Machine Learning Report on

Loan Application Status Prediction

Introduction

Now a days, ever wondered, how great would it be, if we could predict, whether our request for a loan, will be approved or not, simply by the use of machine learning, from the ease and comfort of your home? Sounds fascinating right? Well, in this article, I will be guiding you through that!

This will not only be a practical project, which is applicable in the current times but also will add more to your knowledge of how the system of Loan Approval works.

You see, any bank, approves a loan based on the two most vital points:

- How risky is the borrower currently, (This is the factor, on which the interest rate of the borrower will depend)?
- Should they lend the money to the borrower at the given risks?

If both of these conditions give an affirmatory result, the bank proceeds with the loan approval.

Algorithms, that we are going to use are:

- Logistic Regression
- GaussianNB
- DecisionTreeClassifier
- RandomForestClassifier
- KNeighborsClassifier

You might think, why I am using different types of algorithms, rather than choosing any one algorithm for the model building.

The reason is simple, just to check which algorithm among them will give the best accuracy to predict the desired output.

Dataset

Step 1

To know the Problem Statement:

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
- ApplicantIncome
- 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.

Note: The link of the dataset is below.

Download Files:

• https://github.com/dsrscientist/DSData/blob/master/loan_prediction.csv

Let’s code!

For this project, I have used Python. You can use any python editing environment that you like, e.g., PyCharm, Jupyter Notebook, Sublime, Atom, VSCode, Google Colab Notebook, etc.

What I have used is a Jupyter Notebook however there is no dearth on availability of such tools.

The benefit of using a Jupyter notebook is that I already had installed it in my system so that I could run the code locally at any point of time. Even I don’t need Internet Connection to run the code.

Importing Libraries

Step 2

Let us first import all the modules and libraries that we are going to use in the future while making the project. The dependencies that we will be using are numpy, pandas, seaborn, and ScikitLearn.

```
import numpy as np
import pandas as pd
from scipy.stats import randint
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.metrics import roc_curve, roc_auc_score, auc
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RandomizedSearchCV
import joblib
```

Data Collection and Processing

Simply downloading the dataset will not do. We need to link the dataset to our code so that our code can read the data from the table. The dataset will be in the format of a CSV. Thus, to read it, we will be taking the help of the pandas method, called read_csv()

We are strong the dataset in the variable called "**df**". We can thus now refer to the entire dataset by this variable name.

Step 3

To get a glance at the first 5 rows of the dataset, use the head() method and shows the following output as shown below.

```
df=pd.read_csv(r"C:\Users\dell\Desktop\Data Trained Projects\Project 5\loan_prediction.csv")
df.head()
```

	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

Step 4

Now, let us check the number of rows and columns in the dataset. We run the command and see the output:

```
1 df.shape
```

```
(614, 13)
```

The shape of the dataset **df** is – 614 rows x 13 columns

Step 5

Next, let us check the cosine summary or description of the dataset **df**

```
1 df.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
```

Observation:

The dataset has 12 features which have 3 different datatypes:

- float64 = 4 features
- int64 = 1 feature
- object = 7 feature
- object = 1 label

Step 6

Next step is to check for null values

Check for null values

```
1 df.isnull().sum()
```

```
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
```

We can see lots of null values present in the dataset **df**

Step 7

Check for duplicate variables

```
df.duplicated().sum().sum()
```

```
0
```

We, can observe that there are no duplicates in the dataset **df.**

Step 8

The next step is to check the statistical summary of the dataset **df**

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
ApplicantIncome	614.0	5403.459283	6109.041673	150.0	2877.5	3812.5	5795.00	81000.0
CoapplicantIncome	614.0	1621.245798	2926.248369	0.0	0.0	1188.5	2297.25	41667.0
LoanAmount	592.0	146.412162	85.587325	9.0	100.0	128.0	168.00	700.0
Loan_Amount_Term	600.0	342.000000	65.120410	12.0	360.0	360.0	360.00	480.0
Credit_History	564.0	0.842199	0.364878	0.0	1.0	1.0	1.00	1.0

As there are 5 numerical features, so its shows five statistical summary of the dataset. Here we can find the mean. standard deviation, minimum and maximum values and the Ist, IInd, IIIrd quantile of the features.

Step 9

Check for uniqueness in feature columns

```
df.nunique()
```

```
Loan_ID              614
Gender                 2
Married                2
Dependents             4
Education              2
Self_Employed          2
ApplicantIncome      505
CoapplicantIncome    287
LoanAmount           203
Loan_Amount_Term      10
Credit_History         2
Property_Area          3
Loan_Status            2
dtype: int64
```

We found out that there are some features having uniqueness in them.

Step 10

First, lets change some of the class name for ease handling

```
df['Education'] = df['Education'].replace('Not Graduate', 'Not_Graduate')
df.Education.unique()
df['Dependents'] = df['Dependents'].replace('3+', 'More_then_2')
df['Dependents'] = df['Dependents'].replace('2', 'Two')
df['Dependents'] = df['Dependents'].replace('1', 'One')
df['Dependents'] = df['Dependents'].replace('0', 'Zero')
df.Dependents.unique()
```

After changing name, now let's treat the missing values and encode the features after observing the dataset **df**.

```
def data_pipeline(df):
    df["Gender"].fillna(df["Gender"].mode()[0], inplace=True)
    df.Gender.replace(to_replace=dict(Female=1, Male=0), inplace=True)
    df["Married"].fillna(df["Married"].mode()[0], inplace=True)
    df.Married.replace(to_replace=dict(Yes=1, No=0), inplace=True)
    df["Dependents"].fillna(df["Dependents"].mode()[0], inplace=True)
    df.Dependents.replace(to_replace=dict(Zero=0, One=1, Two=2, More_then_2=3), inplace=True)
    df.Education.replace(to_replace=dict(Not_Graduate=0, Graduate=1), inplace=True)
    df["Self_Employed"].fillna(df["Self_Employed"].mode()[0], inplace=True)
    df.Self_Employed.replace(to_replace=dict(Yes=1, No=0), inplace=True)
    df["LoanAmount"].fillna(df["LoanAmount"].mean(), inplace=True)
    df["Loan_Amount_Term"].fillna(df["Loan_Amount_Term"].mode()[0], inplace=True)
    df["Credit_History"].fillna(df["Credit_History"].mode()[0], inplace=True)
    df.Property_Area.replace(to_replace=dict(Urban=0, Rural=1, Semiurban=2), inplace=True)
    df.Loan_Status.replace(to_replace=dict(Y=1, N=0), inplace=True)
    df=df.drop(["Loan_ID"], axis=1)
    return df
```

```
df = data_pipeline(df)
```

