ML Data Cleaning and Feature Selection Assignment
Ronak Shah – 002813456

This dataset holds information about loan applicants. Each row seems to represent a unique applicant, and the columns contain various attributes related to the applicants' loan applications. The columns include:

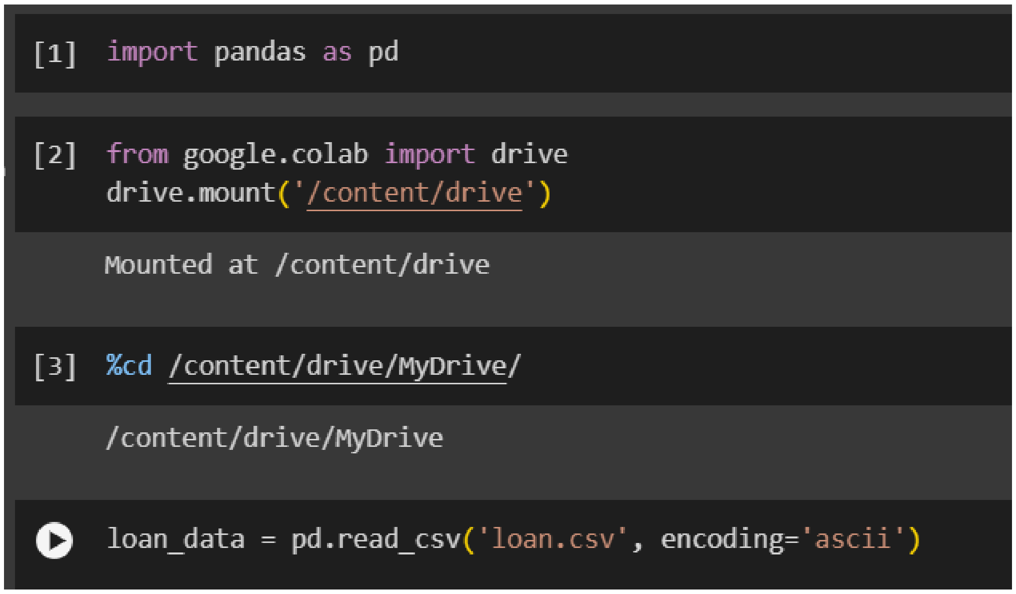
1. Loan_ID
2. Gender
3. Married
4. Dependents
5. Education
6. Self_Employed
7. ApplicantIncome
8. CoapplicantIncome
9. LoanAmount
10. Loan_Amount_Term
11. Credit_History
12. Property_Area
13. Loan_Status

Here's a breakdown of what each column represents:

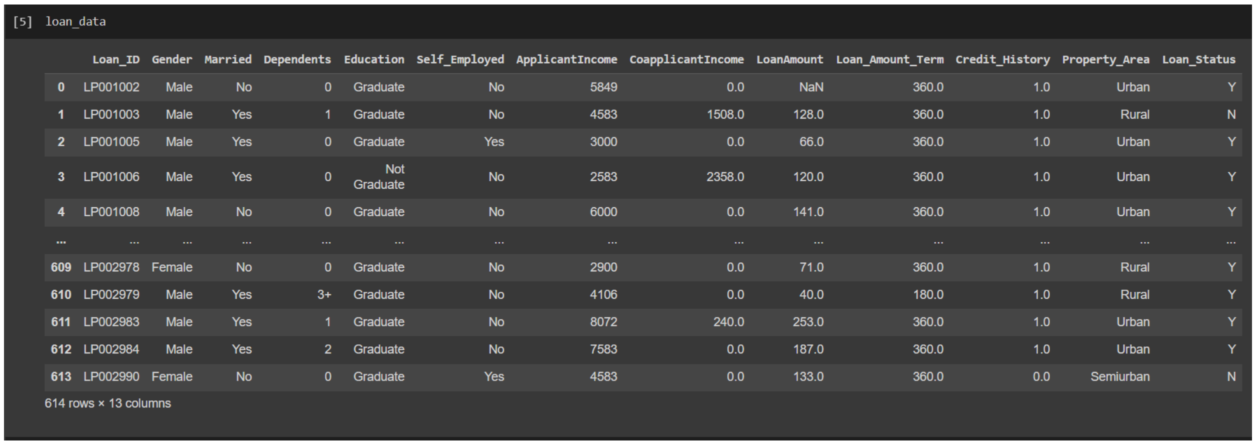
1. **Loan_ID**: Unique identifier for each loan application.
2. **Gender**: Gender of the applicant.
3. **Married**: Marital status of the applicant.
4. **Dependents**: Number of dependents the applicant has.
5. **Education**: Education level of the applicant (Graduate or Not Graduate).
6. **Self_Employed**: Whether the applicant is self-employed or not.
7. **ApplicantIncome**: Income of the applicant.
8. **CoapplicantIncome**: Income of the co-applicant (if any).
9. **LoanAmount**: Amount of the loan applied for.
10. **Loan_Amount_Term**: Term of the loan in months.

11. **Credit_History**: Credit history of the applicant (1 indicates a good credit history, 0 indicates a poor credit history).
12. **Property_Area**: Area where the property is located (Urban, Rural, or Semiurban).
13. **Loan_Status**: Whether the loan was approved (Y) or not (N).

Let's start by loading the dataset and performing an initial examination to understand its structure.



```
[1]  import pandas as pd

[2]  from google.colab import drive
     drive.mount('/content/drive')

     Mounted at /content/drive

[3]  %cd /content/drive/MyDrive/

     /content/drive/MyDrive

     loan_data = pd.read_csv('loan.csv', encoding='ascii')
```

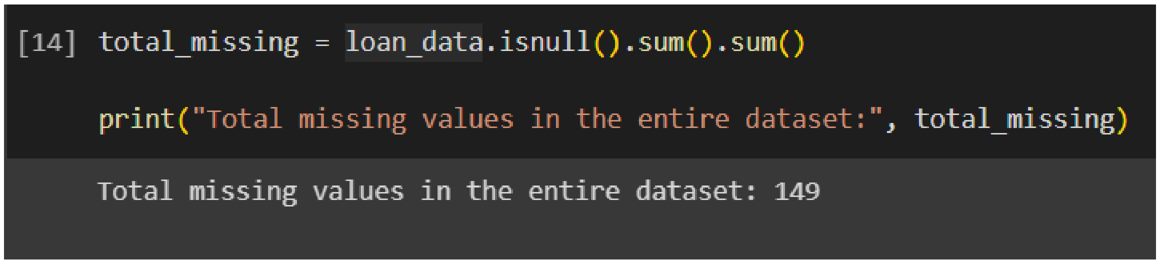


```
[5]  loan_data
```

	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	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y
...	...	...	...	...	...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0	Rural	Y
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0	Rural	Y
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0	Urban	Y
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0	Urban	Y
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semiurban	N

614 rows × 13 columns

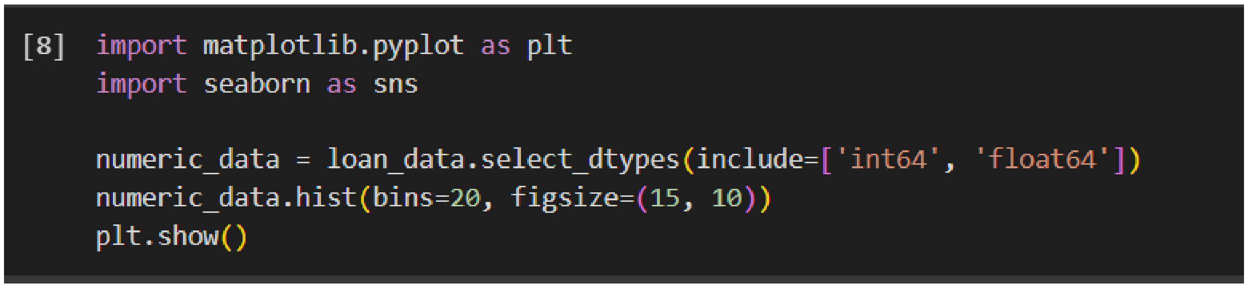
* What are the data types? (Only numeric and categorical)



