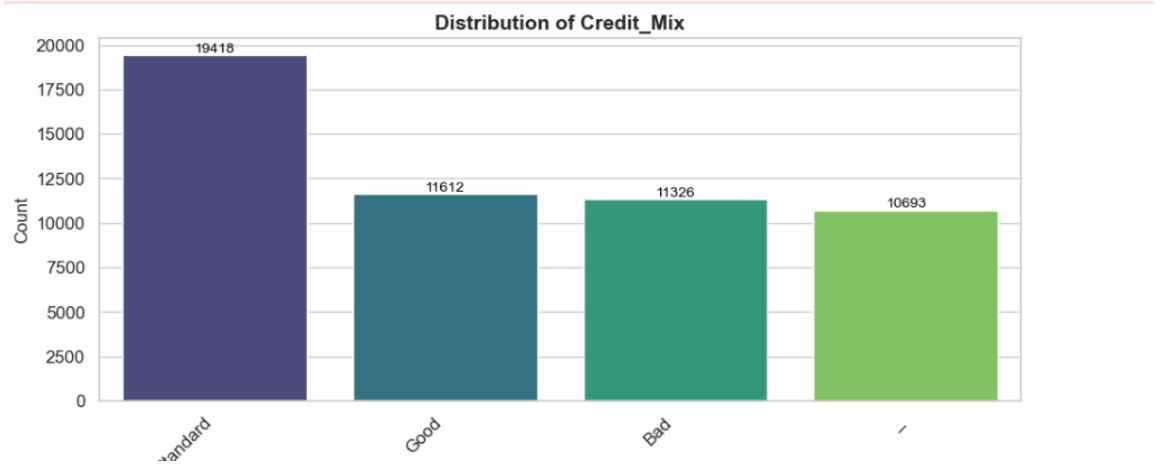
- X-axis: Occupation categories (Lawyer, Mechanic, Doctor, Teacher, etc.)
- Y-axis: Count (number of people in each occupation)
- Bars: Height of each bar = how many records (people) belong to that occupation
- It's sorted from the highest count to lowest.

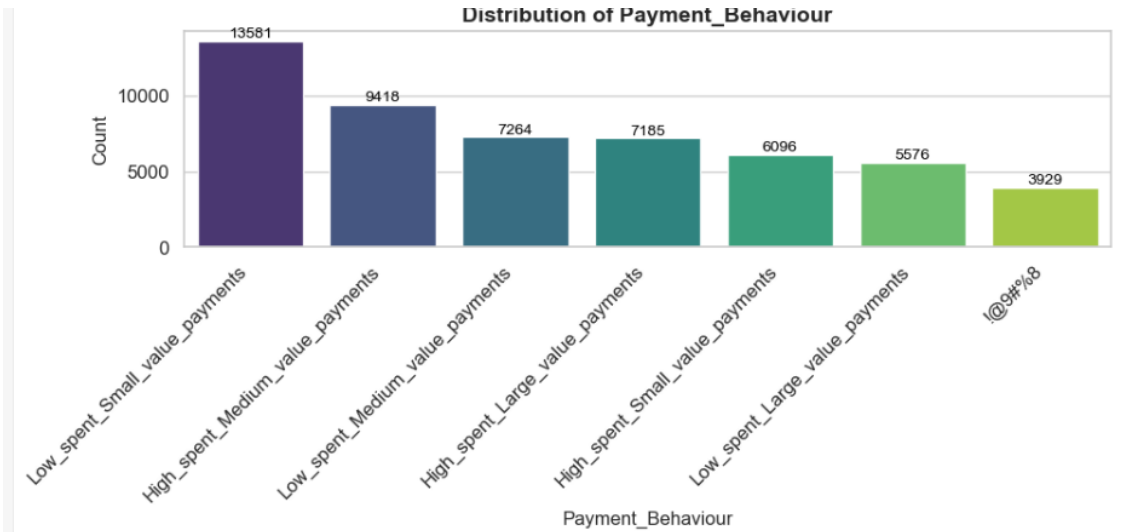
- Highest count:
- Lawyer (3727 records)
- Mechanic (3658 records)Doctor (3419 records)
- Lowest count:
- Manager (3107 records)
- Journalist (3124 records)
- Musician (3136 records)
- The counts are quite similar, but you can see the relative popularity of each occupation.

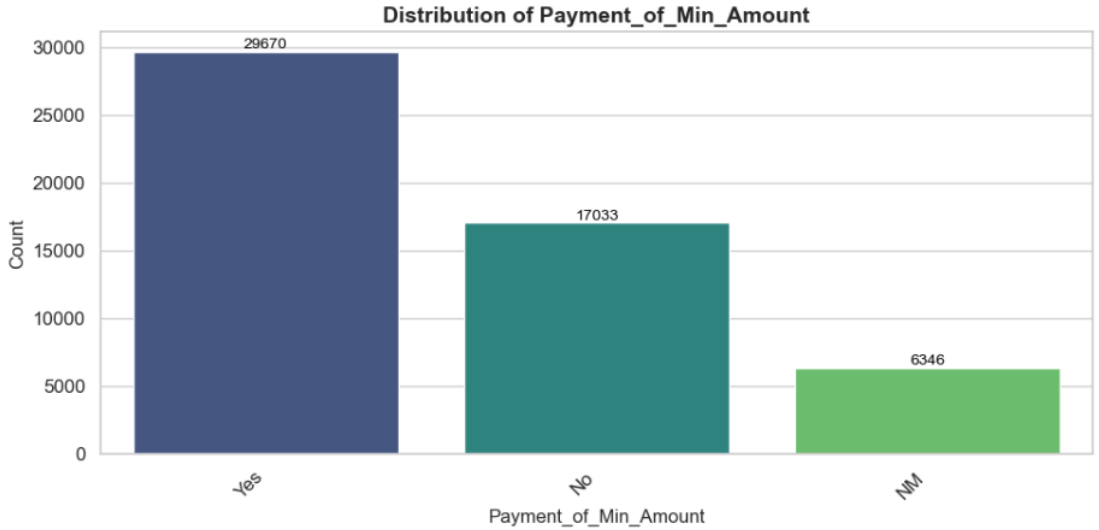
SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

