Loan Application Status Prediction

Submitted by:

Nidhi Charde

ACKNOWLEDGMENT

During completion of this project, I refer various sources like GitHub, Data Trained institute’s reference materials .

INTRODUCTION

Problem Definition –

This dataset includes details of applicants who have applied for loan. The dataset includes details like credit history, loan amount, their income, dependents etc.

Independent Variables:

- Loan_ID

- Gender

- Married

- Dependents

- Education

- Self-employed

- Applicant Income

- CoapplicantIncome

- Loan_Amount

- Loan_Amount_Term

- Credit History

- Property_Area

-Dependent Variable (Target Variable):

- Loan_Status

You have to build a model that can predict whether the loan of the applicant will be approved or not on the basis of the details provided in the dataset.

Analytical Problem Framing

Mathematical/ Analytical Modeling of the Problem

During project building, we run statistical analysis of all available attributes, analyse existing data structure.

Tasks that have been performed from data point of view –

1. Analysis of available data types
2. Visual data analysis
3. Correlation analysis
4. Outlier detection
5. Analysis and definition of the ‘’target’’ variable.

Based on the results and insight obtained regarding these steps, we have a better understanding of what variables we will be able to generate at the data preparation stage and what the system architecture will look like.

Data Analysis –

The dataset that we are going to used can be found on below mentioned link –

https://raw.githubusercontent.com/dsrscientist/DSData/master/loan_prediction.csv

The purpose of this project is to predict whether the loan of the applicant will be approved or not on the basis of the details provided in the dataset.

The datasets has 13 independent variables that range from Loan_ID to Loan Status.

.

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0

614 rows × 13 columns

There is total 614 rows and 13 columns

```
In [7]: df.shape #Dimension of dataset

Out[7]: (614, 13)
```

In above dataset, Loan_ID, Gender, Married, Dependents, Education, Self_Employed, Property Area, Loan Status has object dataset, ApplicantIncome has integer dataset and CoapplicantIncome, Loan Amount, Loan_Amount_Term, Credit_History has float dataset.

df.dtype helps to know about data type.

```
In [10]: df.dtypes   #datatype

Out[10]: 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
         dtype: object
```

In above datasets, the differenicate between numeric and categorical data for Loan application.

```
In [126]: numeric_features=[features for features in df.columns if df[features].dtype!='O']
          print('Number Of Numeric Features=',len(numeric_features))
          numeric_features

          Number Of Numeric Features= 5

Out[126]: ['ApplicantIncome',
           'CoapplicantIncome',
           'LoanAmount',
           'Loan_Amount_Term',
           'Credit_History']

In [127]: categorical_features=[features for features in df.columns if df[features].dtype=='O']
          print('Number Of Categorical Features=',len(categorical_features))
          categorical_features

          Number Of Categorical Features= 8

Out[127]: ['Loan_ID',
           'Gender',
           'Married',
           'Dependents',
           'Education',
           'Self_Employed',
           'Property_Area',
           'Loan_Status']
```

2. Visulization and Data Inputs- Logic- Output Relationship

Input parameters (features) and Output (labels/target values) are two important parameters of any dataset. Based on features, target values changed. So, it is important to analysed features parameter to predict correct target values.

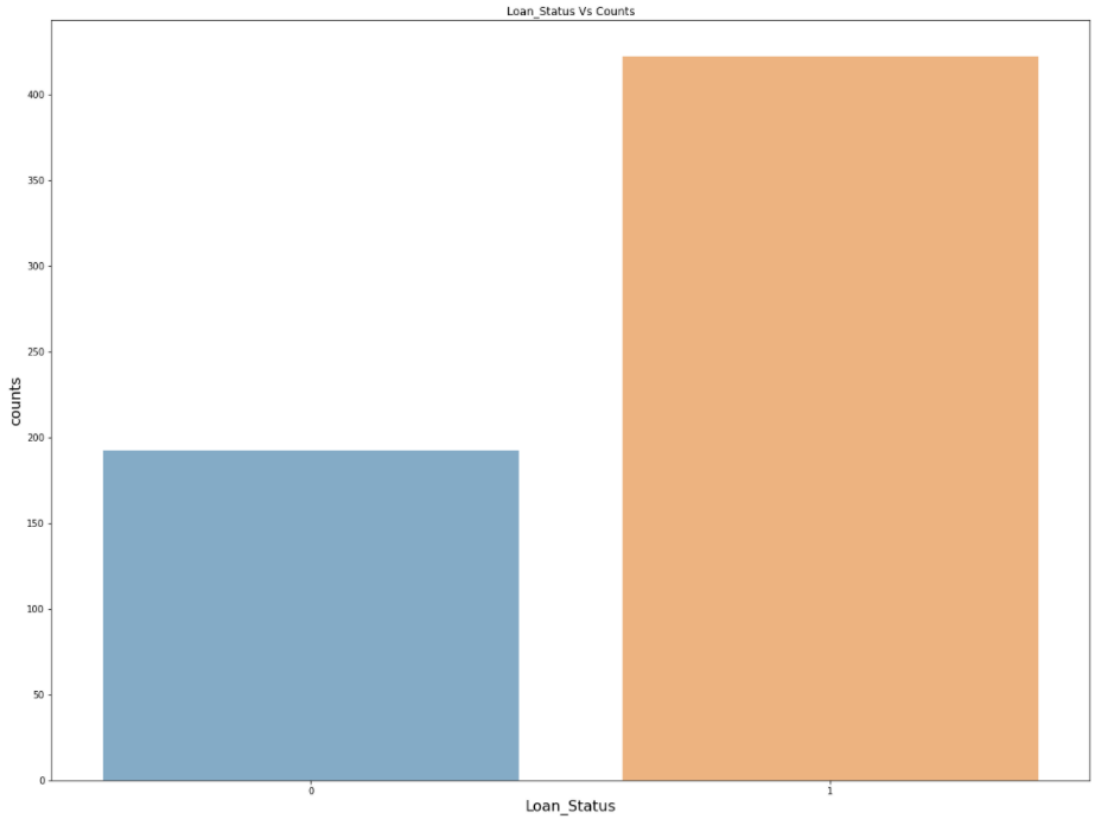
In this project, 'Loan_Status' is taken as output parameter. Output is based on input parameters like Gender, Married, Dependents etc.

```
In [41]: df['Loan_Status'].value_counts() ##gives count
```

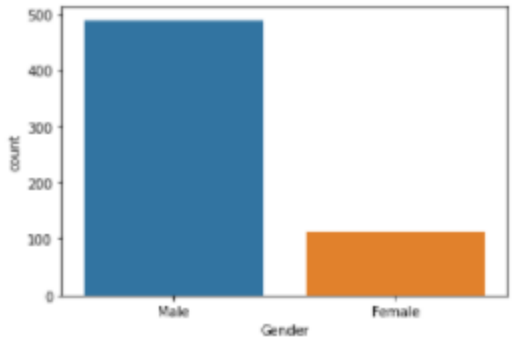
```
Out[41]: 1    422
         0    192
         Name: Loan_Status, dtype: int64
```

```
In above outcome, There are total 422 - 1's(Y) and 192 - 0's (N).
```

