LOAN APPROVAL PREDICTION MODEL ANALYSIS

SABAL SHARMA, OBAID SAYED AND JANBHI SHARMA

THE PROBLEM

The cost of assets is increasing day by day and the capital required to purchase an entire asset is very high. So purchasing it out of your savings is not possible. The easiest way to get the required funds is to apply for a loan. But taking a loan is a very time consuming process. The application has to go through a lot of stages and it's still not necessary that it will be approved. To decrease the approval time and to decrease the risk associated with the loan many loan prediction models were introduced. This model extracts and introduces the essential features of a borrower that influence the customer's loan status. Finally, it produces the planned performance (loan status). These reports make a bank manager's job simpler and quicker.

We'll start by exploratory preprocessing, then data analysis, and finally we'll be testing different models such as Logistic regression and decision trees.

PROPOSED METHODOLOGY

The paper will be comparing different prediction models and deduce their limitations as well as advantages. On the basis of the results, a modified prediction model will be created to ensure maximum accuracy and performance.

IMPLEMENTATION

- **We'll import the necessary libraries and load the data** :

```
#importing required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
#Reading and assigning csv files to variables
df_x=pd.read_csv('loan_train.csv')
df_y=pd.read_csv('loan_test.csv')
```

- **Dataset**

The data consists of the following rows:

```
Loan_ID : Unique Loan ID

Gender : Male/ Female

Married : Applicant married (Y/N)

Dependents : Number of dependents

Education : Applicant Education (Graduate/ Under Graduate)

Self_Employed : Self employed (Y/N)

ApplicantIncome : Applicant income

CoapplicantIncome : Coapplicant income

LoanAmount : Loan amount in thousands of dollars

Loan_Amount_Term : Term of loan in months

Credit_History : credit history meets guidelines yes or no

Property_Area : Urban/ Semi Urban/ Rural

Loan_Status : Loan approved (Y/N) this is the target variable
```

- **UNDERSTANDING DATASET**

```
len(df_x)
```

```
614
```

```
len(df_x.columns)
```

```
13
```

```
df_x.shape
```

```
(614, 13)
```

```
df_x.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
```

- ## **Data Cleaning**

  The first thing to do is to deal with the missing value, lets check first how many there are for each variable.

```
#calclulating null values
df_x.isna().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
```

Here, we fill the null values with the smallest number of the value count.

```
# filling null values with the smaller number from values count
df_x['Gender'].value_counts()
df_x['Gender'].fillna('Female',inplace=True)
df_x.isna().sum()
```

```
Loan_ID               0
Gender                0
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
```

With this there are no null values in the Gender column of our dataset. Similarly, We fill the null values in all the other columns of our dataset.

```python
# filling null values with the smaller number from values count
df_x['Married'].value_counts()
df_x['Married'].fillna('No',inplace=True)
df_x.isna().sum()
```

```
Loan_ID               0
Gender                0
Married               0
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
```

```python
# filling null values with the smaller number from values count
df_x['Dependents'].value_counts()
df_x['Dependents'].fillna('3+',inplace=True)
df_x.isna().sum()
```

```
Loan_ID               0
Gender                0
Married               0
Dependents            0
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
```

```
# filling null values with the smaller number from values count
df_x['Self_Employed'].value_counts()
df_x['Self_Employed'].fillna('Yes',inplace=True)
df_x.isna().sum()
```

```
Loan_ID               0
Gender                0
Married               0
Dependents            0
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

```
# filling null values with the larger number from values count
df_x['Loan_Amount_Term'].value_counts()
df_x['Loan_Amount_Term'].fillna('360.0',inplace=True)
df_x.isna().sum()
```

```
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       50
Property_Area         0
Loan_Status           0
dtype: int64
```

```
# filling null values with the smaller number from values count
df_x['Credit_History'].value_counts()
df_x['Credit_History'].fillna('0.0',inplace=True)
df_x.isna().sum()
```