We have treated –

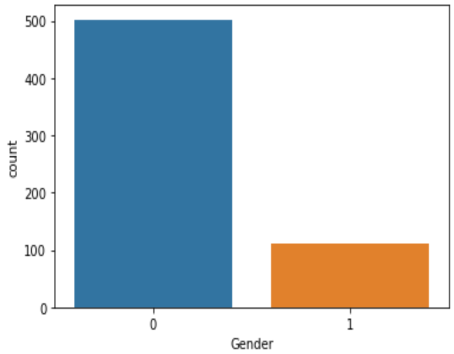
- Filled the null values of 'Gender' by mode. Encoded Female = 1, Male = 0
- Filled the null values of 'Married' by mode. Encoded Yes = 1, No = 0
- Filled the null values of 'Dependents' by mode.
- In Feature 'Education', encoded Not_Graduate = 0, Graduate = 1
- Filled the null values of 'Self_Employed' by mode. Encoded Yes = 1, No = 0
- Filled the null values of 'Loan_Amount' by mean.
- Filled the null values of 'Loan_Amount_Term' by mode.
- Filled the null values of 'Credit_History' by mode.
- In Feature 'Property_Area', encoded Urban=0, Rural=1, Semiurban=2
- In feature 'Loan_Status', encoded Y=1, N=0
- Dropped the column 'Loan_ID', which have no such correlation with 'Loan_Status'

Data Interpretation and Visualization

Step 11

Plotting a count plot for 'Gender' feature in the dataset.

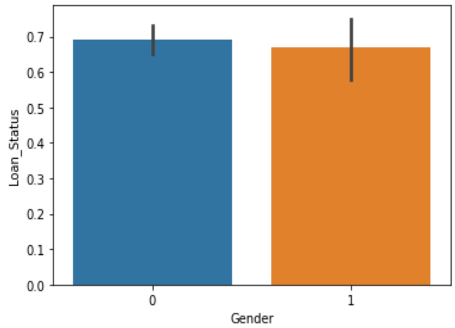
```
sns.countplot(x='Gender', data=df)
plt.show()
```



The above figure shows the number of males and females

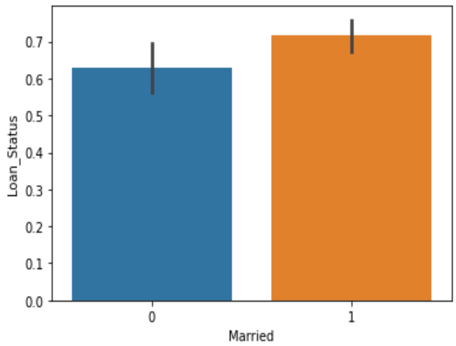
Female = 1, Male = 0

```
sns.barplot(x='Gender', y='Loan_Status', data=df)
plt.show()
```



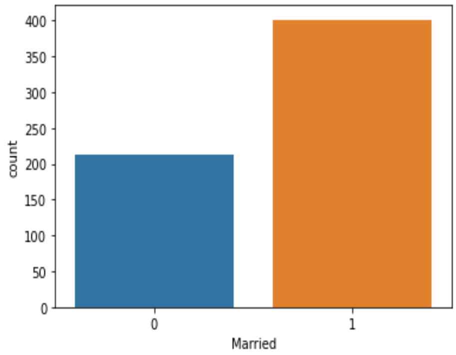
The above graph shows the relation between feature 'Gender' with the label 'Loan_Status'.

```
sns.barplot(x='Married', y='Loan_Status', data=df)
plt.show()
```



The above graph shows the relations between 'Married' feature and the label 'Loan_Status'.

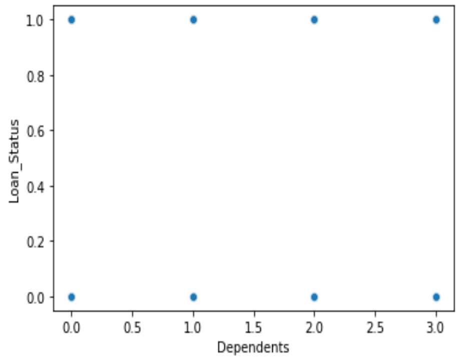
```
sns.countplot(x='Married', data=df)
plt.show()
```



The above shows the number of Married peoples in feature 'Married'.

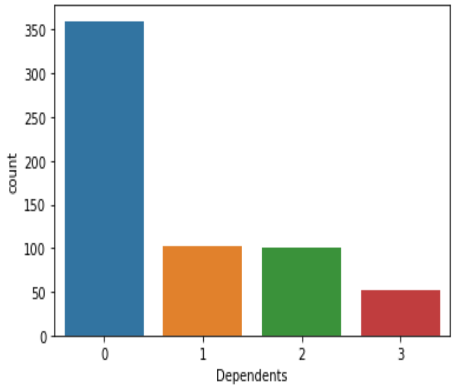
Here, married = 1, unmarried = 0

```
sns.scatterplot(x='Dependents', y='Loan_Status', data=df)
plt.show()
```



The above graph shows the direction of relationship between the variables.

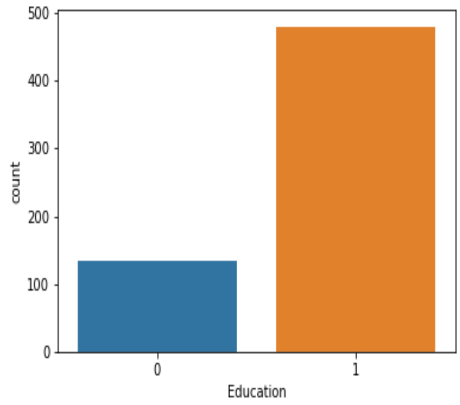
```
sns.countplot(x='Dependents', data=df)
plt.show()
```



The above shows the count of each class of the Dependent feature.

