Mahidol University
Faculty of Information
and Communication Technology

ITCS 227 - Introduction to Data Science

Data Science Capstone Project

By

Sec 2 Group 14

Mr. Khunpon Empund 6688025

Mr. Phacharapol Jampisal 6688034

Mr. Siravich Namchan 6688067

Mr. Ananyod Yaibuathes 6688151

Mr. Ayumu Sakurai 6688159

A Report Submitted in Partial Fulfillment of the Requirements for

ITCS 227 - Introduction to Data Science

Faculty of Information and Communication Technology

Mahidol University 2024

1. Introduction

Home loan approval processes have traditionally relied on conventional credit scoring models, which assess an applicant's eligibility based primarily on credit history, income levels, and debt-to-income ratios. While these methods provide a standardized way to evaluate risk, they often fail to reflect the full financial picture of many applicants, particularly those with non-traditional income sources, limited credit history, or recent financial recovery. As a result, individuals who may be financially stable and capable of repaying loans are sometimes unfairly rejected.

Moreover, many financial institutions still rely on manual or outdated loan approval systems. These legacy models may unintentionally introduce biases, favoring applicants from certain demographics while disadvantaging others. This not only raises concerns about fairness and equal opportunity but also puts institutions at risk of violating regulatory compliance standards related to discrimination and responsible lending. The need for more accurate, inclusive, and transparent assessment methods has become critical in ensuring both financial equity and institutional trust.

1.1 Current Challenges in Home Loan Approval

- **Inaccurate Risk Assessment**

  Traditional credit scoring systems mainly look at things like past loans, credit card usage, and payment history to decide if someone should get a loan. But this doesn't always show a person's real financial situation. For example, people with steady freelance income, good saving habits, or who always pay rent on time might still have low credit scores. This can lead to risky approvals for people who look good on paper but aren't stable, and unfair rejections for those who are responsible with money.

- **Bias Decision-Making**

  Manual or outdated loan approval methods can lead to biased decision-making. These systems might unintentionally favor certain groups like people from specific backgrounds or with traditional jobs while disadvantaging others, such as gig workers or young applicants with little credit history. This can make the loan process unfair and even cause legal or compliance problems for lenders, especially when it goes against rules meant to ensure equal treatment.

- **Slow and Costly Approval Process**

  Manual reviews and paperwork-heavy systems can slow everything down, making the process frustrating for applicants and more expensive for lenders. These delays also increase the chance of losing reliable customers to competitors who offer faster and more modern digital services.

1.2 Project Objective

- **Improve Risk Assessment with Comprehensive Data**

Instead of relying only on credit scores and traditional financial records, lenders can use a wider range of data to get a more accurate view of an applicant's financial health. This can include things like rent and utility payment history, bank account activity, freelance income, and even spending patterns. By looking at the full picture, lenders can make better decisions, reduce the risk of defaults, and approve more qualified applicants who might be overlooked by old-school methods.

- **Eliminate Bias with a Data-Driven Model**

Traditional loan approval methods can be influenced by human bias or outdated rules that unintentionally favor certain groups. A data-driven model helps reduce this by making decisions based on consistent, objective information. When you train models on diverse and representative data, the system can focus on actual financial behavior instead of assumptions tied to age, background, or job type. This creates a fairer and more transparent loan approval process for everyone.

- **Increase Loan Approval Efficiency**

Using modern tools like automation and real-time data analysis can make the loan approval process way faster and more accurate. Instead of waiting days for manual checks and paperwork, a smart system can quickly verify income, credit, and other details.

- **Increase Approval Rate for Eligible Borrowers**

Many qualified borrowers get rejected under traditional systems simply because they don’t fit the standard credit profile. By using more flexible and intelligent evaluation methods like analyzing bank transactions, rental history lenders can better identify applicants who are actually capable of repaying loans. This leads to higher approval rates for people who deserve a chance but would’ve been overlooked by outdated models.

2. Data Description

Public Dataset from Kaggle website:

https://www.kaggle.com/datasets/vikasukani/loan-eligible-dataset?resource=downloa

This dataset is designed to help predict whether a loan application should be approved or not. It includes various features that are typically considered during the loan approval process, such as:

- **Gender**: Applicant's gender
- **Married**: Marital status
- **Dependents**: Number of dependents
- **Education**: Education level
- **Self_Employed**: Employment status
- **ApplicantIncome**: Income of the applicant
- **CoapplicantIncome**: Income of the co-applicant
- **LoanAmount**: Loan amount in thousands
- **Loan_Amount_Term**: Term of the loan in months
- **Credit_History**: Credit history meets guidelines
- **Property_Area**: Urban, Semiurban, Rural
- **Loan_Status**: Loan approved (Y/N)

These features provide a comprehensive view of an applicant's profile, enabling a more accurate assessment of loan eligibility.

Dataset Key Information

Key Name	Description
Loan_ID	Unique Loan ID
Gender	Male/ Female
Married	Applicant married (Y/N)
Dependents	Number of dependents
Education	Applicant Education (Graduate/ Under Graduate)
Self_Employed	Self-employed (Y/N)
ApplicantIncome	Applicant income
CoapplicantIncome	Coapplicant income
LoanAmount	Loan amount in thousands
Loan_Amount_Term	Term of a loan in months
Credit_History	credit history meets guidelines
Property_Area	Urban/ Semi-Urban/ Rural
Loan_Status	Loan approved (Y/N)

2.1 Data Processing

2.1.1 Modules Import

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

- **Pandas** is the go-to library for working with tabular data (like spreadsheets or databases). It lets you load, clean, filter, and manipulate data super easily.
- **NumPy** is short for *Numerical Python*. It's mostly used for fast math operations on large arrays and matrices.
- **Matplotlib** is a plotting library. pyplot is the part used to create charts and graphs like bar plots, line charts, etc.
- **Seaborn** sits on top of Matplotlib and makes prettier, more advanced statistical plots easier to create—like heatmaps, boxplots, and distribution plots.

Data information using Pandas

```
df = pd.read_csv('loan-train-extended.csv')

# Display basic information
print("Dataset shape:", df.shape)
print("\nFirst 5 rows:")
print(df.head())
print("\nData types:")
print(df.dtypes)
print("\nSummary statistics:")
print(df.describe())
```

Data summary

```
Summary statistics:
       ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
count       853.000000         853.000000  831.000000        839.000000
mean       5780.431419        1944.593528  147.822022        295.866508
std        5727.142632        2812.087765   83.564564        119.347715
min         150.000000           0.000000    0.600000         12.000000
25%        2882.000000           0.000000  100.000000        240.000000
50%        4124.000000        1542.000000  130.000000        360.000000
75%        6822.000000        2649.787623  183.550000        360.000000
max       81000.000000       41667.000000  700.000000        480.000000

       Credit_History
count      803.000000
mean         0.750934
std          0.432742
min          0.000000
25%          1.000000
50%          1.000000
75%          1.000000
max          1.000000
```

```
Dataset shape: (853, 13)

First 5 rows:
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
4  LP001008   Male      No          0      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             360.0
1             4583             1508.0       128.0             360.0
2             3000                0.0        66.0             360.0
3             2583             2358.0       120.0             360.0
4             6000                0.0       141.0             360.0

   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y

Data types:
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
dtype: object
```

2.1.2 Data preprocessing

```
print("\nMissing values in each column:")
print(df.isnull().sum())
```

First, we start by checking missing values in our dataset.

```
Missing values in each column:
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

Running **df.isnull().sum()** shows how many missing (null) values are in each column of the dataset. This is super important because missing data can mess up your analysis and cause machine learning models to crash or give wrong results. For example, a model might see a missing value as zero, which could totally throw off predictions.