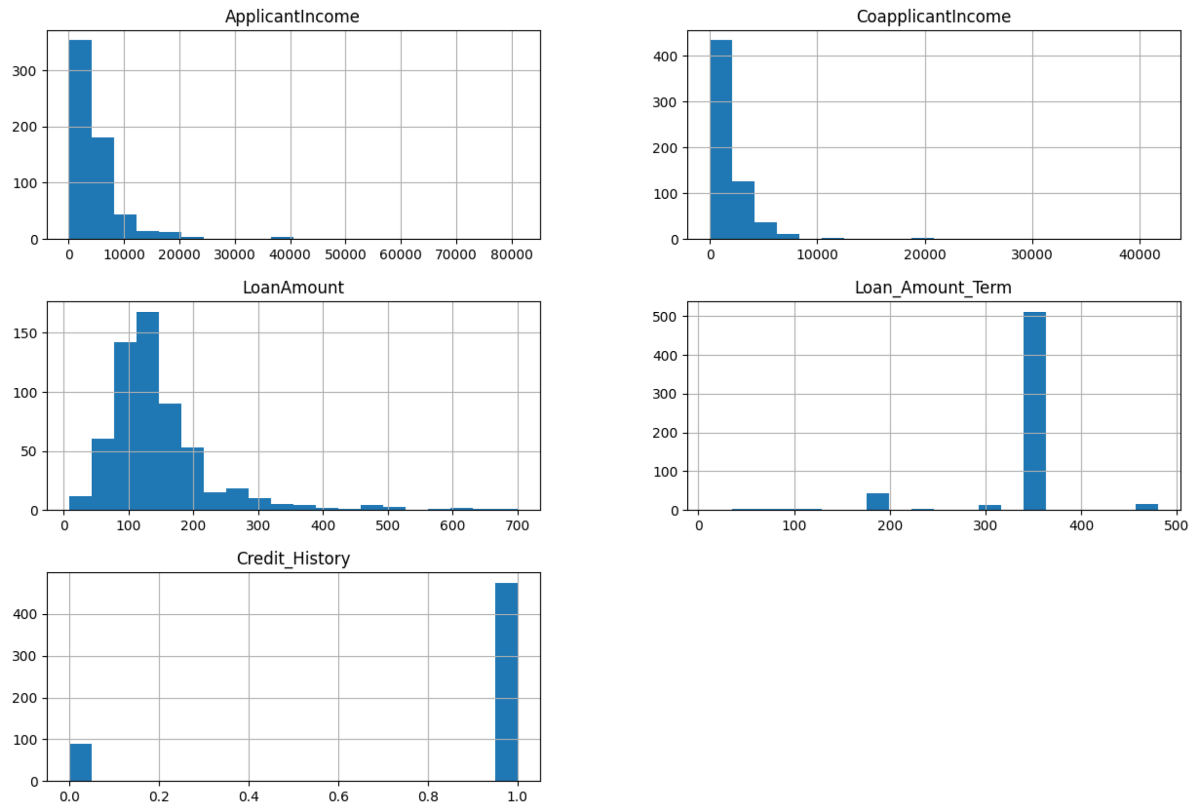
```
[14] total_missing = loan_data.isnull().sum().sum()

     print("Total missing values in the entire dataset:", total_missing)

     Total missing values in the entire dataset: 149
```

There a total of 149 missing values.

* What are the likely distributions of the numeric variables?

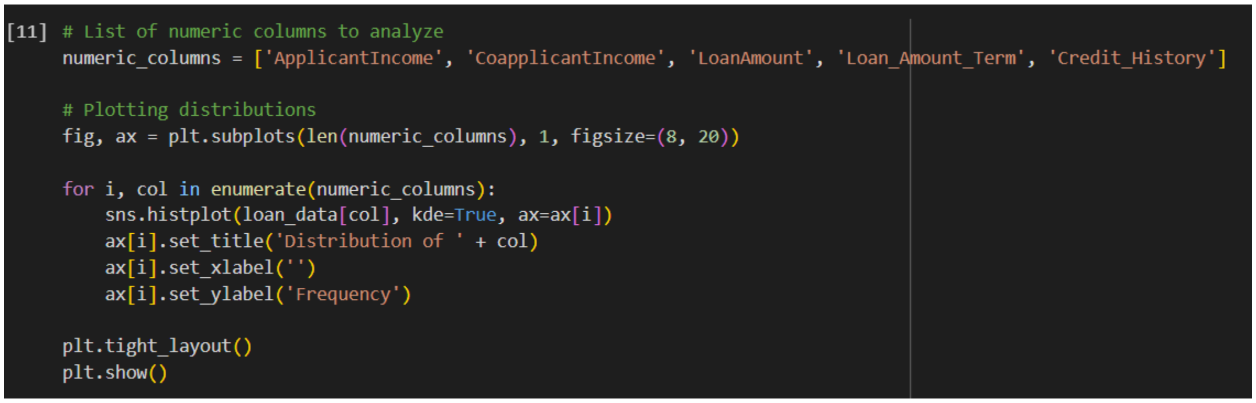


```
[8]  import matplotlib.pyplot as plt
     import seaborn as sns

     numeric_data = loan_data.select_dtypes(include=['int64', 'float64'])
     numeric_data.hist(bins=20, figsize=(15, 10))
     plt.show()
```

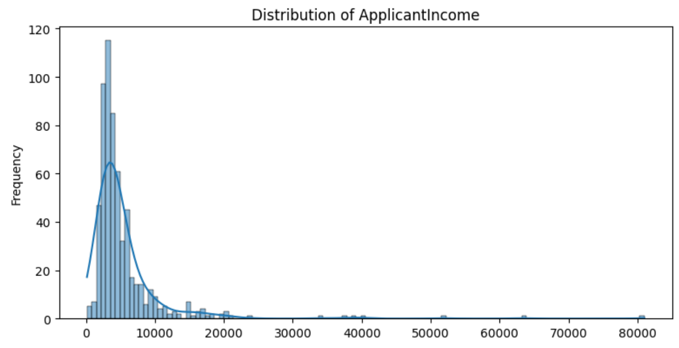


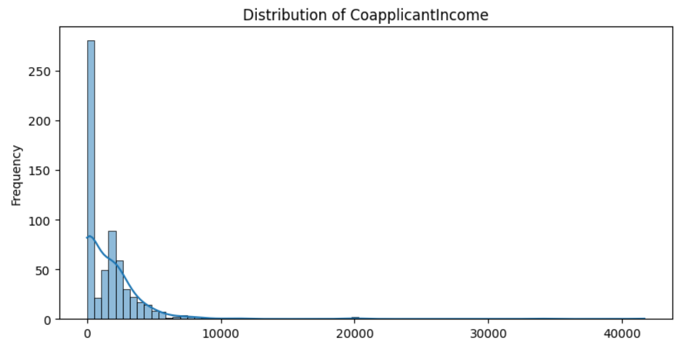
Based on the image, the likely distributions of the numeric variables are:

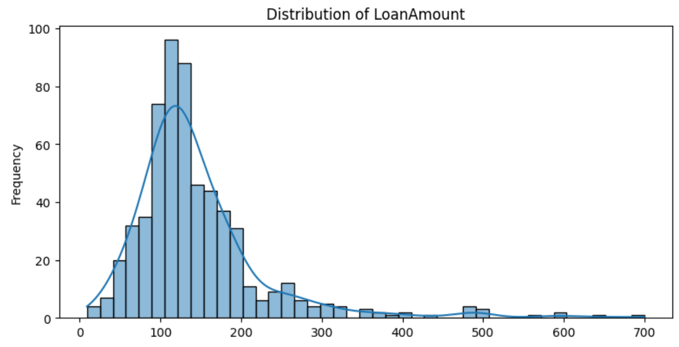
- Applicant income: left-skewed, with a mode around $20,000 and a tail extending up to $80,000.
- Coapplicant income: left-skewed, with a mode around $20,000 and a tail extending up to $70,000.
- Loan amount: right-skewed, with a mode around $200,000 and a tail extending up to $700,000.
- Loan amount term: right-skewed, with a mode around 200 months and a tail extending up to 500 months.
- Credit history: uniform, with values ranging from 0 to 1.

