LAB PRACTICE - 4: MINI-PROJECT (DATA SCIENCE)

Loan Prediction System

COURSE FACULTY: Prof. Leena A. Deshpande

Team Details

Sr.No.	Name	Division and Batch	Roll Number	GR.No.
1	Sanket Sonje	B – B2	322053	17u176
2	Shardul Pathak	B – B2	322057	17u666

Contents

- Description
- Data Set variables
- Objective
- How the data is created?
- Process flow of project:
- Concepts
- Pre-processing of Data
- Classifier: Models
- Code
- Output

Problem Statement

- We wants to automate the loan eligibility process (real time) based on customer detail provided while filling online application form.
- These details are Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History and others.
- To automate this process, they have given a problem to identify the customers segments, those are eligible for loan amount so that they can specifically target these customers.

Objectives of the Project

- **Main Objective: -**
    1. Automate the Loan Eligibility Process (Real Time).

- **Sub-Objectives: -**
    1. Extract the data from **Datahack** as a Resource.
    2. Analyse the data and fit into the classification model.
    3. Perform and Evaluate the model and predict the Loan Status of entries Test Dataset.

Data set Variables

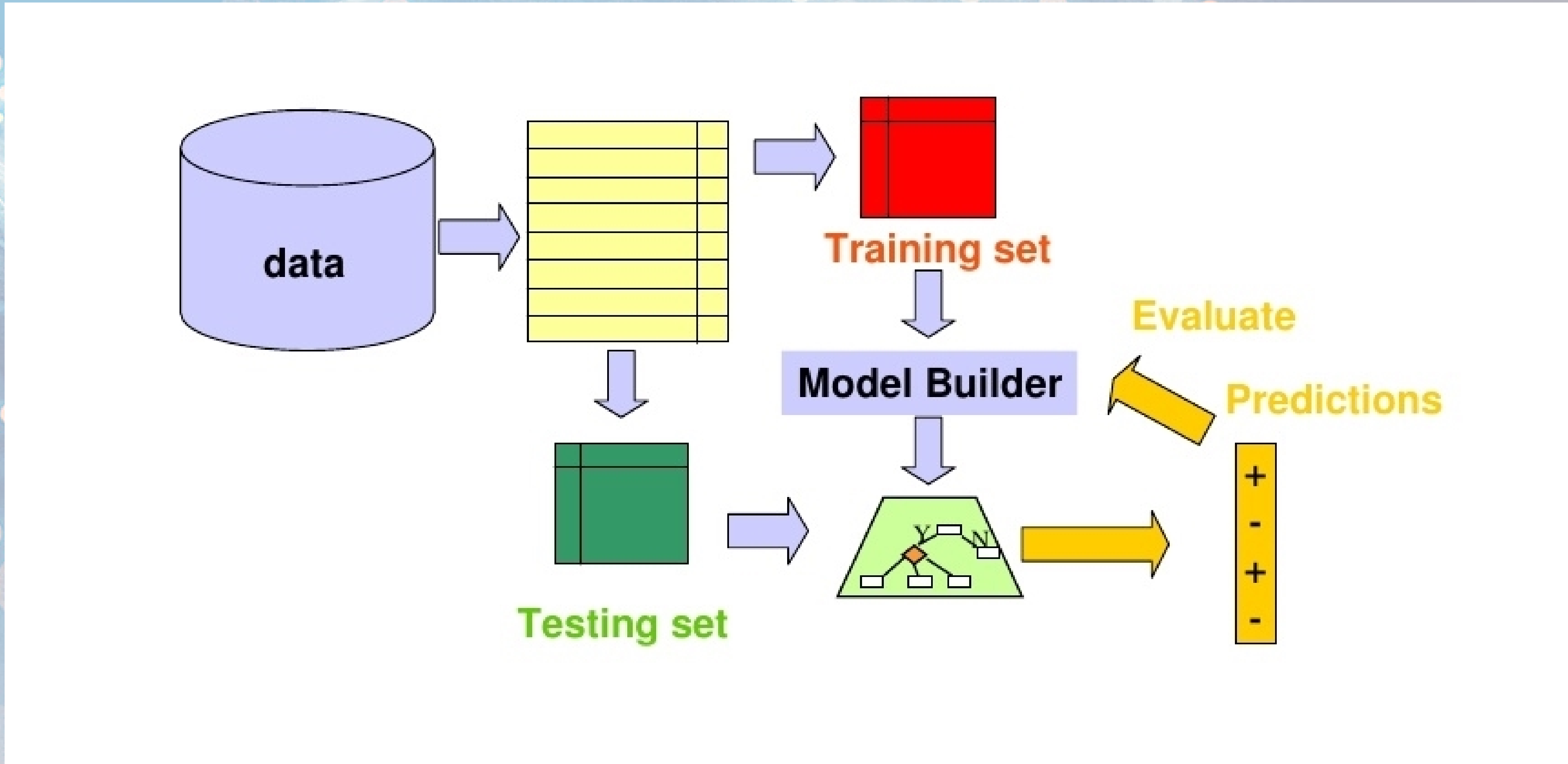
Variable	Description
Loan_ID	Unique Loan ID
Gender	Male/ Female
Married	Applicant married (Y/N)
Dependents	Number of dependents
Education	Applicant Education (Graduate/ Under Graduate)
Self_Employed	Self employed (Y/N)
ApplicantIncome	Applicant income
CoapplicantIncome	Coapplicant income
LoanAmount	Loan amount in thousands
Loan_Amount_Term	Term of loan in months
Credit_History	credit history meets guidelines
Property_Area	Urban/ Semi Urban/ Rural
Loan_Status	Loan approved (Y/N)

