AUTO-INSURANCE FRAUD CLAIM DETECTION MACHINE LEARNING

Author : Kishan Barochiya

: www.linkedin.com/in/kishan-barochiya-138b23115
: https://github.com/kishanbarochiya

PROBLEM DEFINATION:

As we, all know about loan. In today's time, loan is mostly used for Housing, car, Personal, Business etc. Banker who give the loan are not giving loan without any parameter. They have certain process to approve loan. Many times customer want to loan of higher amount but banker survey their background and give conclusion like approve loan or same amount, denied for loan or approve loan with some terms and condition like mutual amount which they like that they can recover from client. Loan approval process have several parameters like Annual income, Dependent, Age, Proposed loan amount, Property of customer, Credit score, History of previous loan in case of any loan taken in past, Education etc.

DATA:

We are using the loan prediction model data that can we download from [here](#).
First, we install all libraries which will require for analytics and modeling process.

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        from collections import Counter
        import warnings
        warnings.filterwarnings('ignore')
        import random
        from sklearn.preprocessing import LabelEncoder
        # machine learning
        from sklearn.linear_model import LogisticRegression
        from sklearn.svm import SVC, LinearSVC
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.naive_bayes import GaussianNB
        from sklearn.linear_model import Perceptron
        from sklearn.linear_model import SGDClassifier
        from sklearn.tree import DecisionTreeClassifier
```

LOAD THE DATASET AND STATATICAL ANALYSIS:

By using below formula, we will load the dataset and we can see the dataset we have, there are many parameters like Loan id, Gender, Married, Income, Loan Amount, Loan_amount_term, Credit history and loan approval status**.**

```
In [2]: df = pd.read_csv('loan.csv')
        df.head(10)
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Pr
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	
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196.0	267.0	360.0	1.0	
6	LP001013	Male	Yes	0	Not Graduate	No	2333	1516.0	95.0	360.0	1.0	
7	LP001014	Male	Yes	3+	Graduate	No	3036	2504.0	158.0	360.0	0.0	
8	LP001018	Male	Yes	2	Graduate	No	4006	1526.0	168.0	360.0	1.0	
9	LP001020	Male	Yes	1	Graduate	No	12841	10968.0	349.0	360.0	1.0	

Here, we know that loan id is not matter in case of loan approved of not, it is just for uniqueness of data so we will remove it and going to check for data shape or null values present in it or not.

```
In [3]: df.shape
Out[3]: (614, 13)
```

```
In [4]: df.drop(['Loan_ID'], inplace = True, axis = 1)
        # AS WE KNOW THAT LOAN ID IS UNIQUE VALUE AND IT IS NOT EFFECTIVE IN LOAN PREDICTION
```

```
In [5]: df.isnull().sum()
        #CHECKING FOR NULL VALUES IN DATA
Out[5]: 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
```

As per above snap, we have 614 rows means data of 614 customers and there are 13 features. We remove 'Loan ID' as it does not affect the loan approval status, then we check for null values and found many null values in Gender, Dependents, Self-employed, loan amount etc.

TECHNIQUES FOR NULL VALUES HANDLING:

1) Remove the rows, which contain null values
2) Replace with Mean/Median/Mode
3) Use imputation technique
4) Replace with unique value
5) Using algorithm

We are going to replace it with Median where data columns are continuous, Ex: Loan Amount and replace with mode where data columns are categorical or non-continuous. Ex: Gender, where only male or female option available. What happened if we use mode for whole column? It leads to make data skewed and unbalanced, we will find the ratio of categorical data and according to it we replace it will null values.

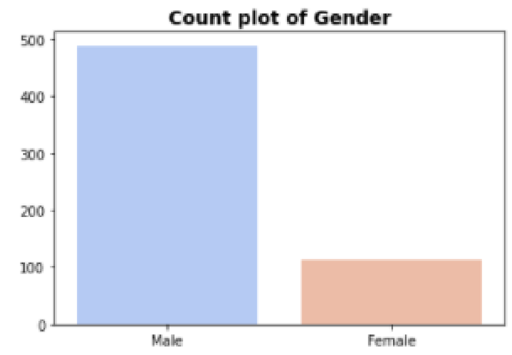
```
In [6]: categorical_df=[j for j in df if df[j].dtype == 'object']

        for i in categorical_df:
            columns = df[i].unique()
            print(i,columns)

        #ALL CATAGORICAL DATA AND UNIQUE VALUE

        Gender ['Male' 'Female' nan]
        Married ['No' 'Yes' nan]
        Dependents ['0' '1' '2' '3+' nan]
        Education ['Graduate' 'Not Graduate']
        Self_Employed ['No' 'Yes' nan]
        Property_Area ['Urban' 'Rural' 'Semiurban']
        Loan_Status ['Y' 'N']
```

```
In [8]: fig = sns.barplot(df['Gender'].value_counts().index,df['Gender'].value_counts().values, palette='coolwarm')
        fig.set_title('Count plot of Gender', fontsize=14, fontweight='bold')
        plt.show()
```



