Mini Project Report
On
Financial Credit Scoring Analysis using Python

Submitted by
[All Group Members]
Name:- Akash Verma
Roll No.:- 2206006
Name:-Rohit Kumar
Roll No.:- 2206045

School of Computer Engineering
KIIT - DU

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY (KIIT)

Deemed to be University U/S 3 of UGC Act, 1956

Table of Contents

Chapter 1

Introduction

Financial Credit Scoring is a crucial task in the banking and finance industry. It helps financial institutions assess the creditworthiness of individuals before approving loans or credit. This project implements a machine learning-based credit scoring system using Python to analyze financial data and predict whether a customer is likely to default on a loan.

Our project is divided into multiple sub-tasks:

- **Data Preprocessing** – Cleaning and preparing financial data
- **Exploratory Data Analysis (EDA)** – Analyzing patterns and trends
- **Feature Selection** – Choosing relevant attributes for prediction
- **Model Training** – Implementing ML models for classification
- **Evaluation & Interpretation** – Analyzing the performance of models

Python, along with libraries like pandas, NumPy, scikit-learn, and matplotlib, is used to build the model and evaluate its accuracy.

Chapter 2

Problem Statement & Objectives

Problem Statement

Loan default is a major risk for financial institutions. Predicting the likelihood of a customer defaulting based on past financial data is a complex task. Traditional credit scoring methods may fail to capture hidden patterns, making ML-based predictions a better alternative.

Objectives

1. **Analyze financial data** to identify key credit risk factors.
2. **Build and evaluate ML models** for accurate credit scoring.
3. **Visualize and interpret results** to provide actionable insights for financial decision-making

Chapter 3

Python Packages Used Details

Name	Functions Used	Explanation
pandas	read_csv(), DataFrame(), info(), dropna(), describe()	Data manipulation and preprocessing
NumPy	array(), mean(), std()	Numerical computations
matplotlib.pyplot	plot(), hist(), scatter()	Data visualization
seaborn	heatmap(),pairplot(), countplot(),boxplot(), histplot()	Advanced data visualization
scikit-learn (sklearn)	train_test_split(), LogisticRegression(), RandomForestClassifier(), accuracy_score(), confusion_matrix(),classification_report()	Model training and evaluation

Chapter 4

Source Code

The detailed source code is to be provided here with proper captions.

- Source code for Financial Credit Scoring Analysis
- Programming Language Used: Python.
- Platform : Google Colab.

Required Libraries

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report,
f1_score, confusion_matrix
```

Loading Dataset with 20 Features
Total instances = 1000

```
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/
german.data"
columns = [
      "Status", "Duration", "CreditHistory", "Purpose", "CreditAmount",
"Savings", "Employment", "InstallmentRate",
      "PersonalStatus", "OtherDebtors", "ResidenceDuration", "Property",
"Age", "OtherInstallmentPlans", "Housing",
            "ExistingCredits", "Job", "PeopleLiable", "Telephone",
"ForeignWorker", "CreditRisk"
]

df = pd.read_csv(url, delimiter=" ", names=columns)
df["CreditRisk"] = df["CreditRisk"].map({1: 0, 2: 1})  # 1 = Good (0), 2 =
Bad (1)
```

I. Data Preprocessing

Convert categorical variables using LabelEncoder.

Scale numerical features.

```
categorical_cols = df.select_dtypes(include=['object']).columns
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])

# Splitting data
X = df.drop(columns=["CreditRisk"])
y = df["CreditRisk"]

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

# Scale numeric data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

II. Exploratory Data Analysis (EDA) on German Credit Dataset

- EDA helps in understanding data patterns, detecting anomalies, and selecting the best features for modeling

```
print("Dataset Shape:", df.shape)
```

Dataset Shape: (1000, 21)

```
print(df.head())
```

```
print(df.info())
```

Data columns (total 21 columns):

#	Column	Non-Null Count	Dtype
0	Status	1000 non-null	int64
1	Duration	1000 non-null	int64
2	CreditHistory	1000 non-null	int64
3	Purpose	1000 non-null	int64

```
 4   CreditAmount          1000 non-null   int64
 5   Savings               1000 non-null   int64
 6   Employment            1000 non-null   int64
 7   InstallmentRate       1000 non-null   int64
 8   PersonalStatus        1000 non-null   int64
 9   OtherDebtors          1000 non-null   int64
 10  ResidenceDuration     1000 non-null   int64
 11  Property              1000 non-null   int64
 12  Age                   1000 non-null   int64
 13  OtherInstallmentPlans 1000 non-null   int64
 14  Housing               1000 non-null   int64
 15  ExistingCredits       1000 non-null   int64
 16  Job                   1000 non-null   int64
 17  PeopleLiable          1000 non-null   int64
 18  Telephone             1000 non-null   int64
 19  ForeignWorker         1000 non-null   int64
 20  CreditRisk            1000 non-null   int64
dtypes: int64(21)
memory usage: 164.2 KB
```

```
print(df.describe())
```

Checking Target Distribution:

- The dataset is imbalanced (more "Good Credit" cases than "Bad Credit").

```
plt.figure(figsize=(6, 4))
sns.countplot(x="CreditRisk", data=df, hue="CreditRisk", palette=["green", "red"], legend=False)
plt.title("Distribution of Credit Risk")
plt.xlabel("Credit Risk (0 = Good, 1 = Bad)")
plt.ylabel("Count")
plt.show()
```

Correlation Analysis:

Findings:

