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

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder,
LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier,
AdaBoostClassifier, GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report, mean_absolute_error, mean_squared_error,
r2_score

# to perform statistical test
from sklearn.feature_selection import chi2 # for categorical fetures
from sklearn.feature_selection import f_classif # for numerical
features (Anova f-test)

# impot pipeline
from sklearn.pipeline import Pipeline

import warnings
warnings.filterwarnings('ignore')

from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer


df = pd.read_csv("Credit_Score_Clean.csv")

df.head()
```

```
   Age Occupation  Annual_Income  Num_Bank_Accounts
Num_Credit_Card  \
0   23  Scientist       19114.12                  3                 4

1   23  Scientist       19114.12                  3                 4

2   23  Scientist       19114.12                  3                 4

3   23  Scientist       19114.12                  3                 4

4   28    Teacher       34847.84                  2                 4
```

```
   Interest_Rate  Num_of_Loan  Delay_from_due_date
Num_of_Delayed_Payment  \
0              3            4                    5
4
1              3            4                    6
0
2              3            4                    3
8
3              3            4                    3
6
4              6            1                    7
1

   Changed_Credit_Limit  ...  Credit_Mix Outstanding_Debt  \
0                  6.27  ...        Good           809.98
1                 11.27  ...        Good           809.98
2                 11.27  ...        Good           809.98
3                 11.27  ...        Good           809.98
4                  7.42  ...        Good           605.03

   Credit_Utilization_Ratio  Payment_of_Min_Amount Total_EMI_per_month
\
0                 31.377862                     No           49.574949

1                 24.797347                     No           49.574949

2                 22.537593                     No           49.574949

3                 23.933795                     No           49.574949

4                 38.550848                     No           18.816215


   Amount_invested_monthly                 Payment_Behaviour
Monthly_Balance  \
0               199.458074    Low_spent_Small_value_payments
223.451310
1                41.420153  High_spent_Medium_value_payments
341.489231
2               178.344067    Low_spent_Small_value_payments
244.565317
3                24.785217  High_spent_Medium_value_payments
358.124168
4                40.391238   High_spent_Large_value_payments
484.591214

   Credit_Score Credit_History_Age_Months
0          Good                       268
```

```
1         Good                          269
2         Good                          271
3     Standard                            0
4         Good                          320

[5 rows x 21 columns]
```

```python
print(f'Dataset has {df.shape[0]} rows and {df.shape[1]} columns')
```

```
Dataset has 31711 rows and 21 columns
```

```python
numerical_features =
df.select_dtypes(include=np.number).columns.tolist()
print("Numerical Features:", numerical_features)
```

```
Numerical Features: ['Age', 'Annual_Income', 'Num_Bank_Accounts',
'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', 'Total_EMI_per_month',
'Amount_invested_monthly', 'Monthly_Balance',
'Credit_History_Age_Months']
```

```python
categorical_features =
df.select_dtypes(include=['object']).columns.tolist()
print("Categorical Features:", categorical_features)
```

```
Categorical Features: ['Occupation', 'Credit_Mix',
'Payment_of_Min_Amount', 'Payment_Behaviour', 'Credit_Score']
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31711 entries, 0 to 31710
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       31711 non-null  int64
 1   Occupation                31711 non-null  object
 2   Annual_Income             31711 non-null  float64
 3   Num_Bank_Accounts         31711 non-null  int64
 4   Num_Credit_Card           31711 non-null  int64
 5   Interest_Rate             31711 non-null  int64
 6   Num_of_Loan               31711 non-null  int64
 7   Delay_from_due_date       31711 non-null  int64
 8   Num_of_Delayed_Payment    31711 non-null  int64
 9   Changed_Credit_Limit      31711 non-null  float64
 10  Num_Credit_Inquiries      31711 non-null  float64
 11  Credit_Mix                31711 non-null  object
 12  Outstanding_Debt          31711 non-null  float64
 13  Credit_Utilization_Ratio  31711 non-null  float64
```