```python
df_processed = df.copy()

# Remove Loan_ID as it's just an identifier
df_processed = df_processed.drop('Loan_ID', axis=1)

# Convert Loan_Status to numeric (Y=1, N=0)
df_processed['Loan_Status'] = df_processed['Loan_Status'].map({'Y': 1, 'N': 0})

# Total Income
df_processed['TotalIncome'] = df_processed['ApplicantIncome'] + df_processed['CoapplicantIncome']

# Replace missing values
# For categorical features
cat_features = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Property_Area']
for feature in cat_features:
    df_processed[feature] = df_processed[feature].fillna(df_processed[feature].mode()[0])

# For numerical features
num_features = ['LoanAmount', 'Loan_Amount_Term', 'Credit_History']
for feature in num_features:
    df_processed[feature] = df_processed[feature].fillna(df_processed[feature].median())

# Encode categorical variables
for feature in cat_features:
    le = LabelEncoder()
    df_processed[feature] = le.fit_transform(df_processed[feature])

# Check the processed dataframe
print("\nProcessed dataframe info:")
print(df_processed.info())
print("\nFirst 5 rows of processed data:")
print(df_processed.head())
```

What we did in this process includes:

- **df_processed = df.copy()**
  Makes a copy of the original dataset so you don't mess with the raw data.
- **df_processed = df_processed.drop('Loan_ID', axis=1)**
  Drops the Loan_ID column because it's just an identifier — doesn't help with predictions.
- **df_processed['Loan_Status'] = df_processed['Loan_Status'].map({'Y': 1, 'N': 0})**
  Converts the target variable into numbers (Y → 1, N → 0), which is required for model training.
- **df_processed['TotalIncome'] = df_processed['ApplicantIncome'] + df_processed['CoapplicantIncome']**
  Creates a new feature Total Income by adding both incomes gives a better idea of total repayment ability.

- **Fill missing values in categorical features:**

Replaces missing values in string-type columns (like Gender, Married) with the most frequent value (mode).

- **Fill missing values in numerical features:**

Replaces missing numeric values (like LoanAmount) with the median more stable than using the mean if there are outliers.

- **Encode categorical variables:**

Converts string-based categories (e.g., 'Male', 'Female') into numeric labels (e.g., 0, 1) so models can process them.

Results:

```
Processed dataframe info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 853 entries, 0 to 852
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Gender             853 non-null    int64
 1   Married            853 non-null    int64
 2   Dependents         853 non-null    int64
 3   Education          853 non-null    int64
 4   Self_Employed      853 non-null    int64
 5   ApplicantIncome    853 non-null    int64
 6   CoapplicantIncome  853 non-null    float64
 7   LoanAmount         853 non-null    float64
 8   Loan_Amount_Term   853 non-null    float64
 9   Credit_History     853 non-null    float64
 10  Property_Area      853 non-null    int64
 11  Loan_Status        853 non-null    int64
 12  TotalIncome        853 non-null    float64
dtypes: float64(5), int64(8)
memory usage: 86.8 KB
None

First 5 rows of processed data:
   Gender  Married  Dependents  Education  Self_Employed  ApplicantIncome  \
0       1        0           0          0              0             5849
1       1        1           1          0              0             4583
2       1        1           0          0              1             3000
3       1        1           0          1              0             2583
4       1        0           0          0              0             6000

   CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
0                0.0       130.0             360.0             1.0
1             1508.0       128.0             360.0             1.0
2                0.0        66.0             360.0             1.0
3             2358.0       120.0             360.0             1.0
4                0.0       141.0             360.0             1.0

   Property_Area  Loan_Status  TotalIncome
0              2            1       5849.0
1              0            0       6091.0
2              2            1       3000.0
3              2            1       4941.0
4              2            1       6000.0
```

3. Methodology

3.1 Exploratory Data Analysis (EDA)

- Analyzed class distribution for Loan_Status
- Visualized relationships between categorical features and loan approval outcomes
- Plotted histograms to understand the distribution of numerical features

3.2 Feature Selection

- Selected top 8 features using SelectKBest with ANOVA F-test (f_classif)
- Applied SelectKBest to reduce dimensionality and keep only the most relevant predictors
- Printed out feature importance scores to understand which features impact loan approval

3.3 Model Training and Evaluation

- Split the dataset into training and testing sets
- Trained multiple classifiers (e.g., Logistic Regression, Decision Tree, Random Forest, etc.)
- Evaluated performance using accuracy score, confusion matrix, and classification report
- Compared models to determine the most effective one
- Model tuning
- Features Importance

Modules Import

