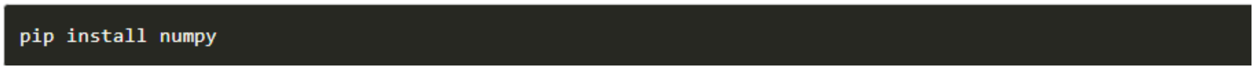
5. Practical Part

5.1 Project Environment

Set up an environment for my project's implementation by installing Python 3.6.8 and a Jupyter notebook. We need to install and import some Python libraries into the Jupyter notebook. My thesis goal is to use to solve business problems using data mining tasks and machine learning algorithms. The main reason for using Python is that its open-sourced libraries make it easy to manipulate data and are user-friendly. Nowadays, the majority of data scientists use Python and related environments to solve ML and DM projects.

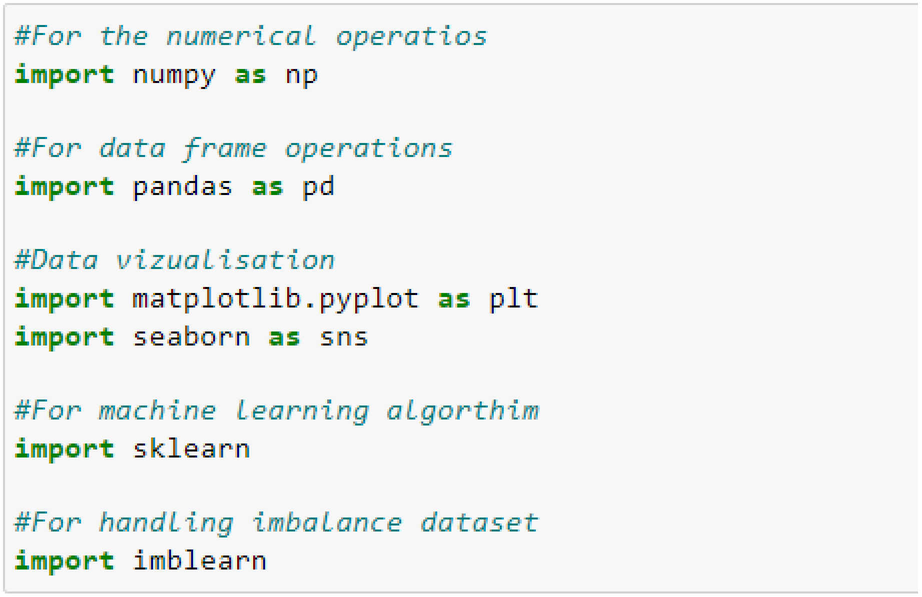
Example of installation python library as below represents NumPy,



```
pip install numpy
```

Source: https://numpy.org/install/

Importing Libraries: Input python code



```
#For the numerical operatios
import numpy as np

#For data frame operations
import pandas as pd

#Data vizualisation
import matplotlib.pyplot as plt
import seaborn as sns

#For machine learning algorthim
import sklearn

#For handling imbalance dataset
import imblearn
```

5.1.1 Business Understanding

A finance company provides in all types of home loans. They have a presence in all urban, suburban, and rural areas. The customer first applies for a home loan, and the company then verifies the customer's loan eligibility.

The company wants to automate the loan eligibility process (in real time) based on the information provided by the customer when filling out the online application form. Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History, and other details are included.

To Automate process of home loan,

- Provided a dataset to identify the customer segments that are eligible for loan.
- Create a robust model by finding hidden trends and patterns.
- Trends and patterns can be utilized to predict a candidate's loan status.

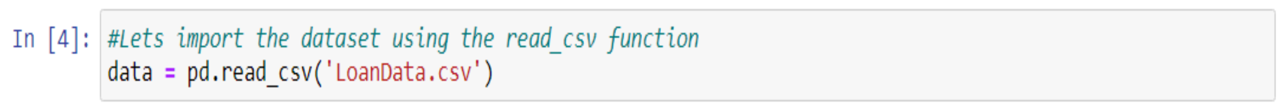
To evaluate performance of model,

- Accuracy.
- Precision, Recall and F1 Score.

We can assess the most accurate model to forecast applicant’s loan status.

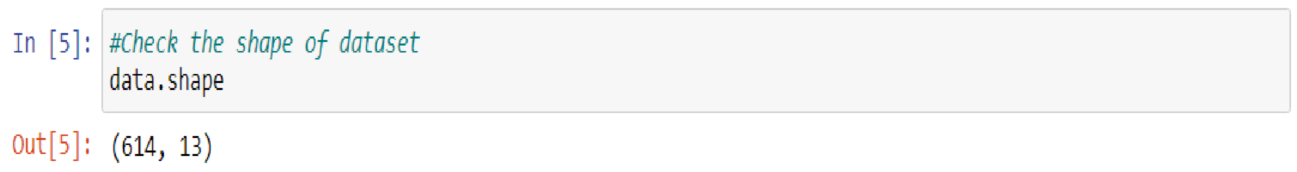
5.1.2 Data Understanding

Load dataset: First, collect this data set from the Kaggle and stored my project environment. The dataset is first imported into Jupyter notebook using the **read.csv()** function from the pandas library.

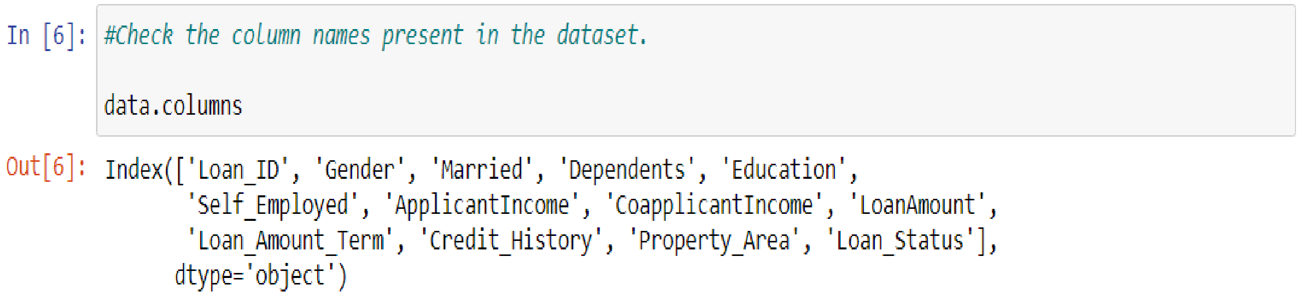


```
In [4]: #Lets import the dataset using the read_csv function
        data = pd.read_csv('LoanData.csv')
```

Let check the shape of the dataset, where have found 614 rows and 13 columns and get columns name using the **.columns** functions,



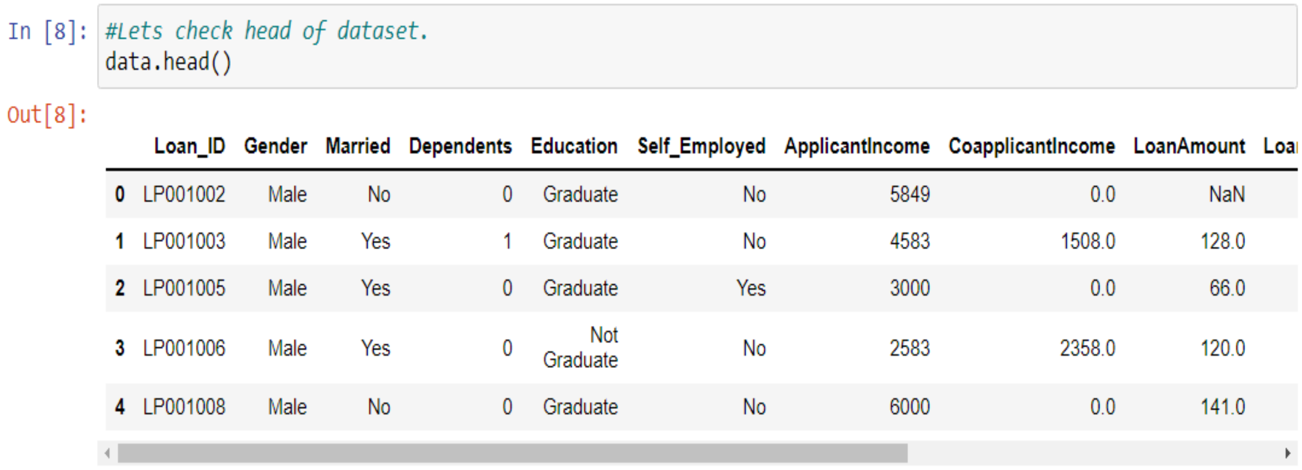
```
In [5]: #Check the shape of dataset
        data.shape

Out[5]: (614, 13)
```



```
In [6]: #Check the column names present in the dataset.

        data.columns
```

```
Out[6]: Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
               'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
               'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
              dtype='object')
```

Now let's check the top five row of the data using with **.head** function, which will help us to check the values present in each of the columns.



```
In [8]: #Lets check head of dataset.
        data.head()
```

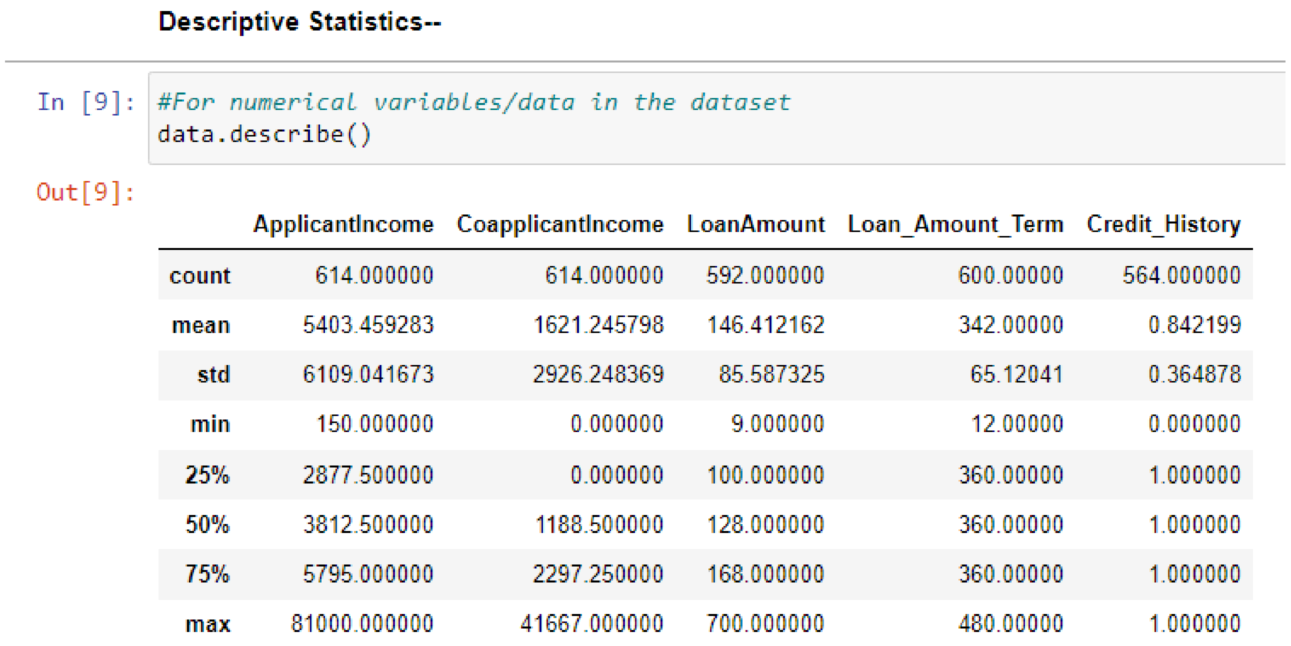
Out[8]:

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

As above output, we know that loan status column is the target column, and the remaining columns, aside from the loan status column is known as independent column.

Execute describe function: Descriptive statistics summarize or describe the characteristics of a dataset. Its measurements of central tendency explain the data set's center (mean, median, mode) and measures of variability describe the data set's dispersion (variance, standard deviation) as well as frequency distribution within the data set (count).[38]

In our dataset two types of variables contains one is numerical or quantitative data and another is categorial or qualitative data. First step, we must check the descriptive statistics using **describe()** function for the numerical data or continuous data, whose values either integer or float.