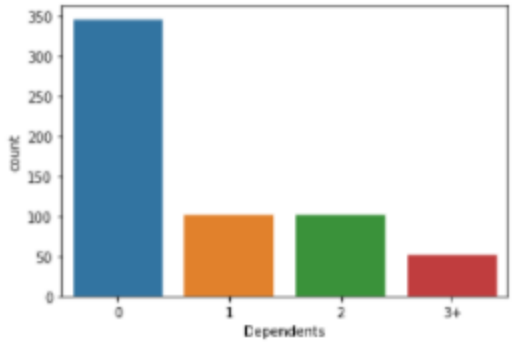
```
plt.figure(figsize=(20,15))
sns.barplot(data.index,data.values,alpha=0.6)
plt.xlabel('Loan_Status', fontsize=16)
plt.ylabel('counts',fontsize=16)
plt.title('Loan_Status Vs Counts')
plt.show()
```



```
Out[133]: <AxesSubplot:xlabel='Gender', ylabel='count'>
```




```
In [134]: sns.countplot(df['Dependents'],data=df)
Out[134]: <AxesSubplot:xlabel='Dependents', ylabel='count'>
```




```
In [135]: sns.countplot(df['Education'],data=df)
Out[135]: <AxesSubplot:xlabel='Education', ylabel='count'>
```

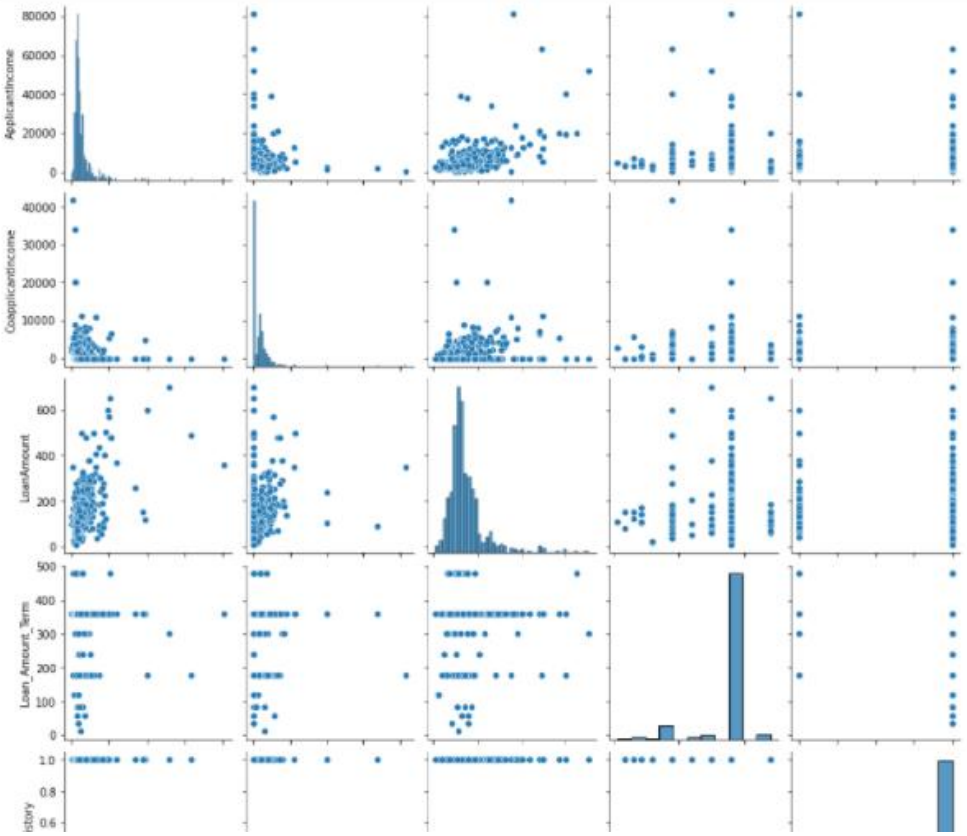
In given we can show all categorical graphs are here by using countplot.

In Given Loan Application datasets, I applying pairplot graphs it means pairplot is the difference between column and start analysis.

Sns.pairplot (data=df,color=blue)

Plt.show

```
In [138]: sns.pairplot(df)
          plt.show()
```



df.describe method gives stastical details like count, mean, std, min, max, 25%, 50%, 75%.

```
In [44]: df.describe()  #statstical summary
```

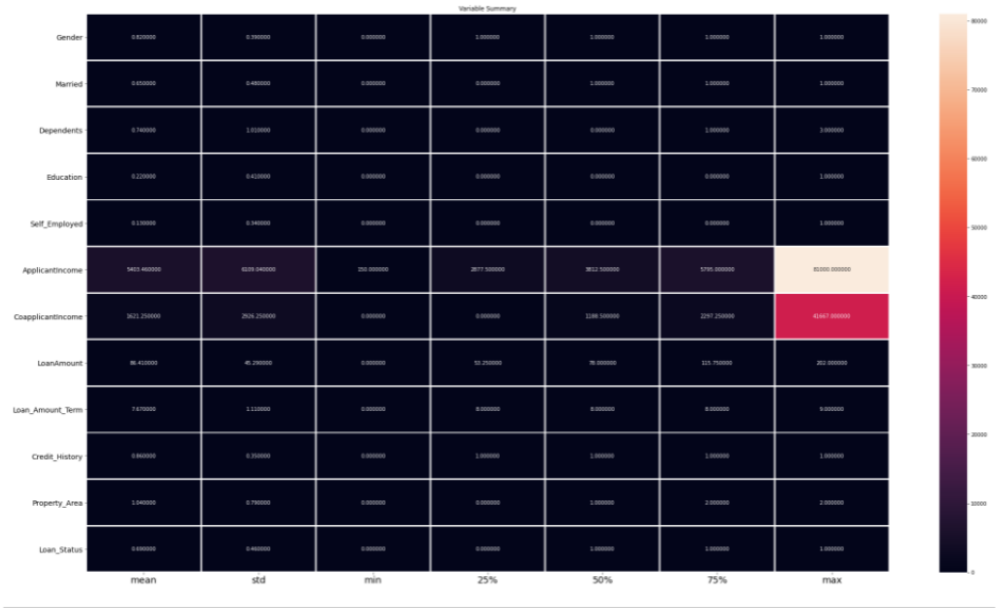
Out[44]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000
mean	0.817590	0.653094	0.744300	0.218241	0.133550	5403.459283	1621.245798	86.410423	7.667752	0.855049
std	0.386497	0.476373	1.009623	0.413389	0.340446	6109.041673	2926.248369	45.292390	1.109224	0.352339
min	0.000000	0.000000	0.000000	0.000000	0.000000	150.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	0.000000	0.000000	0.000000	0.000000	2877.500000	0.000000	53.250000	8.000000	1.000000
50%	1.000000	1.000000	0.000000	0.000000	0.000000	3812.500000	1188.500000	78.000000	8.000000	1.000000
75%	1.000000	1.000000	1.000000	0.000000	0.000000	5795.000000	2297.250000	115.750000	8.000000	1.000000
max	1.000000	1.000000	3.000000	1.000000	1.000000	81000.000000	41667.000000	202.000000	9.000000	1.000000

Above outcome shows, There are outiler present in dataset by comparing mean and 50% value. There is major difference between them like in Married, Dependents, ApplicantIncome, CoapplicantIncome, LoanAmount etc.