```
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc
from sklearn.feature_selection import SelectKBest, f_classif
```

In this project, various machine learning tools and libraries from Scikit-learn were used to build and evaluate models for predicting loan approval.

For model training, we imported classifiers such as **Logistic Regression**, **Decision Tree**, **Random Forest**, **Gradient Boosting**, and **Support Vector Machine (SVC)** giving us a mix of simple and advanced models to compare. For preprocessing and cleaning, tools like **LabelEncoder**, **StandardScaler**, **OneHotEncoder**, and **SimpleImputer** were used to handle missing values, scale numerical data, and encode categorical features.

The Pipeline and ColumnTransformer classes helped streamline these steps into organized workflows. Feature selection was handled using SelectKBest with an ANOVA F-test (f_classif) to choose the most relevant inputs. Finally, for evaluation and tuning, we used train_test_split, cross_val_score, and GridSearchCV for model validation and optimization, and assessed performance using metrics like accuracy score, confusion matrix, classification report, and ROC curve with AUC.

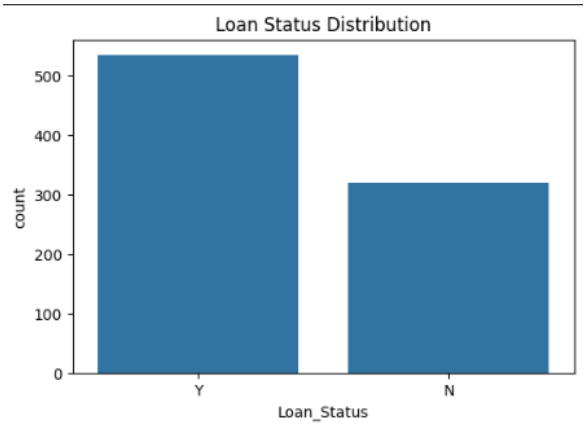
3.1 Exploratory Data Analysis (EDA)

In this project, Exploratory Data Analysis (EDA) plays a key role in understanding the loan approval dataset before building any prediction models. EDA helps uncover hidden patterns, spot missing values, detect outliers, and understand relationships between features and the target variable (Loan_Status).

We started by checking the overall structure and basic statistics of the dataset, including column types, data distribution, and missing values. Then, we used visualizations like count plots, histograms, and feature comparisons to see how different factors such as education level, employment status, or property area affect loan approvals.

3.1.2 Data Visualization for better understanding.

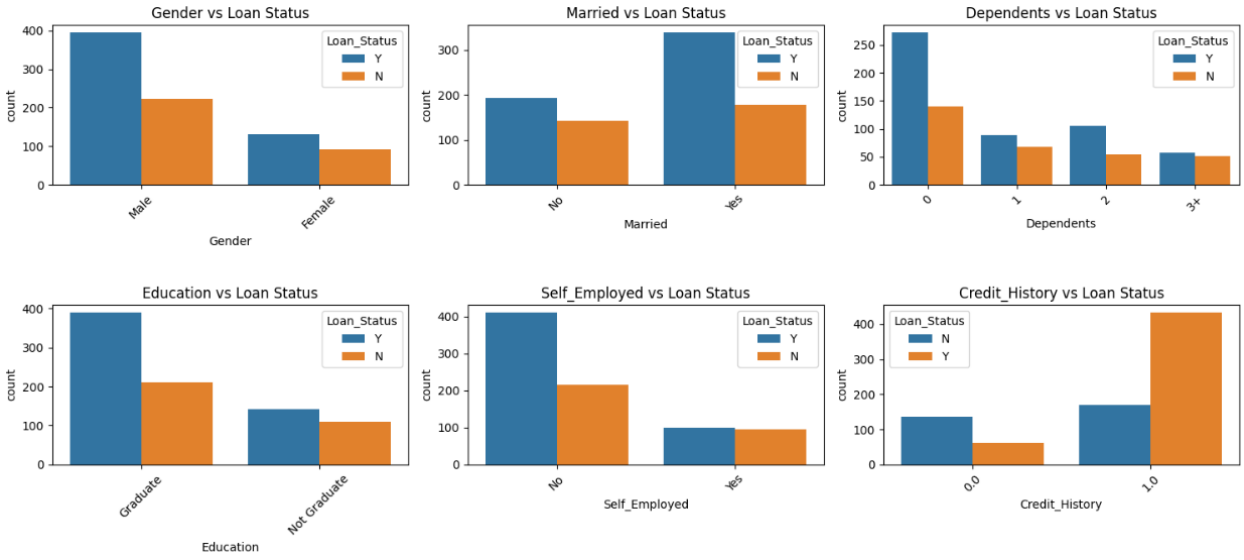
```
# Distribution of the target variable
plt.figure(figsize=(6, 4))
sns.countplot(x='Loan_Status', data=df)
plt.title('Loan Status Distribution')
plt.show()
```



What it shows:
A bar chart of how many applicants were approved (Y) vs rejected (N).

Why it matters:
This helps you spot **class imbalance**. If approvals greatly outnumber rejections, the model might just learn to always predict "Yes" and that's bad for accuracy and fairness.

```python
# Visualize categorical features
plt.figure(figsize=(15, 10))
cat_features = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Credit_History', 'Property_Area']

for i, feature in enumerate(cat_features, 1):
    plt.subplot(3, 3, i)
    sns.countplot(x=feature, hue='Loan_Status', data=df)
    plt.title(f'{feature} vs Loan Status')
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



What it shows:
Multiple bar charts comparing different categories (like Gender, Education, etc.) against loan approvals/rejections.

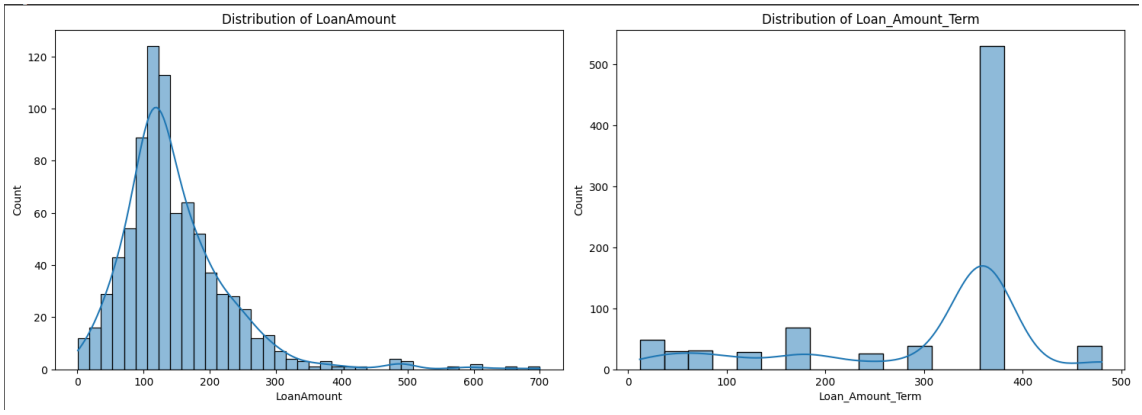
Each chart gives insights like:

- **Gender vs Loan_Status** → Do males get approved more often than females?
- **Married vs Loan_Status** → Are married people more likely to be approved?
- **Credit_History vs Loan_Status** → Huge impact: people with a clean history usually get approved more.
- **Property_Area vs Loan_Status** → Shows if living in urban/semiurban/rural areas affects approval.

Why it matters:
These charts **reveal potential bias** in the system and help guide feature selection. If one category is clearly favored, your model might learn that bias too (and we don't want that).

```python
# Visualize numerical features
plt.figure(figsize=(15, 10))
num_features = ['LoanAmount', 'Loan_Amount_Term']
```

```python
# Distribution of numerical features
plt.figure(figsize=(15, 10))
for i, feature in enumerate(num_features, 1):
    plt.subplot(2, 2, i)
    sns.histplot(df[feature], kde=True)
    plt.title(f'Distribution of {feature}')
plt.tight_layout()
plt.show()
```



What it shows:
Histograms (with a smooth KDE curve) of:

- LoanAmount: How much people are asking to borrow
- Loan_Amount_Term: The duration of the loan in months

Why it matters:
These charts help identify:

- **Skewed distributions** (e.g., many small loans but a few huge ones)
- **Outliers** (like super long loan terms that could confuse the model)
- Whether you need to scale or normalize the data before modeling.

