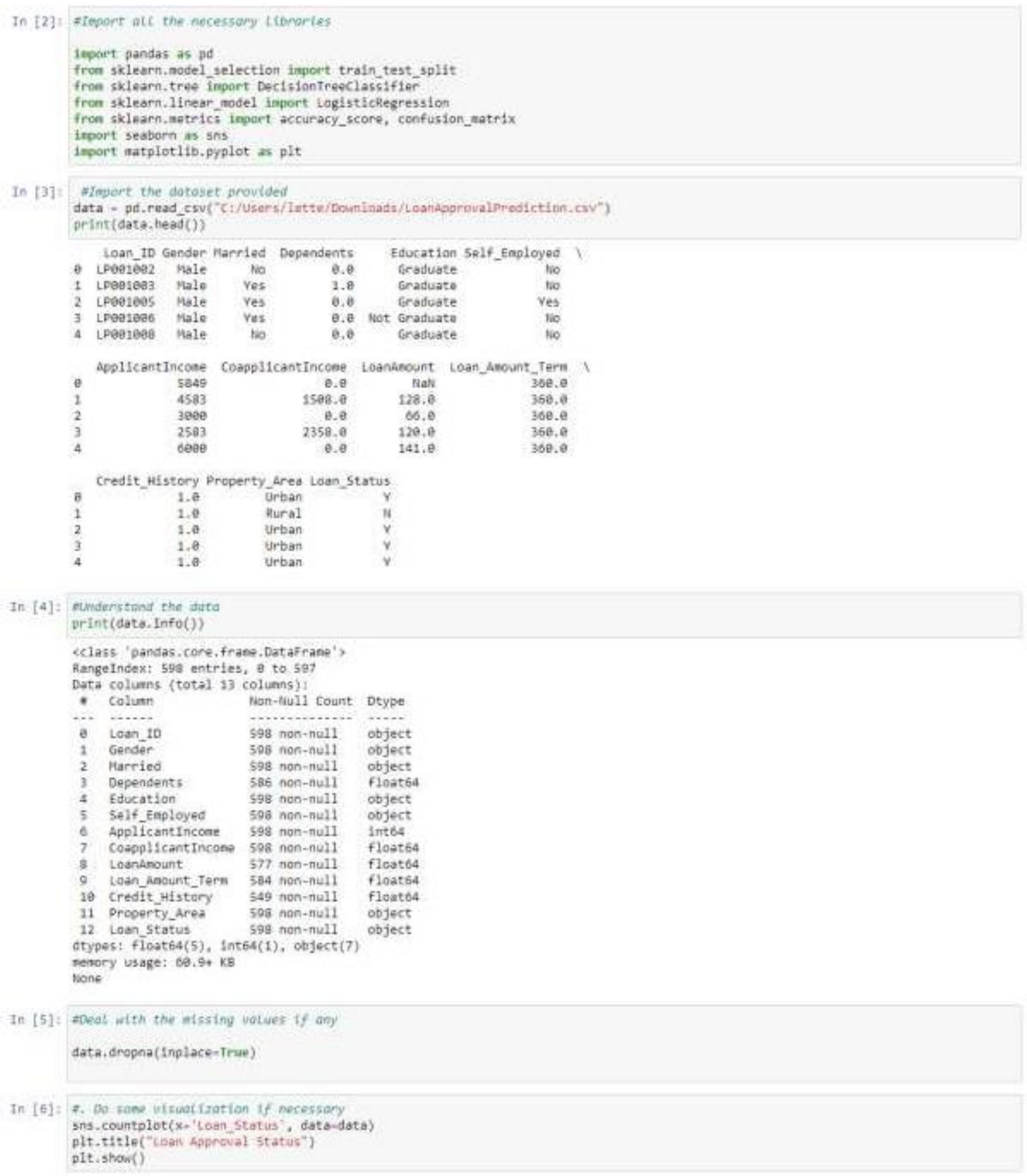
CREATING ML MODELS FOR PREDICTION



```
In [2]: #Import all the necessary libraries

        import pandas as pd
        from sklearn.model_selection import train_test_split
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import accuracy_score, confusion_matrix
        import seaborn as sns
        import matplotlib.pyplot as plt
```

```
In [3]:  #Import the dataset provided
        data = pd.read_csv("C:/Users/lette/Downloads/LoanApprovalPrediction.csv")
        print(data.head())
```

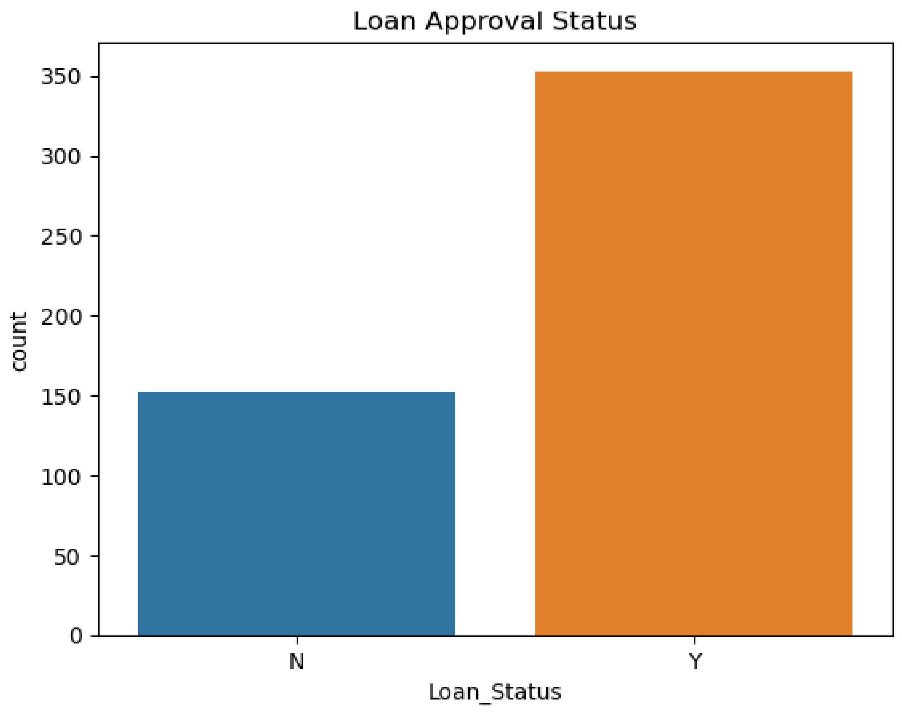
```
    Loan_ID Gender Married  Dependents     Education Self_Employed  \
0  LP001002   Male      No         0.0      Graduate            No
1  LP001003   Male     Yes         1.0      Graduate            No
2  LP001005   Male     Yes         0.0      Graduate           Yes
3  LP001006   Male     Yes         0.0  Not Graduate            No
4  LP001008   Male      No         0.0      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
```

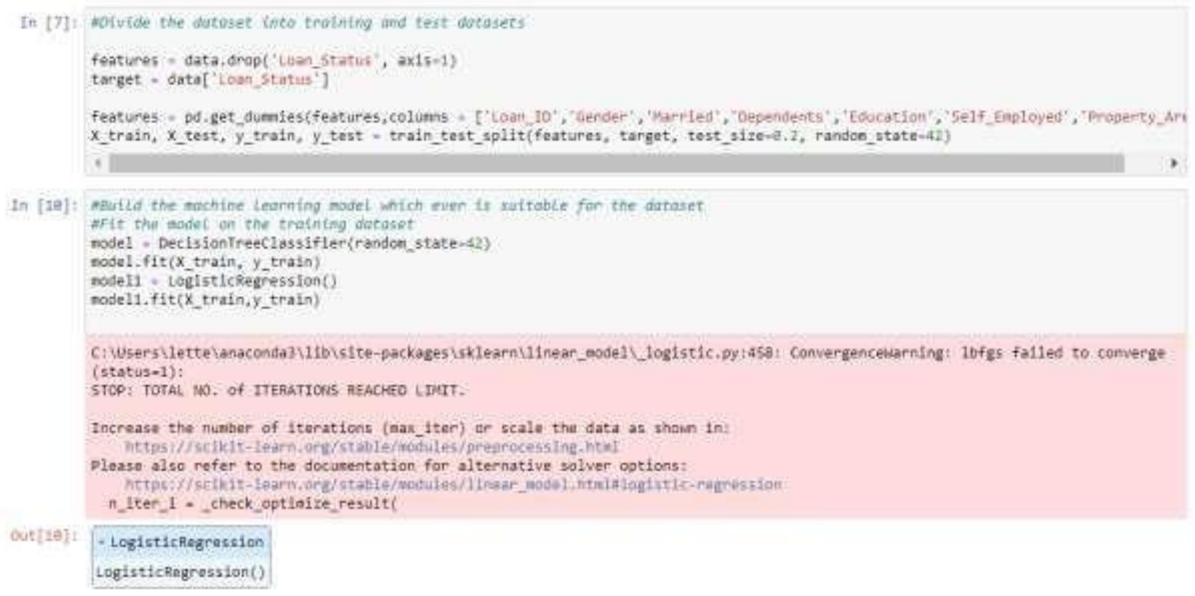
```
In [4]: #Understand the data
        print(data.Info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 598 entries, 0 to 597
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            598 non-null    object
