

**LOAN**

**APPLICATION**

**STATUS**

**PREDICTION**



**Your text here**

BY-:

SHASHANK MISHRA

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

*We 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.*

***Loan Status***

**Dependent Variable**

***Problem Definition***

|  |
| --- |
| **Independent Variables** |
| *Gender* |
| *Married* |
| *Dependents* |
| *Education* |
| *Self Employed* |
| *Applicant Income* |
| *Coapplicant Income* |
| *Loan Amount* |
| *Credit History* |
| *Property Area* |

Importing Libraries-:

1. NumPy
2. Pandas
3. Seaborn
4. Matplotlib.pyplot

When we analysed data using these libraries we got the info about

data which is-:

1. There are in total 614 rows and 12 columns in the dataset
2. Yet the info shows there are non-null data, but the we can there some values missing in the row count.
3. There are 8 object columns 1 int and 4 floats columns.
4. We need to convert our object columns into int columns
5. There are few columns which we don’t need so we can drop

those columns such as-: “Loan ID”.

Target column is the last one “Loan Status”.

<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)

Data Analysis

Treating null values-:

* By dropna() method
* By fillna() method
* By Scikit learn Simple Imputer

Treating Object columns-:

* By Pandas get dummies method (can use in gender column)
* By Scikit learn Label Encoder method
* By Scikit learn One Hot Encoder

We analyse data using seaborn countplot-:

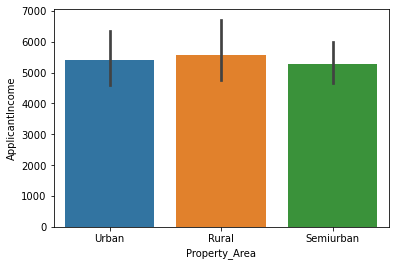
In Univariate analysis we analyse single object columns

eg-: sns.countplot(loan['Loan\_Status'])



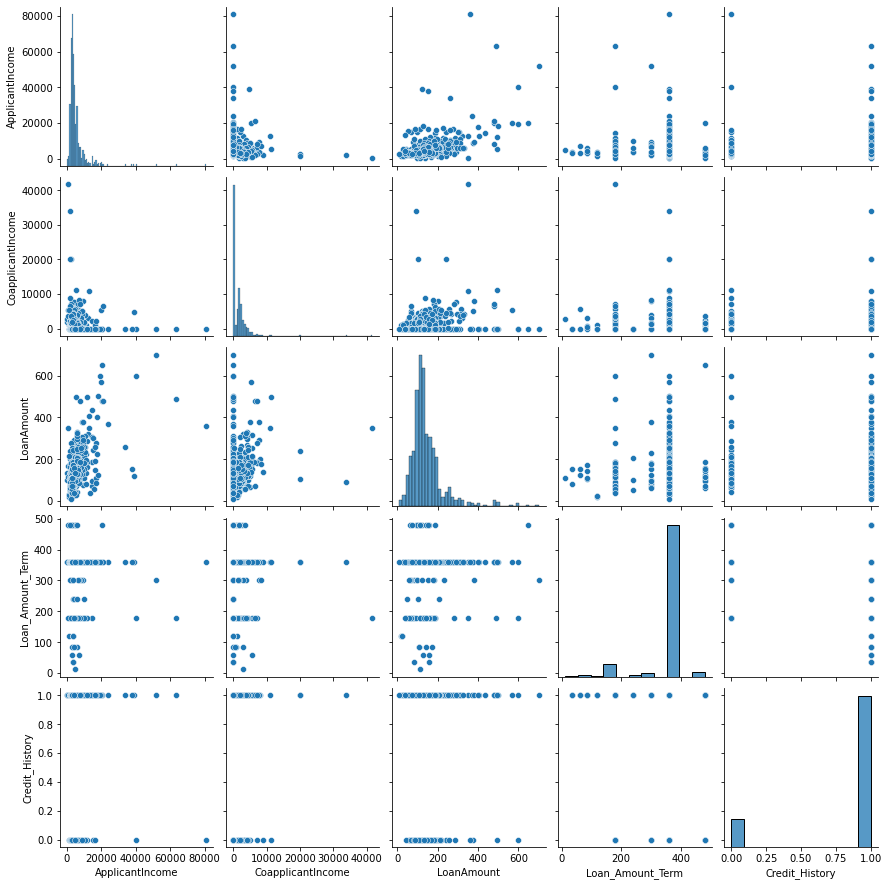
Bivariate Analysis-:

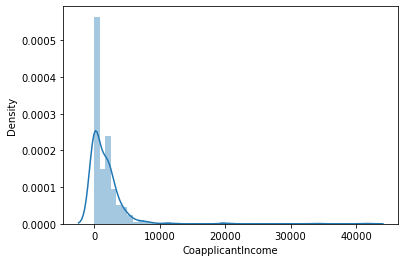
sns.barplot(y = 'ApplicantIncome', x = 'Property\_Area', data = loan)

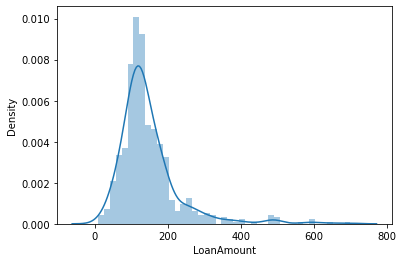


***Exploratory Data Analysis***

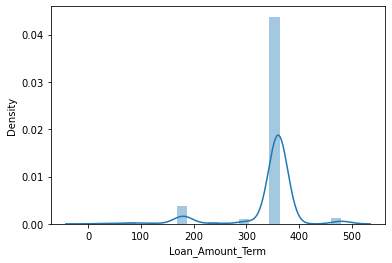
Data Analysis

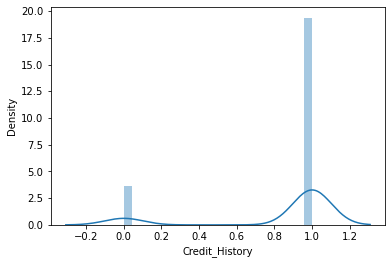






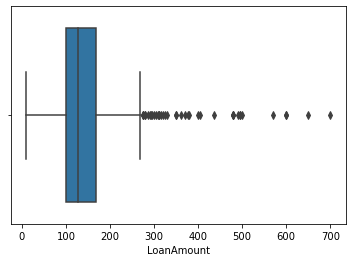


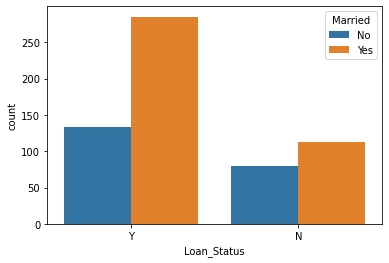
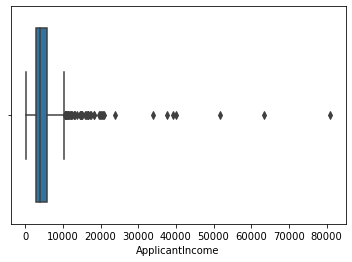




CONCLUSION-:

* Little data is spreaded in Loan Amount and Loan Amount Term column.
* Outliers can be detected by the spreaded data and z-score method.
* Data is present in two classes in Credit History column.
* Data has some skewness in it.
* Credit History is almost imbalance.
* Boxplot can show outliers in the data



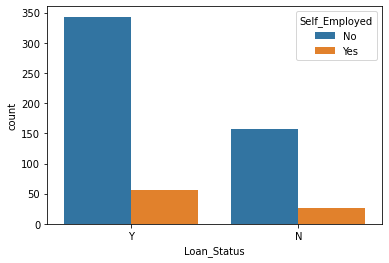


CONCLUSION-:

* The correlation is almost neutral amongst variables except Loan Amount.
* The data is very skewed.
* In Loan Amount section the density shapes bell curve but

very little data is present up to 800 (dispersed data).