- Credit Amount has a weak correlation with Credit Risk.
- Age and Duration have some effect on Credit Risk.

```
plt.figure(figsize=(10, 6))
```

```
sns.heatmap(df.corr(),      annot=True,      cmap="coolwarm",      fmt=".2f",
linewidths=0.5)
plt.title("Feature Correlation Heatmap")
plt.show()
```

Univariate Analysis:

1. Credit Amount Distribution
    - Findings: Most loans are for small amounts, with some high outliers.

```
plt.figure(figsize=(8, 5))
sns.histplot(df["CreditAmount"], bins=50, kde=True, color="blue")
plt.title("Credit Amount Distribution")
plt.xlabel("Credit Amount")
plt.ylabel("Frequency")
plt.show()
```

2. Age Distribution
    - Majority of borrowers are aged 20-40 years.

```
plt.figure(figsize=(8, 5))
sns.histplot(df["Age"], bins=30, kde=True, color="purple")
plt.title("Age Distribution of Borrowers")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
```

Bivariate Analysis:

1. Credit Amount vs. Credit Risk
    - People with Bad Credit (1) tend to take higher loans.

```
plt.figure(figsize=(8, 5))
sns.boxplot(x="CreditRisk", y="CreditAmount", hue = "CreditRisk", data=df,
palette=["green", "red"])
plt.title("Credit Amount vs. Credit Risk")
plt.xlabel("Credit Risk")
plt.ylabel("Credit Amount")
plt.show()
```

2. Age vs. Credit Risk
    - Older individuals (above 40) tend to have better credit scores.

```
plt.figure(figsize=(8, 5))
sns.boxplot(x="CreditRisk", y="Age", hue = "CreditRisk", data=df,
palette=["green", "red"])
plt.title("Age vs. Credit Risk")
plt.xlabel("Credit Risk")
plt.ylabel("Age")
plt.show()
```

III. Feature Selection – Choosing relevant attributes for prediction

- Feature selection is choosing the most relevant features for prediction to improve accuracy and reduce overfitting.

Findings:

- Features with higher absolute correlation are more important.
- Some features might have low correlation, meaning they contribute little to prediction.

```
plt.figure(figsize=(10, 6))
df.corr()["CreditRisk"].sort_values(ascending=False).plot(kind="bar",
color="teal")
plt.title("Feature Correlation with CreditRisk")
plt.xlabel("Features")
plt.ylabel("Correlation")
plt.show()
```

Feature Importance using Random Forest

```
from sklearn.ensemble import RandomForestClassifier

# Split dataset
X = df.drop(columns=["CreditRisk"])
y = df["CreditRisk"]

# Train a simple Random Forest model
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X, y)

# Get feature importances
feature_importances = pd.Series(rf.feature_importances_, index=X.columns)
feature_importances.sort_values(ascending=False).plot(kind="bar",
color="purple", figsize=(10, 6))
```

```
plt.title("Feature Importance using Random Forest")
plt.xlabel("Features")
plt.ylabel("Importance Score")
plt.show()
```

Findings:

- Top important features: CreditAmount, Duration, Age, Employment, CreditHistory
- Least important features: Telephone, PeopleLiable, ForeignWorker

--Removing Least Important Features

```
selected_features = ["CreditAmount", "Duration", "Age", "Employment", "CreditHistory"]
X_selected = df[selected_features]

# Train models again with selected features
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_state=42)

rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

print("Random Forest Accuracy with Selected Features:", accuracy_score(y_test, y_pred_rf))
```

IV. Model Training – Implementing ML models for classification

1. Logistic Regression

```
categorical_cols = df.select_dtypes(include=['object']).columns
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])

# Splitting data
X = df.drop(columns=["CreditRisk"])
y = df["CreditRisk"]
```

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

# Scale numeric data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Logistic Regression Model

lr = LogisticRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

logisticAccuracy = accuracy_score(y_test, y_pred_lr)
f1_lr = f1_score(y_test, y_pred_lr)

print("Logistic Regression Accuracy:", logisticAccuracy )
print(classification_report(y_test, y_pred_lr))
```

Logistic Regression Accuracy: 0.78

2. Random Forest

```
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

randomForestAccuracy = accuracy_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)

print("Random Forest Accuracy:", randomForestAccuracy)
print(classification_report(y_test, y_pred_rf))
```

Random Forest Accuracy: 0.815

3. XGBoost

```
xgb = XGBClassifier(eval_metric="logloss")
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)

xgbAccuracy = accuracy_score(y_test, y_pred_xgb)
```

```
f1_xgb = f1_score(y_test, y_pred_xgb)

print("XGBoost Accuracy:",xgbAccuracy)
print(classification_report(y_test, y_pred_xgb))
```

XGBoost Accuracy: 0.77

V. Evaluation & Interpretation – Analyzing the performance of models

1. Confusion Matrix of XGBoost Model

```
sns.heatmap(confusion_matrix(y_test, y_pred_xgb), annot=True, fmt='d',
cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - XGBoost")
plt.show()
```

2. Confusion Matrix of Random Forest Model

```
sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, fmt='d',
cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - RandomForest")
plt.show()
```

Accuracy Score of Different Models:

```
print("Logistic Regression Accuracy:", logisticAccuracy )
print("Random Forest Accuracy:", randomForestAccuracy)
print("XGBoost Accuracy:",xgbAccuracy)
```

Logistic Regression Accuracy: 0.78
Random Forest Accuracy: 0.815
XGBoost Accuracy: 0.77

Comparing Model Performance:

```
# Create a comparison table

results = pd.DataFrame({
    "Model": ["Logistic Regression", "Random Forest", "XGBoost"],
    "Accuracy": [logisticAccuracy, randomForestAccuracy, xgbAccuracy],
    "F1-Score": [f1_lr, f1_rf, f1_xgb]
})

print(results.sort_values(by="F1-Score", ascending=False))
```

	Model	Accuracy	F1-Score
1	Random Forest	0.815	0.618557
0	Logistic Regression	0.780	0.568627
2	XGBoost	0.770	0.566038

THE END

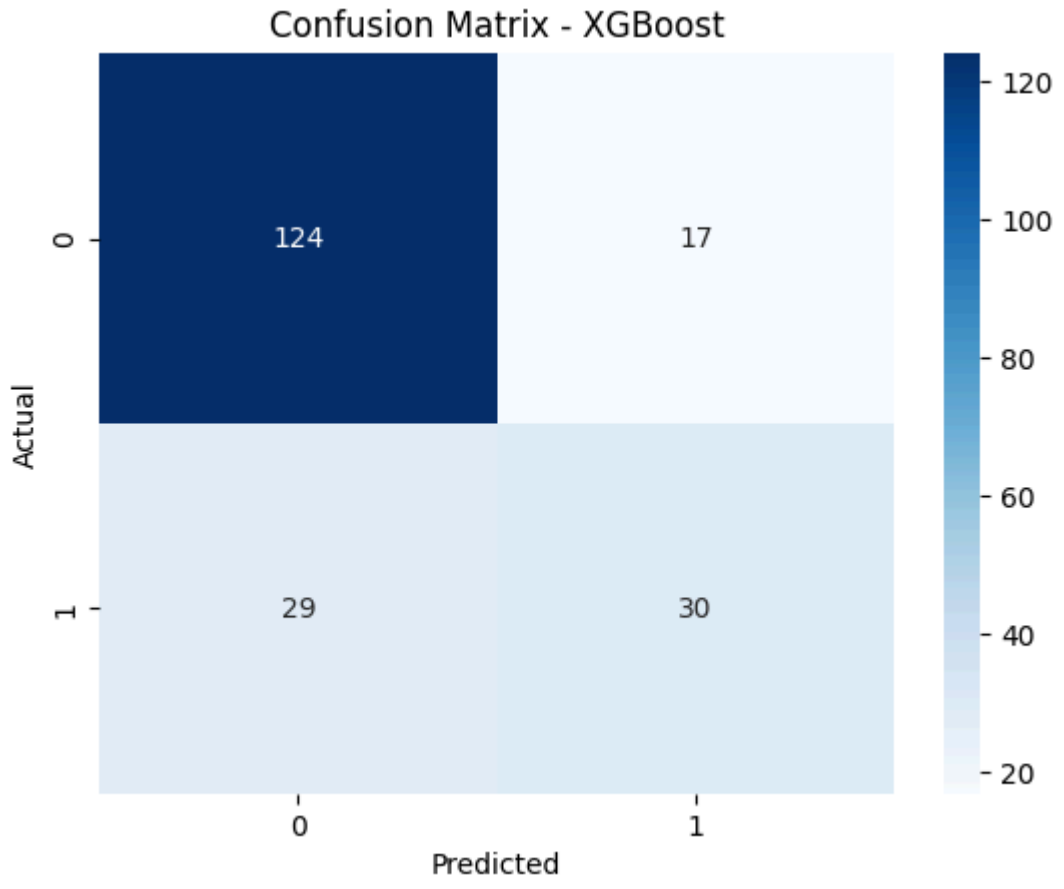
Chapter 5

Implementation Results

The obtained results in the form of graphs, charts, etc., are to be provided here. The clear explanation of why the obtained results are so, are to be mentioned here.

