Loan Approval Prediction using Machine Learning

By Lindokuhle Lufele

Loans are an essential aspect of modern life, enabling individuals to achieve their financial goals, whether it's pursuing higher education, buying a dream home, or acquiring luxury items. Banks and financial institutions rely heavily on loan disbursements to generate a significant portion of their profits. Effective loan management is crucial to ensure that borrowers can repay their debts, and lenders can minimize risks and maximize returns.

A machine learning loan prediction model is a powerful tool used to predict the likelihood of a borrower repaying a loan. This model leverages historical data, machine learning algorithms , and statistical techniques to identify patterns and trends that can inform lending decisions.

Objectives

The primary objective of this model is to accurately predict loan approvals and minimize the risk of default. By doing so, lenders can:

- **Improve Credit Risk Assessment**: Identify high-risk borrowers and make informed lending decisions.
- **Reduce Default Rates**: Minimize financial losses associated with loan defaults.
- **Enhance Customer Experience**: Provide faster and more accurate loan approvals to borrowers.

Methodology

The model employs a combination of machine learning algorithms, including:

- **Logistic Regression**: A statistical method used to predict the probability of a binary outcome (loan approval or rejection).
- **Decision Trees**: A tree-based model used to identify patterns and relationships in data.
- **Random Forest**: An ensemble learning method that combines multiple decision trees to improve prediction accuracy.
- **Support Vector Machines (SVM)**: A machine learning algorithm used to classify data and predict outcomes.

Data Preprocessing

The model utilizes a dataset containing 13 features, including:

Name	Data Type
Applicant Income	vchar
Co-Applicant Income	vchar
Loan Amount	int
Loan Term	int
Credit History	date
Property Area	vchar
Marital Status	vchar
Education	vchar
Self-Employed	vchar
Dependents	int
Gender	vchar

Exploratory Data analysis

- The data is preprocessed to handle missing values, outliers, and duplicates. Feature selection techniques are also employed to identify the most relevant features that impact loan approval decisions.

Model Evaluation

Model	Best Parameters	Accuracy Score (%)
RandomForest	{'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}	100.00%
KNeighbors	{'n_neighbors': 9, 'weights': 'distance'}	100.00%
SVC	{'C': 10, 'kernel': 'linear'}	97.00%
LogisticRegression	{'C': 1, 'solver': 'saga'}	100.00%

- **Random Forest and K Neighbors achieve perfect accuracy**: Both models demonstrate exceptional performance, achieving an accuracy score of 100.00% on the testing set. This suggests that these models are highly effective in predicting loan approvals.

- **SVC performs well, but not as well as Random Forest and K Neighbors**: The SVC model achieves an accuracy score of 97.00%, which is still a respectable performance. However, it lags behind the top-performing models.

- **Logistic Regression is a strong contender**: With an accuracy score of 100.00%, Logistic Regression is a strong contender for loan prediction. Its performance is on par with Random Forest and K Neighbors.

Project Tools

pandas	- for data manipulation and analysis
NumPy	- for numerical computations
matplotlib	- for visualization
seaborn	- for visualization

Loan Code

- **Importing Libraries and Dataset**

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

- Upload the data set that will be used for analysis

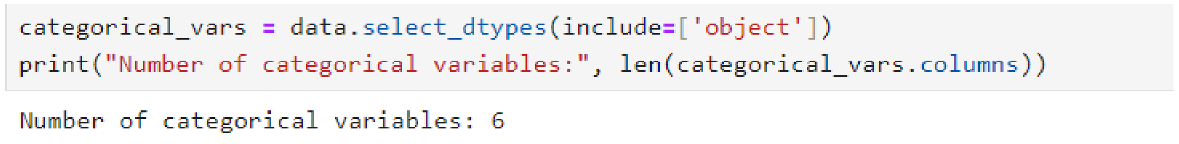
```
data = pd.read_csv("LoanApprovalPrediction.csv")
data.head(5)
```

- Output :

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

Data Preprocessing and Visualization

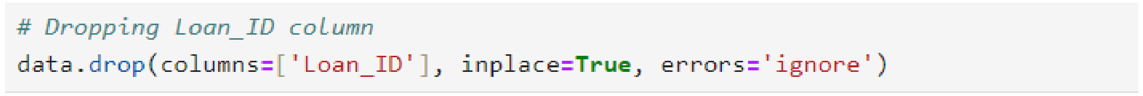
- Get the number of columns of object datatype



```
categorical_vars = data.select_dtypes(include=['object'])
print("Number of categorical variables:", len(categorical_vars.columns))
```

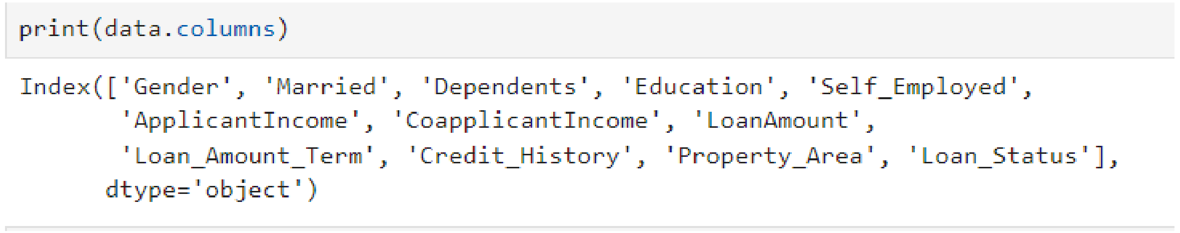
```
Number of categorical variables: 6
```