Let’s visualized above variable summary –

```
In [45]: #lets visulized dataset

plt.figure(figsize=(35,20))
sns.heatmap(round(df.describe()[1:].transpose(),2),linewidth=2,annot=True,fmt="f")
plt.xticks(fontsize=18)
plt.yticks(fontsize=14)
plt.title("Variable Summary")
plt.show()
```



To check for correlation of features with that of label value, we used corr. Below outcome, we received it–

```
In [46]: corr_mat = df.corr() # Checking for correlation
```

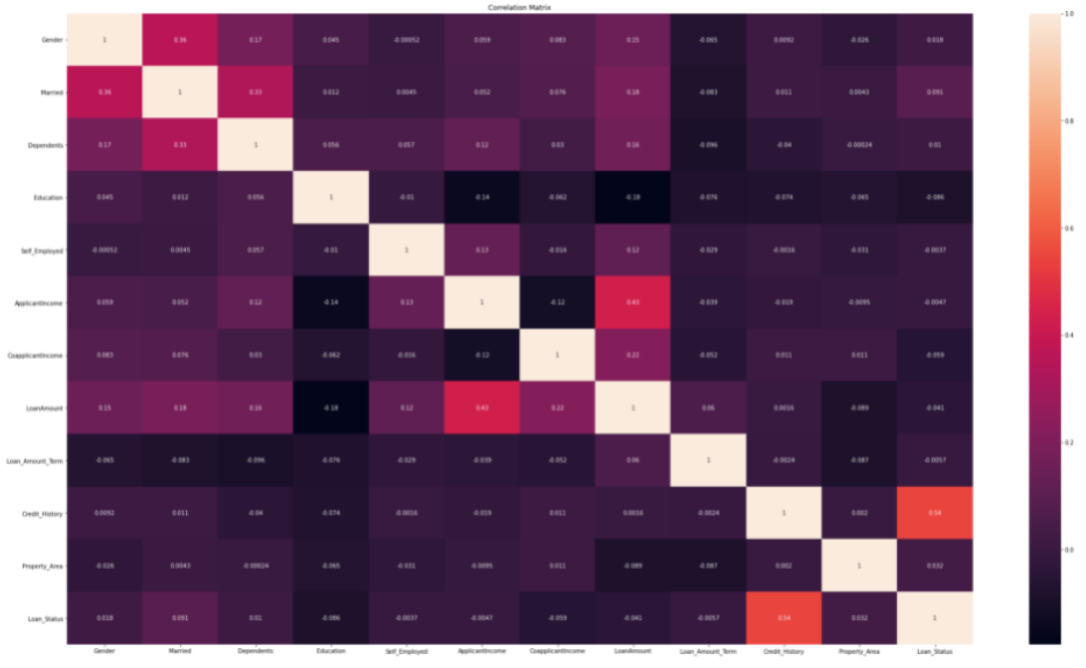
```
In [47]: corr_mat
```

Out[47]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credi
Gender	1.000000	0.364569	0.172914	0.045364	-0.000525	0.058809	0.082912	0.148262	-0.065494	
Married	0.364569	1.000000	0.334216	0.012304	0.004489	0.051708	0.075948	0.183456	-0.082642	
Dependents	0.172914	0.334216	1.000000	0.055752	0.056798	0.118202	0.030430	0.161335	-0.096377	
Education	0.045364	0.012304	0.055752	1.000000	-0.010383	-0.140760	-0.062290	-0.176956	-0.076414	
Self_Employed	-0.000525	0.004489	0.056798	-0.010383	1.000000	0.127180	-0.016100	0.117258	-0.029184	
ApplicantIncome	0.058809	0.051708	0.118202	-0.140760	0.127180	1.000000	-0.116605	0.430883	-0.038808	
CoapplicantIncome	0.082912	0.075948	0.030430	-0.062290	-0.016100	-0.116605	1.000000	0.216525	-0.052472	
LoanAmount	0.148262	0.183456	0.161335	-0.176956	0.117258	0.430883	0.216525	1.000000	0.060160	
Loan_Amount_Term	-0.065494	-0.082642	-0.096377	-0.076414	-0.029184	-0.038808	-0.052472	0.060160	1.000000	
Credit_History	0.009170	0.010938	-0.040160	-0.073658	-0.001550	-0.018615	0.011134	0.001587	-0.002379	
Property_Area	-0.025752	0.004257	-0.000244	-0.065243	-0.030860	-0.009500	0.010522	-0.089163	-0.086578	
Loan_Status	0.017987	0.091478	0.010118	-0.085884	-0.003700	-0.004710	-0.059187	-0.040926	-0.005679	

Let's visualized it –

```
In [48]: # visulization of correlated values.

plt.figure(figsize=(35,20))
sns.heatmap(corr_mat,annot=True)
plt.title("Correlation Matrix")
plt.show()
```



All columns of database are positively correlated.

Gender has 1.8%, Married has 9.1%, Dependents has 1%, Education has -8.6%, Self_Employed has -0.37%, ApplicatIncome has-0.47%, Coapplicant has -5.9%, LoanAmount has -4.1%, Loan_Amount_Term has -0.57%, Credit_History has 54%, Property_Area has 3.2% coorelation with target value.

Max Correlation: Credit_History

Min Correlation: Education

3.Data Pre-processing

At this stage, the main task is to prepare data for machine learning modelling. It is important to properly aggregate data, create all available variables.

It is also very important to define the target variables.

In data processing stage, we checked for dimension of data (df.shape), Type of data (df.info()), Null values (df.isnull().sum()) present in dataset. If null values present in dataset, then fill it with data with help of mean/median or mode methods.

Sns.heatmap(df.isnull()) helps to visualized it better.

There are null values present in this dataset. We can check it with the help of df.isnull().sum().