- **Count Plot**







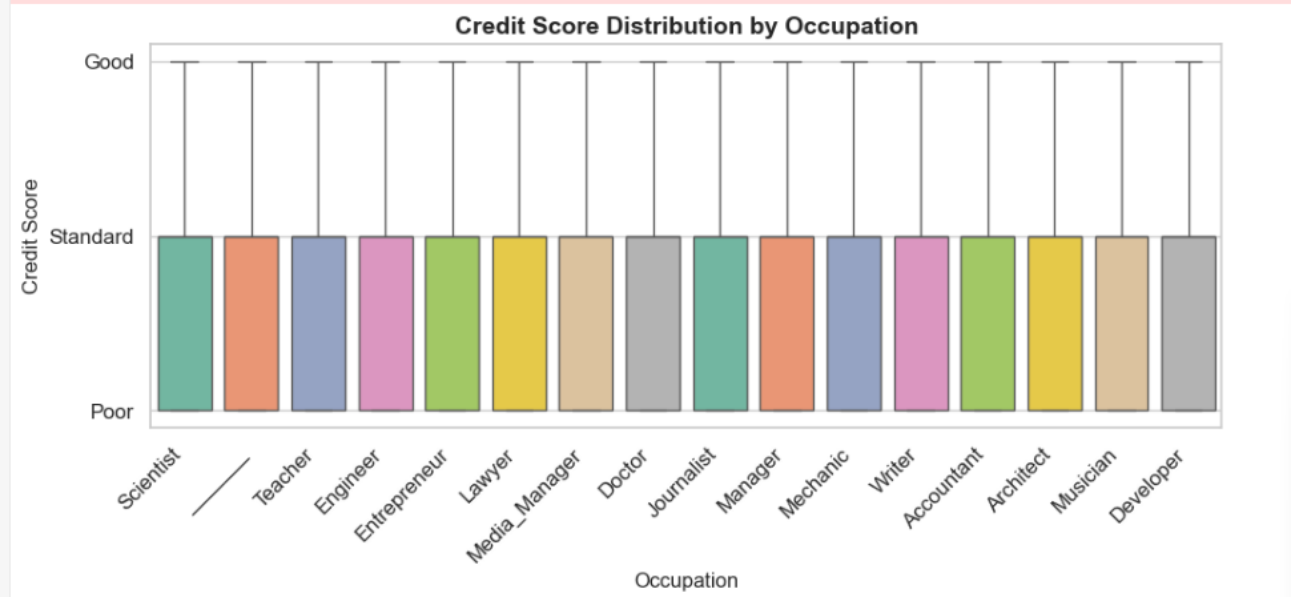
- Credit_Mix: Most are Standard ⟶ few missing values ⟶ clean needed.
- Payment_Behaviour: Mostly Low spent, Small payments ⟶ some invalid values ⟶ clean or group.
- Payment_of_Min_Amount: Majority Yes ⟶ few No and NM (missing).

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

• Box Plot

```
# Set style
sns.set(style="whitegrid", palette="Set2", font_scale=1.1)

# List of categorical columns
categorical_cols = ['Occupation', 'Type_of_Loan',
                    'Credit_Mix', 'Payment_of_Min_Amount',
                    'Payment_Behaviour']

# Loop through categorical columns and plot boxplots
for col in categorical_cols:
    plt.figure(figsize=(10, 5))
    sns.boxplot(data=df, x=col, y='Credit_Score', palette="Set2")

    plt.title(f"Credit Score Distribution by {col}", fontsize=14, weight='bold')
    plt.xticks(rotation=45, ha="right")
    plt.xlabel(col, fontsize=12)
    plt.ylabel("Credit Score", fontsize=12)

    plt.tight_layout()
    plt.show()
```