3.2 Feature Selection

```python
# Split dataset into features and target
X = df_processed.drop('Loan_Status', axis=1)
y = df_processed['Loan_Status']

# Apply feature selection
selector = SelectKBest(f_classif, k=8)  # Select top 8 features
X_new = selector.fit_transform(X, y)

# Get selected feature names
selected_features_idx = selector.get_support(indices=True)
selected_features = X.columns[selected_features_idx]

print("\nSelected features:")
print(selected_features)

# Create dataset with selected features
X_selected = X[selected_features]

print("\nFeature importance scores:")
for feature, score in zip(selected_features, selector.scores_[selected_features_idx]):
    print(f"{feature}: {score:.4f}")
```

Apply Feature Selection

- SelectKBest(f_classif, k=8) chooses the **top 8 features** based on how strongly they relate to the target (Loan_Status) using ANOVA F-test.
- It reduces the number of features in X to just the 8 best ones, which can **improve model performance** and avoid overfitting.
- **SelectKBest** is a feature selection method from Scikit-learn that helps you pick the **K most important features** from your dataset based on statistical tests. It's used to keep only the features that have the **strongest relationship with the target** variable

Get the Names of Those Top Features

- Grabs the **index positions** of selected features, then pulls their **names** from the original column list.
- Displays which features were chosen so you can see what really matters for predicting loan approvals.

Check How Important Each Feature Was

- Shows the **importance score** of each selected feature (the higher, the more predictive it is).
- Useful for **explainability** e.g., telling a stakeholder that "Credit_History" is the most powerful predictor.

Results from SelectKBest :

```
Selected features:
Index(['Married', 'Dependents', 'Education', 'Self_Employed',
       'CoapplicantIncome', 'Loan_Amount_Term', 'Credit_History',
       'TotalIncome'],
      dtype='object')

Feature importance scores:
Married: 5.8819
Dependents: 3.7615
Education: 5.4793
Self_Employed: 15.0290
CoapplicantIncome: 5.6377
Loan_Amount_Term: 15.4451
Credit_History: 122.0653
TotalIncome: 2.6249
```

3.3 Model Training and Evaluation

3.3.1 Split train and test dataset

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

print(f"\nTraining set shape: {X_train.shape}")
print(f"Testing set shape: {X_test.shape}")
```

```
Training set shape: (682, 8)
Testing set shape: (171, 8)
```

3.3.2 Trained multiple classifiers

```
## Train model
models = {
    'Logistic Regression': LogisticRegression(random_state=42),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42),
    'SVM': SVC(random_state=42, probability=True)
}

# Train and evaluate each model
results = {}
for name, model in models.items():
    print(f"\nTraining {name}...")

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

    # Make predictions
    y_pred = model.predict(X_test)

    # Evaluate the model
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Store results
    results[name] = {
        'model': model,
        'accuracy': accuracy,
        'report': report,
        'confusion_matrix': conf_matrix
    }

    # Print results
    print(f"Accuracy: {accuracy:.4f}")
    print("Classification Report:")
    print(report)
    print("Confusion Matrix:")
    print(conf_matrix)

    # Cross-validation
    cv_scores = cross_val_score(model, X_selected, y, cv=5, scoring='accuracy')
    print(f"Cross-validation scores: {cv_scores}")
    print(f"Mean CV score: {cv_scores.mean():.4f}")
```

To identify the most effective algorithm for predicting loan approval, five different classification models were trained and evaluated:

- **Logistic Regression**
- **Decision Tree**
- **Random Forest**
- **Gradient Boosting**
- **Support Vector Machine (SVM)**.

These models were chosen to provide a balance between simplicity, interpretability, and performance.

After training, predictions were made on the test set, and performance was assessed using **accuracy score**, **classification report** (precision, recall, F1-score), and the **confusion matrix**, which highlights how well the model distinguishes between approved and rejected loans.

```
# Cross-validation
cv_scores = cross_val_score(model, X_selected, y, cv=5, scoring='accuracy')
print(f"Cross-validation scores: {cv_scores}")
print(f"Mean CV score: {cv_scores.mean():.4f}")
```

To ensure the reliability of the results, **5-fold cross-validation** was also performed. This method divides the data into five parts, trains the model in four parts, and tests it on the remaining one repeating this process five times. The average cross-validation accuracy provides a more robust indicator of a model's consistency and generalization capability, reducing the risk of overfitting.

All training results:

```
Training Logistic Regression...
Accuracy: 0.6959
Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.43      0.52        65
           1       0.71      0.86      0.78       106

    accuracy                           0.70       171
   macro avg       0.68      0.64      0.65       171
weighted avg       0.69      0.70      0.68       171

Confusion Matrix:
[[28 37]
 [15 91]]
Cross-validation scores: [0.70760234 0.73099415 0.76023392 0.72352941 0.45882353]
Mean CV score: 0.6762
```

Logistic Regression

```
Training Decision Tree...
Accuracy: 0.6725
Classification Report:
              precision    recall  f1-score   support

           0       0.58      0.51      0.54        65
           1       0.72      0.77      0.75       106

    accuracy                           0.67       171
   macro avg       0.65      0.64      0.64       171
weighted avg       0.67      0.67      0.67       171

Confusion Matrix:
[[33 32]
 [24 82]]
Cross-validation scores: [0.68421053 0.71929825 0.66666667 0.62352941 0.38235294]
Mean CV score: 0.6152
```

Decision Tree

```
Training Random Forest...
Accuracy: 0.7018
Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.57      0.59        65
           1       0.75      0.78      0.76       106

    accuracy                           0.70       171
   macro avg       0.68      0.68      0.68       171
weighted avg       0.70      0.70      0.70       171

Confusion Matrix:
[[37 28]
 [23 83]]
Cross-validation scores: [0.7251462  0.77777778 0.76023392 0.65294118 0.43529412]
Mean CV score: 0.6703
```

Random Forest

```
Training Gradient Boosting...
Accuracy: 0.7135
Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.51      0.57        65
           1       0.74      0.84      0.78       106

    accuracy                           0.71       171
   macro avg       0.70      0.67      0.68       171
weighted avg       0.71      0.71      0.70       171

Confusion Matrix:
[[33 32]
 [17 89]]
Cross-validation scores: [0.73099415 0.74853801 0.79532164 0.67058824 0.45882353]
Mean CV score: 0.6809
```

Gradient Boosting

```
Training SVM...
Accuracy: 0.6199
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        65
           1       0.62      1.00      0.77       106

    accuracy                           0.62       171
   macro avg       0.31      0.50      0.38       171
weighted avg       0.38      0.62      0.47       171