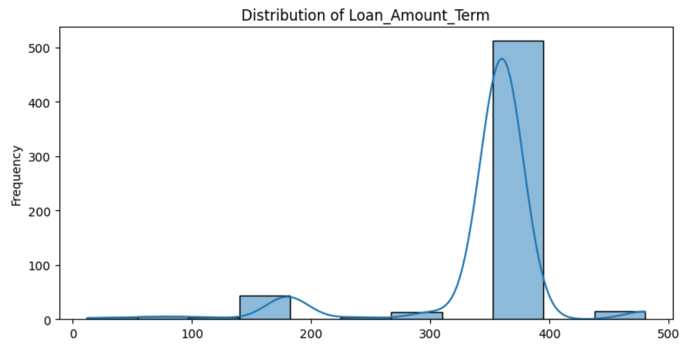


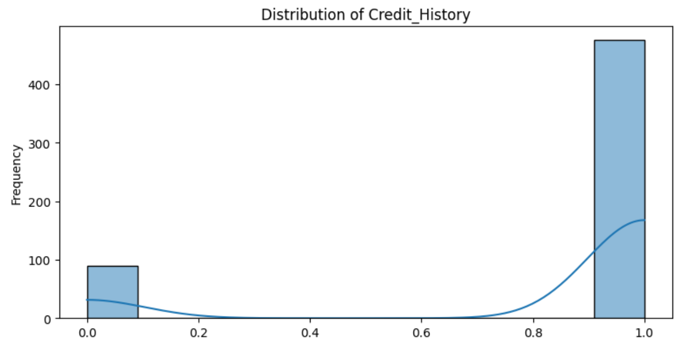
```
[11] # List of numeric columns to analyze
     numeric_columns = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_History']

     # Plotting distributions
     fig, ax = plt.subplots(len(numeric_columns), 1, figsize=(8, 20))

     for i, col in enumerate(numeric_columns):
         sns.histplot(loan_data[col], kde=True, ax=ax[i])
         ax[i].set_title('Distribution of ' + col)
         ax[i].set_xlabel('')
         ax[i].set_ylabel('Frequency')

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


Distribution of ApplicantIncome
Frequency
0
20
40
60
80
100
120
0
10000
20000
30000
40000
50000
60000
70000
80000


Distribution of CoapplicantIncome
Frequency
0
50
100
150
200
250
0
10000
20000
30000
40000


Distribution of LoanAmount
Frequency
0
20
40
60
80
100
0
100
200
300
400
500
600
700


Distribution of Loan_Amount_Term
Frequency
0
100
200
300
400
500
0
100
200
300
400
500

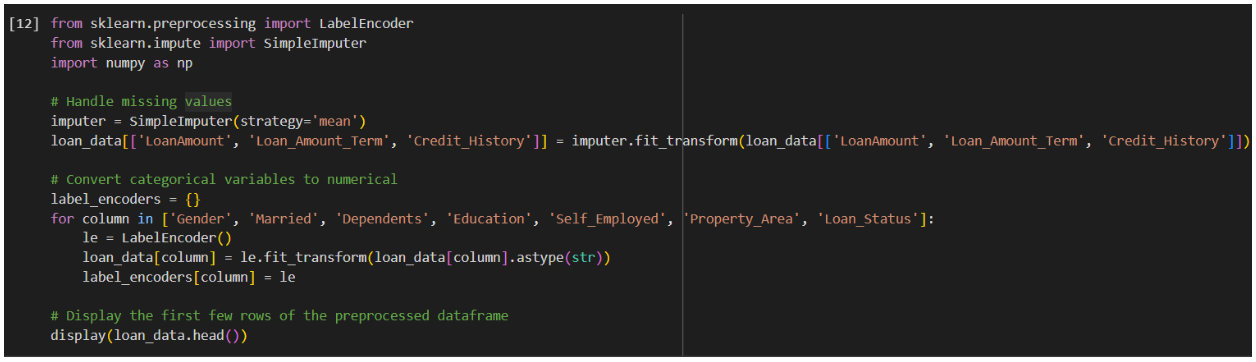

Distribution of Credit_History
Frequency
0
100
200
300
400
0.0
0.2
0.4
0.6
0.8
1.0

Based on the image, here are the likely distributions of the numeric variables in the image you sent:

- Applicant income: left-skewed, with a mode around $20,000 and a tail extending up to $80,000.
- Coapplicant income: left-skewed, with a mode around $20,000 and a tail extending up to $70,000.
- Loan amount: right-skewed, with a mode around $200,000 and a tail extending up to $700,000.
- Loan amount term: right-skewed, with a mode around 200 months and a tail extending up to 500 months.
- Credit history: uniform, with values ranging from 0 to 1.

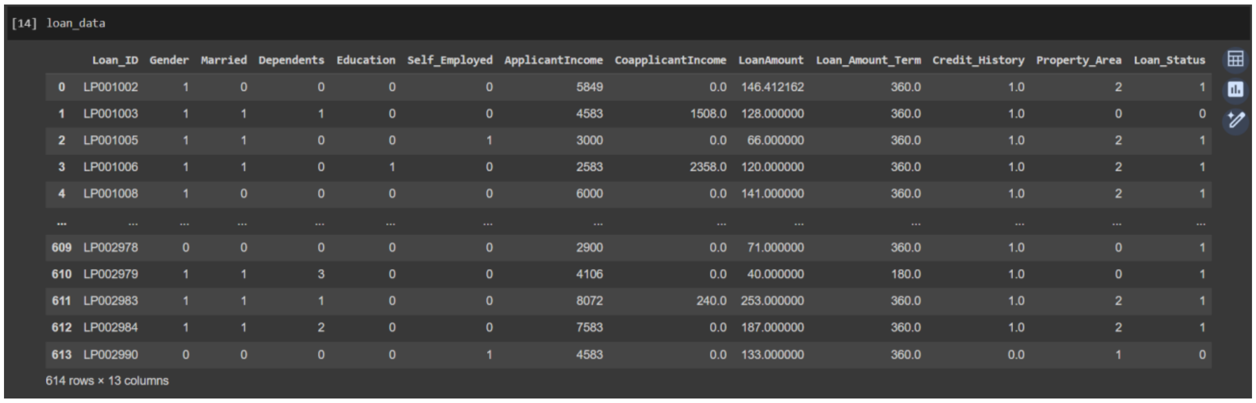
*** Which independent variables are useful to predict a target (dependent variable)? (Use at least three methods)**

The dataset contains a mix of numerical and categorical variables. Before proceeding with the feature selection methods, we'll need to preprocess the data. This includes handling missing values, encoding categorical variables, and potentially normalizing or standardizing numerical variables. Let's start with preprocessing.



```
[12] from sklearn.preprocessing import LabelEncoder
     from sklearn.impute import SimpleImputer
     import numpy as np

     # Handle missing values
     imputer = SimpleImputer(strategy='mean')
     loan_data[['LoanAmount', 'Loan_Amount_Term', 'Credit_History']] = imputer.fit_transform(loan_data[['LoanAmount', 'Loan_Amount_Term', 'Credit_History']])

     # Convert categorical variables to numerical
     label_encoders = {}
     for column in ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Property_Area', 'Loan_Status']:
         le = LabelEncoder()
         loan_data[column] = le.fit_transform(loan_data[column].astype(str))
         label_encoders[column] = le

     # Display the first few rows of the preprocessed dataframe
     display(loan_data.head())
```

Here's a preview of the preprocessed dataset:

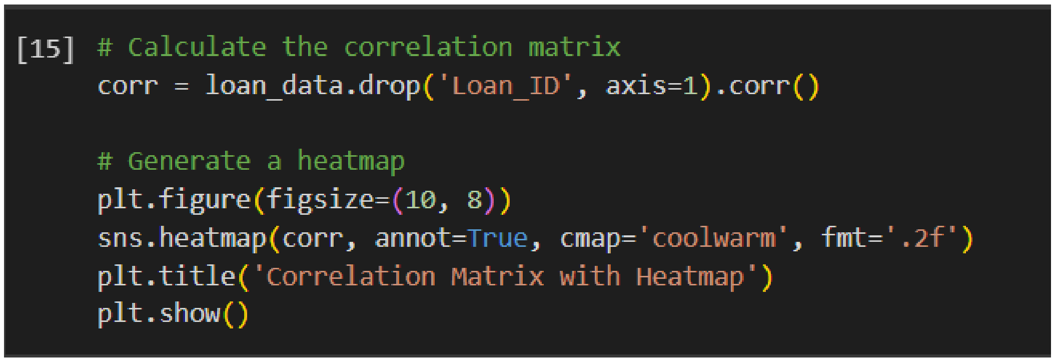


