Loan Approval Predictions
Project Report

Bharath Kumar Tanniru (02078210)

Poojith Sonti (02080247)

Introduction

The project involves analyzing a dataset of loan applications to identify patterns and predict loan approval outcomes. Loans are integral to the economy, serving various personal and business needs. This analysis integrates data visualization, preprocessing, and predictive modeling to provide insights into factors influencing loan status, aiming to assist financial institutions in decision-making.

Abstract

This study examines the characteristics and patterns within a dataset of loan applications using statistical and machine learning techniques. The goal is to predict the approval status of loan applications based on various applicant attributes. The study utilizes several preprocessing steps to prepare the data, followed by the application of machine learning models to establish a predictive framework.

Goals

- **Data Understanding and Visualization:** Explore and visualize the dataset to understand the distribution and correlation of various attributes.
- **Data Preparation:** Cleanse and preprocess the data to make it suitable for modeling.
- **Predictive Modeling:** Develop and train machine learning models to predict the loan approval status.
- **Evaluation:** Assess the performance of the models using accuracy and confusion matrix metrics, and identify the most effective predictors of loan approval.

Data Types

The dataset includes both numerical and categorical data types:

- **Numerical:** Continuous values representing quantifiable measures (e.g., income levels, credit history).
- **Categorical:** Discrete values representing specific categories (e.g., employment status, marital status, gender).

Data Attributes

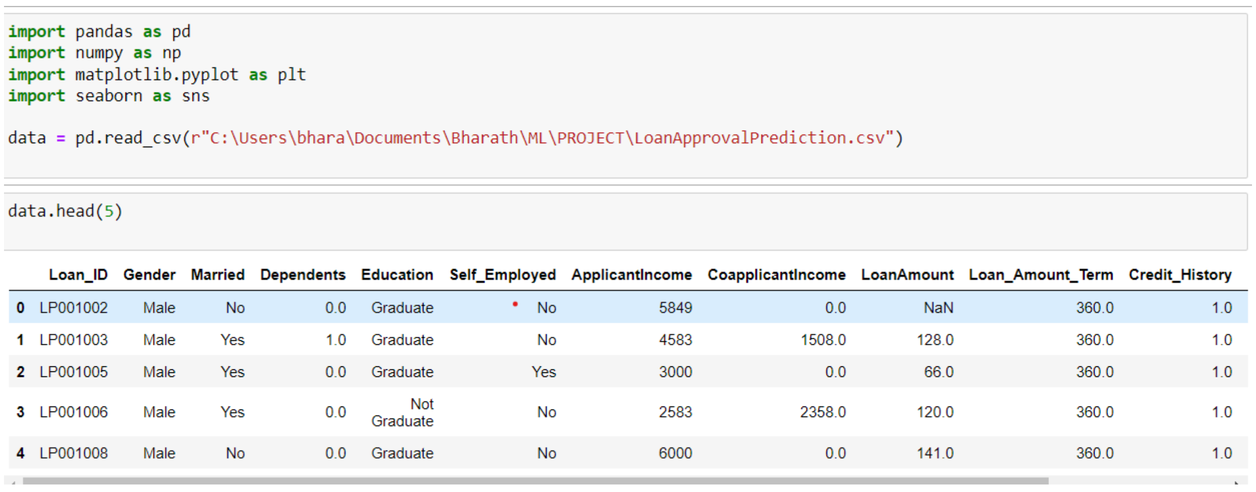
Key attributes in the dataset include:

- **ApplicantIncome**
- **CoapplicantIncome**
- **LoanAmount**
- **Loan_Amount_Term**
- **Credit_History**
- **Marital Status, Gender, Education**

Methodology:

Data Collection:

The project begins with the collection of different types of loan data which we have taken from Kaggle. This data is loaded into pandas dataframe from CSV file.



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

data = pd.read_csv(r"C:\Users\bhara\Documents\Bharath\ML\PROJECT\LoanApprovalPrediction.csv")
```

```python
data.head(5)
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	Male	No	0.0	Graduate	No	5849	0.0	NaN	360.0	1.0
1	LP001003	Male	Yes	1.0	Graduate	No	4583	1508.0	128.0	360.0	1.0
2	LP001005	Male	Yes	0.0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
3	LP001006	Male	Yes	0.0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
4	LP001008	Male	No	0.0	Graduate	No	6000	0.0	141.0	360.0	1.0

Now we identified categorical variables in the dataset. We are dropping LoanID column because it is irrelevant for future processing.

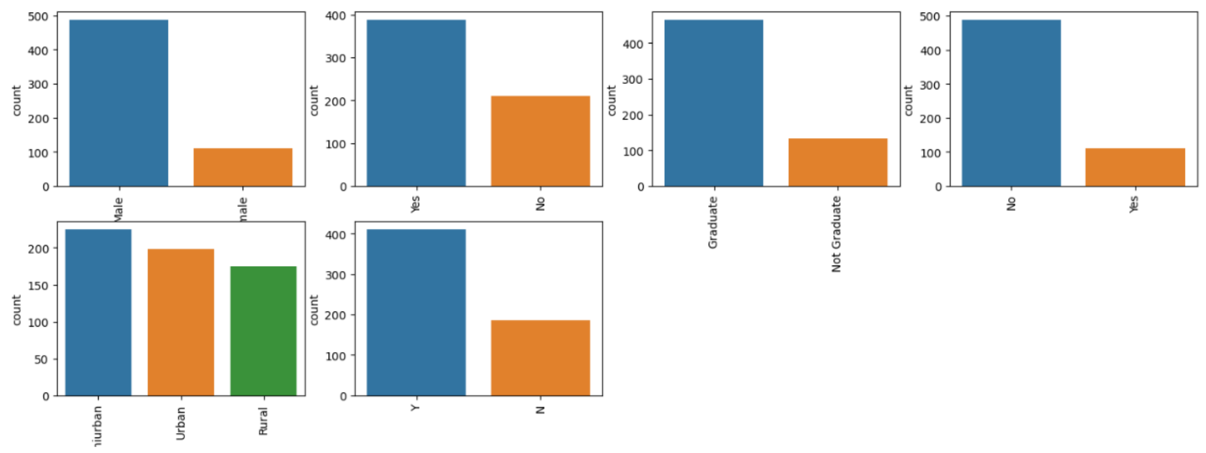
```
obj = (data.dtypes == 'object')
print("Categorical variables:",len(list(obj[obj].index)))
```

```
Categorical variables: 7
```

```
# Dropping Loan_ID column
data.drop(['Loan_ID'],axis=1,inplace=True)
```