Analyze problem and convert it into data

1. How data collected: -
   - Datahack (Resource)

2. How data stored:
   - Data is stored in the format of csv file.
     1. Test.csv
     2. Train.csv

Flow of the Project



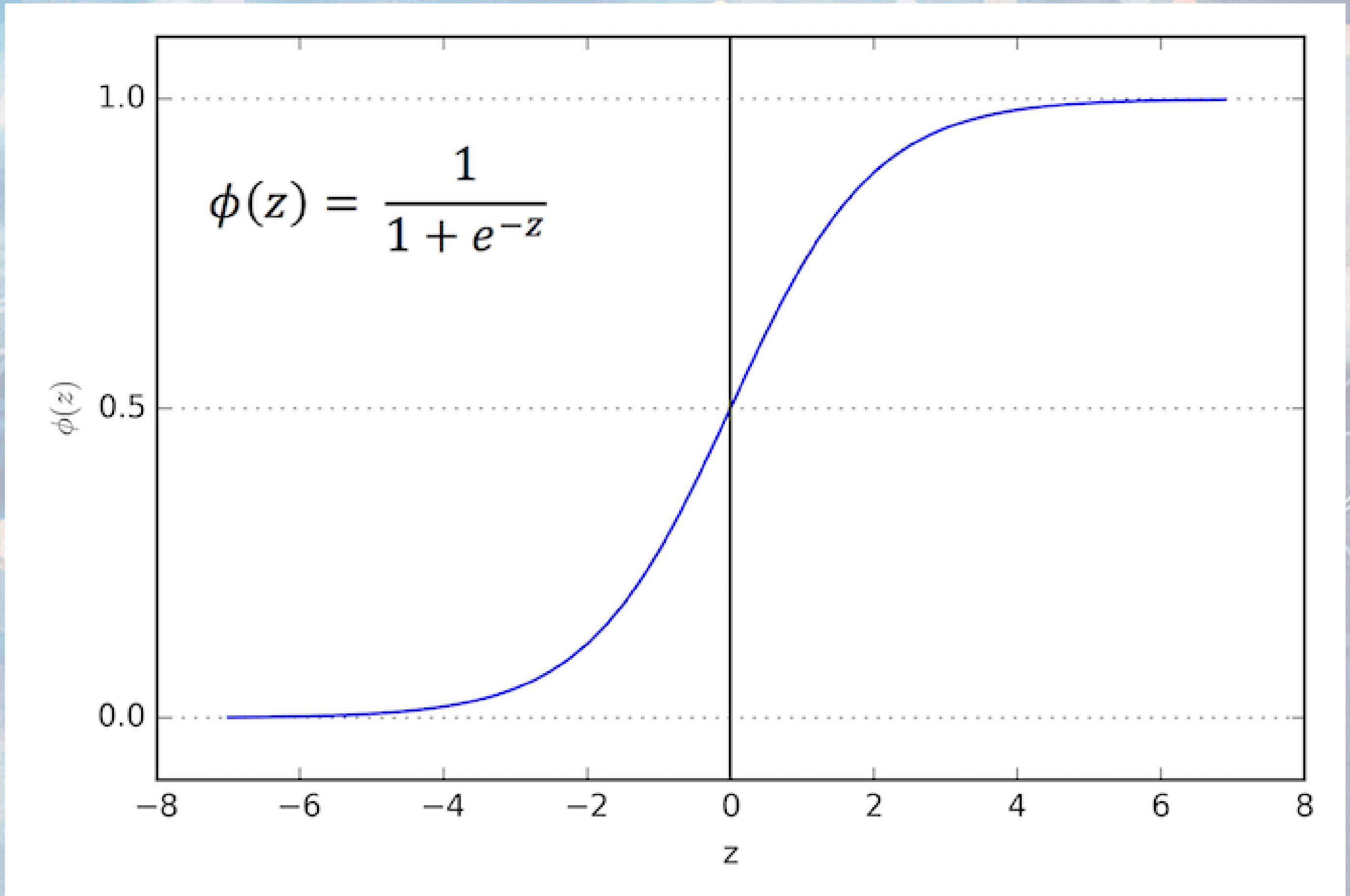
Flow of the Project

- Importing all libraries and storing data sets.
- Understanding various features of the datasets.
- Understanding the Distribution of Non-Categorial values.
- Understanding the Distribution of Categorial values.
- Replacing the Null values of Self Employed column with maximum occurrence of the data
- Nullifying Outliers of Loan Amount and Amount Income.
- Data preparation and model building.
- Testing and Validating score.

Logistic Regression

- Logistic Regression is used when the dependent variable(target) is categorical.
- For example,
  - To predict whether an email is spam (1) or (0)
  - Whether the tumor is malignant (1) or not (0)
- In our example, Our Targeted Value is "Loan Status" and it has values Yes or No.
- To validate our Accuracy, we calculate the Cross Validation Score.

Logistic Regression



Types of Logistic Regression

1. Binary Logistic Regression: -
   - The categorical response has only two 2 possible outcomes. Example: Spam or Not. We use this type.

2. Multinomial Logistic Regression: -
   - Three or more categories without ordering. Example: Predicting which food is preferred more (Veg, Non-Veg, Vegan)

3. Ordinal Logistic Regression: -
   - Three or more categories with ordering. Example: Movie rating from 1 to 5

Analyze problem and convert it into data

1. How data represented using statistical model: -
    - Data distribution: Binary Logistic Regression Model.

2. Statistical operations performed: -
    - Min(), Avg(), describe(), Mode(), etc.

3. Dealing with missing values: -
    - NAN, zero values /variable conversion.

Model building and training

1. Model used : - > Logistic Regression Model
    - Binary output is expected.

2. Libraries to be used : - >
    - SKLEAN, SCIKIT, PANDAS, NUMPY, etc.

Code :- Importing Libraries

```
# Importing Library
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder

# Reading the training dataset in a dataframe using Pandas
df = pd.read_csv("train.csv")

# Reading the test dataset in a dataframe using Pandas
test = pd.read_csv("test.csv")
```

```
# First 20 Rows of training Dataset
df.head(20)
```