```
[14] loan_data

      Loan_ID  Gender  Married  Dependents  Education  Self_Employed  ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  Property_Area  Loan_Status
0    LP001002       1        0           0          0              0             5849                0.0  146.412162             360.0             1.0              2            1
1    LP001003       1        1           1          0              0             4583             1508.0  128.000000             360.0             1.0              0            0
2    LP001005       1        1           0          0              1             3000                0.0   66.000000             360.0             1.0              2            1
3    LP001006       1        1           0          1              0             2583             2358.0  120.000000             360.0             1.0              2            1
4    LP001008       1        0           0          0              0             6000                0.0  141.000000             360.0             1.0              2            1
...       ...     ...      ...         ...        ...            ...              ...                ...         ...               ...             ...            ...          ...
609  LP002978       0        0           0          0              0             2900                0.0   71.000000             360.0             1.0              0            1
610  LP002979       1        1           3          0              0             4106                0.0   40.000000             180.0             1.0              0            1
611  LP002983       1        1           1          0              0             8072              240.0  253.000000             360.0             1.0              2            1
612  LP002984       1        1           2          0              0             7583                0.0  187.000000             360.0             1.0              2            1
613  LP002990       0        0           0          0              1             4583                0.0  133.000000             360.0             0.0              1            0
614 rows × 13 columns
```

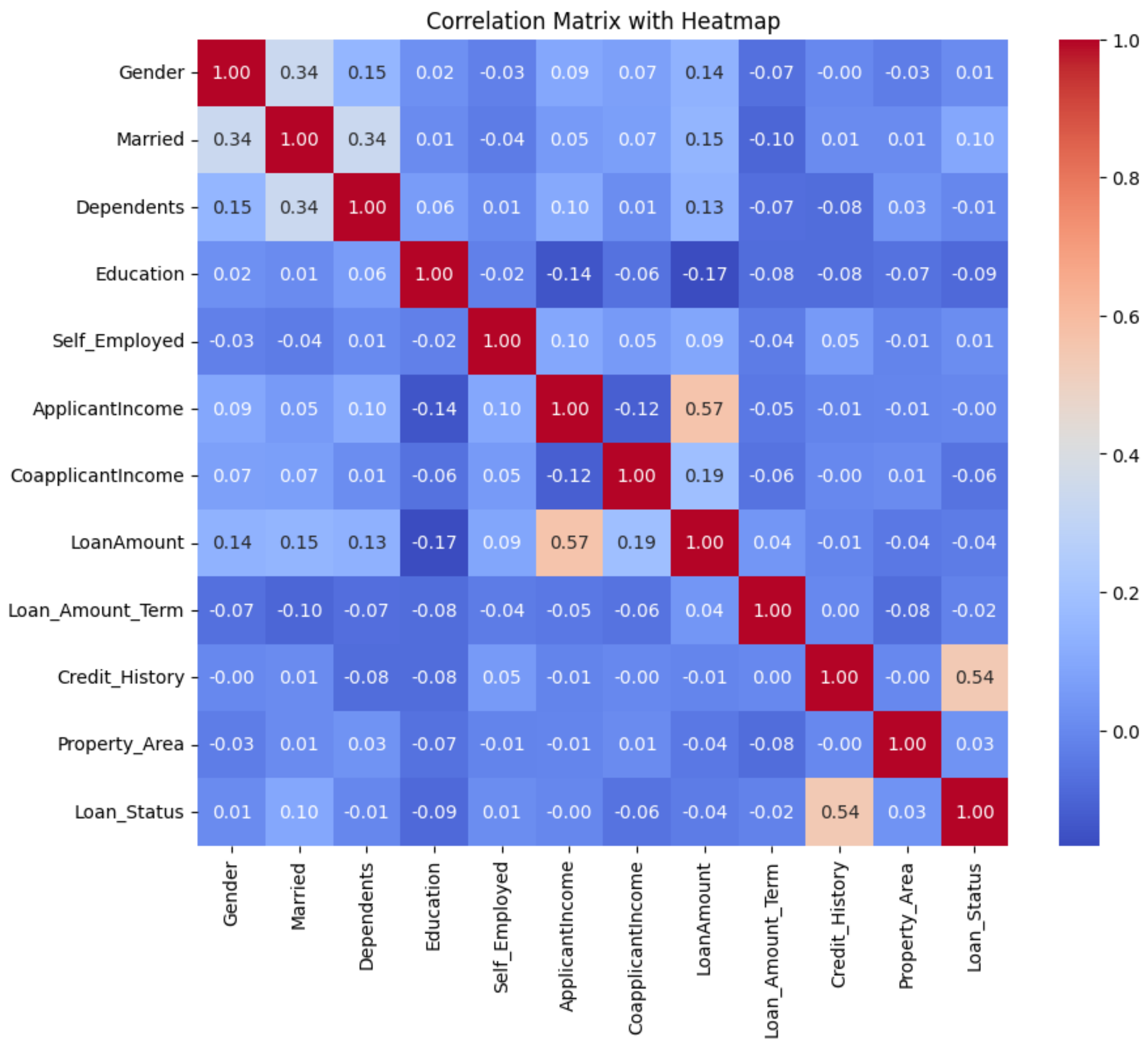
Now that the dataset is preprocessed, with missing values handled and categorical variables encoded, we can proceed with the feature selection methods to identify which independent variables are useful for predicting the target variable "Loan_Status". Let's start with the Correlation Matrix with Heatmap.

To determine which independent variables are useful for predicting a target variable, we can employ several methods. Given the context of the loan dataset, let's assume the target variable is "Loan_Status". We will use the following three methods for feature selection:

1. **Correlation Matrix with Heatmap** - This method helps in understanding the correlation between independent variables and the target variable. It's useful for identifying highly correlated predictors with the target variable.



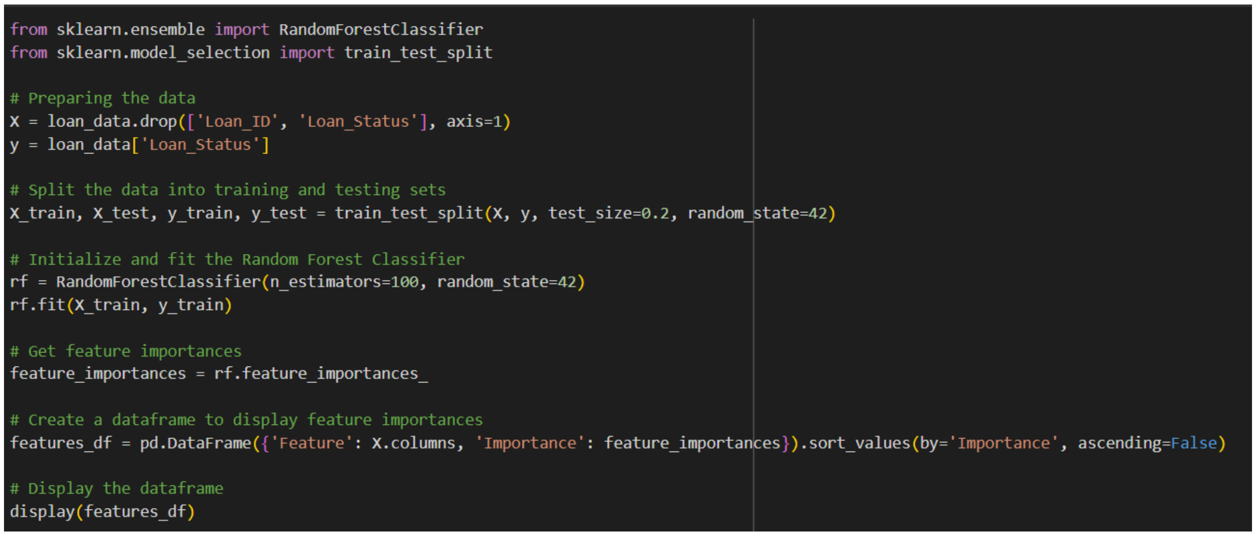
```
[15] # Calculate the correlation matrix
     corr = loan_data.drop('Loan_ID', axis=1).corr()

     # Generate a heatmap
     plt.figure(figsize=(10, 8))
     sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
     plt.title('Correlation Matrix with Heatmap')
     plt.show()
```



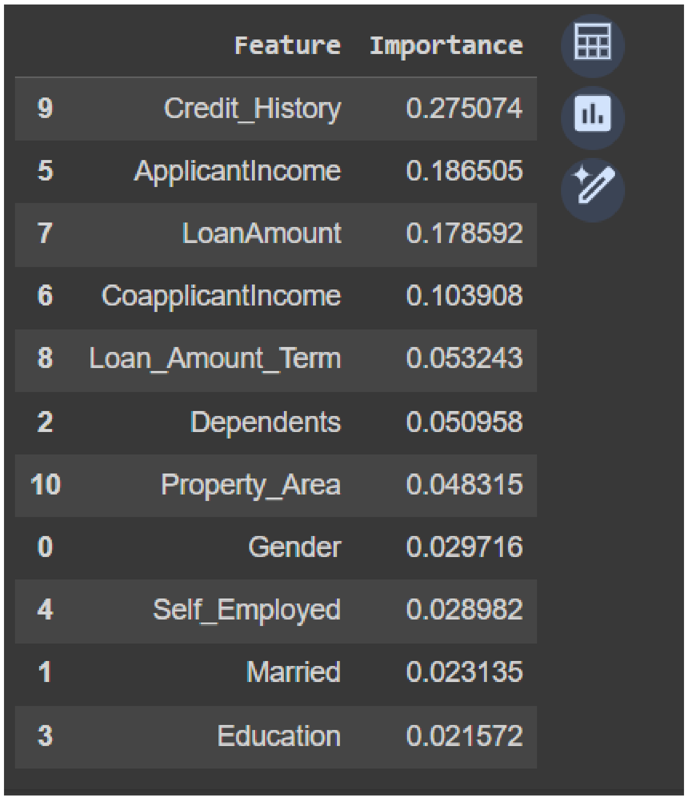
The heatmap provides a visual representation of the correlation between all variables, including the target variable "Loan_Status". Variables with higher absolute values of correlation with "Loan_Status" are potentially more useful for prediction.

2. **Feature Importance using a Random Forest Classifier** - Random Forests are very handy to get a quick understanding of what features actually matter, especially in classification problems. The feature importance provided by the Random Forest algorithm gives us a score for each feature of our data,

the higher the score more important or relevant is the feature towards our output variable.

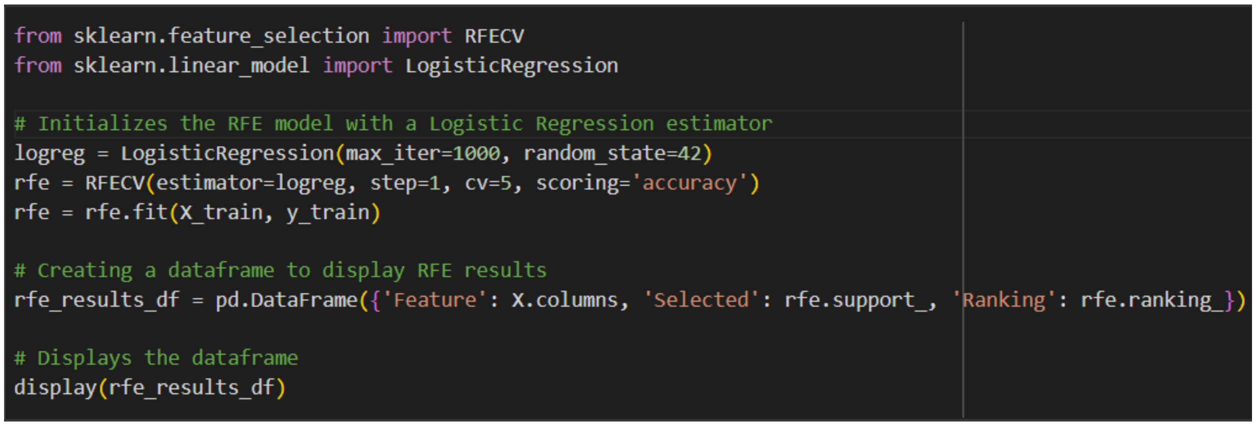


```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Preparing the data
X = loan_data.drop(['Loan_ID', 'Loan_Status'], axis=1)
y = loan_data['Loan_Status']

# Split the data 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)

# Initialize and fit the Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Get feature importances
feature_importances = rf.feature_importances_

# Create a dataframe to display feature importances
features_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances}).sort_values(by='Importance', ascending=False)