Descriptive Statistics--

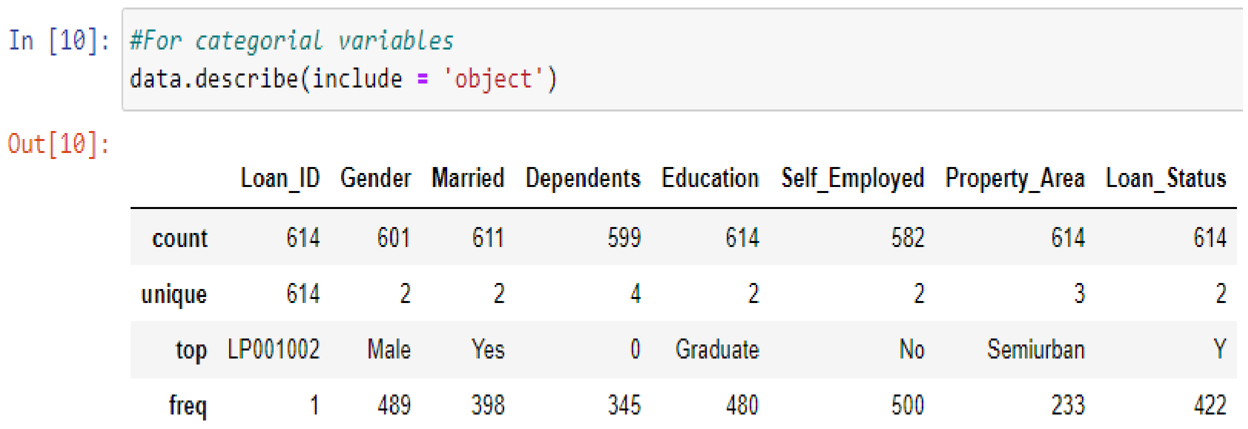
```
In [9]: #For numerical variables/data in the dataset
        data.describe()
```

Out[9]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

There has some interesting insight from this output, It is quite clear that in the columns, applicant income, co-applicant income and loan amount having outliers. Mean value and maximum value of applicant income has huge differences similarly co-applicant income column and loan amount value column. Outliers consider very bad for predictive models as outliers creates lower performance level of models and destroy learning pattern of the data.

Now, execute descriptive statistics on categorical columns or columns with object data. We will use the same **describe()** function again, but this time we will add include is equal to object,

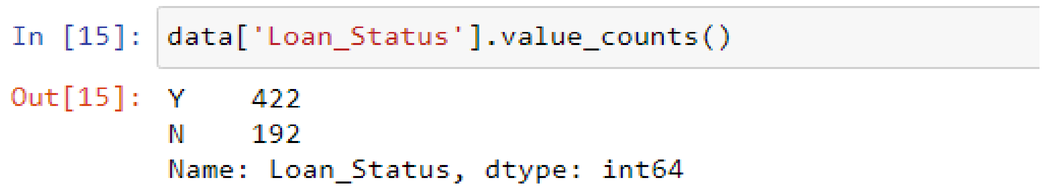

```
In [10]: #For categorial variables
         data.describe(include = 'object')
```

Out[10]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Loan_Status
count	614	601	611	599	614	582	614	614
unique	614	2	2	4	2	2	3	2
top	LP001002	Male	Yes	0	Graduate	No	Semiurban	Y
freq	1	489	398	345	480	500	233	422

Now we have the total number of records in the column, the unique number of records in the column, the top category, which is the category with the highest occurrence among the other categories in the columns, and the frequency of the top category.

Similarly, we can now understand the dynamics and statistics of the detail set's categorical columns, such as the loan status column. This indicates that there are 614 entries, and two distinct values present in this column, which we already know will be 'Y', which indicates that the loan is given, and N, which indicates that the loan is not granted.

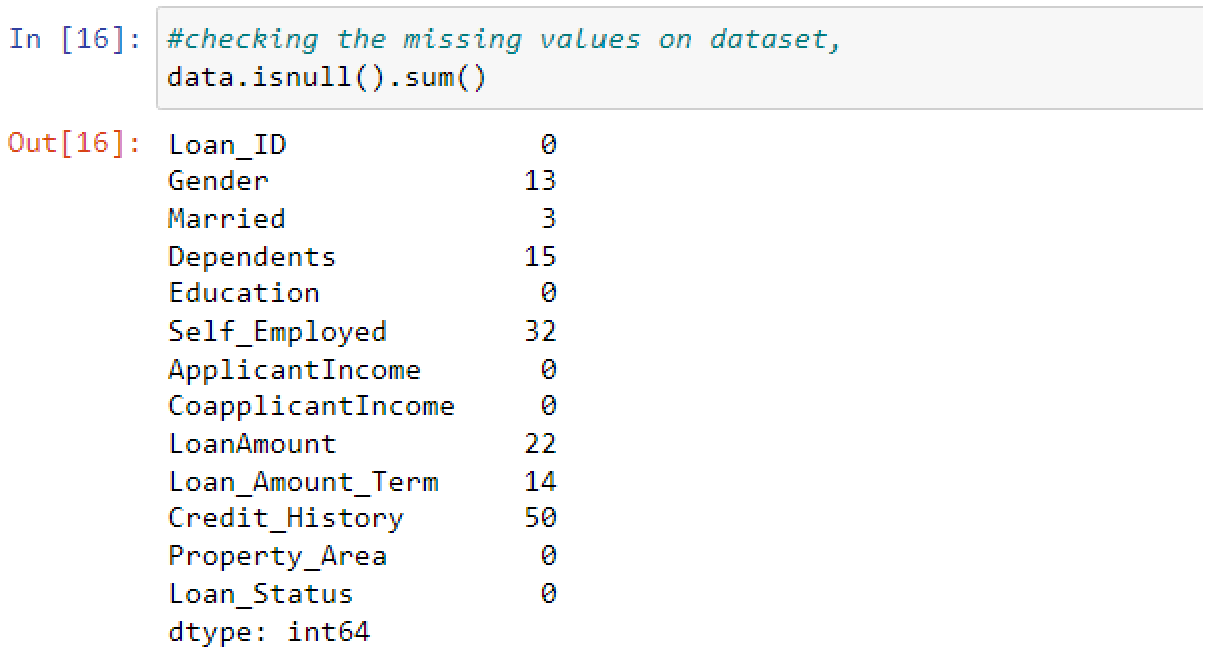


```
In [15]: data['Loan_Status'].value_counts()

Out[15]: Y    422
         N    192
         Name: Loan_Status, dtype: int64
```

The frequency 'Y' has 422 records out of the total 614 recordings in this test set. Using the **value_counts()** function, we can observe that the number of records is more than any record. We might also claim that the data is imbalanced. When we utilize machine learning models within balanced classes, we usually get very bad results that are fully biased towards a class with a higher distribution.

5.1.3 Data Preparation

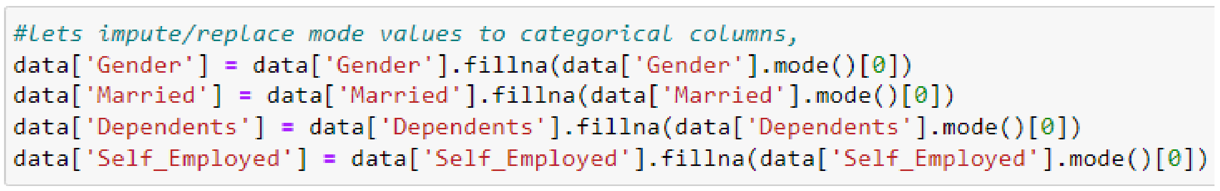
Data Cleaning: We will clean up the dataset before proceeding with the development of our predictive models. With dirty or unclean data, we cannot build a predictive model. Missing values and outlier values are regarded as dirty or unclean data. If there are any missing values in the data set, we can use the **ISNULL** function in combination with the SUM function to determine the number of missing values in each of the columns in our data set.



```
In [16]: #checking the missing values on dataset,
         data.isnull().sum()

Out[16]: 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
```

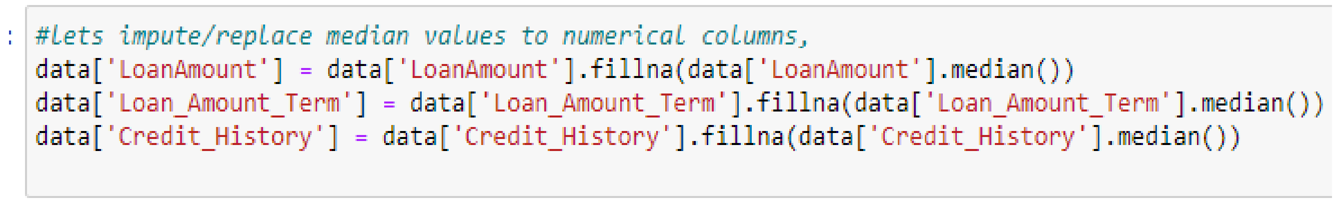
As we can see output, many of the columns in the data set have missing values. In that case, we can impute or replace the missing values with statistical values such as mean, median, or mode.

Imputing Missing Values: The missing values in the categorical columns are usually imputed or replaced using the **mode** values.



```
#lets impute/replace mode values to categorical columns,
data['Gender'] = data['Gender'].fillna(data['Gender'].mode()[0])
data['Married'] = data['Married'].fillna(data['Married'].mode()[0])
data['Dependents'] = data['Dependents'].fillna(data['Dependents'].mode()[0])
data['Self_Employed'] = data['Self_Employed'].fillna(data['Self_Employed'].mode()[0])
```

The loan amount, loan amount term, and credit history are also identified to be numerical columns. Therefore, all the missing values in these columns will be replaced with the **median** value,



```
: #lets impute/replace median values to numerical columns,
  data['LoanAmount'] = data['LoanAmount'].fillna(data['LoanAmount'].median())
  data['Loan_Amount_Term'] = data['Loan_Amount_Term'].fillna(data['Loan_Amount_Term'].median())
  data['Credit_History'] = data['Credit_History'].fillna(data['Credit_History'].median())
```

Let's see how many missing values there are in the dataset.

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

```
Loan_ID              0
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
```

As we can see, the output is zero, indicating that we have successfully replaced all of the missing values in the dataset.

Visualization Outliers: Descriptive statistics shown that, the columns applicant income, co-applicant income, and loan amount contain outliers. Let's use the box plot to visualize the outliers in these columns. We can see that there are a lot of outliers in these columns, so let's remove them from the dataset to make our data clean for modeling.

```
#Visualize the outlier using box plot,


plt.style.use('ggplot')
plt.rcParams['figure.figsize'] = (7, 6)

plt.subplot(2, 2, 1)
sns.boxplot(data['ApplicantIncome'])
plt.xlabel('Applicant Income', fontsize=10)

plt.subplot(2, 2, 2)
sns.boxplot(data['CoapplicantIncome'])
plt.xlabel('Coapplicant Income', fontsize=10)

plt.subplot(2, 2, 3)
sns.boxplot(data['LoanAmount'])
plt.xlabel('Loan Amount', fontsize=10)

plt.suptitle('Findings Outliers in Data')
plt.show()
```

Output:

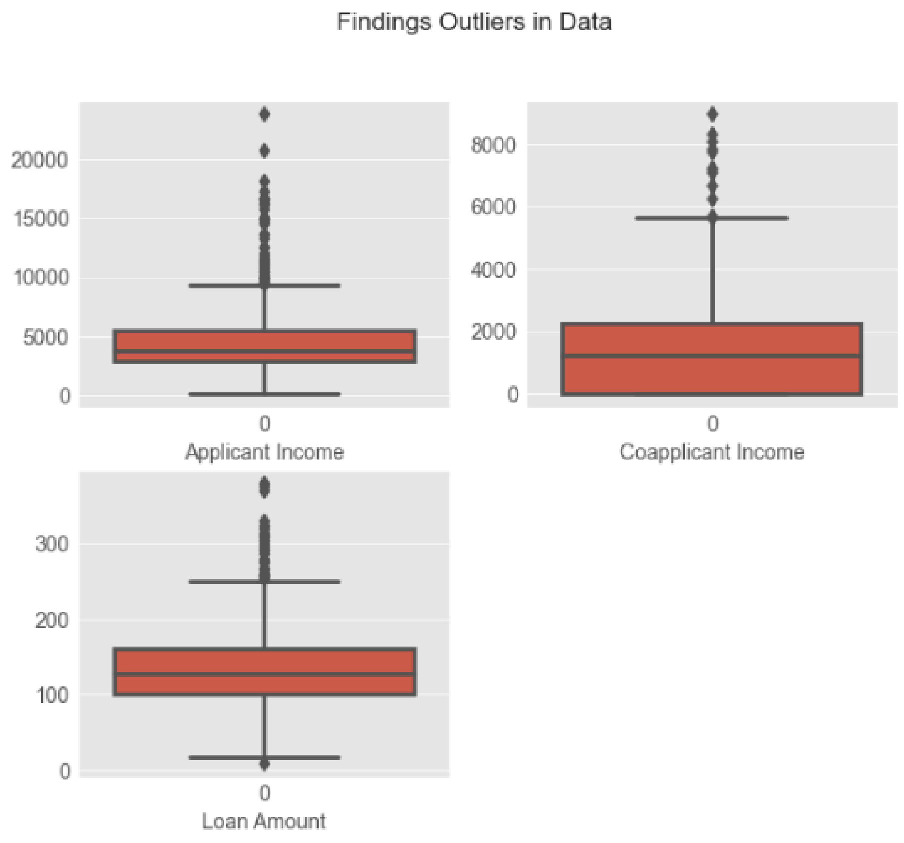
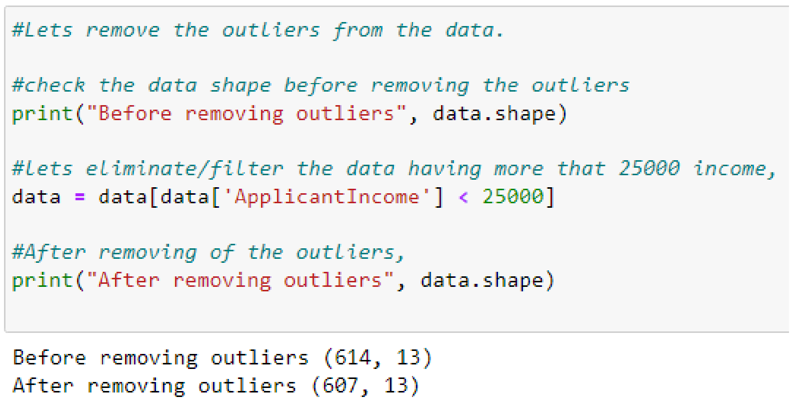


Figure 28: Box plot help us to visualization the outliers.

