Import Packages

```
In [ ]: import numpy as np # linear algebra
        import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

```
In [94]: import os

         for dirname, _, filenames in os.walk(r'C:\Users\ehsan\OneDrive\Desktop\New folder (3)'):
             for filename in filenames:
                 print(os.path.join(dirname, filename))
```

```
C:\Users\ehsan\OneDrive\Desktop\New folder (3)\loan-test.csv
C:\Users\ehsan\OneDrive\Desktop\New folder (3)\loan-train.csv
```

1. Gathering Data

```
In [28]: #  Create New Variable and stores the dataset values as Data Frame

         loan_train = pd.read_csv(r'C:\Users\ehsan\OneDrive\Desktop\New folder (3)\loan-train.csv')
         loan_test = pd.read_csv(r'C:\Users\ehsan\OneDrive\Desktop\New folder (3)\loan-test.csv')
```

- Lets display the some few information from our large datasets Here, We shows the first five rows from datasets.

```
In [95]: loan_train.head()
```

Out[95]:

	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	0	0.0	5849	0.0	146.412162	360.0	1.0	2	1
1	LP001003	1.0	1.0	1	0	0.0	4583	1508.0	128.000000	360.0	1.0	0	0
2	LP001005	1.0	1.0	0	0	1.0	3000	0.0	66.000000	360.0	1.0	2	1
3	LP001006	1.0	1.0	0	1	0.0	2583	2358.0	120.000000	360.0	1.0	2	1
4	LP001008	1.0	0.0	0	0	0.0	6000	0.0	141.000000	360.0	1.0	2	1

- As we can see in the above output, there are too many columns, (columns known as features as well.) We can also use loan_train to show few rows from the first five and last five record from the dataset

```
In [97]: loan_train
```

Out[97]:

	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	0	0.0	5849	0.0	146.412162	360.0	1.0	2	1
1	LP001003	1.0	1.0	1	0	0.0	4583	1508.0	128.000000	360.0	1.0	0	0
2	LP001005	1.0	1.0	0	0	1.0	3000	0.0	66.000000	360.0	1.0	2	1
3	LP001006	1.0	1.0	0	1	0.0	2583	2358.0	120.000000	360.0	1.0	2	1
4	LP001008	1.0	0.0	0	0	0.0	6000	0.0	141.000000	360.0	1.0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
609	LP002978	0.0	0.0	0	0	0.0	2900	0.0	71.000000	360.0	1.0	0	1
610	LP002979	1.0	1.0	3	0	0.0	4106	0.0	40.000000	180.0	1.0	0	1
611	LP002983	1.0	1.0	1	0	0.0	8072	240.0	253.000000	360.0	1.0	2	1
612	LP002984	1.0	1.0	2	0	0.0	7583	0.0	187.000000	360.0	1.0	2	1
613	LP002990	0.0	0.0	0	0	1.0	4583	0.0	133.000000	360.0	0.0	1	0

614 rows × 13 columns

Here, we can see there are many rows and many columns, To know how many records and columns are available in our dataset, we can use the shape attribute or we can use len() to know how many records and how many features available in the dataset.

```
In [32]: print("Rows: ", len(loan_train))

Rows:  614
```

```
In [33]: print("Columns: ", len(loan_train.columns))

Columns:  13
```

```
In [34]: print("Shape : ", loan_train.shape)

Shape :  (614, 13)
```

```
In [35]: loan_train_columns = loan_train.columns # assign to a variable
         loan_train_columns # print the list of columns

Out[35]: Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
                'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
                'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
               dtype='object')
```

Now, Understanding the Data

- First of all we use the loan_train.describe() method to shows the important information from the dataset
- It provides the count, mean, standard deviation (std), min, quartiles and max in its output.

```
In [36]: loan_train.describe()
```

Out[36]:

	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

As I said the above cell, this the information of all the methamatical details from dataset. Like count, mean, standard deviation (std), min, quartiles(25%, 50%, 75%) and max.

```
In [38]: loan_train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

As we can see in the output.

- There are 614 entries
- There are total 13 features (0 to 12)
- There are three types of datatype dtypes: float64(4), int64(1), object(8)
- It's Memory usage that is, memory usage: 62.5+ KB
- Also, We can check how many missing values available in the Non-Null Count column

2. Exploratory Data Analysis

In this section, We learn about extra information about data and it's characteristics.

- First of all, We explore object type of data So let's make a function to know how many types of values available in the column

```
In [39]: def explore_object_type(df ,feature_name):
             """
             To know, How many values available in object('categorical') type of features
             And Return Categorical values with Count.
             """
             if df[feature_name].dtype ==  'object':
                 print(df[feature_name].value_counts())
```

```
In [40]: # Now, Test and Call a function for gender only
         explore_object_type(loan_train, 'Gender')

Male      489
Female    112
Name: Gender, dtype: int64
```

