PROJECT REPORT

CHAPTER 1
INTRODUCTION

1.1 PROJECT OVERVIEW

Technology has made a lot of improvements, and the banking industry is no exception. Every day, there are more applications to approve loans. When choosing an applicant for loan approval, they must take into account a few bank policies. The bank has to choose which is ideal for approval based on a few criteria. It is tough and risky to check out manually every person and then recommended for loan approval. In this work, we use a machine learning technique that will predict the person who is reliable for a loan, based on a few details. This work's primary objective is to predict whether an individual is eligible for loan or not.

Loan prediction is one of the most important and most prominent research areas in the field of banking and insurance sectors. In the modern environment identifying and analyzing the patterns of the obtained sample dataset plays a vital role in this era. The loan prediction involves the application of various machine learning algorithms. There are some prediction systems in the market using deep learning and so on. But those are limited with certain features and cannot assist the users beyond those limits. The loan prediction project is developed using machine learning algorithms such as logistic regression, K-Nearest Neighbour. The Python programming language is used for the implementation of the code and the html pages are developed for deployment of website using Visual Studio code. The proposed system can deliver high accuracy results and moderate loss for training and validate data. Finally, the results show the model implemented with high accuracy.

1.2 PURPOSE

Banks are making the major part of profits through loans. Though lot of people are applying for loans, it is hard to select the genuine applicant, who will repay the loan. While doing the process manually, many mistakes may arise when choosing the genuine applicant. Therefore, we are developing loan prediction system using machine learning so the system automatically selects the eligible candidates. This is helpful to both bank staff and applicant. The time period for the sanction of loan will be drastically reduced.

A loan is the core business part of banks. The main portion of the bank's profit directly comes from the profit earned from the loans. Though bank approves loan after a regress process of verification and testimonial but still there's no guarantee whether the chosen person is the right person or not. This process takes a lot of time while doing it manually. We can prophesy whether that particular person is safe or not and the whole process of testimonial is automated by machine literacy style. Loan Prognostic is really helpful for retainer of banks as well as for the hopeful also.

CHAPTER 2

LITERATURE SURVEY

2.1 EXISTING PROBLEM:

M. A. Sheikh, A. K. Goel and T. Kumar developed Machine learning model using Logistic Regression and achieved an accuracy of 81.11%. It was inferred that those with a good credit score, high income and low loan amount requirement will get their loan approved. Applicants with Credit history not passing fails to get approved, Probably because of that they have a probability of not paying back. Most of the Time, Applicants with high income sanctioning low amount is more likely to get approved which make sense, more likely to pay back their loans.

S. Z. H. Shoumo, M. I. M. Dhruba et al implemented Logistic Regression, Support Vector Machine, Random Forest and Extreme Gradient Boosting algorithms. All algorithms achieved almost the same accuracy. We would like to implement other dimensionality reduction techniques such as genetic algorithm, univariate feature selection methods, tree-based feature selections etc. to gauge their performances and further improve the efficiency of the credit lending sector.

Sivasree M S, Rekha Sunny T implemented Decision Tree Induction data mining technique, that is used to generate the relevant attributes relevant for credibility and also to make the decision in the model. The model is tested only for particular dataset and Other methodologies that perform better than common data mining algorithms must be incorporated and tested for the domain.

B. Patel, H. Patil et al implemented Gradient Boosting, Logistic Regression, Random Forest and CatBoost Classifier. Logistic Regression gave a very low accuracy of 14.96%. Random forest gave a good accuracy of 83.51%. The CatBoost Classifier and Gradient Boosting achieved the best accuracy of 84.04% and 84.03% respectively. Here in this paper, we have only considered home loan prediction, a system could be made for predicting defaulters of other loans as well. Also, whether the non-defaulter would turn out to be a fraudster or not could be predicted.

2.2 REFERENCES:

[1] M. A. Sheikh, A. K. Goel and T. Kumar, "An Approach for Prediction of Loan Approval using Machine Learning Algorithm," 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), 2020, pp. 490-494, doi:10.1109/ICESC48915.2020.9155614.

[2] Sarwesh Site, Dr. Sadhna K. Mishra, “A Review of Ensemble Technique for Improving Majority Voting for Classifier”, Volume 3, Issue 1, January 2013.

[3] S. Z. H. Shoumo, M. I. M. Dhruba, S. Hossain, N. H.Ghani, H. Arif and S. Islam, "Application of Machine Learning in Credit Risk Assessment: A Prelude to Smart Banking," TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON), 2019, pp. 2023-2028, doi:10.1109/TENCON.2019.8929527.