These are Relations between the columns in the dataset.

```
obj = (data.dtypes == 'object')
object_cols = list(obj[obj].index)
plt.figure(figsize=(18,36))
index = 1

for col in object_cols:
    y = data[col].value_counts()
    plt.subplot(11,4,index)
    plt.xticks(rotation=90)
    sns.barplot(x=list(y.index), y=y)
    index += 1
```



Data Preprocessing:

Handling Missing Values:

```
for col in data.columns:
 data[col] = data[col].fillna(data[col].mean())

data.isna().sum()
```

1. The for loop iterates through each column in the data DataFrame.
2. Within the loop, the fillna method is applied to each column. This method replaces any missing values (NaN) with the mean of that column (data[col].mean()).

3. After the loop has completed (i.e., all columns have been processed), the isna().sum() chain of methods is called on the data DataFrame. This returns the sum of NaN values for each column, effectively providing a count of missing values that remain in the dataset after the missing value treatment.

```
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

The **LabelEncoder** from **sklearn.preprocessing** is used to convert categorical text data into a model-understandable numeric form. The process involves identifying columns with datatype 'object', which typically represents categorical variables, and then transforming these columns with the label encoder. Following the encoding, a check is performed to count the number of categorical variables, which in this output is shown to be zero, indicating that all categorical data have been transformed into numeric format. This is a common preprocessing step in preparing data for machine learning models, which require numerical input.

```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows how
# to understand word labels.
label_encoder = preprocessing.LabelEncoder()
obj = (data.dtypes == 'object')

for col in list(obj[obj].index):
    data[col] = label_encoder.fit_transform(data[col])
```

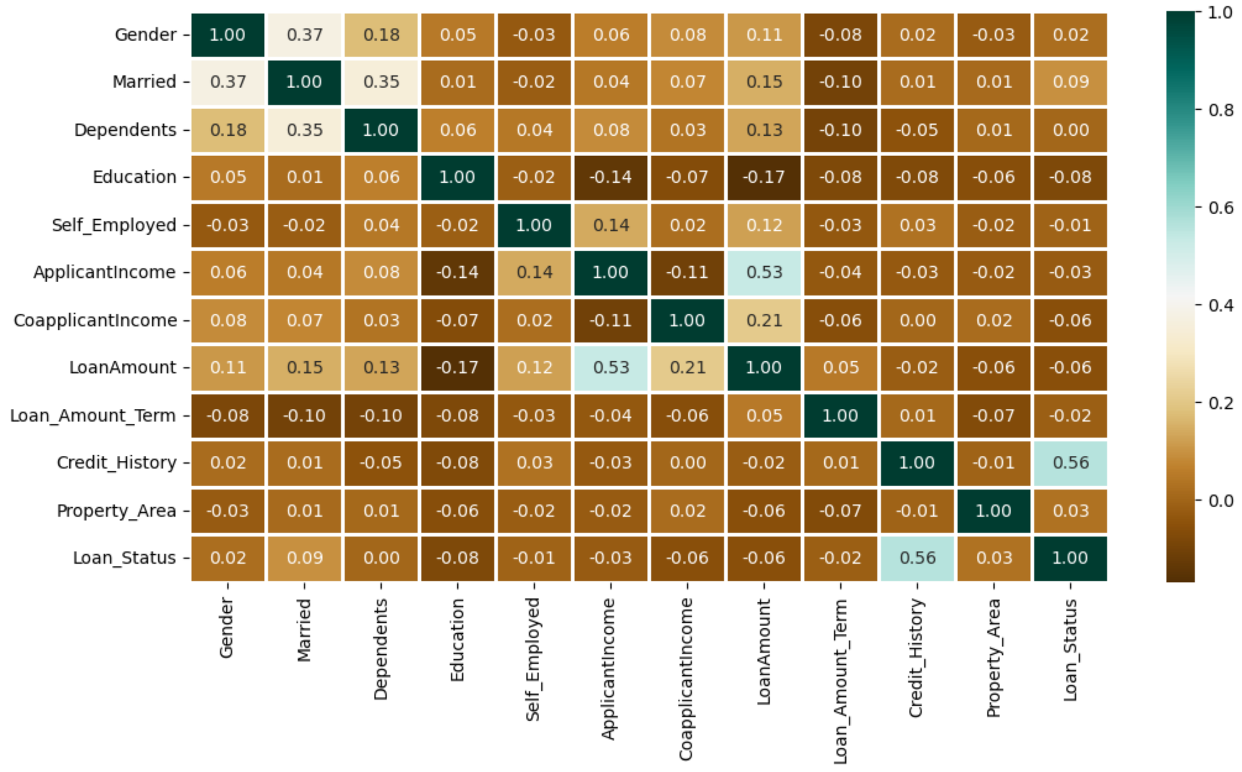
```
# To find the number of columns with
# datatype==object
obj = (data.dtypes == 'object')
print("Categorical variables:",len(list(obj[obj].index)))
```