Here, **0** = Zero, **1** = One, **2** = Two and **3** = 3 or more than 3

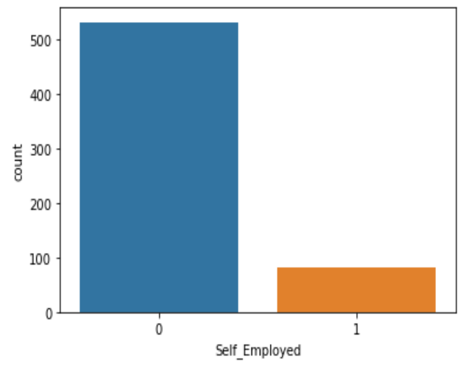
```
sns.countplot(x='Education', data=df)
plt.show()
```



The above shows the number of educated person and uneducated person.

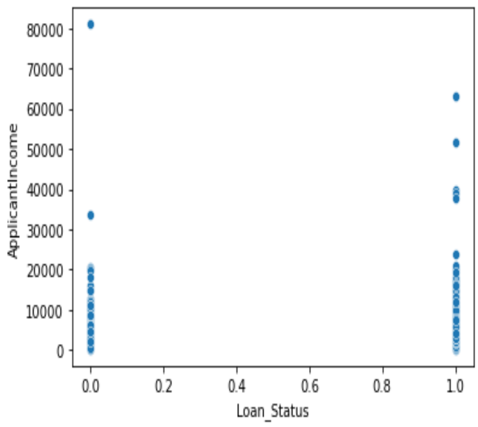
Here, 0 = Not Graduate and 1 = Graduate

```
sns.countplot(x='Self_Employed', data=df)
plt.show()
```



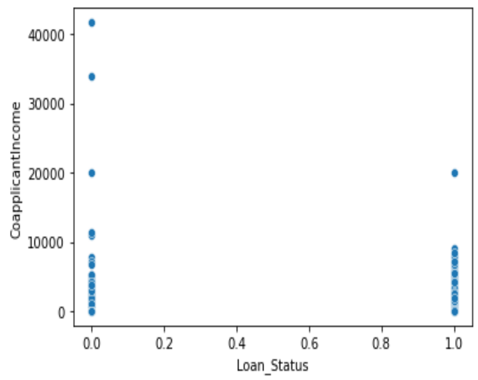
This shows the count of self-employed = 1 and not self-employed = 0

```
sns.scatterplot(y='ApplicantIncome', x='Loan_Status', data=df)
plt.show()
```



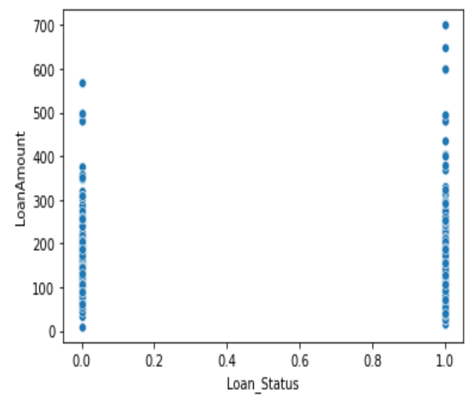
This above plot shows people's income and whether the get loan or not, and we can also observe that their income varies a lot in both the cases.

```
sns.scatterplot(y='CoapplicantIncome', x='Loan_Status', data=df)
plt.show()
```



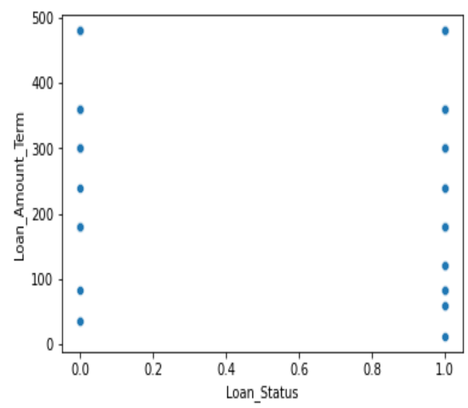
This above plot shows coapplicant's income and whether it helps the applicant to get loan or not.

```
sns.scatterplot(y='LoanAmount', x='Loan_Status', data=df)
plt.show()
```



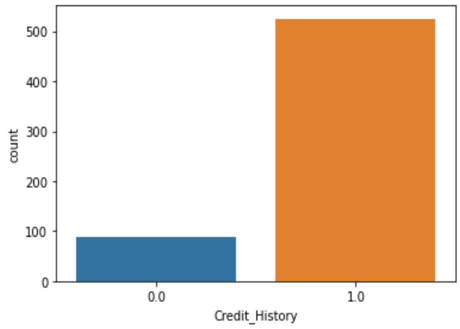
The above shows the relation between LoanAmount and Loan_status and also the count of positive or negative Loan_Status.

```
sns.scatterplot(y='Loan_Amount_Term', x='Loan_Status', data=df)
plt.show()
```



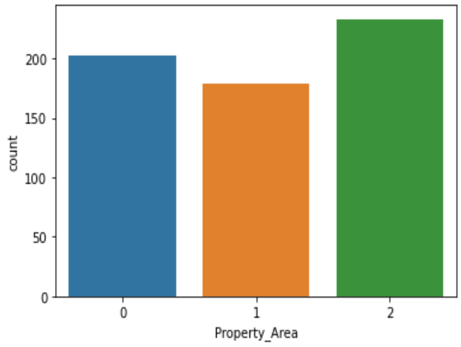
The above shows the relation between Loan_Amount_Term and Loan_status.

```
1 sns.countplot(x='Credit_History', data=df)
2 plt.show()
```



The above graph shows the number of Credit_History or not of the applicants.

```
1 sns.countplot(x='Property_Area', data=df)
2 plt.show()
```



This is the count plot between the applicants living in Urban=0, Rural=1, Semiurban=2 area.