HERE, WE CAN VISULIZE THAT NUMBER OF MALE APPLICANT IS HIGHER THEN FEMALE

```
In [9]: nan = df['Gender'].isna()
        length = sum(nan)
        replacement = random.choices(['Male','Female'], weights=[.7, .3], k=length)
        df.loc[nan,'Gender'] = replacement

        print(df['Gender'].describe())

        count      614
        unique       2
        top       Male
        freq       499
        Name: Gender, dtype: object
```


As we can see in above snap, Male Female ratio is 70:30 so we will randomly replace null values in 70:30 Ratio.
We will do same process for all columns to check ratio and replace according to it. Then we check for null values and got result as per below. Now there is no null values in our dataset.

```
In [34]: df.isnull().sum()
Out[34]: 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
```

Now we are going to use some logic and sorting our data. We will do sum of 'Applicantincome' and 'CoapplicantIncome' and create new column with name 'Total income'.
Convert loan amount term in month and then create new column for EMI by dividing loan amount and loan term.

```
In [37]: df['Total_Income'] = df['ApplicantIncome']+df['CoapplicantIncome']
         df['Loan_Amount_Term'] = df['Loan_Amount_Term']/30
         df['EMI'] = df['LoanAmount']/df['Loan_Amount_Term']

         #create three new column
```

EDA:

Let we check some basics by using below code then we will go with graphical visualization.

In [41]:
```
avg_inc_ = df.groupby(['Loan_Status']).apply(lambda df: round(df['EMI'].mean(), 0))
avg_inc_
```

Out[41]:
```
Loan_Status
N    15.0
Y    14.0
dtype: float64
```

In [42]:
```
    avg_inc_2 = df.groupby(['Property_Area','Loan_Status']).apply(lambda df: round(df['Loan_Status'].count(), 0))
    avg_inc_2
```

Out[42]:
```
Property_Area  Loan_Status
Rural          N               69
               Y              110
Semiurban      N               54
               Y              179
Urban          N               69
               Y              133
dtype: int64
```

In [43]:
```
print('Rural Loan approvel ratio' , (100*110)/169)
print('Semiurban Loan approvel ratio' , (100*179)/233)
print('Urban loan approvel ratio' , (100*133)/202)
```

```
Rural Loan approvel ratio 65.08875739644971
Semiurban Loan approvel ratio 76.82403433476395
Urban loan approvel ratio 65.84158415841584
```


Loan approval ratio is higher in semi urban area compare to rural and urban.

In [44]:
```
sns.catplot(x ='Gender', hue ='Loan_Status',
kind ='count', col ='Property_Area', data = df)
```

Out[44]: <seaborn.axisgrid.FacetGrid at 0x21a5785c2b0>

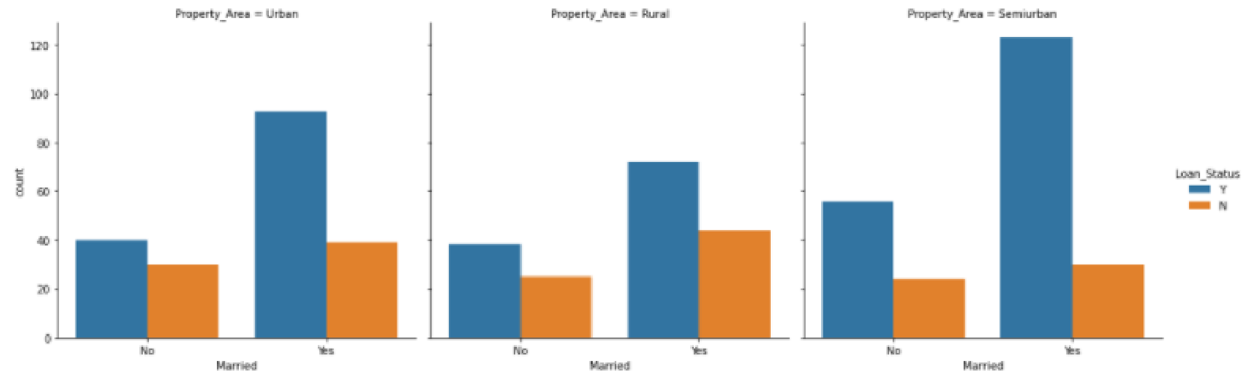
1)Male applicant have higher chances to get loan approved

2)In Urban& Rural if applicant is Female then there is 40 to 50% chances of Rejection

3) Male appilcant from Semiurban area have higher chances to get loan approvel

In [45]:
```
sns.catplot(x ='Married', hue ='Loan_Status',
kind ='count', col ='Property_Area', data = df)
```

