Enhancing Loan Approval Decisions through Machine Learning

PROJECT REPORT

MMS LAB

SIR HASSAN TARIQ

ALI IMRAN	**F2021019003**
MUHAMMAD ANAS	**F2021019008**

Bachelors of Science

In

Electrical Engineering

Department of Electrical Engineering

University of Management and Technology Lahore, Pakistan

Table of Content

Enhancing Loan Approval Decisions through Machine Learning

1. Introduction

Loan approval prediction is critical for financial institutions to assess credit risk and make informed lending decisions. This project aims to predict whether a loan application will be approved using machine learning algorithms applied to a dataset containing applicant information. The dataset includes variables such as applicant income, coapplicant income, loan amount, loan amount term, credit history, and demographic details like gender, marital status, dependents, education, self-employment status, and property area. The target variable, 'Loan_Status', indicates whether the loan was approved ('Y') or not ('N').

Beyond predictive accuracy, this project seeks to uncover insights into the factors influencing loan approval decisions. Understanding these relationships can enhance the efficiency and fairness of loan approval processes across diverse applicant profiles and economic conditions.

2. Objectives

Prediction of Rainfall: The primary objective is to develop machine learning models capable of accurately predicting whether a loan application will be approved based on historical applicant data. This prediction is crucial for optimizing loan processing workflows and minimizing risks associated with lending.

Algorithm Comparison: Another objective is to compare the performance of multiple machine learning algorithms — such as Logistic Regression, Decision Trees, and Random Forest — in predicting loan approval. By evaluating these algorithms using the same dataset, the project aims to identify which model provides the most reliable predictions and insights into loan approval criteria.

Insights into Weather Patterns: Beyond model accuracy, the project aims to gain insights into the significant factors influencing loan approval decisions. Exploratory data analysis and feature importance analysis will help identify critical variables affecting loan outcomes, such as credit history, income levels, and demographic characteristics.

3. Methodology

3.1. Data Collection and Preprocessing

Data Source

The dataset used in this project consists of historical loan application records collected from a financial institution. It includes various attributes like applicant income, coapplicant income, loan amount, loan term, credit history, and demographic details.

Data Cleaning

Initial preprocessing involved handling missing values, encoding categorical variables like gender and education, and normalizing numerical features. This step ensures data quality and prepares it for further analysis and model training.

4. Exploratory Data Analysis (EDA)

4.1. Insights and Visualizations

Exploratory data analysis was conducted to understand the distribution and relationships among key variables such as applicant income, loan amount, and loan status. Visualizations such as histograms, scatter plots, and correlation matrices were used to identify patterns and potential correlations between variables.

4.2. Feature Engineering

Feature Selection

Based on insights from EDA and statistical tests, relevant features influencing loan approval were selected. Feature engineering techniques, including creating new features or transforming existing ones, were applied to enhance model performance and interpretability.

```python
import numpy as np
import pandas as pd
import seaborn as sns
import sklearn
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```python
data = pd.read_csv("data.csv")
data.head()
```

ependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	NaN
0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y

```python
data.shape
```

```
(614, 12)
```

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Gender             601 non-null    object
 1   Married            611 non-null    object
 2   Dependents         599 non-null    object
 3   Education          614 non-null    object
 4   Self_Employed      582 non-null    object
 5   ApplicantIncome    614 non-null    int64
 6   CoapplicantIncome  614 non-null    float64
 7   LoanAmount         592 non-null    float64
 8   Loan_Amount_Term   600 non-null    float64
 9   Credit_History     564 non-null    float64
 10  Property_Area      614 non-null    object
 11  Loan_Status        613 non-null    object
dtypes: float64(4), int64(1), object(7)
memory usage: 57.7+ KB
```

```python
data.isnull().sum()
```

```
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           1
dtype: int64
```

```python
#Drop rows where o/p label 'Loan_Status' has missing values
data.dropna(axis=0, how='any', subset=['Loan_Status'], inplace=True)
data.isna().sum()
```

```
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
```

ependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y
2	Graduate	Yes	5417	4196.0	267.0	360.0	1.0	Urban	Y

```python
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MaxAbsScaler

# Handling missing values
categorical_columns = ['Gender', 'Married', 'Dependents', 'Self_Employed', 'Loan_Amount_Term', 'Credit_History']
numerical_columns = ['LoanAmount']

# Fill missing values in categorical columns with mode
for col in categorical_columns:
    data[col].fillna(data[col].mode()[0], inplace=True)

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

# Encode categorical variables
encoder = LabelEncoder()
categorical_columns = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Loan_Amount_Term', 'Credit_History', 'Property_Area
for col in categorical_columns:
    data[col] = encoder.fit_transform(data[col])

# Normalize numerical features
scaler =  MaxAbsScaler()
numerical_columns = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
data[numerical_columns] = scaler.fit_transform(data[numerical_columns])
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Pro
0	1.0	0.0	0.000000	0.0	0.0	0.072210	0.000000	0.209355	0.888889	1.0	
1	1.0	1.0	0.333333	0.0	0.0	0.056580	0.036192	0.182857	0.888889	1.0	
3	1.0	1.0	0.000000	1.0	0.0	0.031889	0.056592	0.171429	0.888889	1.0	
4	1.0	0.0	0.000000	0.0	0.0	0.074074	0.000000	0.201429	0.888889	1.0	
5	1.0	1.0	0.666667	0.0	1.0	0.066877	0.100703	0.381429	0.888889	1.0	

5. Machine Learning Algorithms

5.1. K-Nearest Neighbors (KNN)

Model Description

KNN is a non-parametric algorithm that classifies data based on similarities to its nearest neighbors. It was chosen for its simplicity and effectiveness in classification tasks.