```
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
```

We fill the null values in the ‘Loan Amount’ column with the mean of all the values from the column since it doesn’t have discrete values

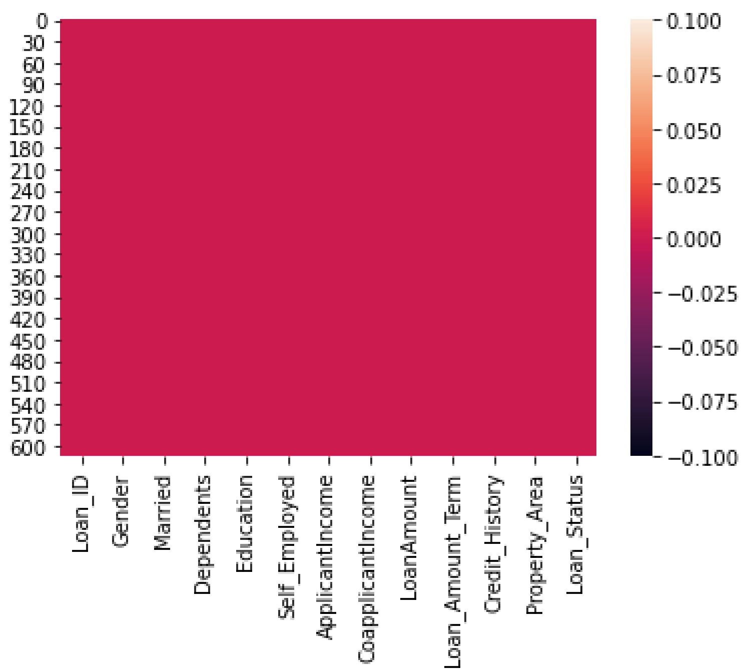
```
# filling null values with the mean of LoanAmount column as it doesn't have discrete values
df_x['LoanAmount'].value_counts()
df_x['LoanAmount'].fillna(df_x['LoanAmount'].mean(),inplace=True)
df_x.isna().sum()
```

```
Loan_ID               0
Gender                0
Married               0
Dependents            0
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

We plot a heatmap to check if any null value remains in our dataset.

```
sns.heatmap(df_x.isnull())
```

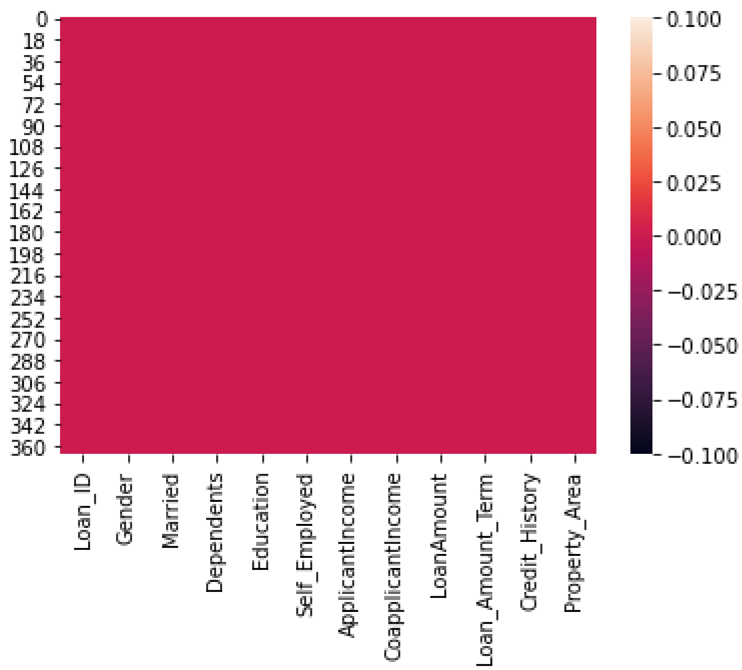
```
<AxesSubplot:>
```



Similarly, We clean our Test dataset.
The Heat map of null values in our Test dataset:-

```
sns.heatmap(df_y.isnull())
```

<AxesSubplot:>



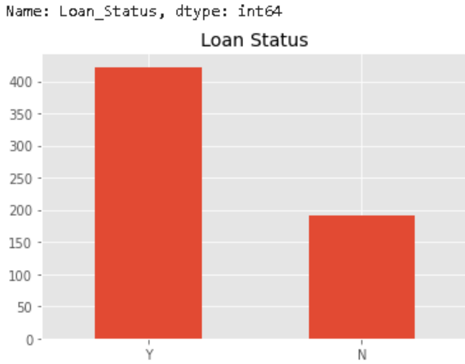
- ## Exploratory Data Analysis

Target variable – Loan Status

We will start first with an independent variable which is our target variable as well. We will analyse this categorical variable using a bar chart as shown below. The bar chart shows that loan of 422 (around 69 %) people out of 614 was approved

```
#Checking loan status for the applicants
plt.style.use('ggplot')
df_x['Loan_Status'].value_counts().plot.bar(title='Loan Status',rot=0)
display(df_x['Loan_Status'].value_counts())
```

```
Y    422
N    192
Name: Loan_Status, dtype: int64
```