Out[45]: <seaborn.axisgrid.FacetGrid at 0x21a57ad7c40>

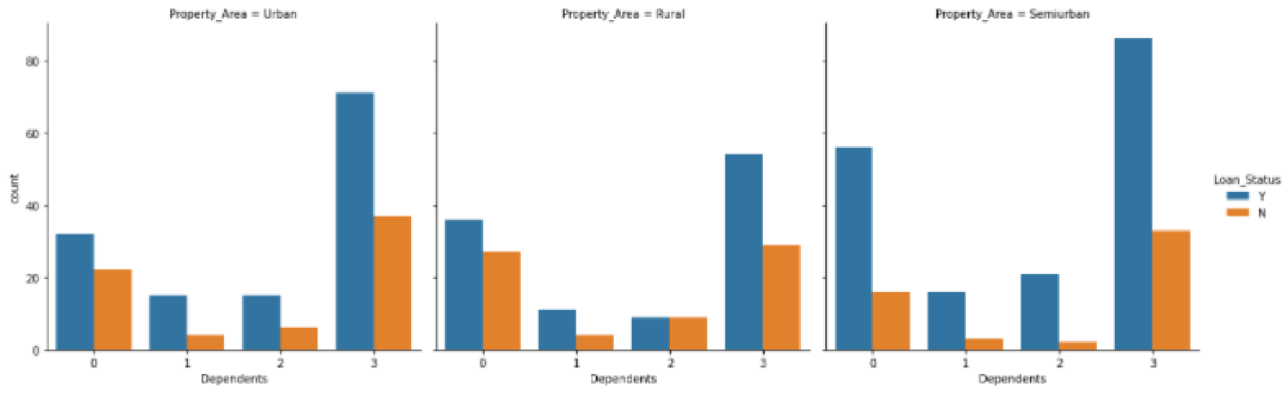


1) Married applicant have higher chances for loan approvel

2) Unmarried applicant from Urban area have higher chances to loan approvel rejection

In [46]:
```
sns.catplot(x ='Dependents', hue ='Loan_Status',
kind ='count', col ='Property_Area', data = df)
```

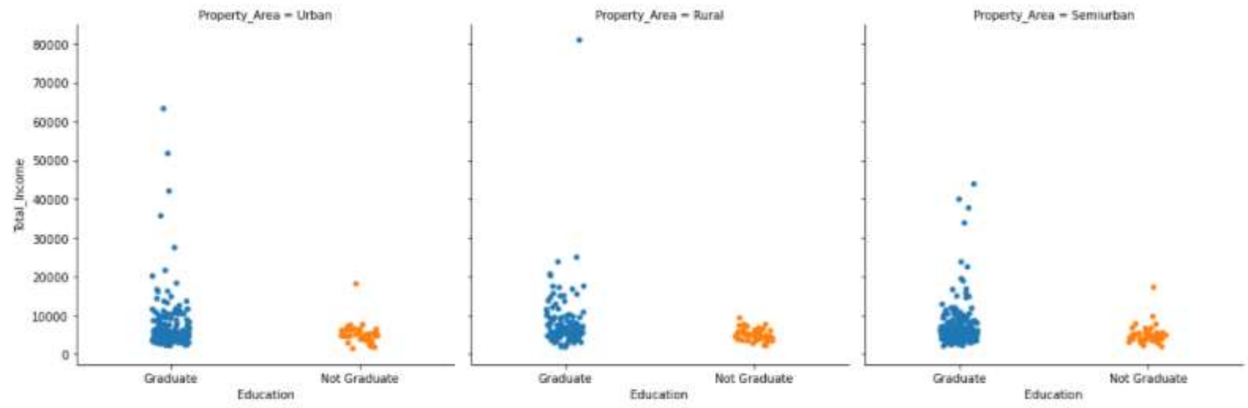
Out[46]: <seaborn.axisgrid.FacetGrid at 0x21a57b2b610>



1) If applicant have 3 or more then 3 dependents then higher chances to loan approvel rejection

2) If applicant have 0 or 1 child then have good chances for loan approvel

```
In [49]: sns.catplot(x ='Education',y ='Total_Income', col ='Property_Area', data = df)
Out[49]: <seaborn.axisgrid.FacetGrid at 0x21a58849f70>
```



In urban:

  Graduate people have high income compare to non-graduate

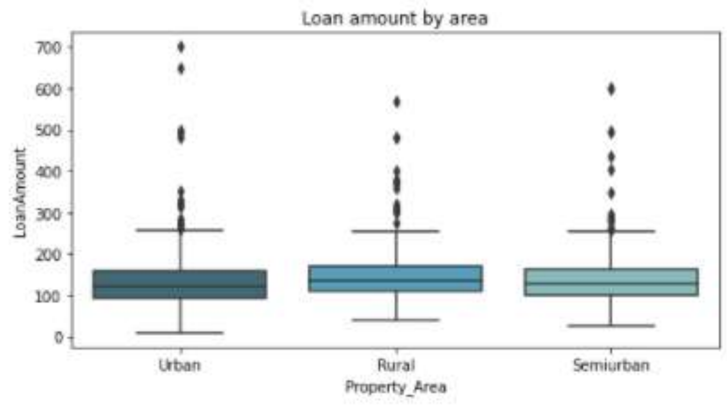
In rural:

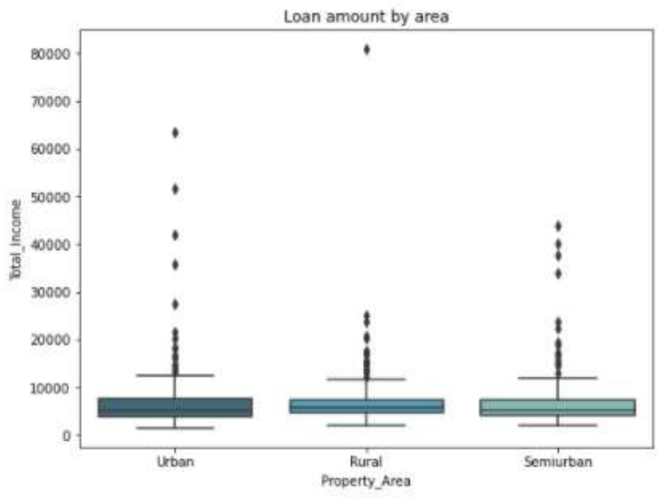
  Graduate people have high income compare to non-graduate

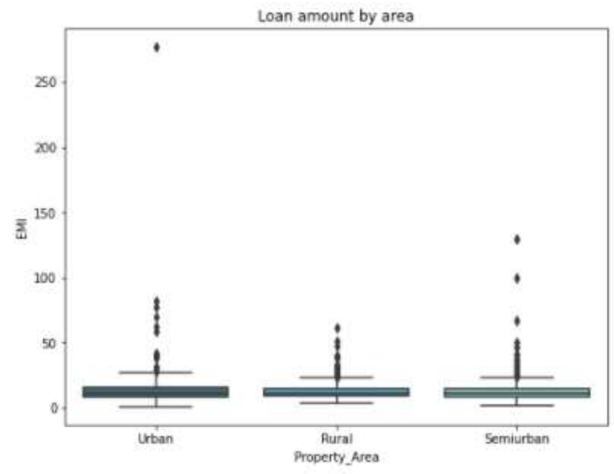
In semiurban:

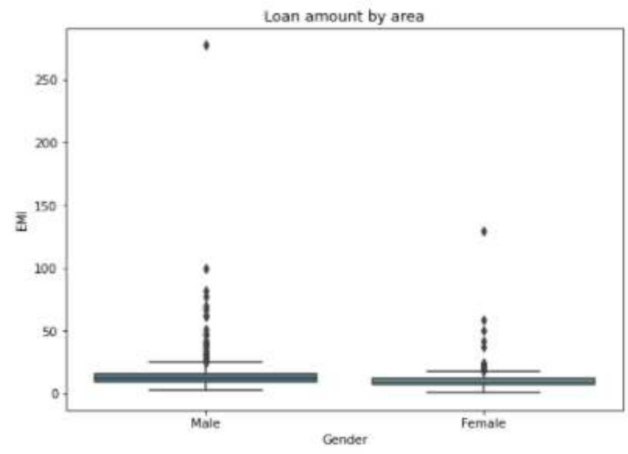
  Graduate people have high income compare to non-graduate

In short we can say that, Graduate people have higher income









EDA CONCLUSION:

GENDER: Male applicant have higher chances to get loan approved and in Urban & Rural area, if applicant is female then 45 to 50% chances of loan rejection.

Married: Married applicant have higher chances to loan approval and unmarried from urban have higher chances to get loan rejection**.**

Self-Employed: Salaried person have higher chances for approval as compare to self-employed. In rural ratio of salaried is lower because there is no jobs in village.

Education: It is common that graduate person have higher income then non-graduate applicants**.** Non-graduate applicant from urban area have higher chances for loan rejection. Graduate applicant have good chances for loan approval.

Dependent: As no of dependent high, there is higher chances to get loan rejection.

Total Income is **higher in Urban** and semi-urban which leads to higher loan approval ratio and **property value** is **high in urban** area compare to rural and semi-urban.

As **Total Income / EMI ratio** goes **up**, **higher chances** to get loan approval. It is applicable for **credit score** as I goes **high**, higher chances for loan approval.

PRE-PROCESSING PIPELINE:

We visualize our data and have some conclusion on it but what are you thinking? Is it sufficient to input for training model? Nop…because there are many issues are available in it like Outliers, Object data types, Skewness. All column need to scale down.