[4] B. Patel, H. Patil, J. Hembram and S. Jaswal, "Loan Default Forecasting using Data Mining," 2020 International Conference for Emerging Technology (INCET), 2020, pp. 1-4, doi:10.1109/INCET49848.2020.9154100.

2.3 PROBLEM STATEMENT DEFINITION

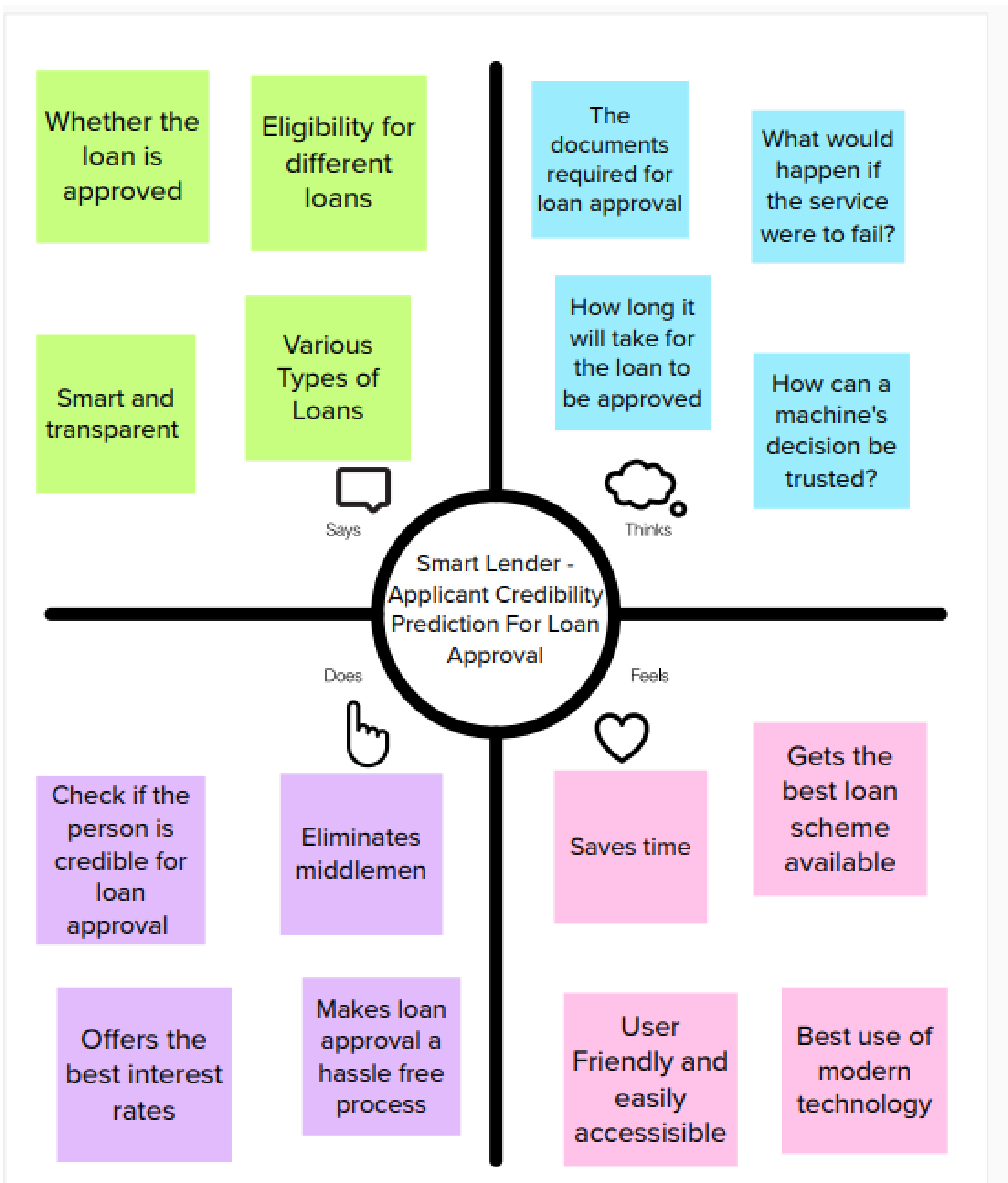
Loan prediction is a very common real-life problem that each retail bank faces at least once in its lifetime. If done correctly, it can save a lot of man-hours at the end of a retail bank.