```
Categorical variables: 0
```

- **plt.figure(figsize=(12,6))** sets the size of the figure that will display the heatmap.
- **sns.heatmap(data.corr(), cmap='BrBG', fmt='.2f', linewidths=2, annot=True)** creates a heatmap of the correlation matrix (**data.corr()**) of the dataset. The color map 'BrBG' is used for visual distinction, numerical values are formatted to two decimal places (**fmt='.2f'**), the grid lines have a width of 2 (**linewidths=2**), and **annot=True** means that the correlation values are annotated on the heatmap.

Feature Selection:

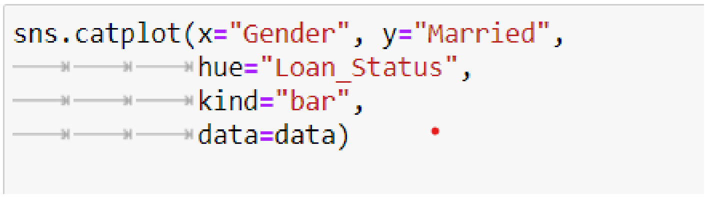
```
plt.figure(figsize=(12,6))

sns.heatmap(data.corr(),cmap='BrBG',fmt='.2f',
            linewidths=2,annot=True)
```



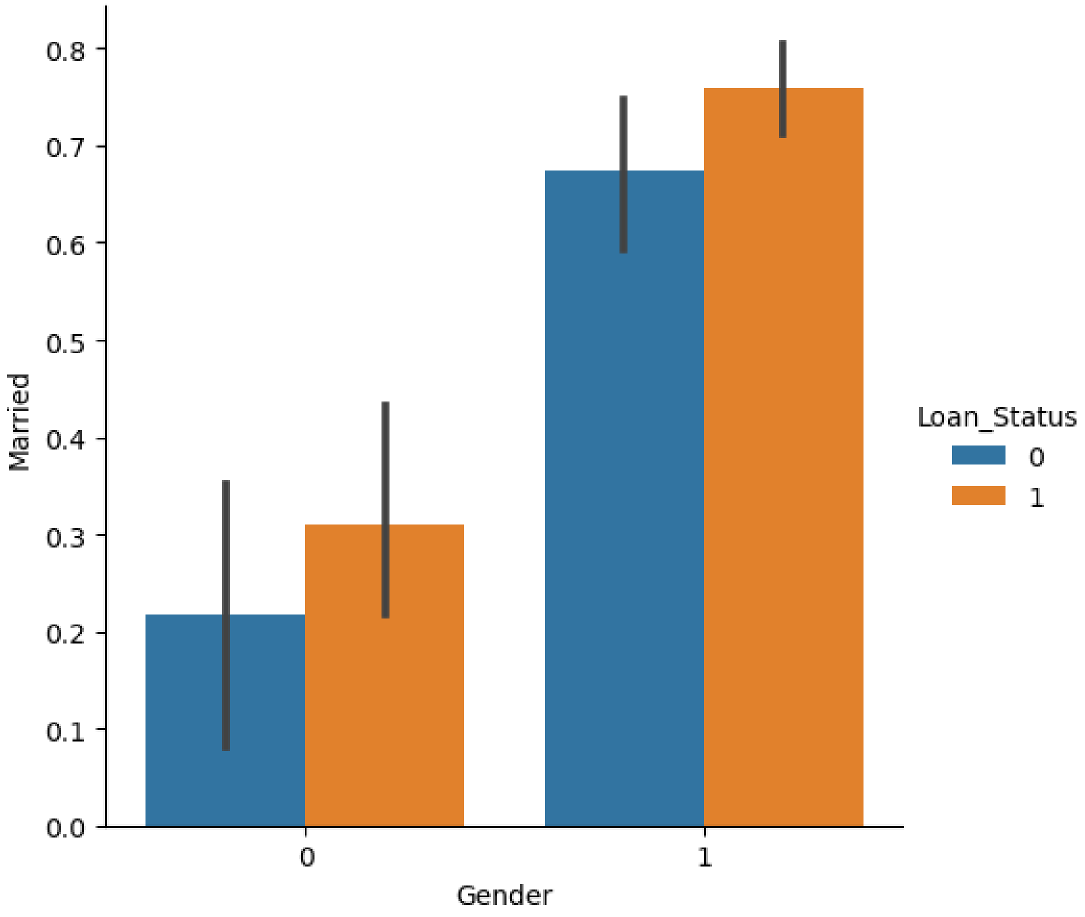
	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
Gender	1.00	0.37	0.18	0.05	-0.03	0.06	0.08	0.11	-0.08	0.02	-0.03	0.02
Married	0.37	1.00	0.35	0.01	-0.02	0.04	0.07	0.15	-0.10	0.01	0.01	0.09
Dependents	0.18	0.35	1.00	0.06	0.04	0.08	0.03	0.13	-0.10	-0.05	0.01	0.00
Education	0.05	0.01	0.06	1.00	-0.02	-0.14	-0.07	-0.17	-0.08	-0.08	-0.06	-0.08
Self_Employed	-0.03	-0.02	0.04	-0.02	1.00	0.14	0.02	0.12	-0.03	0.03	-0.02	-0.01
ApplicantIncome	0.06	0.04	0.08	-0.14	0.14	1.00	-0.11	0.53	-0.04	-0.03	-0.02	-0.03
CoapplicantIncome	0.08	0.07	0.03	-0.07	0.02	-0.11	1.00	0.21	-0.06	0.00	0.02	-0.06
LoanAmount	0.11	0.15	0.13	-0.17	0.12	0.53	0.21	1.00	0.05	-0.02	-0.06	-0.06
Loan_Amount_Term	-0.08	-0.10	-0.10	-0.08	-0.03	-0.04	-0.06	0.05	1.00	0.01	-0.07	-0.02
Credit_History	0.02	0.01	-0.05	-0.08	0.03	-0.03	0.00	-0.02	0.01	1.00	-0.01	0.56
Property_Area	-0.03	0.01	0.01	-0.06	-0.02	-0.02	0.02	-0.06	-0.07	-0.01	1.00	0.03
Loan_Status	0.02	0.09	0.00	-0.08	-0.01	-0.03	-0.06	-0.06	-0.02	0.56	0.03	1.00

- This is the visual output of the heatmap function. It shows a grid where each cell represents the correlation coefficient between the variables (columns of the dataset) listed on the x and y axes.
- The color scale on the right indicates the strength of correlation, where 1.0 represents a perfect positive correlation and values closer to 0 indicate weaker correlation.