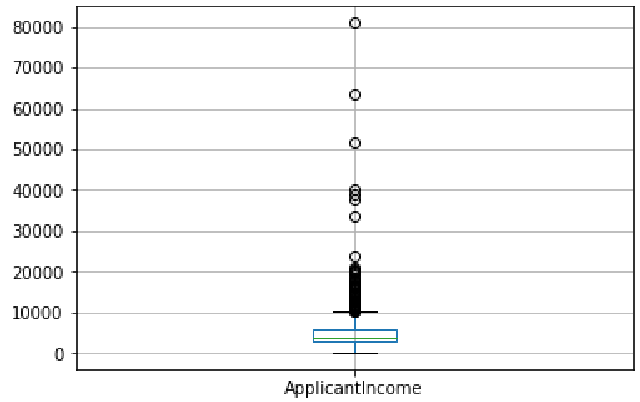
```
# Store total number of observation in training dataset
df_length =len(df)

# Store total number of columns in testing data set
test_col = len(test.columns)
```

Understanding Distribution of Numerical Values
1.ApplicationIncome

```
# Box Plot for understanding the distributions and to observe the outliers.

%matplotlib inline

# Histogram of variable ApplicantIncome

df['ApplicantIncome'].hist()
```

```
# Box Plot for variable ApplicantIncome of training data set

df.boxplot(column='ApplicantIncome')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e8abc75240>
```

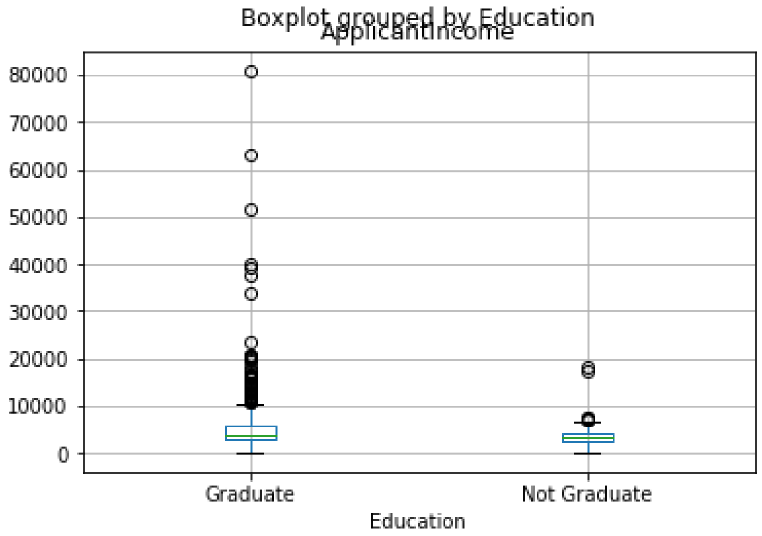


3. The above Box Plot confirms the presence of a lot of outliers/extreme values. This can be attributed to the income disparity in the society.

Understanding Distribution of Numerical Values 1.ApplicationIncome

```
# Box Plot for variable ApplicantIncome by variable Education of training data set

df.boxplot(column='ApplicantIncome', by = 'Education')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e8aba1fc18>
```



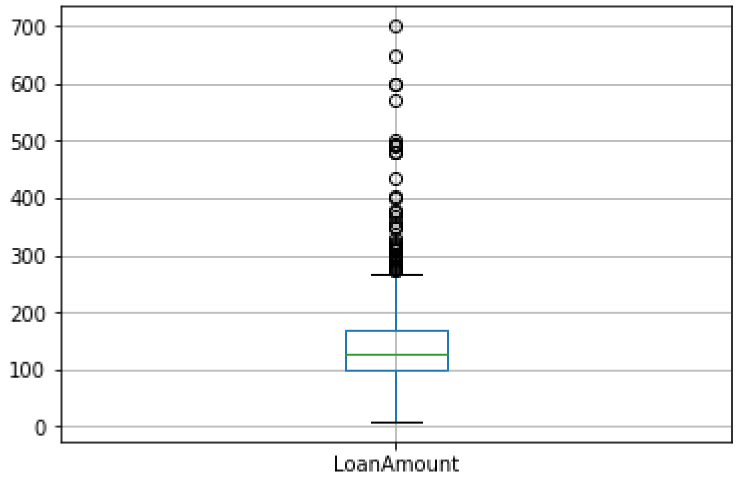
4. We can see that there is no substantial different between the mean income of graduate and non-graduates. But there are a higher number of graduates with very high incomes, which are appearing to be the outliers

