Data Munging, Manipulation, Exploratory analysis using Pandas

```
import pandas as pd
import numpy as np
#Coding for importing csv files in Google colab
from google.colab import files
import io
uploaded = files.upload()
df = pd.read_csv(io.BytesIO(uploaded['loan.csv']))
# Read csv loan.csv into a pandas dataframe
# Take a look at the first few rows
print(df)
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving loan.csv to loan.csv
      Loan_ID Gender Married  ... Loan_Amount_Term Credit_History Property_Area
0    LP001015   Male     Yes  ...            360.0            1.0         Urban
1    LP001022   Male     Yes  ...            360.0            1.0         Urban
2    LP001031   Male     Yes  ...            360.0            1.0         Urban
3    LP001035   Male     Yes  ...            360.0            NaN         Urban
4    LP001051   Male      No  ...            360.0            1.0         Urban
..        ...    ...     ...  ...              ...            ...           ...
362  LP002971   Male     Yes  ...            360.0            1.0         Urban
363  LP002975   Male     Yes  ...            360.0            1.0         Urban
364  LP002980   Male      No  ...            360.0            NaN     Semiurban
365  LP002986   Male     Yes  ...            360.0            1.0         Rural
366  LP002989   Male      No  ...            180.0            1.0         Rural

[367 rows x 12 columns]
```

To view the first 10 rows in the dataset

```
df.head(10)
```

```
df.columns
```

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

To calculate the statistical calculations for all numerical fields

```
df.describe()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	367.000000	367.000000	362.000000	361.000000	338.000000
mean	4805.599455	1569.577657	136.132597	342.537396	0.825444
std	4910.685399	2334.232099	61.366652	65.156643	0.380150
min	0.000000	0.000000	28.000000	6.000000	0.000000
25%	2864.000000	0.000000	100.250000	360.000000	1.000000
50%	3786.000000	1025.000000	125.000000	360.000000	1.000000
75%	5060.000000	2430.500000	158.000000	360.000000	1.000000
max	72529.000000	24000.000000	550.000000	480.000000	1.000000

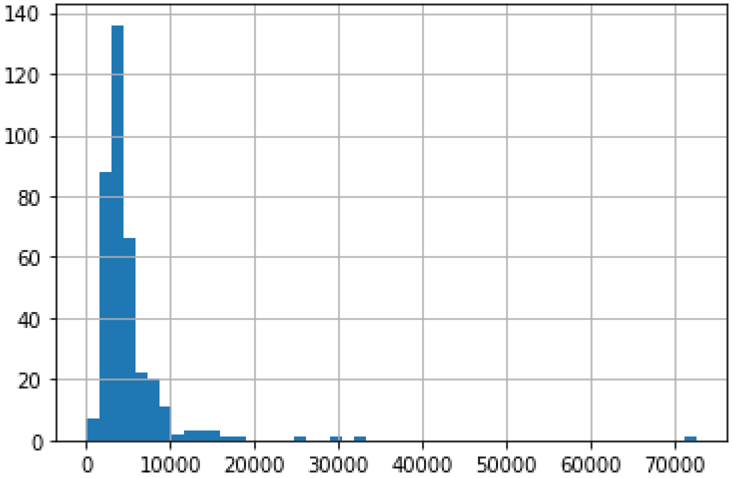
Distribution analysis using EDA

Analysis on Application income alone using histogram

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

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1c6ccd2ba8>
```