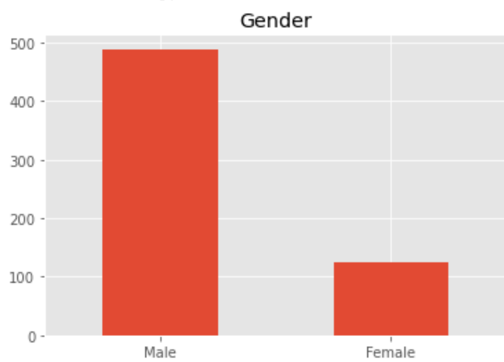
There are 3 types of Independent Variables: Categorical, Ordinal & Numerical.

- **Categorical Variables**
    1. Gender
    2. Marrital status
    3. Employement type
    4. Credit History

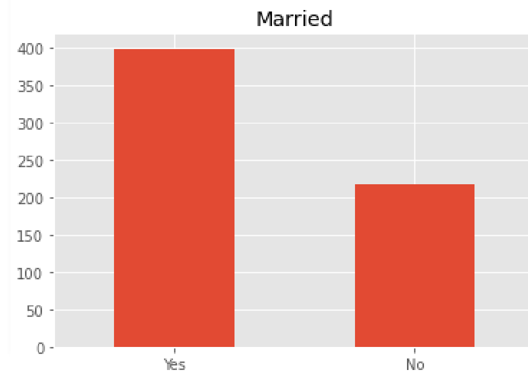
It can be inferred from the graphs plot below that our data

- 80% of the loan applicants are male.
- Nearly 70% are married.
- About 75% of loan graduates are educated.
- Nearly 80-85% applicant are self-employed.
- The loan has been approved for more than 65% of applicants.

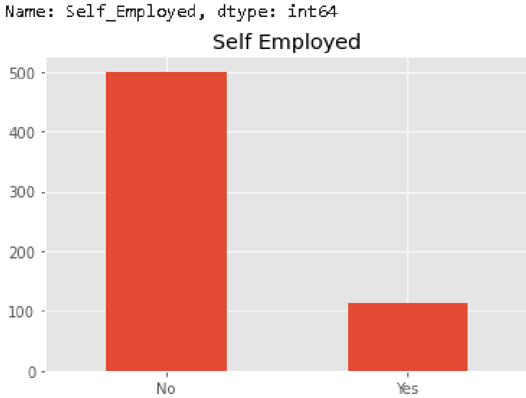
```
Male      489
Female    125
Name: Gender, dtype: int64
```



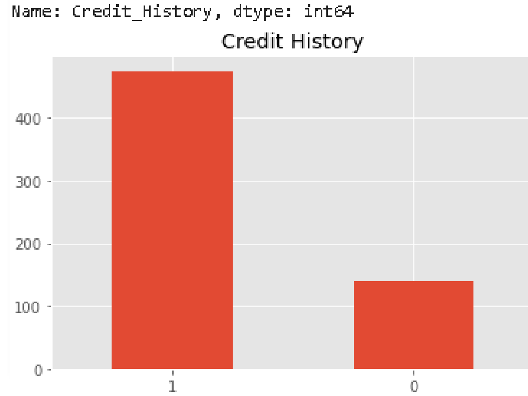

```
Yes    398
No     216
Name: Married, dtype: int64
```




```
No     500
Yes    114
Name: Self_Employed, dtype: int64
```




```
1    475
0    139
Name: Credit_History, dtype: int64
```