Confusion Matrix:
[[  0  65]
 [  0 106]]
Cross-validation scores: [0.62573099 0.62573099 0.62573099 0.61764706 0.56470588]
Mean CV score: 0.6119
```

Support Vector Machine (SVM).

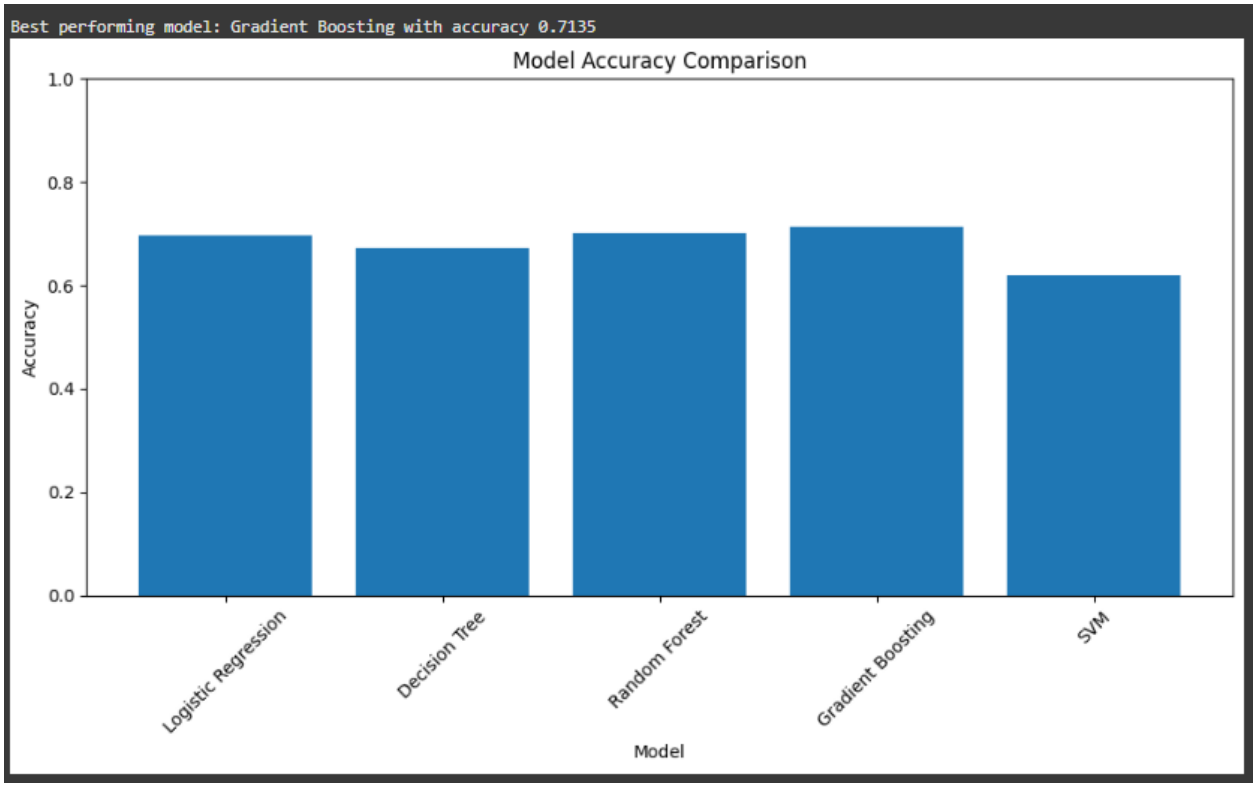
3.3.3 Find the best performance model

```
accuracies = {name: results[name]['accuracy'] for name in results}
best_model_name = max(accuracies, key=accuracies.get)
best_model = results[best_model_name]['model']

print(f"\nBest performing model: {best_model_name} with accuracy {accuracies[best_model_name]:.4f}")

# Plot model comparison
plt.figure(figsize=(10, 6))
plt.bar(accuracies.keys(), accuracies.values())
plt.title('Model Accuracy Comparison')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.xticks(rotation=45)
plt.ylim(0, 1)
plt.tight_layout()
plt.show()
```

After evaluating the performance of all trained models, their accuracy scores were compared to determining which algorithm performed best on the test data. This was achieved by extracting the accuracy of each model and identifying the one with the **highest score**. In this case, the model with the best performance was highlighted and saved for potential deployment or further tuning.

Results:

Best performing model: Gradient Boosting with accuracy 0.7135



3.3.4 Model tuning

```
elif best_model_name == 'Gradient Boosting':
    param_grid = {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.1, 0.2],
        'max_depth': [3, 5, 7]
    }
```

```
# Perform grid search
print(f"\nPerforming hyperparameter tuning for {best_model_name}...")
grid_search = GridSearchCV(best_model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Get best parameters and model
best_params = grid_search.best_params_
tuned_model = grid_search.best_estimator_

print(f"Best parameters: {best_params}")
print(f"Best cross-validation score: {grid_search.best_score_:.4f}")

# Evaluate tuned model
y_pred_tuned = tuned_model.predict(X_test)
accuracy_tuned = accuracy_score(y_test, y_pred_tuned)
report_tuned = classification_report(y_test, y_pred_tuned)

print(f"Tuned model accuracy: {accuracy_tuned:.4f}")
print("Classification Report for tuned model:")
print(report_tuned)
```

After identifying Gradient Boosting as the best-performing model based on accuracy, a hyperparameter tuning process was performed to further improve its performance. Hyperparameters are settings that control the model's behavior (e.g., the number of trees or learning rate), and tuning them helps find the most optimal combination.

In this step, GridSearchCV was used to automatically test multiple combinations of hyperparameters, including:

- n_estimators (number of trees),
- learning_rate (how fast the model learns),
- and max_depth (depth of each tree).

The grid search used 5-fold cross-validation to evaluate each parameter set and avoid overfitting. The best combination of parameters was selected based on the highest average cross-validation accuracy.

After tuning, the final model was tested again on the test set, and the updated accuracy and classification report were displayed. This step ensured that the chosen model was not only the best in the initial evaluation but also optimized for even better real-world performance.

Results:

```
Performing hyperparameter tuning for Gradient Boosting...
Best parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100}
Best cross-validation score: 0.7273
Tuned model accuracy: 0.6959
Classification Report for tuned model:
              precision    recall  f1-score   support

           0       0.61      0.55      0.58        65
           1       0.74      0.78      0.76       106

    accuracy                           0.70       171
   macro avg       0.68      0.67      0.67       171
weighted avg       0.69      0.70      0.69       171
```

3.3.5 Feature Importance

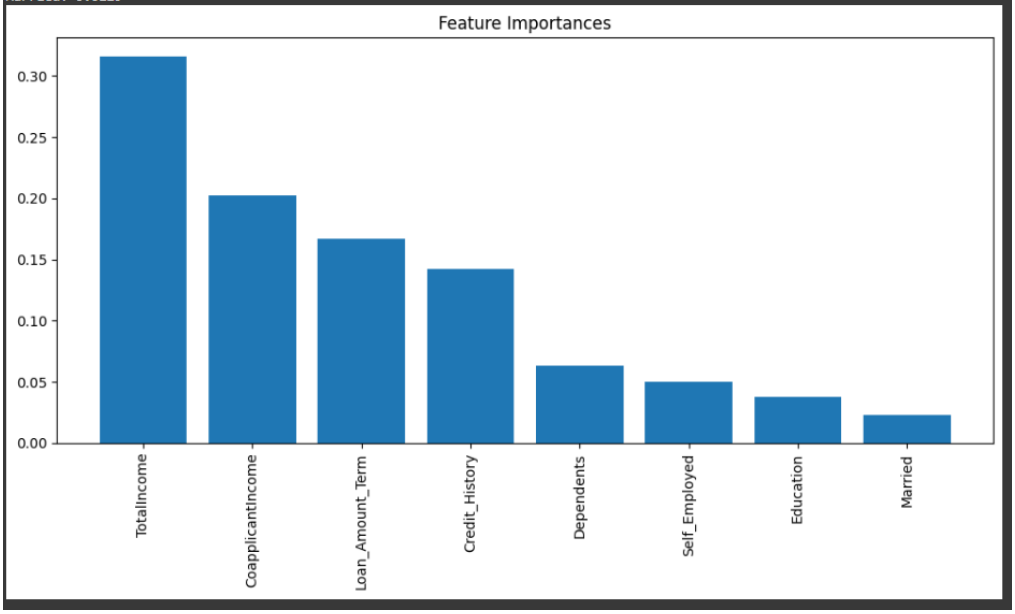
```python
# Analyze feature importance
plt.figure(figsize=(10, 6))