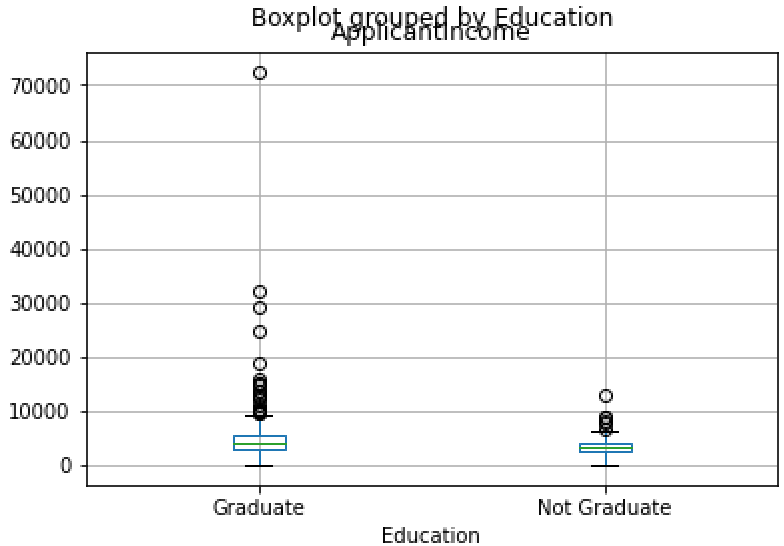
Analysis on Application income alone using boxplot

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

Analysis on Application income and Education using boxplot

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

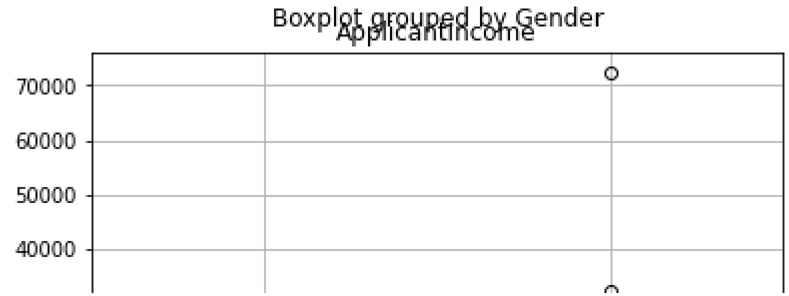
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1c6cbf3eb8>
```



Analysis on Application income and gender using boxplot

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

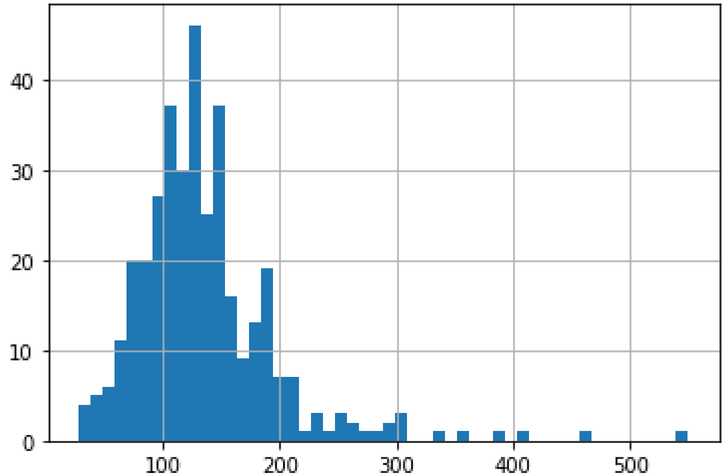
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1c6cd504e0>
```