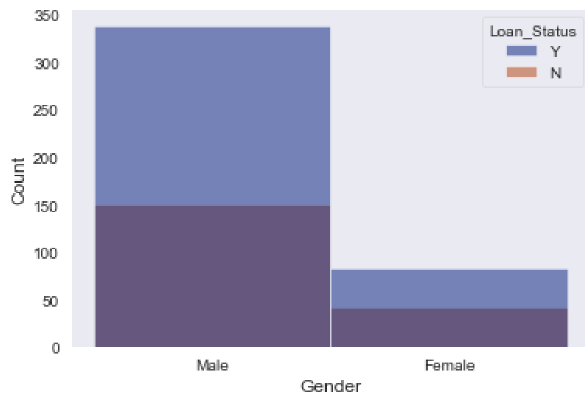


- **Categorical values vs target values**

1. Gender vs loan status

```
sns.histplot(x='Gender',palette='dark',data=df_x,hue=df_x['Loan_Status'])
```

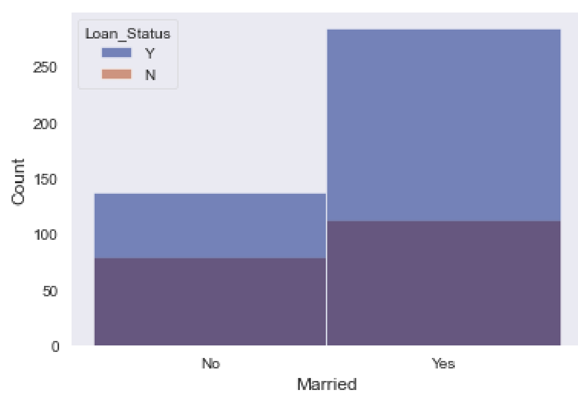
```
<AxesSubplot:xlabel='Gender', ylabel='Count'>
```



2. Married vs loan status

```
sns.histplot(x='Married',palette='dark',data=df_x,hue=df_x['Loan_Status'])
```

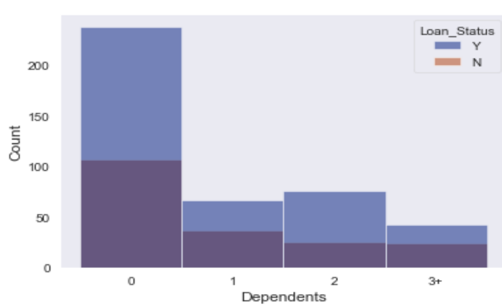
```
<AxesSubplot:xlabel='Married', ylabel='Count'>
```



3. Dependent vs loan status

```
sns.histplot(x='Dependents',palette='dark',data=df_x,hue=df_x['Loan_Status'])
```

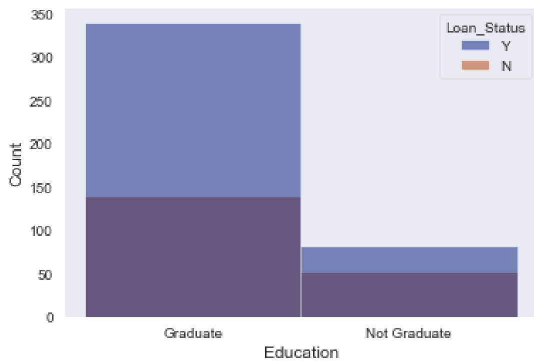
```
<AxesSubplot:xlabel='Dependents', ylabel='Count'>
```



4. Education vs Loan status

```
sns.histplot(x='Education',palette='dark',data=df_x,hue=df_x['Loan_Status'])
```

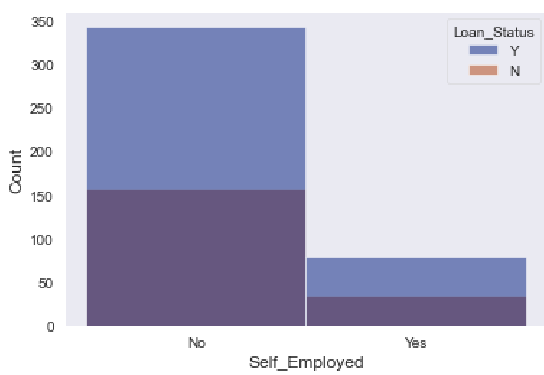
```
<AxesSubplot:xlabel='Education', ylabel='Count'>
```



5. Self-Employed vs Loan status

```
sns.histplot(x='Self_Employed',palette='dark',data=df_x,hue=df_x['Loan_Status'])
```