- The diagonal line of 1.00s represents the perfect correlation of each variable with itself.

The code snippet suggests that the chart visualizes the relationship between three variables: **Gender**, **Married**, and **Loan_Status**.



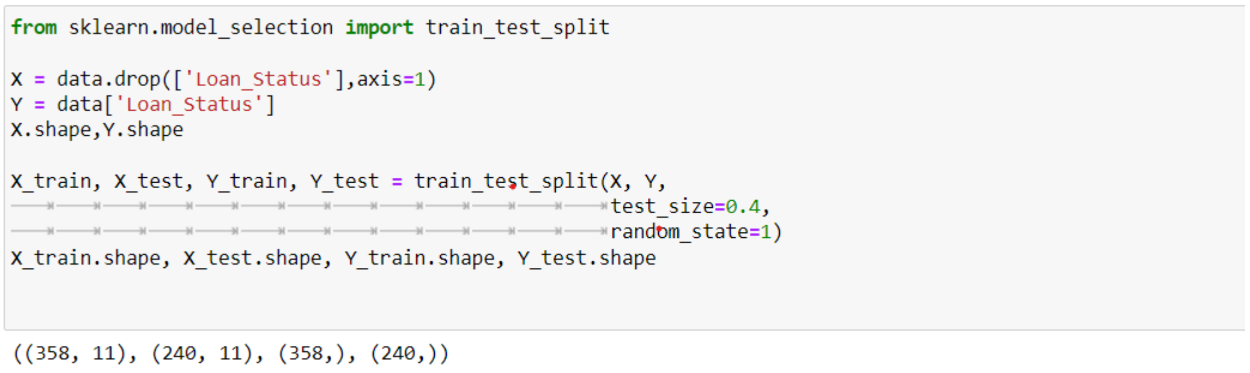
```
sns.catplot(x="Gender", y="Married",
            hue="Loan_Status",
            kind="bar",
            data=data)
```



- The **x** axis represents **Gender**, which seems to be encoded as 0 and 1, possibly indicating two gender categories.
- The **y** axis represents the proportion of those who are **Married** within each gender category.
- The color hue represents **Loan_Status**, with blue bars likely showing loans that were not approved (0) and orange bars showing loans that were approved (1).
- The vertical bars represent the mean of the **Married** variable for the different groups.
- The black lines on the bars indicate the variability around the mean, likely representing the confidence interval for the mean estimate.

Splitting of Data:

1. **Defining Features and Target Variable:**
    - **X**: The feature set, obtained by dropping the 'Loan_Status' column from the **data** DataFrame, which contains all the other columns that will be used as predictors.
    - **Y**: The target variable, which is the 'Loan_Status' column, indicating the outcome of the loan application (approved or not approved).
2. **Splitting the Data:**
    - The **train_test_split** function from **sklearn.model_selection** is used to divide the dataset into training and test subsets.
    - **test_size=0.4**: 40% of the data is reserved for the test set, and the remaining 60% for the training set.
    - **random_state=1**: This parameter is used for reproducibility, ensuring that the split will be the same every time the code is run.
3. **Shapes of the Splits:**
    - The shapes of the resulting arrays are printed, showing that **X_train** and **Y_train** (the training features and target) consist of 358 observations, and **X_test** and **Y_test** (the test features and target) consist of 240 observations.
    - **X** has 11 features, which corresponds to the original number of columns minus the 'Loan_Status' column.