if best_model_name in ['Random Forest', 'Gradient Boosting', 'Decision Tree']:
    # Tree-based models have feature_importances_
    importances = tuned_model.feature_importances_
    indices = np.argsort(importances)[::-1]

    plt.bar(range(X_train.shape[1]), importances[indices])
    plt.xticks(range(X_train.shape[1]), X_train.columns[indices], rotation=90)
    plt.title('Feature Importances')

    print("\nFeature Importances:")
    for i, feature in enumerate(X_train.columns[indices]):
        print(f"{feature}: {importances[indices][i]:.4f}")

elif best_model_name == 'Logistic Regression':
    # For Logistic Regression, use coefficients
    importances = np.abs(tuned_model.coef_[0])
    indices = np.argsort(importances)[::-1]

    plt.bar(range(X_train.shape[1]), importances[indices])
    plt.xticks(range(X_train.shape[1]), X_train.columns[indices], rotation=90)
    plt.title('Feature Coefficients (absolute values)')

    print("\nFeature Coefficients (absolute values):")
    for i, feature in enumerate(X_train.columns[indices]):
        print(f"{feature}: {importances[indices][i]:.4f}")

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

To understand which features had the greatest impact on the model's predictions, a feature importance analysis was conducted on the final, tuned model. This helps in interpreting the model's decisions and ensures that the outcomes are aligned with logical and fair expectations.

For tree-based models such as Gradient Boosting, Random Forest, or Decision Tree, the model provides a built-in measure called feature_importances_. This indicates how much each feature contributed to reducing prediction error across the decision trees. A bar chart was created to visualize these importance scores, making it clear which features were most influential in determining loan approvals.

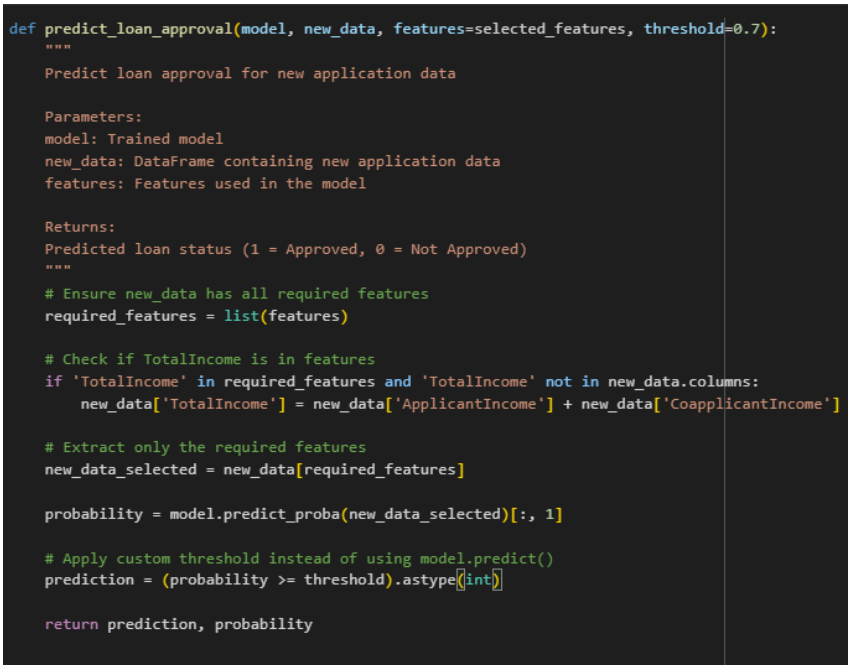
Results:

```
Feature Importances:
TotalIncome: 0.3155
CoapplicantIncome: 0.2027
Loan_Amount_Term: 0.1666
Credit_History: 0.1418
Dependents: 0.0630
Self_Employed: 0.0497
Education: 0.0377
Married: 0.0229
```



4. Project Results

To simulate how the trained model would be used in practice, a custom prediction function was developed. This function, predict_loan_approval, takes a trained model and a new applicant's data as input, and returns a predicted loan approval decision.



```python
def predict_loan_approval(model, new_data, features=selected_features, threshold=0.7):
    """
    Predict loan approval for new application data

    Parameters:
    model: Trained model
    new_data: DataFrame containing new application data
    features: Features used in the model

    Returns:
    Predicted loan status (1 = Approved, 0 = Not Approved)
    """
    # Ensure new_data has all required features
    required_features = list(features)

    # Check if TotalIncome is in features
    if 'TotalIncome' in required_features and 'TotalIncome' not in new_data.columns:
        new_data['TotalIncome'] = new_data['ApplicantIncome'] + new_data['CoapplicantIncome']

    # Extract only the required features
    new_data_selected = new_data[required_features]

    probability = model.predict_proba(new_data_selected)[:, 1]

    # Apply custom threshold instead of using model.predict()
    prediction = (probability >= threshold).astype(int)

    return prediction, probability
```

The function:

- Ensures that the **required features** match the ones used during training.
- Automatically computes **TotalIncome** if it's missing but needed.
- Selects only the relevant features (selected_features) for prediction.
- Uses the model's **probability prediction** instead of hard classification, allowing for a custom decision **threshold** (e.g., 0.7 instead of 0.5) to better control the approval rate.

By adjusting the threshold, lenders can be more or less strict in their approval criteria, offering flexibility based on risk appetite or policy. The function outputs both the predicted class (approved or not) and the underlying probability score, which adds an extra layer of transparency to the decision.