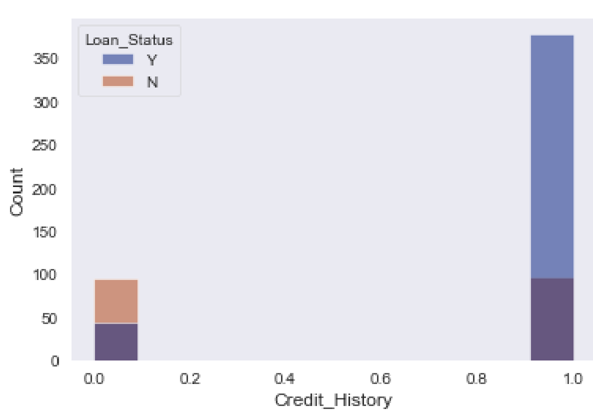
```
<AxesSubplot:xlabel='Self_Employed', ylabel='Count'>
```



6. Credit History vs Loan status

```
sns.histplot(x='Credit_History',palette='dark',data=df_x,hue=df_x['Loan_Status'])
```

```
<AxesSubplot:xlabel='Credit_History', ylabel='Count'>
```



- ## DATA VISUALIZATION

  In this section, We are showing the visual information from the dataset, For that we need some pakages that are matplotlib and seaborn.

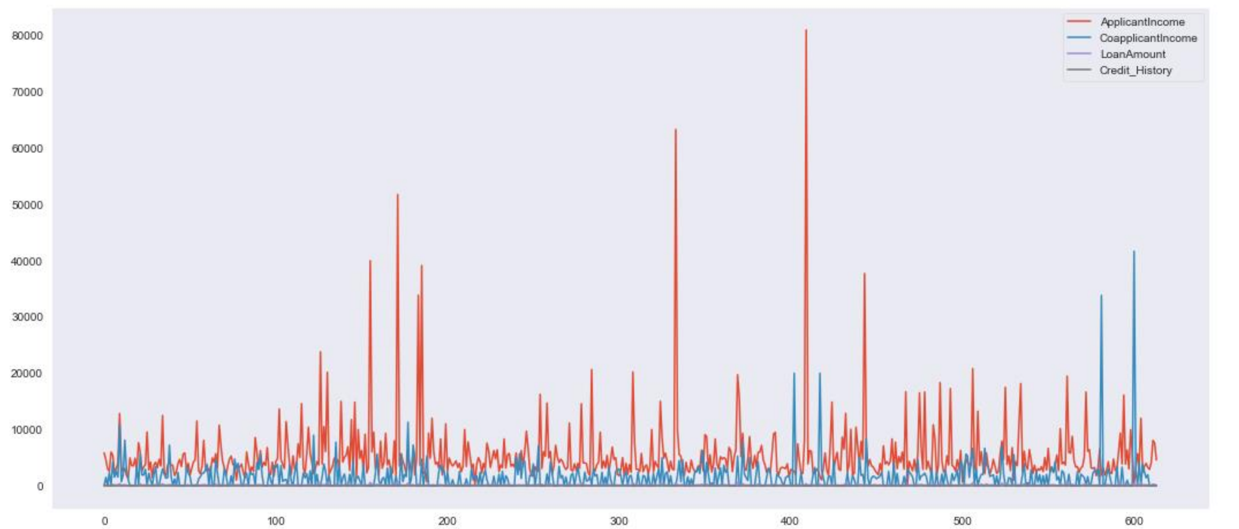
```
import matplotlib.pyplot as plt
%matplotlib inline


import seaborn as sns
sns.set_style('dark')
```

Plotting more graphs and visualization to infer more results:

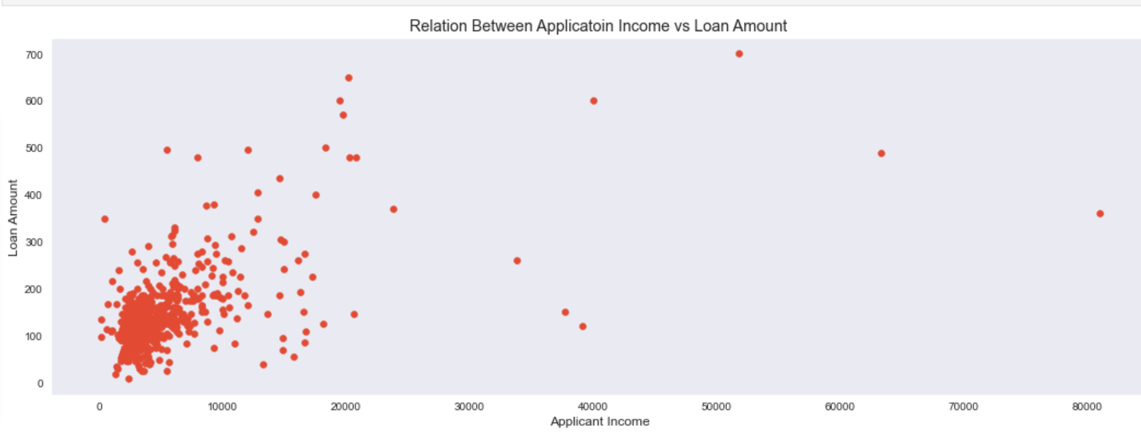
```
df_x.plot(figsize=(18, 8))

plt.show()
```




```
plt.figure(figsize=(18, 6))
plt.title("Relation Between Applicatoin Income vs Loan Amount ")

plt.scatter(df_x['ApplicantIncome'] , df_x['LoanAmount'])
plt.xlabel("Applicant Income")
plt.ylabel("Loan Amount")
plt.show()
```



- ## Modeling

  We're gonna use sklearn for our models, before doing that we need to turn all the categorical variables into numbers. We'll do that using the LabelEncoder in sklearn.

```
#Encoding required columns data
from sklearn.preprocessing import LabelEncoder
cols = ['Gender',"Married","Education",'Self_Employed',"Property_Area"]
le = LabelEncoder()
for col in cols:
    df_x[col] = le.fit_transform(df_x[col])
```

```
#dropping features which has no use
cols = ['ApplicantIncome', 'CoapplicantIncome', "LoanAmount", "Loan_Amount_Term",'Loan_ID', 'CoapplicantIncome', 'Dependents']
df_x = df_x.drop(columns=cols, axis=1)
df_x.head()
```

	Gender	Married	Education	Self_Employed	Credit_History	Property_Area	Loan_Status
0	1	0	0	0	1	2	Y
1	1	1	0	0	1	0	N
2	1	1	0	1	1	2	Y
3	1	1	1	0	1	2	Y
4	1	0	0	0	1	2	Y

We follow similar steps for the Test dataset.

Now all our variables have became numbers that our models can understand.

Assigning columns to variables.

```
cols=['Credit_History', 'Education', 'Gender','Self_Employed']
X_train=df_x[cols].values
y_train=df_x['Loan_Status'].values
```

```
X_test=df_y[cols].values
```

```
le.fit(y_train)
y_train=le.transform(y_train)
```

We will not start to fit different models and measure their accuracy with the dataset.

We will start with random forest

- ## Random Forest

  This is a tree based ensemble model which helps in improving the accuracy of the model . It combines a large number of Decision trees to build a powerful predicting model. It takes a random sample of rows and features of each individual tree to prepare a decision tree model. Final prediction class is either the mode of all the predictors or the mean of all the predictors.

```
from sklearn.ensemble import RandomForestClassifier
```

```
model=RandomForestClassifier()
```

```
model.fit(X_train,y_train)
```

```
RandomForestClassifier()
```

```
y_pred=model.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
score = model.score(X_train, y_train)
print('accuracy_score overall :', score)
print('accuracy_score percent :', round(score*100,2))
print('accuracy_score',accuracy_score(y_test,y_pred1))
```

```
accuracy_score overall : 0.7703583061889251
accuracy_score percent : 77.04
```

Here the accuracy score is 0.77.

We will now try another model for the same.

- ## K Nearest Neighbour

  The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems. It's easy to implement and understand, but has a major drawback of becoming significantly slows as the size of that data in use grows.

```
from sklearn.neighbors import KNeighborsClassifier
```

```
model1=KNeighborsClassifier(n_neighbors=3)
```

```
model1.fit(X_train,y_train)
```

```
KNeighborsClassifier(n_neighbors=3)
```

```
y_pred1=model1.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
print("Model Accuracy:- ",accuracy_score(y_test,y_pred1))
```

```
Model Accuracy:-  0.5844155844155844
```

  K nearest neighbor has an accuracy of 0.5844155 i.e. 58.44%
  Here we see that the model has a lower accuracy as compared to the Random Forrest.

  We will now try another model.