In the above section, I have tried to plot different possible graphs with each other and see their relations with each other.

Step 12

Finding Correlation

Let's check the correlation of the features with other features and the Label - 'Loan_status'.

```
1 df.corr()
```

	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	-0.107930	0.074030	-
Married	-0.364569	1.000000	0.334216	-0.012304	0.004489	0.051708	0.075948	0.147141	-0.100912	
Dependents	-0.172914	0.334216	1.000000	-0.055752	0.056798	0.118202	0.030430	0.163106	-0.103864	-
Education	0.045364	-0.012304	-0.055752	1.000000	0.010383	0.140760	0.062290	0.166998	0.073928	
Self_Employed	0.000525	0.004489	0.056798	0.010383	1.000000	0.127180	-0.016100	0.115260	-0.033739	-
ApplicantIncome	-0.058809	0.051708	0.118202	0.140760	0.127180	1.000000	-0.116605	0.565620	-0.046531	-
CoapplicantIncome	-0.082912	0.075948	0.030430	0.062290	-0.016100	-0.116605	1.000000	0.187828	-0.059383	
LoanAmount	-0.107930	0.147141	0.163106	0.166998	0.115260	0.565620	0.187828	1.000000	0.036475	-
Loan_Amount_Term	0.074030	-0.100912	-0.103864	0.073928	-0.033739	-0.046531	-0.059383	0.036475	1.000000	-
Credit_History	-0.009170	0.010938	-0.040160	0.073658	-0.001550	-0.018615	0.011134	-0.001431	-0.004705	
Property_Area	0.082045	0.003071	0.001781	0.003592	0.021996	-0.007894	-0.028356	0.013799	0.086879	
Loan_Status	-0.017987	0.091478	0.010118	0.085884	-0.003700	-0.004710	-0.059187	-0.036416	-0.022549	

```
df.corr()['Loan_Status'].sort_values()
```

```
CoapplicantIncome   -0.059187
LoanAmount          -0.036416
Loan_Amount_Term    -0.022549
Gender              -0.017987
ApplicantIncome     -0.004710
Self_Employed       -0.003700
Dependents           0.010118
Education            0.085884
Married              0.091478
Property_Area        0.103253
Credit_History       0.540556
Loan_Status          1.000000
Name: Loan_Status, dtype: float64
```

Step 13

Drop features which have zero or very less correlation with each other.

Since Gender, ApplicantIncome and Self_Employed have minor or very less correlation with loan income, lets drop it.

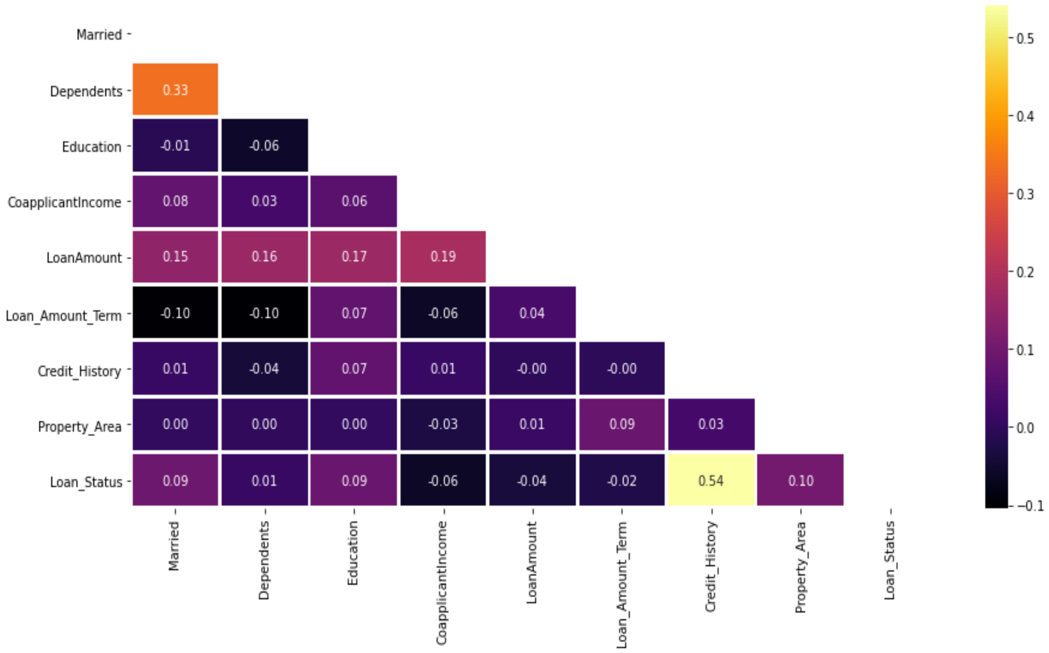
```
df=df.drop(["Gender","ApplicantIncome","Self_Employed"], axis=1)
df.head()
```

	Married	Dependents	Education	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	0	0	1	0.0	146.412162	360.0	1.0	0	1
1	1	1	1	1508.0	128.000000	360.0	1.0	1	0
2	1	0	1	0.0	66.000000	360.0	1.0	0	1
3	1	0	0	2358.0	120.000000	360.0	1.0	0	1
4	0	0	1	0.0	141.000000	360.0	1.0	0	1