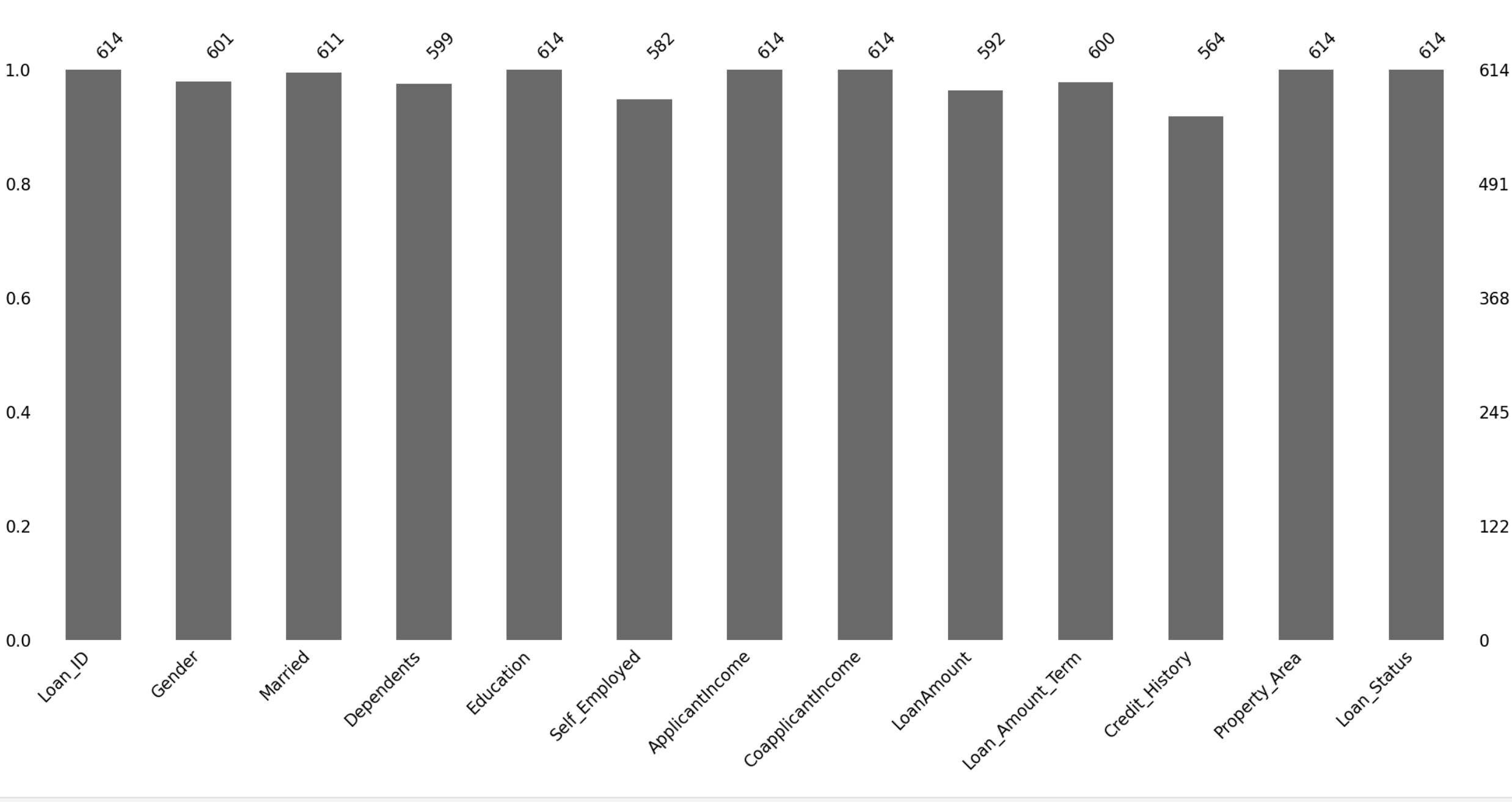
```
In [41]: # Solution is, Do you remember we have variable with name of `loan_train_columns`, Right,  let's use it
         # 'Loan_ID', 'Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Property_Area', 'Loan_Status'

         for featureName in loan_train_columns:
             if loan_train[featureName].dtype == 'object':
                 print('\n"' + str(featureName) + '\'s" Values with count are :')
                 explore_object_type(loan_train, str(featureName))

"Loan_ID's" Values with count are :
LP001002    1
LP002328    1
LP002305    1
LP002308    1
LP002314    1
           ..
LP001692    1
LP001693    1
LP001698    1
LP001699    1
LP002990    1
Name: Loan_ID, Length: 614, dtype: int64

"Gender's" Values with count are :
Male      489
Female    112
Name: Gender, dtype: int64

"Married's" Values with count are :
Yes    398
No     213
Name: Married, dtype: int64

"Dependents's" Values with count are :
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64

"Education's" Values with count are :
Graduate        480
Not Graduate    134
Name: Education, dtype: int64

"Self_Employed's" Values with count are :
No     500
Yes     82
Name: Self_Employed, dtype: int64

"Property_Area's" Values with count are :
Semiurban    233
Urban        202
Rural        179
Name: Property_Area, dtype: int64

"Loan_Status's" Values with count are :
Y    422
N    192
Name: Loan_Status, dtype: int64
```