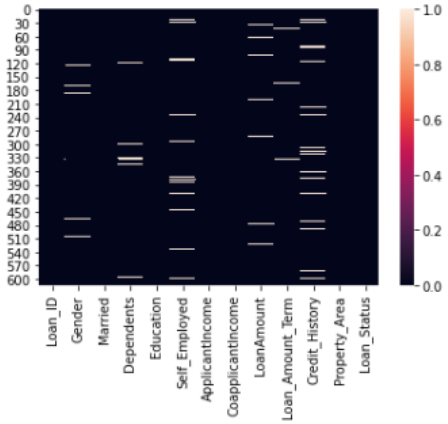
In above outcome, Gender, Married, Dependents, Self_Employed, LoanAmount, Loan_Amount_Term and Credit_History has null values.

```
In [11]: df.isnull().sum()  #to check for null value.

Out[11]: 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
```

Below outcome represent null values. Black colour not uniformly distributed.

```
In [12]: #Let's visualized null values.

         sns.heatmap(df.isnull())
```

```
Out[12]: <AxesSubplot:>
```



In given datasets after fillna()using simple imputer method we have to all null values are fill with help of mean and mode method.

```
In [13]: df['Gender'].fillna(df['Gender'].mode()[0],inplace=True)

In [14]: df['Married'].fillna(df['Married'].mode()[0],inplace=True)

In [15]: df['Dependents'].fillna(df['Dependents'].mode()[0],inplace=True)

In [16]: df['Self_Employed'].fillna(df['Self_Employed'].mode()[0],inplace=True)

In [17]: df['LoanAmount'].fillna(df['LoanAmount'].mode()[0],inplace=True)

In [18]: df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mode()[0],inplace=True)

In [19]: df['Credit_History'].fillna(df['Credit_History'].mode()[0],inplace=True)
```

In above outcome, Null values filled with help of mode method. Let's again check for it and visualized it.

```
In [20]: df.isnull().sum() #checking for null values.
```

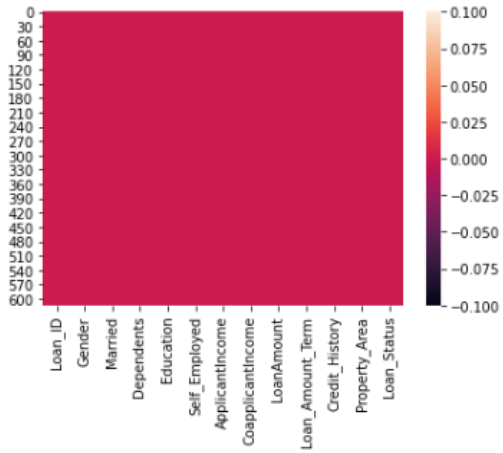
```
Out[20]: Loan_ID              0
         Gender               0
         Married              0
         Dependents           0
         Education            0
         Self_Employed        0
         ApplicantIncome      0
         CoapplicantIncome    0
         LoanAmount           0
         Loan_Amount_Term     0
         Credit_History       0
         Property_Area        0
         Loan_Status          0
         dtype: int64
```

In above outcome, There are no null values now.

Red color uniformly distributed, which represent no null values in dataset-

```
In [21]: sns.heatmap(df.isnull())#Helps to viasalized it better for null values.
```

```
Out[21]: <AxesSubplot:>
```



Label Encoding

For string/object type of data, it is important to convert it into integer datatype. So for these purpose Label Encoder() used.

Married, Dependents, Self_Employed, LoanAmount, Loan_Amount_Term and Credit_history dataset converted from string to integer dataset.

Dataset after converting string dataset into integer one –

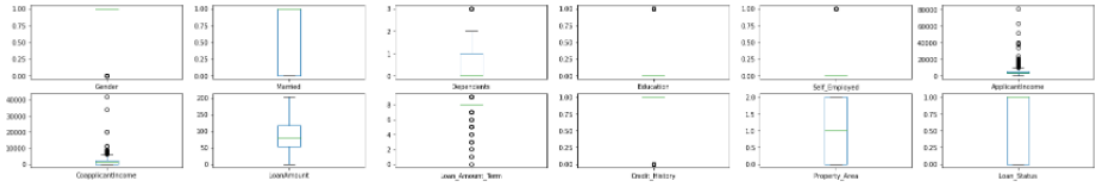
```
In [151]: from sklearn.preprocessing import LabelEncoder #importing required library.

In [152]: list=['Gender','Married','Education','Self_Employed','Property_Area','Loan_Status']

In [153]: lb=LabelEncoder()

In [154]: for i in list:
              df[i]=lb.fit_transform(df[i])

In [155]: df.head()
```

Out[155]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_A
0	1	0	0	0	0	5849	0.0	120.0	360.0	1.0	
1	1	1	1	0	0	4583	1508.0	128.0	360.0	1.0	
2	1	1	0	0	1	3000	0.0	66.0	360.0	1.0	
3	1	1	0	1	0	2583	2358.0	120.0	360.0	1.0	
4	1	0	0	0	0	6000	0.0	141.0	360.0	1.0	

Now visualized outlier present in dataset with the help of box plot –

```
In [49]: df.plot(kind='box',subplots =True, layout=(8,6),figsize=(30,20))

Out[49]: Gender                AxesSubplot(0.125,0.799681;0.110714x0.0803191)
         Married             AxesSubplot(0.257857,0.799681;0.110714x0.0803191)
         Dependents          AxesSubplot(0.390714,0.799681;0.110714x0.0803191)
         Education           AxesSubplot(0.523571,0.799681;0.110714x0.0803191)
         Self_Employed       AxesSubplot(0.656429,0.799681;0.110714x0.0803191)
         ApplicantIncome     AxesSubplot(0.789286,0.799681;0.110714x0.0803191)
         CoapplicantIncome      AxesSubplot(0.125,0.703298;0.110714x0.0803191)
         LoanAmount          AxesSubplot(0.257857,0.703298;0.110714x0.0803191)
         Loan_Amount_Term    AxesSubplot(0.390714,0.703298;0.110714x0.0803191)
         Credit_History      AxesSubplot(0.523571,0.703298;0.110714x0.0803191)
         Property_Area       AxesSubplot(0.656429,0.703298;0.110714x0.0803191)
         Loan_Status         AxesSubplot(0.789286,0.703298;0.110714x0.0803191)
         dtype: object
```



```
Above both outcome show outlier present in columns.
```

df.skew() helps to check skewness present in dataset –

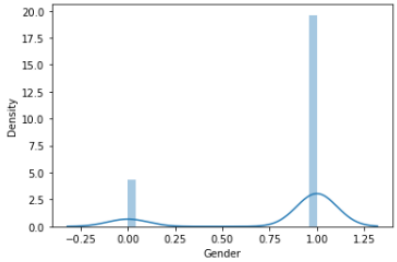
```
In [50]: df.skew() #to check skewness

Out[50]: Gender              -1.648795
         Married             -0.644850
         Dependents           1.015551
         Education            1.367622
         Self_Employed        2.159796
         ApplicantIncome      6.539513
         CoapplicantIncome    7.491531
         LoanAmount           0.517449
         Loan_Amount_Term    -3.316702
         Credit_History      -2.021971
         Property_Area       -0.066196
         Loan_Status         -0.809998
         dtype: float64
```

Normalized data range has skewnees ranges between +0.5 to -0.5.

Columns has skewness - Gender, Married, Dependents, Education, Self_Employed, ApplicantIncome, CoapplicantIncome, LoanAmount, Loan_Amount_Term,Credit_History and Loan_Status.