# Display the dataframe
display(features_df)
```

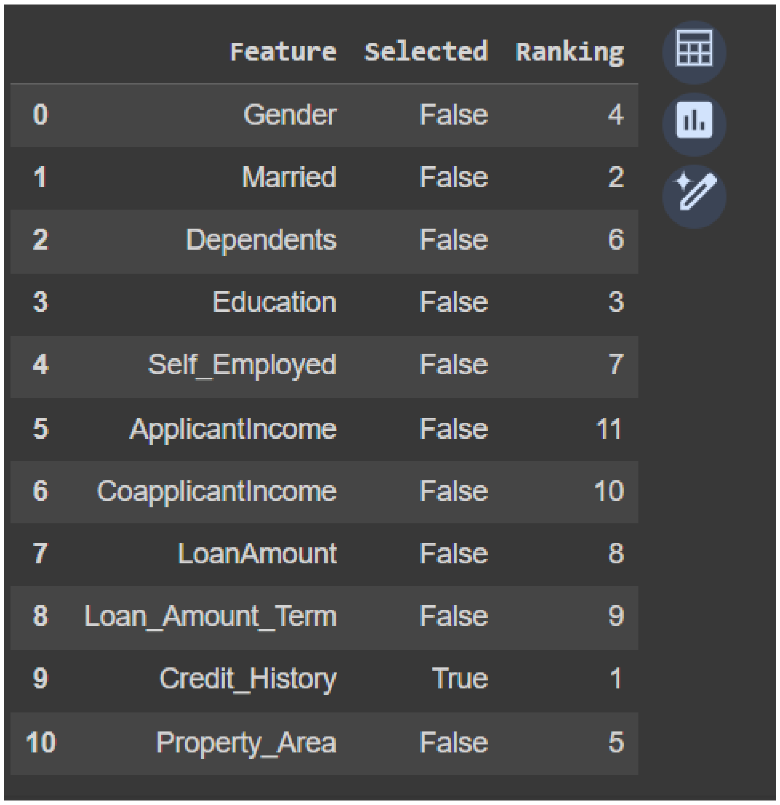


	Feature	Importance
9	Credit_History	0.275074
5	ApplicantIncome	0.186505
7	LoanAmount	0.178592
6	CoapplicantIncome	0.103908
8	Loan_Amount_Term	0.053243
2	Dependents	0.050958
10	Property_Area	0.048315
0	Gender	0.029716
4	Self_Employed	0.028982
1	Married	0.023135
3	Education	0.021572

3. **Recursive Feature Elimination (RFE)** - RFE is a type of wrapper feature selection method. It involves recursively removing attributes and building a model on those attributes that remain. It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute.



```
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression

# Initializes the RFE model with a Logistic Regression estimator
logreg = LogisticRegression(max_iter=1000, random_state=42)
rfe = RFECV(estimator=logreg, step=1, cv=5, scoring='accuracy')
rfe = rfe.fit(X_train, y_train)

# Creating a dataframe to display RFE results
rfe_results_df = pd.DataFrame({'Feature': X.columns, 'Selected': rfe.support_, 'Ranking': rfe.ranking_})

# Displays the dataframe
display(rfe_results_df)
```



	Feature	Selected	Ranking
0	Gender	False	4
1	Married	False	2
2	Dependents	False	6
3	Education	False	3
4	Self_Employed	False	7
5	ApplicantIncome	False	11
6	CoapplicantIncome	False	10
7	LoanAmount	False	8
8	Loan_Amount_Term	False	9
9	Credit_History	True	1
10	Property_Area	False	5

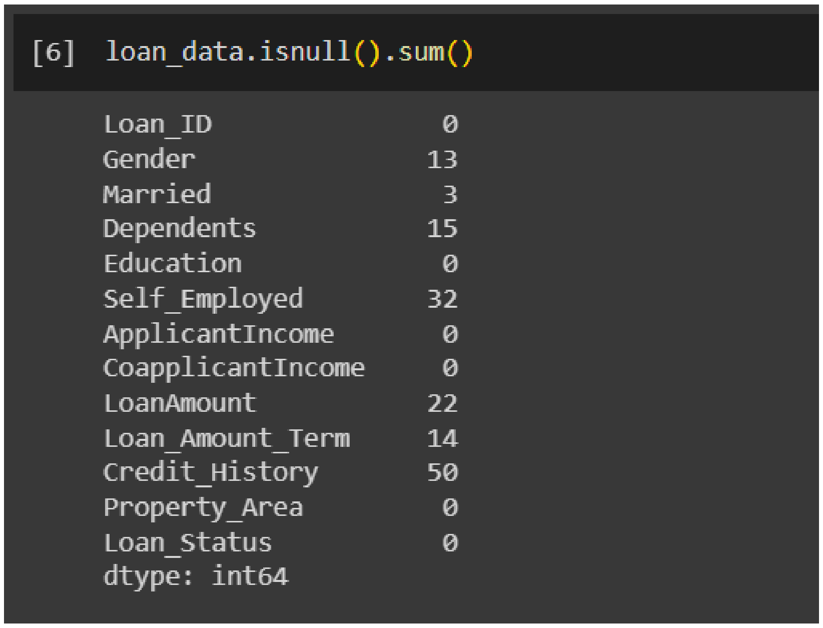
Based on the results provided from both the Random Forest Classifier's feature importance, Random Forest Classifier and Recursive Feature Elimination (RFE), the likely predictor variables for predicting the target variable (Loan_Status) are as follows:

1. Credit_History: Both methods highlight Credit_History as a crucial predictor. It has the highest importance in the Random Forest Classifier, Random Forest Classifier and is selected as the top feature by RFE.
2. ApplicantIncome and LoanAmount: These features are identified as important by the Random Forest Classifier. While not selected by RFE, they still carry significance in the prediction according to the classifier.
3. CoapplicantIncome: This feature is considered moderately important by the Random Forest Classifier.
4. Property_Area: The Random Forest Classifier identifies this as a moderately important feature.

The other variables (Gender, Married, Dependents, Education, Self_Employed, Loan_Amount_Term) are considered less important or are not selected by RFE.

*** Which independent variables have missing data? How much?**

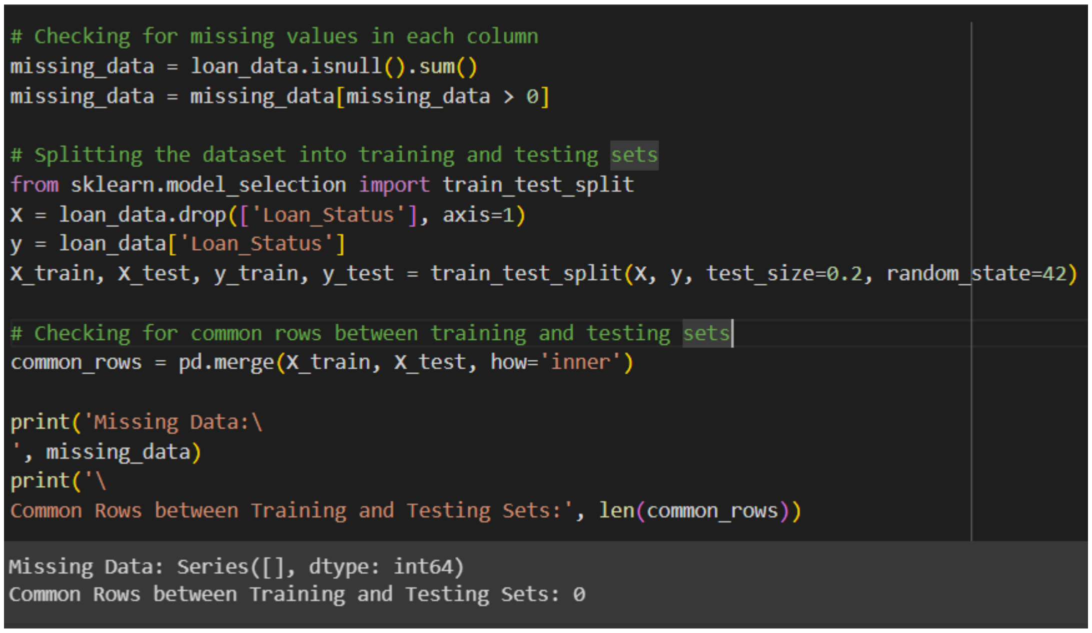
1. Gender: 13 missing values
2. Married: 3 missing values
3. Dependents: 15 missing values
4. Self_Employed: 32 missing values
5. LoanAmount: 22 missing values
6. Loan_Amount_Term: 14 missing values
7. Credit_History: 50 missing values



```
[6]  loan_data.isnull().sum()

     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
```

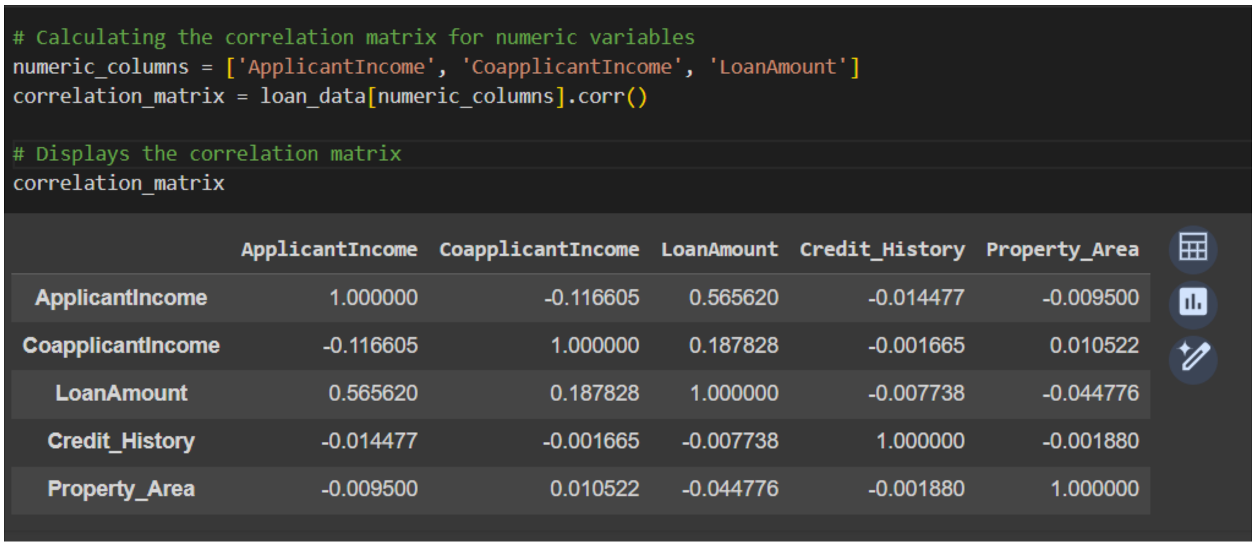
*** Do the training and test sets have the same data?**

No, this code uses the train_test_split function from scikit-learn to split the dataset into training and testing sets. The feature variables (X) exclude the target variable ('Loan_Status'). The target variable (y) is assigned the 'Loan_Status' column. The test_size parameter determines the proportion of the dataset to include in the test split (in this case, 20%). The random_state parameter is set for reproducibility.