- We need to fill null values with mean and median using missingno package

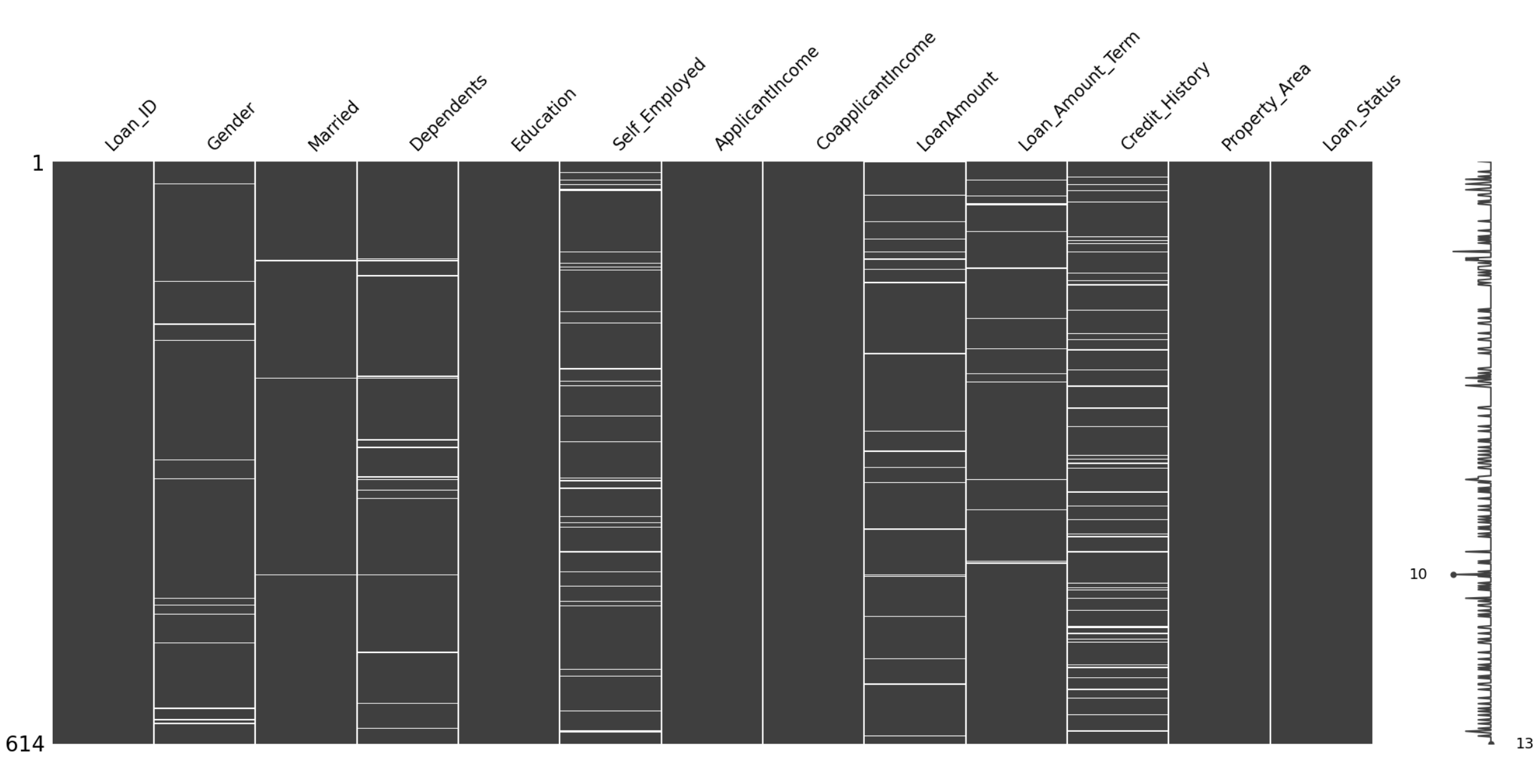
```
In [46]: !pip install missingno
         import missingno as msno

Collecting missingno
  Downloading missingno-0.5.2-py3-none-any.whl (8.7 kB)
Requirement already satisfied: scipy in c:\users\ehsan\anaconda3\lib\site-packages (from missingno) (1.10.0)
Requirement already satisfied: seaborn in c:\users\ehsan\anaconda3\lib\site-packages (from missingno) (0.12.2)
Requirement already satisfied: matplotlib in c:\users\ehsan\anaconda3\lib\site-packages (from missingno) (3.7.0)
Requirement already satisfied: numpy in c:\users\ehsan\anaconda3\lib\site-packages (from missingno) (1.23.5)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\ehsan\anaconda3\lib\site-packages (from matplotlib->missingno) (4.25.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\ehsan\anaconda3\lib\site-packages (from matplotlib->missingno) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\users\ehsan\anaconda3\lib\site-packages (from matplotlib->missingno) (0.11.0)
Requirement already satisfied: packaging>=20.0 in c:\users\ehsan\anaconda3\lib\site-packages (from matplotlib->missingno) (22.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\ehsan\anaconda3\lib\site-packages (from matplotlib->missingno) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\ehsan\anaconda3\lib\site-packages (from matplotlib->missingno) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\ehsan\anaconda3\lib\site-packages (from matplotlib->missingno) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\ehsan\anaconda3\lib\site-packages (from matplotlib->missingno) (1.4.4)
Requirement already satisfied: pandas>=0.25 in c:\users\ehsan\anaconda3\lib\site-packages (from seaborn->missingno) (1.5.3)
Requirement already satisfied: pytz>=2020.1 in c:\users\ehsan\anaconda3\lib\site-packages (from pandas>=0.25->seaborn->missingno) (2022.7)
Requirement already satisfied: six>=1.5 in c:\users\ehsan\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib->missingno) (1.16.0)
Installing collected packages: missingno
Successfully installed missingno-0.5.2
```

```
In [47]: # list of how many percentage values are missing
         loan_train

         loan_train.isna().sum()
         # round((loan_train.isna().sum() / len(loan_train)) * 100, 2)

Out[47]: 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
```

```
In [48]: msno.bar(loan_train)

Out[48]: <Axes: >
```




```
In [49]: msno.matrix(loan_train )

Out[49]: <Axes: >
```