```
 14  Payment_of_Min_Amount      31711 non-null  object 
 15  Total_EMI_per_month        31711 non-null  float64
 16  Amount_invested_monthly    31711 non-null  float64
 17  Payment_Behaviour          31711 non-null  object 
 18  Monthly_Balance            31711 non-null  float64
 19  Credit_Score               31711 non-null  object 
 20  Credit_History_Age_Months  31711 non-null  int64  
dtypes: float64(8), int64(8), object(5)
memory usage: 5.1+ MB
```

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

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

```python
df.describe()
```

```
                Age  Annual_Income  Num_Bank_Accounts  Num_Credit_Card
\
count  31711.000000   3.171100e+04       31711.000000     31711.000000

mean      35.135032   1.749045e+05           4.415818         4.801583

std       11.037186   1.415577e+06           2.305062         1.673844

min       14.000000   7.006520e+03           0.000000         0.000000

25%       26.000000   2.211810e+04           3.000000         4.000000

50%       35.000000   3.699394e+04           4.000000         5.000000
```

```
75%       44.000000   7.452061e+04           6.000000         6.000000

max       56.000000   2.419806e+07          10.000000        10.000000


       Interest_Rate   Num_of_Loan  Delay_from_due_date  \
count   31711.000000  31711.000000         31711.000000
mean       10.256504      2.234114            14.985967
std         5.916633      1.700965             9.353937
min         1.000000      0.000000             0.000000
25%         6.000000      1.000000             8.000000
50%         9.000000      2.000000            13.000000
75%        14.000000      3.000000            22.000000
max        34.000000      9.000000            60.000000

       Num_of_Delayed_Payment  Changed_Credit_Limit
Num_Credit_Inquiries  \
count            31711.000000          31711.000000
31711.000000
mean                26.493299              8.601820
3.903030
std                215.388313              5.119076
2.813889
min                  0.000000              0.000000
0.000000
25%                  6.000000              4.550000
2.000000
50%                 11.000000              8.370000
4.000000
75%                 15.000000             11.620000
6.000000
max               4397.000000             26.900000
12.000000

       Outstanding_Debt  Credit_Utilization_Ratio  Total_EMI_per_month
\
count      31711.000000              31711.000000         31711.000000