1) Data encoding
2) Outliers removal
3) Skewness removal

There are many methods to encode object data or float or int64. Here we use Label Encoding. Apart from it there are techniques like
Get_dummies , OneHotEncoding, Ordinal Encoding, Ordinal Scale
All of the above techniques convert objective variables to int or float but which should we use is depend on data like here we use LabelEncoding because Sequence doesn't matter in any column but suppose we have rating column then we need to encode with sequence as 5 is better than 3 so at that time we need to use Ordinal Encoding

```
In [58]: le = LabelEncoder()
df["Gender"]= le.fit_transform(df["Gender"])
df["Married"]= le.fit_transform(df["Married"])
df["Education"]= le.fit_transform(df["Education"])
df["Self_Employed"]= le.fit_transform(df["Self_Employed"])
df["Property_Area"]= le.fit_transform(df["Property_Area"])
df["Loan_Status"]= le.fit_transform(df["Loan_Status"])
```

Outliers Removal:

IQR method:

- Where we can divide our data in 4 quartile and then remove outliers from 1st and 4th quartile.

Z-score method:

One of the best method where in normally distributed data, data are plotted till -3 to +3. So using Z-score method, we can remove data, which are outside of this range. To use this method we must need all data column are without objective dtypes.

We use here Z-score method to remove outliers and we have 27 outliers. We removed it and now our data is clean.

In [67]:
```
from scipy.stats import zscore

#di=df.columns
z_score=zscore(df)
print(df.shape)
df_1=df.loc[(z_score<3).all(axis=1)]
print(df_1.shape)
```
```
(614, 12)
(587, 12)
```

Skewness removal:

If data is positively skewed then we can use square root, cube root or log method while for negatively skewed data we can use square, cube root, power transformation or logarithmic method. In our dataset, we will use cube root and power transformation techniques.

In [69]:
```
from scipy.stats import boxcox

for col in df_1:
    if df_1[col].skew()>=0.9:
        df_1[col]=np.cbrt(df_1[col])
    if df_1[col].skew()<= -.6:
        df_1[col]=np.power(df_1[col],2)

#remove skewness using cuberoot and power transformation
```

BUILDING MACHINE LEARNING MODELS

We need to predict yes or no, so it is classification problem. First we will split data in testing set and training set and also split target variable 'Loan Status' as y and other all labels in x. We will feed X as input in model and model will learn it and trained then we will feed Y to it and get predicted target variables.

In [71]:
```
x =df.drop('Loan_Status',axis=1)
y =df['Loan_Status']
```

In [72]:
```
from sklearn.preprocessing import StandardScaler

std=StandardScaler()
x_scaled = std.fit_transform(x)
```

After scalling the data our data is ready for model creation

In [73]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_scaled,y)
```

Define function to predict testing score

In [76]:
```
def fun(f):
    f.fit(x_train,y_train)
    pred=f.predict(x_test)
    print('training Score',f.score(x_train,y_train))
    print('Accuracy Score\n',accuracy_score(y_test,pred))
    print('Confusion Matrix\n',confusion_matrix(y_test,pred))
    print('Classification Report',classification_report(y_test,pred))
    print('f1_score',f1_score(y_test,pred))
```

LOGISTIC REGRESSION

Logistic regression is the proper relapse examination to direct when the reliant variable is dichotomous (parallel). Like all relapse examinations, the strategic relapse is a prescient investigation. Strategic relapse is utilized to depict information and to clarify the connection between one ward twofold factor and at least one ostensible, ordinal, stretch or proportion level autonomous factors.

We got 80.65% training score and 75.97% testing score.

In [77]:
```
fun(lg)
```
```
training Score 0.8065217391304348
Accuracy Score
 0.7597402597402597
Confusion Matrix
 [[23 33]
 [ 4 94]]
Classification Report               precision    recall  f1-score   support

           0       0.85      0.41      0.55        56
           1       0.74      0.96      0.84        98

    accuracy                           0.76       154
   macro avg       0.80      0.68      0.69       154
weighted avg       0.78      0.76      0.73       154

f1_score 0.8355555555555556
```

Hyper parameter tuning For Logistic Regression:

We will work some parameters like solvers; we also give penalty, which reduce the error dimensionality and c value. Divide data in 5 split and repeated test for 4 time using RepeatedStratifiedKFold. Then using GridSearchCV , we will get best parameter to fit model. By using this parameter, we will fit model and get accuracy score.

As we can see, that accuracy is getting higher as 80.38%

```
In [78]: # example of grid searching key hyperparametres for logistic regression
         from sklearn.datasets import make_blobs
         from sklearn.model_selection import RepeatedStratifiedKFold
             from sklearn.model_selection import GridSearchCV

         solvers = ['newton-cg','liblinear','lbfgs','sag','saga']
         penalty = ['l2','elasticnet','l1','none']
         c_value = [100, 1.0, 10, 0.1, 0.01,0.001]

         grid = dict(solver=solvers,penalty=penalty,C=c_value)

         cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=4, random_state=1)

         grid_search = GridSearchCV(estimator=lg, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
         lg1 = grid_search.fit(x_train, y_train)


         best_parameters = lg1.best_params_
         print(best_parameters)

         lg1.best_score_

         {'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}

Out[78]: 0.8038043478260871
```


SUPPORT VECTOR CLASSIFIER

The target of the support vector machine calculation is to find a hyperplane in a N-layered space (N - the quantity of highlights) that distinctly characterizes the relevant elements.

To isolate the two classes of elements, numerous conceivable hyperplanes could be picked. Our goal is to observe a plane that has the greatest edge, i.e the most extreme distance between informative items of the two classes. Augmenting the edge distance gives some support so future information focuses can be grouped with more certainty.

We get the score as below: Training score of 81.95% and Test score of 75.97%