Analysis on Loan Amount alone using histogram

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

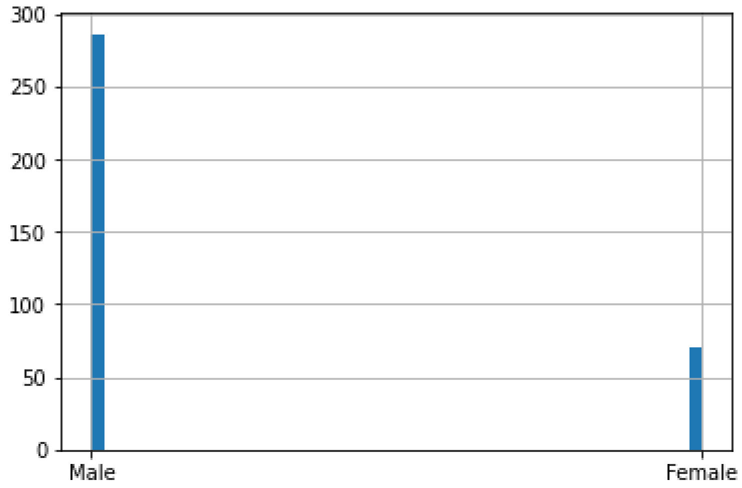
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1c6cb04208>
```



Analysis on Gender alone using histogram

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

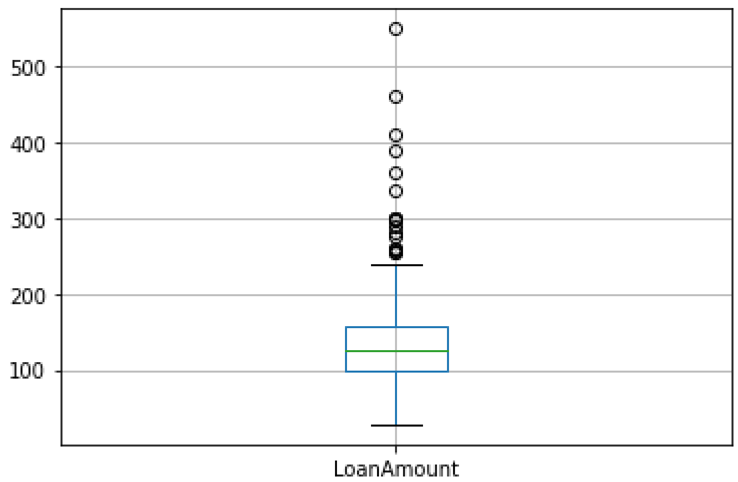
<matplotlib.axes._subplots.AxesSubplot at 0x7f1c6cad7240>



Analysis on Loan Amount alone using boxplot

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

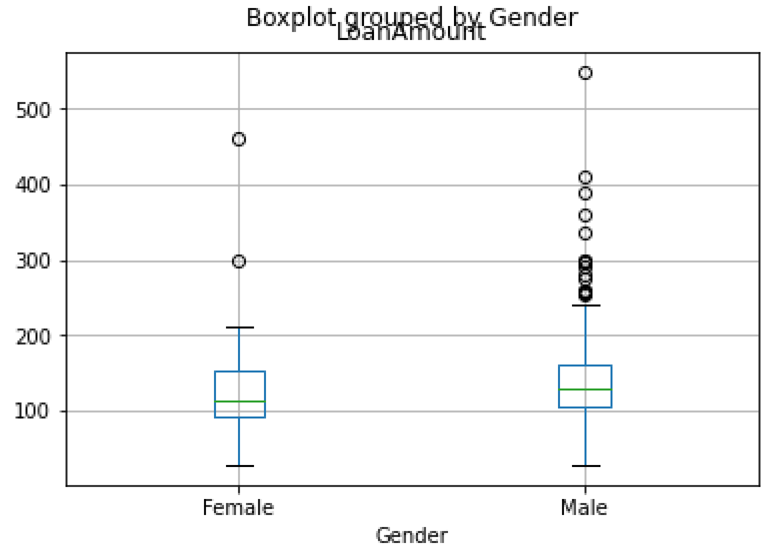
<matplotlib.axes._subplots.AxesSubplot at 0x7f1c6c8bd5c0>



Analysis on Loan Amount and gender using boxplot

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1c6ca97080>
```



Categorical variable analysis

```
print ('Frequency Table for Credit History:')
temp1=df['Credit_History'].value_counts(ascending=True)
print(temp1)

print ('Frequency Table for Education:')
temp2=df['Education'].value_counts(ascending=True)
print(temp2)
```

```
Frequency Table for Credit History:
0.0     59
1.0    279
Name: Credit_History, dtype: int64
Frequency Table for Education:
```

Applicants by Credit_History Analysis

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8,4))