mean         776.983756                 32.522218            59.287714

std          443.968460                  5.135545            53.461204

min            0.230000                 20.832487             0.000000

25%          388.920000                 28.299138            16.414812

50%          780.210000                 32.501616            46.162077

75%         1182.500000                 36.731398            89.163419
```

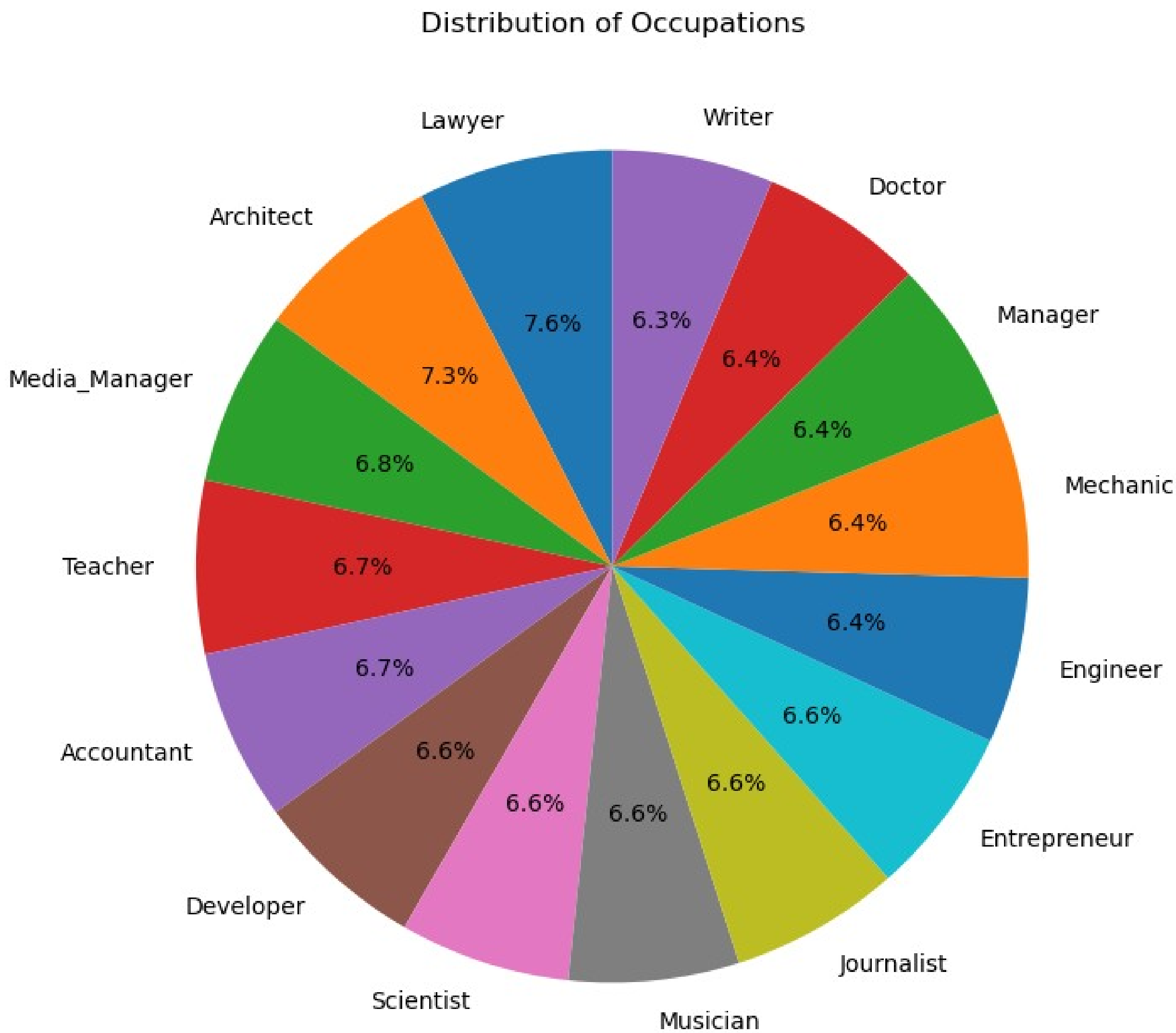
```
max          1499.920000                 49.564519           199.904691


       Amount_invested_monthly  Monthly_Balance
Credit_History_Age_Months
count              31711.000000     31711.000000
31711.000000
mean                 181.502288       439.647331
243.861026
std                  196.253121       225.424866
108.853693
min                    0.000000         0.000000
0.000000
25%                   61.938256       293.841559
195.000000
50%                  121.191802       369.698223
256.000000
75%                  225.891543       523.103061
329.000000
max                 1903.080048      1602.040519
404.000000
```

```python
df['Occupation'].unique()
```

```
array(['Scientist', 'Teacher', 'Entrepreneur', 'Developer', 'Lawyer',
       'Journalist', 'Engineer', 'Accountant', 'Musician',
'Architect',
       'Writer', 'Manager', 'Media_Manager', 'Doctor', 'Mechanic'],
      dtype=object)
```

```python
occupation_counts = df['Occupation'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(occupation_counts, labels=occupation_counts.index,
autopct='%1.1f%%', startangle=90)
plt.title('Distribution of Occupations')
plt.show()
```




```
df['Credit_Mix'].unique()

array(['Good', 'Standard', 'Bad'], dtype=object)

df['Payment_of_Min_Amount'].unique()

array(['No', 'NM', 'Yes'], dtype=object)


df['Payment_Behaviour'].unique()

array(['Low_spent_Small_value_payments',
       'High_spent_Medium_value_payments',
       'High_spent_Large_value_payments',
       'Low_spent_Medium_value_payments',
```