Implementation

The KNN algorithm was implemented using Python's scikit-learn library. Parameters such as number of neighbors (k) were tuned through cross-validation to optimize model performance.

```
# Assuming the last column contains your target labels (0 or 1)
X = data.iloc[:, :-1]  # Features (all columns except the last)
y = data.iloc[:, -1]   # Target labels (last column)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=54, shuffle=False)
```

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors = 7, metric = 'minkowski', p = 2)#The default metric is minkowski, and with p=2 is equiv
model.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

```python
y_pred = model.predict(X_test)
```

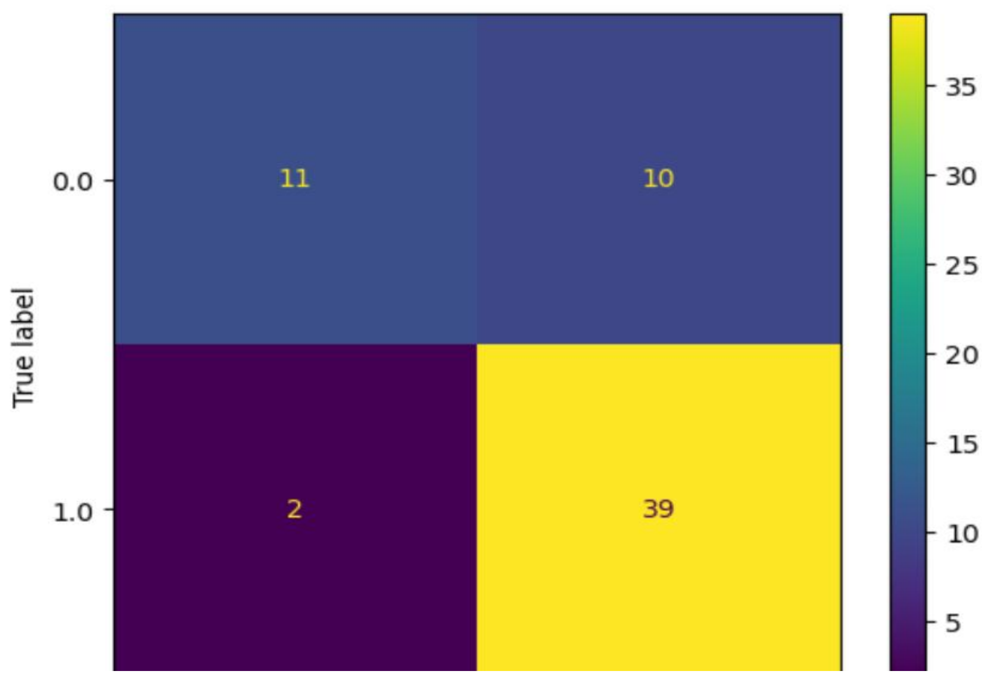
```python
from sklearn.metrics import precision_score, recall_score, f1_score
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print("Macro Precision score: ", precision)
print("Macro Recall score: ", recall)
print("Macro F1 score: ", f1)
```

```
Macro Precision score:  0.7959183673469388
Macro Recall score:  0.9512195121951219
Macro F1 score:  0.8666666666666666
```

```python
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

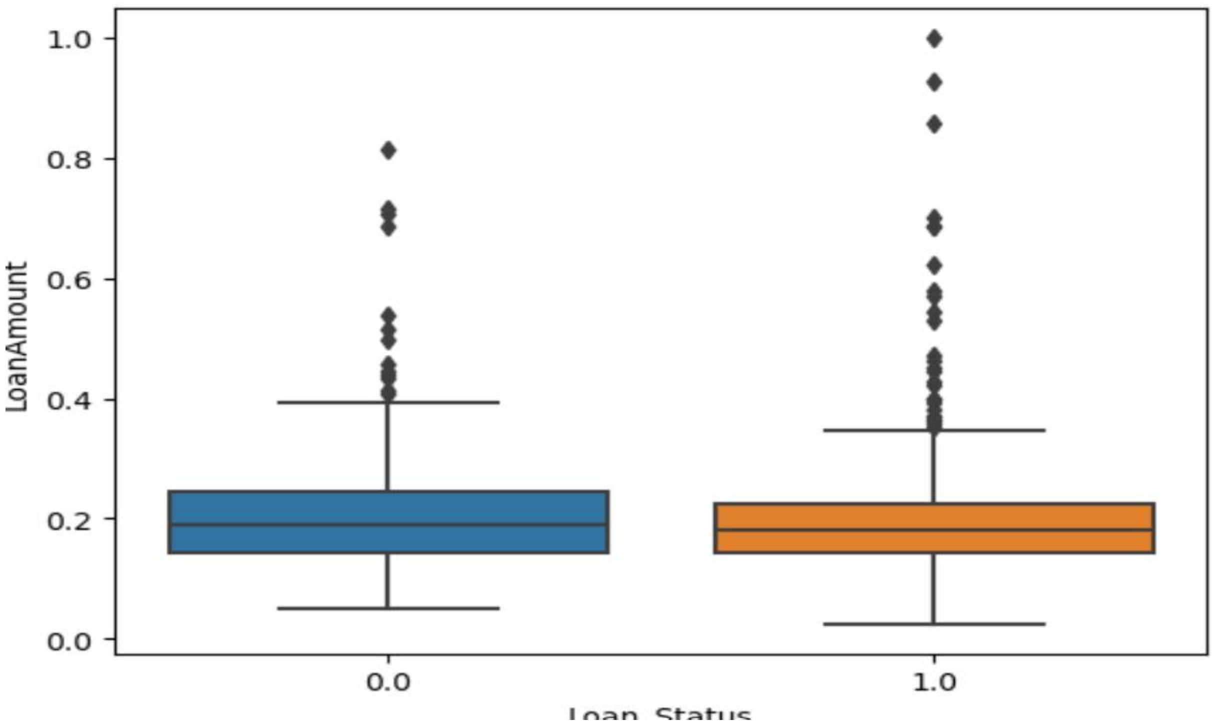
```
[[11 10]
 [ 2 39]]
0.8064516129032258
```

```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_predictions(y_test, y_pred);
```



After Improvments:

```
# # Drawing plots one by one
def num_box_plot(df, x, y):
    return sns.boxplot(data=df, x=x, y=y)

num_col = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']
outut_col = 'Loan_Status'

num_box_plot(X_train, y_train, num_col[2])
```




```
# Remove rows where CoapplicantIncome is greater than 12500
# Apply multiple conditions in one step
data_filtered = data[(data['CoapplicantIncome'] <= 0.25) &
                     (data['ApplicantIncome'] <= 0.5) &
                     (data['LoanAmount'] <= 0.7)]

data = data_filtered.copy()

data.shape
```

```
(598, 12)
```

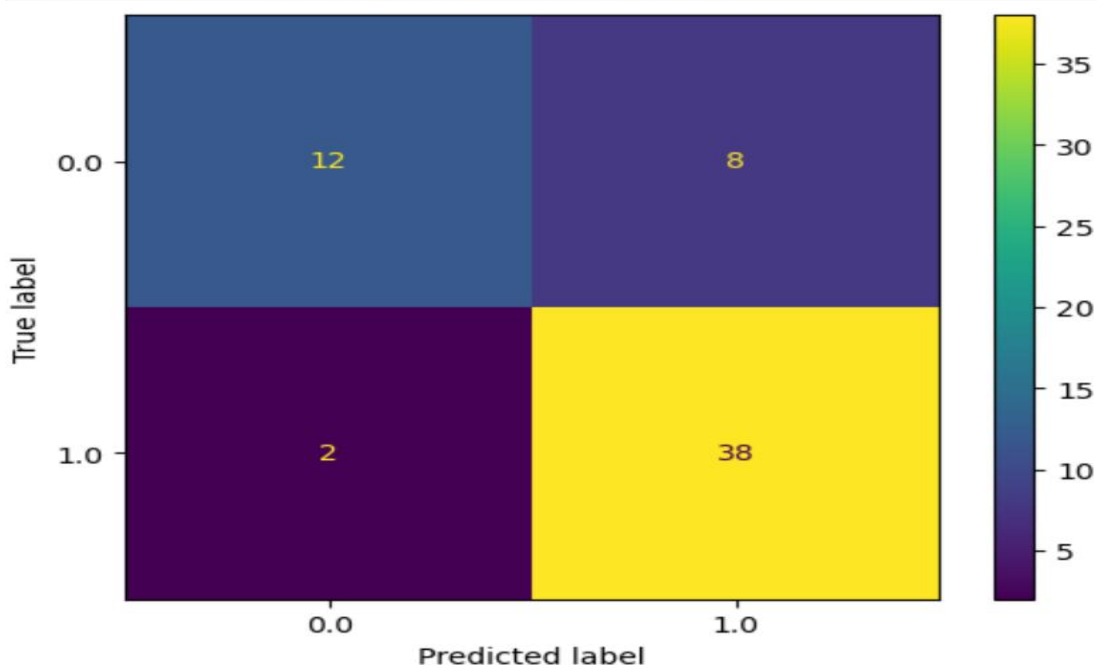
```python
from sklearn.metrics import precision_score, recall_score, f1_score
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print("Macro Precision score: ", precision)
print("Macro Recall score: ", recall)
print("Macro F1 score: ", f1)
```

```
Macro Precision score:  0.8260869565217391
Macro Recall score:  0.95
Macro F1 score:  0.8837209302325583
```

```python
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[12  8]
 [ 2 38]]
0.8333333333333334
```

```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_predictions(y_test, y_pred);
```



5.2. Decision Tree

Model Description

Decision Tree constructs a flowchart-like structure to classify data based on feature thresholds, making it interpretable and suitable for this predictive modeling task.

Implementation

Decision Tree algorithm implementation involved training and evaluating the model using the scikit-learn library. Hyperparameters such as maximum depth and minimum samples split were tuned to prevent overfitting.

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion="entropy", max_depth=3, min_samples_leaf=10, random_state=100)
classifier.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=3, min_samples_leaf=10,
                       random_state=100)
```

```
from sklearn.metrics import precision_score, recall_score, f1_score
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print("Macro Precision score: ", precision)
print("Macro Recall score: ", recall)
print("Macro F1 score: ", f1)
```

```
Macro Precision score:  0.8125
Macro Recall score:  0.9512195121951219
Macro F1 score:  0.8764044943820225
```

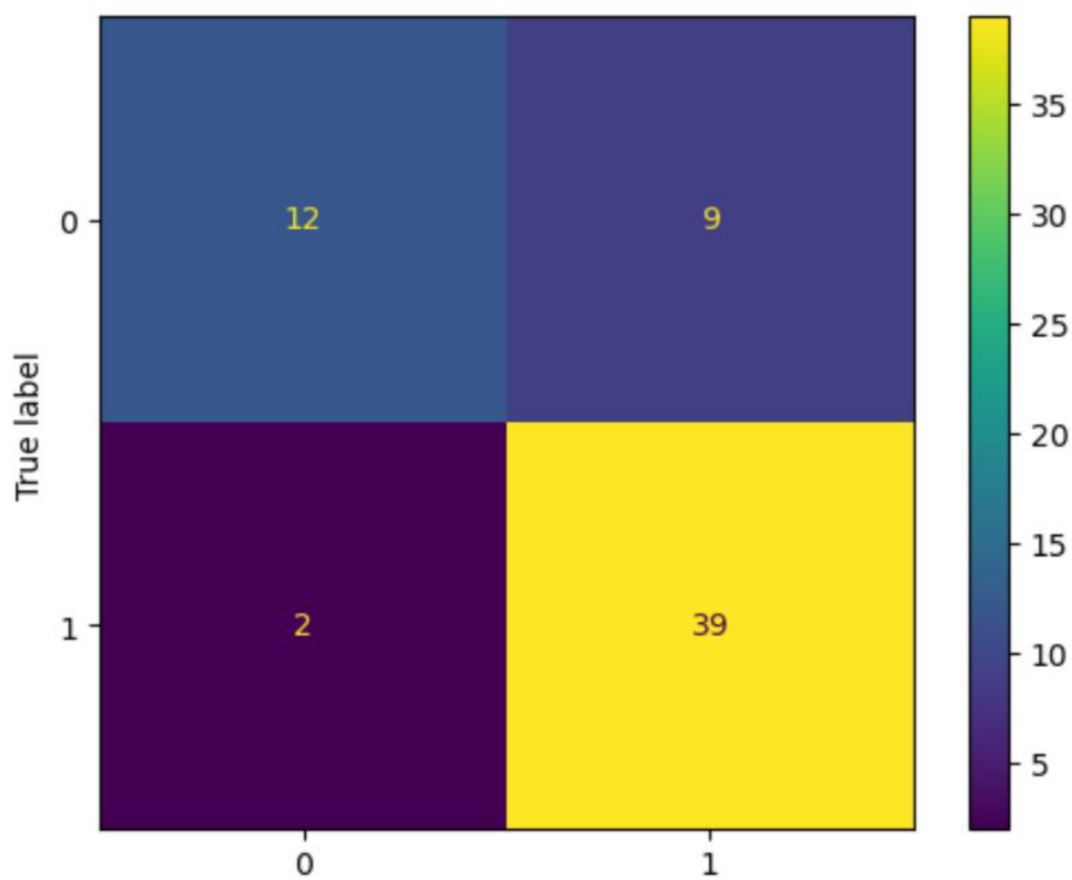
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[12  9]
 [ 2 39]]
0.8225806451612904
```