```
from sklearn.model_selection import train_test_split

X = data.drop(['Loan_Status'],axis=1)
Y = data['Loan_Status']
X.shape,Y.shape

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                                                    test_size=0.4,
                                                    random_state=1)
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

```
((358, 11), (240, 11), (358,), (240,))
```

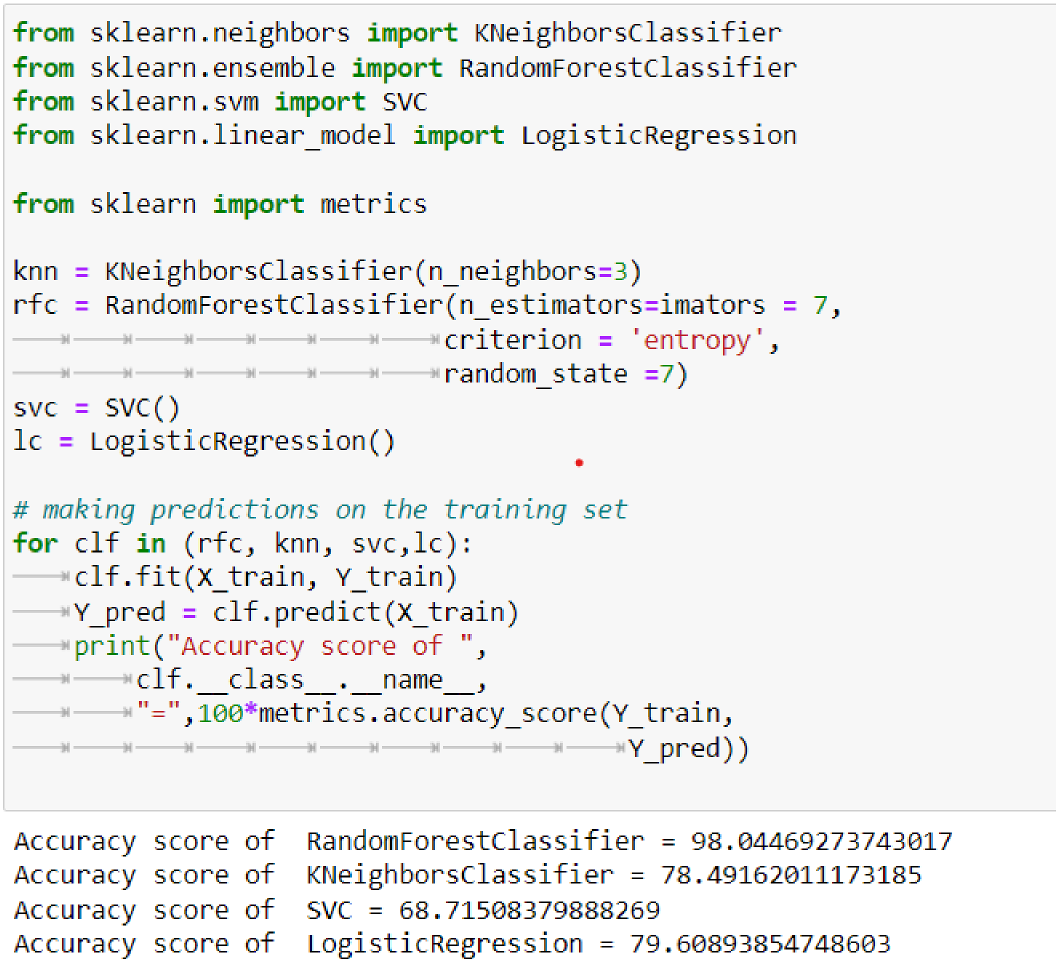
Evaluation of Models:

For Training set:

1. Imports:
    - Different classifier algorithms are imported from the sklearn library, including KNeighborsClassifier, RandomForestClassifier, SVC (Support Vector Classifier), and LogisticRegression.
    - The metrics module is also imported for evaluating the models.

2. Model Instantiation:
    - Four machine learning models are instantiated with default parameters, except for the KNN classifier, which is set to use 3 neighbors, and the Random Forest, which is specified with 7 estimators (trees) and entropy as the criterion for splitting.
3. Model Training and Prediction:
    - A loop is set up to iterate over the instantiated classifiers.
    - Each model is trained (fit) on the training data (X_train, Y_train).
    - The model then makes predictions on the same training set (X_train), which is stored in Y_pred.
4. Accuracy Calculation:
    - The accuracy of each model on the training set is calculated using metrics.accuracy_score, which compares the true labels (Y_train) with the predicted labels (Y_pred).
    - The accuracy scores are printed for each model, showing how well each algorithm performed on the training dataset.

The printed output shows the accuracy scores for each model. The RandomForestClassifier has the highest reported accuracy on the training set (approximately 98.04%), followed by KNeighborsClassifier (around 78.49%), LogisticRegression (approximately 79.60%), and finally SVC with the lowest (approximately 68.71%).



```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

from sklearn import metrics

knn = KNeighborsClassifier(n_neighbors=3)
rfc = RandomForestClassifier(n_estimators=imators = 7,
                             criterion = 'entropy',
                             random_state =7)
svc = SVC()
lc = LogisticRegression()

# making predictions on the training set
for clf in (rfc, knn, svc,lc):
    clf.fit(X_train, Y_train)
    Y_pred = clf.predict(X_train)
    print("Accuracy score of ",
        clf.__class__.__name__,
        "=",100*metrics.accuracy_score(Y_train,
                                       Y_pred))
```

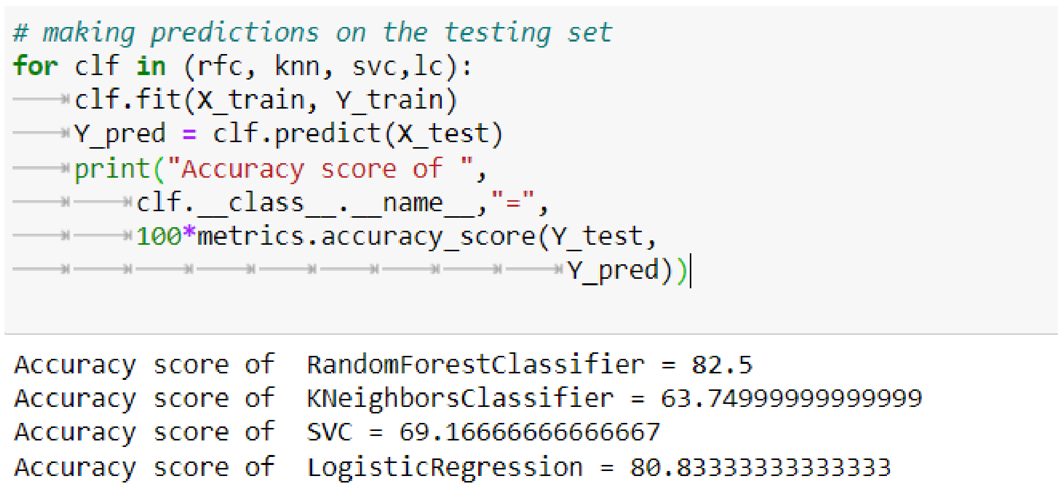
```
Accuracy score of  RandomForestClassifier = 98.04469273743017
Accuracy score of  KNeighborsClassifier = 78.49162011173185
Accuracy score of  SVC = 68.71508379888269
Accuracy score of  LogisticRegression = 79.60893854748603
```

For Test set:

1. Model Prediction:
    - The code iterates over the trained models (presumably trained as shown in the previous snippet you provided).
    - It uses each model to predict outcomes (Y_pred) on the test set features (X_test).
2. Accuracy Computation:
    - The accuracy of the predictions is calculated by comparing them to the actual outcomes (Y_test) using the accuracy_score function from sklearn.metrics.
    - The results are multiplied by 100 to convert the score into a percentage.
3. Output Display:
    - The accuracy percentage for each model is printed out, showing how well each model performs on the test set, which provides an indication of the model's ability to generalize to new, unseen data.