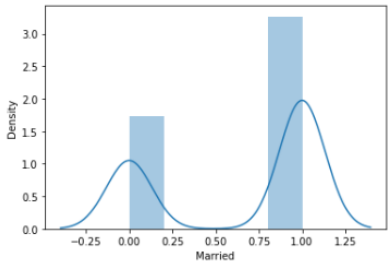
Let's visualized skewness with help of distplot.

```
In [51]: sns.distplot(df['Gender'])
Out[51]: <AxesSubplot:xlabel='Gender', ylabel='Density'>
```



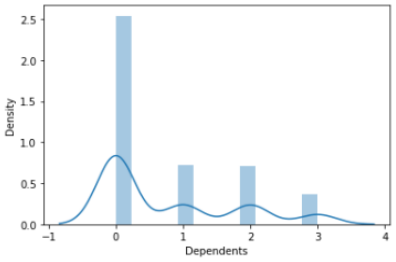

```
It showing skewness for Gender dataset.
```

```
In [52]: sns.distplot(df['Married'])
Out[52]: <AxesSubplot:xlabel='Married', ylabel='Density'>
```



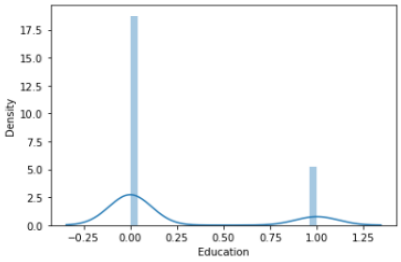

```
It showing skewness for Married dataset.
```

```
In [53]: sns.distplot(df['Dependents'])
Out[53]: <AxesSubplot:xlabel='Dependents', ylabel='Density'>
```



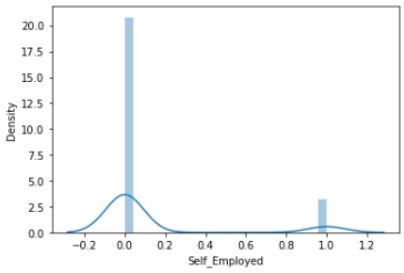

```
It showing skewness for Dependents dataset.
```

```
In [54]: sns.distplot(df['Education'])
Out[54]: <AxesSubplot:xlabel='Education', ylabel='Density'>
```



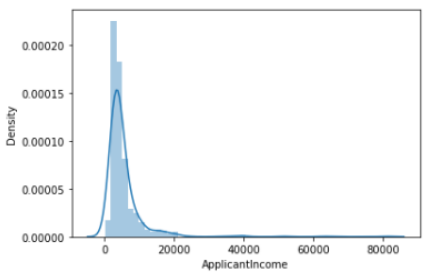

```
It showing skewness for Education dataset.
```

```
In [55]: sns.distplot(df['Self_Employed'])
Out[55]: <AxesSubplot:xlabel='Self_Employed', ylabel='Density'>
```



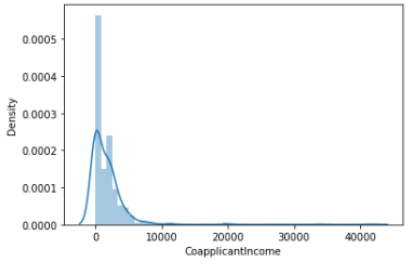

```
It showing skewness for Self_Employed dataset.
```

```
In [56]: sns.distplot(df['ApplicantIncome'])
Out[56]: <AxesSubplot:xlabel='ApplicantIncome', ylabel='Density'>
```



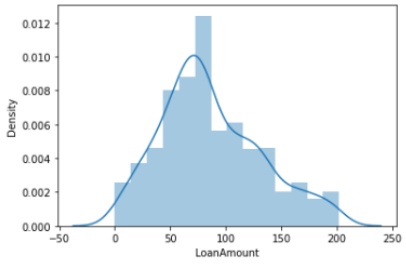

```
It showing skewness for ApplicantIncome dataset.
```

```
In [57]: sns.distplot(df['CoapplicantIncome'])
Out[57]: <AxesSubplot:xlabel='CoapplicantIncome', ylabel='Density'>
```




```
It showing skewness for CoapplicantIncome dataset.
```

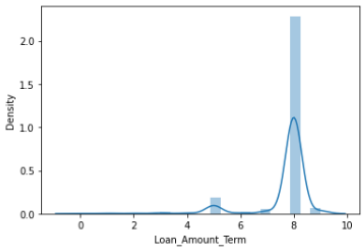
```
In [58]: sns.distplot(df['LoanAmount'])
Out[58]: <AxesSubplot:xlabel='LoanAmount', ylabel='Density'>
```




```
It showing skewness for LoanAmount dataset.
```

In [59]:
```
sns.distplot(df['Loan_Amount_Term'])
```

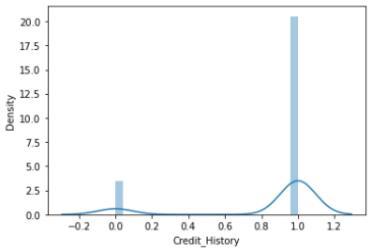
Out[59]:
```
<AxesSubplot:xlabel='Loan_Amount_Term', ylabel='Density'>
```




```
It showing skewness for Loan_Amount_Term dataset.
```

In [60]:
```
sns.distplot(df['Credit_History'])
```

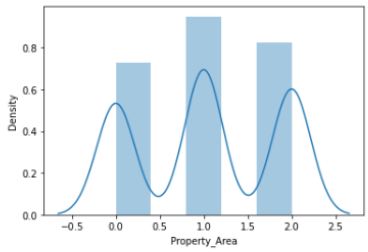
Out[60]:
```
<AxesSubplot:xlabel='Credit_History', ylabel='Density'>
```




```
It showing skewness for Credit_History dataset.
```

In [61]:
```
sns.distplot(df['Property_Area'])
```

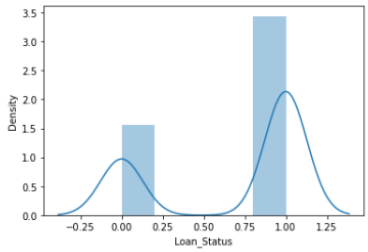
Out[61]:
```
<AxesSubplot:xlabel='Property_Area', ylabel='Density'>
```




```
It showing skewness for Property_Area dataset.
```

In [62]:
```
sns.distplot(df['Loan_Status'])
```

Out[62]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='Density'>
```




```
It showing skewness for Loan_Status dataset.
```

Remove skewness with the help of zscore. After applying zscore 2.44% data is lost.