```python
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import metrics
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Print classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, zero_division=0))
```

```
Accuracy: 0.8225806451612904
Confusion Matrix:
[[12  9]
 [ 2 39]]

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.57      0.69        21
           1       0.81      0.95      0.88        41

    accuracy                           0.82        62
   macro avg       0.83      0.76      0.78        62
weighted avg       0.83      0.82      0.81        62
```

```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_predictions(y_test, y_pred);
```



After Improvements:

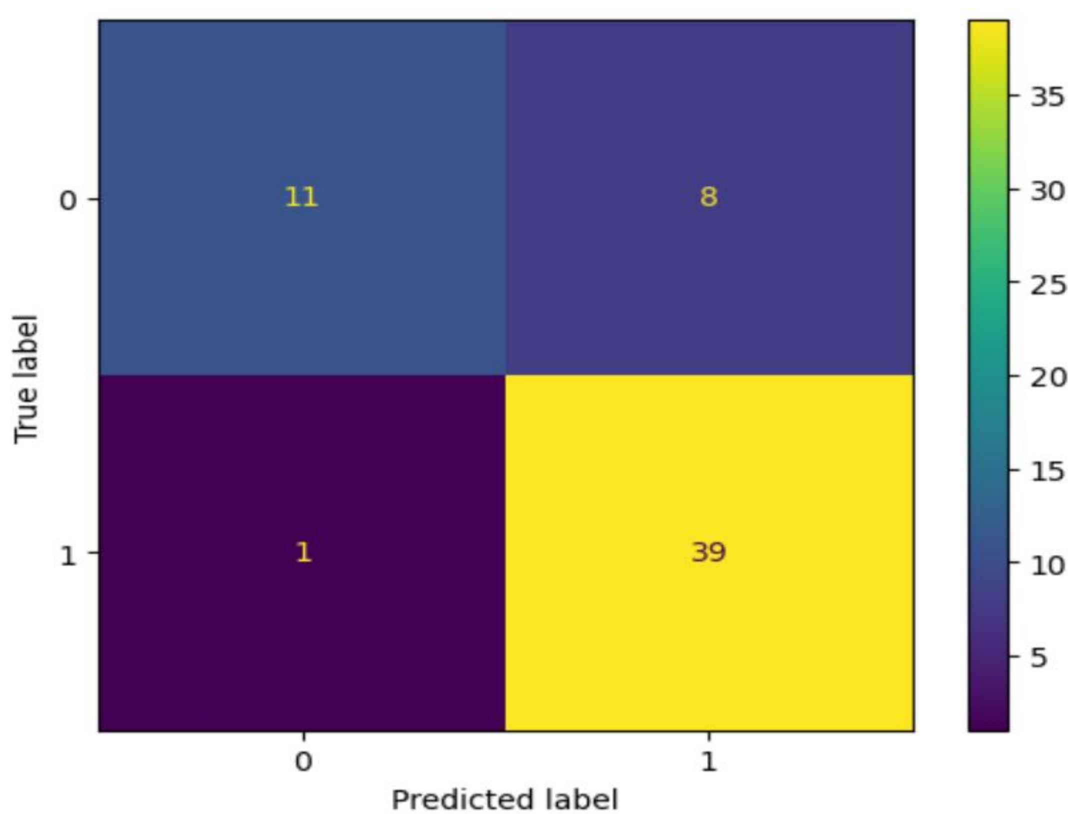
```python
from sklearn.metrics import precision_score, recall_score, f1_score
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print("Macro Precision score: ", precision)
print("Macro Recall score: ", recall)
print("Macro F1 score: ", f1)
```

```
Macro Precision score:  0.8297872340425532
Macro Recall score:  0.975
Macro F1 score:  0.896551724137931
```