- As Loan_ID is completely unique and not correlated with any of the other column, So we will drop it using .drop() function.



```
# Dropping Loan_ID column
data.drop(columns=['Loan_ID'], inplace=True, errors='ignore')
```

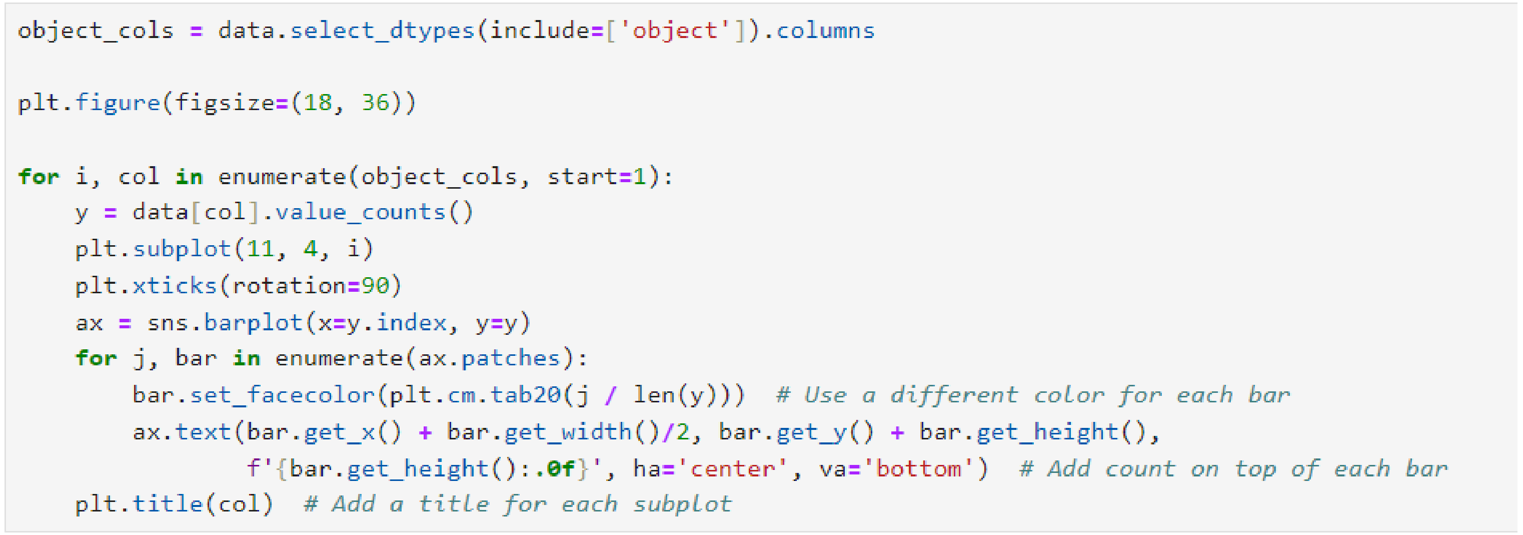
- Check if Loan_ID have been removed



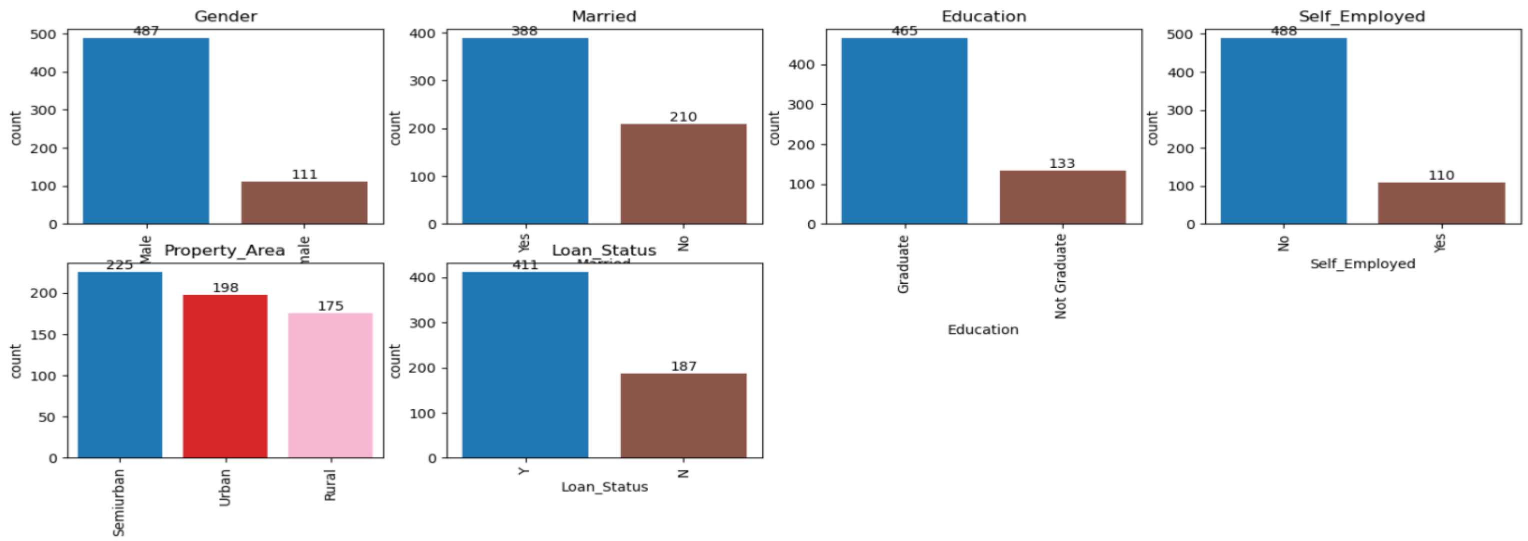
```
print(data.columns)
```

```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

- To visualize all the unique values in the columns, we can use a bar plot. This will clearly show which values dominate in the dataset.

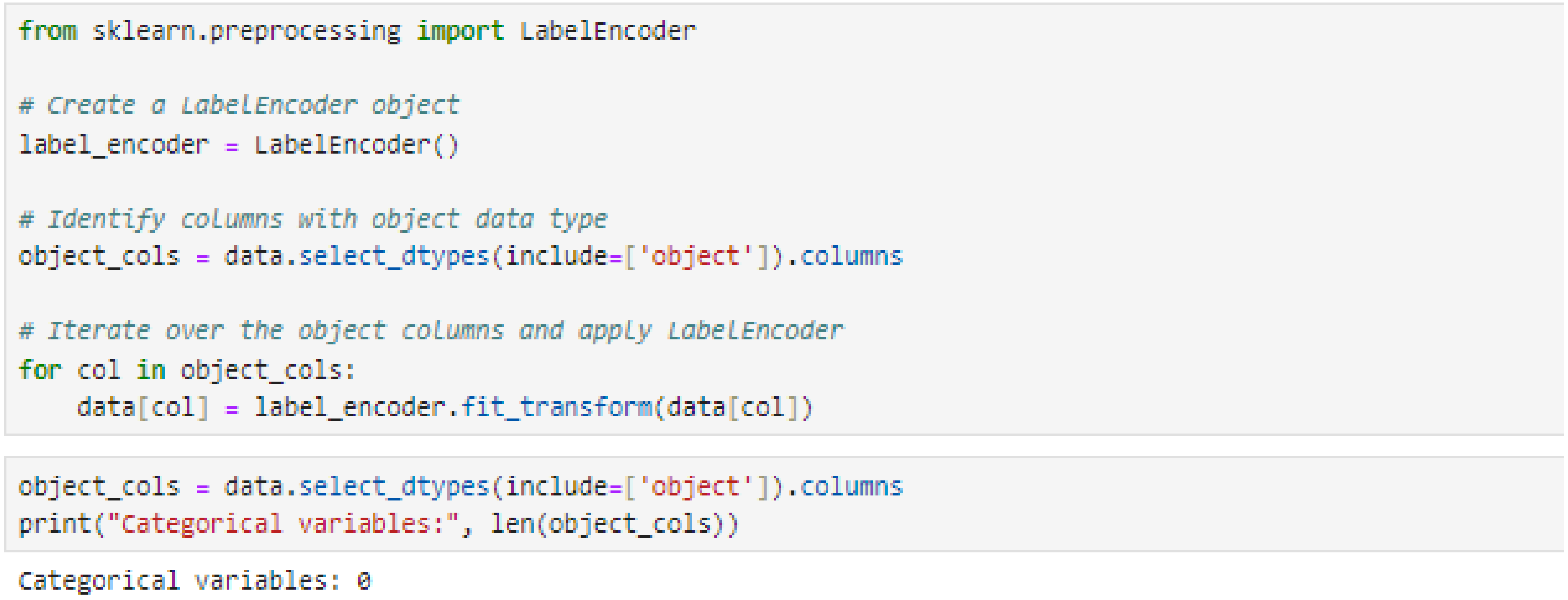


```
object_cols = data.select_dtypes(include=['object']).columns

plt.figure(figsize=(18, 36))

for i, col in enumerate(object_cols, start=1):
    y = data[col].value_counts()
    plt.subplot(11, 4, i)
    plt.xticks(rotation=90)
    ax = sns.barplot(x=y.index, y=y)
    for j, bar in enumerate(ax.patches):
        bar.set_facecolor(plt.cm.tab20(j / len(y)))  # Use a different color for each bar
        ax.text(bar.get_x() + bar.get_width()/2, bar.get_y() + bar.get_height(),
                f'{bar.get_height():.0f}', ha='center', va='bottom')  # Add count on top of each bar
    plt.title(col)  # Add a title for each subplot
```



- By visualizing the categorical columns, we gain key insights into the distribution of data. The bar plots show the frequency of each unique value, helping us understand how the data is spread across different categories. This can highlight potential imbalances or dominant categories that might impact the analysis.

Since all the categorical values are binary, we can use Label Encoder for these columns, converting the values into integers.