It is a classification problem where we have to predict whether a loan would be approved or not. In these kinds of problems, we have to predict discrete values based on a given set of independent variables.

CHAPTER 3

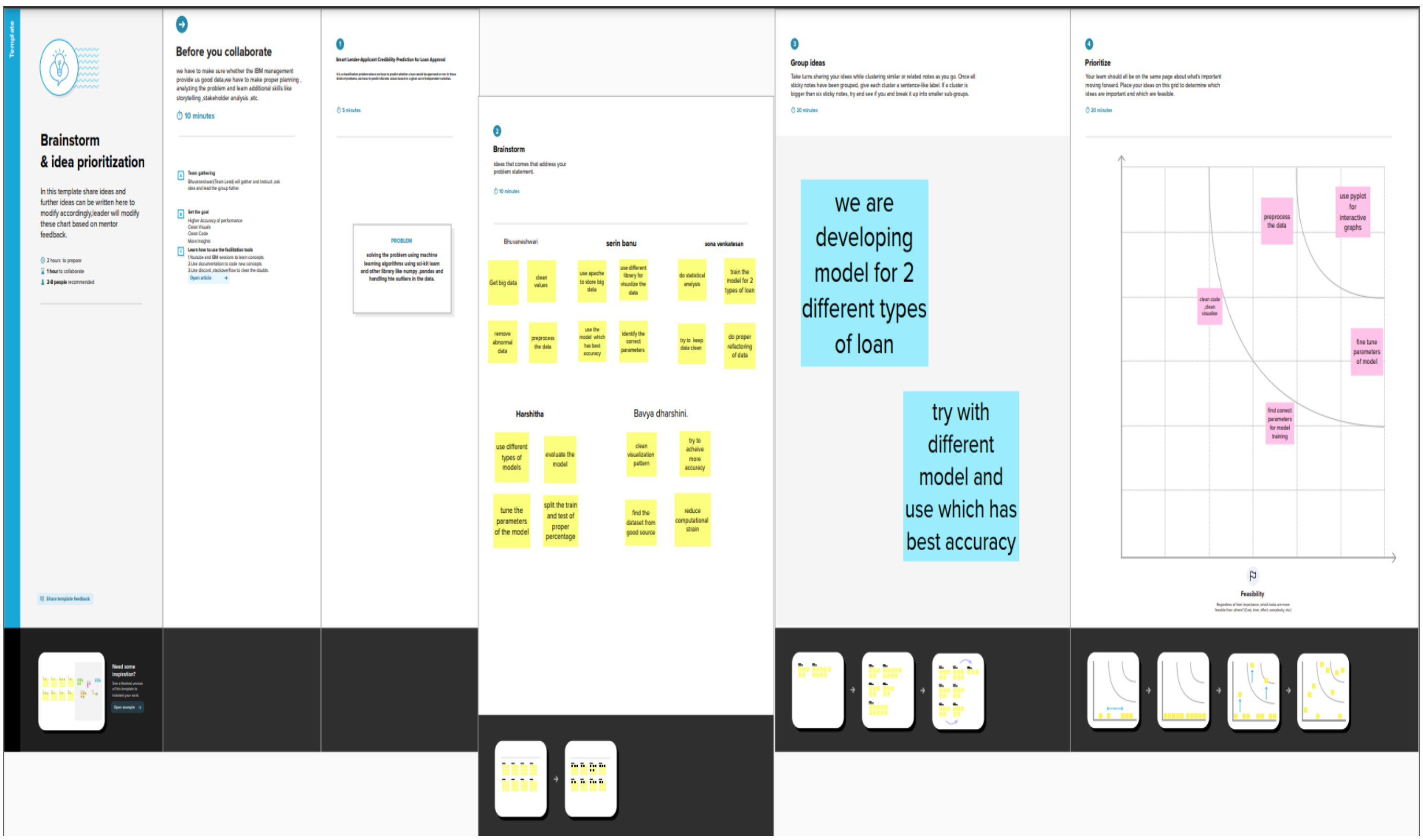
IDEATION & PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS:



3.2 IDEATION AND BRAINSTORMING:

The project implements machine learning techniques such as decision tree models, random forest models, KNN models, and XGBoost models to predict loan defaults. We train and test these models on the given dataset. The best model is selected by comparing the accuracy and this model is saved. This model is integrated into a flask-based web application.