```
       'Low_spent_Large_value_payments',
       'High_spent_Small_value_payments'], dtype=object)
```

```
df['Credit_Score'].unique()
```

```
array(['Good', 'Standard', 'Poor'], dtype=object)
```

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

```
Credit_Score
Standard    19730
Good         7551
Poor         4430
Name: count, dtype: int64
```

```
# Create a new column 'Credit_Score' with 1 for 'Good' and 'Standard'
and 0 for 'Poor'
df['Credit_Score'] = df['Credit_Score'].apply(lambda x: 1 if x in
['Good', 'Standard'] else 0)

# Now you have a binary classification target variable
print(df['Credit_Score'].value_counts())
```

```
Credit_Score
1    27281
0     4430
Name: count, dtype: int64
```

```
df.sample(5)
```

```
       Age  Occupation  Annual_Income  Num_Bank_Accounts
Num_Credit_Card  \
25371   23      Doctor      90794.280                  5
4
15027   22      Writer      11980.385                  5
3
8406    48     Manager      29648.760                  0
2
6537    26    Mechanic       9127.260                  7
9
16501   35  Accountant       7309.155                  6
8

       Interest_Rate  Num_of_Loan  Delay_from_due_date  \
25371              5            3                   27
15027             18            3                   25
8406              10            0                    6
6537              26            2                   41
16501             19            4                   38
```

```
       Num_of_Delayed_Payment  Changed_Credit_Limit  ...
Credit_Mix  \
25371                      16                 14.34  ...    Standard

15027                      18                 17.40  ...    Standard

8406                        3                  4.85  ...        Good

6537                       24                  1.65  ...         Bad

16501                      11                  9.56  ...    Standard


      Outstanding_Debt  Credit_Utilization_Ratio
Payment_of_Min_Amount  \
25371           622.21                 38.602708
Yes
15027            29.81                 35.954519
Yes
8406           1104.99                 36.543651
No
6537           1399.89                 37.419987
Yes
16501          1318.12                 31.740847
Yes

      Total_EMI_per_month  Amount_invested_monthly  \
25371          166.918982               218.653934
15027           26.265887                65.662145
8406             0.000000               179.861535
6537            12.881162                30.135925
16501           20.290771                54.800691

                     Payment_Behaviour Monthly_Balance
Credit_Score  \
25371  Low_spent_Medium_value_payments      668.146083             0

15027   Low_spent_Small_value_payments      301.108509             1

8406   Low_spent_Medium_value_payments      374.211465             1

6537    Low_spent_Large_value_payments      326.143413             0

16501  Low_spent_Medium_value_payments      261.618163             0


       Credit_History_Age_Months
25371                        211
15027                        380
8406                         250
6537                         101
```

```
16501                              94

[5 rows x 21 columns]
```

```
categorical_cols = ['Occupation', 'Credit_Mix',
'Payment_of_Min_Amount', 'Payment_Behaviour', 'Credit_Score']

# Label encode each categorical column
label_enc = LabelEncoder()
for col in categorical_cols:
    df[col] = label_enc.fit_transform(df[col])


# Define X1 (independent variables) and y1 (target)
X1 = df[['Occupation', 'Credit_Mix', 'Payment_of_Min_Amount',
'Payment_Behaviour']]
y1 = df['Credit_Score']

# Perform Chi-Square test
chi_scores = chi2(X1, y1)
chi2_scores = pd.DataFrame({"Feature": X1.columns, "Score":
chi_scores[0]})
print(chi2_scores.sort_values(by="Score", ascending=False))
```

```
                 Feature       Score
1             Credit_Mix  288.745515
3      Payment_Behaviour  127.452384
0             Occupation    4.753546
2  Payment_of_Min_Amount    0.001643
```