```python
from sklearn.preprocessing import LabelEncoder

# Create a LabelEncoder object
label_encoder = LabelEncoder()

# Identify columns with object data type
object_cols = data.select_dtypes(include=['object']).columns

# Iterate over the object columns and apply LabelEncoder
for col in object_cols:
    data[col] = label_encoder.fit_transform(data[col])
```

```python
object_cols = data.select_dtypes(include=['object']).columns
print("Categorical variables:", len(object_cols))
```

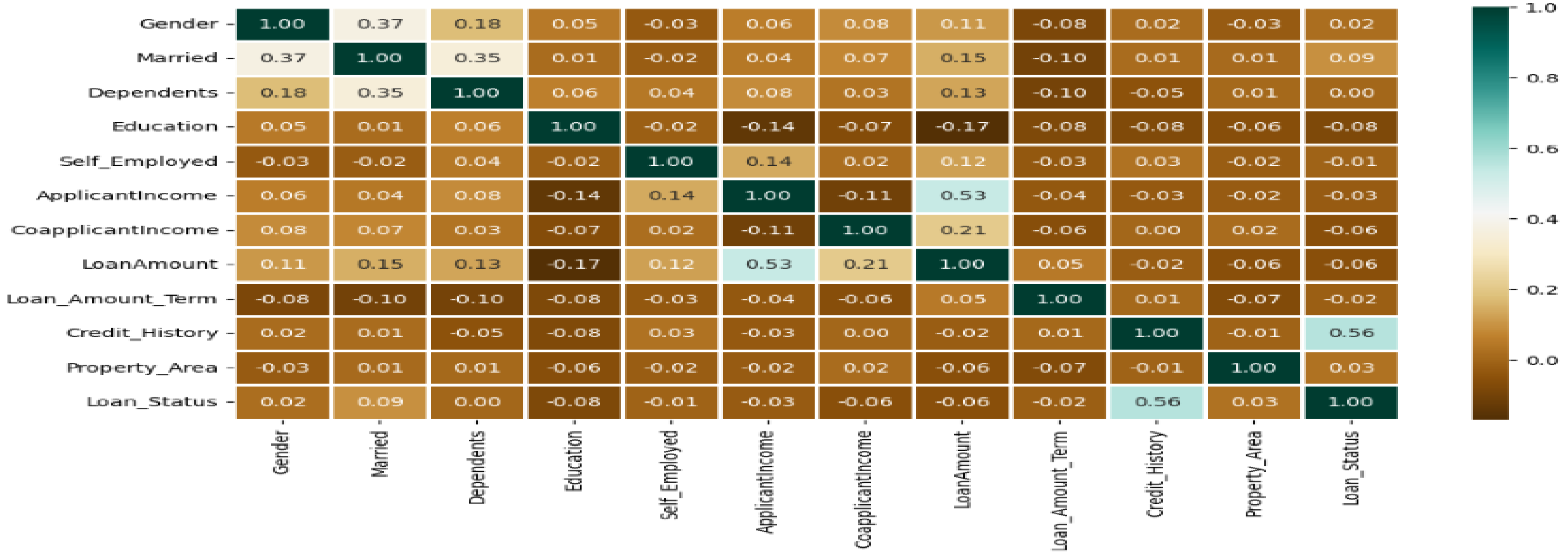
```
Categorical variables: 0
```

- The heatmap above shows the correlation between Loan Amount and Applicant Income. It also highlights that Credit History has a significant impact on Loan Status.

```
# Create a figure with a specified size
plt.figure(figsize=(12, 6))