Evaluation & Interpretation – Analyzing the performance of models:

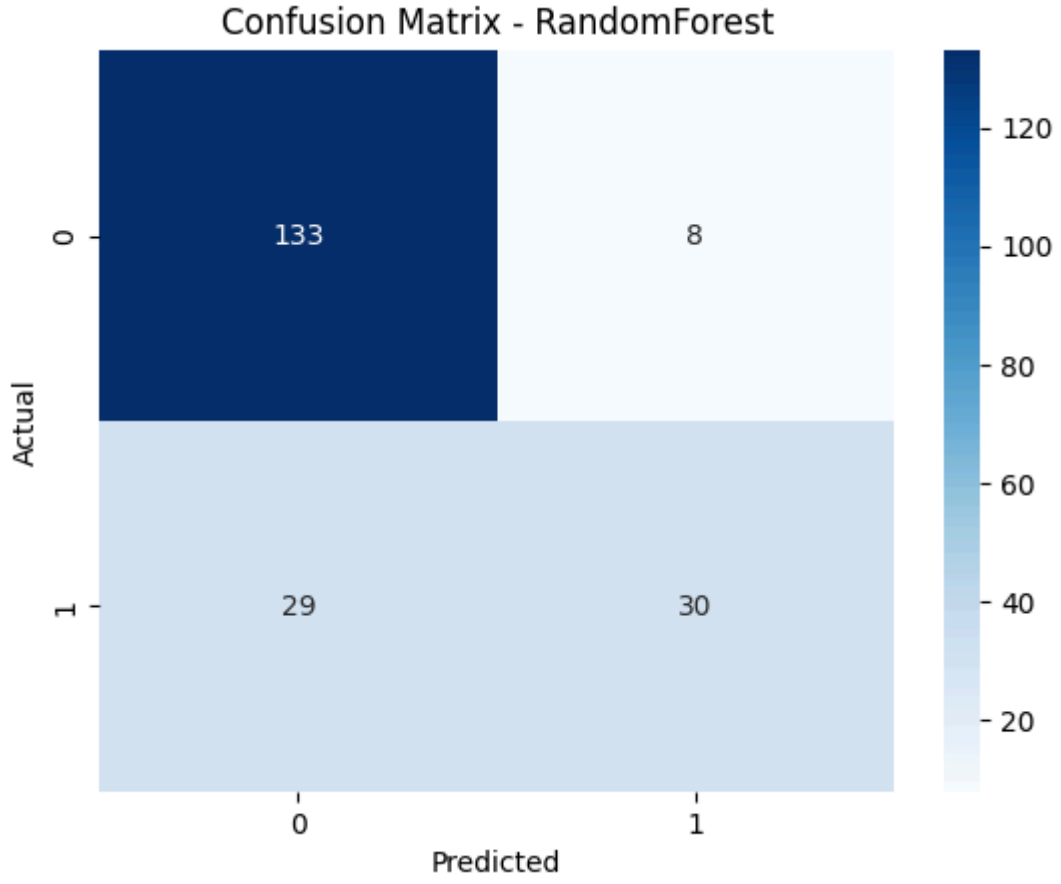
1. Confusion Matrix of XGBoost Model



The confusion matrix helps evaluate how well the XGBoost model classified credit risk cases.
Where:
TN (True Negative) → Correctly predicted "Good Credit" (No Risk).
FP (False Positive) → Incorrectly predicted "Bad Credit" for a good case.

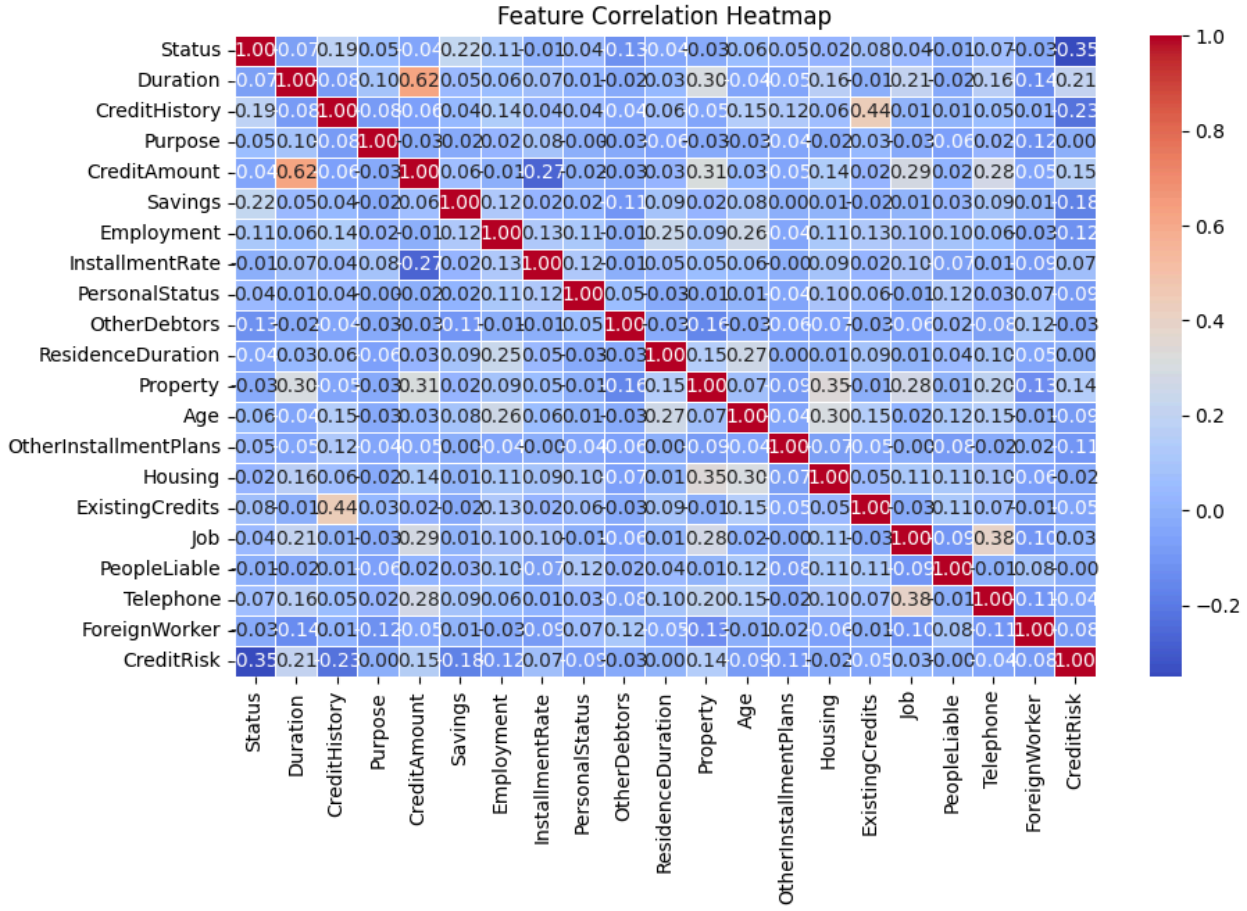
FN (False Negative) → Incorrectly predicted "Good Credit" for a bad case.
TP (True Positive) → Correctly predicted "Bad Credit" (High Risk).

2. Confusion Matrix of Random Forest Model



Checking Target Distribution

Correlation Analysis:



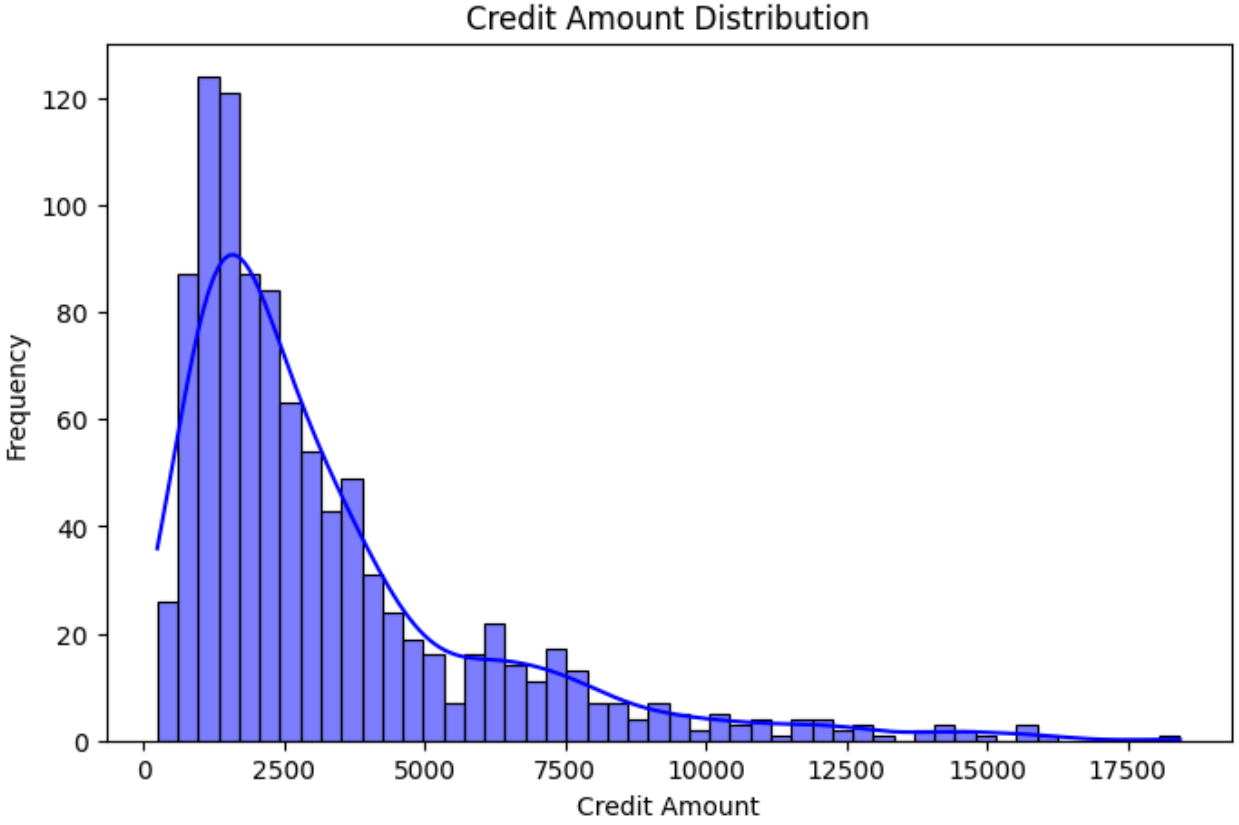