We can see that there are a lot of outliers in these columns, so let's remove them from the data set to make our data clean for modeling.

Remove Outliers: First, let's remove all the records from the applicant income column that have a value more than 25,000 from the data,

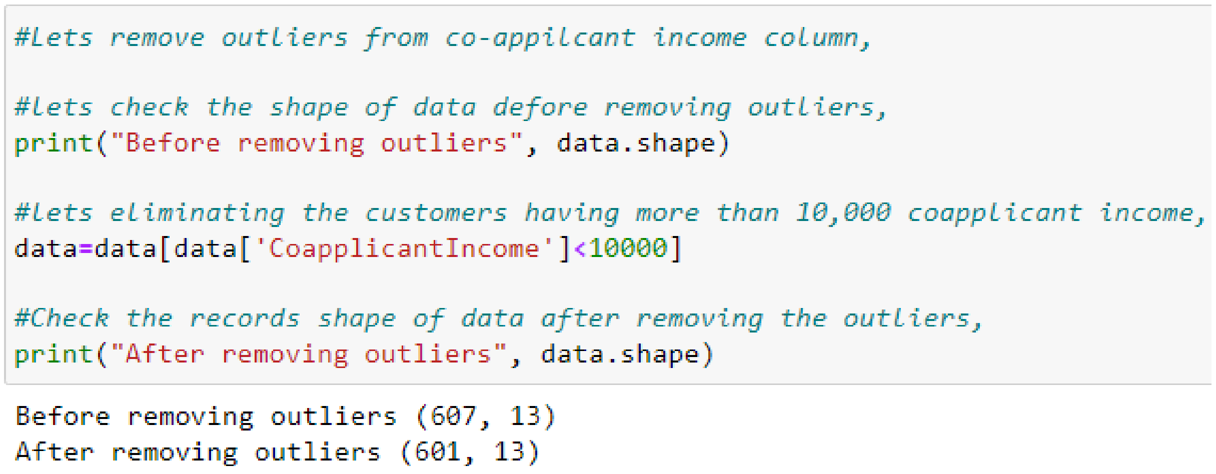


```
#Lets remove the outliers from the data.

#check the data shape before removing the outliers
print("Before removing outliers", data.shape)

#Lets eliminate/filter the data having more that 25000 income,
data = data[data['ApplicantIncome'] < 25000]

#After removing of the outliers,
print("After removing outliers", data.shape)
```

```
Before removing outliers (614, 13)
After removing outliers (607, 13)
```

We can observe that the total number of records before eliminating these records was 614. After eliminating it became 607, indicating that there are seven outliers in this column.

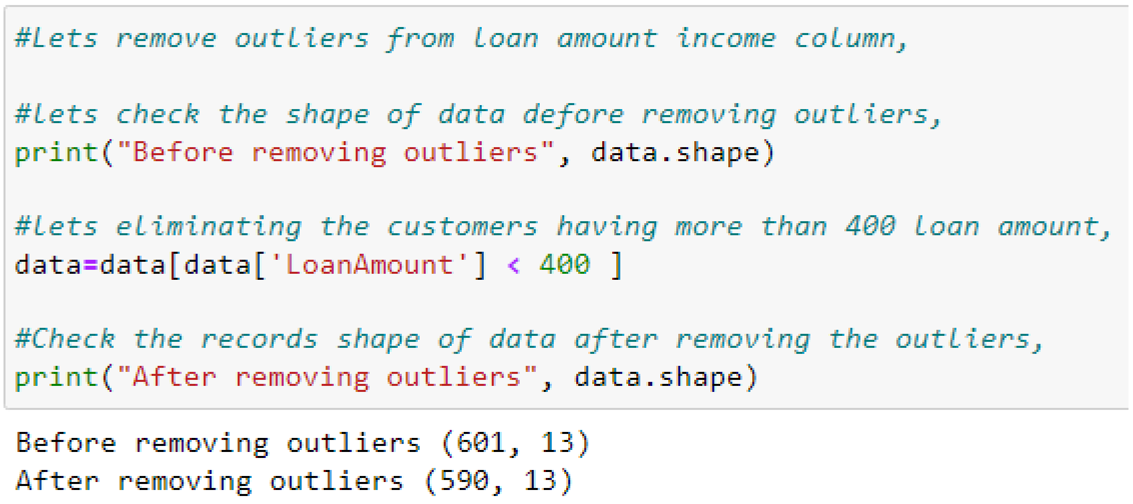
After that, we will eliminate any records with values more than 10,000 from the co-applicant income,



```
#Lets remove outliers from co-appilcant income column,

#lets check the shape of data defore removing outliers,
print("Before removing outliers", data.shape)

#lets eliminating the customers having more than 10,000 coapplicant income,
data=data[data['CoapplicantIncome']<10000]

#Check the records shape of data after removing the outliers,
print("After removing outliers", data.shape)
```

```
Before removing outliers (607, 13)
After removing outliers (601, 13)
```

Finally, we will remove any records with loan amount values more than 400,



```
#Lets remove outliers from loan amount income column,

#lets check the shape of data defore removing outliers,
print("Before removing outliers", data.shape)

#lets eliminating the customers having more than 400 loan amount,
data=data[data['LoanAmount'] < 400 ]

#Check the records shape of data after removing the outliers,
print("After removing outliers", data.shape)
```

```
Before removing outliers (601, 13)
After removing outliers (590, 13)
```

After eliminating all of the outliers from the dataset, the total number of records in the dataset is 590.

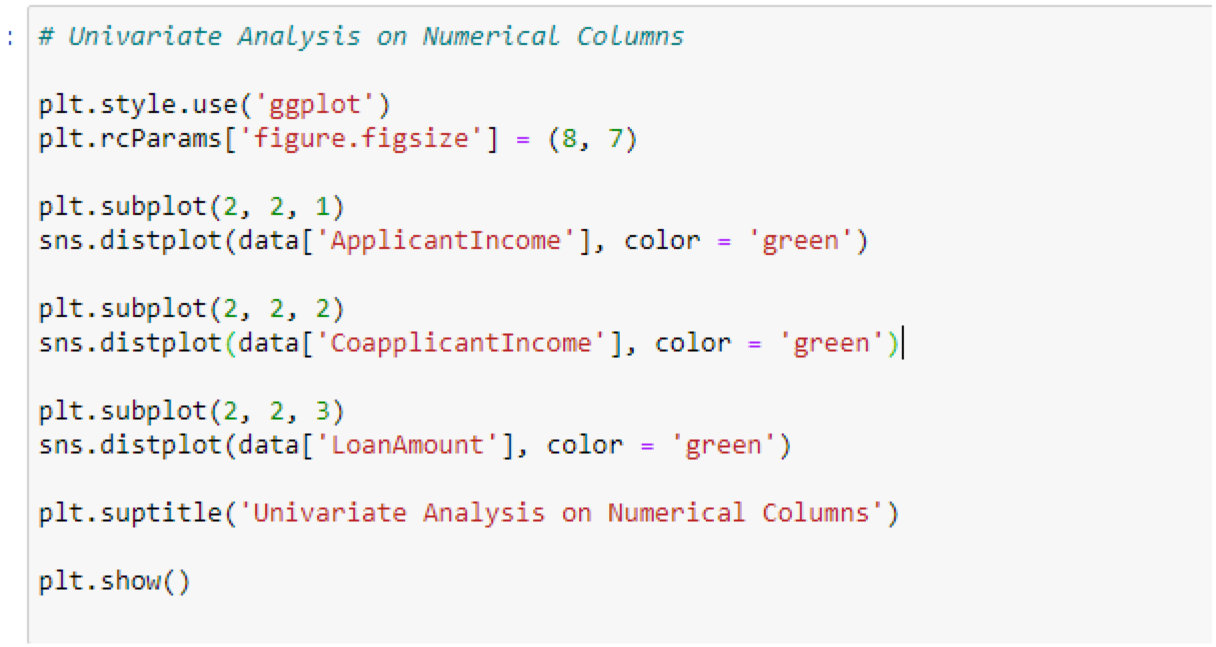
Distribution of data using univariate analysis:[37] In our dataset, we may perform a univariate analysis and discover some interesting facts about the features in the details.

- Univariate analysis is the most basic type of statistical analysis, and it can be influencing or descriptive.
- The important point is that just one variable is involved.
- When multivariate analysis is more suited, if univariate analysis can produce misleading results.

This is a necessary step in understanding the variables in the data set,

- Analyze the numerical column distribution in the dataset.
- Use pie charts and count plots to analyze the distribution of categorical columns.
- Use pie charts when there are few categories in the categorical column and count plots when there are many.

Let's check the distribution of three numerical columns first: applicant income, co-applicant income, and loan amount. To plot all three of these columns together, we are using the **subplot()** function from the matplotlib library, which is part of the seaborn library for data visualization,



```
# Univariate Analysis on Numerical Columns

plt.style.use('ggplot')
plt.rcParams['figure.figsize'] = (8, 7)

plt.subplot(2, 2, 1)
sns.distplot(data['ApplicantIncome'], color = 'green')

plt.subplot(2, 2, 2)
sns.distplot(data['CoapplicantIncome'], color = 'green')

plt.subplot(2, 2, 3)
sns.distplot(data['LoanAmount'], color = 'green')

plt.suptitle('Univariate Analysis on Numerical Columns')

plt.show()
```

Output:

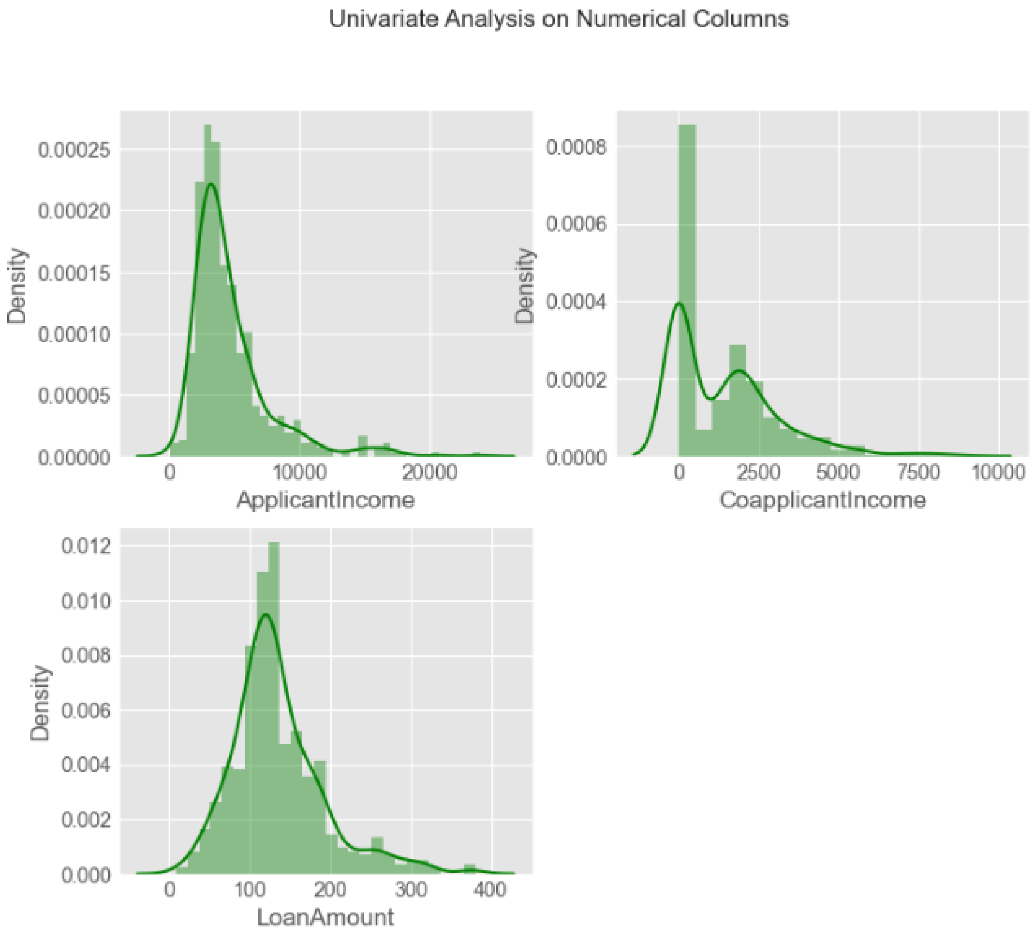
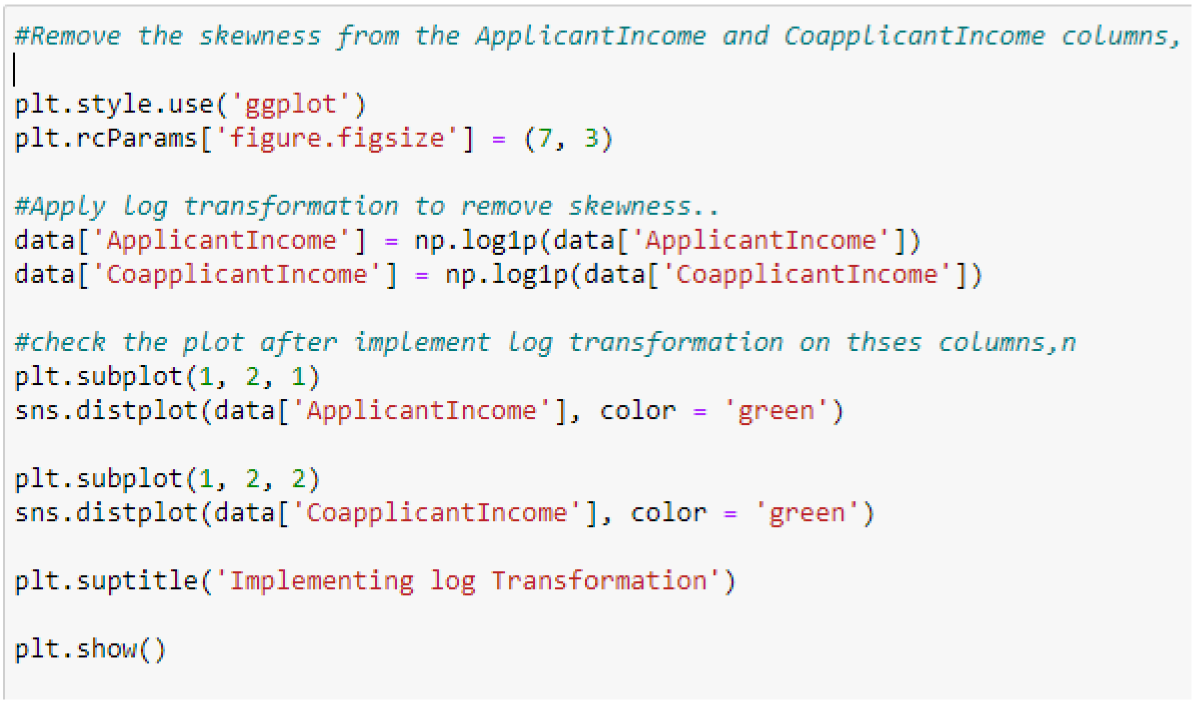


