Sri Lanka Institute of Information Technology

Financial Risk Prediction

Project Report

Fundamental of Data Mining – IT3051

Group 16/Data Miners

Submitted by:

1. Bandara K M W G L A – IT22089304
2. Jayasinghe J M H C – IT22597342
3. Abeysinghe E A – IT22166692
4. Wicramasinghe D A T N – IT22113504
5. Mesandu W M S – IT22108890

Submitted to:

Dr. Prasanna S. Haddela
Name of the supervisor/lecturer in charge

10/03/2024
Date of submission

Video link : WhatsApp Video 2024-10-03 at 23.49.40.mp4
Deployed link:
Github link: https://github.com/IT22089304/Financial-risk-prediction.git

Contents

Introduction

Project Background

This project focuses on building a machine learning model to evaluate financial risks for individuals applying for loans, aiming to reduce defaults and improve credit risk prediction accuracy. In the financial industry, assessing an individual's risk profile is crucial for minimizing loan defaults and ensuring responsible lending practices.

Our model will be developed using a publicly available dataset, which includes data on demographics, financial status, and past credit behavior of individuals. By analyzing this data, the model will predict the individual's financial risk level categorized as Low or High. The ultimate goal is to provide more accurate risk evaluations, helping financial institutions make informed decisions and manage credit risks effectively.

Problem Statement

Current Challenge

Financial institutions face difficulties in accurately assessing the credit risk of individuals applying for loans. Rigid scoring systems and other traditional credit evaluation techniques frequently fall short of capturing the complexity of an applicant's financial history and behavior. With regard to this, it may be challenging to estimate the actual risk of loan default using those approaches, which could result in inaccurate creditworthiness ratings. This challenge leads to higher default rates, increased financial risk for lenders, and inaccurate loan decisions.

Significance

Inaccurate risk predictions have adverse effects on lenders as well as loan applicants. For lenders, surmising incorrectly that an applicant is a lower risk leads to increased defaults on loans and hence lower profitability. Applicants, on the other hand, may be unfairly denied loans or offered unfavorable loan terms based on incorrect risk classification. This affects not only individuals' ability to access credit but also the fairness and efficiency of the overall lending process.

Proposed Solution

The purpose of this project is to create a model based on machine learning that will examine the financial risk status of loan applicants. The model utilizes a dataset comprising demographic data, economic data and credit history data predicting the risk level of the applicant low or high. By leveraging machine learning, the model will help to detect trends and relationships in the data that may not be obtained using the conventional approaches further making the model superior.

Impact

The implementation of a more accurate and data-driven financial risk assessment model will significantly reduce loan default rates, thereby enhancing the safety and profitability of financial institutions. It will also make credit evaluations more equitable by de-biasing the processes creating barriers to high-risk borrowers who have been misclassified in conventional screening methods. In the long term, this project will enhance the practices of lending and borrowing in the whole financial market to be more efficient, accountable and more transparent.

Dataset analysis and preparation

Variable	Variable Name	Description	Variable Type
1	Age	The applicant's age.	numerical
2	Gender	Gender (“Male”, “Female”, “Non-binary”)	categorical
3	Education	Education status (“PhD, “Master's”, “Bachelor's”, “High School”)	categorical
4	Marital Status	Education status (“divorced”, “married”, “single”, “widowed”)	categorical
5	Income	The annual income of the applicant.	numerical
6	Credit Score	The applicant's credit score, which reflects their creditworthiness.	numerical
7	Loan Amount	The amount of the loan being applied for.	numerical
8	Loan Purpose	Type of Loan purpose (“Business”, “Home”, “Personal”, “Auto”)	categorical
9	Employment Status	Employment Status (“Employed”, “Unemployed”, “Self-employed”)	categorical
10	Years at Current Job	The number of years the applicant has been at their current job.	numerical
12	Debt-to-Income Ratio	The ratio of the applicant's debt compared to their income.	numerical
13	Assets Value	The total value of the applicant’s assets.	numerical

14	Number of Dependents	The number of dependents the applicant is financially responsible for.	numerical
15	City	The city where the applicant resides. (Too many unique values to list here, some examples - “Port Elizabeth”, “South Scott”, “Robin haven”, “New Heather”)	categorical
16	State	The state where the applicant resides. (some examples-AS, OH, OK, PR)	categorical
17	Country	The country where the applicant resides. (some examples-Cyprus, Turkmenistan, Luxembourg, Uganda, Namibia)	categorical
18	Previous Defaults	The number of times the applicant has defaulted on a loan in the past.	numerical
19	Marital Status Change	Indicates any recent change in marital status.	numerical
20	Risk Rating	The final risk assessment of the applicant’s financial situation (Low or High).	categorical

Implementation

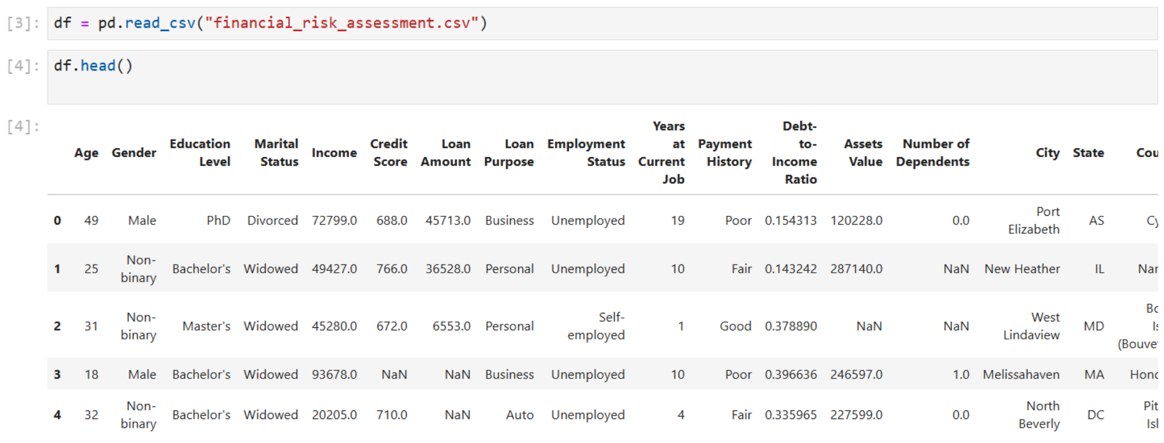
Data Preparation

The dataset is partitioned into:

- Training set - 80%
- Testing set - 20%

Data Preprocessing

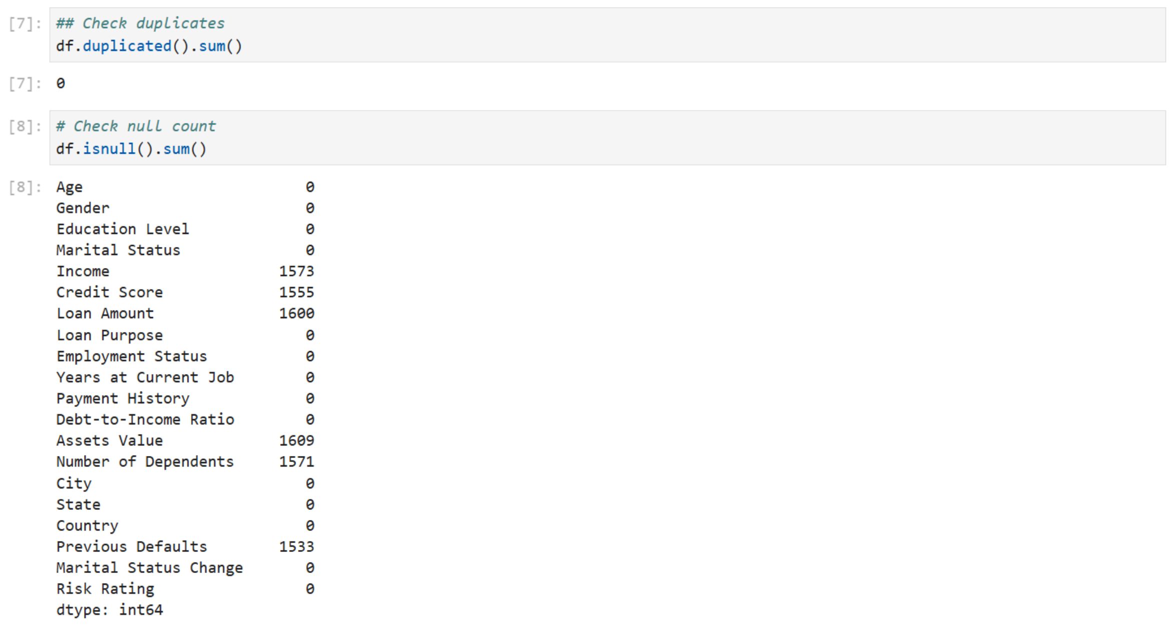
Import the data set



```
[3]: df = pd.read_csv("financial_risk_assessment.csv")

[4]: df.head()

[4]:
```

	Age	Gender	Education Level	Marital Status	Income	Credit Score	Loan Amount	Loan Purpose	Employment Status	Years at Current Job	Payment History	Debt-to-Income Ratio	Assets Value	Number of Dependents	City	State	Cou
0	49	Male	PhD	Divorced	72799.0	688.0	45713.0	Business	Unemployed	19	Poor	0.154313	120228.0	0.0	Port Elizabeth	AS	Cy
1	25	Non-binary	Bachelor's	Widowed	49427.0	766.0	36528.0	Personal	Unemployed	10	Fair	0.143242	287140.0	NaN	New Heather	IL	Nar
2	31	Non-binary	Master's	Widowed	45280.0	672.0	6553.0	Personal	Self-employed	1	Good	0.378890	NaN	NaN	West Lindaview	MD	Bo I (Bouve
3	18	Male	Bachelor's	Widowed	93678.0	NaN	NaN	Business	Unemployed	10	Poor	0.396636	246597.0	1.0	Melissahaven	MA	Honc
4	32	Non-binary	Bachelor's	Widowed	20205.0	710.0	NaN	Auto	Unemployed	4	Fair	0.335965	227599.0	0.0	North Beverly	DC	Pit Isl

Check duplicate values, missing values & garbage values.



```
[7]: ## Check duplicates
     df.duplicated().sum()

[7]: 0

[8]: # Check null count
     df.isnull().sum()

[8]: Age                        0
     Gender                     0
     Education Level            0
     Marital Status             0
     Income                  1573
     Credit Score            1555
     Loan Amount             1600
     Loan Purpose               0
     Employment Status          0
     Years at Current Job       0
     Payment History            0
     Debt-to-Income Ratio       0
     Assets Value            1609
     Number of Dependents    1571
     City                       0
     State                      0
     Country                    0
     Previous Defaults       1533
     Marital Status Change      0
     Risk Rating                0
     dtype: int64
```

```
[9]: # Check null count as percentage
     col_num=0
     TotalObjects =df.shape[0]
     print ("Column\t\t\t\t\t Null Values%")
     for x in df:
      nullCount =df[x].isnull().sum();
      nullPercent = nullCount*100 / (TotalObjects)
      print(str(x)+"\t\t\t\t\t "+str(nullPercent))
```

```
Column                                  Null Values%
Age                                     0.0
Gender                                  0.0
Education Level                                 0.0
Marital Status                                  0.0
Income                                  14.980952380952381
Credit Score                                    14.80952380952381
Loan Amount                                     15.238095238095237
Loan Purpose                                    0.0
Employment Status                                       0.0
Years at Current Job                                    0.0
Payment History                                 0.0
Debt-to-Income Ratio                                    0.0
Assets Value                                    15.323809523809524
Number of Dependents                                    14.961904761904762
City                                    0.0
State                                   0.0
Country                                 0.0
Previous Defaults                                       14.6
Marital Status Change                                   0.0
```

```
[11]: #identifying garbage values
      for i in df.select_dtypes(include="object").columns:
          print(df[i].value_counts())
          print("***"*10)
```

```
Gender
Non-binary    3565
Female        3499
Male          3436
Name: count, dtype: int64
******************************
Education Level
Bachelor's     2677
High School    2627
PhD            2624
Master's       2572
Name: count, dtype: int64
******************************
Marital Status
Widowed       2713
```