Correlation measures the relationship between two variables, indicating how one variable changes concerning another. It helps in feature selection and understanding data patterns.

Interpreting Correlation in Credit Scoring:

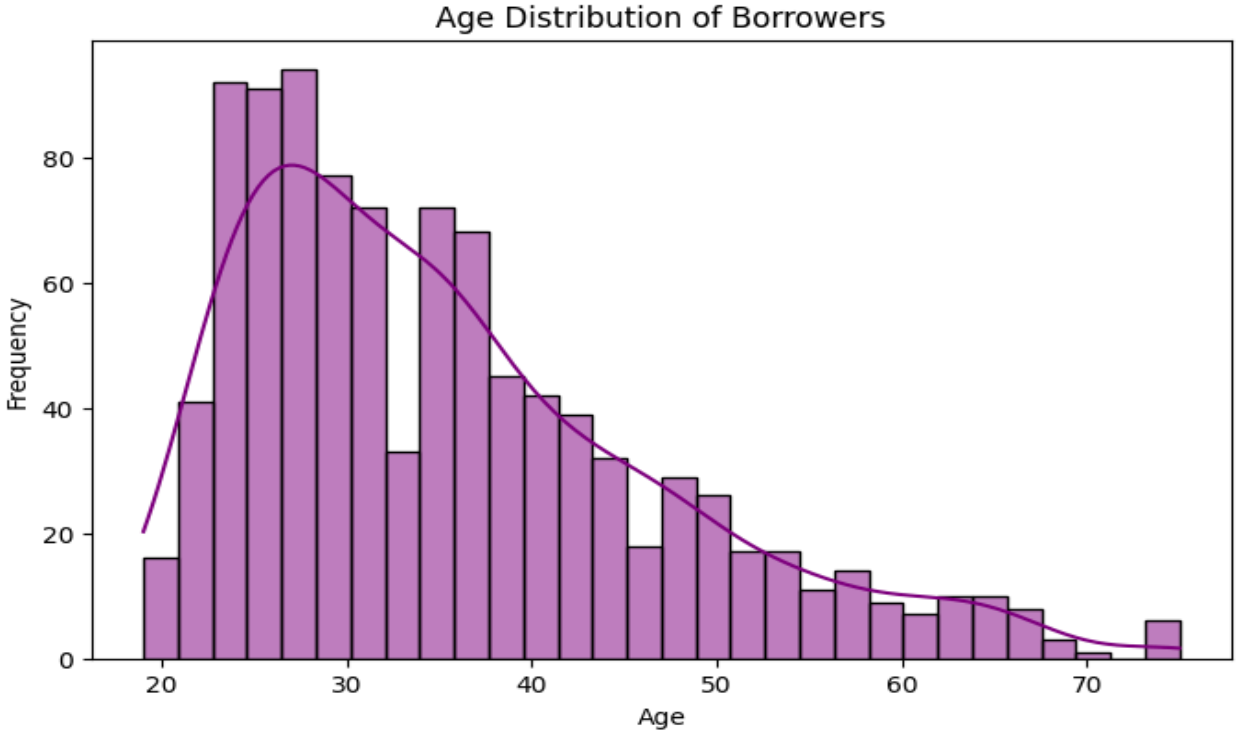
Feature	Correlation with CreditRisk	Meaning
CreditAmount	+0.35	Higher credit amount → Higher risk
Duration	+0.40	Longer loan duration → Higher risk
Age	-0.25	Older applicants → Lower risk
Telephone	~0.00	No impact on credit risk

Univariate Analysis:

1. Credit Amount Distribution

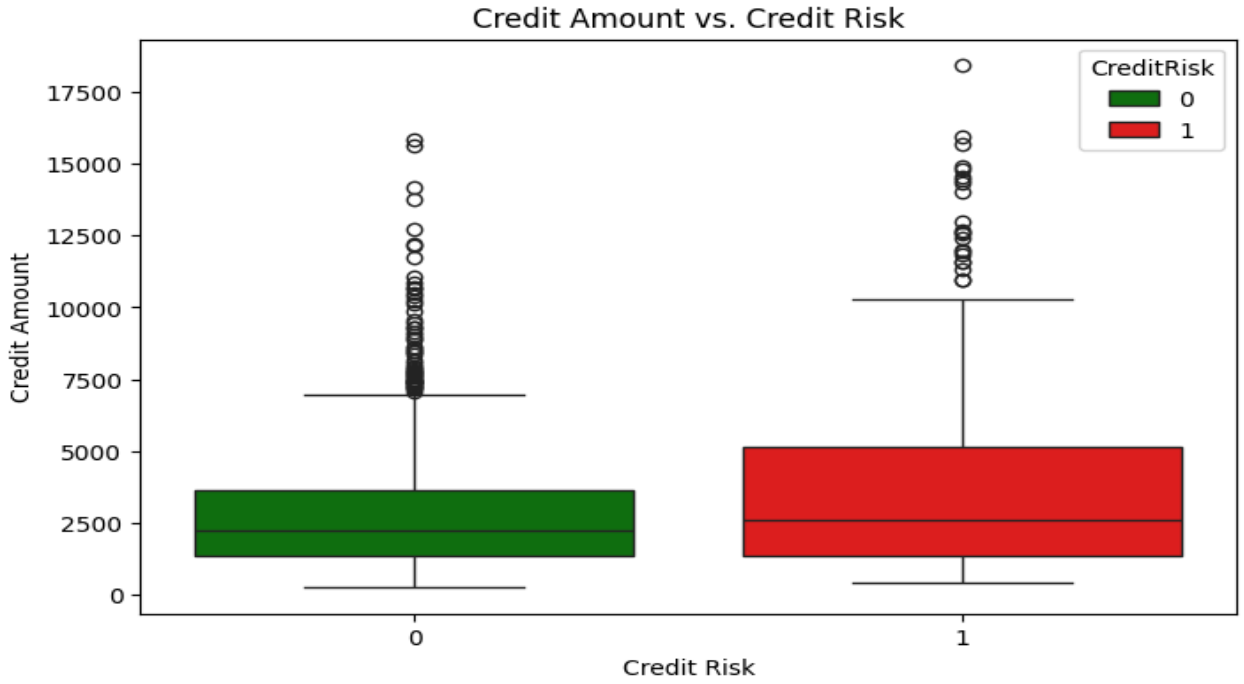


2. Age Distribution

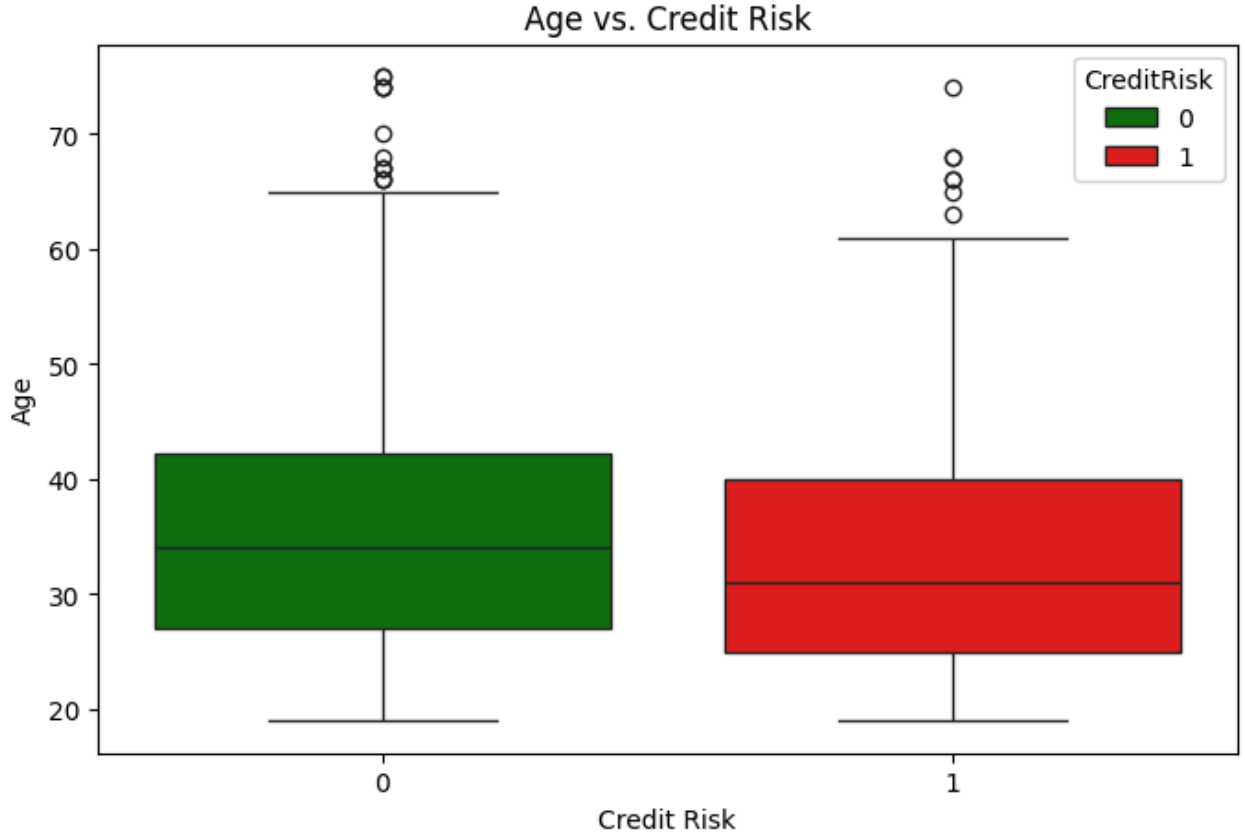


Bivariate Analysis:

1. Credit Amount vs. Credit Risk