- As we can see here, there are too many columns missing with small amount of null values so we use mean amd mode to replace with NaN values

```
In [50]: loan_train['Credit_History'].fillna(loan_train['Credit_History'].mode(), inplace=True) # Mode
         loan_test['Credit_History'].fillna(loan_test['Credit_History'].mode(), inplace=True) # Mode


         loan_train['LoanAmount'].fillna(loan_train['LoanAmount'].mean(), inplace=True) # Mean
         loan_test['LoanAmount'].fillna(loan_test['LoanAmount'].mean(), inplace=True) # Mean
```

convert Categorical variable with Numerical values

- Loan_Status feature boolean values, So we replace Y values with 1 and N values with 0 and same for other Boolean types of columns

```
In [51]: loan_train.Loan_Status = loan_train.Loan_Status.replace({"Y": 1, "N" : 0})
         # loan_test.Loan_Status = loan_test.Loan_Status.replace({"Y": 1, "N" : 0})

         loan_train.Gender = loan_train.Gender.replace({"Male": 1, "Female" : 0})
         loan_test.Gender = loan_test.Gender.replace({"Male": 1, "Female" : 0})

         loan_train.Married = loan_train.Married.replace({"Yes": 1, "No" : 0})
         loan_test.Married = loan_test.Married.replace({"Yes": 1, "No" : 0})

         loan_train.Self_Employed = loan_train.Self_Employed.replace({"Yes": 1, "No" : 0})
         loan_test.Self_Employed = loan_test.Self_Employed.replace({"Yes": 1, "No" : 0})
```

```
In [52]: loan_train['Gender'].fillna(loan_train['Gender'].mode()[0], inplace=True)
         loan_test['Gender'].fillna(loan_test['Gender'].mode()[0], inplace=True)

         loan_train['Dependents'].fillna(loan_train['Dependents'].mode()[0], inplace=True)
         loan_test['Dependents'].fillna(loan_test['Dependents'].mode()[0], inplace=True)

         loan_train['Married'].fillna(loan_train['Married'].mode()[0], inplace=True)
         loan_test['Married'].fillna(loan_test['Married'].mode()[0], inplace=True)

         loan_train['Credit_History'].fillna(loan_train['Credit_History'].mean(), inplace=True)
         loan_test['Credit_History'].fillna(loan_test['Credit_History'].mean(), inplace=True)
```

- Here, Property_Area, Dependents and Education has multiple values so now we can use LabelEncoder from sklearn package

```
In [53]: from sklearn.preprocessing import LabelEncoder
         feature_col = ['Property_Area','Education', 'Dependents']
         le = LabelEncoder()
         for col in feature_col:
             loan_train[col] = le.fit_transform(loan_train[col])
             loan_test[col] = le.fit_transform(loan_test[col])
```

Finally, We have all the features with numerical values

3. Data Visualizations

- In this section, We are showing the visual information from the dataset, For that we need some pakages that are matplotlib and seaborn

```
In [54]: import matplotlib.pyplot as plt
         %matplotlib inline


         import seaborn as sns
         sns.set_style('dark')
```