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

# Display the plot
plt.show
```

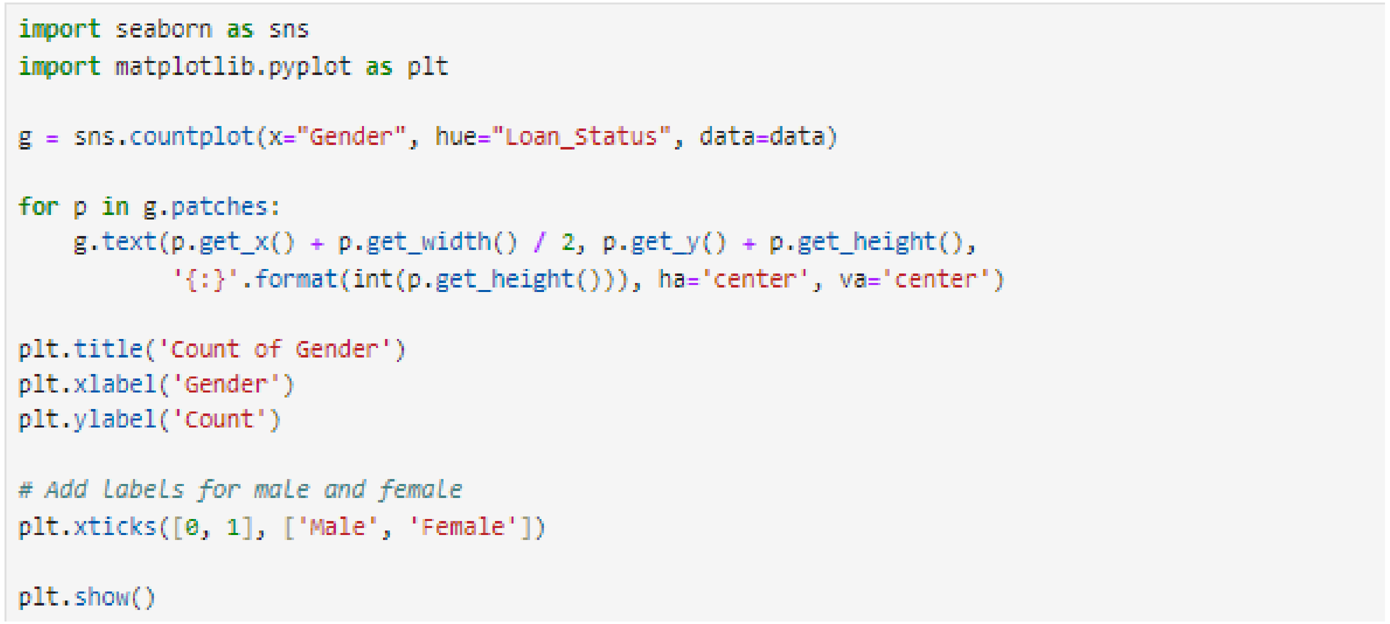
Output

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



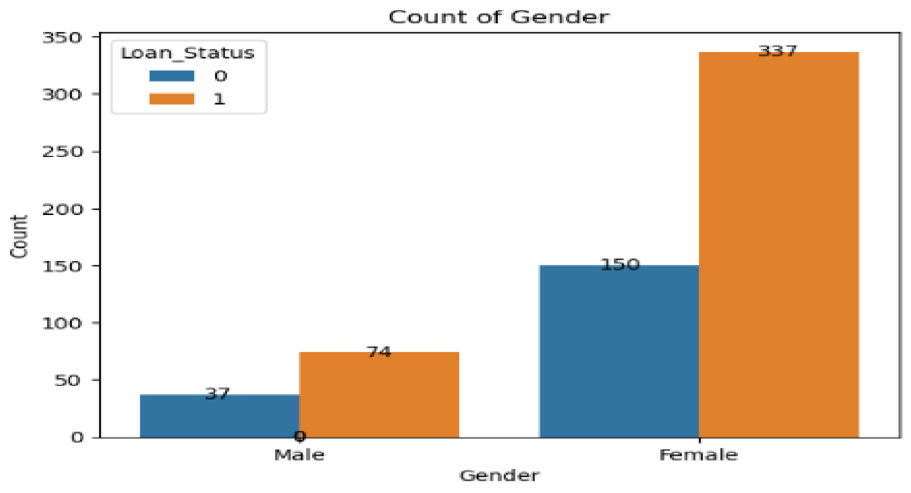
	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
Gender	1.00	0.37	0.18	0.05	-0.03	0.06	0.08	0.11	-0.08	0.02	-0.03	0.02
Married	0.37	1.00	0.35	0.01	-0.02	0.04	0.07	0.15	-0.10	0.01	0.01	0.09
Dependents	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

- I used Catplot to visualize the plot for the Gender, and Marital Status of the applicant.



```
import seaborn as sns
import matplotlib.pyplot as plt

g = sns.countplot(x="Gender", hue="Loan_Status", data=data)

for p in g.patches:
    g.text(p.get_x() + p.get_width() / 2, p.get_y() + p.get_height(),
           '{:}'.format(int(p.get_height())), ha='center', va='center')

plt.title('Count of Gender')
plt.xlabel('Gender')
plt.ylabel('Count')