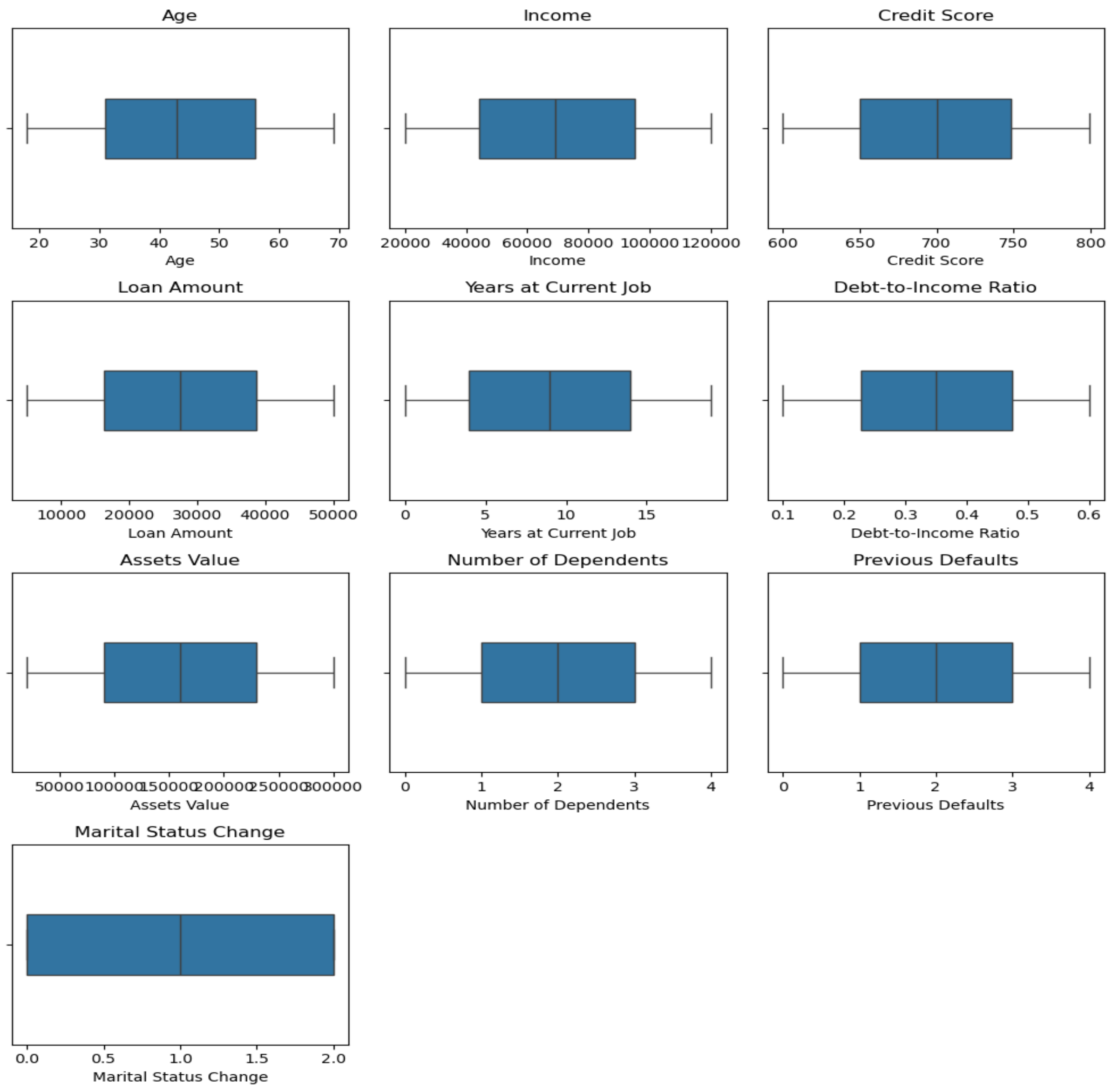
Check data types

```
[10]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10500 entries, 0 to 10499
Data columns (total 20 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Age                    10500 non-null  int64
 1   Gender                 10500 non-null  object
 2   Education Level        10500 non-null  object
 3   Marital Status         10500 non-null  object
 4   Income                 8927 non-null   float64
 5   Credit Score           8945 non-null   float64
 6   Loan Amount            8900 non-null   float64
 7   Loan Purpose           10500 non-null  object
 8   Employment Status      10500 non-null  object
 9   Years at Current Job   10500 non-null  int64
 10  Payment History        10500 non-null  object
 11  Debt-to-Income Ratio   10500 non-null  float64
 12  Assets Value           8891 non-null   float64
 13  Number of Dependents   8929 non-null   float64
 14  City                   10500 non-null  object
 15  State                  10500 non-null  object
 16  Country                10500 non-null  object
 17  Previous Defaults      8967 non-null   float64
 18  Marital Status Change  10500 non-null  int64
 19  Risk Rating            10500 non-null  object
dtypes: float64(7), int64(3), object(10)
memory usage: 1.6+ MB
```

Check outliers

```
[31]: import warnings
      import matplotlib.pyplot as plt
      import seaborn as sns
      warnings.filterwarnings("ignore")
      # Select numerical columns
      numerical_cols = df.select_dtypes(include="number").columns
      # Set up the figure and axes with a smaller size
      n = len(numerical_cols)
      cols = 3  # Number of columns in the grid
      rows = (n // cols) + (n % cols > 0)  # Calculate number of rows needed
      fig, axes = plt.subplots(rows, cols, figsize=(10, 3 * rows))  # Smaller figure size
      axes = axes.flatten()  # Flatten the axes array for easy indexing
      # Plot each boxplot
      for i, col in enumerate(numerical_cols):
          sns.boxplot(data=df, x=col, ax=axes[i], width=0.3)  # Adjust width for smaller plots
          axes[i].set_title(col)
      # Hide any unused subplots
      for j in range(i + 1, len(axes)):
          fig.delaxes(axes[j])
      plt.tight_layout()
      plt.show()
```



Handle missing values

```
[15]: #Missing value treatements using mean for continous value columns
      for i in ["Income","Credit Score","Loan Amount","Assets Value"]:
          df[i].fillna(df[i].mean(),inplace=True)
```

```
[16]: #Missing value treatements using mode for descrete value columns

      from sklearn.impute import SimpleImputer

      # Create an imputer object with most_frequent strategy
      imputer = SimpleImputer(strategy='most_frequent')

      # Fit and transform the data
      df['Number of Dependents'] = imputer.fit_transform(df[['Number of Dependents']])
      df['Previous Defaults'] = imputer.fit_transform(df[['Previous Defaults']])
```

```
[17]: df.isnull().sum()
```

```
[17]: Age                      0
      Gender                   0
      Education Level          0
      Marital Status           0
      Income                   0
      Credit Score             0
      Loan Amount              0
      Loan Purpose             0
      Employment Status        0
      Years at Current Job     0
      Payment History          0
      Debt-to-Income Ratio     0
      Assets Value             0
      Number of Dependents     0
      City                     0
      State                    0
      Country                  0
      Previous Defaults        0
      Marital Status Change    0
      Risk Rating              0
      dtype: int64
```

Find correlation between Risk Rating column with other categorical columns

```
[20]:
      from scipy.stats import chi2_contingency
      def cramers_v(contingency_table):
          chi2, p, dof, expected = chi2_contingency(contingency_table)
          n = contingency_table.sum().sum()  # Total sample size
          r, k = contingency_table.shape
          cramers_v = np.sqrt(chi2 / (n * (min(r, k) - 1)))
          return cramers_v

      contingency_table = pd.crosstab(df['Gender'], df['Risk Rating'])
      cramers_v_value = cramers_v(contingency_table)
      print(f"Cramér's V for Gender and Risk Rating: {cramers_v_value}")

      contingency_table = pd.crosstab(df['Education Level'], df['Risk Rating'])
      cramers_v_value = cramers_v(contingency_table)
      print(f"Cramér's V for Education Level and Risk Rating: {cramers_v_value}")

      contingency_table = pd.crosstab(df['Marital Status'], df['Risk Rating'])
      cramers_v_value = cramers_v(contingency_table)
      print(f"Cramér's V for Marital Status and Risk Rating: {cramers_v_value}")

      contingency_table = pd.crosstab(df['Loan Purpose'], df['Risk Rating'])
      cramers_v_value = cramers_v(contingency_table)
      print(f"Cramér's V for Loan Purpose and Risk Rating: {cramers_v_value}")
```

```
contingency_table = pd.crosstab(df['Employment Status'], df['Risk Rating'])
cramers_v_value = cramers_v(contingency_table)
print(f"Cramér's V for Employement Status and Risk Rating: {cramers_v_value}")

contingency_table = pd.crosstab(df['Payment History'], df['Risk Rating'])
cramers_v_value = cramers_v(contingency_table)
print(f"Cramér's V for Payment History and Risk Rating: {cramers_v_value}")


contingency_table = pd.crosstab(df['City'], df['Risk Rating'])
cramers_v_value = cramers_v(contingency_table)
print(f"Cramér's V for City and Risk Rating: {cramers_v_value}")

contingency_table = pd.crosstab(df['State'], df['Risk Rating'])
cramers_v_value = cramers_v(contingency_table)
print(f"Cramér's V for State and Risk Rating: {cramers_v_value}")

contingency_table = pd.crosstab(df['Country'], df['Risk Rating'])
cramers_v_value = cramers_v(contingency_table)
print(f"Cramér's V for Country and Risk Rating: {cramers_v_value}")
```

```
Cramér's V for Gender and Risk Rating: 0.024397047231932888
Cramér's V for Education Level and Risk Rating: 0.012680084204131905
Cramér's V for Marital Status and Risk Rating: 0.022375185950592878
Cramér's V for Loan Purpose and Risk Rating: 0.025668056146136632
Cramér's V for Employement Status and Risk Rating: 0.015433015305277563
Cramér's V for Payment History and Risk Rating: 0.019797386112888005
Cramér's V for City and Risk Rating: 0.8802450179057865
Cramér's V for State and Risk Rating: 0.07595302023802004
Cramér's V for Country and Risk Rating: 0.15003672741293336
```