Step 14

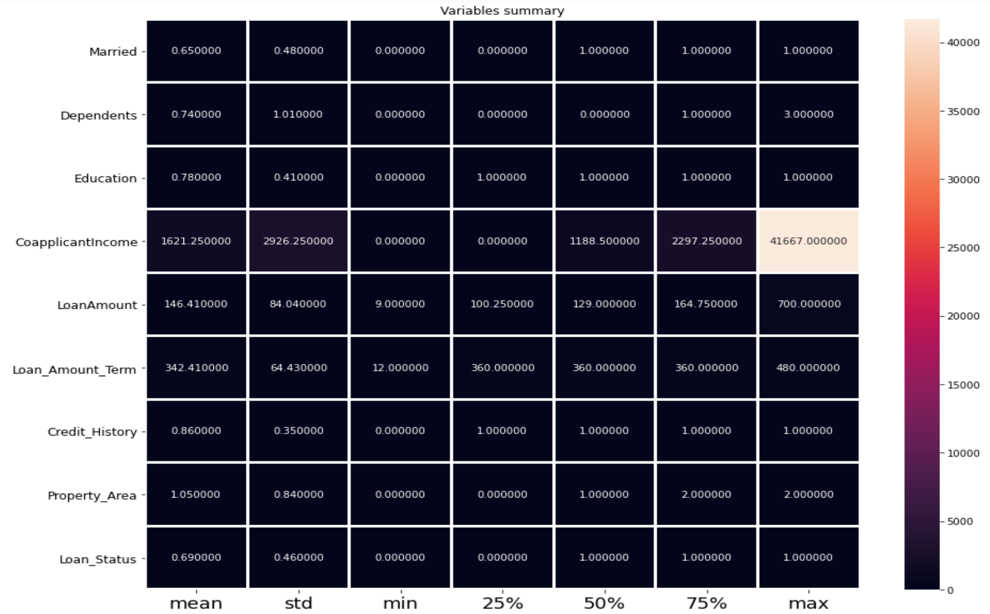
Visualizing Correlation matrix using heatmap

```
plt.figure(figsize = (15,7))
matrix = np.triu(df.corr())
sns.heatmap(df.corr(), annot = True, fmt = '.2f',linewidths=2, mask=matrix, cmap='inferno')
plt.show()
```



Step 15

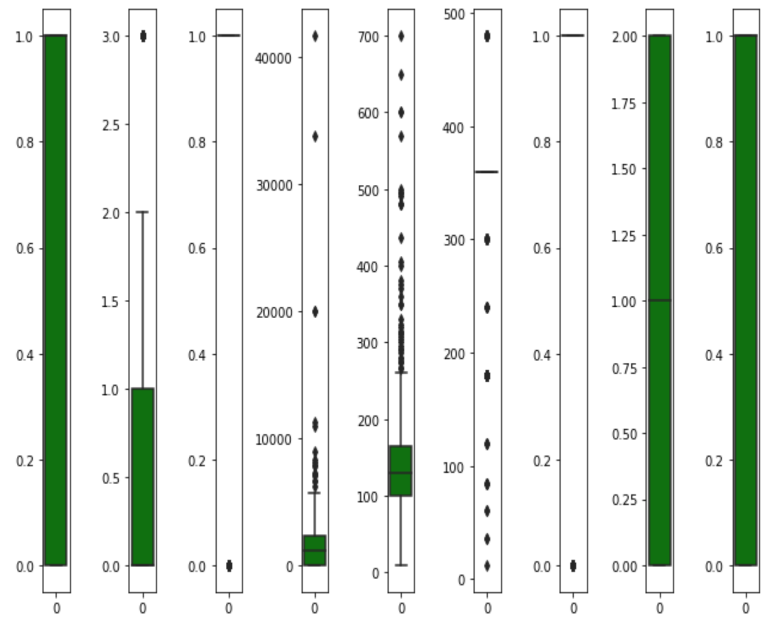
Visualizing Variables summary using Heat map



Step 16

Finding Outliers

```
collist = df.columns.values
ncol = 30
nrows = 14
plt.figure(figsize=(ncol,3*ncol))
for i in range(0, len(collist)):
    plt.subplot(nrows, ncol, i+1)
    sns.boxplot(data = df[collist[i]], color='green', orient='v')
    plt.tight_layout()
```



The above plot shows the outliers present in some features in the dataset **df.**

Step 17

Checking Skewness in the features

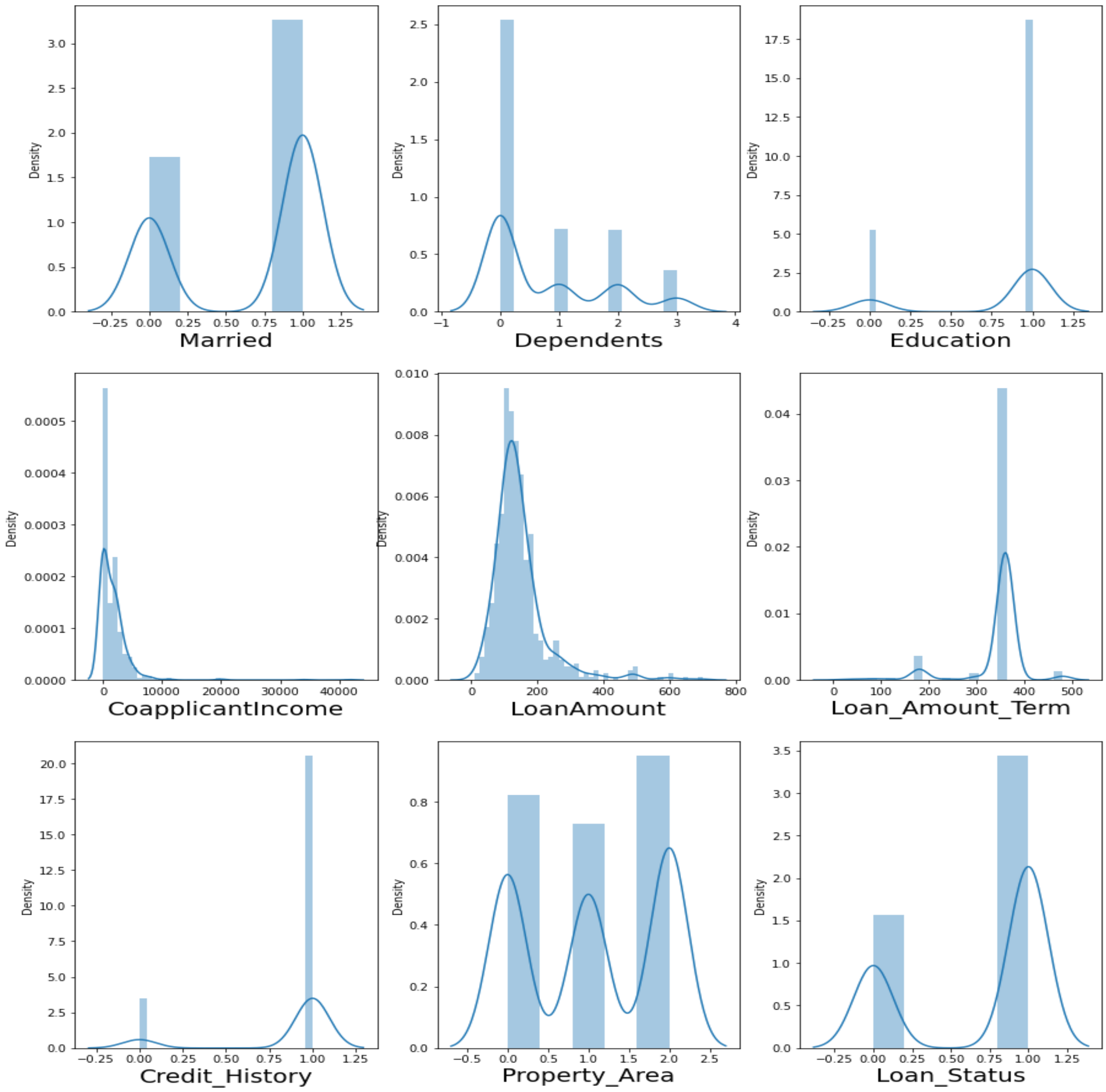
```
df.skew()
```

```
Married             -0.644850
Dependents           1.015551
Education           -1.367622
CoapplicantIncome    7.491531
LoanAmount           2.726601
Loan_Amount_Term    -2.402112
Credit_History      -2.021971
Property_Area       -0.095601
Loan_Status         -0.809998
dtype: float64
```