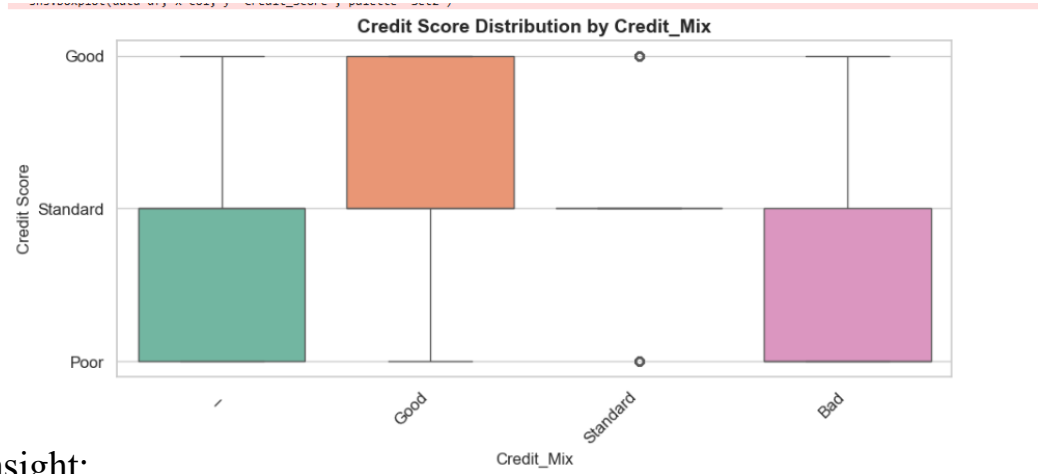


- Credit scores are similarly distributed across all occupations.
- Most fall in the “Standard” range, with some in “Good” and “Poor.”
- No single occupation shows a clear advantage or difference in credit score.

SPINNAKER ANALYTICS

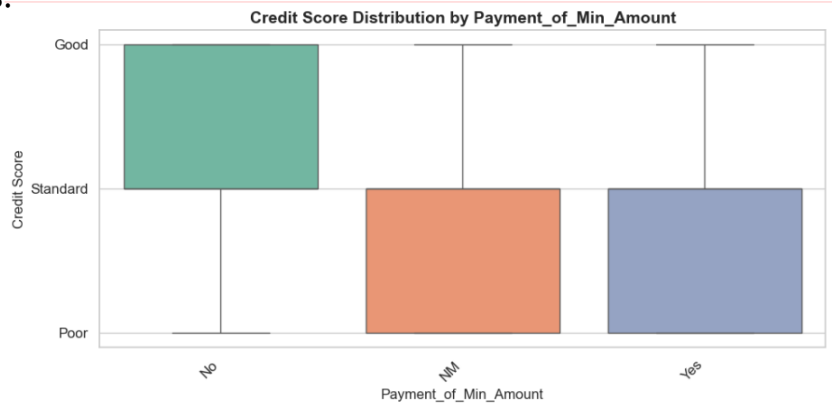
BIA BOSTON INSTITUTE OF ANALYTICS ®

Box Plots



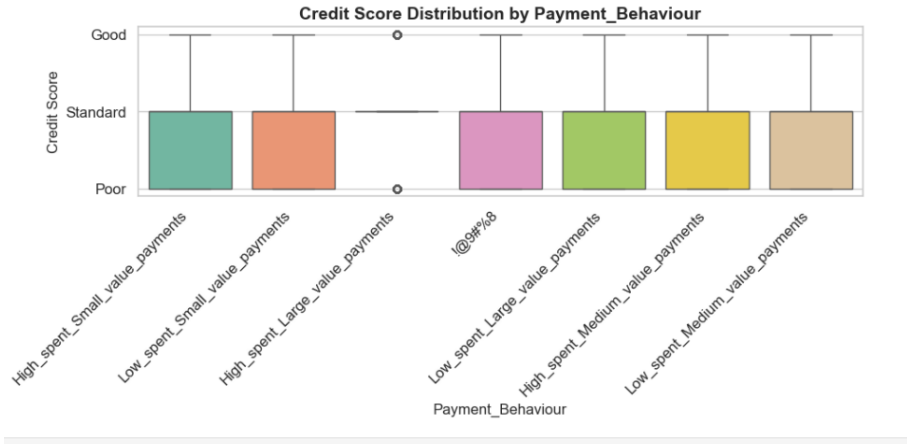
Insight:

- People with a Good credit mix tend to have better credit scores, whereas those with Bad or Standard credit mix lean toward lower scores.



Insight:

- Paying the minimum amount regularly is associated with higher credit scores, showing a clear relationship between responsible payment behaviour and creditworthiness.



Insight:

- Payment behaviour has some effect but not a strong differentiator. Most customers, regardless of payment pattern, fall into the Standard credit score range. Data cleaning is also needed for invalid values.

❑ Business Use:

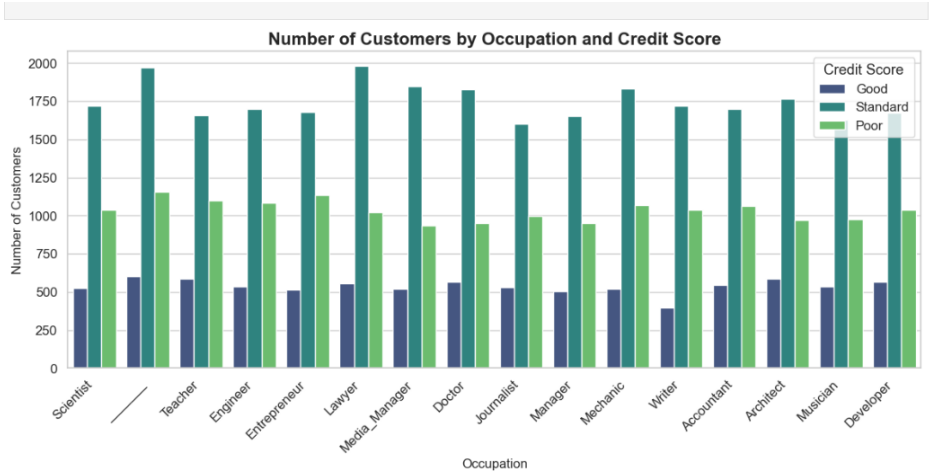
- These insights can be used to build better credit risk models,
- Identify good vs risky customers, and
- Support decision-making in loan approvals and credit management.