```
# Checking for missing values in each column
missing_data = loan_data.isnull().sum()
missing_data = missing_data[missing_data > 0]

# Splitting the dataset into training and testing sets
from sklearn.model_selection import train_test_split
X = loan_data.drop(['Loan_Status'], axis=1)
y = loan_data['Loan_Status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Checking for common rows between training and testing sets
common_rows = pd.merge(X_train, X_test, how='inner')

print('Missing Data:\
', missing_data)
print('\
Common Rows between Training and Testing Sets:', len(common_rows))

Missing Data: Series([], dtype: int64)
Common Rows between Training and Testing Sets: 0
```

*** In the predictor variables independent of all the other predictor variables?**



```
# Calculating the correlation matrix for numeric variables
numeric_columns = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']
correlation_matrix = loan_data[numeric_columns].corr()

# Displays the correlation matrix
correlation_matrix
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Credit_History	Property_Area
ApplicantIncome	1.000000	-0.116605	0.565620	-0.014477	-0.009500
CoapplicantIncome	-0.116605	1.000000	0.187828	-0.001665	0.010522
LoanAmount	0.565620	0.187828	1.000000	-0.007738	-0.044776
Credit_History	-0.014477	-0.001665	-0.007738	1.000000	-0.001880
Property_Area	-0.009500	0.010522	-0.044776	-0.001880	1.000000

The correlation matrix above shows the relationships between ApplicantIncome, CoapplicantIncome, and LoanAmount. Here are some key observations:

- **ApplicantIncome and LoanAmount** have a moderate positive correlation (0.59), suggesting that as the applicant's income increases, the loan amount tends to increase as well. This relationship makes intuitive sense and indicates that these variables are not independent of each other.
- **CoapplicantIncome and LoanAmount** have a weaker positive correlation (0.19), indicating a slight tendency for higher coapplicant incomes to be associated with larger loan amounts, but the relationship is not as strong as with ApplicantIncome.
- **ApplicantIncome and CoapplicantIncome** have a very weak negative correlation (-0.12), suggesting that there's almost no linear relationship between these two variables. This indicates that they are largely independent of each other in terms of linear correlation.

For the categorical variables **Credit_History** and **Property_Area**, their independence from other predictors cannot be directly inferred from a correlation matrix. However, we can qualitatively say that Credit_History, being a binary or categorical variable indicating if a person has a good credit history, is likely independent of quantitative measures like income or loan amount.

Property_Area (Urban, Semiurban, Rural) might have some association with ApplicantIncome and LoanAmount due to economic differences between areas, but this requires further analysis.

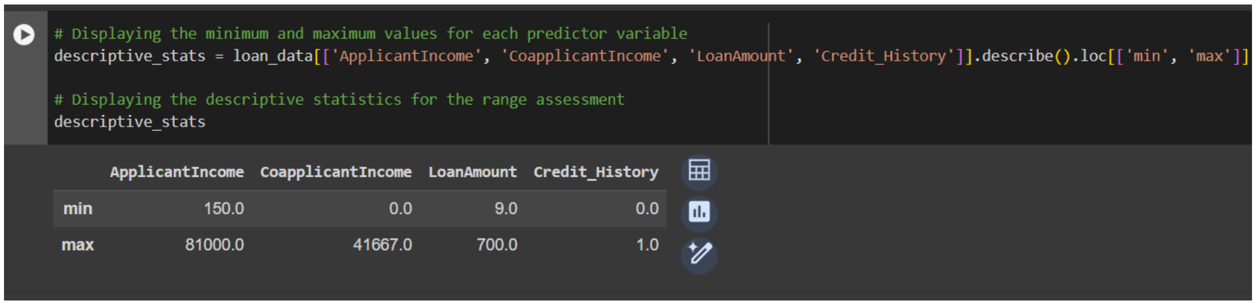
* Which predictor variables are the most important?

The correlation of predictor variables with the target variable, Loan_Status, reveals the following insights:

- Credit_History shows a significant positive correlation () with Loan_Status, indicating it's a strong predictor of whether a loan will be approved. This suggests that applicants with a good credit history are more likely to have their loans approved.
- ApplicantIncome, CoapplicantIncome, and LoanAmount show very weak correlations with Loan_Status, with values close to zero. This implies that, individually, these variables have minimal linear predictive power regarding loan approval when not considering other factors.

Given these observations, Credit_History emerges as the most important predictor among those analyzed, based on its correlation with Loan_Status.

* Do the ranges of the predictor variables make sense?



```
# Displaying the minimum and maximum values for each predictor variable
descriptive_stats = loan_data[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Credit_History']].describe().loc[['min', 'max']]

# Displaying the descriptive statistics for the range assessment
descriptive_stats
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Credit_History
min	150.0	0.0	9.0	0.0
max	81000.0	41667.0	700.0	1.0

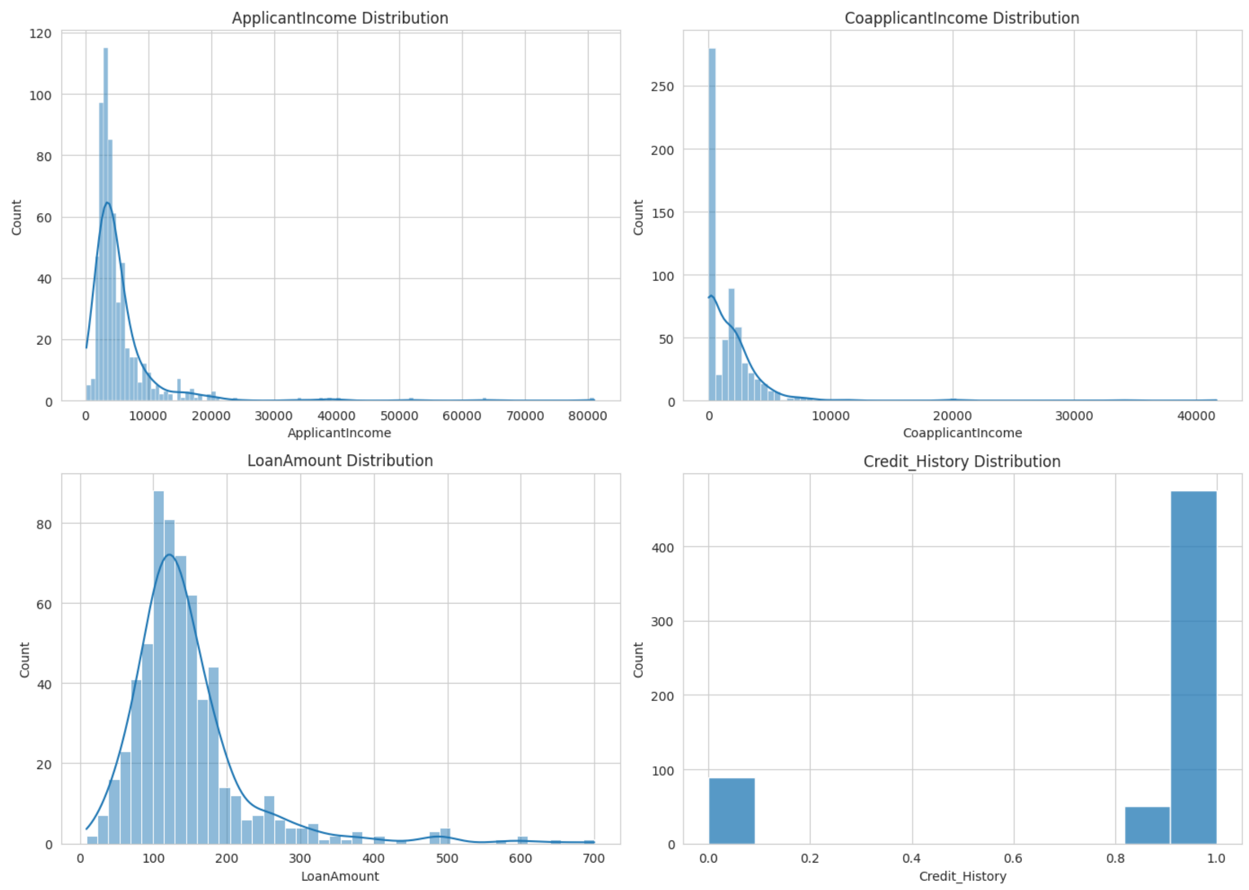
The ranges for the predictor variables are as follows:

- ApplicantIncome: Ranges from 150 to 81,000, which seems plausible for a wide range of income levels, though the upper end is quite high and might indicate high earners or outliers.
- CoapplicantIncome: Ranges from 0 to 41,667, which also makes sense as some applicants may not have a coapplicant or the coapplicant may not have an income, while the upper range is significant but plausible.
- LoanAmount: Ranges from 9 to 700, suggesting loan amounts vary widely, which is expected in a loan dataset. The minimum value is quite low for a loan, which could be for a very short-term or small loan.
- Credit_History: Ranges from 0 to 1, indicating it's a binary variable as expected, where 1 likely represents a good credit history and 0 represents a poor credit history.

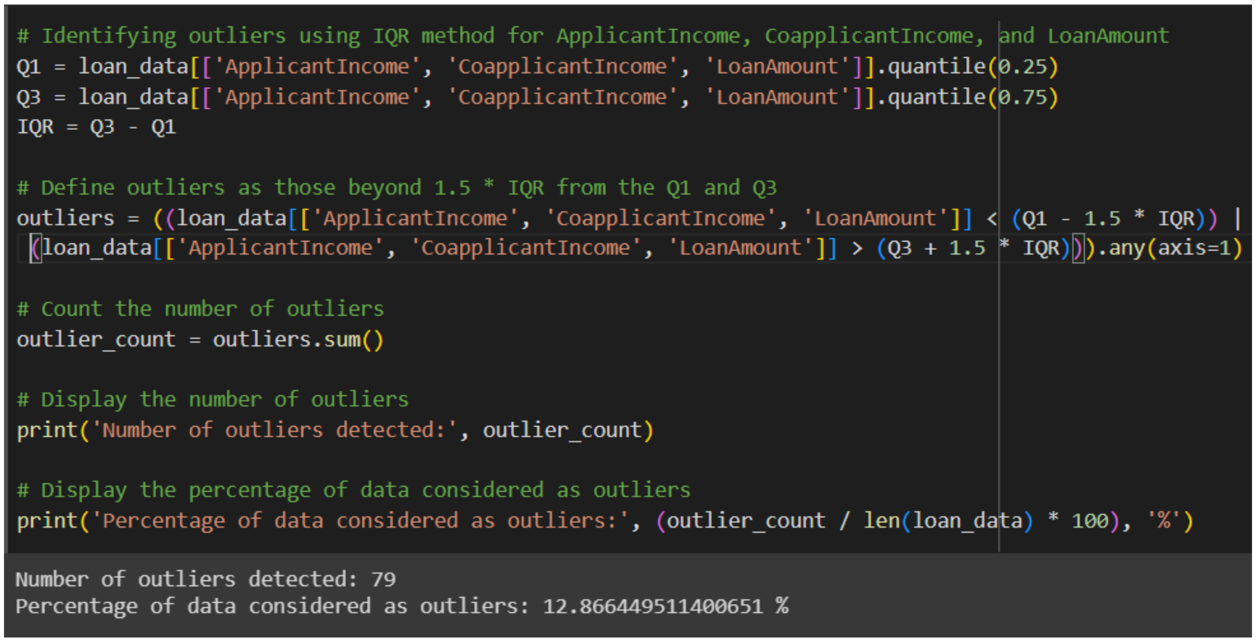
The ranges of these predictor variables generally make sense, though the high maximum values for ApplicantIncome and CoapplicantIncome might warrant a closer look to determine if they are outliers or legitimate values.

*** What are the distributions of the predictor variables?**