```
X_num_for_f_test = df[['Age', 'Annual_Income', 'Num_Bank_Accounts',
'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', 'Total_EMI_per_month',
'Amount_invested_monthly', 'Monthly_Balance',
'Credit_History_Age_Months']]


y_for_f_test = df['Credit_Score']

f_scores, p_values = f_classif(X_num_for_f_test, y_for_f_test)

f_scores_df = pd.DataFrame({'Feature': X_num_for_f_test.columns, 'F-
Score': f_scores})

print(f_scores_df.sort_values(by='F-Score', ascending=False))
```

```
                      Feature      F-Score
6         Delay_from_due_date  1290.385124
3             Num_Credit_Card  1030.110256
9        Num_Credit_Inquiries   585.906436
```

```
4                 Interest_Rate   453.047239
10             Outstanding_Debt   451.549924
5                   Num_of_Loan   274.356620
15  Credit_History_Age_Months    92.515418
12         Total_EMI_per_month    28.425328
8         Changed_Credit_Limit    27.130593
13    Amount_invested_monthly    20.350875
0                           Age    11.370137
2             Num_Bank_Accounts     6.047901
11   Credit_Utilization_Ratio     2.873588
7       Num_of_Delayed_Payment     2.290066
1                 Annual_Income     0.064910
14              Monthly_Balance     0.030461

selected_features = [
    'Credit_Mix', 'Payment_of_Min_Amount', 'Payment_Behaviour',
    'Delay_from_due_date', 'Interest_Rate', 'Num_Credit_Card',
    'Num_Bank_Accounts', 'Changed_Credit_Limit',
'Num_Credit_Inquiries',
    'Num_of_Loan', 'Outstanding_Debt', 'Occupation'
]


X = df[selected_features]
y = df['Credit_Score']  # Assuming 'loan_eligibility' is the target


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


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)



    from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier, AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris

# Example data (replace with your dataset)
```

```python
data = load_iris()
X = data.data
y = data.target

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# List of models
models = [
    ('Random Forest', RandomForestClassifier(random_state=42)),
    ('Gradient Boosting',
GradientBoostingClassifier(random_state=42)),
    ('Support Vector Machine', SVC(random_state=42)),
    ('Logistic Regression', LogisticRegression(random_state=42,
max_iter=1000)),
    ('K-Nearest Neighbors', KNeighborsClassifier()),
    ('Decision Tree', DecisionTreeClassifier(random_state=42)),
    ('Ada Boost', AdaBoostClassifier(random_state=42)),
    ('Naive Bayes', GaussianNB())

]

best_model = None
best_accuracy = 0.0


    # Cross-validation on training data
    scores = cross_val_score(pipeline, X_train, y_train, cv=5)
    mean_accuracy = scores.mean()

    # Fit and predict
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    test_accuracy = accuracy_score(y_test, y_pred)

    # Print metrics
    print(f"Model: {name}")
    print(f"Cross-validation Accuracy: {mean_accuracy:.4f}")
    print(f"Test Accuracy: {test_accuracy:.4f}")
    print()

    # Update best model
    if test_accuracy > best_accuracy:
        best_accuracy = test_accuracy
        best_model = pipeline