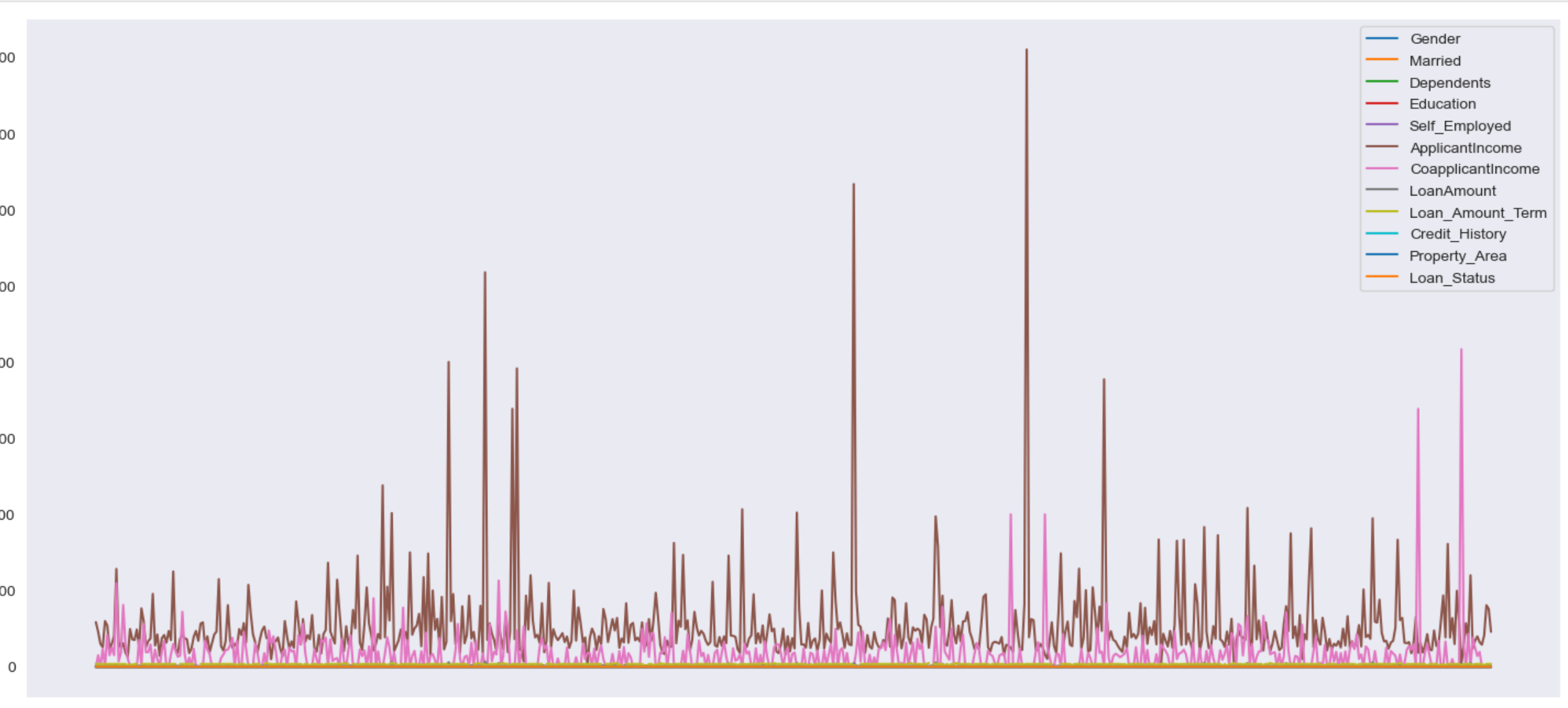
```
In [55]: loan_train
```

Out[55]:

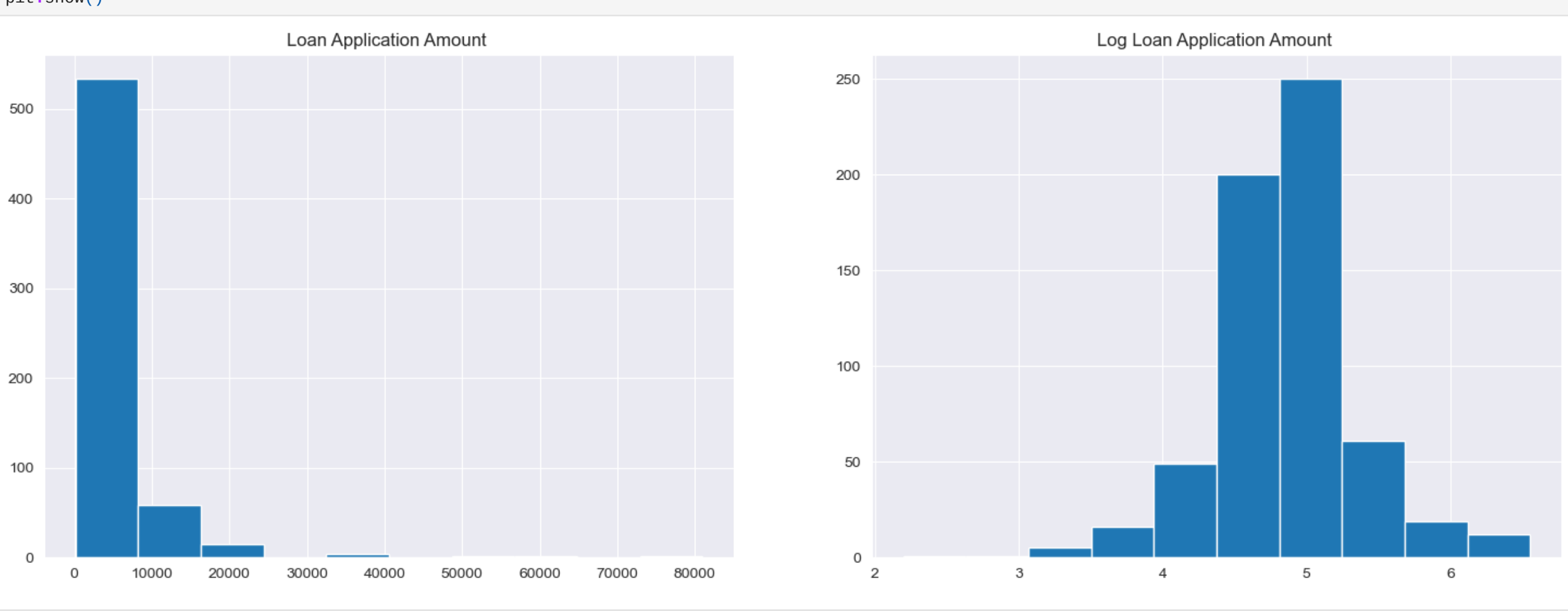
	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	1.0	0.0	0	0	0.0	5849	0.0	146.412162	360.0	1.0	2	1
1	LP001003	1.0	1.0	1	0	0.0	4583	1508.0	128.000000	360.0	1.0	0	0
2	LP001005	1.0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