```
In [173]: df.skew()  #to check skewness

Out[173]: Gender              -1.622920
          Married             -0.630211
          Education            1.306588
          Self_Employed        2.252848
          ApplicantIncome      2.148522
          CoapplicantIncome    1.350517
          LoanAmount           1.163426
          Loan_Amount_Term    -2.098806
          Credit_History      -1.976043
          Property_Area       -0.055332
          Loan_Status         -0.822635
          dtype: float64

In [174]: dfNumCols = df[['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term','Credit_History']]

In [175]: Q1 = df.quantile(0.25)
          Q3 = df.quantile(0.75)
          IQR = Q3 - Q1

          df_new= df[~((new_df < (Q1 - 1.5 * IQR)) |(new_df > (Q3 + 1.5 * IQR))).any(axis=1)]

In [176]: df_new.shape

Out[176]: (214, 12)
```

In given datsets ,after removing skewness using quantile method the new shape is 214 rows and 12 columns here and loss percentage will be

To applyimg z score method on numercal data on given datasets the new shape will be 577 rows and 12 coulmns

```
In [167]: df.columns

Out[167]: Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
                 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
                 'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
                dtype='object')

In [168]: dfNumCols = df[['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term','Credit_History']]

In [169]: #applying zscore on only the dataset which has numeric columns

          z=np.abs(zscore(dfNumCols))

          df_new =df[(z<3).all(axis=1)]

In [170]: df_new.shape

Out[170]: (577, 12)

          #After applying zscore method,We got new dataset with 577 rows and 12 columns

In [171]: loss_of_data=(614-577)/614*100
          loss_of_data

Out[171]: 6.026058631921824
```

After applying zscore method,we got new datasets and loss percentage data is 6.0260

Dividing dataset into label and features –

Splitting datasets into x and y only numeric datasets

```
Out[176]: (214, 12)
```

Step 5. Creation of train and test data sets using optimum parameters

```
In [179]: #splitting dataset into x and y only numeric daat
          dfNumCols = df[['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term','Credit_History','Loan_Status']]
          y = df['Loan_Status']
          x = dfNumCols.drop(columns=['Loan_Status'])
```

```
In [180]: x.head()
```

Out[180]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	5849	0.0	120.0	360.0	1.0
1	4583	1508.0	128.0	360.0	1.0
2	3000	0.0	66.0	360.0	1.0
3	2583	2358.0	120.0	360.0	1.0
4	6000	0.0	141.0	360.0	1.0

```
In [181]: y.head()
Out[181]: 0    1
          1    0
          2    1
```

Data standardization and splitting dataset into train and test dataset-

```
In [182]: from sklearn.preprocessing import StandardScaler
```

```
In [183]: std=StandardScaler()
```

```
In [187]: X=std.fit_transform(x)
```

```
In [188]: X
Out[188]: array([[ 0.39926641, -0.86103617, -0.26770712,  0.23085296,  0.41851254],
                 [-0.02802029,  0.05284031, -0.12532481,  0.23085296,  0.41851254],
                 [-0.56229742, -0.86103617, -1.22878771,  0.23085296,  0.41851254],
                 ...,
                 [ 1.14954946, -0.71559164,  2.09939878,  0.23085296,  0.41851254],
                 [ 0.98450744, -0.86103617,  0.92474472,  0.23085296,  0.41851254],
                 [-0.02802029, -0.86103617, -0.03633587,  0.23085296, -2.38941464]])
```

```
In [190]: x=X
```

```
In [191]: x.shape
Out[191]: (577, 5)
```

```
In [192]: # Split data into train and test. Model will be built on training data and tested on test data
          from sklearn.model_selection import train_test_split  #importing requried lib.

          x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=340)
```

Hardware and Software Requirements and Tools Used

Libraries used while building model are –

1. pandas and numpy – pandas is mainly used for data analysis. Pandas allows importing data from various file formats such as comma-separated values etc. Pandas allows various data manipulation operations such as data cleaning, data wrangling, selecting etc.
   numpy provides a multidimensional array object. It can be used for various math operations.

2. Matplotlib.pyplot and seaborn – These are visualization techniques. It helps to plot various graph based on datatypes like scatter plot, Bar graph, distplot etc which are used in this model.
3. Warnings – used to avoid any unnecessary popup while running model.
4. LabelEncoder – It helps to convert string/object dataset into integer dataset.
5. Zscore – Helps to remove skewness present in dataset.
6. Classification_report, accuracy_report, confusion_matrix – classification_report is used to measure the quality of prediction from classification algorithm, accuracy_report gives Number of correct predictions to Total number of predictions, confusion_matrix is a tabular summary of the number of correct and incorrect predictions made by a classifier.

7. LogisticRegression/KNeighborsClassifier/RandomForestClassifier/Decision TreeClassifier – It helps for model instantiating and training.
8. Cross_val_score – It gives cross validation score.
9. GridSearchCV – It helps to give correct accuracy score for model after adjusting any overperformance of model.
10. Joblib – It helps to save the model.

Model/s Development and Evaluation

Identification of possible problem-solving approaches (methods)

At this stage, It is important to create a proper machine leaning model in accordance with best practices. It involves steps –

- Data pre-processing – Clean and transform data into an appropriate format
- Conduct features selection in order to choose the most relevant set of variables.
- Selecting appropriate metrics to measures the performance of the model.
- Train several models
- Validate stability of the model
- Analyse result of model

Testing of Identified Approaches (Algorithms)

List of algorithms used in models are –

1. Random State Algorithms
2. LogisticRegression
3. KNeighborsClassifier
4. RandomForestClassifier
5. DecisionTreeClassifier

Now, our data is ready to apply to the model.

Try to different model

1)Random State Algorithm-

```
In [193]: from sklearn.model_selection  import train_test_split
          from sklearn.linear_model import LogisticRegression
          from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,f1_score

In [194]: lg=LogisticRegression()

In [200]: for i in range(1,50):
              x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=340)
              lg.fit(x_train,y_train)
              pred_test=lg.predict(x_test)
              pred_train=lg.predict(x_train)
              if round(accuracy_score(y_test,pred_test)*100,1)==round(accuracy_score(y_train,pred_train)*100,1):
                  print('Random state',i,'score is well')
                  print('test score',accuracy_score(y_test,pred_test)*100)
                  print('train score',accuracy_score(y_train,pred_train)*100)
```

Random state Algorithm – Random state generate are reproducible. Scikit-learn uses random permutations to generate the splits. The random state that you provide is used as a seed to the random number generator. This ensures that the random numbers are generated in the same order.

```
                  print('test score',accuracy_score(y_test,pred_test)*100)
                  print('train score',accuracy_score(y_train,pred_train)*100)

In [201]: x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=340)

In [211]: from sklearn.svm import SVC
          from sklearn.naive_bayes import MultinomialNB
```

```
In [211]: from sklearn.svm import SVC
          from sklearn.naive_bayes import MultinomialNB
          from sklearn.tree import DecisionTreeClassifier
          from sklearn.neighbors import KNeighborsClassifier
          from sklearn.model_selection import GridSearchCV
          from sklearn.ensemble import RandomForestClassifier

In [212]: svc=SVC()
          dtc=DecisionTreeClassifier()
          knn=KNeighborsClassifier()
          nb=MultinomialNB()
          rfc=RandomForestClassifier()

In [213]: def fun(f):
              f.fit(x_train,y_train)
              pred=f.predict(x_test)
              print('Accuracy Score',accuracy_score(y_test,pred))
              print('Confusion Matrix\n',confusion_matrix(y_test,pred))
              print('Classification Report\n',classification_report(y_test,pred))
              print('F1 score',f1_score(y_test,pred,average='micro'))
```

In given data sets I used random state algorithm, by using function method to implement all machine learnig algorithms.

1)LogisticRegression – It is classification Model. It is used to predict the probability of the

classification. It is widely used for binary classification problem. its structure is tree based. Where internal nodes represent the feature od datasets and brances represents the desciosn rules and each leaf node represent the outcome.