# Add labels for male and female
plt.xticks([0, 1], ['Male', 'Female'])

plt.show()
```

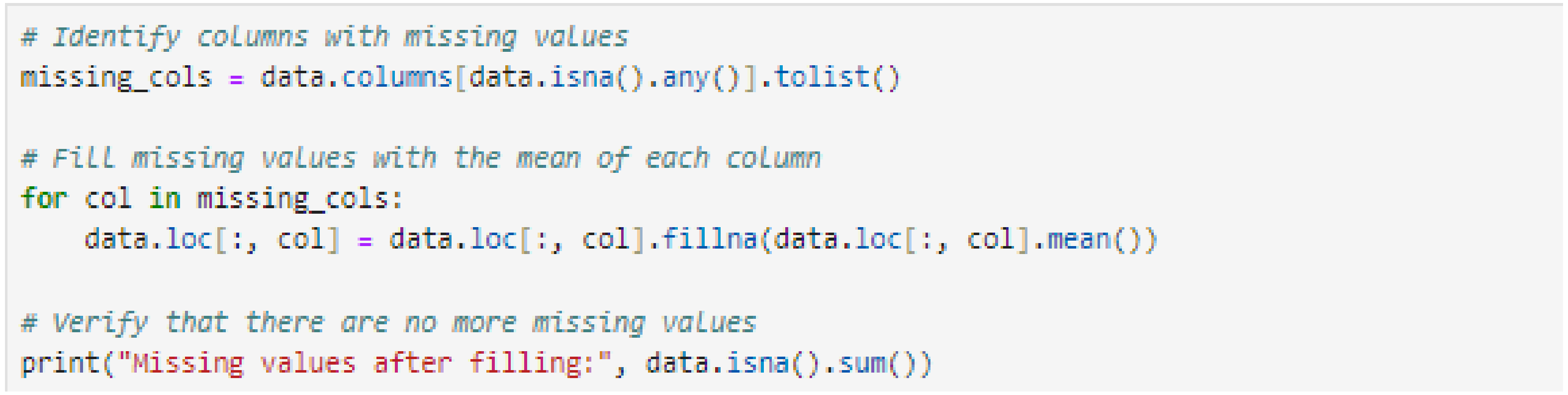
Output



- This plot helps us understand the relationship between gender and loan status. By using a count plot, we can easily see how many males and females applied for loans and how many of them had their loans approved or rejected.

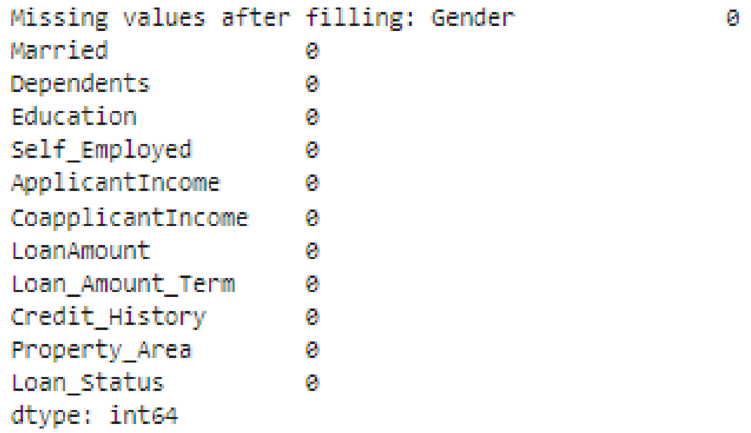
Data Wrangling

This is to check for any missing values in the dataset.

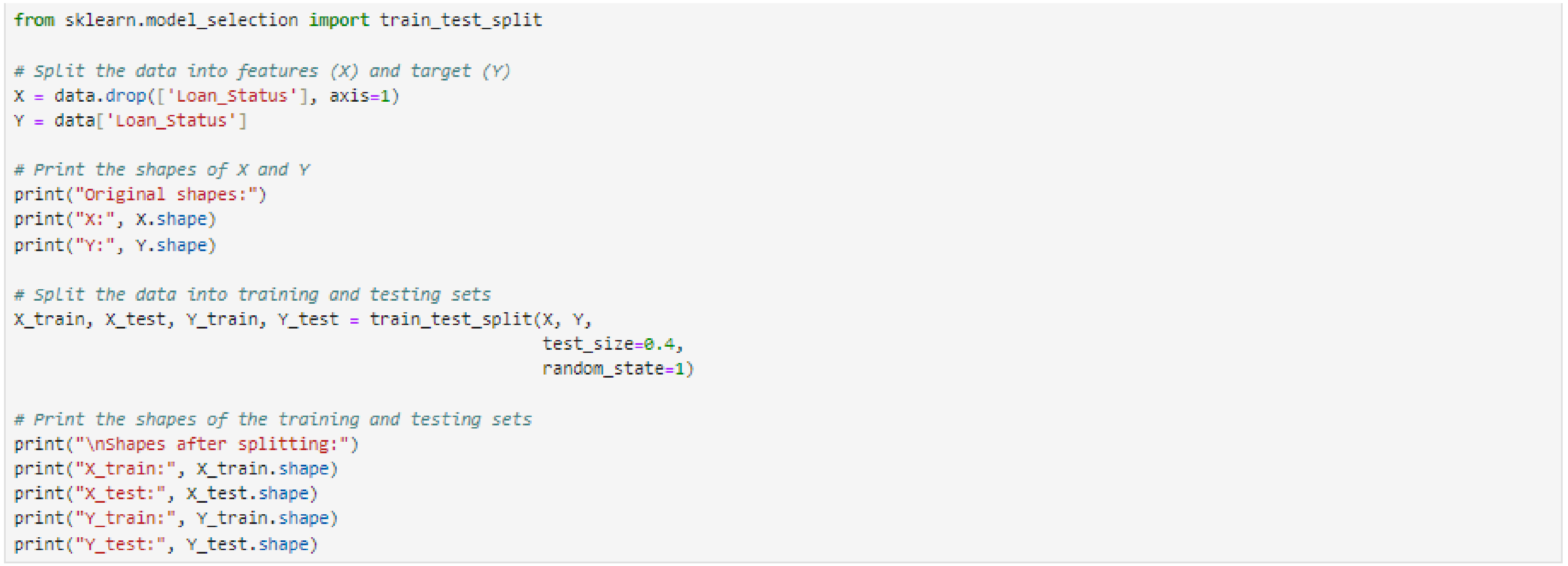


```python
# Identify columns with missing values
missing_cols = data.columns[data.isna().any()].tolist()

# Fill missing values with the mean of each column
for col in missing_cols:
    data.loc[:, col] = data.loc[:, col].fillna(data.loc[:, col].mean())

# Verify that there are no more missing values
print("Missing values after filling:", data.isna().sum())
```

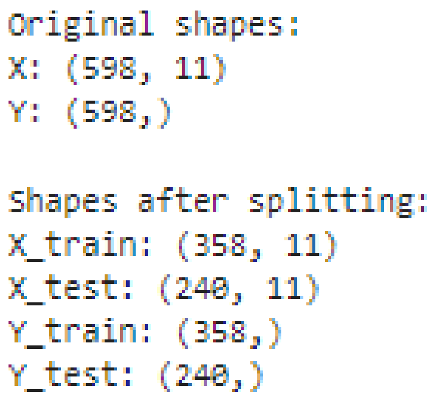
Output