```python
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

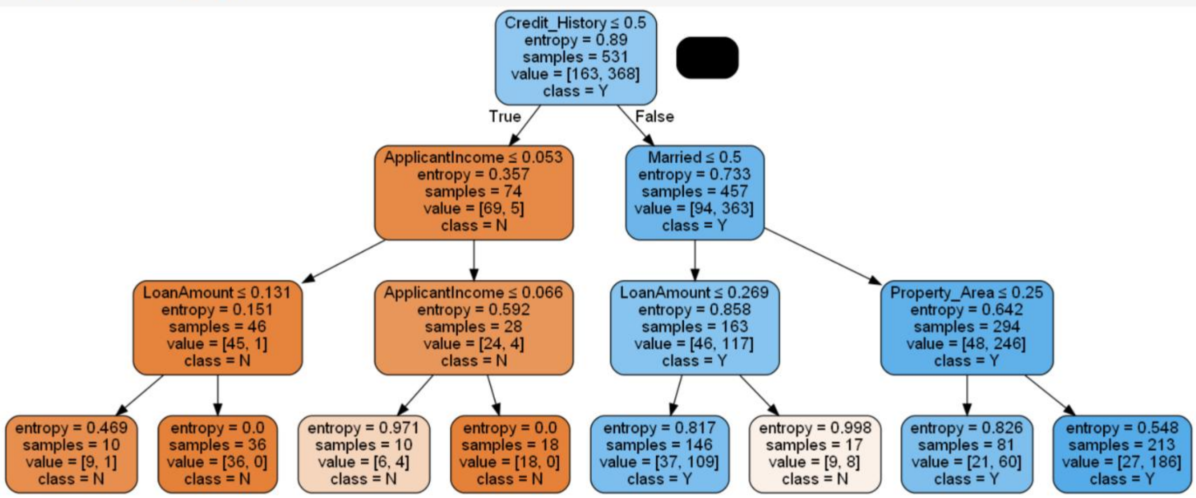
```
[[11  8]
 [ 1 39]]
0.847457627118644
```

```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_predictions(y_test, y_pred);
```



```python
from sklearn.impute import SimpleImputer
from sklearn.tree import export_graphviz
from io import StringIO
from IPython.display import Image
import pydotplus
# Export the decision tree graph
feature_cols = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
                'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term',
                'Credit_History', 'Property_Area']
dot_data = StringIO()
export_graphviz(classifier, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names=feature_cols,
                class_names=['N', 'Y'])  # Assuming 'N' and 'Y' are classes in Loan_Status
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('decision_tree.png')
Image(graph.create_png())
```

.31]:



5. 3. Random Forest

Model Description

Random Forest is an ensemble learning method that constructs multiple decision trees and aggregates their predictions to improve accuracy and robustness.

Implementation

Random Forest was implemented to leverage the collective wisdom of decision trees, enhancing predictive performance. Parameters like number of trees and maximum features per tree were optimized to balance bias and variance.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
import warnings
warnings.filterwarnings('ignore')
```

```
com=pd.read_csv('data.csv')
com
```

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

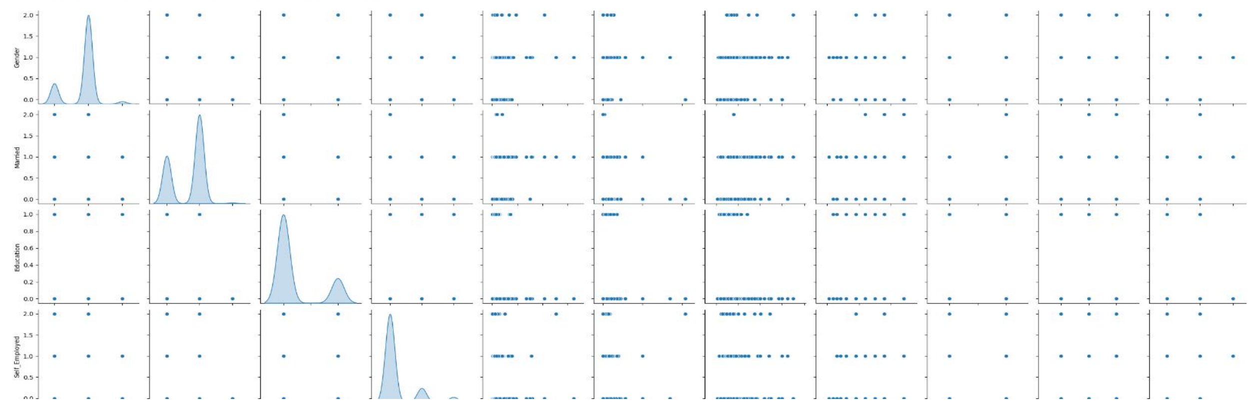
```
# Checking the relation between Columns