Figure 29: Distribution of Numerical Columns (applicant income, co-applicant income and loan amount).

We can see the findings, and one key finding is that all these charts are skewed in nature and can lead to misleading results when doing predictive modeling. As a result, we must use various transformation techniques to eliminate skewness from the data.

Remove Skewness using Log Transform: In this scenario, we will utilize the log transform to eliminate the skewness from these columns.



```
#Remove the skewness from the ApplicantIncome and CoapplicantIncome columns,

plt.style.use('ggplot')
plt.rcParams['figure.figsize'] = (7, 3)

#Apply log transformation to remove skewness..
data['ApplicantIncome'] = np.log1p(data['ApplicantIncome'])
data['CoapplicantIncome'] = np.log1p(data['CoapplicantIncome'])

#check the plot after implement log transformation on thses columns,n
plt.subplot(1, 2, 1)
sns.distplot(data['ApplicantIncome'], color = 'green')

plt.subplot(1, 2, 2)
sns.distplot(data['CoapplicantIncome'], color = 'green')

plt.suptitle('Implementing log Transformation')

plt.show()
```

Output:

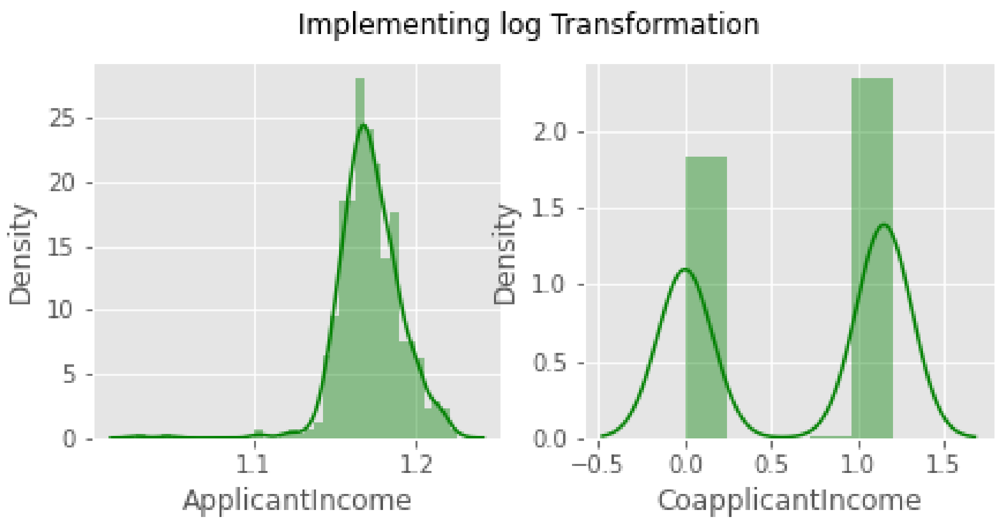
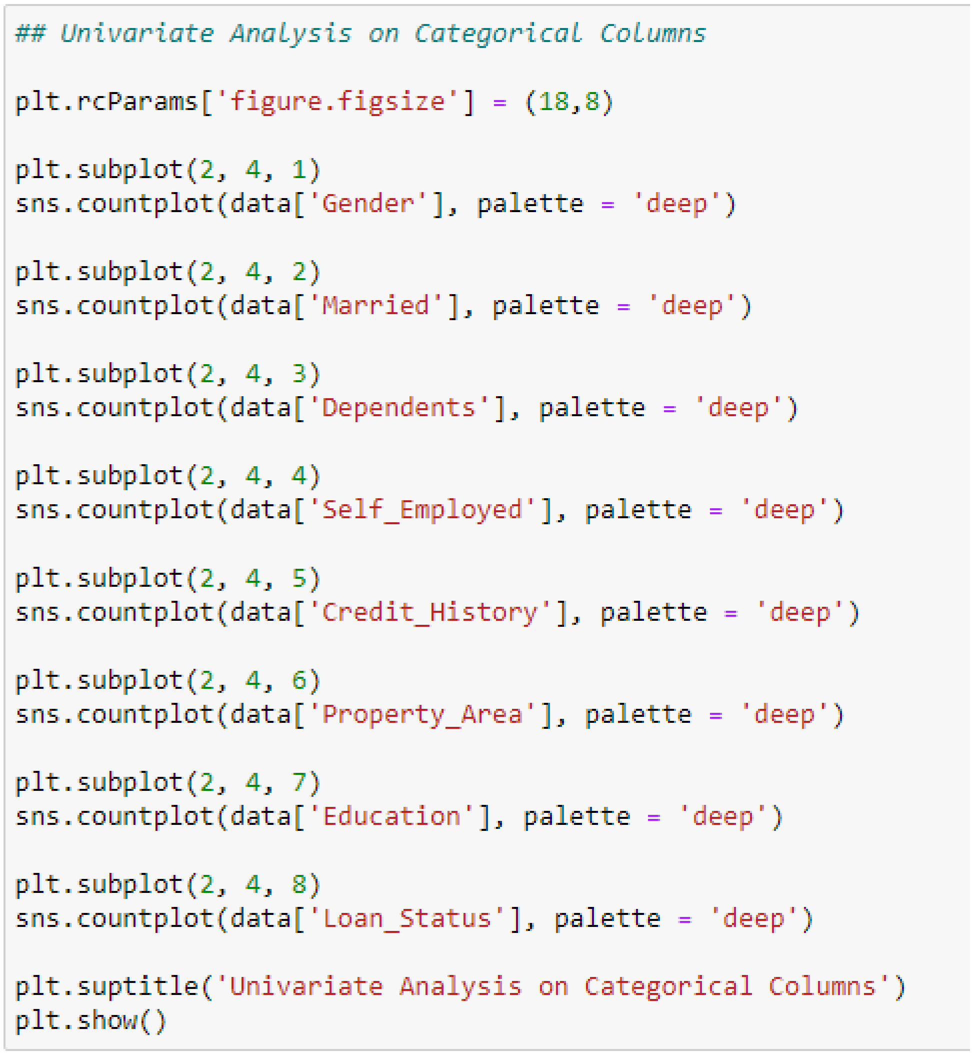


Figure 30: Applicant income and co-applicant income depicting normal distribution after log transform.

These distributions now closely look like the normal distribution. As a result, we have successfully overcome the skewness problem from numerical columns.

Next, we are going to check the distribution of categorical columns present in the detail set using the **countplot** function with **subplot**,



```
## Univariate Analysis on Categorical Columns

plt.rcParams['figure.figsize'] = (18,8)

plt.subplot(2, 4, 1)
sns.countplot(data['Gender'], palette = 'deep')

plt.subplot(2, 4, 2)
sns.countplot(data['Married'], palette = 'deep')

plt.subplot(2, 4, 3)
sns.countplot(data['Dependents'], palette = 'deep')

plt.subplot(2, 4, 4)
sns.countplot(data['Self_Employed'], palette = 'deep')

plt.subplot(2, 4, 5)
sns.countplot(data['Credit_History'], palette = 'deep')

plt.subplot(2, 4, 6)
sns.countplot(data['Property_Area'], palette = 'deep')

plt.subplot(2, 4, 7)
sns.countplot(data['Education'], palette = 'deep')

plt.subplot(2, 4, 8)
sns.countplot(data['Loan_Status'], palette = 'deep')

plt.suptitle('Univariate Analysis on Categorical Columns')
plt.show()
```

Output:

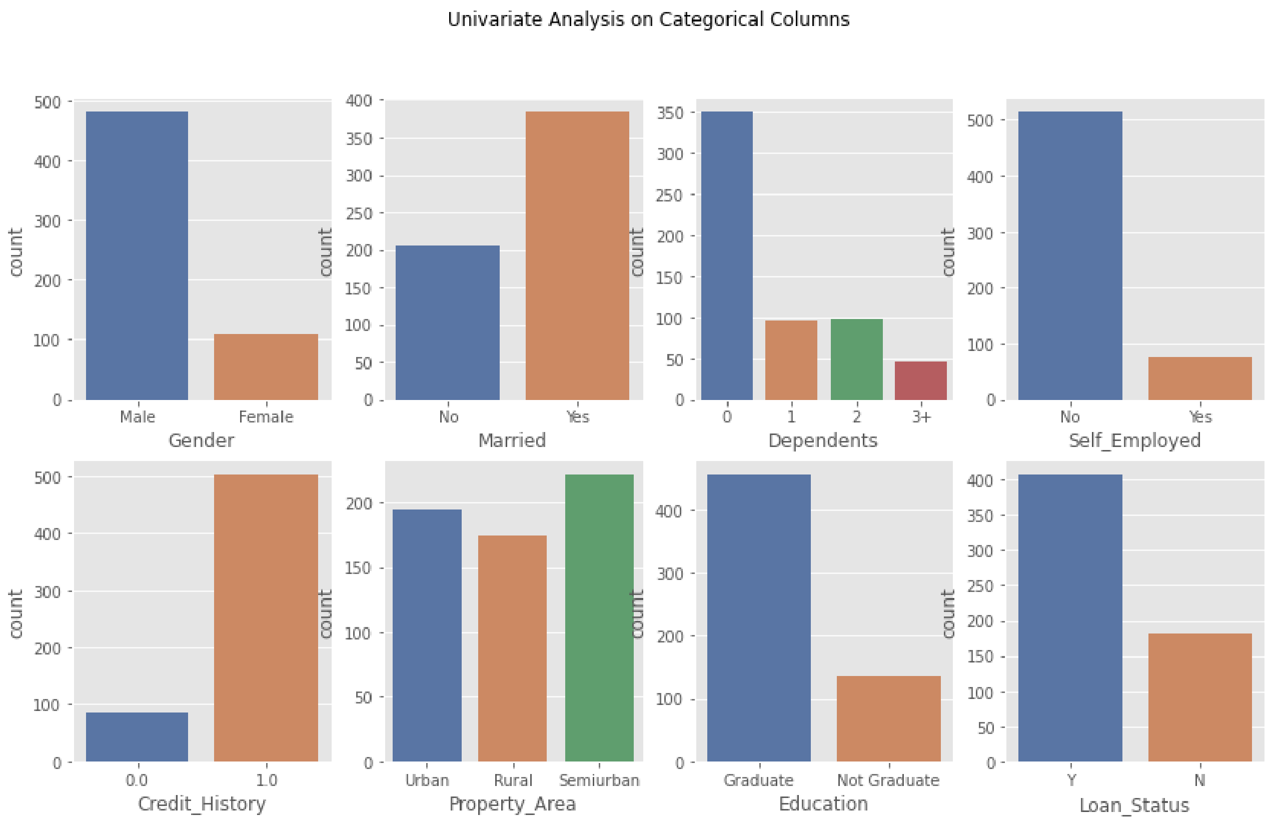


Figure 31: Visualization of distribution of categorical columns on dataset using bar plot.

We can observe the distribution of different categories in several columns. As we can observe, there are more male applications than female applicants. Similar to how married applications are more significant than single applicants, etc. Such a large variety of hidden patterns exist.