Understanding Distribution of Numerical Values
2.LoanAmount

```python
# Histogram of variable LoanAmount

df['LoanAmount'].hist(bins=50)
```

```python
# Box Plot for variable LoanAmount of training data set

df.boxplot(column='LoanAmount')
```

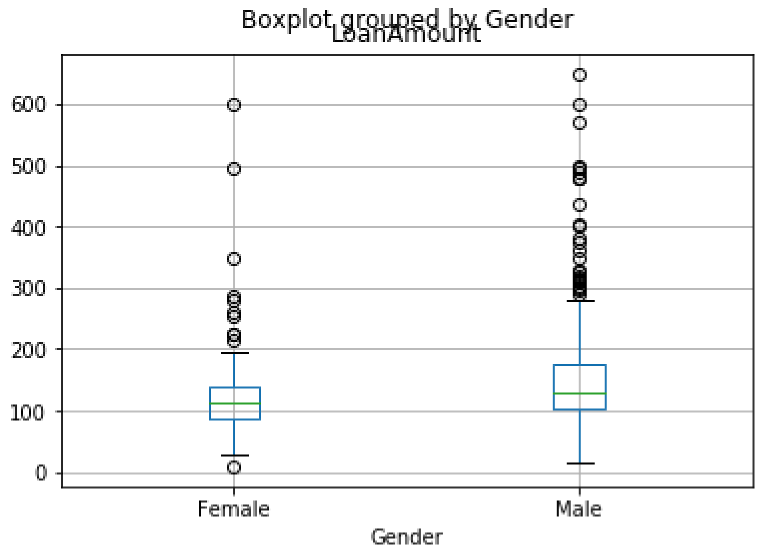
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e8abd4f7b8>
```



Understanding Distribution of Numerical Values
2.LoanAmount

```
# Box Plot for variable LoanAmount by variable Gender of training data set

df.boxplot(column='LoanAmount', by = 'Gender')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e8abd96f98>
```



Understanding Distribution of Categorical Variables

```python
# Loan approval rates in absolute numbers
loan_approval = df['Loan_Status'].value_counts()['Y']
print(loan_approval)
```

```
422
```

- 422 number of loans were approved.

```python
# Credit History and Loan Status
pd.crosstab(df ['Credit_History'], df ['Loan_Status'], margins=True)
```

Loan_Status Credit_History	N	Y	All
0.0	82	7	89
1.0	97	378	475
All	179	385	564

```python
#Function to output percentage row wise in a cross table

def percentageConvert(ser):
    return ser/float(ser[-1])

df=pd.crosstab(df['Credit_History'], df['Loan_Status'], margins=True).apply(percentageConvert, axis=1)
loan_approval_with_credit_1 = df['Y'][1]

print(loan_approval_with_credit_1*100)
```

```
79.57894736842105
```

- 79.58 % of the applicants whose loans were approved have Credit_History equals to 1.

Data Preparation And Model Building

```
# Impute missing values for Gender
df['Gender'].fillna(df['Gender'].mode()[0],inplace=True)

# Impute missing values for Married
df['Married'].fillna(df['Married'].mode()[0],inplace=True)

# Impute missing values for Dependents
df['Dependents'].fillna(df['Dependents'].mode()[0],inplace=True)

# Impute missing values for Credit_History
df['Credit_History'].fillna(df['Credit_History'].mode()[0],inplace=True)

# Convert all non-numeric values to number
cat=['Gender','Married','Dependents','Education','Self_Employed','Credit_History','Property_Area']

df.dtypes
```

```
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
TotalIncome          float64
LoanAmount_log       float64
dtype: object
```