 1   Gender             598 non-null    object
 2   Married            598 non-null    object
 3   Dependents         586 non-null    float64
 4   Education          598 non-null    object
 5   Self_Employed      598 non-null    object
 6   ApplicantIncome    598 non-null    int64
 7   CoapplicantIncome  598 non-null    float64
 8   LoanAmount         577 non-null    float64
 9   Loan_Amount_Term   584 non-null    float64
 10  Credit_History     549 non-null    float64
 11  Property_Area      598 non-null    object
 12  Loan_Status        598 non-null    object
dtypes: float64(5), int64(1), object(7)
memory usage: 60.9+ KB
None
```

```
In [5]: #Deal with the missing values if any

        data.dropna(inplace=True)
```

```
In [6]: #. Do some visualization if necessary
        sns.countplot(x='Loan_Status', data=data)
        plt.title("Loan Approval Status")
        plt.show()
```

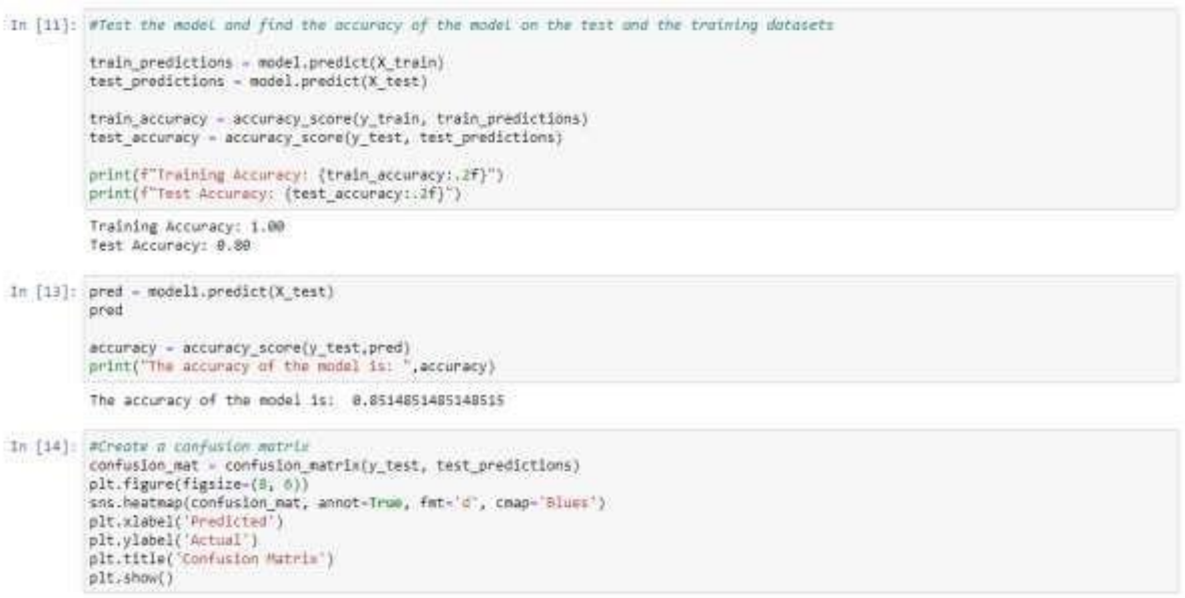



```
In [7]: #Divide the dataset into training and test datasets

        features = data.drop('Loan_Status', axis=1)
        target = data['Loan_Status']

        features = pd.get_dummies(features,columns = ['Loan_ID','Gender','Married','Dependents','Education','Self_Employed','Property_Are
        X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