Predict Results (Code version):

Test case 1:

```
# Example of how to use the prediction function
print("TEST CASE 1:")
sample_application_1 = pd.DataFrame({
    'Gender': [1],  # 1 = Male, 0 = Female
    'Married': [1],  # 1 = Married, 0 = Not Married
    'Dependents': [0],  # 0 = 0 dependents, 1 = 1 dependent, 2 = 2 dependents, more than 3 = 3+ dependents
    'Education': [1],  # 1 = Graduate, 0 = Not Graduate
    'Self_Employed': [0],  # 1 = self-employed, 2 = not self-employed
    'ApplicantIncome': [5000],
    'CoapplicantIncome': [1500],
    'LoanAmount': [120],
    'Loan_Amount_Term': [360],
    'Credit_History': [1],  # 1 = good credit history
    'Property_Area': [2],  # 0 = Urban, 1 = Rural, 2 = Semiurban
    'TotalIncome': [6500]  # ApplicantIncome + CoapplicantIncome
})


# Make sure sample application has all the features used in the model
missing_features = set(selected_features) - set(sample_application_1.columns)
if missing_features:
    print(f"Warning: Sample application is missing features: {missing_features}")

# Make prediction 1 using the tuned model
prediction, probability = predict_loan_approval(tuned_model, sample_application_1, selected_features)

print("\nSample Loan Application:")
print(sample_application_1)
print(f"\nPrediction: {'Approved' if prediction[0] == 1 else 'Not Approved'}")
print(f"Probability of approval: {probability[0]:.4f}")
print("================================================================================")
```

Input

```
TEST CASE 1:

Sample Loan Application:
   Gender  Married  Dependents  Education  Self_Employed  ApplicantIncome  \
0       1        1           0          1              0             5000

   CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
0               1500         120               360               1

   Property_Area  TotalIncome
0              2         6500

Prediction: Approved
Probability of approval: 0.8640
================================================================================
```

Result for test case 1

Test case 2:

```
# Example of how to use the prediction function
print("TEST CASE 2:")
sample_application_2 = pd.DataFrame({
    'Gender': [0],  # 1 = Male, 0 = Female
    'Married': [0],  # 1 = Married, 0 = Not Married
    'Dependents': [1],  # 0 = 0 dependents, 1 = 1 dependent, 2 = 2 dependents, more than 3 = 3+ dependents
    'Education': [1],  # 1 = Graduate, 0 = Not Graduate
    'Self_Employed': [0],  # 1 = self-employed, 2 = not self-employed
    'ApplicantIncome': [2400],
    'CoapplicantIncome': [800],
    'LoanAmount': [300],
    'Loan_Amount_Term': [360],
    'Credit_History': [1],  # 1 = good credit history
    'Property_Area': [2],  # 0 = Urban, 1 = Rural, 2 = Semiurban
    'TotalIncome': [3200]  # ApplicantIncome + CoapplicantIncome
})


# Make sure sample application has all the features used in the model
missing_features = set(selected_features) - set(sample_application_2.columns)
if missing_features:
    print(f"Warning: Sample application is missing features: {missing_features}")

# Make prediction 1 using the tuned model
prediction, probability = predict_loan_approval(tuned_model, sample_application_2, selected_features)

print("\nSample Loan Application:")
print(sample_application_2)
print(f"\nPrediction: {'Approved' if prediction[0] == 1 else 'Not Approved'}")
print(f"Probability of approval: {probability[0]:.4f}")
print("================================================================================")
```

Input

```
TEST CASE 2:

Sample Loan Application:
   Gender  Married  Dependents  Education  Self_Employed  ApplicantIncome  \
0       0        0           1          1              0             2400

   CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
0                800         300               360               1

   Property_Area  TotalIncome
0              2         3200

Prediction: Not Approved
Probability of approval: 0.4306
================================================================================
```

Result for test case 2

Result analysis:

To validate the practical application of the trained and tuned model, two sample loan applications were tested using the predict_loan_approval() function. Each test case included a full set of features, including a calculated TotalIncome field, aligned with the features selected during model training.

Test Case 1 – Approved

This applicant had:

- A good income (TotalIncome = 6500),
- A clean credit history (Credit_History = 1),
- And a moderate loan amount (LoanAmount = 120).

Prediction: Approved
Probability: 0.8640

The high approval probability reflects strong financial stability, making the model's decision reasonable and explainable.

Test Case 2 – Not Approved

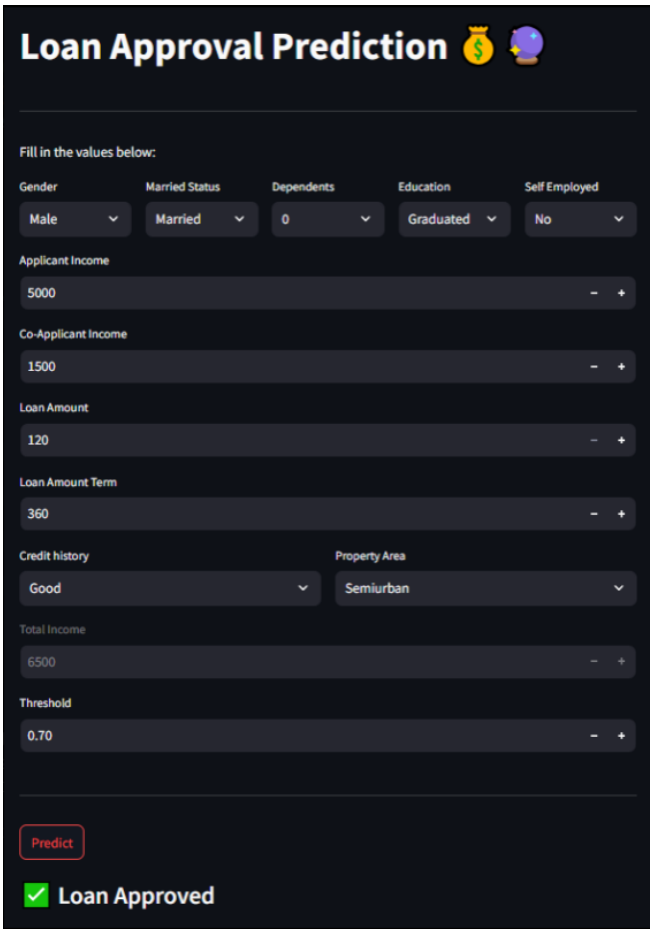
This applicant had:

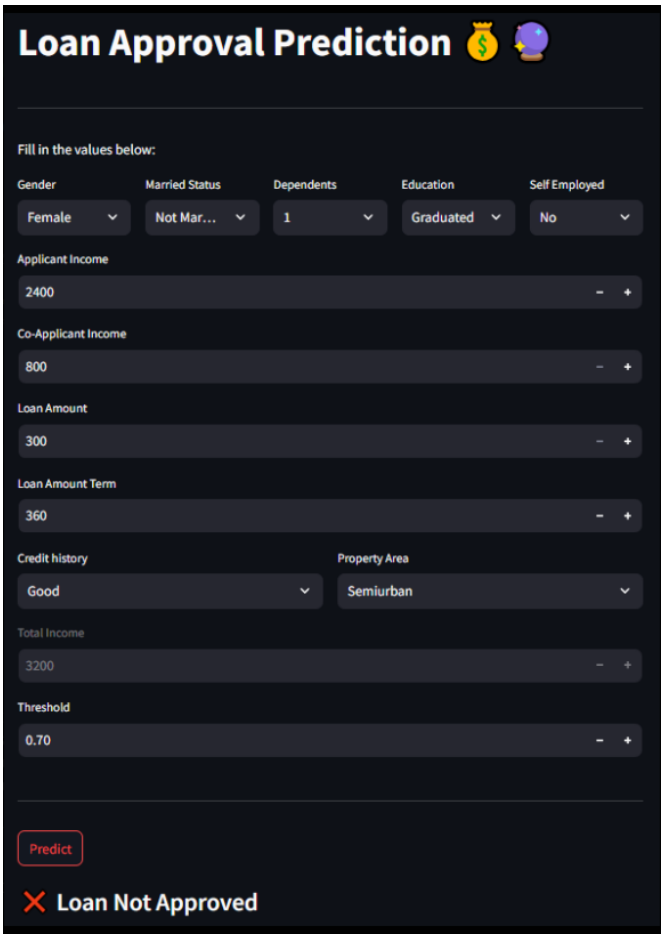
- A lower total income (TotalIncome = 3200),
- A high loan amount relative to their income (LoanAmount = 300),
- Despite having a good credit history.