#applicants by credit history
ax1 = fig.add_subplot(121)
ax1.set_xlabel('Credit_History')
ax1.set_ylabel('Count of Applicants')
ax1.set_title("Applicants by Credit_History")
temp1.plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0c9cb7a518>
```

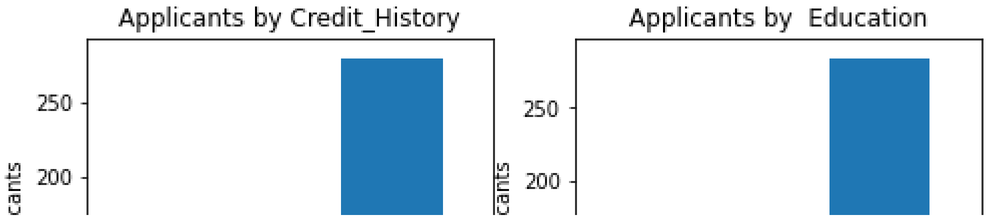
Applicants by Credit_History Analysis and Applicants by Education Analysis both hand in hand

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8,4))

#applicants by credit history
ax1 = fig.add_subplot(121)
ax1.set_xlabel('Credit_History')
ax1.set_ylabel('Count of Applicants')
ax1.set_title("Applicants by Credit_History")
temp1.plot(kind='bar')
print('')

#applicants by education
ax2 = fig.add_subplot(122)
ax2.set_xlabel('Education')
ax2.set_ylabel('Count of Applicants')
ax2.set_title("Applicants by  Education")
temp2.plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0c9c9fd6a0>
```



Check missing values in the dataset

```
df.apply(lambda x: sum(x.isnull()),axis=0)
```

```
Loan_ID              0
Gender              11
Married              0
Dependents          10
Education            0
Self_Employed       23
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           5
Loan_Amount_Term     6
Credit_History      29
Property_Area        0
dtype: int64
```

replacing missing loan amount with mean of the loanamount

```
df['LoanAmount'].fillna(df['LoanAmount'].mean(), inplace=True)
```

viewing the data set

```
df
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	C
0	LP001015	Male	Yes	0	Graduate	No	5720	0	110.0	360.0	
1	LP001022	Male	Yes	1	Graduate	No	3076	1500	126.0	360.0	
2	LP001031	Male	Yes	2	Graduate	No	5000	1800	208.0	360.0	
3	LP001035	Male	Yes	2	Graduate	No	2340	2546	100.0	360.0	
4	LP001051	Male	No	0	Not Graduate	No	3276	0	78.0	360.0	
...	...	...	...	...	...	...	...	...	...	...	
362	LP002971	Male	Yes	3+	Not Graduate	Yes	4009	1777	113.0	360.0	
363	LP002975	Male	Yes	0	Graduate	No	4158	709	115.0	360.0	
364	LP002980	Male	No	0	Graduate	No	3250	1993	126.0	360.0	
365	LP002986	Male	Yes	0	Graduate	No	5000	2393	158.0	360.0	
366	LP002989	Male	No	0	Graduate	Yes	9200	0	98.0	180.0	

367 rows × 12 columns

once again checking empty values

```
df.apply(lambda x: sum(x.isnull()),axis=0)
```

```
Loan_ID              0
Gender              11
Married              0
Dependents          10
Education            0
Self_Employed       23
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
```

checking Self_Employed

```
df['Self_Employed'].value_counts()
```

```
No     330
Yes     37
Name: Self_Employed, dtype: int64
```

As No is dominating, replacing the empty values with No

```
df['Self_Employed'].fillna('No',inplace=True)
```

checking Self_Employed once again

```
df['Self_Employed'].value_counts()
```

```
No     330
Yes     37
Name: Self_Employed, dtype: int64
```

checking Dependents

```
df['Dependents'].value_counts()
```

```
0     210
2      59
1      58
3+     40
Name: Dependents, dtype: int64
```

As 0 is dominating , replace empty values with 0

```
df['Dependents'].fillna('0',inplace=True)
```

once again checking Dependents

```
df['Dependents'].value_counts()
```

```
0     210
2      59
1      58
3+     40
Name: Dependents, dtype: int64
```

once again checking empty values