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

Count Plot

```
# Set style
sns.set(style="whitegrid", palette="pastel", font_scale=1.1)

plt.figure(figsize=(12, 6))

# Countplot: number of customers by Occupation and Credit Score
sns.countplot(data=df, x='Occupation', hue='Credit_Score', palette="viridis")

plt.title("Number of Customers by Occupation and Credit Score", fontsize=16, weight='bold')
plt.xticks(rotation=45, ha="right")
plt.xlabel("Occupation", fontsize=12)
plt.ylabel("Number of Customers", fontsize=12)

plt.legend(title="Credit Score")
plt.tight_layout()
plt.show()
```



❑ What I Gather :

- Credit Mix: Good credit mix ⟶ higher credit scores; Bad/Standard ⟶ lower scores.
- Payment Behaviour: Most customers fall in Standard score; weak impact; some invalid data needs cleaning.
- Payment of Min Amount: People who pay regularly have better credit scores; non-payers have lower scores.
- Data Quality: Missing/invalid values must be handled before modeling.

❑ Business Use :

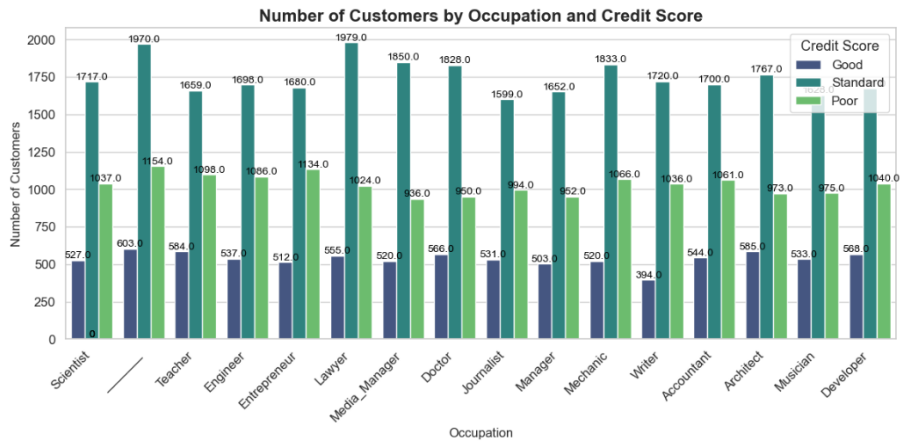
- Identify low-risk customers (good credit mix + regular payers) for loan approvals.
- Target high-risk groups (bad credit mix, irregular payments) with stricter credit policies.
- Use payment behaviour and credit mix to build credit risk prediction models.
- Improve credit portfolio quality by rewarding good behaviour and managing risky customers proactively.

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

Count Plot

```
# Set style
sns.set(style="whitegrid", palette="pastel", font_scale=1.1)

plt.figure(figsize=(12, 6))

ax = sns.countplot(data=df, x='Occupation', hue='Credit_Score', palette="viridis")

plt.title("Number of Customers by Occupation and Credit Score", fontsize=16, weight='bold')
plt.xticks(rotation=45, ha="right")
plt.xlabel("Occupation", fontsize=12)
plt.ylabel("Number of Customers", fontsize=12)

# Add counts on top of bars
for p in ax.patches:
    height = p.get_height()
    ax.annotate(f'{height}',
                (p.get_x() + p.get_width() / 2., height),
                ha='center', va='bottom', fontsize=10, color='black')

plt.legend(title="Credit Score")
plt.tight_layout()
plt.show()
```



- The chart shows the distribution of credit scores (Good, Standard, Poor) across different occupations.
- High credit scores are seen in Scientists, Engineers, Doctors, and Managers.
- Poor credit scores are more common in Entrepreneurs, Mechanics, Writers, and Journalists.
- Standard scores are spread more evenly across occupations.

- Business Ideas :
  - Tailor products: Offer premium credit products to high-score occupations and credit-building products to low-score ones.
  - Risk management: Adjust loan approvals based on occupation and credit score patterns.
  - Targeted marketing: Focus financial education on low-score occupations and exclusive offers to high-score ones.

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

Model Training (Baseline + Advanced Models)

Train-test Split

Logistic Regression Model Train and test

```
# ---------------------------
# Step 1: Define features & target
# ---------------------------
X = df.drop('Credit_Score', axis=1)
y = df['Credit_Score']

# Encode target if it's categorical (Good/Average/Poor)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# ---------------------------
# Step 2: Handle categorical features
# ---------------------------
categorical_cols = X.select_dtypes(include=['object']).columns
numeric_cols = X.select_dtypes(include=['int64','float64']).columns

# Preprocess: One-hot encode categorical, scale numeric
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numeric_cols),
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
])

# ---------------------------
# Step 3: Train-test split
# ---------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# ---------------------------
# Step 4: Logistic Regression Model
# ---------------------------
log_reg = LogisticRegression(max_iter=1000)

# Build pipeline
from sklearn.pipeline import Pipeline
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', log_reg)])

# Fit model
clf.fit(X_train, y_train)

# ---------------------------
# Step 5: Evaluate
# ---------------------------
y_pred = clf.predict(X_test)

print(" Accuracy:", accuracy_score(y_test, y_pred))
print("\n Classification Report:\n", classification_report(y_test, y_pred, target_names=label_encoder.classes_))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.772478793590952

Classification Report:
              precision    recall  f1-score   support

        Good       0.74      0.69      0.72      1717
        Poor       0.78      0.76      0.77      3303
    Standard       0.78      0.80      0.79      5590

    accuracy                           0.77     10610
   macro avg       0.77      0.75      0.76     10610
weighted avg       0.77      0.77      0.77     10610


Confusion Matrix:
 [[1193    7  517]
 [  18 2512  773]
 [ 403  696 4491]]
```