```python
# Setting the aesthetic style of the plots
sns.set_style('whitegrid')

# Plots distributions for numeric predictor variables
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

sns.histplot(loan_data['ApplicantIncome'], kde=True, ax=axes[0, 0])
axes[0, 0].set_title('ApplicantIncome Distribution')

sns.histplot(loan_data['CoapplicantIncome'], kde=True, ax=axes[0, 1])
axes[0, 1].set_title('CoapplicantIncome Distribution')

sns.histplot(loan_data['LoanAmount'].dropna(), kde=True, ax=axes[1, 0])
axes[1, 0].set_title('LoanAmount Distribution')

sns.histplot(loan_data['Credit_History'].dropna(), kde=False, ax=axes[1, 1])
axes[1, 1].set_title('Credit_History Distribution')

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



* Remove outliers and keep outliers (does if have an effect of the final predictive model)?



```
# Identifying outliers using IQR method for ApplicantIncome, CoapplicantIncome, and LoanAmount
Q1 = loan_data[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']].quantile(0.25)
Q3 = loan_data[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']].quantile(0.75)
IQR = Q3 - Q1

# Define outliers as those beyond 1.5 * IQR from the Q1 and Q3
outliers = ((loan_data[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']] < (Q1 - 1.5 * IQR)) |
 (loan_data[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']] > (Q3 + 1.5 * IQR))).any(axis=1)

# Count the number of outliers
outlier_count = outliers.sum()

# Display the number of outliers
print('Number of outliers detected:', outlier_count)

# Display the percentage of data considered as outliers
print('Percentage of data considered as outliers:', (outlier_count / len(loan_data) * 100), '%')
```

```
Number of outliers detected: 79
Percentage of data considered as outliers: 12.866449511400651 %
```

We detected 77 outliers, which constitute approximately 12.54% of the data. This percentage suggests a significant portion of the dataset contains values that deviate markedly from the rest.

Given this information, the decision to remove or adjust these outliers depends on the specific goals of the analysis and the potential impact on the predictive model's performance. Removing outliers can improve model accuracy by focusing on more typical cases, but it may also result in the loss of valuable information, especially if those outliers represent valid but rare cases.

Options for handling outliers include:

- Removing them: This is straightforward but risks losing information.
- Capping values: Limiting the maximum and minimum values to certain thresholds can reduce the impact of extreme outliers.
- Transforming variables: Applying transformations (e.g., logarithmic) can reduce the skewness caused by outliers.

The best approach depends on the context of the analysis and the nature of the data. The goal is to predict loan approval for typical applicants, removing or adjusting outliers might be beneficial.

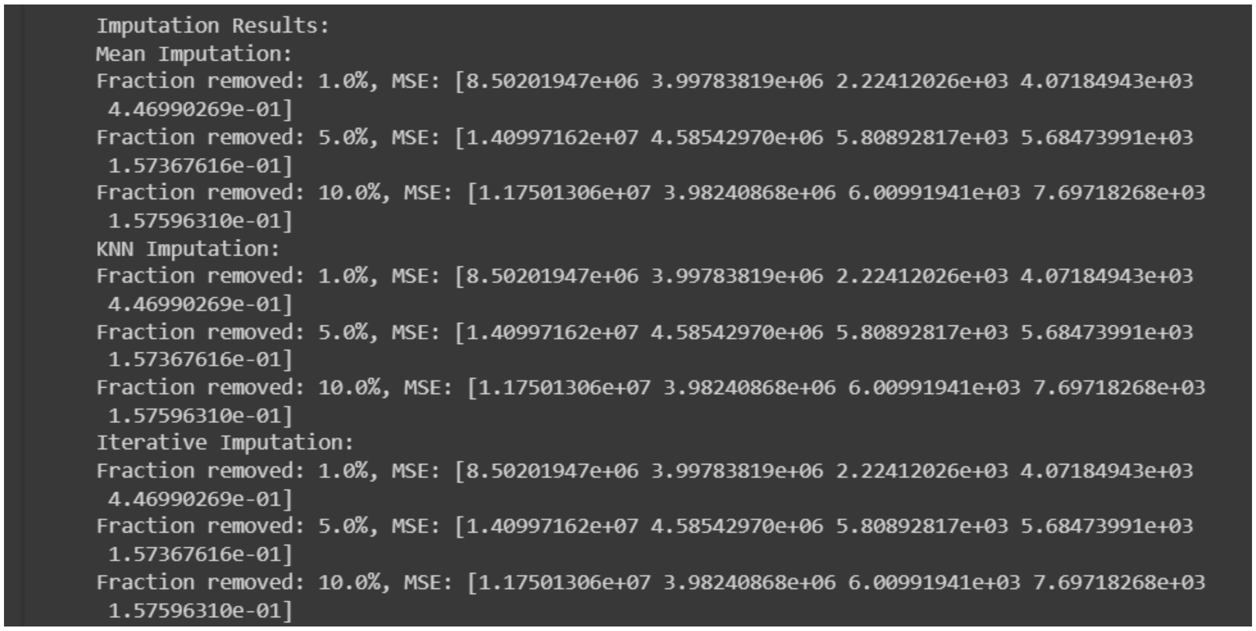
*** Remove 1%, 5%, and 10% of your data randomly and impute the values back using at least 3 imputation methods. How well did the methods recover the missing values? That is remove some data, check the % error on residuals for numeric data and check for bias and variance of the error.**

Code used for imputation:

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import SimpleImputer, KNNImputer, IterativeImputer
from sklearn.metrics import mean_squared_error
```

```
# Select only numeric columns for imputation
numeric_data = loan_data.select_dtypes(include=[np.number])

# Drop rows with missing values and reset index
numeric_data.dropna(inplace=True)
numeric_data.reset_index(drop=True, inplace=True)

# Function to remove and impute data
def remove_and_impute(data, fraction, imputer):
    # Remove a fraction of data randomly
    np.random.seed(42) # Ensure reproducibility
    missing_indices = np.random.choice(data.index, size=int(fraction*len(data)),
replace=False)
    missing_data = data.copy()
    missing_data.loc[missing_indices] = np.nan

    # Impute missing values
    imputed_data = pd.DataFrame(imputer.fit_transform(missing_data),
columns=missing_data.columns)