We found that the dataframe **df**, have skewness in almost all the features.

Step 18

Let's visualize how the data is distributed for every column



Feature Engineering

Step 19

Removing Outliers using Z-Score Technique

The Z-score is the signed number of standard deviations by which the value of an observation or data point is above the mean value of what is being observed or measured.

```
z= np.abs(zscore(df))
z.shape
```

```
(614, 9)
```

As, it is difficult to say which data point is an outlier. Let's try and define a threshold to identify an outlier.

```
threshold=3
# index number
print(np.where(z>3))
```

```
(array([  9,  14,  68,  94, 130, 133, 155, 171, 177, 177, 242, 262, 278,
       308, 313, 333, 369, 402, 417, 432, 487, 495, 497, 506, 523, 525,
       546, 561, 575, 581, 585, 600, 604], dtype=int64), array([3, 5, 5, 5, 4, 5, 4, 4, 3, 4, 5, 5, 4, 4, 5, 4, 4, 3, 3, 4, 4,
5,
       5, 4, 4, 4, 5, 4, 5, 3, 5, 3, 4], dtype=int64))
```

Don't be confused by the results. The first array contains the list of row numbers and second array respective column numbers, which mean z[9][1] have a Z-score higher than 3.

Since, we found the outliers, now it's time to remove the outliers by the below simple codes.

```
len(np.where(z>3)[0])
```

```
33
```

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

```
(614, 9)
(582, 9)
```

Let's, check the total data loss percentage

```
loss_percent=(614-582)/614*100
print(loss_percent, '%')
```

```
5.211726384364821 %
```

Step 20

Dividing data in features and vectors

```
# independent column/features
x=df_new.iloc[:,:-1]
# target
y=df_new.iloc[:,-1]
```

Above step, I have separated the features with the label into x and y.

Step 21

Transforming data to remove skewness using power_transform

```
x=power_transform(x,method='yeo-johnson')
x
```

```
array([[-1.36638028e+00, -8.14462909e-01,  5.38948636e-01, ...,
         1.29897450e-01,  4.19234512e-01, -1.28609606e+00],
       [ 7.31860679e-01,  8.88791720e-01,  5.38948636e-01, ...,
         1.29897450e-01,  4.19234512e-01, -2.14622347e-03],
       [ 7.31860679e-01, -8.14462909e-01,  5.38948636e-01, ...,
         1.29897450e-01,  4.19234512e-01, -1.28609606e+00],
       ...,
       [ 7.31860679e-01,  8.88791720e-01,  5.38948636e-01, ...,
         1.29897450e-01,  4.19234512e-01, -1.28609606e+00],
       [ 7.31860679e-01,  1.33659450e+00,  5.38948636e-01, ...,
         1.29897450e-01,  4.19234512e-01, -1.28609606e+00],
       [-1.36638028e+00, -8.14462909e-01,  5.38948636e-01, ...,
         1.29897450e-01, -2.38529981e+00,  1.10656114e+00]])
```

Step 22

Standardizing the data using StandardScaler

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=sc.fit_transform(x)
x
```

```
array([[-1.36638028e+00, -8.14462909e-01,  5.38948636e-01, ...,
         1.29897450e-01,  4.19234512e-01, -1.28609606e+00],
       [ 7.31860679e-01,  8.88791720e-01,  5.38948636e-01, ...,
         1.29897450e-01,  4.19234512e-01, -2.14622347e-03],
       [ 7.31860679e-01, -8.14462909e-01,  5.38948636e-01, ...,
         1.29897450e-01,  4.19234512e-01, -1.28609606e+00],
       ...,
       [ 7.31860679e-01,  8.88791720e-01,  5.38948636e-01, ...,
         1.29897450e-01,  4.19234512e-01, -1.28609606e+00],
       [ 7.31860679e-01,  1.33659450e+00,  5.38948636e-01, ...,
         1.29897450e-01,  4.19234512e-01, -1.28609606e+00],
       [-1.36638028e+00, -8.14462909e-01,  5.38948636e-01, ...,
         1.29897450e-01, -2.38529981e+00,  1.10656114e+00]])
```

I have used both power transform and standard scaler to remove skewness from the dataset and standardizing the data.

Since the outcome "Loan_Status" has only two variable we will use binary classification model.

Pre-processing and Classification

Step 23

Split the "**Loan_Status"** column from the other columns.