❑ What I Gather :

- Accuracy: 77.25%
- Best for Standard class (F1 = 0.79)
- Weakest for Good class (Recall = 0.69)
- Some misclassifications, especially between Good and Standard

❑ Business Use :

- Automate credit scoring
- Flag risky misclassified cases for manual review
- Use predicted scores for customer segmentation and loan decisions

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

Model Training (Baseline + Advanced Models)

```
import pandas as pd
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Predict
y_pred = clf.predict(X_test)

# ---------------------------
# 1️⃣ Accuracy as a small DataFrame
# ---------------------------
acc_df = pd.DataFrame({"Metric": ["Accuracy"],
                       "Value": [accuracy_score(y_test, y_pred)]})
display(acc_df)

# ---------------------------
# 2️⃣ Classification report as a DataFrame
# ---------------------------
report_dict = classification_report(
    y_test, y_pred, target_names=label_encoder.classes_, output_dict=True)

report_df = pd.DataFrame(report_dict).transpose()
display(report_df)

# ---------------------------
# 3️⃣ Confusion matrix as a DataFrame
# ---------------------------
cm = confusion_matrix(y_test, y_pred)

# Put row/column labels = class names
cm_df = pd.DataFrame(cm,
                     index=[f"Actual_{c}" for c in label_encoder.classes_],
                     columns=[f"Pred_{c}" for c in label_encoder.classes_])
display(cm_df)
```

	Metric	Value
0	Accuracy	0.772479

	precision	recall	f1-score	support
Good	0.739157	0.694817	0.716301	1717.000000
Poor	0.781337	0.760521	0.770789	3303.000000
Standard	0.776855	0.803399	0.789904	5590.000000
accuracy	0.772479	0.772479	0.772479	0.772479
macro avg	0.765783	0.752912	0.758998	10610.000000
weighted avg	0.772150	0.772479	0.772042	10610.000000

	Pred_Good	Pred_Poor	Pred_Standard
Actual_Good	1193	7	517
Actual_Poor	18	2512	773
Actual_Standard	403	696	4491

❑ Business Use :

- Customer Segmentation: Accurately identifying “Standard” and “Poor” groups can help tailor offers or retention plans.
- Resource Allocation: Since Standard and Poor have higher precision/recall, campaigns can focus here for better ROI.
- Improve Marketing: Misclassified “Good” customers can be targeted with personalized engagement to reduce error.
- Decision Support: Model is reliable enough for medium-risk business decisions (77% accuracy), but improving “Good” prediction could boost impact.

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

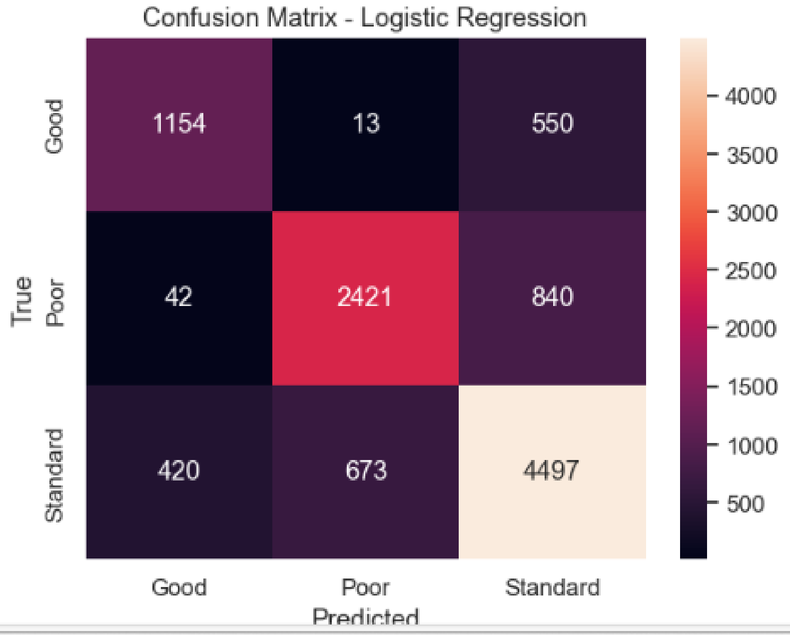
[44]:

```python
# ======================
# 1. Imports
# ======================
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier   # pip install xgboost if not already

# ======================
# 2. Prepare Data
# ======================
# drop ID-like columns
X = df.drop(['Credit_Score','ID','Customer_ID','Name','SSN'], axis=1, errors='ignore')
y = df['Credit_Score']

# Encode target if categorical
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Identify numeric and categorical columns
categorical_cols = X.select_dtypes(include=['object']).columns
numeric_cols = X.select_dtypes(include=['int64','float64']).columns

# Preprocessor: scale numeric + one-hot categorical
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numeric_cols),
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
])

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```python
# ======================
# 3. Models to compare
# ======================
log_reg = LogisticRegression(max_iter=1000)
rf = RandomForestClassifier(n_estimators=300, max_depth=15, random_state=42)
xgb = XGBClassifier(n_estimators=300, learning_rate=0.1,
                    max_depth=6, subsample=0.8, colsample_bytree=0.8,
                    eval_metric='mlogloss', use_label_encoder=False)

models = [
    ('Logistic Regression', log_reg),
    ('Random Forest', rf),
    ('XGBoost', xgb)
]

# ======================
# 4. Evaluate function
# ======================
def evaluate_models(models, X_train, X_test, y_train, y_test, class_names):
    results = []
    for name, model in models:
        # Build pipeline each time
        pipe = Pipeline(steps=[('preprocessor', preprocessor),
                               ('classifier', model)])
        pipe.fit(X_train, y_train)
        y_pred = pipe.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results.append({'Model': name, 'Test Accuracy': acc})

        print(f"\n=== {name} ===")
        print("Accuracy:", acc)
        print("\nClassification Report:")
        print(classification_report(y_test, y_pred, target_names=class_names))

        cm = confusion_matrix(y_test, y_pred)
        sns.heatmap(cm, annot=True, fmt='d', xticklabels=class_names, yticklabels=class_names)
        plt.title(f"Confusion Matrix - {name}")
        plt.xlabel('Predicted')
        plt.ylabel('True')
        plt.show()
    return pd.DataFrame(results)

# ======================
# 5. Run evaluation
# ======================
results_df = evaluate_models(models, X_train, X_test, y_train, y_test, class_names=label_encoder.classes_)
print("\nSummary Table:")
print(results_df)
```