3.3 PROPOSED SOLUTION:

S.NO	PARAMETER	DESCRIPTION
1	Problem Statement (Problem to be solved)	The business intends to automate (in real-time) the loan eligibility process using information that customers supply on online application forms. They have offered a dataset to determine the client segments that are eligible for loan amounts in order to automate this procedure and target these customers directly. The profitability or loss of a bank is mostly determined by the loans it makes, namely whether or not its customers are making their loan repayments. The bank can lower its Non-Performing Assets by anticipating loan defaulters. Predicting whether a certain loan will default based on the initial information provided by the borrowers and their credit report will increase the bank's profit.
2	Idea / Solution description	The project implements machine learning techniques such as decision tree models, random forest models, KNN models, and XGBoost models to predict loan defaults. We train and test these models on the given dataset. The best model is selected by comparing the accuracy and this model is saved. This model is integrated into a flask-based web application.
3	Novelty / Uniqueness	This project optimizes the model by continuously changing the parameters of the model so that the right model with the right parameters is selected as the best model
4.	Social Impact / Customer Satisfaction	This application evaluates a borrower's credibility before approving a loan. Credit score or prediction is used in this case to assist bank staff in precisely and efficiently categorizing credit defaulters. So, the loan is recovered by the bank with little loss. Additionally, this application gets rid of intermediaries. With the use of this app, everyone in the nation will be able to contact a bank, and users in remote areas can access the bank's loan approval procedure.

5.	Business Model (Revenue Model)	While providing good performance and yielding effective results, this application can be implemented with minimum cost. A pay per month use plan for the model is an option. Employees of the bank can purchase a subscription on a monthly or annual basis. Selling the model to the bank that pays the sum that is most advantageous to developers is an additional choice
6.	Scalability of the Solution	This application's front end is developed using the Python Web Framework, while the bank end makes use of a flask integration. Consequently, it is simple to add new features. The application is hence scalable.

3.4 PROPOSED SOLUTION FIT:

CHAPTER 4

REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENTS:

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through entering details such as name, email ,password.
FR-2	User Login	Login using the registered email id and password.
FR-3	Model Building	Build various machine learning model to predict Applicant Credibility and compare them.
FR-4	Check Details	Get the user details and display if the user has credibility for loan approval or not.
FR-5	Integration	Integrate the front end and the developed ML model using Flask.
FR-6	Alert Message	Notify the user through email or phone regarding the loan approval.

4.2 NON-FUNCTIONAL REQUIREMENTS:

Following are the non-functional requirements of the proposed solution.

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	Any valid user data must be accepted for prediction.
NFR-2	Security	Alert message must be sent to the users.
NFR-3	Reliability	Loan approval of applicant must be predicted accurately and the result must be reliable.
NFR-4	Performance	The performance and interface must be user friendly.
NFR-5	Availability	Anyone who has the valid bank account.
NFR-6	Scalability	It must be able to handle increase in the number of users.