```
In [79]: fun(svc)

         training Score 0.8195652173913044
         Accuracy Score
          0.7597402597402597
         Confusion Matrix
          [[23 33]
          [ 4 94]]
         Classification Report               precision    recall  f1-score   support

                    0       0.85      0.41      0.55        56
                    1       0.74      0.96      0.84        98

             accuracy                           0.76       154
            macro avg       0.80      0.68      0.69       154
         weighted avg       0.78      0.76      0.73       154

         f1_score 0.8355555555555556
```

SVC Hyper parameter tuning

By using below parameter, we find best parameter and fit it with our dataset.
Kernel : 'linear','poly','rbf','sigmoid']
gamma: ['scale','auto']
max_iter : (1,10000)}

We got Training score of 80.21% and 75.97% in testing which is lower than regular svc model

```
In [103]: grid_param = {'kernel' : ['linear','poly','rbf','sigmoid'],'gamma' : ['scale','auto'],'max_iter' : (1,10000)}

In [104]: svc_1 = GridSearchCV(estimator = svc,param_grid = grid_param,cv = 5,n_jobs =-1)
          svc_1.fit(x_train,y_train)

Out[104]: GridSearchCV(cv=5, estimator=SVC(), n_jobs=-1,
                       param_grid={'gamma': ['scale', 'auto'],
                                   'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
                                   'max_iter': (1, 10000)})

In [105]: best_parameters = svc_1.best_params_
          print(best_parameters)

          {'gamma': 'scale', 'kernel': 'linear', 'max_iter': 10000}

In [106]: svc_1.best_score_

Out[106]: 0.8021739130434782

In [107]: svc1 = SVC(gamma= 'auto', max_iter= 10000, kernel = 'rbf')
          svc1.fit(x_train,y_train)

Out[107]: SVC(gamma='auto', max_iter=10000)

In [108]: svc1.score(x_test,y_test)

Out[108]: 0.7597402597402597
```

DECISION TREE CLASSIFIER

Very easy and best model, IN DTC data is split in branches according to possibilities like yes or no, greater than it or not, thus it splited and reached at final outcomes.

Here we get Training score 100 but testing score is very low of 64%

```
n [87]: fun(dtc)

        training Score 1.0
        Accuracy Score
         0.6428571428571429
        Confusion Matrix
         [[23 33]
         [22 76]]
        Classification Report               precision    recall  f1-score   support

                   0       0.51      0.41      0.46        56
                   1       0.70      0.78      0.73        98

            accuracy                           0.64       154
           macro avg       0.60      0.59      0.59       154
        weighted avg       0.63      0.64      0.63       154

        f1_score 0.7342995169082126
```

DTC Hyper parameter tuning

Gini and Entropy : Both are used for building the tree by splitting as per features.

Max_depth : The maximum depth of the tree. If none, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
Min_sample_leaf : will assess the quantity of tests in the hub, and assuming the number is not exactly the base the split will be kept away from and the hub will be a leaf.

As we can see that testing score is increase from 64 to 74%

```
In [88]: grid_param = {
             'criterion' : ['gini','entropy'],
             'max_depth' : range(2,20,3),
             'min_samples_leaf' : range(1,10,2),
             'min_samples_split' : range(2,20,2),
             'splitter' : ['best','random']
         }


         grid_search = GridSearchCV(estimator = dtc,
                              param_grid = grid_param,
                              cv = 5,
                              n_jobs =-1)

         grid_search.fit(x_train,y_train)


         best_parameters = grid_search.best_params_
         print(best_parameters)

         grid_search.best_score_


         dtc1 = DecisionTreeClassifier(criterion= 'gini', max_depth= 3, min_samples_leaf= 5, min_samples_split= 2, splitter= 'random')
         dtc1.fit(x_train,y_train)

         dtc1.score(x_test,y_test)

         {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 1, 'min_samples_split': 4, 'splitter': 'best'}

Out[88]: 0.7402597402597403
```


KNN CLASSIFIER

It called lazy learner classifier as it store the data and worked at output time only. This model save the particular data point place, find out nearest relatable group of dataset, and then categorize this point in that related dataset.
As we can see that Training score is 82% and testing score is 74% in KNN classifier

KNN Classifier

```
In [89]: fun(kn)

         training Score 0.8195652173913044
         Accuracy Score
          0.7402597402597403
         Confusion Matrix
          [[24 32]
          [ 8 90]]
         Classification Report               precision    recall  f1-score   support