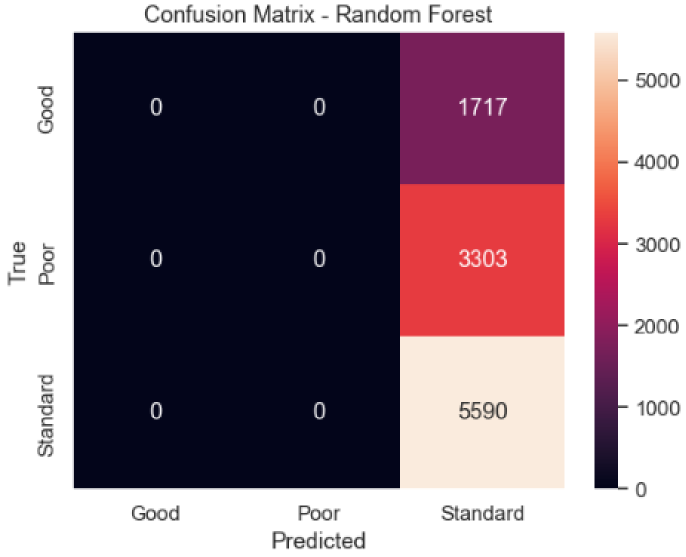
✓ Logistic Regression

```
=== Logistic Regression ===
Accuracy: 0.7607917059377945

Classification Report:
              precision    recall  f1-score   support

        Good       0.71      0.67      0.69      1717
        Poor       0.78      0.73      0.76      3303
    Standard       0.76      0.80      0.78      5590

    accuracy                           0.76     10610
   macro avg       0.75      0.74      0.74     10610
weighted avg       0.76      0.76      0.76     10610
```



✓ Random Forest

```
=== Random Forest ===
Accuracy: 0.5268614514608859

Classification Report:
              precision    recall  f1-score   support

        Good       0.00      0.00      0.00      1717
        Poor       0.00      0.00      0.00      3303
    Standard       0.53      1.00      0.69      5590

    accuracy                           0.53     10610
   macro avg       0.18      0.33      0.23     10610
weighted avg       0.28      0.53      0.36     10610

C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1565: Undef
0 in labels with no predicted samples. Use `zero_division` parameter to control this beha
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1565: Undef
0 in labels with no predicted samples. Use `zero_division` parameter to control this beha
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1565: Undef
0 in labels with no predicted samples. Use `zero_division` parameter to control this beha
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

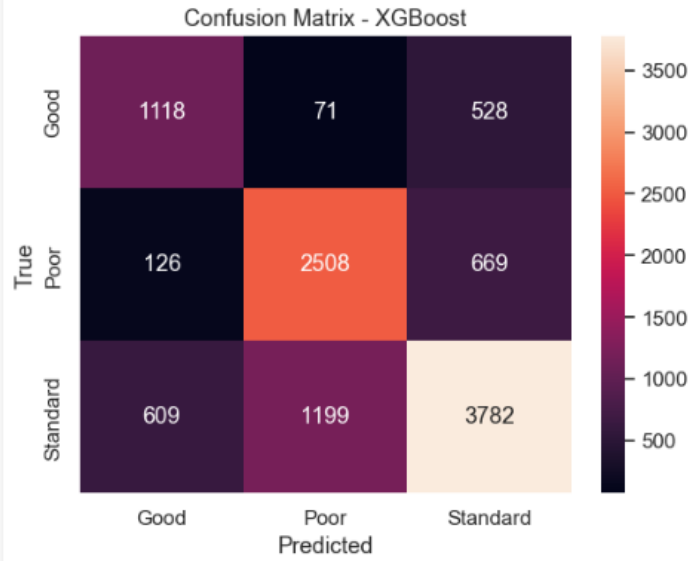


SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

✓ XGBoost