Drop columns

```
[24]: # Drop unnecessary categorical columns
df = df.drop(['State', 'Country', 'Education Level','Marital Status','Marital Status Change','Gender','Years at Current Job',

print(df.columns)
```

```
Index(['Age', 'Income', 'Loan Amount', 'Loan Purpose', 'Employment Status',
       'Payment History', 'Assets Value', 'City', 'Previous Defaults',
       'Risk Rating'],
      dtype='object')
```

For Ordinal Categorical Data Encoding using Label Encoding

```
[24]:
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

# defined order for ordinal columns
#education_order = ["High School", "Bachelor's", "Master's", 'PhD']
payment_history_order = ['Poor', 'Fair', 'Good', 'Excellent']
risk_rating_order = ['Low', 'Medium', 'High']

#df['Education Level'] = pd.Categorical(df['Education Level'], categories=education_order, ordered=True)
df['Payment History'] = pd.Categorical(df['Payment History'], categories=payment_history_order, ordered=True)
df['Risk Rating'] = pd.Categorical(df['Risk Rating'], categories=risk_rating_order, ordered=True)

#df['Education Level Encoded'] = label_encoder.fit_transform(df['Education Level'])
df['Payment History Encoded'] = label_encoder.fit_transform(df['Payment History'])
df['Risk Rating Encoded'] = label_encoder.fit_transform(df['Risk Rating'])
```

Group city by risk rating mean

```
[25]: # Group by City and calculate the mean Risk Rating for each city
      city_risk_means = df.groupby('City')['Risk Rating Encoded'].mean()

      # Assign groups based on risk level
      df['City_grouped'] = df['City'].apply(lambda x: 'High_Risk_Cities' if city_risk_means[x] > 0.5
                                                  else 'Low_Risk_Cities')


      print(df[['City', 'City_grouped']])
```

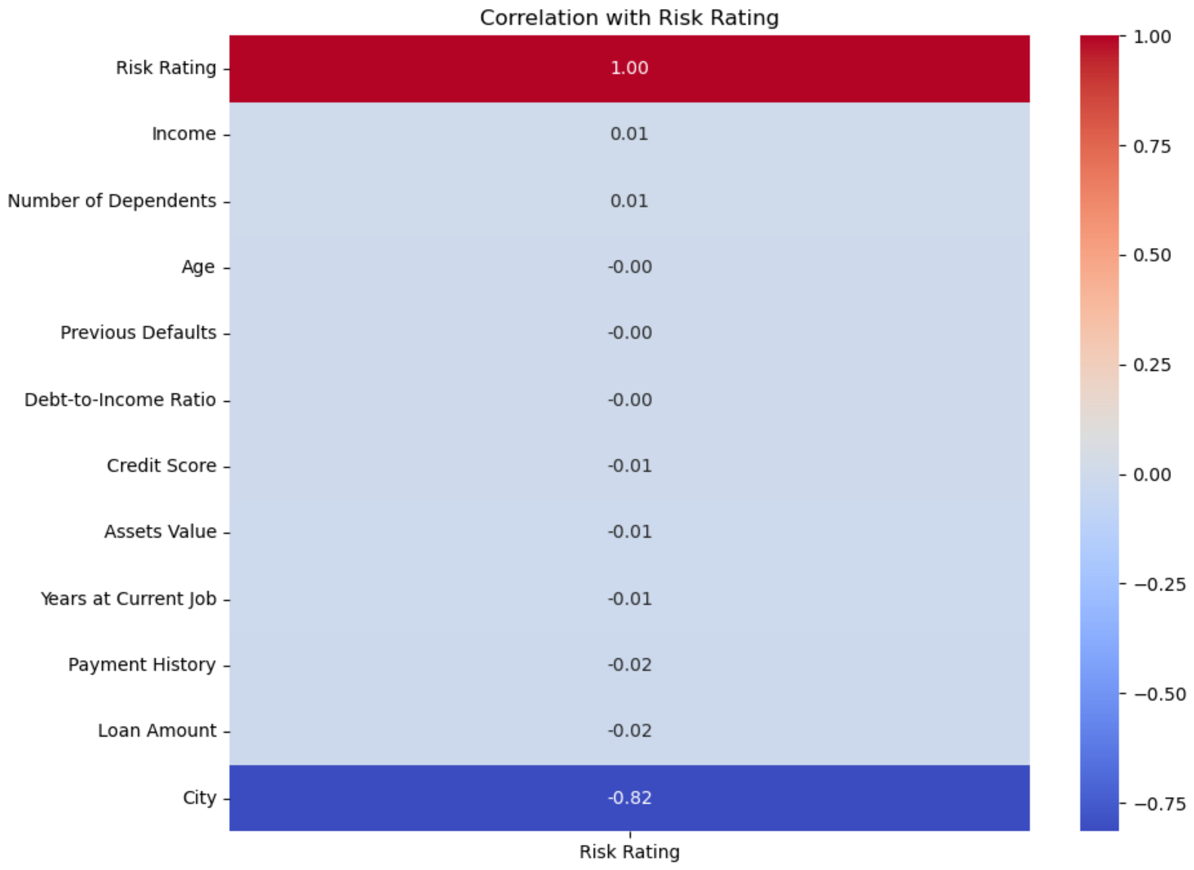
```
                      City      City_grouped
0           Port Elizabeth  High_Risk_Cities
1              New Heather  High_Risk_Cities
2           West Lindaview  High_Risk_Cities
3             Melissahaven  High_Risk_Cities
4            North Beverly  High_Risk_Cities
...                    ...               ...
10495          Curtismouth  High_Risk_Cities
10496            Susanstad  High_Risk_Cities
10497  South Morganchester   Low_Risk_Cities
10498            Port Wayne   Low_Risk_Cities
10499          South Stacy  High_Risk_Cities

[10500 rows x 2 columns]
```

```
[26]: df['City_grouped'].value_counts()
```

```
[26]: City_grouped
      High_Risk_Cities    8916
      Low_Risk_Cities     1584
      Name: count, dtype: int64
```