```
x = df.drop(columns=["Loan_Status"])
y = df["Loan_Status"]

# Show distribution of 0 and 1
y.value_counts()
```

```
1    422
0    192
Name: Loan_Status, dtype: int64
```

I found that the label is not equally distributed equally, so I use "**SMOTE"** to make the "Loan_Status" column distribution equal as shown below.

Step 24

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
x, y = sm.fit_resample(x, y)
y.value_counts()
```

```
0    422
1    422
Name: Loan_Status, dtype: int64
```

Equally distributed the label i.e., 0 = 422 and 1 = 422.

Split train and test Dataset

Step 25

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=14)
```

In this step, I have split the dataset into X_train, X_test, y_train and y_test with test_size = 0.2 and random_state = 14.

Let's understand each of the variables by knowing what type of values they will be storing :

X_train: contains a random set of values from variable ' X '

y_train: contains the output (the Loan Status) of the corresponding value of X_train.

X_test: contains a random set of values from variable ' X ', excluding the ones already present in X_train(as they are already taken).

Y_test: contains the output (the Loan Status) of the corresponding value of X_test.

test_size: represents the ratio of how the data is distributed among X_train and X_test (Here 0.2 means that the data will be segregated in the X_train and X_test variables in an 80:20 ratio). You can use any value you want. A value < 0.3 is preferred.

random_state: Controls the shuffling applied to the data before applying the split.

Model Building

Step 26

```
# Create empty list and append each model to list
models = []
models.append(("LOGR", LogisticRegression()))
models.append(("NAIVE BAYES", GaussianNB()))
models.append(("DT", DecisionTreeClassifier()))
models.append(("RF", RandomForestClassifier()))
models.append(("KNN", KNeighborsClassifier()))

# Empty list for results of the evaluation
model_results = []
```

Creating an empty list and append it with different types of algorithms.

Step 27

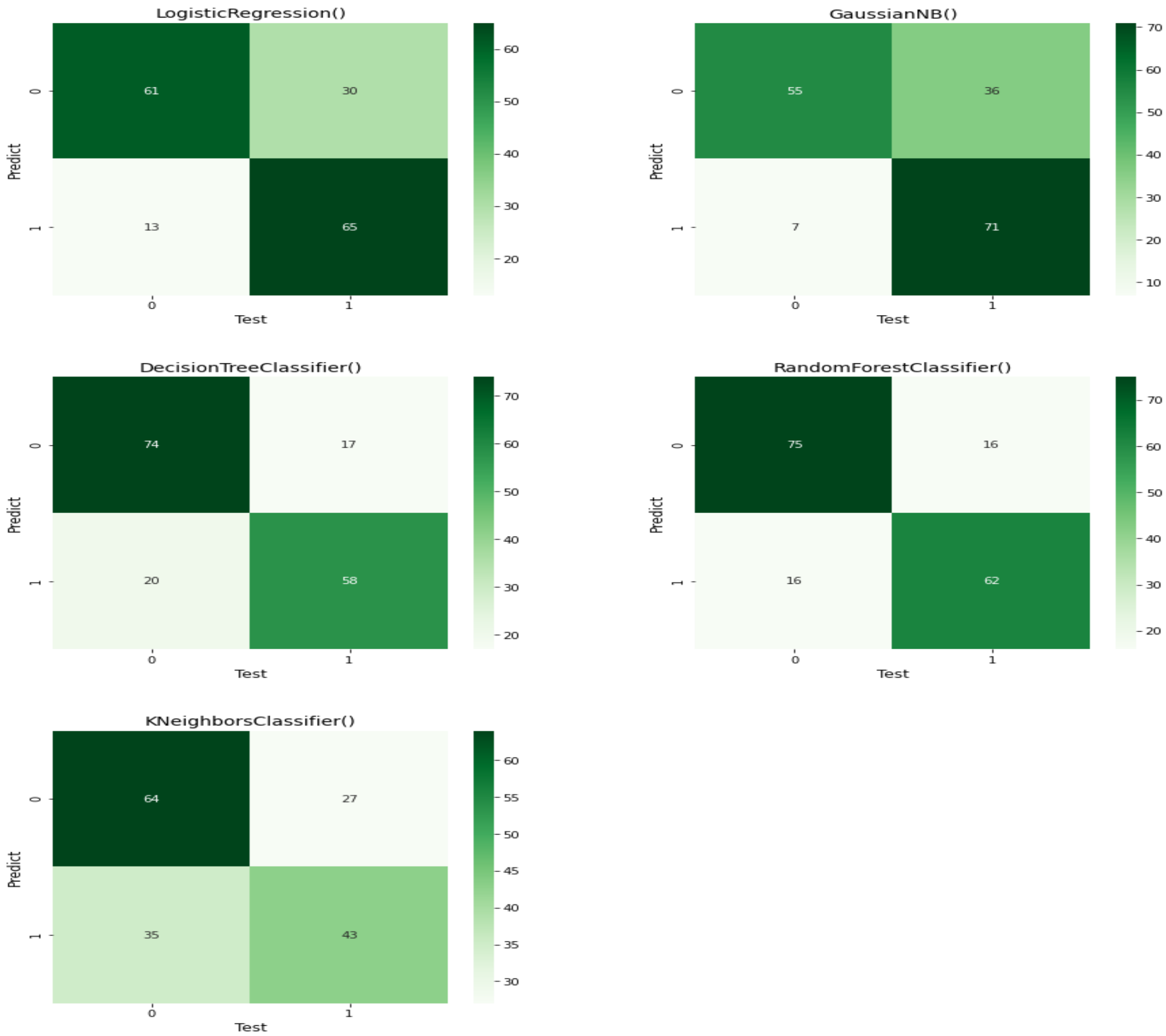
Defining a function, which will be called for fitting the model, finding f_score & confusion_matrix