```
df.apply(lambda x: sum(x.isnull()),axis=0)
```

```
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
dtype: int64
```

checking Gender

```
df['Gender'].value_counts()
```

```
Male      297
Female     70
Name: Gender, dtype: int64
```

male is dominated with 80% so replace empty values with Male

```
df['Gender'].fillna('Male',inplace=True)
```

once again checking Gender

```
df['Gender'].value_counts()
```

```
Male      297
Female     70
Name: Gender, dtype: int64
```

once again checking empty values

```
df.apply(lambda x: sum(x.isnull()),axis=0)
```

```
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
dtype: int64
```

checking Loan_Amount_Term

```
df['Loan_Amount_Term'].value_counts()
```

```
360.0    317
180.0     22
480.0      8
300.0      7
240.0      4
84.0       3
6.0        1
120.0      1
36.0       1
350.0      1
12.0       1
60.0       1
```

As Loan_Amount_Term=360 is dominating,replace empty values with 360

```
df['Loan_Amount_Term'].fillna(360.0,inplace=True)
```

checking Loan_Amount_Term

```
df['Loan_Amount_Term'].value_counts()
```

```
360.0    317
180.0     22
480.0      8
300.0      7
240.0      4
84.0       3
6.0        1
120.0      1
36.0       1
350.0      1
12.0       1
60.0       1
Name: Loan_Amount_Term, dtype: int64
```

once again checking empty values

```
df.apply(lambda x: sum(x.isnull()),axis=0)
```

```
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
dtype: int64
```

checking Credit_History

```
df['Credit_History'].value_counts()
```

```
1.0    308
0.0     59
Name: Credit_History, dtype: int64
```

yes (1.0) is dominating

```
df['Credit_History'].fillna(1.0,inplace=True)
```

once again checking empty values

```
df.apply(lambda x: sum(x.isnull()),axis=0)
```

```
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
dtype: int64
```

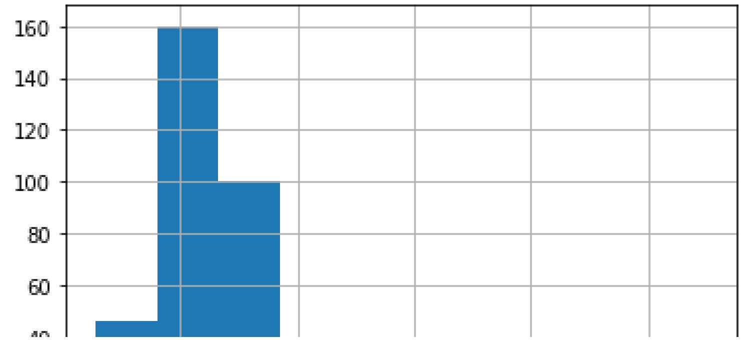
Finally all missing values are clear

Then go to the next phase of normalization

how to treat for extreme values in distribution of LoanAmount and ApplicantIncome

```
df['LoanAmount'].hist(bins=10)
```

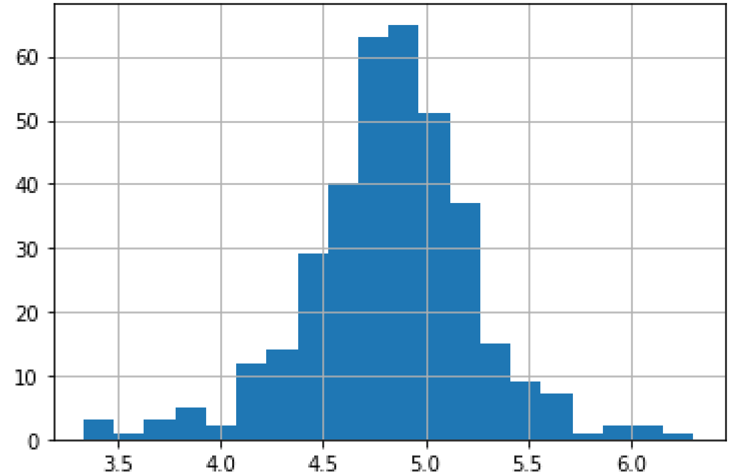
<matplotlib.axes._subplots.AxesSubplot at 0x7f1c6c725198>



creating LoanAmount_log column to treate outliers and extreme values