```
#Coverting all non-numerical values to numbers:
for var in cat:
    le = preprocessing.LabelEncoder()
    df[var]=le.fit_transform(df[var].astype('str'))
df.dtypes
```

```
Loan_ID               object
Gender                 int32
Married                int32
Dependents             int32
Education              int32
Self_Employed          int32
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History         int32
Property_Area          int32
Loan_Status           object
TotalIncome          float64
LoanAmount_log       float64
dtype: object
```

Data Preparation And Model Building

```python
#Combining both train and test dataset to work with missing values

#Create a flag for Train and Test Data set
df['Type']='Train'
test['Type']='Test'
fullData = pd.concat([df,test], axis=0)

#Look at the available missing values in the dataset
fullData.isnull().sum()
```

```
F:\Anaconda\lib\site-packages\ipykernel_launcher.py:6: FutureWarning: Sorting because non-concatenation axis is not aligned. A
future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.
```

```
ApplicantIncome        0
CoapplicantIncome      0
Credit_History        29
Dependents            10
Education              0
Gender                11
LoanAmount            27
LoanAmount_log       389
Loan_Amount_Term      20
Loan_ID                0
Loan_Status          367
Married                0
Property_Area          0
Self_Employed         23
TotalIncome          367
Type                   0
dtype: int64
```

Data Preparation And Model Building

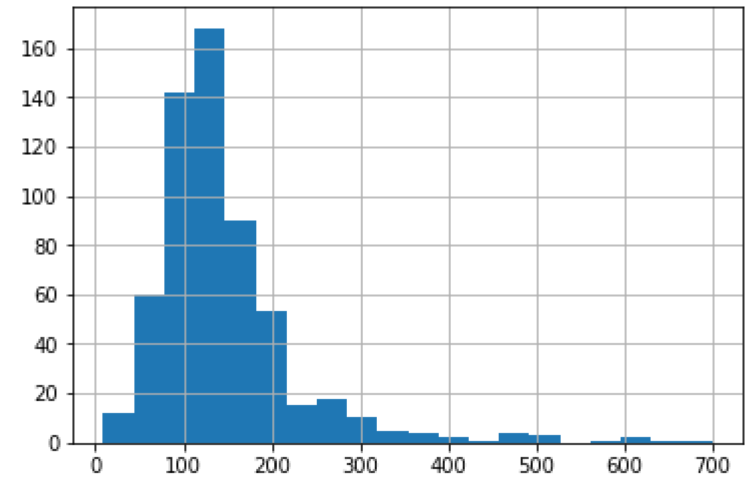
```
#Identify categorical and continuous variables
ID_col = ['Loan_ID']
target_col = ["Loan_Status"]
cat_cols = ['Credit_History','Dependents','Gender','Married','Education','Property_Area','Self_Employed']
```

```
#Imputing Missing values with mean for continuous variable
fullData['LoanAmount'].fillna(fullData['LoanAmount'].mean(), inplace=True)
fullData['LoanAmount_log'].fillna(fullData['LoanAmount_log'].mean(), inplace=True)
fullData['Loan_Amount_Term'].fillna(fullData['Loan_Amount_Term'].mean(), inplace=True)
fullData['ApplicantIncome'].fillna(fullData['ApplicantIncome'].mean(), inplace=True)
fullData['CoapplicantIncome'].fillna(fullData['CoapplicantIncome'].mean(), inplace=True)