* Co-applicant income is very much spreaded from 0-40,000 and data is mostly accumulated in the range of 0-10,000.
* The object columns also have nearly imbalance nature present in it.



See the imbalance nature in the

Object columns.

Data pre-processing flow-:

1. Treating missing values-: the missing values has been treated by the scikit learn method simple imputing-:

eg-: the mean method-:

Pre-processing pipeline

from sklearn.impute import SimpleImputer

si = SimpleImputer(missing\_values=np.nan, strategy= 'mean')

loan\_amt = si.fit\_transform(num['LoanAmount'].to\_numpy().reshape(-1,1))

the mode method-:

si = SimpleImputer(missing\_values=np.nan, strategy='most\_frequent')

gender = si.fit\_transform(obj['Gender'].to\_numpy().reshape(-1,1))

1. Removing outliers-: the outliers has been removed by zscore method-:

Eg-: from scipy.stats import zscore

z = np.abs(zscore(num))

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

1. Data scaling-: we scaled the data using Robust scaler as this scaler is robust to outliers-:

Eg-: from sklearn.preprocessing import RobustScaler

rs = RobustScaler()

appln\_inc = rs.fit\_transform(num['ApplicantIncome'].to\_numpy().reshape(-1,1))

coappln\_inc = rs.fit\_transform(num['CoapplicantIncome'].to\_numpy().reshape(-1,1))

loan\_amt = rs.fit\_transform(num['LoanAmount'].to\_numpy().reshape(-1,1))

loan\_amt\_term = rs.fit\_transform(num['Loan\_Amount\_Term'].to\_numpy().reshape(-1,1))

num['ApplicantIncome'] = appln\_inc

num['CoapplicantIncome'] = coappln\_inc

num['LoanAmount'] = loan\_amt

num['Loan\_Amount\_Term'] = loan\_amt\_term

1. Encoding variables-: we encode the object variables as ML models only take numeric intake-:

Eg-: from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

gender = le.fit\_transform(obj['Gender'])

married = le.fit\_transform(obj['Married'])

depend = le.fit\_transform(obj['Dependents'])

edu = le.fit\_transform(obj['Education'])

employ = le.fit\_transform(obj['Self\_Employed'])

prop\_ar = le.fit\_transform(obj['Property\_Area'])

loan\_status = le.fit\_transform(obj['Loan\_Status'])

obj['Gender'] = gender

obj['Married'] = married

obj['Dependents'] = depend

obj['Education'] = edu

obj['Self\_Employed'] = employ

obj['Property\_Area'] = prop\_ar

obj['Loan\_Status'] = loan\_status

1. Now we had created the final dataframe concating both the numeric and object dataframe-:

Eg- : loan\_new = pd.concat([num,obj], axis = 1)

1. Setting X and y variables-:

ML Model Building

eg-: X = loan\_new.drop(['Loan\_Amount\_Term','Loan\_Status'], axis = 1)

y = loan\_new.Loan\_Status.

1. From libraries importing models and setting model instances-:

Eg-: lr = LogisticRegression()

gnb = GaussianNB()

dtc = DecisionTreeClassifier()

etc = ExtraTreeClassifier()

knc = KNeighborsClassifier()

svc = LinearSVC()

rfc = RandomForestClassifier()

abc = AdaBoostClassifier()

gbc = GradientBoostingClassifier()

1. Giving data to train test splits-:

Eg-: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=101)

1. Setting models into for loop-:

Eg-: models = [lr,gnb,dtc,etc,knc,svc,rfc,abc,gbc]

for m in models:

print(m)

m.fit(X\_train,y\_train)

print(m.score(X\_train,y\_train))

pred\_m = m.predict(X\_test)

print('Model Report')

print(confusion\_matrix(y\_test,pred\_m))

print(classification\_report(y\_test,pred\_m))

print('\n')