Correlation Matrix

```
[30]: # Select only numerical columns from the dataframe
      numerical_cols = df.select_dtypes(include=['int64','int32', 'float64'])

      # Calculate correlation matrix for numerical columns
      corr_matrix = numerical_cols.corr()

      # Check if 'Risk Rating' is an existing column
      if 'Risk Rating' in corr_matrix.columns:
          # Get the correlation between 'Risk Rating' and other numerical features
          risk_rating_correlation = corr_matrix['Risk Rating'].sort_values(ascending=False)

          # Visualize the correlation matrix as a heatmap using seaborn
          import seaborn as sns
          import matplotlib.pyplot as plt

          plt.figure(figsize=(10, 8))
          sns.heatmap(risk_rating_correlation.to_frame(), annot=True, cmap='coolwarm', fmt='.2f')
          plt.title('Correlation with Risk Rating')
          plt.show()
      else:
          print("'Risk Rating' column is not found in the correlation matrix.")
```



Nominal Categorical data encoding with One-Hot Encoding

```
[31]: # List of columns to One-Hot Encode
categorical_columns = [ 'Loan Purpose', 'Employment Status']

# Apply One-Hot Encoding to the specified categorical columns
df_encoded = pd.get_dummies(df, columns=categorical_columns, drop_first=False)

# Convert only the newly created one-hot encoded columns to int
one_hot_encoded_columns = df_encoded.columns.difference(df.columns)

# Apply astype(int) only to these new one-hot encoded columns
df_encoded[one_hot_encoded_columns] = df_encoded[one_hot_encoded_columns].astype(int)

# Replace original DataFrame with the encoded version
df = df_encoded.copy()


print("DataFrame after One-Hot Encoding with original columns replaced:")
print(df)
```

Scaling numerical features

[32]:
```
from sklearn.preprocessing import MinMaxScaler

#  Only scale certain numerical columns
numerical_columns_to_scale = ['Age', 'Income', 'Loan Amount', 'Credit Score','Assets Value','Debt-to-Income Ratio','Years at
df_numerical = df[numerical_columns_to_scale]

# Apply Min-Max Scaling
scaler = MinMaxScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df_numerical), columns=df_numerical.columns)

# Combine with the rest of the data that doesn't require scaling
df_rest = df.drop(columns=numerical_columns_to_scale)
df = pd.concat([df_rest, df_scaled], axis=1)

print("Final DataFrame after selective scaling:")
print(df.head())
```

[33]:
```
# Calculate the correlation matrix
correlation_matrix = df.corr()

# Get the correlation between 'Risk_Rating' and other features
risk_rating_correlation = correlation_matrix['Risk Rating'].sort_values(ascending=False)

# Display the correlation values
print(risk_rating_correlation)
```

```
Risk Rating                       1.000000
Loan Purpose_Business             0.022970
Employment Status_Unemployed      0.014693
Income                            0.006749
Number of Dependents              0.006014
Loan Purpose_Home                 0.001526
Age                              -0.002167
Previous Defaults                -0.002766
Debt-to-Income Ratio             -0.003034
Employment Status_Self-employed  -0.003292
Credit Score                     -0.005079
Loan Purpose_Personal            -0.007322
Assets Value                     -0.007482
Employment Status_Employed       -0.011398
Years at Current Job             -0.012975
Payment History                  -0.015373
Loan Amount                      -0.015577
Loan Purpose_Auto                -0.017108
City                             -0.815725
Name: Risk Rating, dtype: float64
```

Split the data

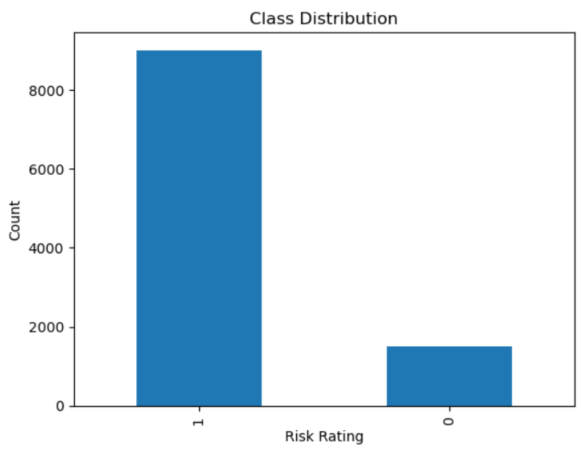
[36]:
```
from sklearn.model_selection import train_test_split

# Splitting the data into features (X) and target (y)
X = df.drop(columns='Risk Rating')
y = df['Risk Rating']

# 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)
```

[37]:
```
print(f"Training samples: {X_train.shape}")
print(f"Validation samples: {X_test.shape}")
print(f"Training samples: {y_train.shape}")
print(f"Validation samples: {y_test.shape}")
```

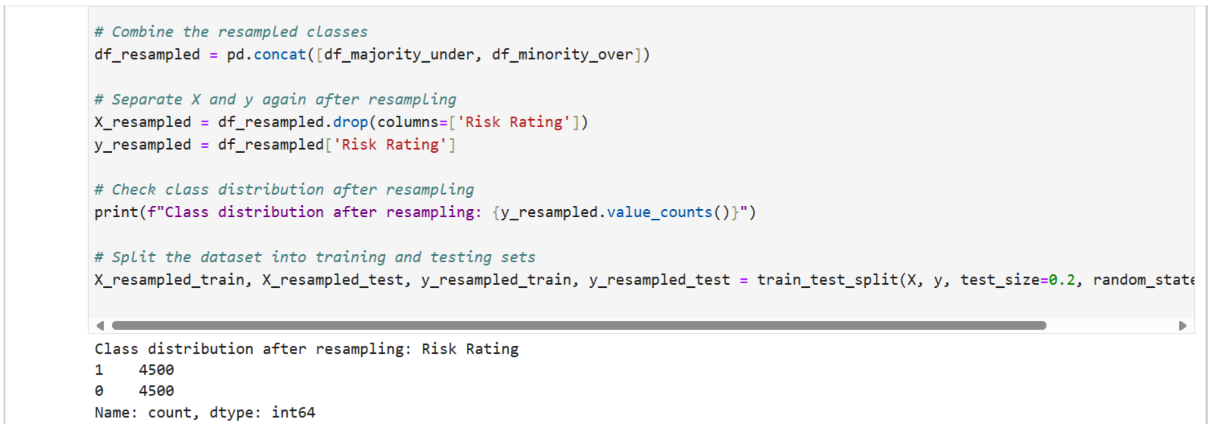
```
Training samples: (8400, 18)
Validation samples: (2100, 18)
Training samples: (8400,)
Validation samples: (2100,)
```

Handling imbalanced dataset

[34]:
```python
# Check class column distribution
import matplotlib.pyplot as plt

# Assuming 'df' is your DataFrame and 'Risk Rating' is your target variable
risk_rating_counts = df['Risk Rating'].value_counts()

# Plot the class distribution as a bar chart
risk_rating_counts.plot(kind='bar')
plt.title("Class Distribution")
plt.xlabel("Risk Rating")
plt.ylabel("Count")
plt.show()
```

