Task: Determine the eligibility for granting Home loan.

Objective of this notebook is:

1. To understand the patterns in the data.
2. How to Handle the categorical features.
3. How to deal with missing data.
4. Feature Engineering
5. Finding the most important features while taking the decision of granting a loan application.
6. Understanding the Normalization and standardisation of the data.

Load data and libraries

[ ] ↳ *11 cells hidden*

Basic Data Exploration

[ ] ↳ *9 cells hidden*

Basic Data visualization: Univariate

[ ] ↳ *11 cells hidden*

Simple Feature Engineering

```
# Feature binning: income
bins=[0,2500,4000,6000, 8000, 10000, 20000, 40000, 81000]
group=['Low','Average','medium', 'H1', 'h2', 'h3', 'h4' , 'Very high']
data['Income_bin']= pd.cut(data['ApplicantIncome'],bins,labels=group)
```

```
data.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coap
0	Male	No	0	Graduate	No	5849	
1	Male	Yes	1	Graduate	No	4583	
2	Male	Yes	0	Graduate	Yes	3000	

Incomes

[ ] ↳ *11 cells hidden*

Loan Amount and Loan Term

[ ] ↳ *11 cells hidden*

Dependents and Loan **approval**

[ ] ↳ *6 cells hidden*

Credit Score vs Loan Approval

[ ] ↳ *3 cells hidden*

Missing Values & Data Cleaning

```
data.isna().sum()
```

```
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
TotalIncome             0
Loan_Amount_per_year   36
EMI                    36
Able_to_pay_EMI         0
dtype: int64
```

```
# Function to create a data frame with number and percentage of missing data in a d
```

```
def missing_to_df(df):
    #Number and percentage of missing data in training data set for each column
    total_missing_df = df.isnull().sum().sort_values(ascending =False)
    percent_missing_df = (df.isnull().sum()/df.isnull().count()*100).sort_values(as
    missing_data_df = pd.concat([total_missing_df, percent_missing_df], axis=1, key
    return missing_data_df
```

```
missing_df = missing_to_df(data)
missing_df[missing_df['Total'] > 0]
```

	Total	Percent
Credit_History	50	8.143322
Loan_Amount_per_year	36	5.863192
EMI	36	5.863192
Self_Employed	32	5.211726
LoanAmount	22	3.583062
Dependents	15	2.442997
Loan_Amount_Term	14	2.280130
Gender	13	2.117264
Married	3	0.488599

```
# Credit History=2 for nan/missing values.
data['Credit_History'] = data['Credit_History'].fillna(2)
```

```
# Self_Employed = 'Other' for nan
data.Self_Employed.unique()
```

```
array(['No', 'Yes', nan], dtype=object)
```

```
data['Self_Employed'] = data['Self_Employed'].fillna('Other')
```

```
# median imputation for numerical columns.
from sklearn.impute import SimpleImputer

num_missing = ['EMI', 'Loan_Amount_per_year',  'LoanAmount',  'Loan_Amount_Term']

median_imputer = SimpleImputer(strategy = 'median')
for col in num_missing:
    data[col] = pd.DataFrame(median_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
# Highest Freq imputation for some categorical columns.
cat_missing = ['Gender', 'Married','Dependents']
```

```
freq_imputer = SimpleImputer(strategy = 'most_frequent')
for col in cat_missing:
    data[col] = pd.DataFrame(freq_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
missing_df = missing_to_df(data)
missing_df[missing_df['Total'] > 0]
```

Total	Percent

Categorical to Numerical encoding

Nominal vs Ordinal variables

1. One Hot Encoding
2. Label encoding
3. Target Encoding

Appropriate encoding depends on what our task is (and) what we do next?

Task: Compute Correlation (PCC and SRCC) between each feature and the Loan-Status

```
s = (data.dtypes == 'object')
object_cols = list(s[s].index)
object_cols
```

```
['Gender',
 'Married',
 'Education',
 'Self_Employed',
 'Property_Area',
 'Loan_Status']
```

Loan Status

[ ] ↳ *3 cells hidden*

Gender

```
#Gender
data['Gender'].value_counts()
```

```
Male      502
Female    112
Name: Gender, dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
col='Gender'
data[col] = label_encoder.fit_transform(data[col])
```

```
data[col].value_counts()
```

```
1    502
0    112
Name: Gender, dtype: int64
```

Married

[ ] ↳ *2 cells hidden*

Property Area