```
df['LoanAmount_log'] = np.log(df['LoanAmount'])
df['LoanAmount_log'].hist(bins=20)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f1c6c76b4e0>



The normalized data set with artificial field LoanAmount_log

```
df
```

```
df
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	C
0	LP001015	Male	Yes	0	Graduate	No	5720	0	110.0	360.0	
1	LP001022	Male	Yes	1	Graduate	No	3076	1500	126.0	360.0	
2	LP001031	Male	Yes	2	Graduate	No	5000	1800	208.0	360.0	
3	LP001035	Male	Yes	2	Graduate	No	2340	2546	100.0	360.0	
4	LP001051	Male	No	0	Not Graduate	No	3276	0	78.0	360.0	
...	...	...	...	...	...	...	...	...	...	...	
362	LP002971	Male	Yes	3+	Not Graduate	Yes	4009	1777	113.0	360.0	
363	LP002975	Male	Yes	0	Graduate	No	4158	709	115.0	360.0	
364	LP002980	Male	No	0	Graduate	No	3250	1993	126.0	360.0	
365	LP002986	Male	Yes	0	Graduate	No	5000	2393	158.0	360.0	
366	LP002989	Male	No	0	Graduate	Yes	9200	0	98.0	180.0	

367 rows × 13 columns

Loading already treated dataset from repository to predict the loan approval with Loan_status field

```
url = "https://raw.githubusercontent.com/callxpert/datasets/master/Loan-applicant-details.csv"
names = ['Loan_ID','Gender','Married','Dependents','Education','Self_Employed','ApplicantIncome','CoapplicantIncome'
df = pd.read_csv(url, names=names)
print(df)
```

```
      Loan_ID  Gender Married  ... Credit_History Property_Area Loan_Status
0    LP001003    Male     Yes  ...              1         Rural           N
1    LP001005    Male     Yes  ...              1         Urban           Y
2    LP001006    Male     Yes  ...              1         Urban           Y
3    LP001008    Male      No  ...              1         Urban           Y
4    LP001011    Male     Yes  ...              1         Urban           Y
..        ...     ...     ...  ...            ...           ...         ...
475  LP002978  Female      No  ...              1         Rural           Y
476  LP002979    Male     Yes  ...              1         Rural           Y
477  LP002983    Male     Yes  ...              1         Urban           Y
478  LP002984    Male     Yes  ...              1         Urban           Y
479  LP002990  Female      No  ...              0     Semiurban           N

[480 rows x 13 columns]
```

let us check the null values in the new dataset downloaded from the repository

```
df.apply(lambda x: sum(x.isnull()),axis=0)
```

```
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
```

Lets take a peek at the data

```
print(dataset.head(20))
```

```
      Loan_ID  Gender Married  ... Credit_History Property_Area Loan_Status
0   LP001003    Male     Yes  ...              1         Rural           N
1   LP001005    Male     Yes  ...              1         Urban           Y
2   LP001006    Male     Yes  ...              1         Urban           Y
3   LP001008    Male      No  ...              1         Urban           Y
4   LP001011    Male     Yes  ...              1         Urban           Y
5   LP001013    Male     Yes  ...              1         Urban           Y
6   LP001014    Male     Yes  ...              0     Semiurban           N
7   LP001018    Male     Yes  ...              1         Urban           Y
8   LP001020    Male     Yes  ...              1     Semiurban           N
9   LP001024    Male     Yes  ...              1         Urban           Y
10  LP001028    Male     Yes  ...              1         Urban           Y
11  LP001029    Male      No  ...              1         Rural           N
12  LP001030    Male     Yes  ...              1         Urban           Y
13  LP001032    Male      No  ...              1         Urban           Y
14  LP001036  Female      No  ...              0         Urban           N
15  LP001038    Male     Yes  ...              1         Rural           N
16  LP001043    Male     Yes  ...              0         Urban           N
17  LP001046    Male     Yes  ...              1         Urban           Y
18  LP001047    Male     Yes  ...              0     Semiurban           N
19  LP001066    Male     Yes  ...              1     Semiurban           Y