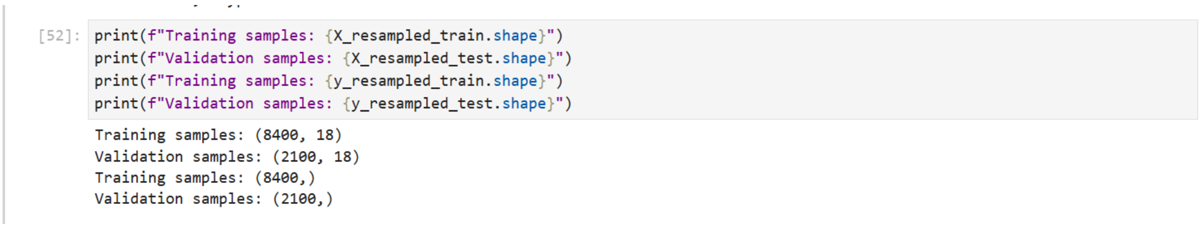


[51]:
```python
from sklearn.utils import resample
from sklearn.model_selection import train_test_split

#'Risk Rating' is the target column
X = df.drop(columns=['Risk Rating'])  # Features
y = df['Risk Rating']  # Target

# Combine X and y into one DataFrame for easy manipulation
df_combined = pd.concat([X, y], axis=1)

# Find the value counts for each class in the target column
class_counts = y.value_counts()

# Identify the majority and minority classes
majority_class = class_counts.idxmax()
minority_class = class_counts.idxmin()

# Separate each class
df_majority = df_combined[df_combined['Risk Rating'] == majority_class]
df_minority = df_combined[df_combined['Risk Rating'] == minority_class]

# Undersample the majority class to 4500 records
df_majority_under = resample(df_majority, replace=False, n_samples=4500, random_state=42)

# Oversample the minority class to 4500 records if needed
df_minority_over = resample(df_minority, replace=True, n_samples=4500, random_state=42)
```



```
# Combine the resampled classes
df_resampled = pd.concat([df_majority_under, df_minority_over])

# Separate X and y again after resampling
X_resampled = df_resampled.drop(columns=['Risk Rating'])
y_resampled = df_resampled['Risk Rating']

# Check class distribution after resampling
print(f"Class distribution after resampling: {y_resampled.value_counts()}")

# Split the dataset into training and testing sets
X_resampled_train, X_resampled_test, y_resampled_train, y_resampled_test = train_test_split(X, y, test_size=0.2, random_state
```

```
Class distribution after resampling: Risk Rating
1    4500
0    4500
Name: count, dtype: int64
```



```
[52]: print(f"Training samples: {X_resampled_train.shape}")
print(f"Validation samples: {X_resampled_test.shape}")
print(f"Training samples: {y_resampled_train.shape}")
print(f"Validation samples: {y_resampled_test.shape}")
```

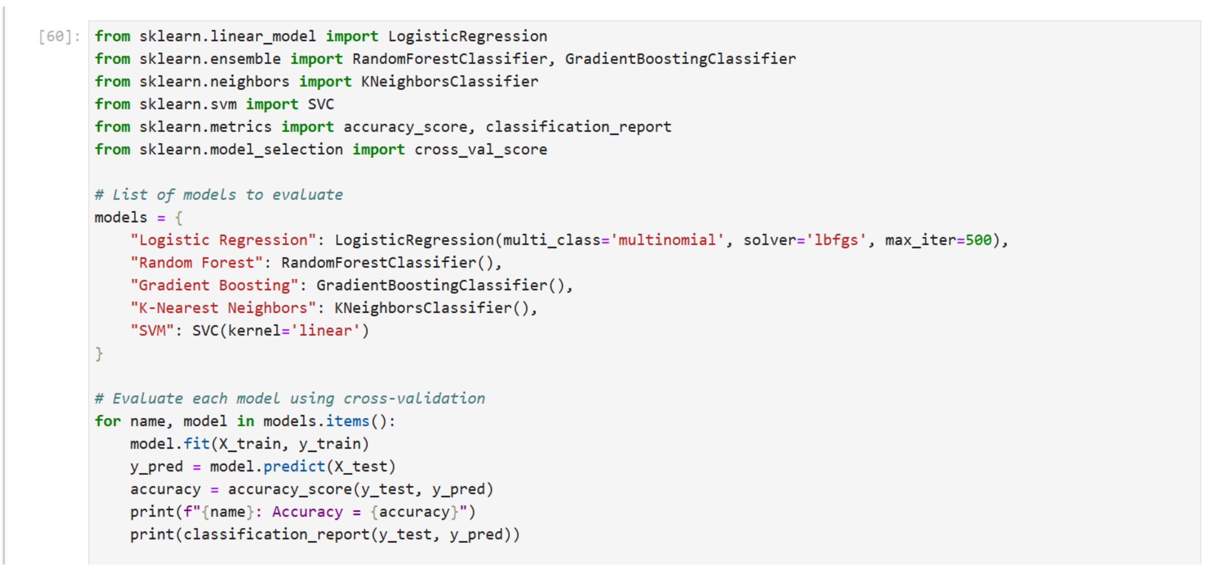
```
Training samples: (8400, 18)
Validation samples: (2100, 18)
Training samples: (8400,)
Validation samples: (2100,)
```

Model selection

Model training is the process of teaching a machine learning model to identify patterns and make predictions using data. This involves feeding the model a large dataset consisting of input examples and their corresponding outcomes. Through this data, the model learns to recognize the relationships between the inputs and the target outputs by making iterative adjustments to its internal parameters, allowing it to improve its accuracy over time.

Our Model are,

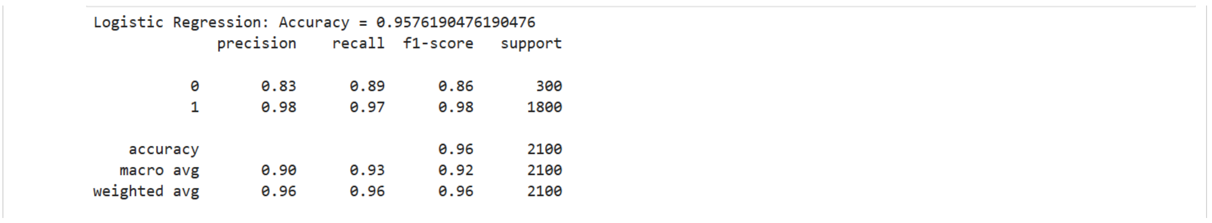
- Logistic Regression Model
- K - Neighbors Classifier
- Support Vector Machine
- Random Forest Classifier
- Gradient Boosting Classifier



```
[60]: from sklearn.linear_model import LogisticRegression
      from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
      from sklearn.neighbors import KNeighborsClassifier
      from sklearn.svm import SVC
      from sklearn.metrics import accuracy_score, classification_report
      from sklearn.model_selection import cross_val_score

      # List of models to evaluate
      models = {
          "Logistic Regression": LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=500),
          "Random Forest": RandomForestClassifier(),
          "Gradient Boosting": GradientBoostingClassifier(),
          "K-Nearest Neighbors": KNeighborsClassifier(),
          "SVM": SVC(kernel='linear')
      }

      # Evaluate each model using cross-validation
      for name, model in models.items():
          model.fit(X_train, y_train)
          y_pred = model.predict(X_test)
          accuracy = accuracy_score(y_test, y_pred)
          print(f"{name}: Accuracy = {accuracy}")
          print(classification_report(y_test, y_pred))
```