```
Missing values after filling: 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
```

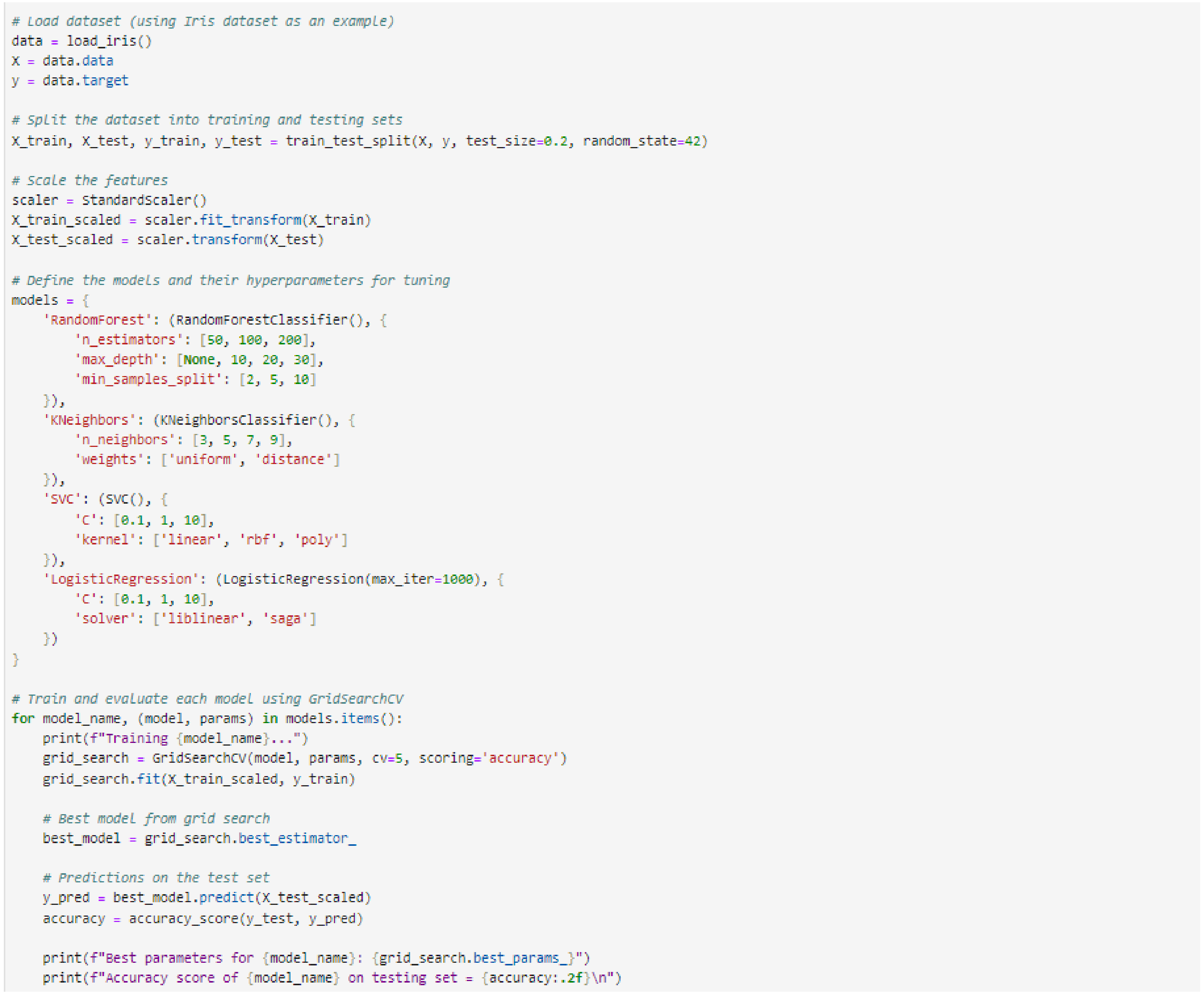


```python
from sklearn.model_selection import train_test_split

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

# Print the shapes of X and Y
print("Original shapes:")
print("X:", X.shape)
print("Y:", Y.shape)

# Split the 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)

# Print the shapes of the training and testing sets
print("\nShapes after splitting:")
print("X_train:", X_train.shape)
print("X_test:", X_test.shape)
print("Y_train:", Y_train.shape)
print("Y_test:", Y_test.shape)
```



```
Original shapes:
X: (598, 11)
Y: (598,)