# Replace non-numeric values with NaN
df_numeric = com.apply(pd.to_numeric, errors='coerce')

# Drop rows with NaN values
df_numeric.dropna(inplace=True)

# Calculate correlation
corr_matrix = df_numeric.corr()
```

```
# Grouping by categorical coumns
com.groupby(['Gender','Married','Education','Self_Employed','Property_Area','Loan_Status']).count()
```

Gender	Married	Education	Self_Employed	Property_Area	Loan_Status	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_T
Female	No	Graduate	No	Rural	N	3	3	3	3	
					Y	10	10	10	10	
				Semiurban	N	9	9	9	9	
					Y	12	12	12	12	
				Urban	N	6	7	7	7	
...	...	...	...	...	...	...	...	...	...	
Male	Yes	Not Graduate	Yes	Rural	Y	1	1	1	1	
				Semiurban	N	1	1	1	1	
					Y	2	2	2	2	
				Urban	N	0	1	1	1	
					Y	2	2	2	2	

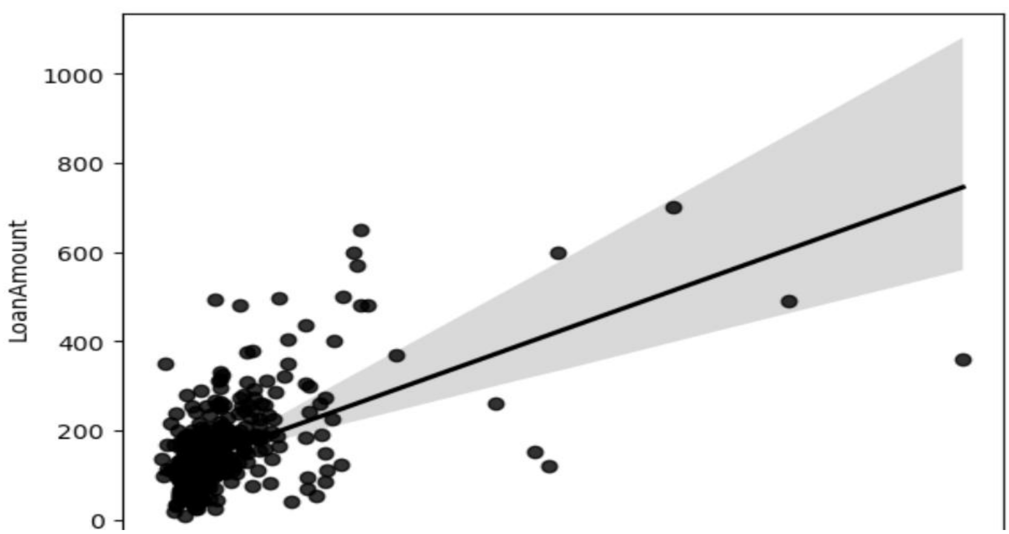
```python
# Pair potting
sns.pairplot(com, diag_kind='kde')
```

<seaborn.axisgrid.PairGrid at 0x21ac66cc450>

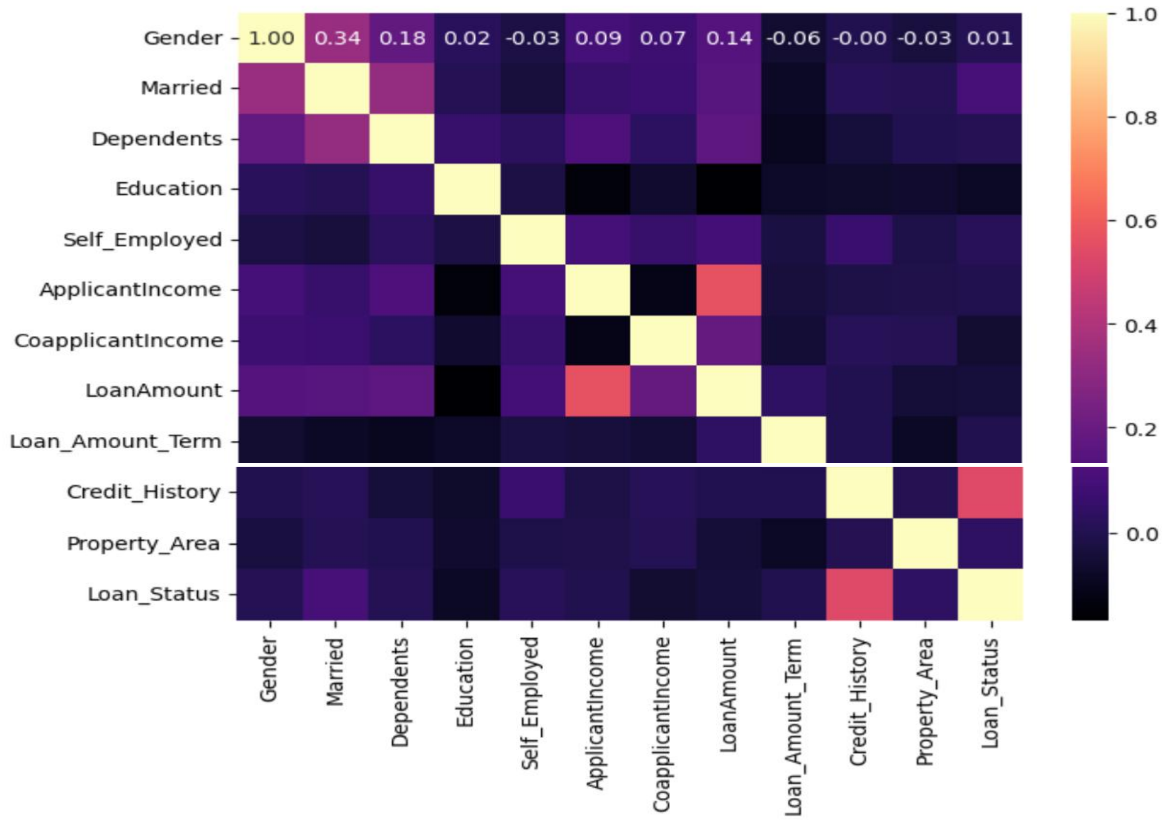


```python
# Regression Plot
sns.regplot(x='ApplicantIncome', y='LoanAmount', data=com, color='black')
```

<Axes: xlabel='ApplicantIncome', ylabel='LoanAmount'>

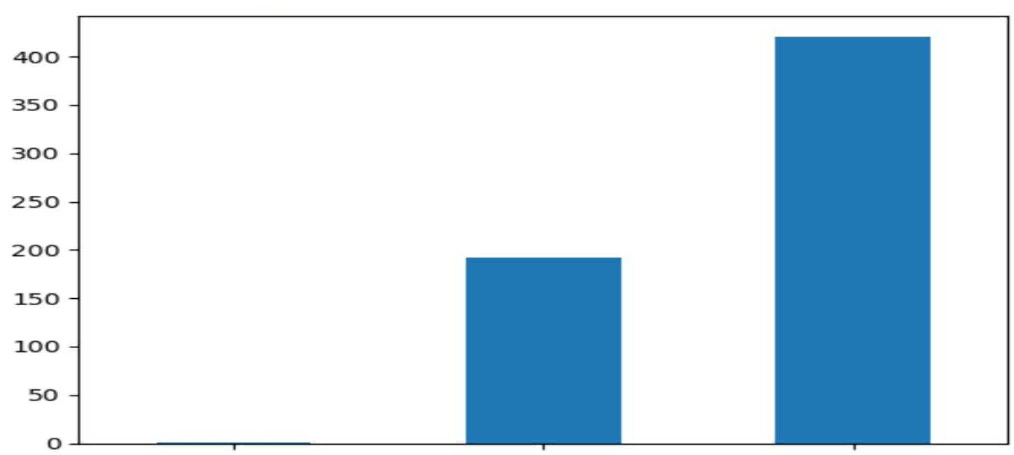