```
# col='Property_Area'
col='Property_Area'
data[col].value_counts()
```

```
Semiurban    233
Urban        202
Rural        179
Name: Property_Area, dtype: int64
```

```
!pip install category_encoders
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
Requirement already satisfied: category_encoders in /usr/local/lib/python3.7/d
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: pandas>=0.21.1 in /usr/local/lib/python3.7/dist
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.7/
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-p
```

```
from category_encoders import TargetEncoder

te = TargetEncoder()
```

```
data[col] = te.fit_transform(data[col], data['Loan_Status'])
```

```
col='Property_Area'
data[col].value_counts()
```

```
0.768240    233
0.658416    202
0.614525    179
Name: Property_Area, dtype: int64
```

Education

```
col='Education'
data[col].value_counts()
```

```
Graduate        480
Not Graduate    134
Name: Education, dtype: int64
```

```
label_encoder = LabelEncoder()
data[col] = label_encoder.fit_transform(data[col])
data[col].value_counts()
```

```
0    480
1    134
Name: Education, dtype: int64
```

Self Employed

[ ] ↳ *4 cells hidden*

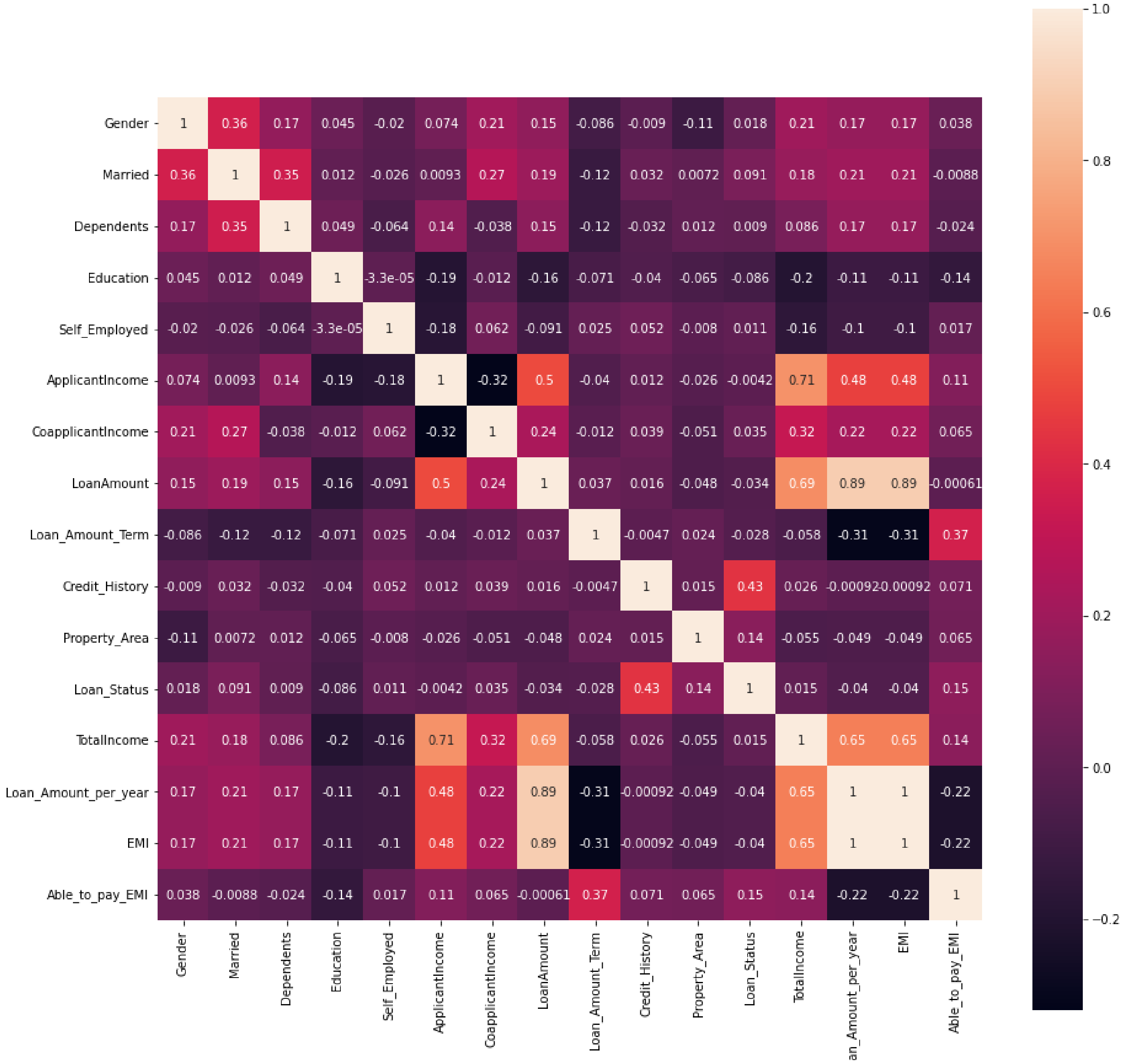
Correlation Coefficients