```python
# Function: for each element in model list there will be an evaluation -> Results will be added to results df
def train_all_models(models):
    i = 1
    plt.figure(figsize=(15, 20))
    for method, model in models:
        model.fit(X_train, y_train)
        test_pred = model.predict(X_test)

        f_score = model.score(X_test, y_test)
        model_results.append((method, f_score))

        plt.subplot(3, 2, i)
        plt.subplots_adjust(hspace=0.3, wspace=0.3)
        sns.heatmap(confusion_matrix(y_test, test_pred), annot=True, cmap="Greens")
        plt.title(model, fontsize=14)
        plt.xlabel('Test', fontsize=12)
        plt.ylabel('Predict', fontsize=12)
        df = pd.DataFrame(model_results).transpose()
        i+=1

# Show confusion matrix for each trained model
    plt.show()
    df = pd.DataFrame(model_results)
    return df
```

```python
# Sort results df for later visualizations
best_models = train_all_models(models)
best_models = best_models.sort_values([1], ascending=False)
```

Step 28

Visualizing the confusing matrix



Step 29

Making a Dataframe containing scores of models and visualizing it using graph

```
best_models
```

	0	1
3	RF	0.810651
2	DT	0.781065
0	LOGR	0.745562
1	NAIVE BAYES	0.745562
4	KNN	0.633136

```
y_pos = np.arange(len(best_models[0]))
plt.figure(figsize=(10, 6))
plt.bar(y_pos, best_models[1], color=(0.2, 0.4, 0.6, 0.6))
plt.xticks(y_pos, best_models[0])
plt.title('F-Score of all trained models')
plt.xlabel('Model Type')
plt.ylabel('F-Score')
plt.show()
```

I found that Random Forest is showing the maxing f-score of the dataset while predestining. Where as KNN classifier shows the least f-score.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

The sub-sample size is controlled with the max_samples parameter if bootstrap=True, otherwise the whole dataset is used to build each tree.

This algorithm is widely used in E-commerce, banking, medicine, the stock market, etc.

WHAT IS RANDOM FOREST?

Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

Hyper Parameter Tuning of the model with maximum score

Step 30

Since, RandomForestClassifier gave the maximum score, so let's hyper parameter tune it with best parameters

```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

```
RandomForestClassifier()
```

```
from sklearn.model_selection import RandomizedSearchCV

# Grid Search for RandomForestClassifier
grid_param_RF = {
    "n_estimators": randint(low=1, high=100),
    "max_depth": randint(low=10, high=100),
    "max_features": randint(low=1, high=4)
}

RF_grid_search = RandomizedSearchCV(estimator=rf, param_distributions=grid_param_RF, cv= 10, verbose=1, random_state=14)
RF_grid_search.fit(X_train, y_train)

RF_best_grid = RF_grid_search.best_estimator_
print(RF_best_grid)
print(RF_grid_search.best_score_)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
RandomForestClassifier(max_depth=36, max_features=2, n_estimators=88)
0.8074846356453029
```

After Hyper-Parameter Tuning of RandomForestClassifier, got best parameter with max_depth=36, max_features=2 and n_estimators=88.

Cross Validation for Random Forest Classifier

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model

Step 31

```
rfc=accuracy_score(y_test,pred)
for j in range(2,10):
    rfscore=cross_val_score(rf,x,y,cv=j)
    rnfc=rfscore.mean()
    print('At cv:-', j)
    print('Cross validation score is:-',rnfc*100)
    print('Accuracy_score is:-',rfc*100)
    print('\n')
```

Output of Cross Validatins – Next Page

```
At cv:- 2
Cross validation score is:- 79.38388625592417
Accuracy_score is:- 81.06508875739645


At cv:- 3
Cross validation score is:- 79.27109361197346
Accuracy_score is:- 81.06508875739645


At cv:- 4
Cross validation score is:- 79.38388625592417
Accuracy_score is:- 81.06508875739645


At cv:- 5
Cross validation score is:- 80.6931530008453
Accuracy_score is:- 81.06508875739645


At cv:- 6
Cross validation score is:- 81.05369807497468
Accuracy_score is:- 81.06508875739645


At cv:- 7
Cross validation score is:- 80.23612750885476
Accuracy_score is:- 81.06508875739645


At cv:- 8
Cross validation score is:- 80.95350404312669
Accuracy_score is:- 81.06508875739645


At cv:- 9
Cross validation score is:- 80.82055974986653
Accuracy_score is:- 81.06508875739645
```

At cv = 6, it is giving the best accuracy score.

cv = 81.053, accuracy = 81.065

ROC AUC curve for Random Forest

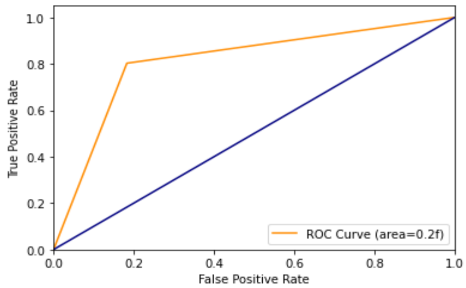
Step 32

```
rf_roc=roc_auc_score(y_test, pred)
rf_roc
```

```
0.8086080586080586
```

Hence, I got the random forest roc score of 80.86%. Hence, let plot the roc curve next.

```
fpr, tpr, thresholds = roc_curve(pred, y_test)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr,tpr,color='darkorange', label='ROC Curve (area=0.2f)'% roc_auc)
plt.plot([0,1],[0,1], color='navy')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```



Saving the model

Step 33

```
import joblib
joblib.dump(rf,'Loan Application Status Prediction.pkl')
```

```
['Loan Application Status Prediction.pkl']
```

Hence, I saved the model with the algorithm which is showing the best accuracy score in .pkl format file so that it will further use ahead for deployment.

Let's predict with our model

Step 34

```
a = np.array(y_test)
predicted=np.array(rf.predict(X_test))
df_con=pd.DataFrame({'original':a,'predicted':predicted}, index=range(len(a)))
df_con.index=df_con.index+1
df_con.head()
```

	original	predicted
1	0	0
2	1	1
3	1	1
4	1	1
5	1	1

Hence, the model is predicting well.

Conclusion

Hence, by following all the steps properly, at the end, we were successfully able to train our classification model 'Random Forest Classifier' to predict the Loan Status of a person with an accuracy score of 81%, and have achieved the required task successfully.