The reported accuracy scores are as follows:

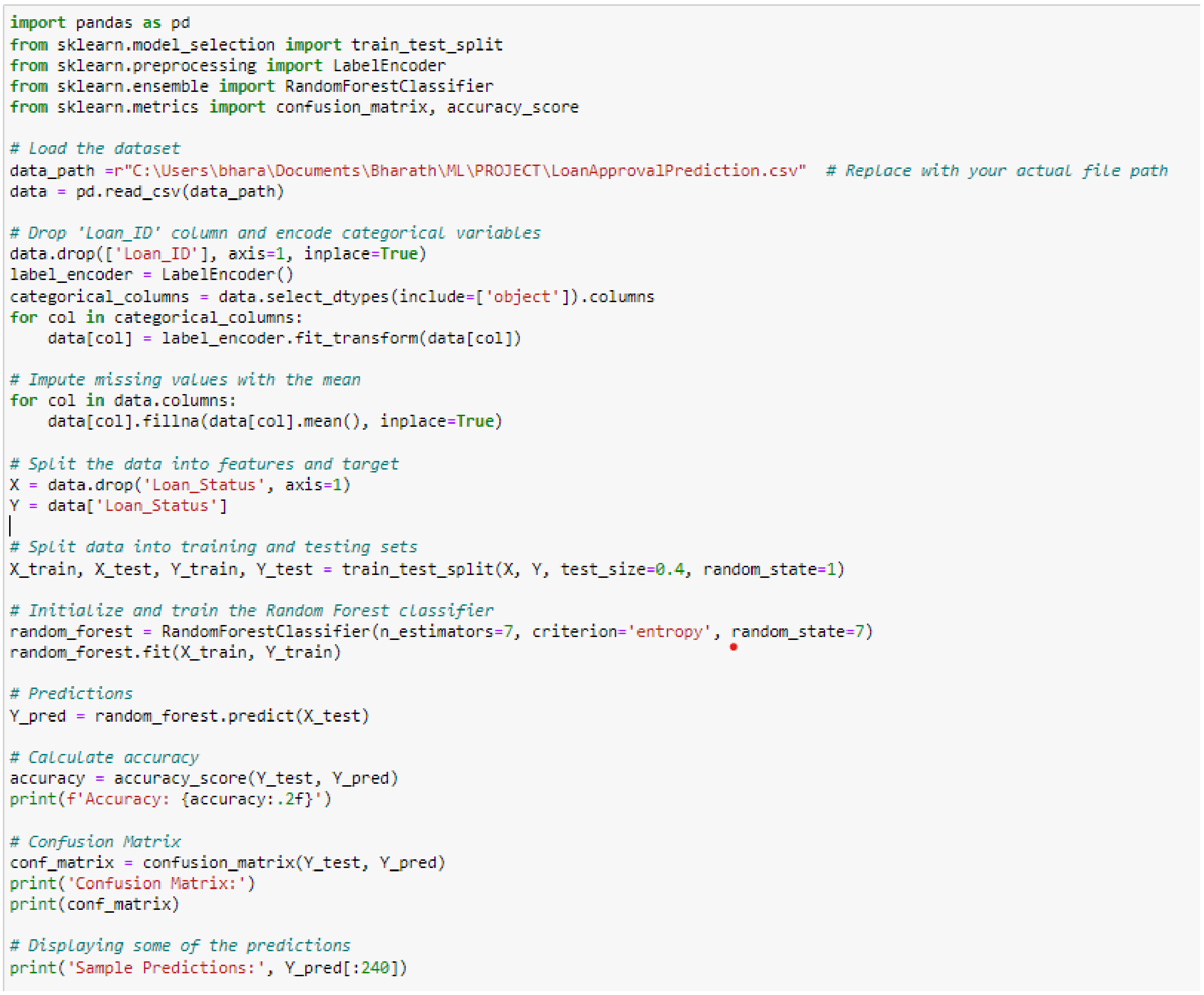
- RandomForestClassifier: 82.5%
- KNeighborsClassifier: Approximately 63.75%
- SVC: Approximately 69.17%
- LogisticRegression: Approximately 80.83%



```python
# making predictions on the testing set
for clf in (rfc, knn, svc,lc):
    clf.fit(X_train, Y_train)
    Y_pred = clf.predict(X_test)
    print("Accuracy score of ",
            clf.__class__.__name__,"=",
            100*metrics.accuracy_score(Y_test,
                                    Y_pred))
```

```
Accuracy score of  RandomForestClassifier = 82.5
Accuracy score of  KNeighborsClassifier = 63.74999999999999
Accuracy score of  SVC = 69.16666666666667
Accuracy score of  LogisticRegression = 80.83333333333333
```

Creating Random Forest Classifier Model:



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score

# Load the dataset
data_path =r"C:\Users\bhara\Documents\Bharath\ML\PROJECT\LoanApprovalPrediction.csv"  # Replace with your actual file path
data = pd.read_csv(data_path)