Using bivariate analysis to determine relationships between variables:[37] Bivariate analysis is one of the most basic types of quantitative analysis. It involves two variables to determine their empirical relationship and is useful in testing simple hypotheses of association.

To understand the relationship between two variables in a data set, three types of bivariate analysis can be used,

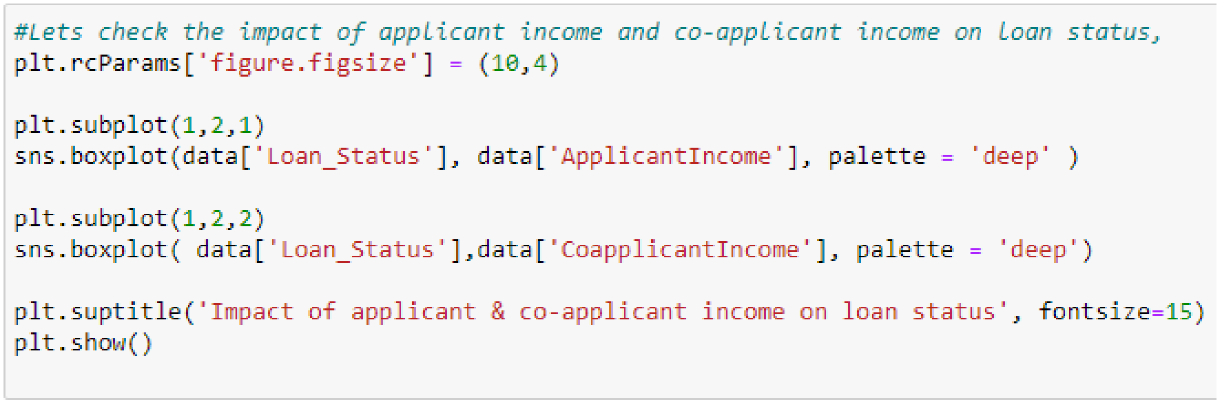
1. Categorical vs Numerical

2. Categorical vs Categorical

3. Numerical vs Numerical

Categorical vs Numerical columns analysis: As we know, our target column is categorical, and we will perform bivariate analysis only on the target data. So, first, we will analyze the impact of numerical columns on the target column using categorical vs numerical analysis.

Using a box plot, we will analyze the impact of applicant and co-applicant income on loan status.



```
#Lets check the impact of applicant income and co-applicant income on loan status,
plt.rcParams['figure.figsize'] = (10,4)

plt.subplot(1,2,1)
sns.boxplot(data['Loan_Status'], data['ApplicantIncome'], palette = 'deep' )

plt.subplot(1,2,2)
sns.boxplot( data['Loan_Status'],data['CoapplicantIncome'], palette = 'deep')

plt.suptitle('Impact of applicant & co-applicant income on loan status', fontsize=15)
plt.show()
```

Output:

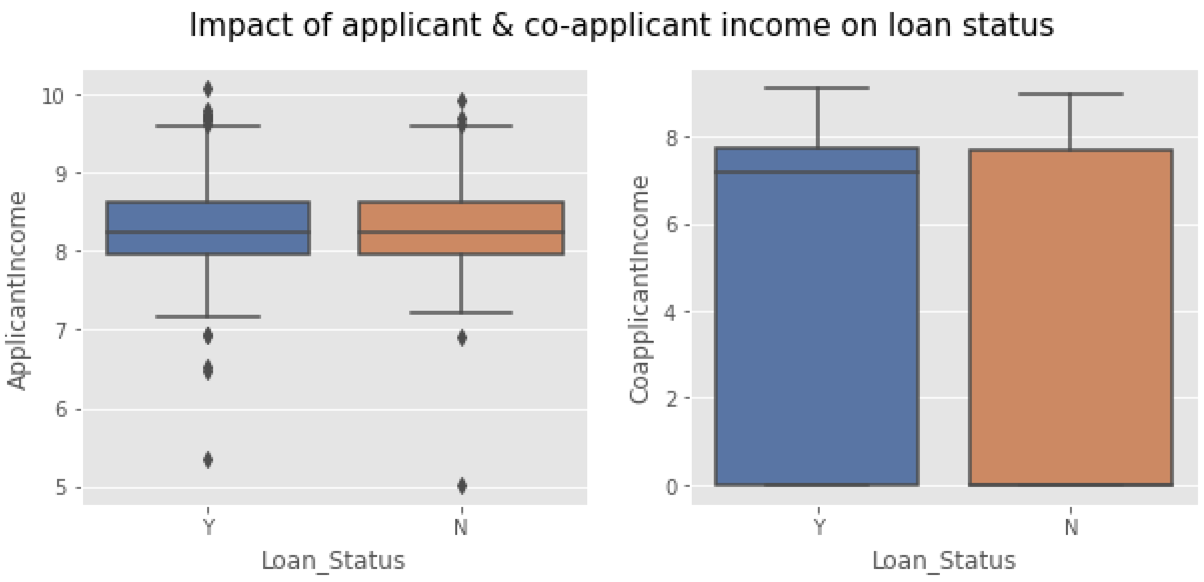
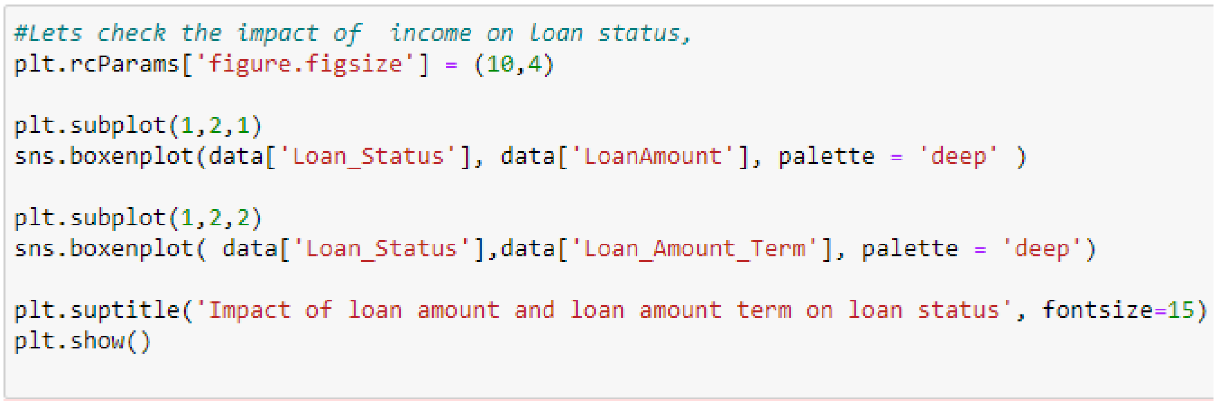


Figure 32: Impact of applicant and co-applicant income on loan status using box plot.

We can see from the plots that there is no clear pattern. Let's use the boxen plot again to analyze the impact of loan amount and loan amount term,



```
#Lets check the impact of  income on loan status,
plt.rcParams['figure.figsize'] = (10,4)

plt.subplot(1,2,1)
sns.boxenplot(data['Loan_Status'], data['LoanAmount'], palette = 'deep' )

plt.subplot(1,2,2)
sns.boxenplot( data['Loan_Status'],data['Loan_Amount_Term'], palette = 'deep')

plt.suptitle('Impact of loan amount and loan amount term on loan status', fontsize=15)
plt.show()
```

Output:

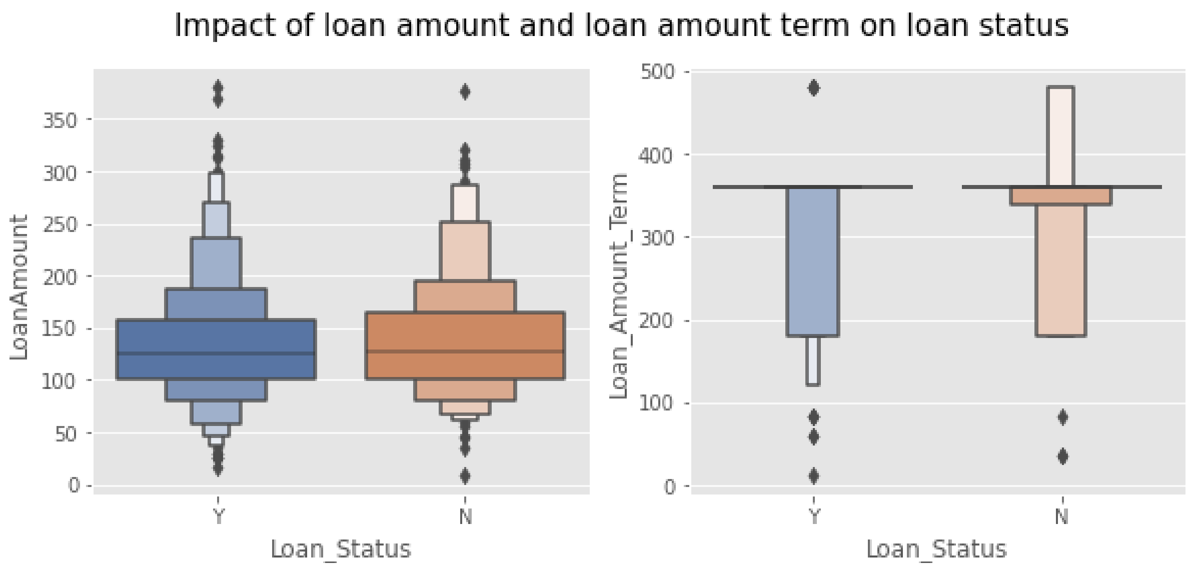
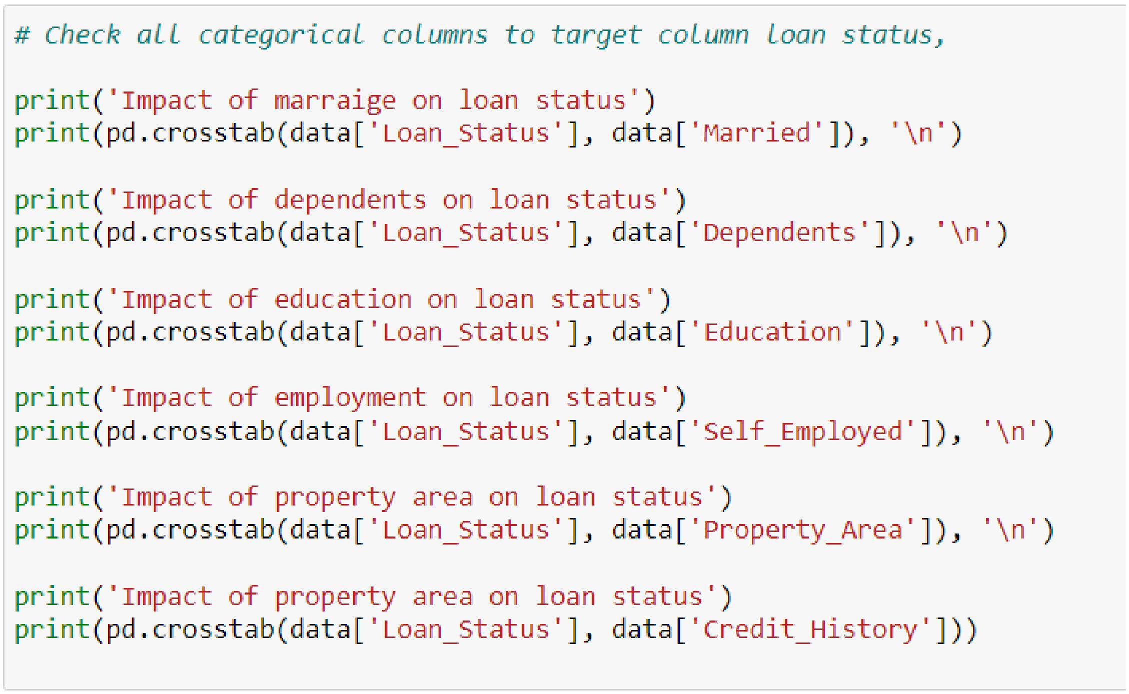


Figure 33: Impact of loan amount and loan amount term on loan status.

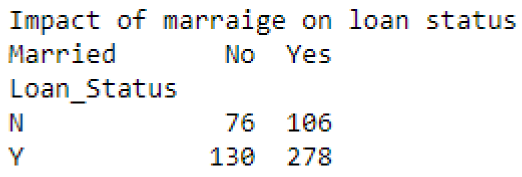
We can see that the loan amount has no clear pattern, but it is clear that the longer the loan amount term, the lower chances of getting a loan approved. So, we analyzed the effect of numerical columns on the target column (Loan_Status).