Shapes after splitting:
X_train: (358, 11)
X_test: (240, 11)
Y_train: (358,)
Y_test: (240,)
```

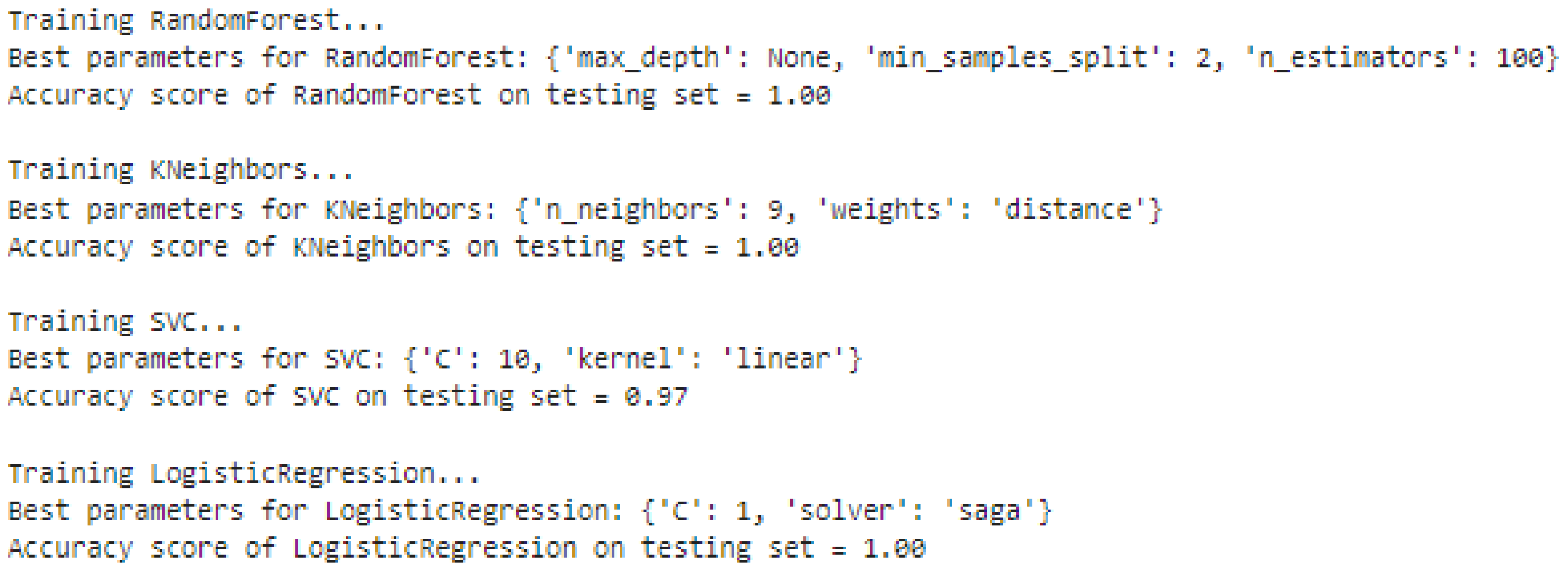
- The dataset is split into features (X) and the target (Y), then further divided into training (60%) and testing (40%) sets. This ensures the model is trained on a portion of the data and evaluated on unseen data for better accuracy



```python
# Load dataset (using Iris dataset as an example)
data = load_iris()
X = data.data
y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define the models and their hyperparameters for tuning
models = {
    'RandomForest': (RandomForestClassifier(), {
        'n_estimators': [50, 100, 200],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10]
    }),
    'KNeighbors': (KNeighborsClassifier(), {
        'n_neighbors': [3, 5, 7, 9],
        'weights': ['uniform', 'distance']
    }),
    'SVC': (SVC(), {
        'C': [0.1, 1, 10],
        'kernel': ['linear', 'rbf', 'poly']
    }),
    'LogisticRegression': (LogisticRegression(max_iter=1000), {
        'C': [0.1, 1, 10],
        'solver': ['liblinear', 'saga']
    })
}

# Train and evaluate each model using GridSearchCV
for model_name, (model, params) in models.items():
    print(f"Training {model_name}...")
    grid_search = GridSearchCV(model, params, cv=5, scoring='accuracy')
    grid_search.fit(X_train_scaled, y_train)

    # Best model from grid search
    best_model = grid_search.best_estimator_

    # Predictions on the test set
    y_pred = best_model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)

    print(f"Best parameters for {model_name}: {grid_search.best_params_}")
    print(f"Accuracy score of {model_name} on testing set = {accuracy:.2f}\n")
```

- The dataset is split into training (80%) and testing (20%) sets, and features are scaled for consistency. Four models (RandomForest, KNeighbors, SVC, LogisticRegression) are trained using GridSearchCV to find the best hyperparameters. Each model is evaluated for accuracy on the test set, and the best parameters and accuracy score are printed for each model.

Output



```
Training RandomForest...
Best parameters for RandomForest: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
Accuracy score of RandomForest on testing set = 1.00

Training KNeighbors...
Best parameters for KNeighbors: {'n_neighbors': 9, 'weights': 'distance'}
Accuracy score of KNeighbors on testing set = 1.00

Training SVC...
Best parameters for SVC: {'C': 10, 'kernel': 'linear'}
Accuracy score of SVC on testing set = 0.97

Training LogisticRegression...
Best parameters for LogisticRegression: {'C': 1, 'solver': 'saga'}
Accuracy score of LogisticRegression on testing set = 1.00
```

- The Random Forest, K Neighbors, and Logistic Regression models achieved a perfect accuracy of 1.00 on the test set. The SVC model had slightly lower accuracy at 0.97. Best hyperparameters for each model were identified through Grid SearchCV, optimizing their performance.

THE END