1.Logistic Regression Model

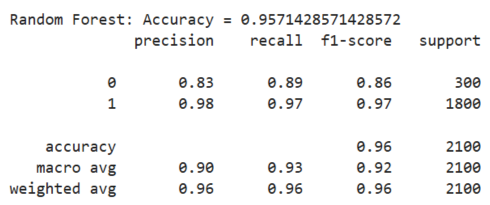
Linear classification; mostly applied to binary classification problems. It models probabilities with a logistic function and predicts class labels by selecting probabilities that are maximum. In the code, the algorithm has been set up for binary-class classification.



```
Logistic Regression: Accuracy = 0.9576190476190476
              precision    recall  f1-score   support

           0       0.83      0.89      0.86       300
           1       0.98      0.97      0.98      1800

    accuracy                           0.96      2100
   macro avg       0.90      0.93      0.92      2100
weighted avg       0.96      0.96      0.96      2100
```

2. Random Forest Classifier

The ensemble learning method works on the construction of multiple decision trees and then combines the results for a better classification outcome. It helps in reducing overfitting and improves generalization.



```
Random Forest: Accuracy = 0.9571428571428572
              precision    recall  f1-score   support

           0       0.83      0.89      0.86       300
           1       0.98      0.97      0.97      1800

    accuracy                           0.96      2100
   macro avg       0.90      0.93      0.92      2100
weighted avg       0.96      0.96      0.96      2100
```

3. Gradient Boosting Classifier

Another ensemble technique in which models are built in sequence, with each correcting the errors of the previously built model. It uses boosting for improving performance from weaker models, hence powerful for complicated datasets.

```
Gradient Boosting: Accuracy = 0.9566666666666667
              precision    recall  f1-score   support

           0       0.82      0.89      0.85       300
           1       0.98      0.97      0.97      1800

    accuracy                           0.96      2100
   macro avg       0.90      0.93      0.91      2100
weighted avg       0.96      0.96      0.96      2100
```

4. K-Nearest Neighbors Classifier

A simple instance-based learning algorithm that classifies an object based on the majority vote of its neighbors. It is easy to understand but computationally expensive for large data sets.

```
K-Nearest Neighbors: Accuracy = 0.9352380952380952
              precision    recall  f1-score   support

           0       0.82      0.71      0.76       300
           1       0.95      0.97      0.96      1800

    accuracy                           0.94      2100
   macro avg       0.88      0.84      0.86      2100
weighted avg       0.93      0.94      0.93      2100
```

5. Support Vector Machine (SVMs)

This is a classification algorithm that works by essentially finding the best hyperplane that separates the classes. The keyword kernel='linear' simply specifies a linear decision boundary, and is working best for data that are linearly separable.

```
SVM: Accuracy = 0.9576190476190476
              precision    recall  f1-score   support

           0       0.83      0.89      0.86       300
           1       0.98      0.97      0.98      1800

    accuracy                           0.96      2100
   macro avg       0.90      0.93      0.92      2100
weighted avg       0.96      0.96      0.96      2100
```

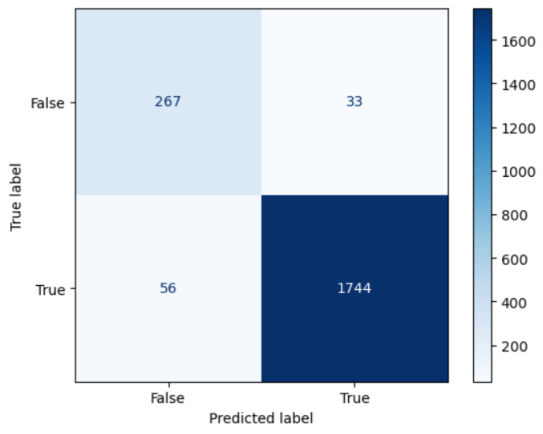
Model training

```
[65]: import joblib

      # Assuming RandomForest was the best-performing model
      best_model = RandomForestClassifier()
      best_model.fit(X_train, y_train)

      # Save the model to a file
      joblib.dump(best_model, 'best_model.pkl')
      print("Model saved successfully!")
```

```
Model saved successfully!
```

Confusion Matrix for Model Performance Evaluation

```
[39]: from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
      import matplotlib.pyplot as plt

      # Calculate the confusion matrix
      cm = confusion_matrix(y_test, y_pred)

      # Create a confusion matrix display with custom labels ('False', 'True')
      disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['False', 'True'])

      # Plot the confusion matrix with the 'Blues' color map
      disp.plot(cmap='Blues')
      plt.show()
```



Hyperparameter tuning

[67]:
```
from sklearn.model_selection import GridSearchCV

# Initialize RandomForestClassifier
rf = RandomForestClassifier(random_state=42)

# Define hyperparameters to tune
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Perform GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=2)

# Train the model
grid_search.fit(X_train, y_train)

# Get the best parameters
print(f"Best parameters: {grid_search.best_params_}")
```

```
# Evaluate on test data
y_pred = grid_search.predict(X_test)
print(f"Test Accuracy: {accuracy_score(y_test, y_pred)}")
print(classification_report(y_test, y_pred))
```

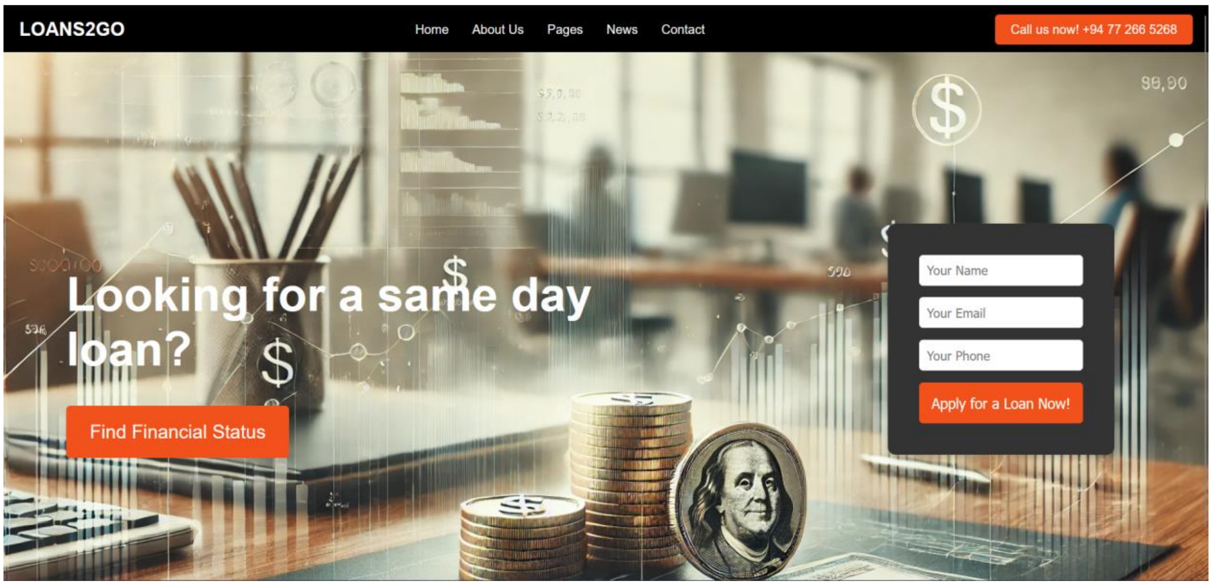
```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best parameters: {'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 100}
Test Accuracy: 0.9576190476190476
              precision    recall  f1-score   support

           0       0.83      0.89      0.86       300
           1       0.98      0.97      0.98      1800

    accuracy                           0.96      2100
   macro avg       0.90      0.93      0.92      2100
weighted avg       0.96      0.96      0.96      2100
```