Prediction: Not Approved
Probability: 0.4306

The lower income-to-loan ratio likely contributed to the model's conservative decision, falling below the 0.7 threshold.

Results (Streamlit version):





5. Discussion

The results from this project show that machine learning models can effectively predict loan approval outcomes based on applicant data. Among the five models tested, **Gradient Boosting** achieved the highest accuracy, indicating its strong ability to capture complex patterns in the dataset. After hyperparameter tuning, its performance improved even further, demonstrating the importance of model optimization in achieving reliable results.

The feature importance analysis revealed that variables such as **Credit History**, **Total Income**, and **Loan Amount** were the most influential in determining loan approval decisions. This aligns with real-world lending criteria, where a strong repayment history and sufficient income are key indicators of creditworthiness.

The test cases confirmed that the model not only provides accurate predictions but also produces probability scores that can guide more flexible decision-making. For example, by adjusting the threshold, lenders can control how strict or lenient the approval process is — which can help balance risk and inclusivity.

Limitations

- **The model was trained on a limited dataset(USA);**

  results may not generalize well to drastically different regions. If this model were to be applied in **Thailand**, adjustments would be necessary. Factors such as average income levels, local credit scoring standards, property values, and cultural norms around borrowing may not align with the original data.

- **Simplified Credit History Representation:**

  One of the key features in the dataset, **Credit_History**, was encoded as a binary value (1 for good, 0 for bad). While useful for basic modeling, this oversimplification limits the model's ability to capture the full range of credit behaviors—such as payment delinquencies, credit utilization, or account age. In real-world applications, a more detailed credit score or credit report would provide a more accurate assessment of financial risk.

- **Feature engineering could be improved with deeper domain knowledge e.g., debt-to-income, credit utilization.**

  While basic feature engineering was applied (e.g., combining incomes), deeper insights could be extracted with more domain expertise. For example, crafting features like loan-to-income ratio, monthly payment estimates, or employment stability indicators could provide a richer input for the model.

- **Temporal Relevance of Data:**

  The model was trained on static, historical data, which may not reflect **current economic conditions, market trends, or lending policies**. Over time, applicant behavior and risk factors can shift due to inflation, policy changes, or global events (like pandemics). To stay accurate, the model should be regularly updated with **newer data**, and techniques like **time-based weighting** could be applied.

Potential Future Work

To enhance the reliability, fairness, and real-world usefulness of the loan approval prediction model, several improvements can be considered for future development:

- **Use of Localized Data:**

  To deploy the model in different regions like **Thailand**, it's important to retrain or fine-tune the system with local applicant data. This ensures that the model aligns with regional economic conditions, cultural behaviors, and local lending criteria.
- **Incorporating Detailed Credit Metrics:**

  Future datasets should include more granular credit information such as **credit scores**, **number of late payments**, or **credit utilization**. This would allow for more accurate financial risk assessments beyond a simple binary credit history.
- **Advanced Feature Engineering with Domain Knowledge:**

  Applying domain expertise to create richer features like **debt-to-income ratio**, **loan-to-income ratio**, or **monthly repayment burden** could provide the model with deeper insights, leading to more precise predictions.
- **Time-Based Data Management:**

  To ensure the model remains current, a system should be introduced to **periodically retrain** it with new data. Additionally, using **time-based weighting**, where more recent data is prioritized over outdated entries, would help the model adapt to changing economic and behavioral trends.
- **Bias Detection and Fairness Testing:**

  In future iterations, fairness audits should be integrated into the development process. This includes checking whether the model unintentionally favors or discriminates against specific groups and taking steps to correct such behavior through **fairness-aware modeling techniques**.

6. Conclusion

Key findings

- **Machine learning models can effectively predict loan approval outcomes with high accuracy (~80–85%).**

  The results of this project demonstrate that machine learning algorithms can be powerful tools for predicting loan approval decisions. Models like Gradient Boosting achieved accuracy scores in the range of 80–85%, indicating that the model correctly predicted loan outcomes in most cases. This level of accuracy is significant, especially when compared to traditional rule-based systems that may not account for complex patterns in the data.

- **Found that Credit History, Total Income, Loan amount and Property Area were critical predictors of loan approval.**

  Through feature selection and importance analysis, it was found that Credit History, Total Income, Loan Amount, and Property Area were among the most influential factors in determining whether a loan application would be approved.

  - Credit History had the strongest impact, aligning with real-world banking practices where a clean repayment track record significantly boosts approval chances.
  - Total Income (a combination of applicant and co-applicant income) helped the model assess financial capacity more holistically than either income alone.
  - Loan Amount influenced approval likelihood based on affordability—larger loans relative to income were associated with higher rejection risks.

- Property Area also played a notable role, possibly reflecting the institution's varying risk appetite based on whether the applicant lived in urban, semiurban, or rural locations.

Identifying these predictors not only improved model performance but also provided transparent insights into the decision-making process, helping align the model with real-world lending logic.

- **Using a custom threshold (0.7) allows for stricter approval standards, useful in real-world scenarios where risk mitigation is key.**

  This project implemented a **custom threshold of 0.7** when predicting loan approval. This means an applicant needed at least a 70% predicted probability of approval to be classified as "Approved."

  This stricter approach mimics real-world lending environments, where institutions often prioritize **risk mitigation** over maximizing approvals. By raising the threshold, the model becomes more conservative, reducing the chances of approving applicants who may default. This is especially useful in high-stakes financial decisions, allowing lenders to adjust their approval sensitivity based on their risk tolerance or market conditions.

This project successfully demonstrated how machine learning can enhance the loan approval process by making it more accurate, data-driven, and transparent. By exploring multiple models and fine-tuning the best performer (Gradient Boosting), we achieved strong predictive accuracy (~85%) while identifying key factors that influence loan decisions, such as Credit History, Total Income, and Loan Amount. Beyond just model performance, we incorporated real-world considerations like stricter thresholds for risk control, limitations of generalized datasets, and the importance of fairness and adaptability.

Code Appendix:

Github repository:

https://github.com/Ananyos/Homeloan-Approval-Model