```
In [215]: fun(lg)

Accuracy Score 0.8068965517241379
Confusion Matrix
 [[19 26]
 [ 2 98]]
Classification Report
              precision    recall  f1-score   support

           0       0.90      0.42      0.58        45
           1       0.79      0.98      0.87       100

    accuracy                           0.81       145
   macro avg       0.85      0.70      0.73       145
weighted avg       0.83      0.81      0.78       145

F1 score 0.8068965517241379
```

2) Decision Tree Classifier (DTC)- can be used by both classification and regression both.but mostly its used for the classification problem.its structure is tree based .where internal nodes represents the features of the datasets and branches represents the decion rules and each leaf nodes are represents the outcome.

```
In [216]: fun(dtc)

Accuracy Score 0.7655172413793103
Confusion Matrix
 [[24 21]
 [13 87]]
Classification Report
              precision    recall  f1-score   support

           0       0.65      0.53      0.59        45
           1       0.81      0.87      0.84       100

    accuracy                           0.77       145
   macro avg       0.73      0.70      0.71       145
weighted avg       0.76      0.77      0.76       145

F1 score 0.7655172413793103
```

3) KNeighborsClassifier – It is also classification model. It looks for the 5 nearest neighbours

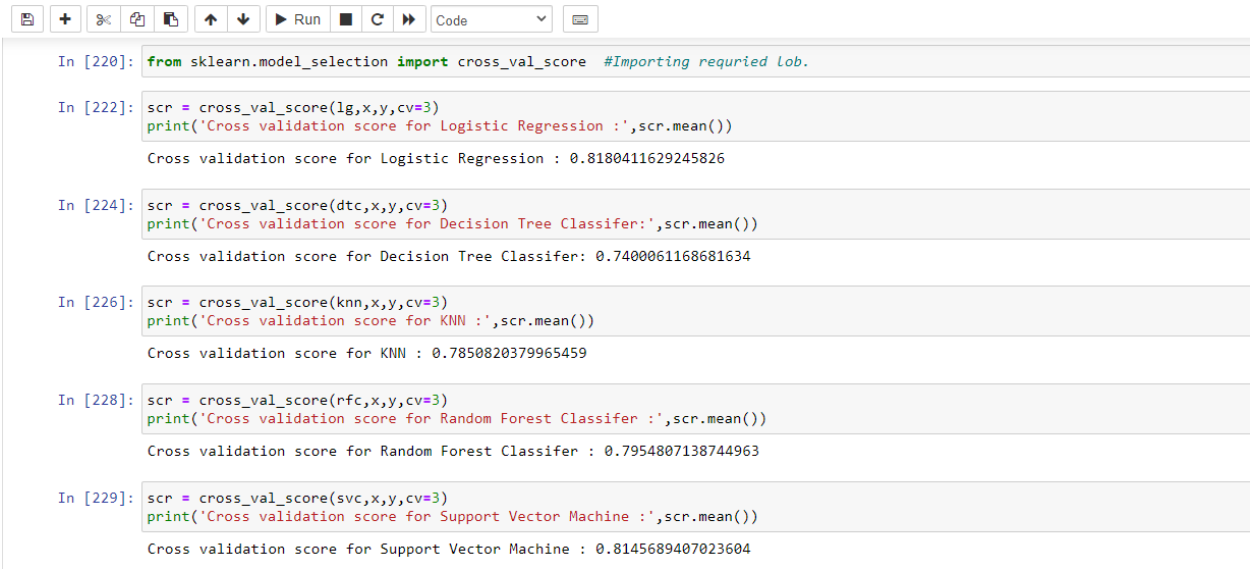
```
In [217]: fun(knn)

Accuracy Score 0.7724137931034483
Confusion Matrix
 [[19 26]
 [ 7 93]]
Classification Report
              precision    recall  f1-score   support

           0       0.73      0.42      0.54        45
           1       0.78      0.93      0.85       100

    accuracy                           0.77       145
   macro avg       0.76      0.68      0.69       145
weighted avg       0.77      0.77      0.75       145

F1 score 0.7724137931034483
```

4) Random Forest Classifier – It is ensemble learning method for classification constructing a multitude of decision trees at training

```
In [218]: fun(rfc)

Accuracy Score 0.7862068965517242
Confusion Matrix
 [[21 24]
 [ 7 93]]
Classification Report
              precision    recall  f1-score   support

           0       0.75      0.47      0.58        45
           1       0.79      0.93      0.86       100

    accuracy                           0.79       145
   macro avg       0.77      0.70      0.72       145
weighted avg       0.78      0.79      0.77       145

F1 score 0.7862068965517242
```

Key Metrics for success in solving problem under consideration

Cross validation score and Hyperparameter tuning used to avoid any overperformance of model**.** After taking cross validation score into consideration, RandomForestClassifier is best model. So, it is used in hyperparameter tuning. After running hyperparameter tunning, we got model score of 80.60%.



```
In [220]: from sklearn.model_selection import cross_val_score  #Importing requried lob.

In [222]: scr = cross_val_score(lg,x,y,cv=3)
          print('Cross validation score for Logistic Regression :',scr.mean())

          Cross validation score for Logistic Regression : 0.8180411629245826

In [224]: scr = cross_val_score(dtc,x,y,cv=3)
          print('Cross validation score for Decision Tree Classifer:',scr.mean())

          Cross validation score for Decision Tree Classifer: 0.7400061168681634

In [226]: scr = cross_val_score(knn,x,y,cv=3)
          print('Cross validation score for KNN :',scr.mean())

          Cross validation score for KNN : 0.7850820379965459

In [228]: scr = cross_val_score(rfc,x,y,cv=3)
          print('Cross validation score for Random Forest Classifer :',scr.mean())

          Cross validation score for Random Forest Classifer : 0.7954807138744963

In [229]: scr = cross_val_score(svc,x,y,cv=3)
          print('Cross validation score for Support Vector Machine :',scr.mean())

          Cross validation score for Support Vector Machine : 0.8145689407023604
```

Hyper Tuning Parameter –

Hyper parameter optimization in machine learning is used to find parameter of given machine learning algorithm that perform best as measured on validation I used GridSerachCV for hyper tuning.

Step No 7. Hyper parameter tuning

```
In [230]: from sklearn.model_selection import GridSearchCV    #Importing requried lib.