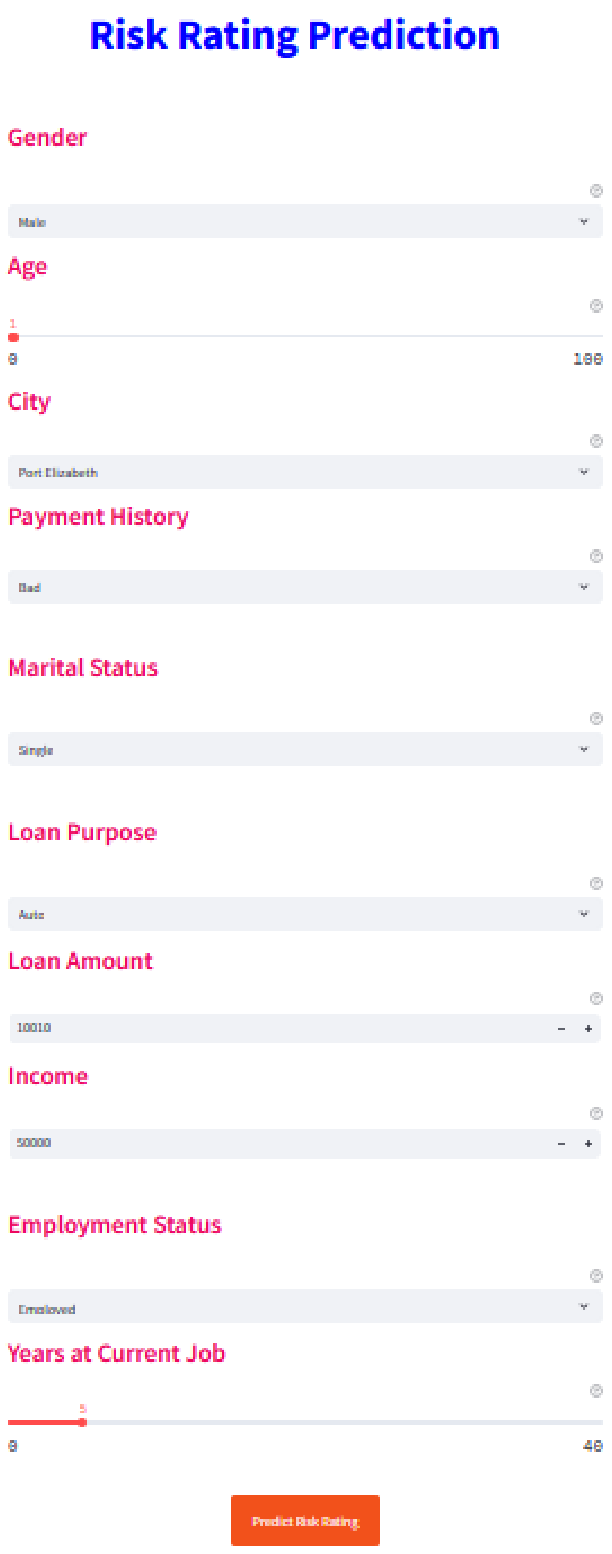
Cross-validation

[69]:
```
# Cross-validation on training data
cv_scores = cross_val_score(grid_search.best_estimator_, X_train, y_train, cv=5, scoring='accuracy')

print(f"Cross-validation accuracy: {cv_scores.mean()}")
```

```
Cross-validation accuracy: 0.9528571428571428
```

Frontend


LOANS2GO
Home
About Us
Pages
News
Contact
Call us now! +94 77 266 5268
Looking for a same day loan?
Find Financial Status
Your Name
Your Email
Your Phone
Apply for a Loan Now!



Risk Rating Prediction

Gender

Male

Age

1

0 100

City

Port Elizabeth

Payment History

Bad

Marital Status

Single

Loan Purpose

Auto

Loan Amount

10010

Income

50000

Employment Status

Employed

Years at Current Job

5

0 40

Conclusion

This project successfully proposes a machine learning-based solution for financial risk prediction that can enable financial institutions to make more appropriate assessments in the case of loan applications. Using demographic, financial, and behavioral data, we proposed the following challenges in the implementation: handling imbalanced classes via SMOTE, rectification of wrong and inconsistent data across columns, and removal of irrelevant features that would degrade the model's performance.

The final model optimized on cross-validation and hyperparameter tuning is therefore accurate in the prediction of risk levels such as Low or High. This system not only makes better decisions on loan approvals but also automates the process of risk evaluation involved, saving much manual effort and enhancing impartiality in assessments.

The work, in the near future, will be extended to real-time data for dynamic predictions; explain ability will also be implemented in AI models, and the dataset will be increased with more financial variables. This can help upscale and refine the system further to enhance its usefulness for the risk management of the financial sector.

Work Distribution

Student	Roles and Responsibilities
Bandara K M W G L A	• Data selection, preparation, preprocessing • Implementing and Training the model • Testing the model • Model Evaluation • Tuning and Optimizing the model • Model Deployment • Data visualizing • Implement user Interface
Jayasinghe J M H C	• Data selection, preparation, preprocessing • Implementing and Training the model • Testing the model • Model Evaluation • Tuning and Optimizing the model • Model Deployment • Data visualizing • Documentation
Abeysinghe E A	• Data selection, preparation, preprocessing • Implementing and Training the model • Testing the model • Model Evaluation • Tuning and Optimizing the model • Model Deployment • Data visualizing • Documentation
Wicramasinghe D A T N	• Data selection, preparation, preprocessing • Implementing and Training the model • Testing the model • Model Evaluation • Tuning and Optimizing the model • Model Deployment • Data visualizing • Documentation
Mesandu W M S	• Data selection, preparation, preprocessing • Implementing and Training the model • Testing the model • Model Evaluation • Tuning and Optimizing the model • Model Deployment • Data visualizing • Documentation