```

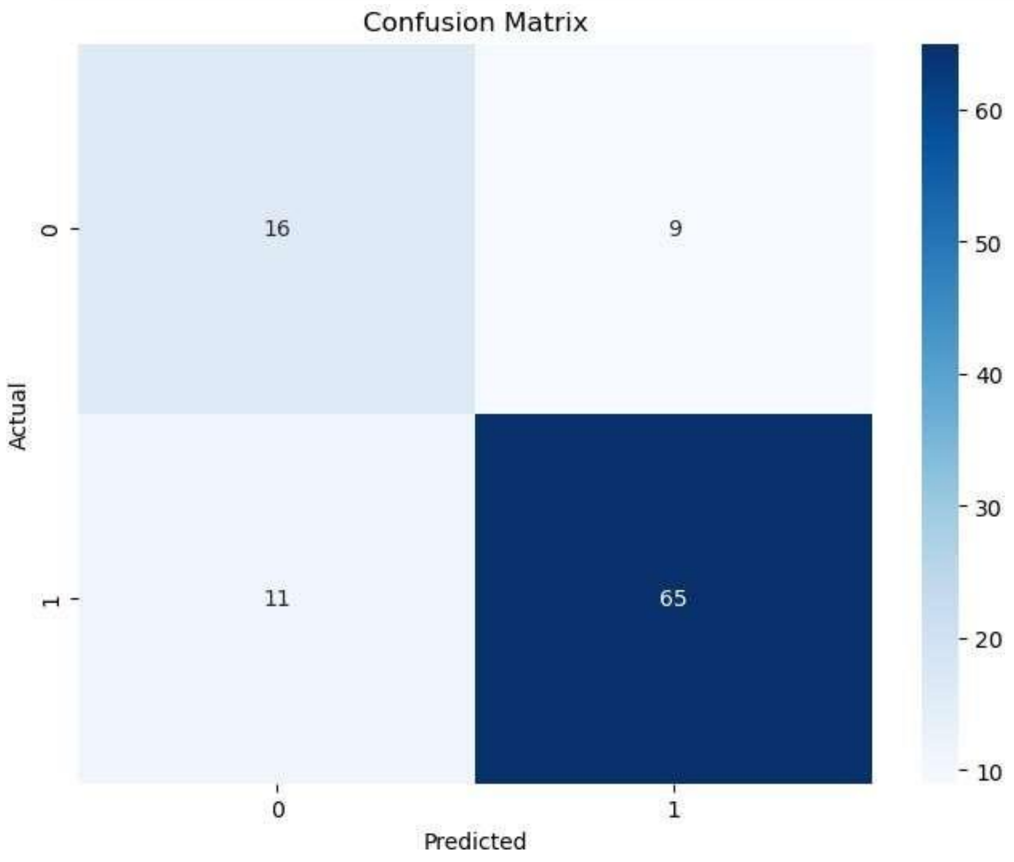
```
In [10]: #Build the machine learning model which ever is suitable for the dataset
         #Fit the model on the training dataset
         model = DecisionTreeClassifier(random_state=42)
         model.fit(X_train, y_train)
         model1 = LogisticRegression()
         model1.fit(X_train,y_train)
```

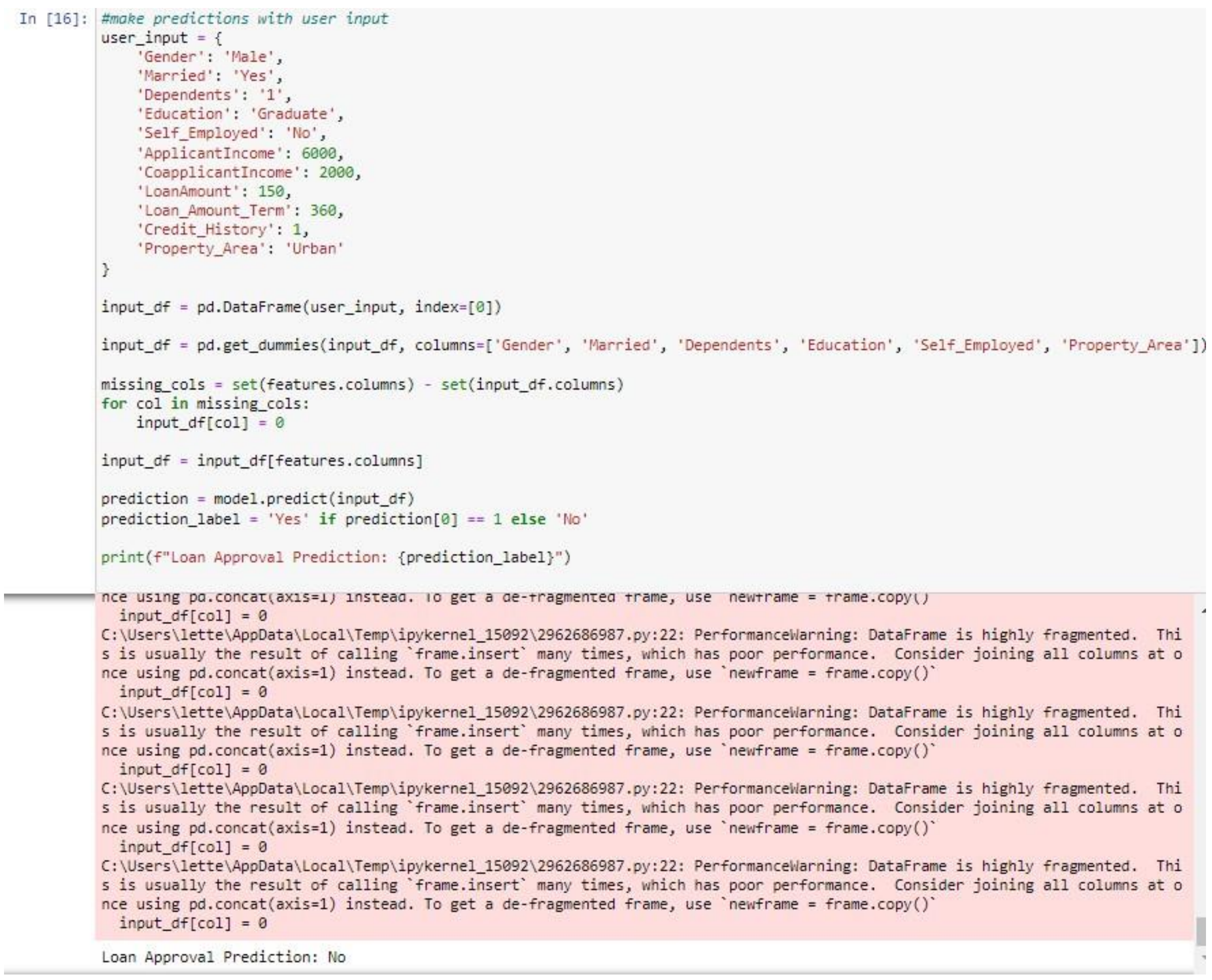
```
C:\Users\lette\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Out[10]: ▾ LogisticRegression
         LogisticRegression()
```



```
In [11]: #Test the model and find the accuracy of the model on the test and the training datasets

         train_predictions = model.predict(X_train)
         test_predictions = model.predict(X_test)

         train_accuracy = accuracy_score(y_train, train_predictions)
         test_accuracy = accuracy_score(y_test, test_predictions)

         print(f"Training Accuracy: {train_accuracy:.2f}")
         print(f"Test Accuracy: {test_accuracy:.2f}")

         Training Accuracy: 1.00
         Test Accuracy: 0.80

In [13]: pred = model1.predict(X_test)
         pred

         accuracy = accuracy_score(y_test,pred)
         print("The accuracy of the model is: ",accuracy)

         The accuracy of the model is:  0.8514851485148515