#### output-: we selected 5 models which are-:

* logistic regression ----> 0.82
* gaussian NB ----> 0.82
* linear svc ----> 0.82
* random forest classifier -----> 0.80
* gradient boost classifeir ----> 0.81

**on the basis of accuracy score, good f1 score, good precision, and less errors**[**¶**](http://localhost:8888/notebooks/datatrained%20projects/Loan%20Application%20Status.ipynb#on-the-basis-of-accuracy-score,-good-f1-score,-good-precision,-and-less-errors)

1. checking models with cross validation-:

for m in models:

cv = cross\_val\_score(m,X,y,cv=5)

print(m)

print('cv:', cv)

print('mean cv:', cv.mean())

print('\n')

output-: LogisticRegression()

cv: [0.81034483 0.79310345 0.79130435 0.86086957 0.82608696]

mean cv: 0.8163418290854573

GaussianNB()

cv: [0.81034483 0.79310345 0.79130435 0.82608696 0.80869565]

mean cv: 0.8059070464767617

DecisionTreeClassifier()

cv: [0.74137931 0.69827586 0.69565217 0.66956522 0.72173913]

mean cv: 0.7053223388305847

ExtraTreeClassifier()

cv: [0.77586207 0.71551724 0.70434783 0.77391304 0.65217391]

mean cv: 0.7243628185907047

KNeighborsClassifier()

cv: [0.72413793 0.72413793 0.75652174 0.74782609 0.73913043]

mean cv: 0.7383508245877062

LinearSVC()

cv: [0.81034483 0.79310345 0.79130435 0.86086957 0.83478261]

mean cv: 0.8180809595202397

RandomForestClassifier()

cv: [0.77586207 0.78448276 0.76521739 0.80869565 0.82608696]

mean cv: 0.7920689655172414

AdaBoostClassifier()

cv: [0.77586207 0.78448276 0.73913043 0.83478261 0.8173913 ]

mean cv: 0.7903298350824588

GradientBoostingClassifier()

cv: [0.73275862 0.75862069 0.75652174 0.85217391 0.83478261]

mean cv: 0.7869715142428786

1. now we are checking with grid search CV-:

eg-: lr\_param = {

'C':[0.001,0.1,1,10],

'fit\_intercept': [True, False],

'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],

'multi\_class': ['auto', 'ovr', 'multinomial'],

'verbose': [1,10,100,200]

}

# gnb has not much parameters

lsvc\_param = {

'loss':['hinge','squared\_hinge'],

'C': [0.001,0.1,1,10],

'multi\_class': ['ovr','crammer\_singer'],

'fit\_intercept': [True, False],

'verbose': [1,10,100,200]

}

rfc\_param = {

'n\_estimators': [100,300,500],

'criterion': ['gini','entropy'],

'min\_samples\_split': [2,3,4,5],

'verbose': [1,10,100,200]

}

gbc\_param = {

'loss': ['deviance','exponential'],

'n\_estimators': [100,300,500],

'criterion': ['friedman\_mse','mse','mae'],

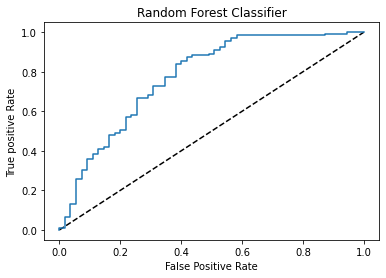
'max\_depth': [1,3,5,7,9,10]

}

Output-: we got the best model -: Random forest classifier with the best params-:

rfc2 = RandomForestClassifier(n\_estimators=100, criterion='gini', min\_samples\_split=5, verbose=10 )

and with ROC-AUC curve and score-:



And AUC score-: 0.7052139037433156

1. we save the best model by importing joblib with the name “rfc2file.obj”.

Concluding Remarks-:

The model has predicted 80% of the test data correctly.

And it can predict better with future data.

The model has the probability of 70% of falling data

under the curve, so it tends to have good prediction with good score.