print("Best Model:", best_model.named_steps['model'])
```

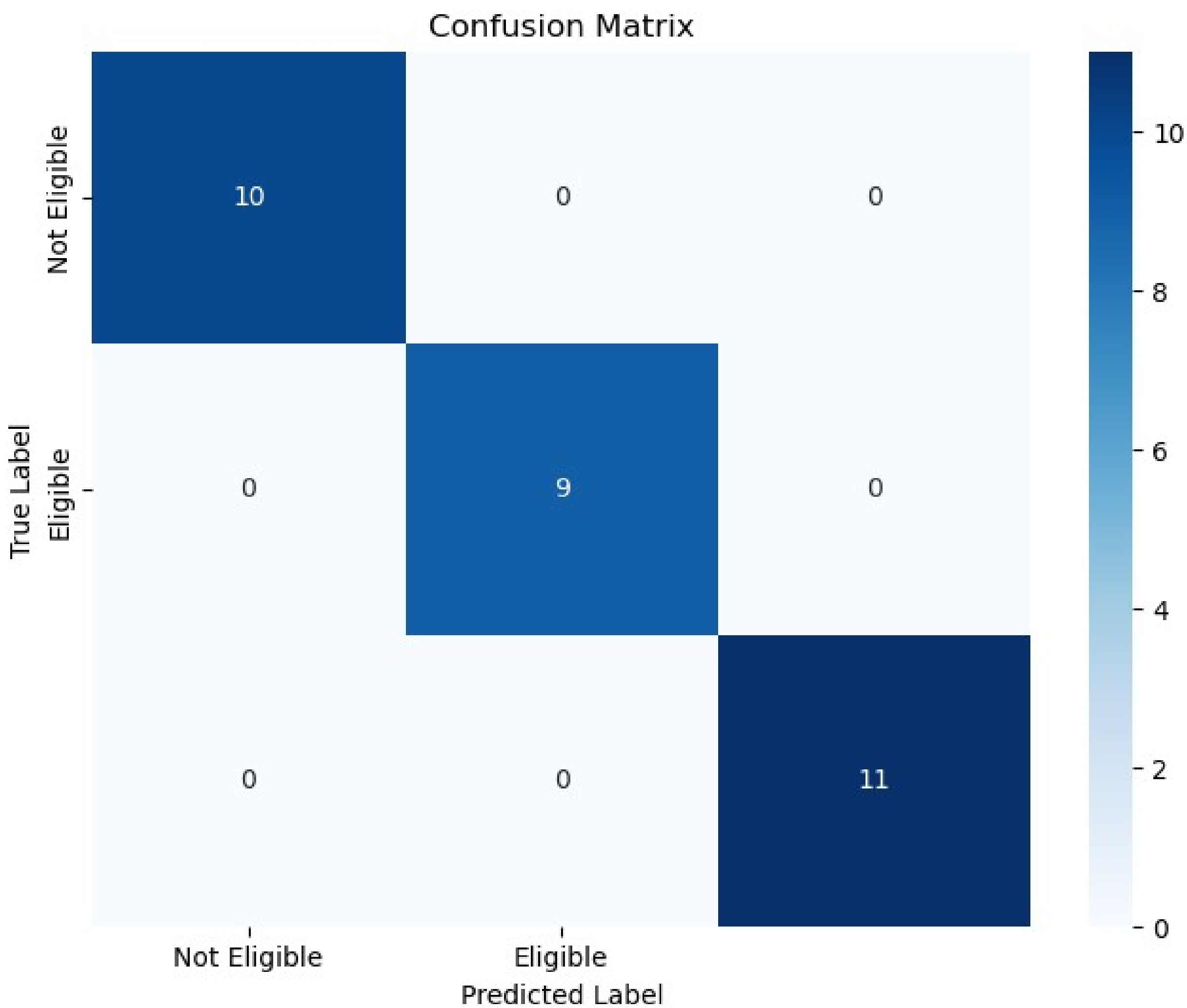
```
Model: Naive Bayes
Cross-validation Accuracy: 0.9417
Test Accuracy: 1.0000

Best Model: GaussianNB()
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'y_pred' contains the predictions from your best model
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Not Eligible', 'Eligible'],
            yticklabels=['Not Eligible', 'Eligible'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```




```python
import pickle
pickle.dump(best_model, open('01_credit_scoring_model.pkl', 'wb'))
loaded_model = best_model

sample_data = pd.DataFrame({
    'Credit_Mix': [2],
    'Payment_of_Min_Amount': [1],
    'Payment_Behaviour': [0],
    'Delay_from_due_date': [10],
    'Interest_Rate': [10.5],
    'Num_Credit_Card': [3],
    'Num_Bank_Accounts': [5],
    'Changed_Credit_Limit': [1],
    'Num_Credit_Inquiries': [2],
    'Num_of_Loan': [2],
    'Outstanding_Debt': [5000],
    'Occupation': [2]
```

```
})


# Make prediction
model.predict
if prediction == 1:
  print("User should be given a loan.")
else:
  print("User should not be given a loan.")
```

```
User should not be given a loan.
```