                    0       0.75      0.43      0.55        56
                    1       0.74      0.92      0.82        98

             accuracy                           0.74       154
            macro avg       0.74      0.67      0.68       154
         weighted avg       0.74      0.74      0.72       154

         f1_score 0.8181818181818182
```


GAUSSIAN_NB CLASSIFIER:

```
In [90]: fun(gb)

         training Score 0.808695652173913
         Accuracy Score
          0.7402597402597403
         Confusion Matrix
          [[24 32]
          [ 8 90]]
         Classification Report               precision    recall  f1-score   support

                    0       0.75      0.43      0.55        56
                    1       0.74      0.92      0.82        98

             accuracy                           0.74       154
            macro avg       0.74      0.67      0.68       154
         weighted avg       0.74      0.74      0.72       154

         f1_score 0.8181818181818182
```

RANDOM FOREST CLASSIFIER:

It is contain number of decision trees with various subset and take average of it to increase predictive accuracy and we can see that testing accuracy is 77.92% while others are nearly 74 to 75%.

```
In [92]: fun(rf)
         pred=rf.predict(x_test)

         training Score 1.0
         Accuracy Score
          0.7792207792207793
         Confusion Matrix
          [[27 29]
          [ 5 93]]
         Classification Report               precision    recall  f1-score   support

                    0       0.84      0.48      0.61        56
                    1       0.76      0.95      0.85        98

             accuracy                           0.78       154
            macro avg       0.80      0.72      0.73       154
         weighted avg       0.79      0.78      0.76       154

         f1_score 0.8454545454545455
```

Random Forest Classifier with Hyper parameter tuning

We worked on max_depth, max_features = sqrt and estimators =10 and get accuracy of 77%.

```
In [125]: rf1 = RandomForestClassifier(n_estimators = 1000,min_samples_split= 5,min_samples_leaf= 5,max_features= 'sqrt',max_depth= 3,boot:
          rf1.fit(x_train,y_train)

          rf1.score(x_test,y_test)

Out[125]: 0.7727272727272727
```

BOOSTING TECHNIQUES

1) Adaboost classifier
2) Gradientboosting classifier

```
In [95]: fun(ad)
         pred=ad.predict(x_test)

training Score 0.8456521739130435
Accuracy Score
 0.7337662337662337
Confusion Matrix
 [[23 33]
 [ 8 90]]
Classification Report               precision    recall  f1-score   support

           0       0.74      0.41      0.53        56
           1       0.73      0.92      0.81        98

    accuracy                           0.73       154
   macro avg       0.74      0.66      0.67       154
weighted avg       0.74      0.73      0.71       154

f1_score 0.8144796380090498
```

```
In [96]: fun(gd)
         pred=gd.predict(x_test)

training Score 0.9130434782608695
Accuracy Score
 0.7467532467532467
Confusion Matrix
 [[25 31]
 [ 8 90]]
Classification Report               precision    recall  f1-score   support

           0       0.76      0.45      0.56        56
           1       0.74      0.92      0.82        98

    accuracy                           0.75       154
   macro avg       0.75      0.68      0.69       154
weighted avg       0.75      0.75      0.73       154

f1_score 0.8219178082191779
```

Cross Validation

This method is used to predict and check accuracy on unseen or untrained data. If we get 85%, training score it means not that model is always give us accuracy of that level because it is for what we trained it and for that particular pattern. In CV, Model-splitting dataset in different parts, studied multiple time, and then give result.