In [231]: parameters = {'n_estimators':[200,700],
                        'max_features':['auto','sqrt','log2'],
                        'max_depth':[4,5,6,7,8],
                        'criterion':['gini','entropy']}

In [232]: GCV = GridSearchCV(RandomForestClassifier(),parameters,cv=2)

In [233]: GCV.fit(x_train,y_train)

Out[233]: GridSearchCV(cv=2, estimator=RandomForestClassifier(),
                       param_grid={'criterion': ['gini', 'entropy'],
                                   'max_depth': [4, 5, 6, 7, 8],
                                   'max_features': ['auto', 'sqrt', 'log2'],
                                   'n_estimators': [200, 700]})
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python

▶ Run Code

```
In [233]: GCV.fit(x_train,y_train)

Out[233]: GridSearchCV(cv=2, estimator=RandomForestClassifier(),
                       param_grid={'criterion': ['gini', 'entropy'],
                                   'max_depth': [4, 5, 6, 7, 8],
                                   'max_features': ['auto', 'sqrt', 'log2'],
                                   'n_estimators': [200, 700]})

In [234]: GCV.best_params_

Out[234]: {'criterion': 'entropy',
           'max_depth': 4,
           'max_features': 'sqrt',
           'n_estimators': 200}

In [235]: mod = RandomForestClassifier(criterion='entropy',max_features='auto',max_depth=4, n_estimators=200)

          mod.fit(x_train,y_train)
          pred = mod.predict(x_test)
          print(accuracy_score(y_test,pred)*100)

          80.6896551724138
```

Score improving after hyper tuning = 80.689-0.7862

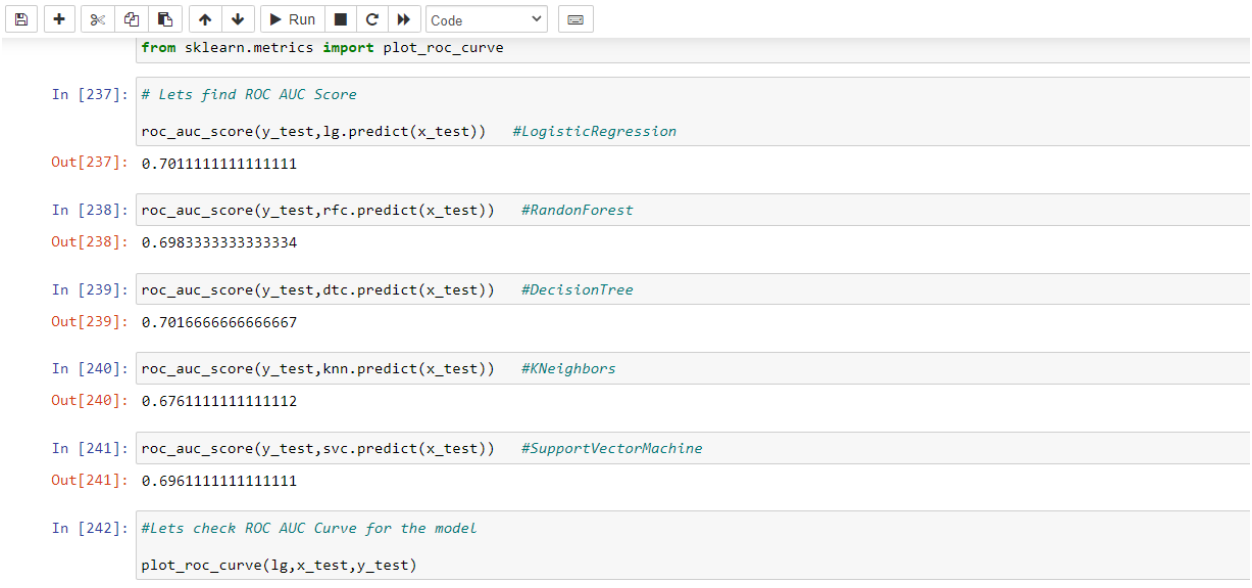
Visualizations

ROC curves typically feature true positive rate on the Y axis and false positive rate on the X axis. This means that the top left corner of the plot is the ideal point – a false positive rate of zero and a true positive rate is one.

The steepness of ROC curve is also important, since it is ideal to maximize the true positive rate while minimize the false positive rate.

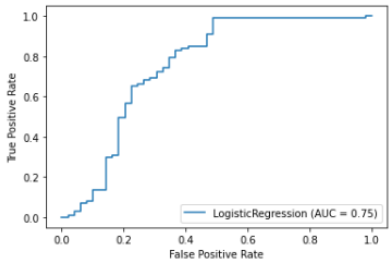
ROC curves are typically used in binary classification to study the output of a classifier. In order to extend ROC curve and ROC area to multi-label classification, it is necessary to binarize the output. One ROC curve can be drawn per label, but one can also draw ROC curve by considering each element of the label indicator matrix as a binary prediction.

ROC curves for this dataset for respective models are –



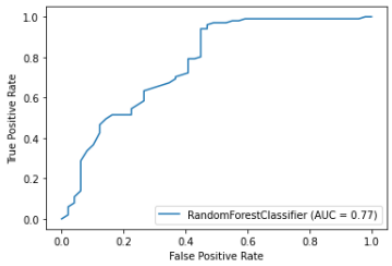
In [283]:
```
#Lets check ROC AUC Curve for the model

plot_roc_curve(LR,x_test,y_test)
```

Out[283]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x23b6a16adc0>
```



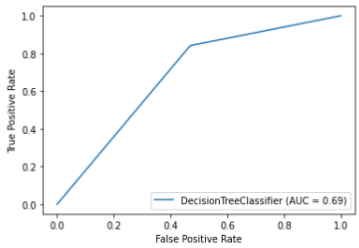
In [284]:
```
plot_roc_curve(RAN,x_test,y_test)
```

Out[284]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x23b68f58b50>
```



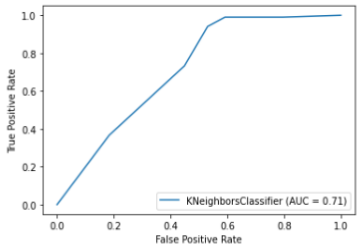
In [285]:
```
plot_roc_curve(dt,x_test,y_test)
```

Out[285]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x23b6c7bfe80>
```



In [286]:
```
plot_roc_curve(KNN,x_test,y_test)
```

Out[286]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x23b6b7bcd00>
```



CONCLUSION

1. Applicants who are male and married tends to have more applicant income whereas applicant who are female and married have least applicant income
2. Applicants who are male and are graduated have more applicant income over the applicants who have not graduated.
3. Again the applicants who are married and graduated have the more applicant income.
4. Applicants who are not self-employed have more applicant income than the applicants who are self-employed.
5. Applicants who have more dependents have least applicant income whereas applicants which have no dependents have maximum applicant income.
6. Applicants who have property in urban and have credit history have maximum applicant income
7. Applicants who are graduate and have credit history have more applicant income.
8. Loan Amount is linearly dependent on Applicant income
9. From heatmaps, applicant income and loan amount are highly positively correlated.
10. Male applicants are more than female applicants.
11. No of applicants who are married are more than no of applicants who are not married.
12. Applicants with no dependents are maximum.
13. Applicants with graduation are more than applicants with no graduation.
14. Property area is to be find more in semi urban areas and minimum in rural areas.