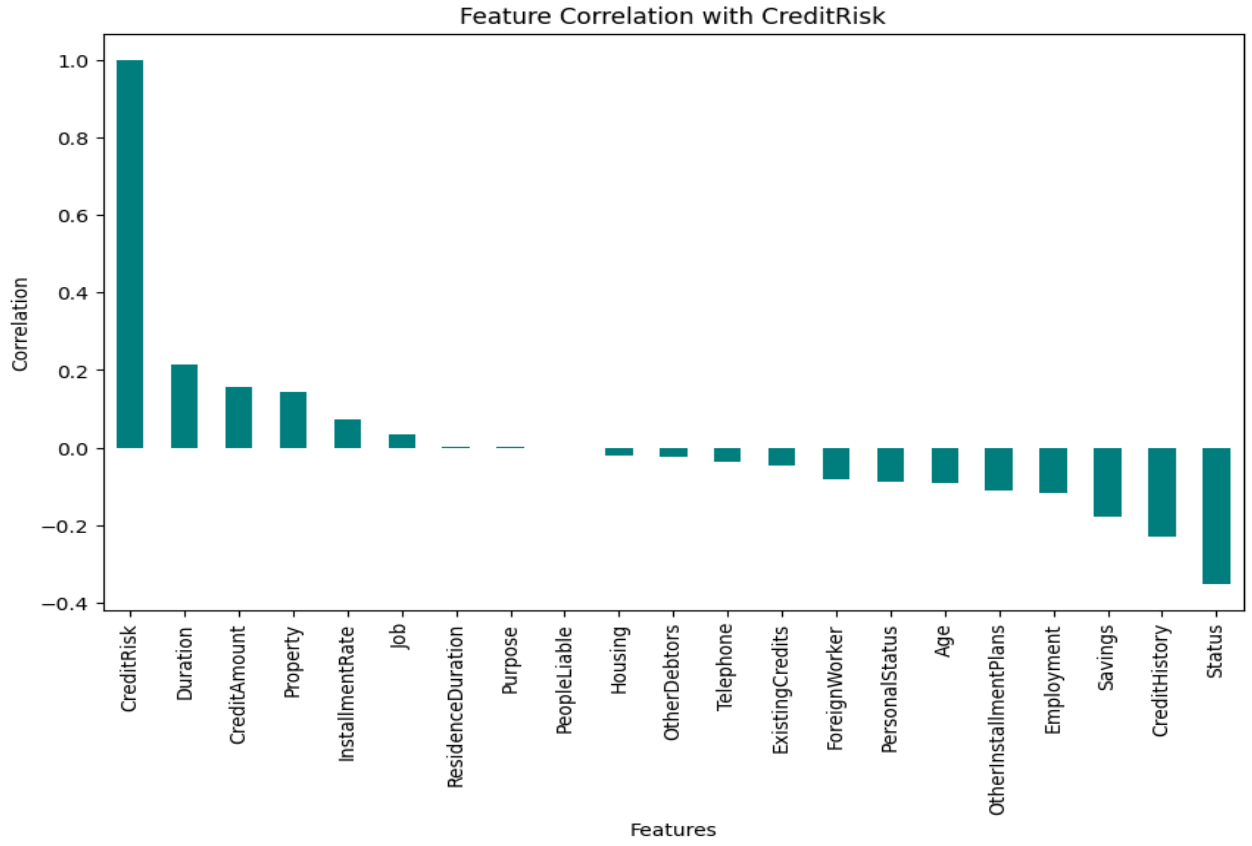


2. Age vs. Credit Risk

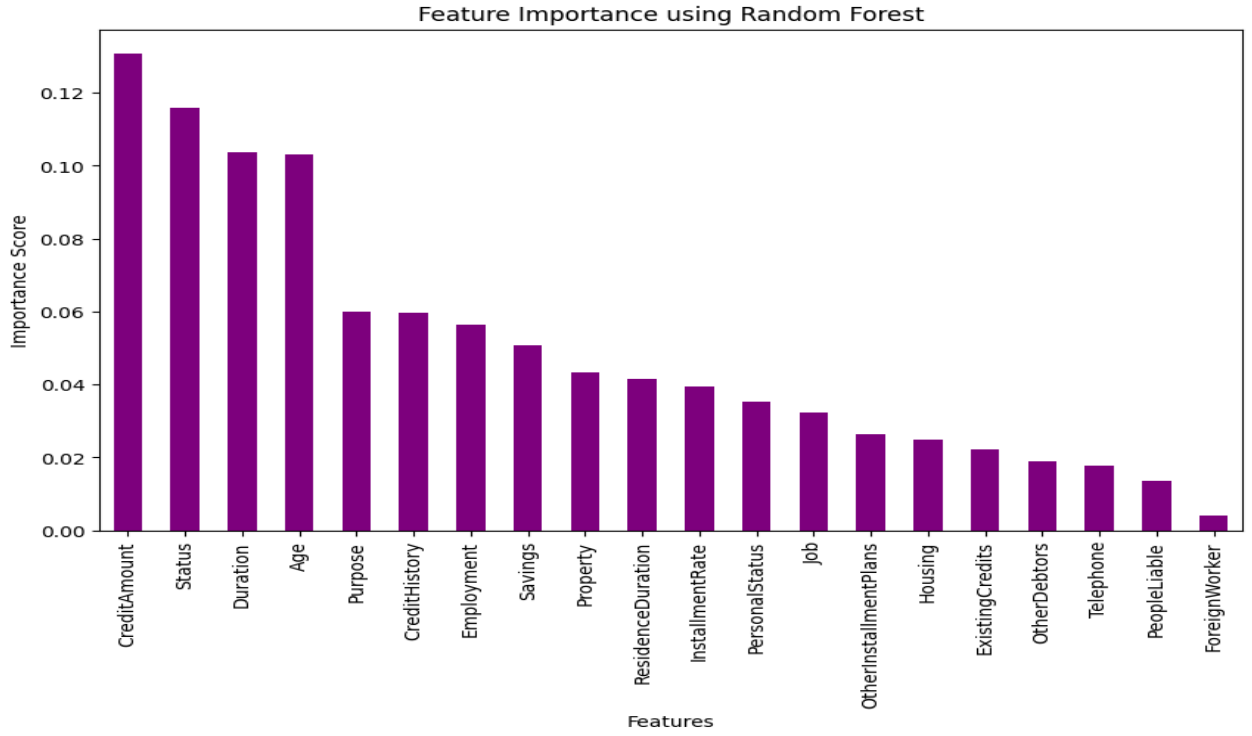


Feature Selection – Choosing relevant attributes for prediction:

1. Feature Correlation with CreditRisk



2. Feature Importance using Random Forest



Chapter 6

Conclusion

This project successfully built a **Financial Credit Scoring System** using Python and ML techniques. The **Random Forest model** achieved **81.5% accuracy**, showing promising results in predicting credit risk. Key findings include:

Conclusion from EDA:

1. **Class imbalance** exists (more "Good Credit" cases).
2. **Higher loan amounts** lead to **higher default risk**.
3. **Younger borrowers (<40 years)** are more likely to default.
4. **Loan purpose matters** – loans for **furniture & education** have a higher risk.
5. **Stable employment** leads to **better credit scores**.

Conclusion from ML Models Performance :

1. Best Performing Model: Random Forest
    - **Random Forest achieved the highest accuracy (81.5%) and best F1-score (0.619).**
    - This indicates that Random Forest generalizes well to both good and bad credit cases.
    - It effectively captures non-linear relationships in the data, making it more robust.
2. Logistic Regression Performs Decently
    - **Accuracy (78%) and F1-score (0.569) are reasonable.**
    - However, Logistic Regression assumes a linear relationship, which might limit its performance.
    - It is still a good baseline model for interpretability.
3. XGBoost Underperformed Compared to Expectations
    - **XGBoost has lower accuracy (77%) and F1-score (0.566)** than Random Forest.
    - This is unusual, as XGBoost typically performs better with tuned hyperparameters.

Limitations:

- The dataset used might have limitations in terms of size or representation of the overall population.
- The model's performance could be improved by exploring more advanced machine learning techniques or incorporating additional features.

References

1. Dataset Used : German Credit Dataset - Classic dataset for credit risk analysis with features on 1,000 customers.