#Imputing Missing values with mode for categorical variables
fullData['Gender'].fillna(fullData['Gender'].mode()[0], inplace=True)
fullData['Married'].fillna(fullData['Married'].mode()[0], inplace=True)
fullData['Dependents'].fillna(fullData['Dependents'].mode()[0], inplace=True)
fullData['Loan_Amount_Term'].fillna(fullData['Loan_Amount_Term'].mode()[0], inplace=True)
fullData['Credit_History'].fillna(fullData['Credit_History'].mode()[0], inplace=True)
```

Nullify the Effect of Outliers
1. Loan Amount

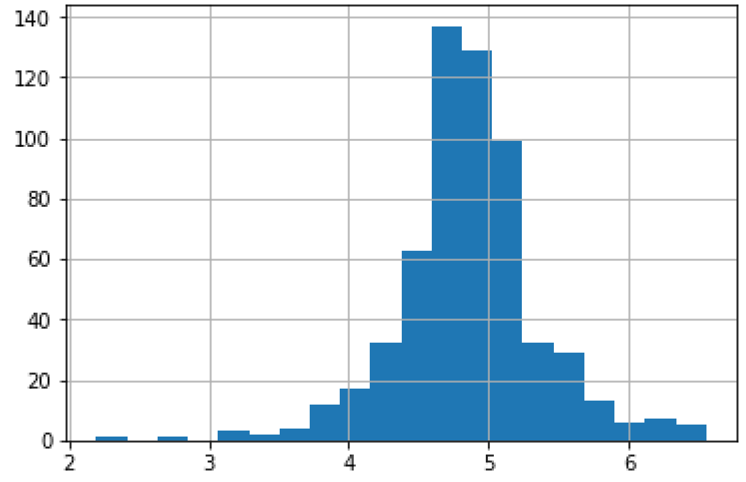
```python
# Add both ApplicantIncome and CoapplicantIncome to TotalIncome
df['TotalIncome'] = df['ApplicantIncome'] + df['CoapplicantIncome']

# Looking at the distribtion of TotalIncome
df['LoanAmount'].hist(bins=20)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e8abc5a550>
```




```python
# Perform log transformation of TotalIncome to make it closer to normal
df['LoanAmount_log'] = np.log(df['LoanAmount'])

# Looking at the distribtion of TotalIncome_log
df['LoanAmount_log'].hist(bins=20)
```

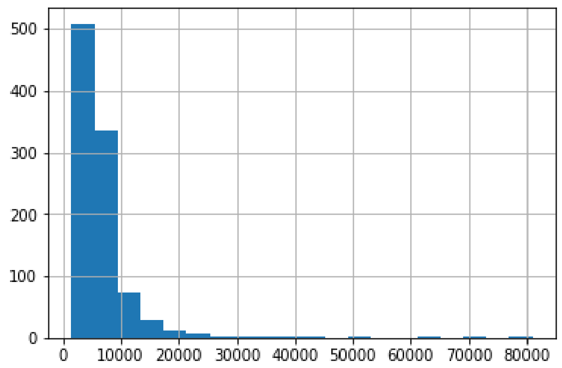
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e8a88e72b0>
```



Nullify the Effect of Outliers
2. Total Income

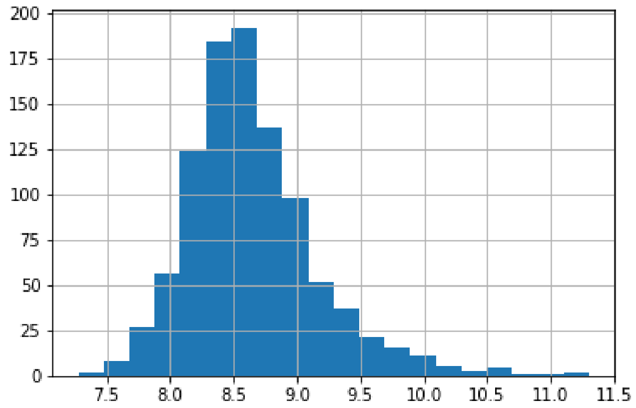
```
#Create a new column as Total Income

fullData['TotalIncome']=fullData['ApplicantIncome'] + fullData['CoapplicantIncome']

fullData['TotalIncome'].hist(bins=20)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e8abe92a90>
```




```
#Nullify the effect of outliers using log transformation
fullData['TotalIncome_log'] = np.log(fullData['TotalIncome'])

#Histogram for Total Income
fullData['TotalIncome_log'].hist(bins=20)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e8abeec2b0>
```