Categorical vs Categorical columns analysis: Now, we are checking the impact of categorical columns on the target column using the **crosstab()** function, which returns across tabulation of two categorical columns,

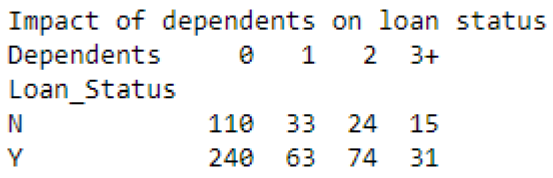


```
# Check all categorical columns to target column loan status,

print('Impact of marraige on loan status')
print(pd.crosstab(data['Loan_Status'], data['Married']), '\n')

print('Impact of dependents on loan status')
print(pd.crosstab(data['Loan_Status'], data['Dependents']), '\n')

print('Impact of education on loan status')
print(pd.crosstab(data['Loan_Status'], data['Education']), '\n')

print('Impact of employment on loan status')
print(pd.crosstab(data['Loan_Status'], data['Self_Employed']), '\n')

print('Impact of property area on loan status')
print(pd.crosstab(data['Loan_Status'], data['Property_Area']), '\n')

print('Impact of property area on loan status')
print(pd.crosstab(data['Loan_Status'], data['Credit_History']))
```

Output:



```
Impact of marraige on loan status
Married        No  Yes
Loan_Status
N              76  106
Y             130  278
```



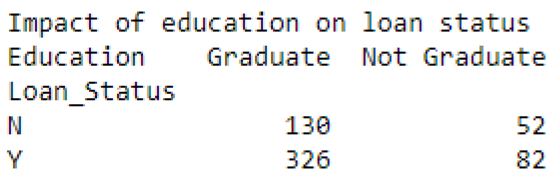
```
Impact of dependents on loan status
Dependents      0   1   2  3+
Loan_Status
N             110  33  24  15
Y             240  63  74  31
```

According to the first column, married candidates had their loan rejected 76 times and approved 130 times. Similarly, the married candidates were approved 278 times and rejected 130 times.

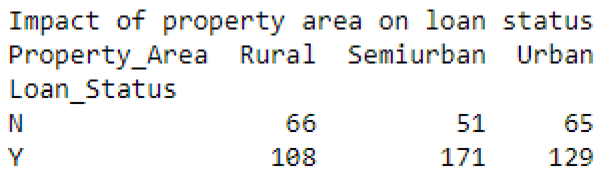
We can now understand the impact of marriage on loan status, even though the difference in loan eligibility between married and unmarried candidates is very small.

However, married candidates have an opportunity over unmarried candidates. Likewise, there are relationships between all of the columns and the target column like graduated applicant more eligible than the non-graduate applicants.

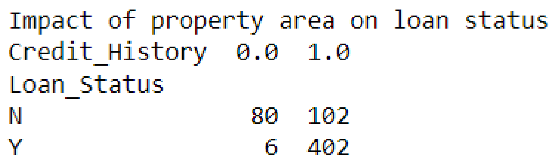
Output:



```
Impact of education on loan status
Education    Graduate  Not Graduate
Loan_Status
N                 130            52
Y                 326            82
```



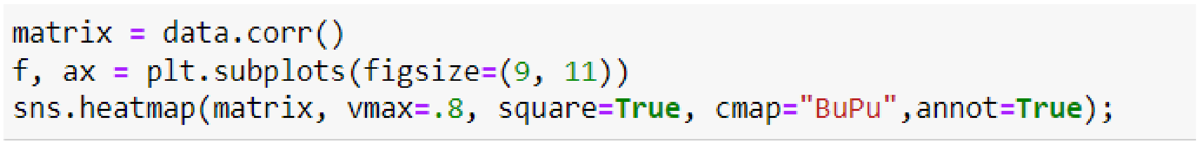
```
Impact of property area on loan status
Property_Area  Rural  Semiurban  Urban
Loan_Status
N                 66         51     65
Y                108        171    129
```



```
Impact of property area on loan status
Credit_History  0.0  1.0
Loan_Status
N                80  102
Y                 6  402
```

We can see that in the dependent column. The impact of having 1, 2, 3+ dependents is quite similar. That means we can merge these three categories. Also, we can observe that educated applicants are eligible to grant to loan and applicant with credit history 1.0 and loan approved 402 time, it seems more influential factor to approve loan. There are so many patterns hidden in these cross tabulations to understand data insights.

Visualization of Correlation: Let us now analyze the correlation between all variables. To visualize the correlation, we will use a heat map. Heatmaps use color variations to visualize data. The variables with a darker color have a stronger correlation.



```python
matrix = data.corr()
f, ax = plt.subplots(figsize=(9, 11))
sns.heatmap(matrix, vmax=.8, square=True, cmap="BuPu",annot=True);
```

Output:

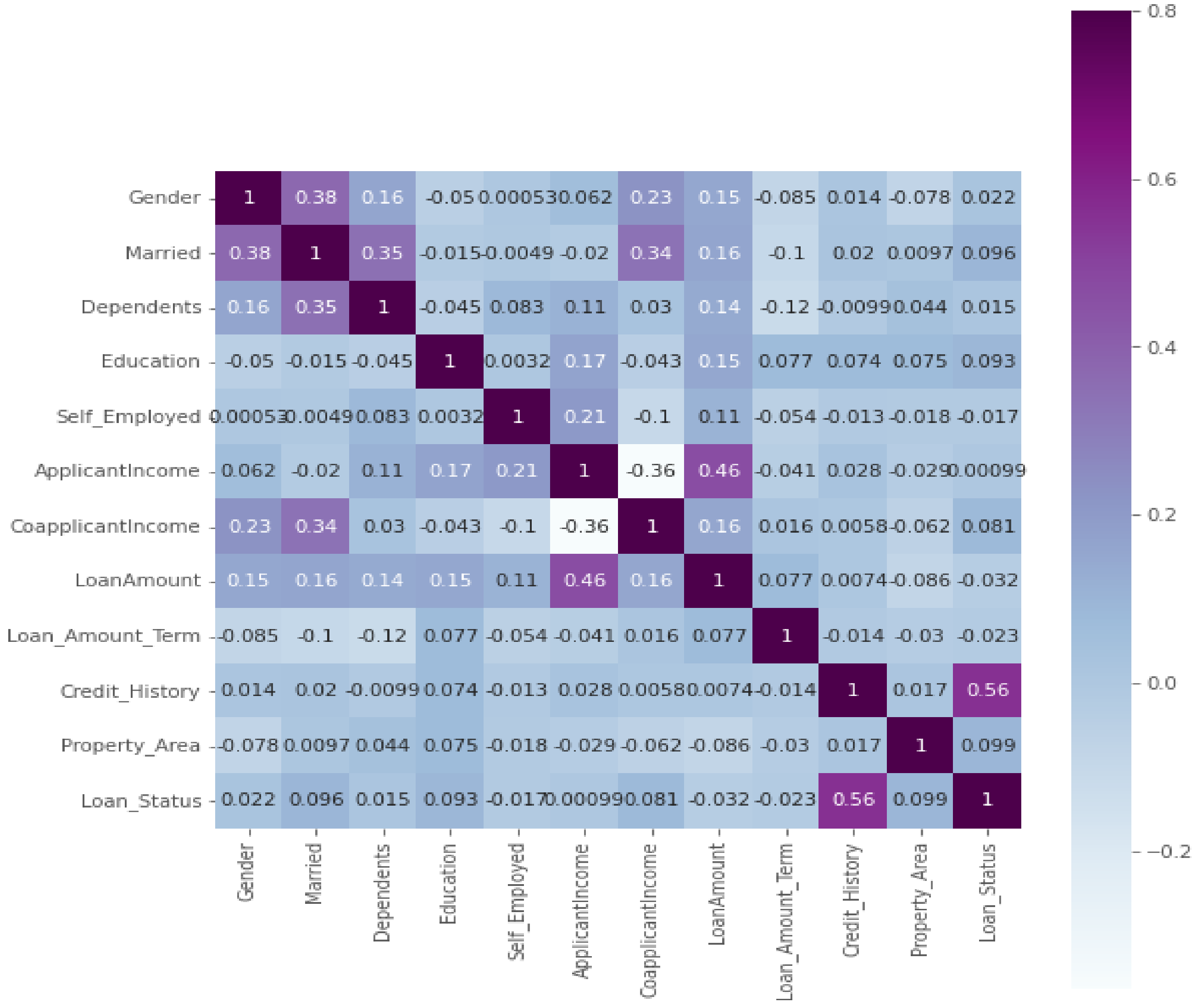
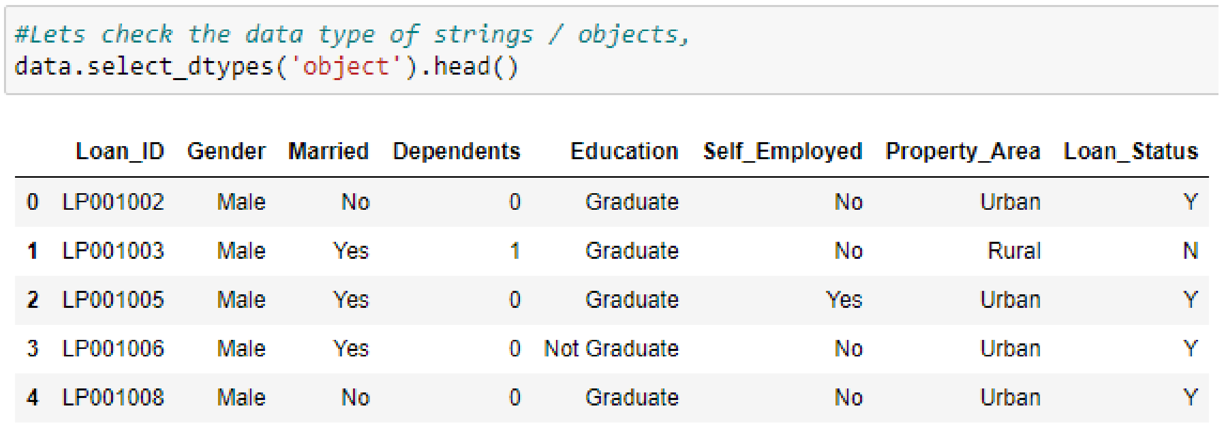


Figure 34: Correlation heatmap for understanding relationship between variables.

We can see that (ApplicantIncome - LoanAmount) and (Credit History - Loan Status) are the most correlated variables.

5.1.4 Data Preparation for Modelling

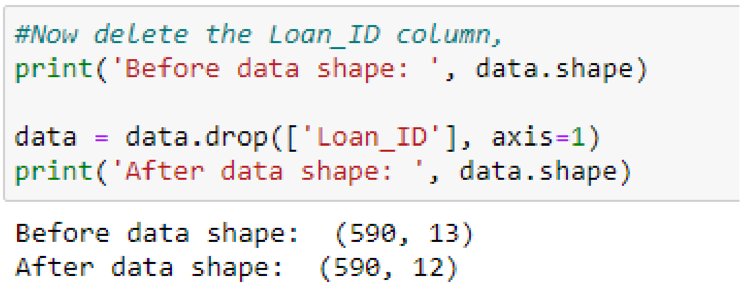
Now we will prepare the data for fitting into a machine learning model. We already know that strings cannot be accepted by a machine learning model. For that, we'll need to encode all of the categorical columns.



```python
#Lets check the data type of strings / objects,
data.select_dtypes('object').head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	Urban	Y
4	LP001008	Male	No	0	Graduate	No	Urban	Y

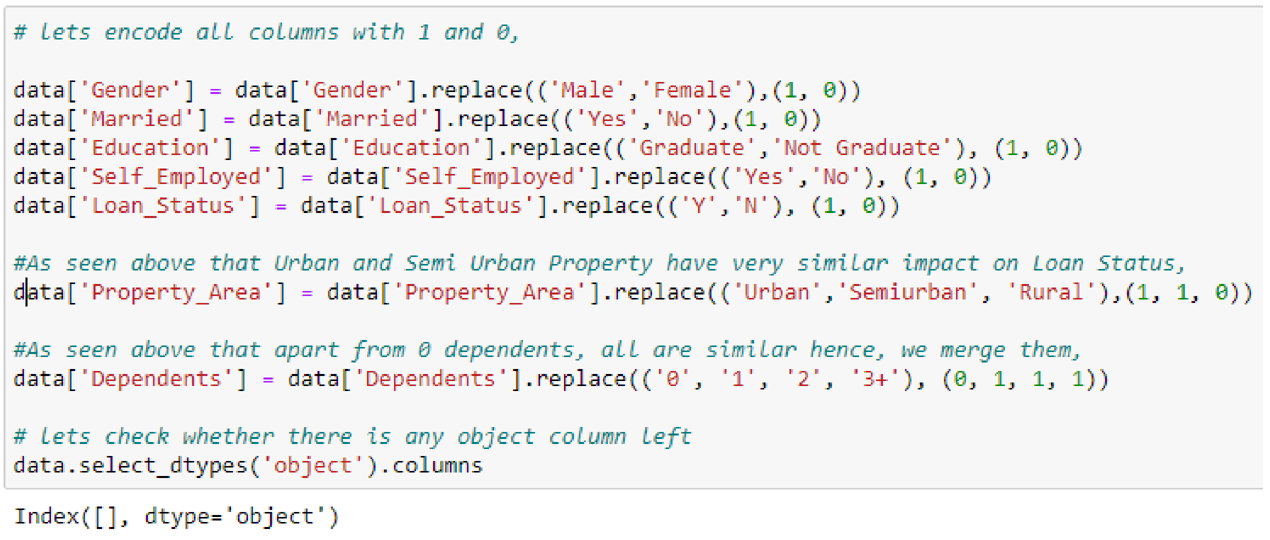
Remove Loan_ID Column: We can see that there is a loan ID column, which is useless for predicting a candidate's loan status. So, let's use the drop function to remove this column,



```python
#Now delete the Loan_ID column,
print('Before data shape: ', data.shape)

data = data.drop(['Loan_ID'], axis=1)
print('After data shape: ', data.shape)
```

```
Before data shape:  (590, 13)
After data shape:  (590, 12)
```

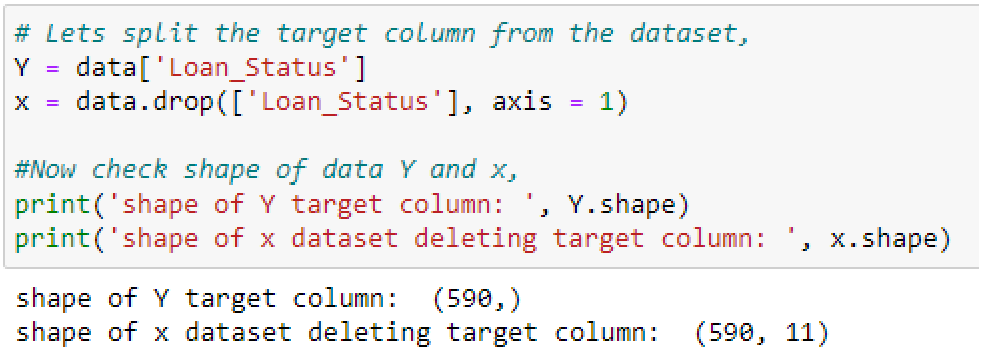
Encoding categorical columns: Let's now start encoding the remaining categorical columns. We will apply the **replace()** function to do that. We can see that the categories of gender, marriage, education, self-employment, and loan status are encoded with a '1' for the first category and a '0' for the second.