CHAPTER 5

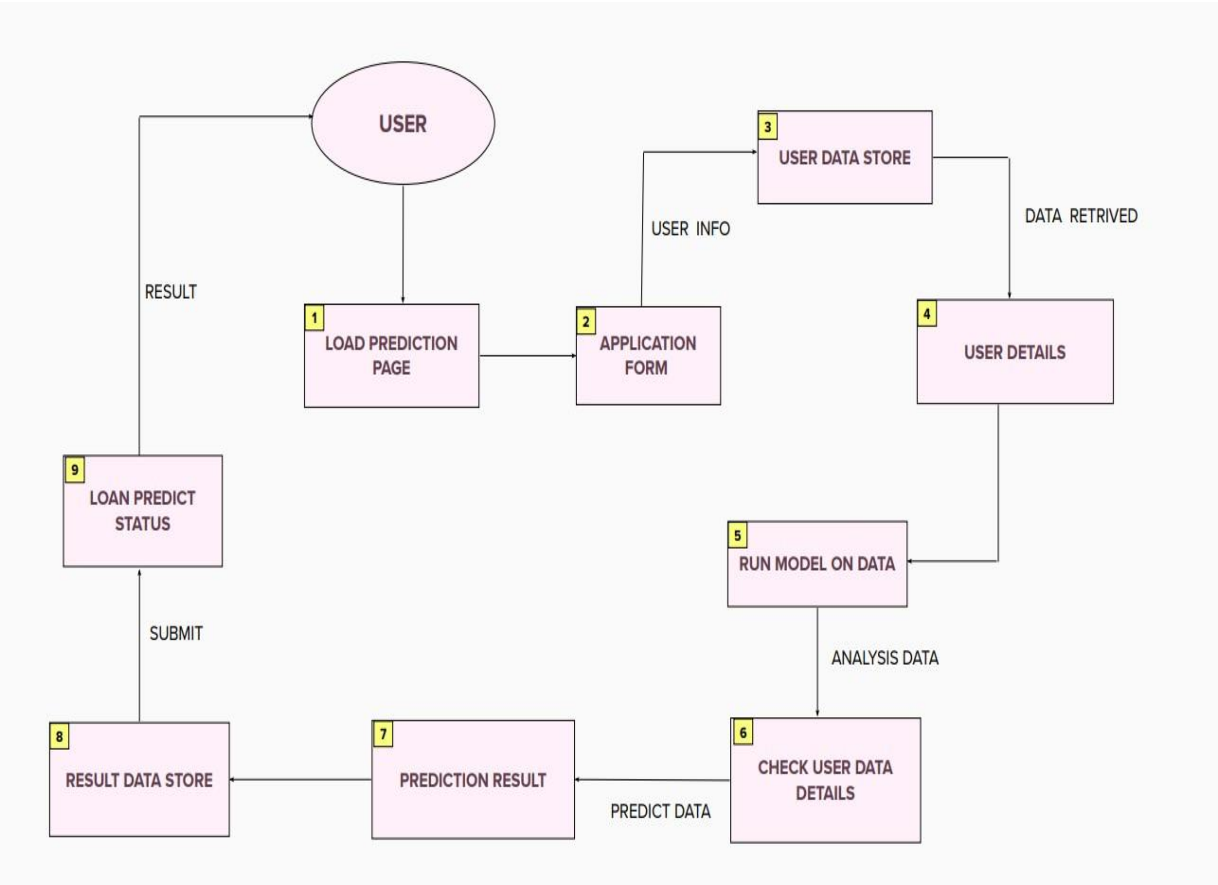
PROJECT DESIGN

5.1 Data Flow Diagram

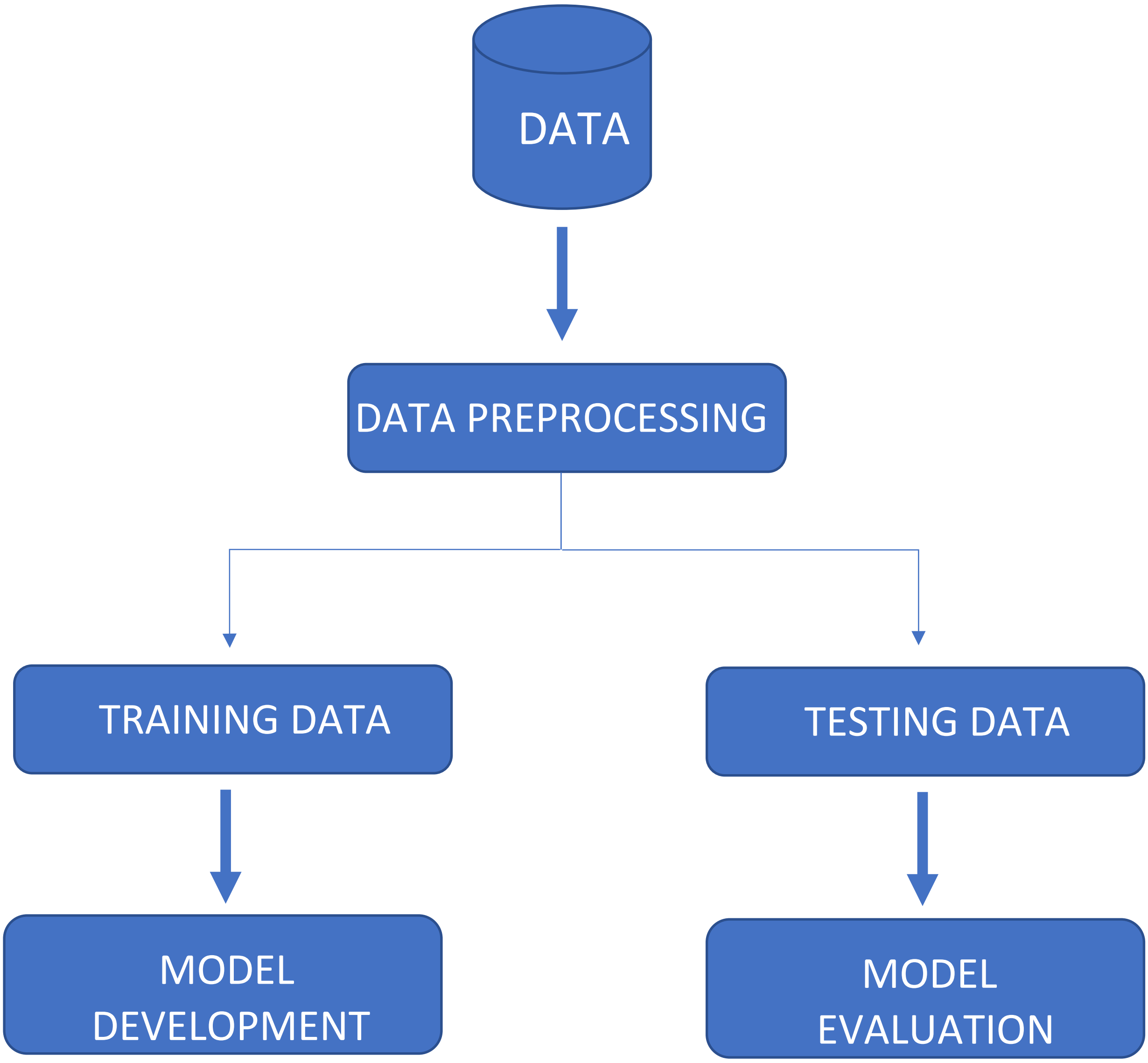
A data flow diagram is a visualization tool used to illustrate the flow of processes in a company or a specific project within it. It highlights the movement of information as well as the sequence of steps or events required to complete a work task.