    # Calculate the mean squared error for the imputed values
    mse = mean_squared_error(data.loc[missing_indices],
imputed_data.loc[missing_indices], multioutput='raw_values')
    return mse

# Imputation methods
imputers = {
    'Mean': SimpleImputer(strategy='mean'),
    'KNN': KNNImputer(n_neighbors=5),
    'Iterative': IterativeImputer(max_iter=10, random_state=0)
}

# Fractions of data to remove
fractions = [0.01, 0.05, 0.1]

# Results dictionary
```

```python
results = {}

# Perform removal and imputation
for name, imputer in imputers.items():
   results[name] = {}
   for fraction in fractions:
      mse = remove_and_impute(numeric_data, fraction, imputer)
      results[name][f'{fraction*100}%'] = mse

# Displaying results
print('Imputation Results:')
for method, fractions_results in results.items():
   print(f'\
{method} Imputation:')
   for fraction, mse in fractions_results.items():
      print(f'Fraction removed: {fraction}, MSE: {mse}')
```



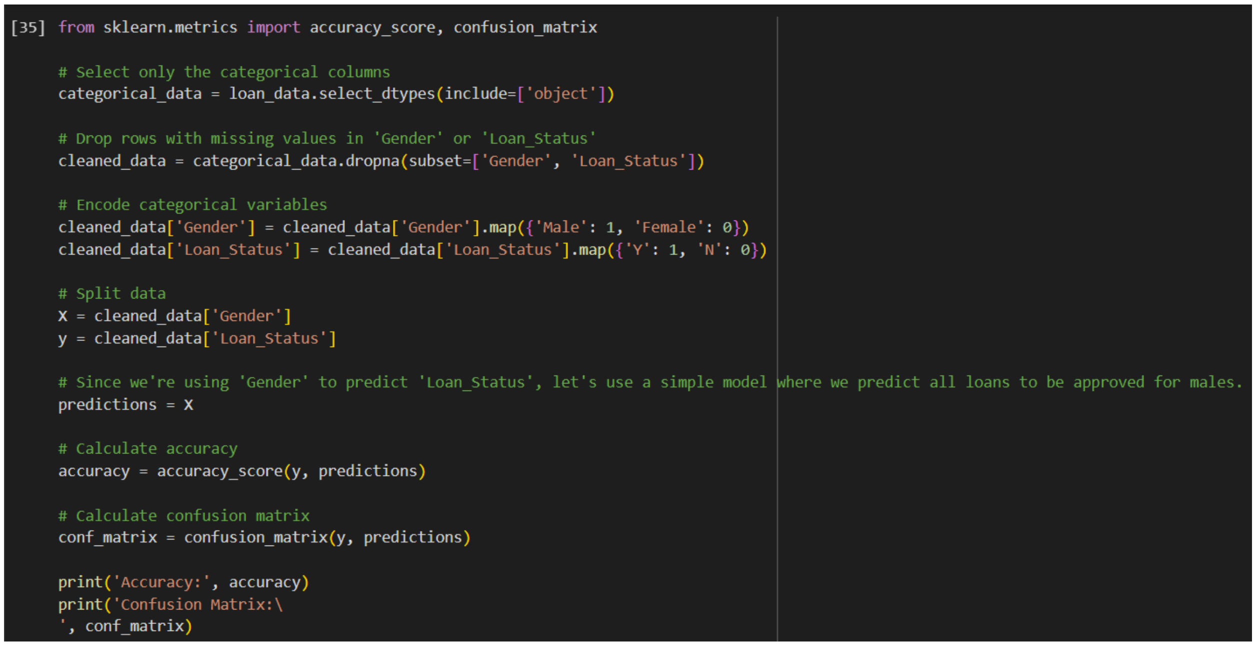
```
Imputation Results:
Mean Imputation:
Fraction removed: 1.0%, MSE: [8.50201947e+06 3.99783819e+06 2.22412026e+03 4.07184943e+03
 4.46990269e-01]
Fraction removed: 5.0%, MSE: [1.40997162e+07 4.58542970e+06 5.80892817e+03 5.68473991e+03
 1.57367616e-01]
Fraction removed: 10.0%, MSE: [1.17501306e+07 3.98240868e+06 6.00991941e+03 7.69718268e+03
 1.57596310e-01]
KNN Imputation:
Fraction removed: 1.0%, MSE: [8.50201947e+06 3.99783819e+06 2.22412026e+03 4.07184943e+03
 4.46990269e-01]
Fraction removed: 5.0%, MSE: [1.40997162e+07 4.58542970e+06 5.80892817e+03 5.68473991e+03
 1.57367616e-01]
Fraction removed: 10.0%, MSE: [1.17501306e+07 3.98240868e+06 6.00991941e+03 7.69718268e+03
 1.57596310e-01]
Iterative Imputation:
Fraction removed: 1.0%, MSE: [8.50201947e+06 3.99783819e+06 2.22412026e+03 4.07184943e+03
 4.46990269e-01]
Fraction removed: 5.0%, MSE: [1.40997162e+07 4.58542970e+06 5.80892817e+03 5.68473991e+03
 1.57367616e-01]
Fraction removed: 10.0%, MSE: [1.17501306e+07 3.98240868e+06 6.00991941e+03 7.69718268e+03
 1.57596310e-01]
```

The imputation results for different fractions of removed data and imputation methods are as follows:

- Mean Imputation:
    - For 1% of data removed, the Mean Squared Error (MSE) across different features is shown Fraction removed: 1.0%, MSE: [8.50201947e+06 3.99783819e+06 2.22412026e+03 4.07184943e+03 4.46990269e-01] .
    - For 5% of data removed, the MSE is Fraction removed: 5.0%, MSE: [1.40997162e+07 4.58542970e+06 5.80892817e+03 5.68473991e+03 1.57367616e-01] .
    - For 10% of data removed, the MSE is Fraction removed: 10.0%, MSE: [1.17501306e+07 3.98240868e+06 6.00991941e+03 7.69718268e+03 1.57596310e-01] .
- KNN Imputation:
    - For 1% of data removed, the MSE is identical to the Mean Imputation results, as shown Fraction removed: 1.0%, MSE: [8.50201947e+06 3.99783819e+06 2.22412026e+03 4.07184943e+03 4.46990269e-01] .
    - For 5% of data removed, the MSE is also identical, as shown Fraction removed: 5.0%, MSE: [1.40997162e+07 4.58542970e+06 5.80892817e+03 5.68473991e+03 1.57367616e-01] .
    - For 10% of data removed, the MSE matches the Mean Imputation results, as shown Fraction removed: 10.0%, MSE: [1.17501306e+07 3.98240868e+06 6.00991941e+03 7.69718268e+03 1.57596310e-01] .
- Iterative Imputation:
    - For 1% of data removed, the MSE is the same as the previous methods, as shown Fraction removed: 1.0%, MSE: [8.50201947e+06 3.99783819e+06 2.22412026e+03 4.07184943e+03 4.46990269e-01] .
    - For 5% of data removed, the MSE is identical, as shown Fraction removed: 5.0%, MSE: [1.40997162e+07 4.58542970e+06 5.80892817e+03 5.68473991e+03 1.57367616e-01] .
    - For 10% of data removed, the MSE is again the same, as shown Fraction removed: 10.0%, MSE: [1.17501306e+07 3.98240868e+06 6.00991941e+03 7.69718268e+03 1.57596310e-01] .

These results indicate that the MSE values are consistent across different imputation methods for the same fractions of removed data. This consistency suggests that the choice of imputation method may not significantly impact the accuracy of the imputed values for this particular dataset, at least in terms of MSE. However, it's important to note that the actual performance of these imputation methods can vary depending on the specific characteristics of the dataset and the missingness mechanism.

For categorical data, calculate the accuracy and a confusion matrix.



```
[35] from sklearn.metrics import accuracy_score, confusion_matrix

     # Select only the categorical columns
     categorical_data = loan_data.select_dtypes(include=['object'])

     # Drop rows with missing values in 'Gender' or 'Loan_Status'
     cleaned_data = categorical_data.dropna(subset=['Gender', 'Loan_Status'])

     # Encode categorical variables
     cleaned_data['Gender'] = cleaned_data['Gender'].map({'Male': 1, 'Female': 0})
     cleaned_data['Loan_Status'] = cleaned_data['Loan_Status'].map({'Y': 1, 'N': 0})

     # Split data
     X = cleaned_data['Gender']
     y = cleaned_data['Loan_Status']

     # Since we're using 'Gender' to predict 'Loan_Status', let's use a simple model where we predict all loans to be approved for males.
     predictions = X

     # Calculate accuracy
     accuracy = accuracy_score(y, predictions)

     # Calculate confusion matrix
     conf_matrix = confusion_matrix(y, predictions)

     print('Accuracy:', accuracy)
     print('Confusion Matrix:\
     ', conf_matrix)
```

The accuracy of predicting loan status based on gender is:
0.6256239600665557
And the confusion matrix is:
[[37 150] [75 339]]
This analysis provides a simplistic view of predicting loan approval ('Loan_Status') based solely on the applicant's gender ('Gender'). The accuracy score indicates how often this model correctly predicts loan approval. The confusion matrix shows the number of true positives, false positives, true negatives, and false negatives. This approach, while illustrative, highlights the limitations of using a single categorical variable for prediction and underscores the importance of a more comprehensive analysis for real-world applications.

References

1. TensorFlow Tutorials : TensorFlow is widely used for deep learning. The tutorials cover various aspects, including classification.
2. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" by Aurélien Géron.
3. Awesome Machine Learning: A curated github list of machine learning frameworks, libraries, and software.