In [14]: #Create a confusion matrix
         confusion_mat = confusion_matrix(y_test, test_predictions)
         plt.figure(figsize=(8, 6))
         sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues')
         plt.xlabel('Predicted')
         plt.ylabel('Actual')
         plt.title('Confusion Matrix')
         plt.show()
```





```
In [16]: #make predictions with user input
         user_input = {
             'Gender': 'Male',
             'Married': 'Yes',
             'Dependents': '1',
             'Education': 'Graduate',
             'Self_Employed': 'No',
             'ApplicantIncome': 6000,
             'CoapplicantIncome': 2000,
             'LoanAmount': 150,
             'Loan_Amount_Term': 360,
             'Credit_History': 1,
             'Property_Area': 'Urban'
         }

         input_df = pd.DataFrame(user_input, index=[0])

         input_df = pd.get_dummies(input_df, columns=['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Property_Area'])

         missing_cols = set(features.columns) - set(input_df.columns)
         for col in missing_cols:
             input_df[col] = 0

         input_df = input_df[features.columns]

         prediction = model.predict(input_df)
         prediction_label = 'Yes' if prediction[0] == 1 else 'No'

         print(f"Loan Approval Prediction: {prediction_label}")
```

```
nce using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
  input_df[col] = 0
C:\Users\lette\AppData\Local\Temp\ipykernel_15092\2962686987.py:22: PerformanceWarning: DataFrame is highly fragmented.  Thi
s is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at o
nce using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
  input_df[col] = 0
C:\Users\lette\AppData\Local\Temp\ipykernel_15092\2962686987.py:22: PerformanceWarning: DataFrame is highly fragmented.  Thi
s is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at o
nce using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
  input_df[col] = 0
C:\Users\lette\AppData\Local\Temp\ipykernel_15092\2962686987.py:22: PerformanceWarning: DataFrame is highly fragmented.  Thi
s is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at o
nce using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
  input_df[col] = 0
C:\Users\lette\AppData\Local\Temp\ipykernel_15092\2962686987.py:22: PerformanceWarning: DataFrame is highly fragmented.  Thi
s is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at o
nce using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
  input_df[col] = 0

Loan Approval Prediction: No
```