DFDs can vary in design and complexity, depending on the process it represents. It can be a simple outline of a general system or a more granular sketch of a multi-level procedure.

Data Flow Diagram Level 0:



5.2 Solution & Technical Architecture



5.3 User Stories

User Type	Functional Requirement	User Story Number	User Story /Task	User Number Story	Acceptance Criteria	Priority	Release
Customer (Web User)	Registration	USN-1	As a user, I can register using name, email, and password	USN-1	I can register	High	Sprint-1
	Login	USN-2	As a user, I can login successfully	USN-2	I can access the application form	High	Sprint -2
	Application form	USN - 3	As a user, I can access the application form and can enter the details	USN-3	I can view the application form	High	Sprint-3
	Application Status	USN - 4	As a user, I get my loan prediction status	USN-4	Application status can be viewed	High	Sprint-4

CHAPTER 6

PROJECT PLANNING & SCHEDULING

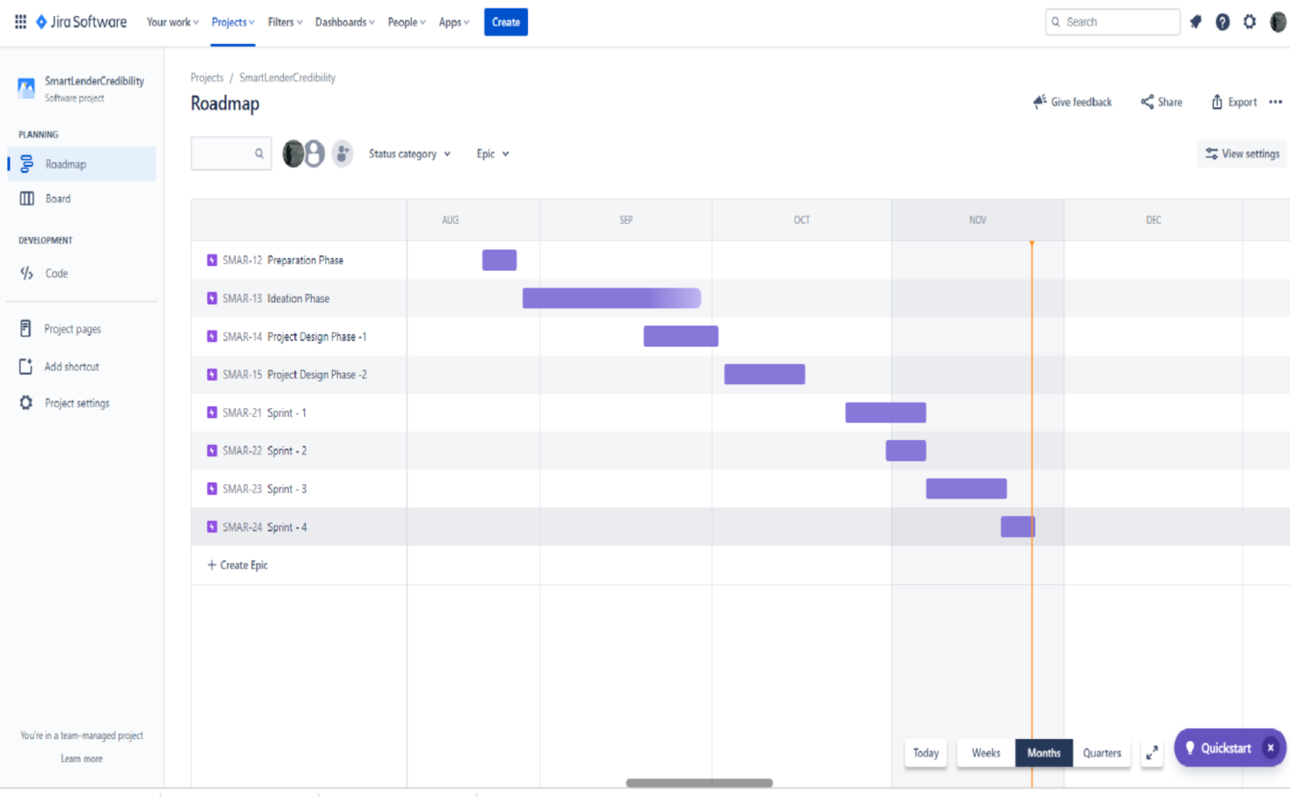
6.1 Sprint Planning

Sprint	**Functional Requirement (Epic)**	**User Story Number**	**User Story / Task**	**Story Points**	**Priority**	**Team Members**
Sprint-1	Registration	USN-1	As a user, I can register using name, email, and password	2	High	Bhuvaneshwari Harshitha Serin Banu Sona Bavya Dharshini
Sprint-2	Login	USN-2	As a user, I can login successfully	1	High	Bhuvaneshwari Harshitha Serin Banu Sona Bavya Dharshini
Sprint-3	Application form	USN-3	As a user, I can access the application form and can enter the details	2	Low	Bhuvaneshwari Harshitha Serin Banu Sona Bavya Dharshini
Sprint-4	Application status	USN-4	As a user, I get my loan prediction status	2	Medium	Bhuvaneshwari Harshitha Serin Banu Sona Bavya Dharshini

6.2 Sprint Delivering Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	29 Oct 2022	04 Nov 2022	20	06 Nov 2022
Sprint-2	20	6 Days	31 Oct 2022	06 Nov 2022	20	06 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	18 Nov 2022	20	20 Nov 2022
Sprint-4	20	6 Days	20 Nov 2022	25 Nov 2022	20	25 Nov 2022

6.3 Reports from jira



CHAPTER 7

CODING & SOLUTIONING

7.1 Machine Learning

DATA DESCRIPTION:

```
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
```

```
df.mean()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarnin
g: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=N
one') is deprecated; in a future version this will raise TypeError.  Select o
nly valid columns before calling the reduction.
  """Entry point for launching an IPython kernel.
```

```
ApplicantIncome      5403.459283
CoapplicantIncome    1621.245798
LoanAmount            146.412162
Loan_Amount_Term      342.000000
Credit_History          0.842199
dtype: float64
```

```
df.median()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarnin
g: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=N
one') is deprecated; in a future version this will raise TypeError.  Select o
nly valid columns before calling the reduction.
  """Entry point for launching an IPython kernel.
```

```
ApplicantIncome      3812.5
CoapplicantIncome    1188.5
LoanAmount            128.0
Loan_Amount_Term      360.0
Credit_History          1.0
dtype: float64
```

```
df.mode()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coap
0	LP001002	Male	Yes	0	Graduate	No	2500.0	
1	LP001003	NaN	NaN	NaN	NaN	NaN	NaN	
2	LP001005	NaN	NaN	NaN	NaN	NaN	NaN	
3	LP001006	NaN	NaN	NaN	NaN	NaN	NaN	
4	LP001008	NaN	NaN	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	
609	LP002978	NaN	NaN	NaN	NaN	NaN	NaN	
610	LP002979	NaN	NaN	NaN	NaN	NaN	NaN	
611	LP002983	NaN	NaN	NaN	NaN	NaN	NaN	
612	LP002984	NaN	NaN	NaN	NaN	NaN	NaN	
613	LP002990	NaN	NaN	NaN	NaN	NaN	NaN	

614 rows × 13 columns

```
df.std()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarnin
g: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=N
one') is deprecated; in a future version this will raise TypeError.  Select o
nly valid columns before calling the reduction.
  """Entry point for launching an IPython kernel.
```