# Drop 'Loan_ID' column and encode categorical variables
data.drop(['Loan_ID'], axis=1, inplace=True)
label_encoder = LabelEncoder()
categorical_columns = data.select_dtypes(include=['object']).columns
for col in categorical_columns:
    data[col] = label_encoder.fit_transform(data[col])

# Impute missing values with the mean
for col in data.columns:
    data[col].fillna(data[col].mean(), inplace=True)

# Split the data into features and target
X = data.drop('Loan_Status', axis=1)
Y = data['Loan_Status']

# Split data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4, random_state=1)

# Initialize and train the Random Forest classifier
random_forest = RandomForestClassifier(n_estimators=7, criterion='entropy', random_state=7)
random_forest.fit(X_train, Y_train)

# Predictions
Y_pred = random_forest.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(Y_test, Y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Confusion Matrix
conf_matrix = confusion_matrix(Y_test, Y_pred)
print('Confusion Matrix:')
print(conf_matrix)

# Displaying some of the predictions
print('Sample Predictions:', Y_pred[:240])
```

Importing Libraries: Essential libraries are imported for data handling, machine learning, and evaluation.

1. Data Loading: The LoanApprovalPrediction.csv file is loaded into a pandas DataFrame from a specified file path.
2. Data Preprocessing: The Loan_ID column is dropped as it's likely to be a unique identifier that doesn't contribute to the prediction model. Categorical variables are encoded into numeric form using LabelEncoder to make them suitable for machine learning algorithms. Missing values are imputed with the column mean, ensuring that all rows can be used for model training and predictions.
3. Feature Selection: The dataset is split into features (X) and target (Y, the 'Loan_Status' column).
4. Data Splitting: The features and target are split into training and test sets with a 60-40 ratio, using a consistent random state for reproducibility.

5. Model Training: A RandomForestClassifier with 7 trees and entropy as the splitting criterion is trained on the training set.
6. Model Prediction: The trained model makes predictions on the test set.
7. Evaluation: The model's accuracy is computed by comparing the predictions with the actual loan statuses. The confusion matrix is also computed to understand the true positives, true negatives, false positives, and false negatives.
8. Output: The script prints the accuracy, the confusion matrix, and a sample of 240 predictions made by the model.

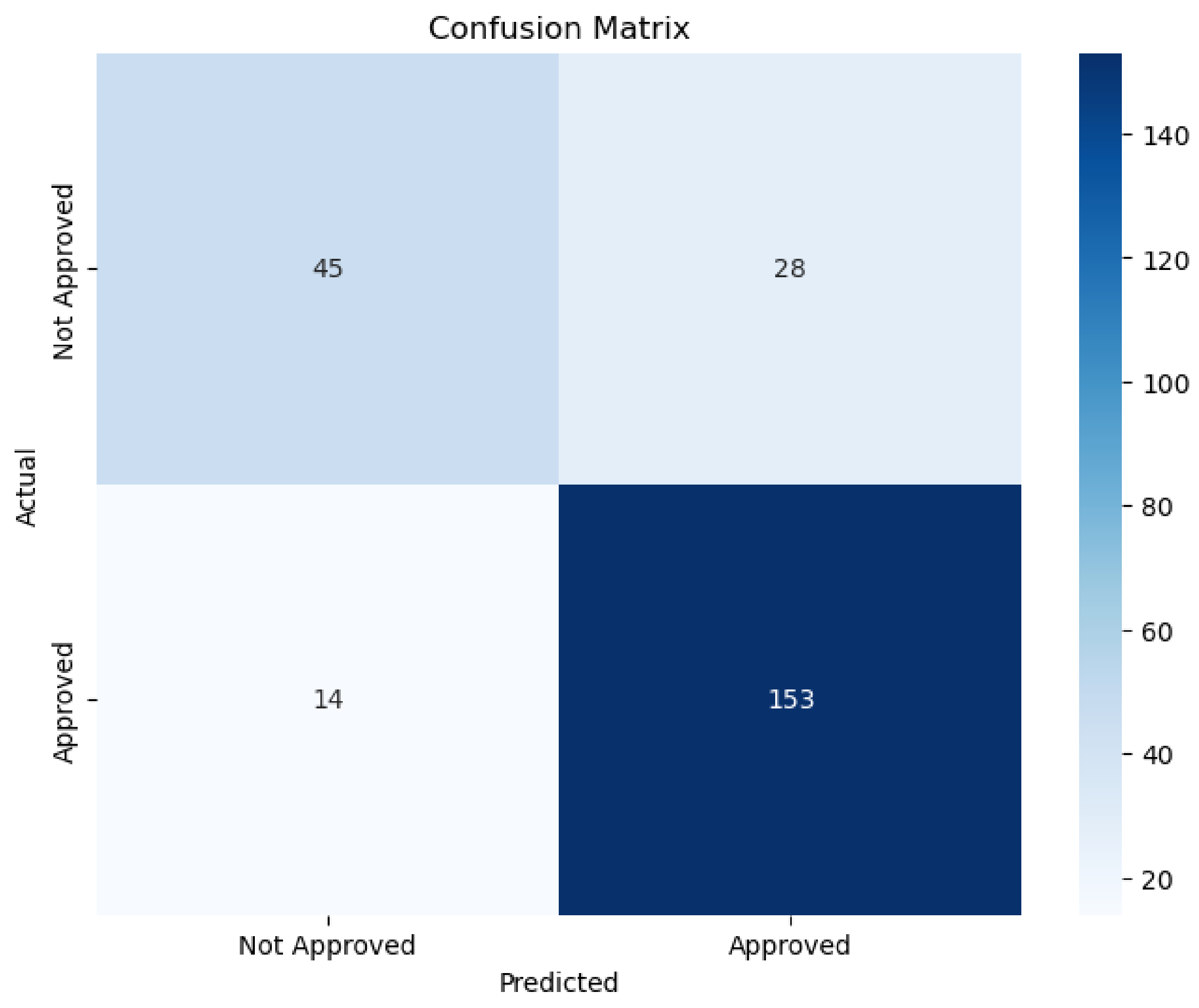
Model Evaluation:

```
Accuracy: 0.82
Confusion Matrix:
[[ 45  28]
 [ 14 153]]
Sample Predictions: [1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 0 0 1 1 1 0 1 1 1 0
 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1
 0 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 1 1 0 0 0 1
 1 0 0 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 0 0 0 0 1
 1 1 1 1 1 0 0 1 1 1 0 1 1 0 1 1 1 1 0 0 1 1 0 0 1 0 1 1 1 1 1 1 1 0 0 1 0
 1 1 0 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1]
```

1. Accuracy: The model achieved an accuracy of 0.82 or 82%, which indicates that 82% of the time, the model's predictions on the test dataset were correct.
2. Sample Predictions: An array of 0s and 1s showing individual predictions for the test dataset, where 1 might represent a loan being approved and 0 representing a loan not being approved.

Visualization of Model Predictions:

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

# Assuming 'Y_test' and 'Y_pred' are already defined as your actual and predicted labels
conf_matrix = confusion_matrix(Y_test, Y_pred)

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



Confusion Matrix: Presented in a 2x2 array format:

- Top-left (45): True negatives (TN), the count of actual negatives (loan not approved) correctly predicted by the model.
- Top-right (28): False positives (FP), the count of actual negatives wrongly predicted as positives (loan approved).
- Bottom-left (14): False negatives (FN), actual positives (loan approved) wrongly predicted as negatives.
- Bottom-right (153): True positives (TP), actual positives correctly predicted by the model.

Conclusion

In this project, we developed a predictive model to determine the approval status of loan applications. Through a comprehensive workflow involving data preprocessing, feature engineering, model selection, and evaluation, we employed various machine learning algorithms, with the Random Forest Classifier emerging as the most accurate model based on our dataset. An accuracy of 82% on the test set indicates that our model can reasonably predict loan approval outcomes, which can significantly streamline the decision-making process for financial institutions. The confusion matrix revealed that while the model is robust in identifying true positives and true negatives, there is room to reduce false positives and false negatives, enhancing the trustworthiness of the predictions.

Future Scope:

1. **Model Optimization:** Future work could involve hyperparameter tuning for the Random Forest Classifier to further improve its performance. Techniques like grid search and random search could yield more optimized parameters that enhance the model's predictive power.
2. **Feature Engineering:** Additional feature engineering could provide deeper insights into the predictive capability of the dataset. For instance, creating new features that capture the interactions between existing features might improve the model's performance.
3. **Alternative Models:** Experimenting with more advanced machine learning algorithms, including ensemble methods like Gradient Boosting or Extreme Gradient Boosting (XGBoost), could lead to better outcomes. Deep learning approaches could also be explored, especially if the dataset is large and complex.
4. **Data Augmentation:** Incorporating more data points and additional features, such as macroeconomic indicators or more detailed credit history, may lead to more accurate predictions.

What we have learnt:

Throughout this project, we delved into the practical application of machine learning to predict loan approval outcomes, which enhanced our collective knowledge and skill set in several key areas of data science.

In terms of data preprocessing, we learned the criticality of meticulously cleaning and preparing our dataset. This encompassed managing missing values, encoding categorical data, and selecting pertinent features that contribute meaningfully to the predictive models. The process underscored the necessity of thorough data inspection and preprocessing to ensure that the input data would lead to reliable predictions.

Our exploration of machine learning algorithms was both broad and deep. We familiarized ourselves with various algorithms, understanding how and when to apply models like K-Nearest

Neighbors, Support Vector Machines, Logistic Regression, and Random Forest. Through comparative analysis, we grasped the nuances of each model’s performance and suitability depending on the nature of the data and the prediction task at hand.

The evaluation phase of the project was particularly enlightening. We ventured beyond mere accuracy to assess our models using a variety of metrics, including confusion matrices. This not only informed us about the predictive power of our models but also about their reliability and the types of errors they were prone to make. These insights were pivotal in refining our models to ensure that they could be trusted for real-world application.

During this process, we sharpened our technical acumen, particularly with Python programming and key data science libraries such as pandas, scikit-learn, and matplotlib. Working in Jupyter Notebooks, we cultivated a proficient workflow that allowed for seamless transition from data analysis to machine learning and visualizations.

Critical thinking was another domain we collectively nurtured. We learned to question our models critically, probing the 'why' and 'how' behind their predictions. A model's decision-making process can be as critical as its accuracy, particularly in a domain as impactful as financial lending.

Our problem-solving capabilities were also put to the test, often prompting us to research extensively to overcome obstacles and improve our models' performance. This not only improved our technical know-how but also kept us engaged with the latest trends and best practices in machine learning.

Furthermore, we confronted the ethical dimensions of machine learning. We recognized that algorithms could inadvertently introduce or perpetuate bias, which taught us to prioritize fairness and transparency in our predictive modeling efforts.

The project also polished our ability to communicate complex ideas effectively. We learned to distill technical results into comprehensible insights, presenting our findings through clear visualizations and articulate reports, which is a crucial skill when liaising with stakeholders from non-technical backgrounds.

Finally, this project was a testament to the iterative nature of machine learning. We embraced the philosophy of continuous improvement, acknowledging that there's always scope for refinement, whether through model tuning, incorporating additional data, or experimenting with emerging algorithms.

This collaborative venture has not only bolstered our technical expertise but also enhanced our teamwork and problem-solving abilities, equipping us to confidently address more intricate challenges in the field of data science together.