[20 rows x 13 columns]
```

Lets load the required libraries for analyzing our developed Model

```
#Load libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.model_selection import train_test_split
```

sklearn requires all inputs to be numeric, we should convert all our categorical variables into numeric by encoding the categories. This can be done using the following code:

```
from sklearn.preprocessing import LabelEncoder
var_mod = ['Gender','Married','Dependents','Education','Self_Employed','Property_Area','Loan_Status']
le = LabelEncoder()
for i in var_mod:
    dataset[i] = le.fit_transform(dataset[i])
```

Splitting the Data set As we have seen already, In Machine learning we have two kinds of datasets

Training dataset - used to train our model ***Testing dataset*** - used to test if our model is making accurate predictions Our dataset has 480 records. We are going to use 80% of it for training the model and 20% of the records to evaluate our model. copy paste the below commands to prepare our data sets

Though our dataset has lot of columns, we are only going to use the Income fields, loan amount, loan duration and credit history fields to train our model.

```
array = dataset.values
X = array[:,6:11]
Y = array[:,12]
Y=Y.astype('int')
x_train, x_test, y_train, y_test = model_selection.train_test_split(X, Y, test_size=0.2, random_state=7)
print(X)
print(Y)
```

```
[[4583 1508.0 128 360 1]
 [3000 0.0 66 360 1]
 [2583 2358.0 120 360 1]
 ...
 [8072 240.0 253 360 1]
 [7583 0.0 187 360 1]
 [4583 0.0 133 360 0]]
[0 1 1 1 1 1 0 1 0 1 1 0 1 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1 1 1 1 0 1
 1 1 0 0 0 1 1 0 1 1 1 1 0 0 0 0 1 1 0 1 1 1 1 0 0 0 0 1 0 1 1 1 1 1 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 1 1 1 1 0 0 1 0 0 0 1 1 1 1 1
 1 1 0 1 0 1 0 0 1 1 1 1 1 0 0 1 1 0 1 0 1 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 1
 0 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 1
 1 0 1 1 1 0 1 1 1 1 0 0 1 1 0 1 0 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 0 0 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 0 1 1 0 1
 1 1 1 1 0 0 1 1 1 1 1 1 0 1 0 1 1 0 0 1 0 1 1 1 0 0 1 0 1 1 1 0 1 1 0 1 1
 1 1 0 1 1 1 1 1 1 1 0 1 1 0 0 0 1 1 0 1 1 1 0 0 0 0 1 0 1 0 0 1 1 1 1 1 1
 0 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 0 1 0 1 1 0 1 0 0 1 0 1 1
 1 1 1 1 1 1 1 1 1 0 0 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 1 0 1 1 0 0 1 1 0 0 1
 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 0 1 1 0 1 0 1 1 0 0 1 1 1 1 1 1 1 1 0 1 0
 0 0 1 1 0 0 1 1 1 1 0 1 0 0 1 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0]
```

Evaluating the model and training the Model

Double-click (or enter) to edit

```
model = LogisticRegression()
model.fit(x_train,y_train)
predictions = model.predict(x_test)
print(accuracy_score(y_test, predictions))
```

```
0.7708333333333334
```

Double-click (or enter) to edit

```
model = DecisionTreeClassifier()
```

```
model.fit(x_train,y_train)
predictions = model.predict(x_test)
print(accuracy_score(y_test, predictions))
```

```
0.6354166666666666
```

Double-click (or enter) to edit

```
model = RandomForestClassifier(n_estimators=100)
model.fit(x_train,y_train)
predictions = model.predict(x_test)
print(accuracy_score(y_test, predictions))
```

```
0.7708333333333334
```