```
In [98]: score=cross_val_score(lg,x_scaled,y,cv=10)
         print('lg',score.mean())
         score=cross_val_score(lg1,x_scaled,y,cv=10)
         print('lg1',score.mean())
         score=cross_val_score(svc,x_scaled,y,cv=10)
         print('SVC',score.mean())
         score=cross_val_score(svc1,x_scaled,y,cv=10)
         print('SVC1',score.mean())
         score=cross_val_score(dtc,x_scaled,y,cv=10)
         print('dtc',score.mean())
         score=cross_val_score(dtc1,x_scaled,y,cv=10)
         print('dtc1',score.mean())
         score=cross_val_score(kn,x_scaled,y,cv=14)
         print('knn',score.mean())
         score=cross_val_score(gb,x_scaled,y,cv=14)
         print('gb',score.mean())
         score=cross_val_score(rf,x_scaled,y,cv=14)
         print('rf',score.mean())
         score=cross_val_score(rf1,x_scaled,y,cv=14)
         print('rndf1',score.mean())
         score=cross_val_score(ad,x_scaled,y,cv=10)
         print('ad',score.mean())
         score=cross_val_score(gd,x_scaled,y,cv=10)
         print('gd',score.mean())
```


Model Selection:

Here, we worked and develop many models. Now we will learn which factors should be conclude in model selection.
1) Accuracy score: As accuracy score higher, we can conclude it as good model
2) Difference between training and testing score: Low difference with higher accuracy score can be conclude as best model
3) CV score: Higher CV score can be conclude as best model
4) F1 Score: Near to F1 score is the best and near to 0 is the worst.

We should go with model, which perform good prediction with precision on testing score, training score and CV score.

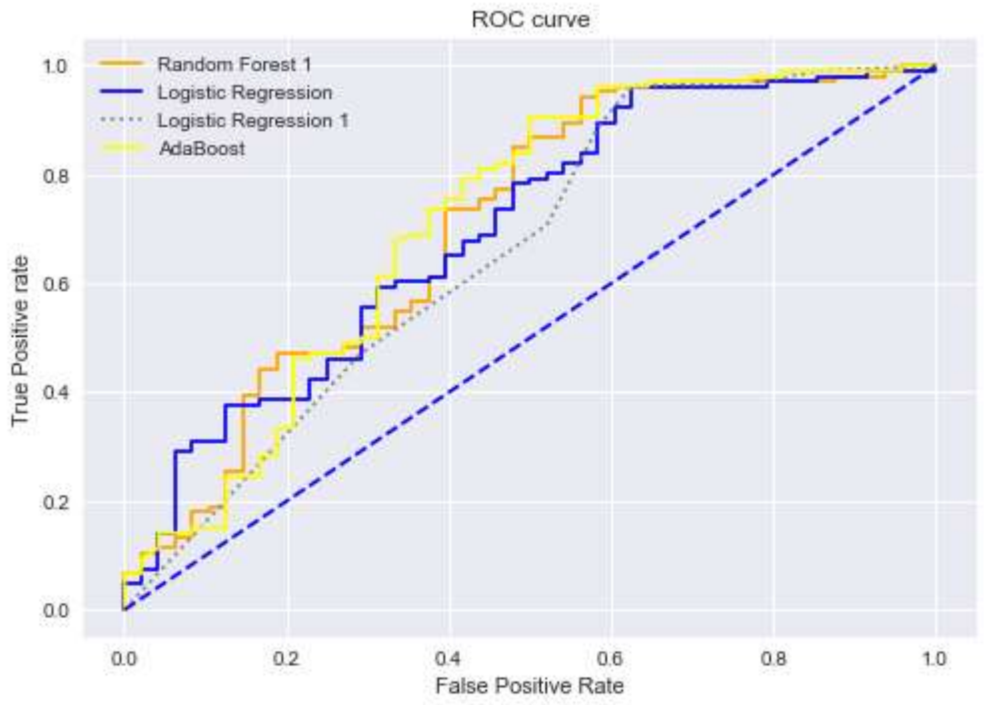
Here, we can see that if we conclude above criteria, RF1 is looking perfect model for us with 80% CV score and 81% testing score. So we will dump it as our final model.

```
In [101]: sd
Out[101]:
```

	Model Name	score	Accuracy Score	F1 Score	CV Score
0	LG	0.81	0.81	0.87	0.79
1	LG1	0.80	0.79	0.80	0.80
2	SVC	0.81	0.81	0.87	0.79
3	SVC1	0.80	0.80	0.86	0.79
4	DTC	1.00	0.64	0.80	0.70
5	DTC1	0.85	0.74	0.84	0.79
6	KNN	0.80	0.78	0.85	0.76
7	GB	0.80	0.81	0.87	0.78
8	RF	1.00	0.81	0.87	0.77
9	RF1	1.00	0.81	0.87	0.80
10	AD	0.83	0.83	0.88	0.78
11	GD	0.90	0.80	0.86	0.77

ROC-AUC CURVE

If we have confusion in model selection due to nearby accuracy or CV score then ROC-AUC curve help us to choose model. Model, which can cover maximum area, can be conclude as good model at initial stage.



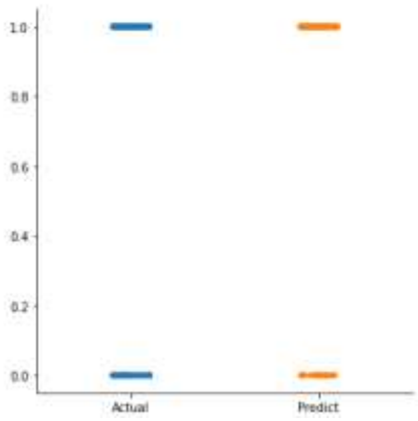
Model Dumping:

We are going to use pickle to dump our model as per below code and will check for prediction after dumping.

```
In [102]: import pickle
          filename='loan.pkl'
          pickle.dump(rf1,open(filename,'wb'))
```

```
In [103]: res=pd.DataFrame()
          res['Actual']=y_test
          res['Predict']=rf1.predict(x_test)

          print(res)
          sns.catplot(data=res)
```



CONCLUDING REMARKS:

By using our model, we can get accurate 80% result about either applicant will get loan or not. We used Random Forest Classifier to develop our model. First, we check basic data then Remove null values, dimensionality reduction, data visualization; Labelencoding then goes for skewness, standardscaler. We divide data in 2 parts of training and testing then develop many model, check score, try to improve it via hyper parameter tuning and finally we can get better accurate result with random forest classifier.