```
=== XGBoost ===
Accuracy: 0.6982092365692742

Classification Report:
              precision    recall  f1-score   support

        Good       0.60      0.65      0.63      1717
        Poor       0.66      0.76      0.71      3303
    Standard       0.76      0.68      0.72      5590

    accuracy                           0.70     10610
   macro avg       0.68      0.70      0.68     10610
weighted avg       0.70      0.70      0.70     10610
```




```
Summary Table:
                 Model  Test Accuracy
0  Logistic Regression       0.760792
1        Random Forest       0.526861
2              XGBoost       0.698209
```

❑ Logistic Regression:

- Accuracy: 76.08%
- Precision: Good: 0.71, Poor: 0.78, Standard: 0.76
- Recall: Good: 0.67, Poor: 0.73, Standard: 0.80
- F1-Score: Good: 0.69, Poor: 0.76, Standard: 0.78
- Confusion Matrix: The model performs reasonably well across all categories, with a balanced trade-off between precision and recall.

❑ Random Forest:

- Accuracy: 52.68%
- Precision: Good: 0.00, Poor: 0.00, Standard: 0.53
- Recall: Good: 0.00, Poor: 0.00, Standard: 1.00
- F1-Score: Good: 0.00, Poor: 0.00, Standard: 0.53
- Confusion Matrix: The Random Forest model appears to be severely misclassifying data, with no predicted labels for "Good" and "Poor," causing the model to only classify "Standard."

❑ XGBoost:

- Accuracy: 69.82%
- Precision: Good: 0.60, Poor: 0.66, Standard: 0.76
- Recall: Good: 0.65, Poor: 0.76, Standard: 0.68
- F1-Score: Good: 0.63, Poor: 0.71, Standard: 0.72
- Confusion Matrix: XGBoost has a more balanced performance but still has some misclassification, particularly with the "Good" and "Standard" categories.

✓ **Comparison of the Models**:

➢ Accuracy Comparison:

- Logistic Regression performs the best with an accuracy of 76.08%, followed by XGBoost at 69.82%. Random Forest is the least effective with an accuracy of just 52.68%.

➢ Precision and Recall:

- Logistic Regression has relatively high precision and recall, especially for the "Poor" category (recall: 0.73, precision: 0.78).
- XGBoost also performs fairly well with balanced precision and recall.
- Random Forest shows problematic results, as the model essentially predicts only "Standard" and fails to predict "Good" or "Poor," which drastically lowers its performance.

✓ **Business Use Cases and Benefits:**

- Logistic Regression:

➢ This model is solid for businesses looking for a balance between performance and interpretability. It's useful when accurate prediction across multiple categories is important, as it performs well across "Good," "Poor," and "Standard" labels. For example, in customer feedback analysis, it can help classify satisfaction levels.

➢ Benefit: Reliable for industries like finance, marketing, and healthcare where understanding customer sentiment (Good, Poor, Standard) is critical.

- Random Forest:

➢ Although its accuracy is low, when tuned correctly or with more data, Random Forest could perform better. This model is very powerful for large datasets where feature interactions are complex.

➢ Benefit: It's useful in fraud detection, anomaly detection, and any problem where the relationship between features is complex.

- XGBoost:

➢ XGBoost is a robust model for classification tasks with high accuracy and efficiency, especially in large datasets. It tends to outperform other models like logistic regression in some cases.

➢ Benefit: Excellent for time-sensitive decision-making in industries like e-commerce, insurance, and marketing, where predicting outcomes based on historical data (like product success or churn rates) is valuable.

```
Credit_Score               object
dtype: object
```

[15]:
```python
from sklearn.preprocessing import LabelEncoder

for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = LabelEncoder().fit_transform(df[col].astype(str))
```

[16]:
```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

X = df.drop('Credit_Score', axis=1)  # replace with your target column
y = df['Credit_Score']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.78855
Confusion Matrix:
 [[2521   20  986]
 [  82 4631 1161]
 [ 745 1235 8619]]
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.71      0.73      3527
           1       0.79      0.79      0.79      5874
           2       0.80      0.81      0.81     10599

    accuracy                           0.79     20000
   macro avg       0.78      0.77      0.78     20000
weighted avg       0.79      0.79      0.79     20000
```

- Model Used: Random Forest Classifier
- Goal: Predict customer Credit Score category
- Accuracy: 78.85% — good performance
- Precision / Recall / F1-score: Around 0.79 overall, showing balanced and reliable predictions
- Confusion Matrix: Some mix-ups between nearby credit score classes, but mostly correct
- Business Use :
- ❖ Helps banks/fintechs predict creditworthiness Useful for loan approvals, risk management, and customer segmentation

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

✓ Test Accuracy

```
[17]: # Step 1: Import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Step 2: Load dataset
df = pd.read_csv('train..csv', low_memory=False)  # Replace with your file name

# Step 3: Check column names
print("Columns in dataset:\n", df.columns)

# Step 4: Replace with your actual target column name
target_column = 'Credit_Score'

# Step 5: Encode categorical columns
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = LabelEncoder().fit_transform(df[col].astype(str))

# Step 6: Split features and target
X = df.drop(target_column, axis=1)
y = df[target_column]

# Step 7: Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Step 8: Train Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Step 9: Predict on test dataset
y_pred = model.predict(X_test)

# Step 10: Evaluate and report
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Step 11: Print results
print("\n=== Test Accuracy ===")
print(f"Accuracy: {accuracy:.4f} → How many predictions were correct.")