```
#PCC
plt.figure(figsize=(15, 15))
sns.heatmap(data.corr(method='pearson'), square=True,annot=True)
```

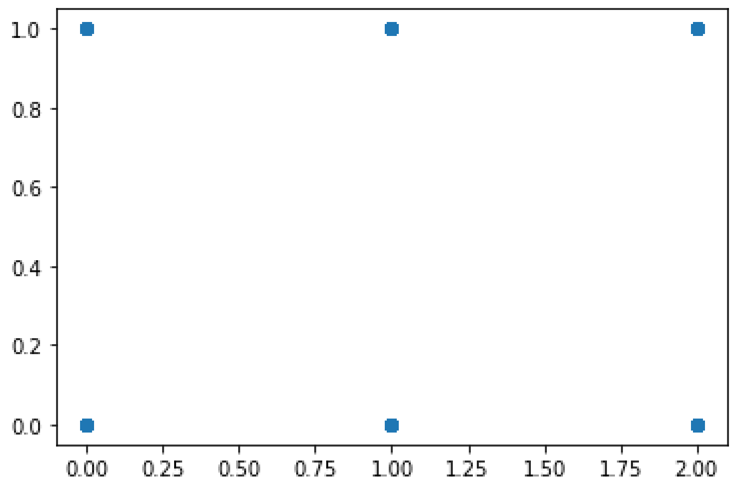
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2ed98f50>
```


```
#SRCC
plt.figure(figsize=(15, 15))
sns.heatmap(data.corr(method='spearman'), square=True,annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2f27f950>
```

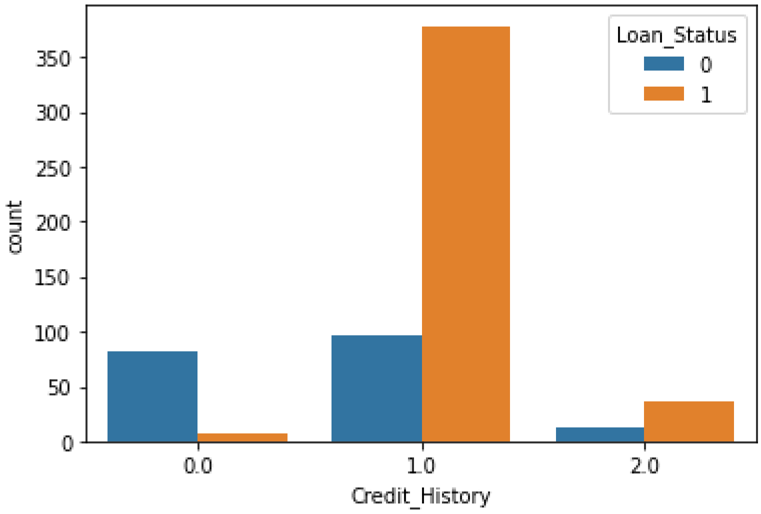


```
plt.scatter(data['Credit_History'], data['Loan_Status'])
plt.show()
#sometimes scatter plots can be misleading do to catgeorical nature of the data
```



```
sns.countplot(data =data, x = 'Credit_History', hue = 'Loan_Status')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2eb7d950>
```



Column Standarization and Normalization

- Mean centering and Variance scaling (Stndard Scaling)
- MinMax Scaling

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

scaler = StandardScaler()
std_data = scaler.fit_transform(data)
std_data = pd.DataFrame(std_data, columns=data.columns)
std_data.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Co
0	0.472343	-1.372089	-0.737806	-0.528362	-0.174052	0.072991	
1	0.472343	0.728816	0.253470	-0.528362	-0.174052	-0.134412	
2	0.472343	0.728816	-0.737806	-0.528362	-0.586643	-0.393747	
3	0.472343	0.728816	-0.737806	1.892641	-0.174052	-0.462062	
4	0.472343	-1.372089	-0.737806	-0.528362	-0.174052	0.097728	

✓ 0s completed at 19:08