```
# lets encode all columns with 1 and 0,

data['Gender'] = data['Gender'].replace(('Male','Female'),(1, 0))
data['Married'] = data['Married'].replace(('Yes','No'),(1, 0))
data['Education'] = data['Education'].replace(('Graduate','Not Graduate'), (1, 0))
data['Self_Employed'] = data['Self_Employed'].replace(('Yes','No'), (1, 0))
data['Loan_Status'] = data['Loan_Status'].replace(('Y','N'), (1, 0))

#As seen above that Urban and Semi Urban Property have very similar impact on Loan Status,
data['Property_Area'] = data['Property_Area'].replace(('Urban','Semiurban', 'Rural'),(1, 1, 0))

#As seen above that apart from 0 dependents, all are similar hence, we merge them,
data['Dependents'] = data['Dependents'].replace(('0', '1', '2', '3+'), (0, 1, 1, 1))

# lets check whether there is any object column left
data.select_dtypes('object').columns
```

```
Index([], dtype='object')
```

Separate target column from dataset: Finally, we will split the target column from the dataset. It's a significant stage because if it's left in the data set, the machine learning model will pick up all the patterns from the target column as well. But as we all know, the target column is absent in real-world cases. We employ these prediction models for that reason. We can see that we have been successful in separating the target column from the data and placing it in a variable named Y.

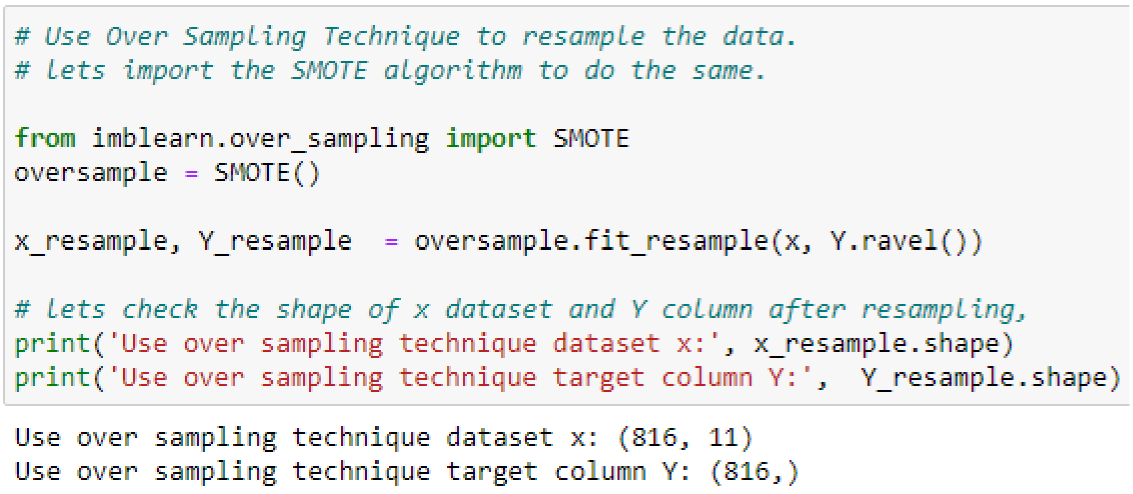


```
# Lets split the target column from the dataset,
Y = data['Loan_Status']
x = data.drop(['Loan_Status'], axis = 1)

#Now check shape of data Y and x,
print('shape of Y target column: ', Y.shape)
print('shape of x dataset deleting target column: ', x.shape)
```

```
shape of Y target column:  (590,)
shape of x dataset deleting target column:  (590, 11)
```

Resampling: We previously observed that the target column is highly imbalanced. We must balance the data using statistical approaches. We may resample the data using a variety of statistical approaches. Resampling is a statistical method that includes drawing repeated samples from the original samples, such as over sampling, cluster-based sampling, and under sampling. In data analysis, oversampling and under sampling are techniques for adjusting the class distribution of a data set.

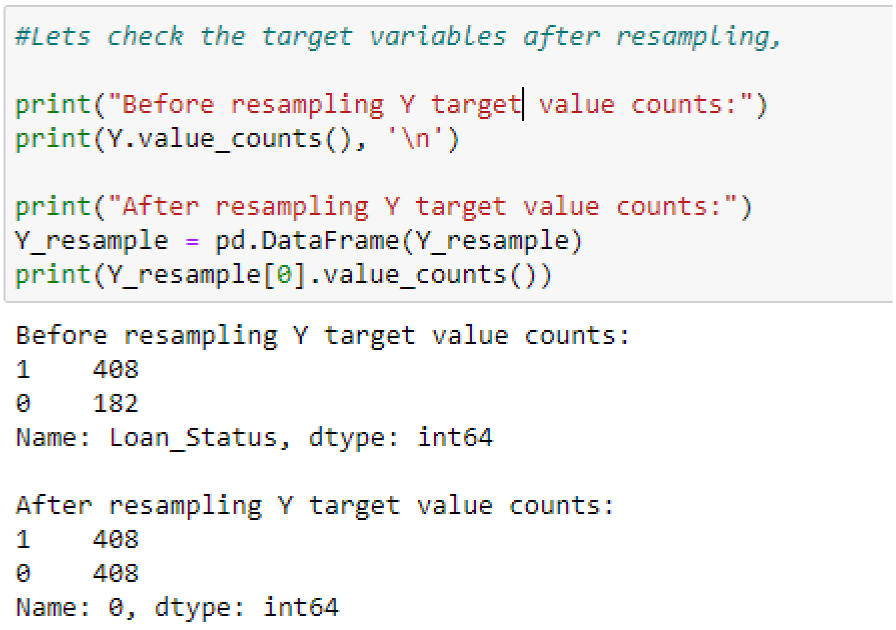
To minimize data loss, we will employ over sampling rather than under sampling in this scenario. The **imblearn** library will be used to perform over sampling. To perform oversampling, we will utilize the SMOTE algorithm from the imblearn library.



```
# Use Over Sampling Technique to resample the data.
# lets import the SMOTE algorithm to do the same.

from imblearn.over_sampling import SMOTE
oversample = SMOTE()

x_resample, Y_resample  = oversample.fit_resample(x, Y.ravel())

# lets check the shape of x dataset and Y column after resampling,
print('Use over sampling technique dataset x:', x_resample.shape)
print('Use over sampling technique target column Y:',  Y_resample.shape)
```

```
Use over sampling technique dataset x: (816, 11)
Use over sampling technique target column Y: (816,)
```

After implementing an over sampling, we will count the number of applicants whose loan has been refused and accepted to determine the discrepancy.



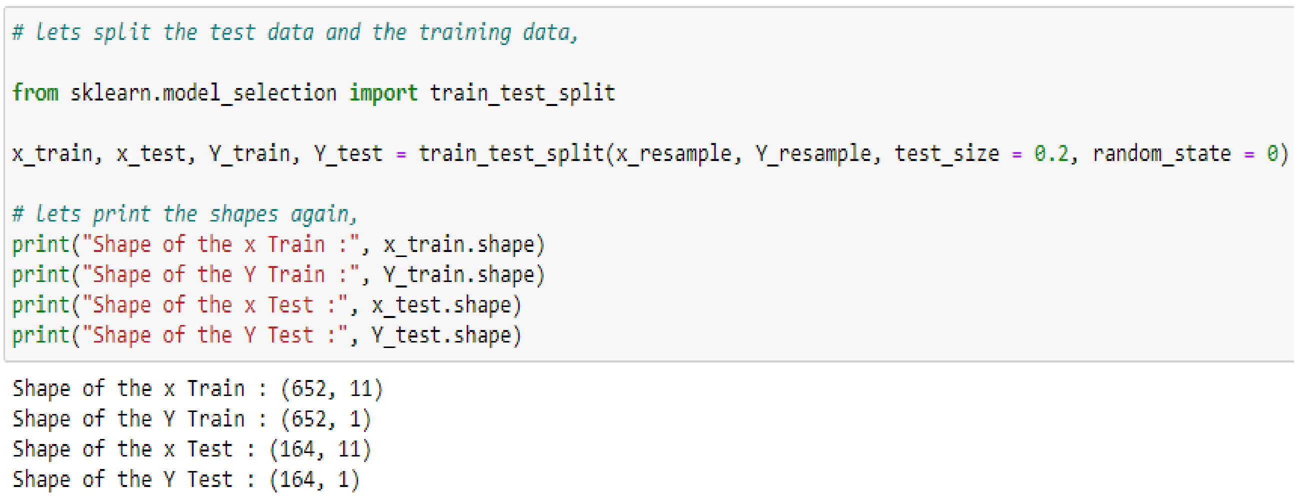
```
#Lets check the target variables after resampling,

print("Before resampling Y target value counts:")
print(Y.value_counts(), '\n')

print("After resampling Y target value counts:")
Y_resample = pd.DataFrame(Y_resample)
print(Y_resample[0].value_counts())
```

```
Before resampling Y target value counts:
1    408
0    182
Name: Loan_Status, dtype: int64

After resampling Y target value counts:
1    408
0    408
Name: 0, dtype: int64
```

We can clearly see that, before to the resampling, we had 408 applicants whose loan applications got accepted but only 108 candidates whose loan applications got rejected.

However, after resampling, the number of applicants who had their loan application granted and refused remained at 408.

Prepare train and test dataset for modeling: Finally, We also split the dataset into train and test using the **train_test_split()** utility from scikit-learn, and check the shape of all newly formed dataset to ensure they are correctly formed.



```
# lets split the test data and the training data,

from sklearn.model_selection import train_test_split

x_train, x_test, Y_train, Y_test = train_test_split(x_resample, Y_resample, test_size = 0.2, random_state = 0)

# lets print the shapes again,
print("Shape of the x Train :", x_train.shape)
print("Shape of the Y Train :", Y_train.shape)
print("Shape of the x Test :", x_test.shape)
print("Shape of the Y Test :", Y_test.shape)
```

```
Shape of the x Train : (652, 11)
Shape of the Y Train : (652, 1)
Shape of the x Test : (164, 11)
Shape of the Y Test : (164, 1)
```

We can observe that the x train and Y train contain 652 records each, but the x test and Y test have 164 records each. Also, the Y train and Y test contain only one column, which is the target column.

5.1.5 Modeling and Evaluation

Logistic Regression Modeling: Now, we are going to train dataset on a machine learning predictive model. We will implement logistic regression algorithm to logistic to train the model. Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable. Now let's import the logistic regression algorithm from **sklearn.linear_model** package. After that, we will define a model and then train our training dataset that is x train and Y train using the **.fit()** function.