- ## Logistic Regression

  This is a classification algorithm which uses a logistic function to predict binary outcome (True/False, 0/1, Yes/No) given an independent variable. The aim of this model is to find a relationship between features and probability of particular outcome. The logistic function used is a logit function which is a log of odds in the favor of the event. Logit function develops a s-shaped curve with the probability estimate similar to a step function.

```
from sklearn.linear_model import LogisticRegression
```

```
model2=LogisticRegression()
```

```
model2.fit(X_train,y_train)
```

```
LogisticRegression()
```

```
y_pred2=model2.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
print("Model Accuracy:- ",accuracy_score(y_test,y_pred2))
```

```
Model Accuracy:-  0.7987012987012987
```

Here we see that the accuracy for logistic Regression is the highest i.e 0.798701
Or 79.87%

Now let us Build a text report to show the main classification metrics.

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
print(confusion_matrix(y_test,y_pred2))
print(classification_report(y_test,y_pred2))
```

```
[[ 22  21]
 [ 10 101]]
              precision    recall  f1-score   support

           0       0.69      0.51      0.59        43
           1       0.83      0.91      0.87       111

    accuracy                           0.80       154
   macro avg       0.76      0.71      0.73       154
weighted avg       0.79      0.80      0.79       154
```

```
y_pred2
```

```
array([1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0])
```

```
y_pred2 = model2.predict([[    1,      0.0,    2811,   1666.0, 1544.0, 360.0,   1.0,    2]])
print(y_pred2)
```

```
[1]
```

Now You can save it to a file

```
import pickle
# now you can save it to a file
file = 'predictor.pkl'
with open(file, 'wb') as f:
    pickle.dump(model2, f)
```

```
with open(file, 'rb') as f:
    k = pickle.load(f)
```

```
test = k.predict([[    1,      0.0,    4230,    0.0,    112.0,  360.0,  1.0,    1]])
print(test)
```

```
[1]
```

- ## Model Deployement

  Now, We will deploy our model using **Streamlit**.

  https://share.streamlit.io/oga997/loan_predictor/Loan-Predictor.py

  Click the link to check the deployed Model.

- ## Conclusion

  The predictive models based on Logistic Regression, K Nearest Neighbour and Random Forest, give the accuracy as 79.87%, 58.44% and 77.44% respectively. This shows that for the given dataset, the accuracy of model based on logistic regression is highest.

THE SOLUTION

The aim of this project was to compare the various Loan Prediction Models and show which is the best one with the least amount of error and could be used by banks in real world to predict if the loan should be approved or not taking the risk factor in mind. After comparing and analyzing the models, it was found that the prediction model based on **Logistic Regression** proved to be the most accurate and fitting of them all. This can be useful in reducing the time and manpower required to approve loans and filter out the perfect candidates for providing loans.

BUSINESS RECOMMENDATION

Using the deployed model as a standard Basic idea and creating a more interactive application/website This will reduce the human error as there will be less manual judgement for the approval of loan instead the application has to do all the work on the basis of all the features provided and can predict if an applicant should be approved for loan.

REFRENCES

[1] Vaidya and Ashlesha, Predictive and probabilistic approach using logistic regression: Application to prediction of loan approval, 2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT). IEEE, 2017.

[2] Amin, Rafik Khairul and Yuliant Sibaroni, Implementation of decision tree using C4. 5 algorithm in decision making of loan application by debtor (Case study: Bank pasar of Yogyakarta Special Region), 2015 3rd International Conference on Information and Communication Technology (ICoICT). IEEE, 2015.

[3] Arora, Nisha and Pankaj Deep Kaur, A Bolasso based consistent feature selection enabled random forest classification algorithm: An application to credit risk assessment, Applied Soft Computing 86 (2020), 105936.

[4] Yang, Baoan, et al, An early warning system for loan risk assessment using artificial neural networks, Knowledge-Based Systems 14.5-6 (2001), 303-306.

[5] Metawa, Noura, M. Kabir Hassan and Mohamed Elhoseny, Genetic algorithm based model for optimizing bank lending decisions, Expert Systems with Applications 80 (2017), 75-82.

[6] Hassan, Amira Kamil Ibrahim and Ajith Abraham. "Modeling consumer loan default prediction using ensemble neural networks, 2013 International Conference On Computing, Electrical And Electronic Engineering (ICCEEE). IEEE, 2013