614 rows × 13 columns

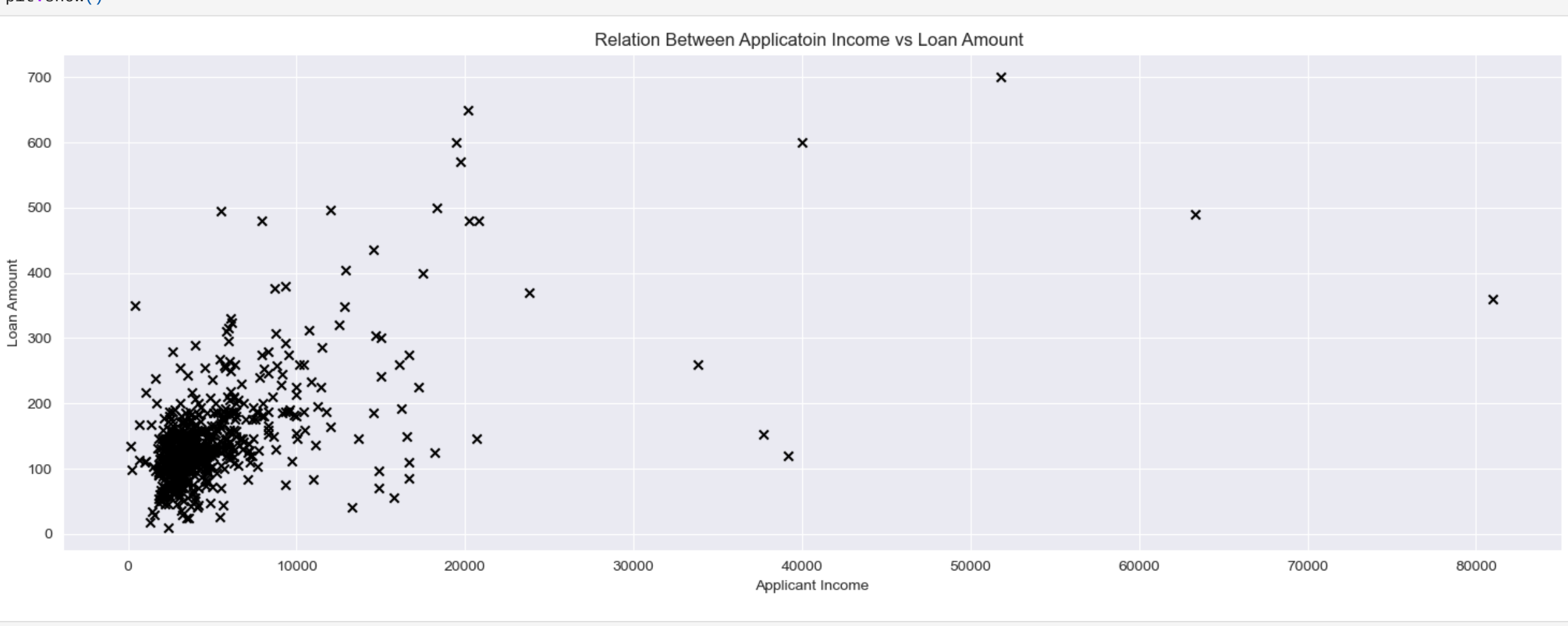
```
In [56]: loan_train.plot(figsize=(18, 8))

         plt.show()
```