```
# lets apply Logistic Regression

from sklearn.linear_model import LogisticRegression

model = LogisticRegression(random_state = 0)
model.fit(x_train, Y_train)

y_pred = model.predict(x_test)

print("Training Accuracy :", model.score(x_train, Y_train))
print("Testing Accuracy :", model.score(x_test, Y_test))
```

```
Training Accuracy : 0.7730061349693251
Testing Accuracy : 0.8170731707317073
```

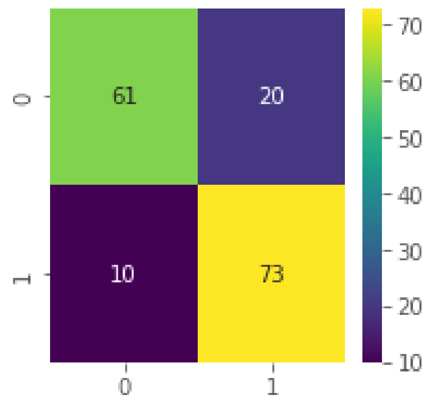
Now, as the model has been trained on the training data. It's time to perform some predictive analysis using the predict function.

We are going to make the predictions on the testing set that is x test. Finally, we can check the training and testing accuracy using the score function. We can see that the training accuracy for a logistic regression model is around 77 percent, whereas the testing accuracy comes out to be around 81 percent.

Performance Metrics for Logistics Regression: Let's import the **confusion_matrix** and **classification_report** from the **sklearn.metrics** package and check them as well,

```
# lets analyze the performance using confusion matrix,

from sklearn.metrics import confusion_matrix, classification_report

cm = confusion_matrix(Y_test, y_pred)
plt.rcParams['figure.figsize'] = (3, 3)
sns.heatmap(cm, annot = True, cmap = 'viridis', fmt = '.8g')
plt.show()

# lets also use classification report for performance analysis,
cr = classification_report(Y_test, y_pred)
print(cr)
```

Output:



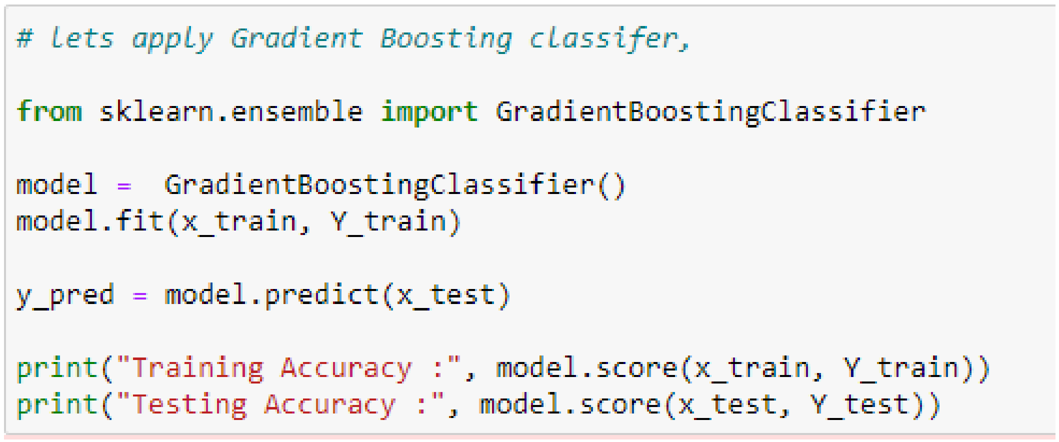
```
              precision    recall  f1-score   support

           0       0.86      0.75      0.80        81
           1       0.78      0.88      0.83        83

    accuracy                           0.82       164
   macro avg       0.82      0.82      0.82       164
weighted avg       0.82      0.82      0.82       164
```

Figure 35: Confusion matrix for logistics regression with accuracy, precision, Recall, F1 Score.

We can see that the logistic regression model recognizes the loan rejection cases incorrectly 20 times while correctly identifying the loan acceptance cases only 10 times. If we look at the precision, we can see that class '0' has a precision of 86%, while class '1' has a precision of 78%. The recall rate for classes "0" and "1" is also 75% and 88%, respectively. This means that logistic regression is not the best model for this data set.

We trained on a logistic regression model previously, and it performed reasonably well. However, we are going to try one more machine learning algorithm that is more advanced than the logistic regression model.

Gradient Boosting Modeling: We are going to try a gradient boosting model and learn more about this model in the boosting model scores. First, let's import the gradient boosting algorithm from **sklearn.ensemble** package. After that will define a model and then train our training data set that is x train and Y train using the **.fit()** function.



```
# lets apply Gradient Boosting classifer,

from sklearn.ensemble import GradientBoostingClassifier

model =  GradientBoostingClassifier()
model.fit(x_train, Y_train)

y_pred = model.predict(x_test)

print("Training Accuracy :", model.score(x_train, Y_train))
print("Testing Accuracy :", model.score(x_test, Y_test))
```

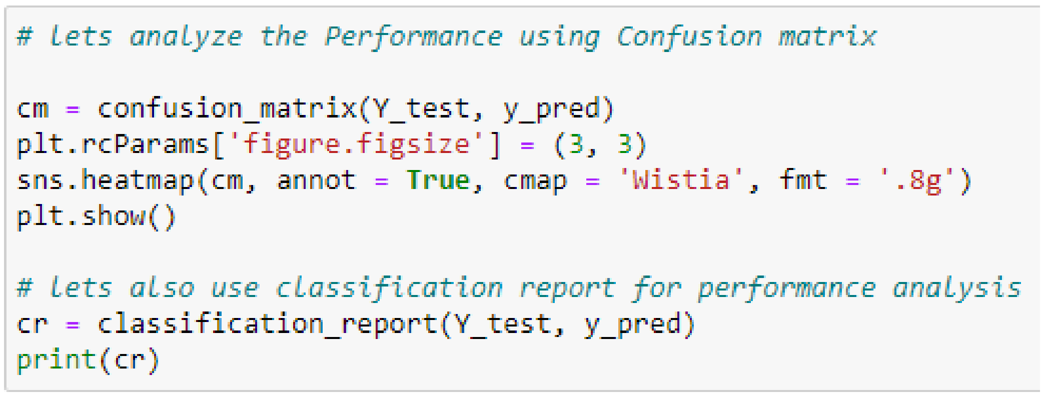
Now as the model has been trained on the training data. It's time to perform some predictive analysis using the predict function. We're going to make the predictions on the testing set, that is x test, finally, we can check the training and testing accuracy using the score function.

Output:

```
Training Accuracy : 0.9187116564417178
Testing Accuracy : 0.8353658536585366
```

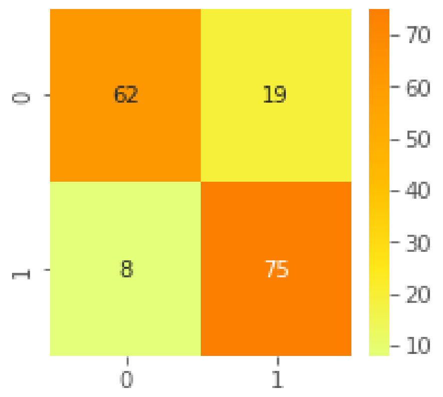
We can see that the training accuracy of a gradient boosting model is slightly higher than that of a logistic regression model.

Performance Metrics for Gradient Boosting: Let us also look at the **confusion_matrix** and **classification_report** to ensure that this model performs better the previous one,



```
# lets analyze the Performance using Confusion matrix

cm = confusion_matrix(Y_test, y_pred)
plt.rcParams['figure.figsize'] = (3, 3)
sns.heatmap(cm, annot = True, cmap = 'Wistia', fmt = '.8g')
plt.show()

# lets also use classification report for performance analysis
cr = classification_report(Y_test, y_pred)
print(cr)
```

Output:



```
              precision    recall  f1-score   support

           0       0.89      0.77      0.82        81
           1       0.80      0.90      0.85        83

    accuracy                           0.84       164
   macro avg       0.84      0.83      0.83       164
weighted avg       0.84      0.84      0.83       164
```

Figure 36: Confusion matrix for Gradient Boosting with accuracy, precision, Recall, F1 Score.

Let us also look at the confusion matrix and classification report to ensure that this model performs better the previous one.

If we look at the precision and recall, they are quite similar. We can check the cross-validation scores also and we can see that the scores are not bearing too much, meaning that the model has been trained well.

```
from sklearn.model_selection import cross_val_score

clf = GradientBoostingClassifier(random_state = 0)
scores = cross_val_score(clf, x_train, Y_train, cv=10)
print(scores)
```

Output:

```
[0.74242424 0.86363636 0.81538462 0.90769231 0.81538462 0.73846154
 0.8        0.8        0.83076923 0.81538462]
```

We cannot expect this model to work very well as the number of records in the training data is very less. This means that gradient boosting is a good model to go with.

Feature Importance: Let us now determine the feature importance, that is, which features are most important to solve the business problem. We will do it with sklearn's feature importance.

```
model_feature_importances = pd.Series(data= model.feature_importances_ ,index=x_train.columns)
model_feature_importances = model_feature_importances.sort_values()
model_feature_importances
```

```
Self_Employed        0.001289
Gender               0.003919
Dependents           0.008078
Loan_Amount_Term     0.018064
Married              0.018688
Education            0.025197
Property_Area        0.036805
CoapplicantIncome    0.056295
ApplicantIncome      0.107861
LoanAmount           0.157355
Credit_History       0.566449
dtype: float64
```

```
fig = px.bar(model_feature_importances)
fig.show()
```

Output:

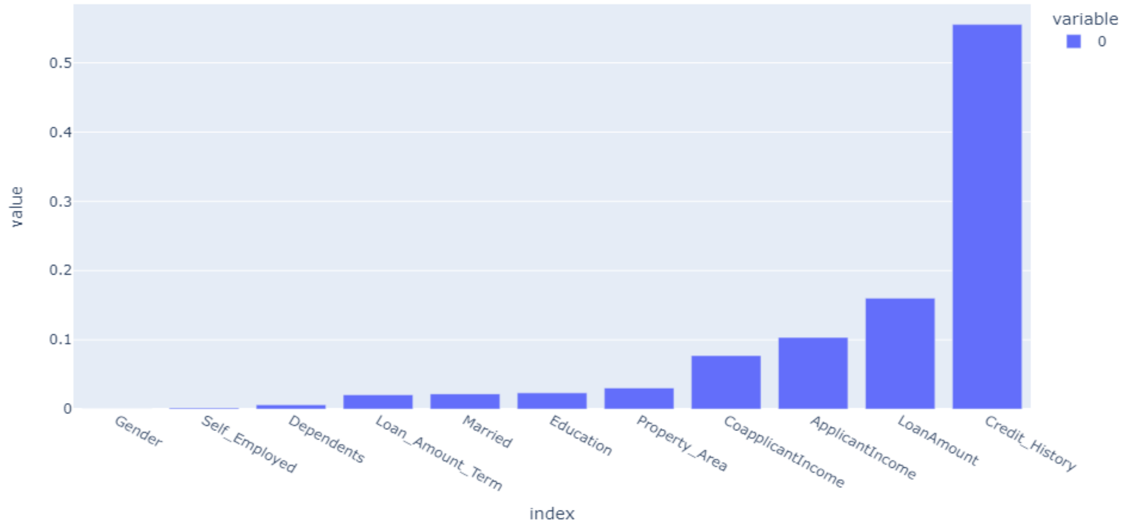


Figure 37: Feature importance obtain from our trained model.

After creating the bar plot and observe that, we can see that the 'Credit History' feature is the most important. As a result, feature engineering guided us in predicting our target variable.

5.1.6 Summaries in Practical Part

01. To clearly understand the business issue statement that must be solved by data mining tasks such as customer segment classification problem and the use of ML models to automate the system or real-time prediction system for finance organization.

02. To evaluate the general frequency of data in a dataset using descriptive statistics.

03. Determine missing data and impute/replace them with statistical values such as mean, median, or mode.

04. Using the box plot visualization, identify outliers and eliminate them from the dataset for smooth bivariate and univariate analysis.

05. In univariate analysis, find the skewness in categorical columns and eliminate it with the log transformation.

06. Use bivariate analysis to discover hidden patterns and the impact of other factors on loan status target value.

07. To use linear regression and gradient boosting techniques, separate the target variable from the dataset as well as the train and test data.

08. Our predictions are almost 0.773 accurate, i.e., we have identified 77% of the loan status correctly for our logistics regression trained model

09. And our predictions are almost 0.918 accurate, i.e., we have identified 91% of the loan status correctly for implemented gradient boosting model.

10. According to the performance matrix gradient boosting model accuracy is better than logistics regression model and which factors are more essential in granting a loan based on our findings credit history on trained ML model.

Finally, Loan prediction is a relatively common real-life challenge that every retail bank experiences. If the model is performed correctly, it can save a lot of man hours at the end of a retail bank.

Although this practical section is primarily designed to provide a walkthrough of the Loan Prediction analysis, we may gain a full understanding of how to solve a classification problem and create a machine learning model to enhance business performance.

6. Conclusion

The goal of the diploma thesis was to research related literature, scientific papers, and online resources to learn about data mining tasks and machine learning algorithms, as well as various Python libraries for understanding and best use cases of ML and DM applicants, as well as determine loan prediction analysis using historical data.

This diploma thesis represents the iterative cycle of the CRISM-DM, DM and ML project, which includes various stages such as accessing datasets, data cleaning, different statistical analysis and visualization, outlier detection and removal, machine learning methods and algorithms, and finally model creation. In the second chapter of this thesis, I explored the numerous studies of data mining standard process, data mining - environment, architectures, and in detail techniques used to get desired outcomes from this process.

The three and four chapters of the research present a literature review to understand theoretical concepts of topics such as machine learning techniques and algorithms, business use cases, the Python eco - systems and its libraries, and literature.

In the fifth chapter of this research, I completed a practical part based on approaches to achieve the thesis targets. whilst solving a single business problem through customer segmentation data mining with the development of a machine learning model for the analysis of business data and making important business decisions.

To summarize, every day a massive amount of data is recorded in a database in any enterprise. We can identify important factors for business progress based on past data. Utilizing data techniques and machine learning algorithms, it is possible to discover hidden patterns and knowledge through data analysis, making it simple to make business decisions and advance technical advancements for improved corporate performance. I believe that every sector or corporation should approach new data-driven technology. Whether a company is a new startup or a large corporation, whether it is a product-based company or a service-based company, it should utilize data properly to make appropriate decisions that impact on financially.