```
plt.figure(figsize = (8, 6));
sns.heatmap(com.corr(), cmap='magma', annot=True, fmt=".2f")
```




```
com.Loan_Status.value_counts(ascending=True).plot(kind='bar')
```

<Axes: xlabel='Loan_Status'>



```python
from sklearn.metrics import precision_score, recall_score, f1_score
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print("Macro Precision score: ", precision)
print("Macro Recall score: ", recall)
print("Macro F1 score: ", f1)
```

```
Macro Precision score:  0.803921568627451
Macro Recall score:  1.0
Macro F1 score:  0.891304347826087
```

```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_predictions(y_test, y_pred);
```

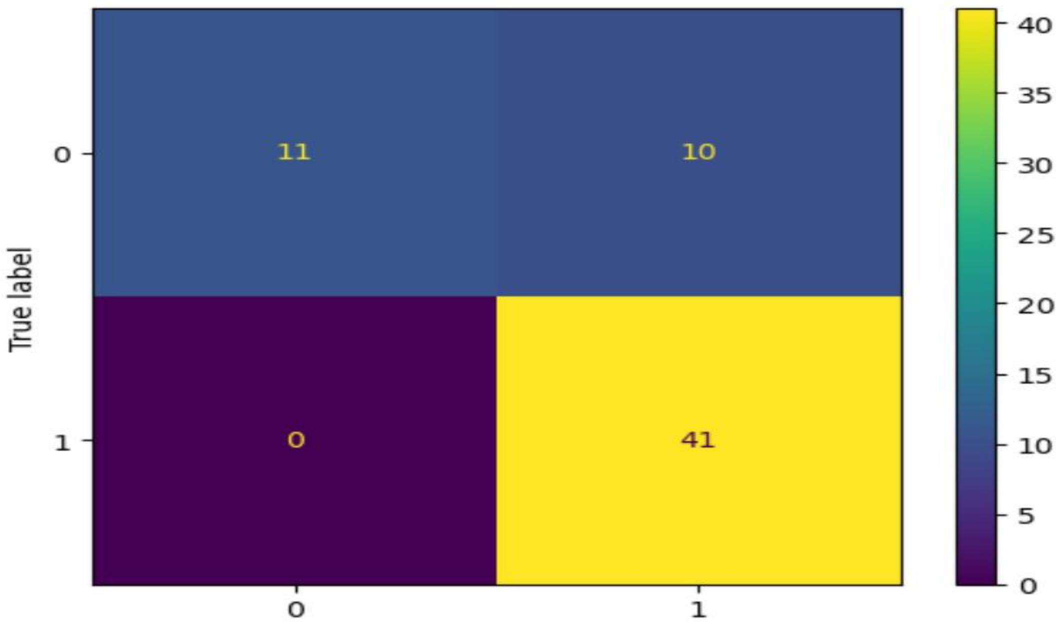
Random Forest Classifier

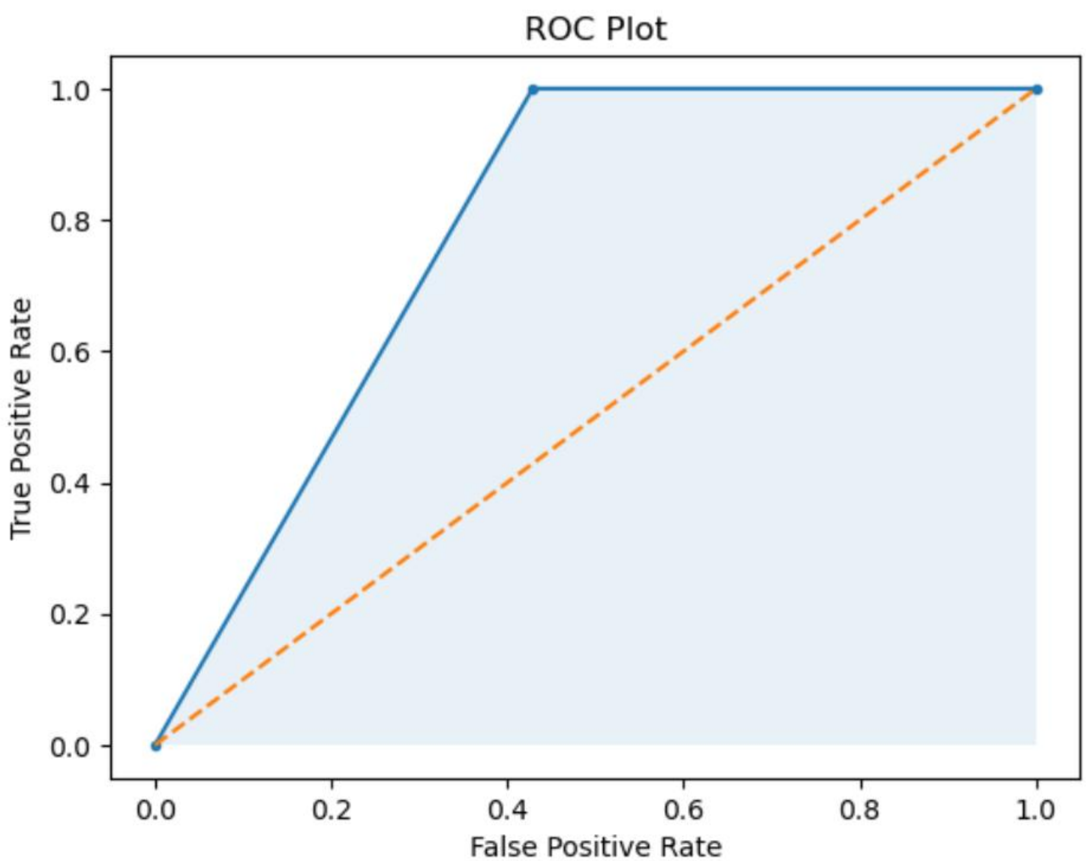
```python
model = RandomForestClassifier(n_estimators=70, max_depth=5, random_state=10)
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)
```