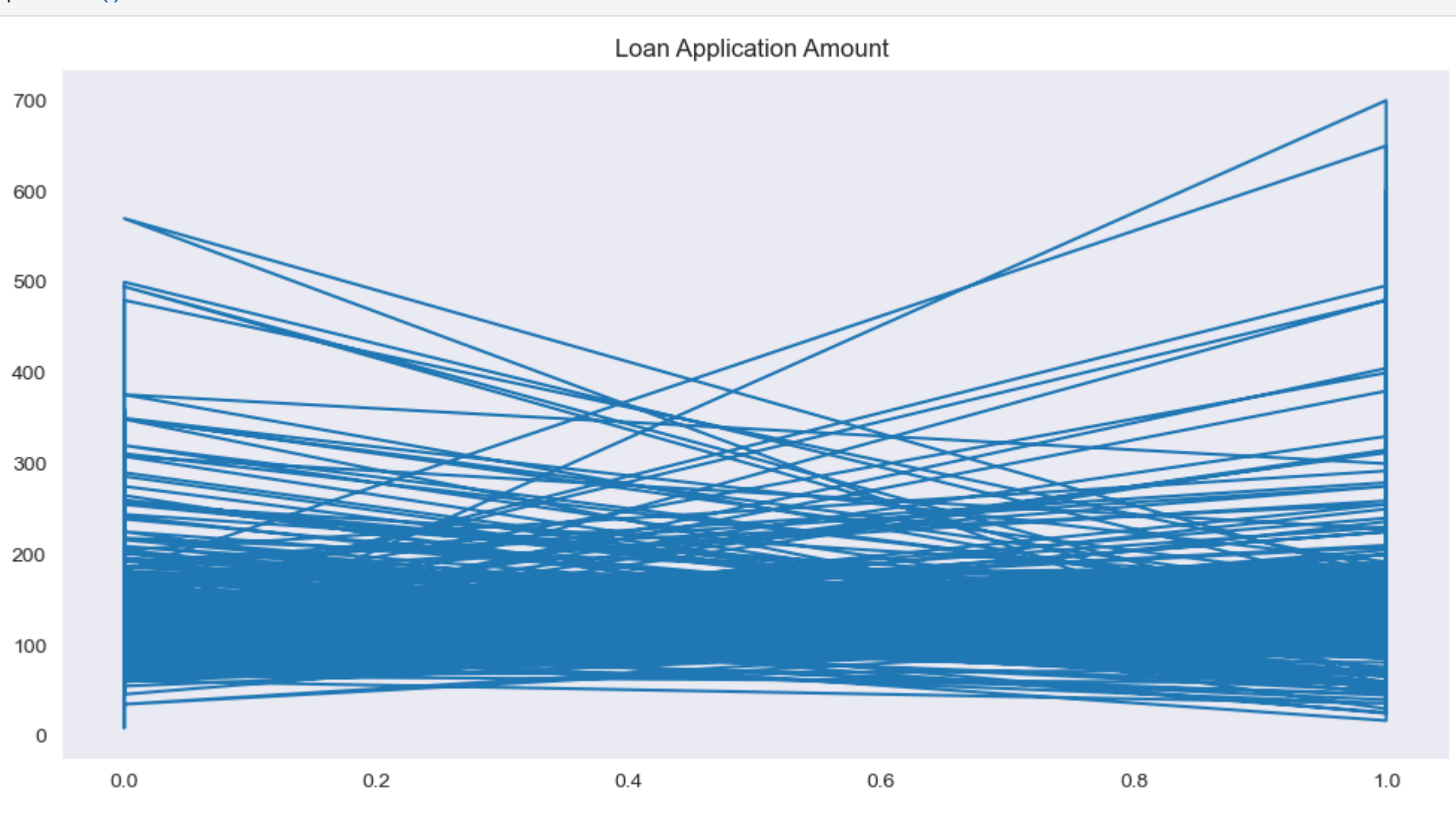

```
In [57]: plt.figure(figsize=(18, 6))
         plt.subplot(1, 2, 1)


         loan_train['ApplicantIncome'].hist(bins=10)
         plt.title("Loan Application Amount ")

         plt.subplot(1, 2, 2)
         plt.grid()
         plt.hist(np.log(loan_train['LoanAmount']))
         plt.title("Log Loan Application Amount ")

         plt.show()
```



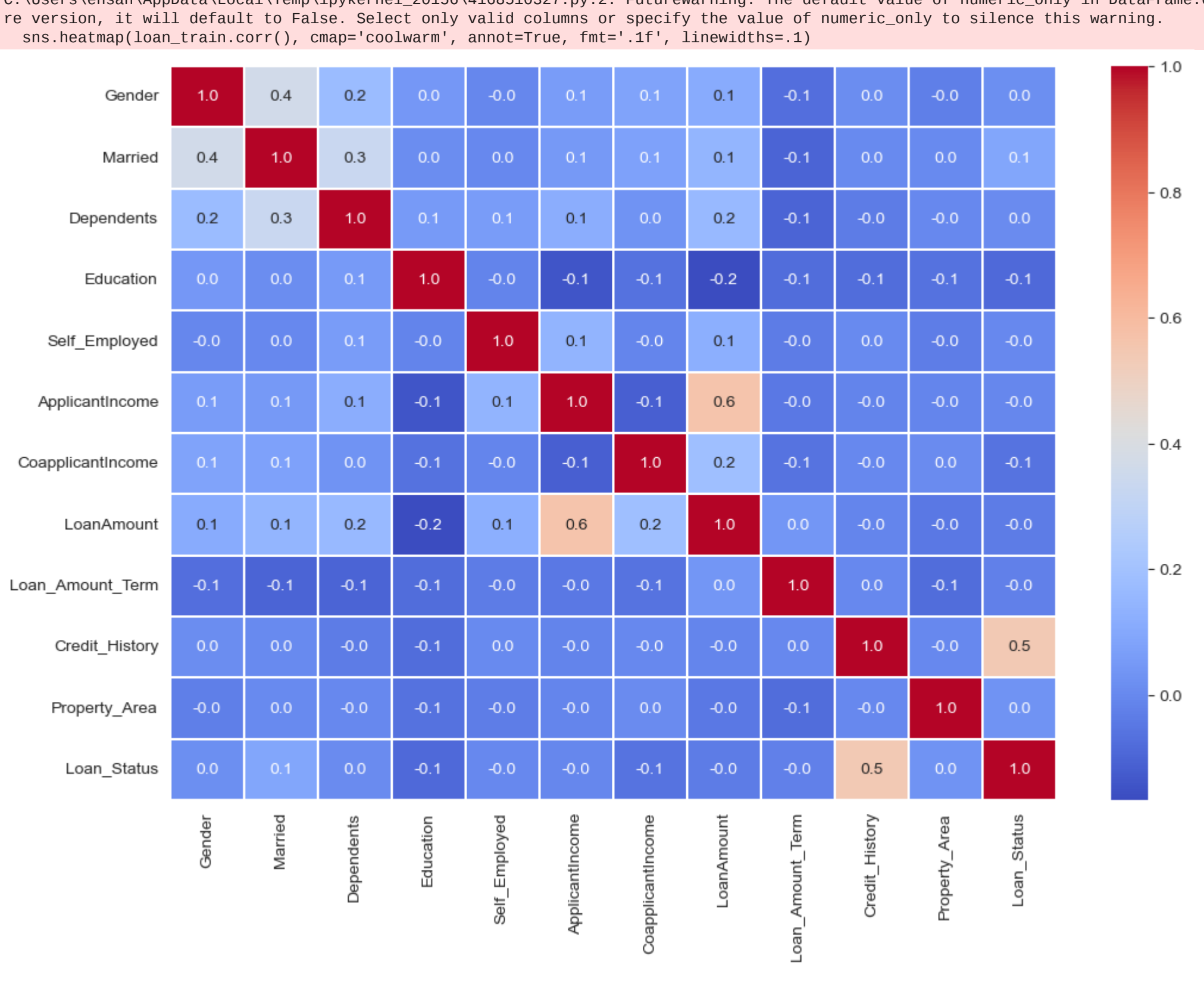

```
In [58]: plt.figure(figsize=(18, 6))
         plt.title("Relation Between Applicatoin Income vs Loan Amount ")

         plt.grid()
         plt.scatter(loan_train['ApplicantIncome'] , loan_train['LoanAmount'], c='k', marker='x')
         plt.xlabel("Applicant Income")
         plt.ylabel("Loan Amount")
         plt.show()
```




```
In [59]: plt.figure(figsize=(12, 6))
         plt.plot(loan_train['Loan_Status'], loan_train['LoanAmount'])
         plt.title("Loan Application Amount ")
         plt.show()
```




```
In [60]: plt.figure(figsize=(12,8))
         sns.heatmap(loan_train.corr(), cmap='coolwarm', annot=True, fmt='.1f', linewidths=.1)
         plt.show()

C:\Users\ehsan\AppData\Local\Temp\ipykernel_20156\4168510327.py:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a futu
re version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(loan_train.corr(), cmap='coolwarm', annot=True, fmt='.1f', linewidths=.1)
```