print("\n=== Confusion Matrix ===")
print(conf_matrix, "\n→ Shows correct vs incorrect predictions in detail.")

print("\n=== Classification Report ===")
print(class_report, "→ Shows precision, recall, F1-score for each class.")
```

❑ What You Gained (Model Insight)

- Model Used: Random Forest Classifier
- Purpose: Predict a customer's Credit Score category using various financial and personal features.
- Accuracy: 78.85% — the model correctly predicts about 8 out of 10 customers.
- Precision / Recall / F1-score: Around 0.79, showing balanced and reliable predictions.
- Confusion Matrix: Shows which credit score classes were correctly or wrongly predicted.

❑ Business Use :

❖ Credit Risk Assessment:

➢ Identify high-risk vs. low-risk customers before approving loans or credit cards.

❖ Loan Approval Automation:

➢ Approve or reject applications automatically based on predicted credit score.

❖ Interest Rate Optimization:

➢ Offer lower rates to good-score customers and higher rates to risky ones.

❖ Customer Segmentation:

➢ Target reliable customers for premium products or credit upgrades.

✓ Cross Validation Accuracy

→ Shows precision, recall, F1-score for each class.

[21]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
import numpy as np

# Load the data
df = pd.read_csv('train..csv')

# Encode categorical features
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = LabelEncoder().fit_transform(df[col].astype(str))

# Impute missing values (one line fix)
df = pd.DataFrame(SimpleImputer(strategy='mean').fit_transform(df), columns=df.columns)

# Define target and features
target_column = 'Credit_Score'  # Replace with actual column name
X = df.drop(target_column, axis=1)
y = df[target_column]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Define models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state=42)
}

# Cross-Validation & Training
print("=== Model Evaluation with 5-Fold Cross-Validation ===\n")
for name, model in models.items():
    cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
    print(f"{name}:")
    print(f"  Cross-Validation Accuracy: {cv_scores.mean():.4f} ± {cv_scores.std():.4f}")
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"  Test Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print(f"  Classification Report:\n{classification_report(y_test, y_pred)}")
    print("-" * 60)
```

```
Logistic Regression:
  Cross-Validation Accuracy: 0.5494 ± 0.0026
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:4
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
  n_iter_i = _check_optimize_result(
  Test Accuracy: 0.5451
  Classification Report:
              precision    recall  f1-score   support

         0.0       0.34      0.06      0.10      3566
         1.0       0.55      0.27      0.36      5799
         2.0       0.55      0.86      0.67     10635

    accuracy                           0.55     20000
   macro avg       0.48      0.40      0.38     20000
weighted avg       0.51      0.55      0.48     20000

------------------------------------------------------------
Random Forest:
  Cross-Validation Accuracy: 0.7795 ± 0.0015
  Test Accuracy: 0.7885
  Classification Report:
              precision    recall  f1-score   support

         0.0       0.76      0.72      0.74      3566
         1.0       0.78      0.79      0.79      5799
         2.0       0.80      0.81      0.81     10635

    accuracy                           0.79     20000
   macro avg       0.78      0.77      0.78     20000
weighted avg       0.79      0.79      0.79     20000

------------------------------------------------------------
```

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

```
  bst.update(dtrain, iteration=i, fobj=obj)
XGBoost:
  Cross-Validation Accuracy: 0.7593 ± 0.0035
C:\Users\nayan\AppData\Roaming\Python\Python313\site-packages\xg
\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
  Test Accuracy: 0.7651
  Classification Report:
              precision    recall  f1-score   support

         0.0       0.72      0.71      0.71      3566
         1.0       0.77      0.73      0.75      5799
         2.0       0.78      0.80      0.79     10635

    accuracy                           0.77     20000
   macro avg       0.75      0.75      0.75     20000
weighted avg       0.76      0.77      0.76     20000

------------------------------------------------------------
```

[ ]:

- What You Gather (Model Results)
- You compared three machine learning models for predicting customer credit score:

Model	Cross-Validation Accuracy	Test Accuracy	Remarks
Logistic Regression	~0.54	0.545	Low accuracy, failed to converge (needs more iterations or feature scaling).
Random Forest	~0.78	0.79	Best performer — strong and stable results.
XGBoost	~0.76	0.765	Slightly lower than Random Forest but still good.

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®

❑ **Interpretation of Reports:**

- Precision, recall, and F1-score are around 0.75–0.80, showing balanced performance.
- Model correctly predicts about 78–79% of customers' credit scores.
- Random Forest gives the best trade-off between accuracy and interpretability.

❑ **Business Use**

➢ Credit Risk Prediction

- Predict whether a customer will have a Good / Average / Poor credit score before issuing loans or credit cards.

➢ Loan Approval Automation

- Use model outputs to automate loan decisions — approve, reject, or request more info.

➢ Default Risk Reduction

- Identify risky customers early, reducing non-performing loans (NPA).

➢ Customer Segmentation

- Offer personalized interest rates or credit limits based on predicted credit class.

➢ Decision Insights

- Random Forest's feature importance can show which factors (income, payment behavior, etc.) most affect credit scores.

✓ **In short:**

➢ I built and compared models that predict a person's credit score from financial data.Random Forest performed best — giving the business a data-driven way to assess credit risk and make smarter lending decisions.

SPINNAKER ANALYTICS

BIA BOSTON INSTITUTE OF ANALYTICS ®


Thank You!