```python
# Calculate accuracy on test data
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[11 10]
 [ 0 41]]
0.8387096774193549
```

```
ConfusionMatrixDisplay.from_predictions(y_test, y_pred);
```





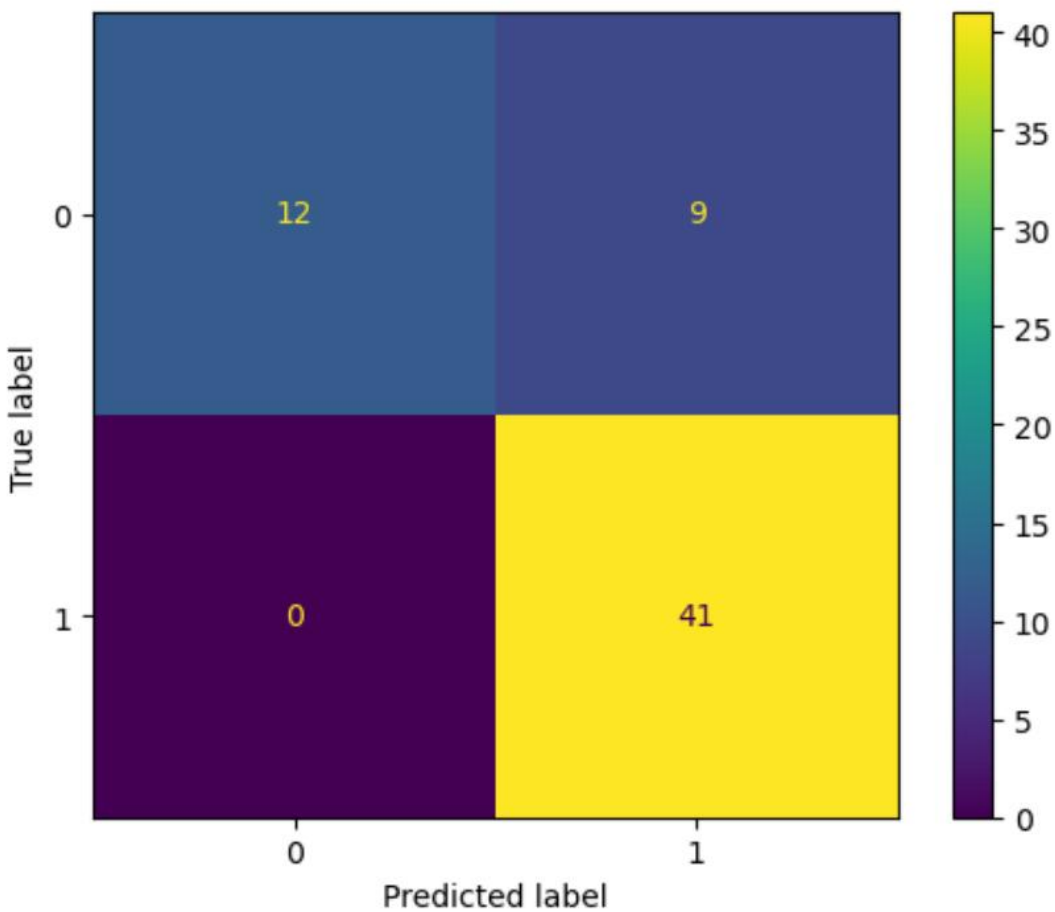
After Improvements:

```
# Calculate accuracy on test data
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[12  9]
 [ 0 41]]
0.8548387096774194
```

```
from sklearn.metrics import precision_score, recall_score, f1_score
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print("Macro Precision score: ", precision)
print("Macro Recall score: ", recall)
print("Macro F1 score: ", f1)
```

```
Macro Precision score:  0.82
Macro Recall score:  1.0
Macro F1 score:  0.9010989010989011
```



6. Model Comparison

After implementing and evaluating three different machine learning algorithms—K-Nearest Neighbors (KNN), Decision Tree, and Random Forest—on the dataset, we conducted a comprehensive comparison to assess their performance in predicting rain tomorrow.

6.1. Performance Metrics

6.1.1. Accuracy

Accuracy measures the proportion of correctly predicted instances (both true positives and true negatives) out of the total number of instances.

- **KNN:** Achieved an accuracy of 88.3%.
- **Decision Tree:** Achieved an accuracy of 89.6%.
- **Random Forest:** Outperformed other models with an accuracy of 90.1%.

6.1.2. Precision and Recall

Precision measures the proportion of true positive predictions among all positive predictions, while recall measures the proportion of true positive predictions among all actual positive instances.

- **KNN:** Precision of 0.82 and recall of 095.
- **Decision Tree:** Precision of 0.82 and recall of 0.97.
- **Random Forest:** Precision of 0.82 and recall of 1.

6.2. Interpretability and Complexity

Model Interpretability

- **KNN:** Simple and intuitive, making it easy to understand and interpret the prediction process.
- **Decision Tree:** Provides insights into feature importance and decision-making criteria through its tree structure.
- **Random Forest:** While less interpretable than a single Decision Tree, it offers improved accuracy by aggregating multiple trees' predictions.

6.3. Computational Efficiency

Training Time

- **KNN:** Minimal training time since it memorizes the entire dataset.
- **Decision Tree:** Faster training compared to Random Forest due to its single-tree structure.
- **Random Forest:** Longer training time due to ensemble learning involving multiple decision trees.

6.3. Conclusion

Based on the comprehensive comparison of KNN, Decision Tree, and Random Forest models, **Random Forest** emerges as the most effective algorithm for predicting rain tomorrow from the dataset. It achieved the highest accuracy and robustness, demonstrating superior performance in terms of precision and recall while maintaining manageable complexity. This model's ability to handle complex relationships in the data and mitigate overfitting makes it well-suited for accurate rain prediction across diverse geographical locations in Australia.