- In this heatmap, we can clearly seen the relation between two variables

4. Choose ML Model.

- In this step, We have a lots of Machine Learning Model from sklearn package, and we need to decide which model is give us the better performance. then we use that model in final stage and send to the production level.

```
In [74]: from sklearn.linear_model import LogisticRegression

         from sklearn.metrics import accuracy_score
```

- Let's build the model

```
In [75]: logistic_model = LogisticRegression()
```

5. Traing the ML Model

Before fitting the model, We need to decide how many feature are available for testing and training, then after complete this step. fitt the model

Currently, we are using Credit_History', 'Education', 'Gender features for training so let's create train and test variables

```
In [81]: train_features = ['Credit_History', 'Education', 'Gender']

         X_train = loan_train[train_features].values
         Y_train = loan_train['Loan_Status'].values

         X_test = loan_test[train_features].values
```

```
In [82]: logistic_model.fit(X_train, Y_train)

Out[82]: ▾ LogisticRegression
         LogisticRegression()
```

6. Predict Mode

```
In [83]: # Predict the model for testin data

         predicted = logistic_model.predict(x_test)
```

```
In [84]: # check the coefficeints of the trained model
         print('Coefficient of model :', logistic_model.coef_)

Coefficient of model : [[ 3.316164   -0.3059193   0.09398266]]
```

```
In [85]: # check the intercept of the model
         print('Intercept of model',logistic_model.intercept_)

Intercept of model [-1.98307795]
```

```
In [86]: # Accuray Score on train dataset
         # accuracy_train = accuracy_score(x_test, predicted)
         score = logistic_model.score(x_train, y_train)
         print('accuracy_score overall :', score)
         print('accuracy_score percent :', round(score*100,2))

accuracy_score overall : 0.8094462540716613
accuracy_score percent : 80.94
```

```
In [87]: # predict the target on the test dataset
         predict_test = logistic_model.predict(x_test)
         print('Target on test data',predict_test)

Target on test data [1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 0 0 1 0 1 1 1 1
 1 1 1 1 1 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1
 1 1 1 1 1 1 0 0 0 1 1 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0
 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 0 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 0 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
 1 1 0 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 1
 1 1 1 1 1 1 0 1 0 1 1 1 1 0 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1
 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1]
```

7. Deploy Model

- Finally, we are done so far. The last step is to deploy our model in production map. So we need to export our model and bind with web application API. Using pickle we can export our model and store in to logistic_model.pkl file, so we can ealy access this file and calculate customize prediction using Web App API.

A little bit information about pickle:

Pickle is the standard way of serializing objects in Python. You can use the pickle operation to serialize your machine learning algorithms and save the serialized format to a file. Later you can load this file to deserialize your model and use it to make new predictions

```
In [91]: import pickle as pkl
```

```
In [99]: # save the model to disk
         filename = 'logistic_model.pkl'
         pkl.dump(logistic_model, open(filename, 'wb')) # wb means write as binary
```

```
In [ ]:
```