Data Preparation And Model Building

```python
#create label encoders for categorical features
for var in cat_cols:
    number = LabelEncoder()
    fullData[var] = number.fit_transform(fullData[var].astype('str'))

train_modified=fullData[fullData['Type']=='Train']
test_modified=fullData[fullData['Type']=='Test']
train_modified["Loan_Status"] = number.fit_transform(train_modified["Loan_Status"].astype('str'))
```

```
F:\Anaconda\lib\site-packages\ipykernel_launcher.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
```

Classification Model

```python
#Import models from scikit learn module:
from sklearn import metrics
from sklearn.model_selection import cross_val_predict, KFold
from sklearn.model_selection import train_test_split

#Generic function for making a classification model and accessing performance:

def classification_model(model, data, predictors, outcome):
    #Fit the model:
    model.fit(data[predictors],data[outcome])

    #Make predictions on training set:
    predictions = model.predict(data[predictors])

    #Print accuracy
    accuracy = metrics.accuracy_score(predictions,data[outcome])
    print ("Accuracy : %s" % "{0:.3%}".format(accuracy))

    #Perform k-fold cross-validation with 5 folds
    kf = KFold(n_splits=5)
    error = []
    for train, test in kf.split(data[predictors]):
    #Filter training data
       train_predictors = (data[predictors].iloc[train,:])

    #The target we're using to train the algorithm.
       train_target = data[outcome].iloc[train]

    #Training the algorithm using the predictors and target.
       model.fit(train_predictors, train_target)

    #Record error from each cross-validation run
       error.append(model.score(data[predictors].iloc[test,:], data[outcome].iloc[test]))

    print ("Cross-Validation Score : %s" % "{0:.3%}".format(accuracy))

    #Fit the model again so that it can be refered outside the function:
    #model.fit(data[predictors],data[outcome])
```

Logistic Regression Model and Output

```
from sklearn.linear_model import LogisticRegression


predictors_Logistic=['Credit_History','Education','Gender']

x_train = train_modified[list(predictors_Logistic)].values
y_train = train_modified["Loan_Status"].values

x_test=test_modified[list(predictors_Logistic)].values
```

```
# Create logistic regression object
model = LogisticRegression()

# Train the model using the training sets
model.fit(x_train, y_train)

#Predict Output
predicted= model.predict(x_test)

#Reverse encoding for predicted outcome
predicted = number.inverse_transform(predicted)

#Store it to test dataset
test_modified['Loan_Status']=predicted

outcome_var = 'Loan_Status'

classification_model(model, df,predictors_Logistic,outcome_var)

test_modified.to_csv("Logistic_Prediction.csv",columns=['Loan_ID','Loan_Status'])
```

```
F:\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in
0.22. Specify a solver to silence this warning.
  FutureWarning)
```

```
Accuracy : 80.945%
Cross-Validation Score : 80.945%
```

Output

Logistic Regression csv File.

	Unnamed: 0	Loan_ID	Loan_Status
0	0	LP001015	Y
1	1	LP001022	Y
2	2	LP001031	Y
3	3	LP001035	Y
4	4	LP001051	Y
5	5	LP001054	Y
6	6	LP001055	Y
7	7	LP001056	N
8	8	LP001059	Y
9	9	LP001067	Y
10	10	LP001078	Y
11	11	LP001082	Y
12	12	LP001083	Y
13	13	LP001094	N
14	14	LP001096	Y
15	15	LP001099	Y
16	16	LP001105	Y
17	17	LP001107	Y
18	18	LP001108	Y
19	19	LP001115	Y

Conclusion And Observations

- The chances of getting a loan will be higher for:
  1. Applicants having a credit history (we observed this in exploration.)
  2. Applicants with higher applicant and co-applicant incomes
  3. Applicants with higher education level
  4. Properties in urban areas with high growth perspectives

Analyzing and interpreting the results

1. <u>Test the result on different samples: -</u>
   - Training data
   - Testing dataset

2. <u>Accuracy: -</u> **80.945%**

3. <u>Cross – Validation Score: -</u> **80.945%**