```
ApplicantIncome      6109.041673
CoapplicantIncome    2926.248369
LoanAmount             85.587325
Loan_Amount_Term       65.120410
Credit_History          0.364878
dtype: float64
```

```
df.var()
```

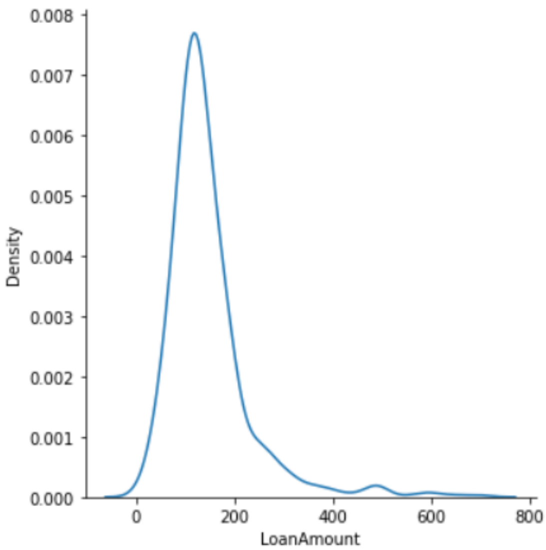
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarnin
g: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=N
one') is deprecated; in a future version this will raise TypeError.  Select o
nly valid columns before calling the reduction.
  """Entry point for launching an IPython kernel.
```

```
ApplicantIncome      3.732039e+07
CoapplicantIncome    8.562930e+06
LoanAmount           7.325190e+03
Loan_Amount_Term     4.240668e+03
Credit_History       1.331362e-01
dtype: float64
```

VISUALIZATION:

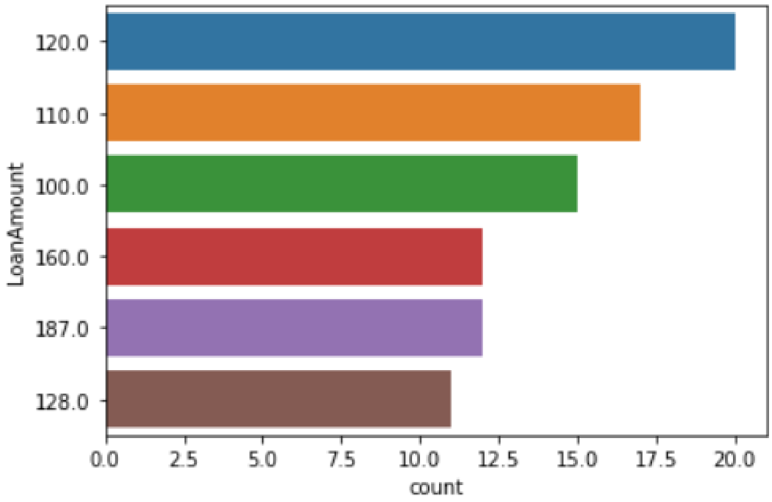
```
print(sns.displot(df['LoanAmount'], kind ='kde'))
```

```
<seaborn.axisgrid.FacetGrid object at 0x7f37f5c1cf10>
```



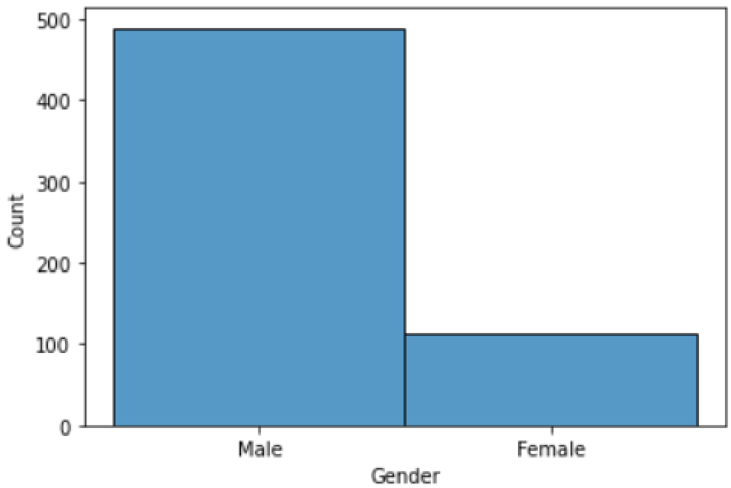
```
sns.countplot(y=df['LoanAmount'],order=df['LoanAmount'].value_counts().head(6)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f37f53db210>

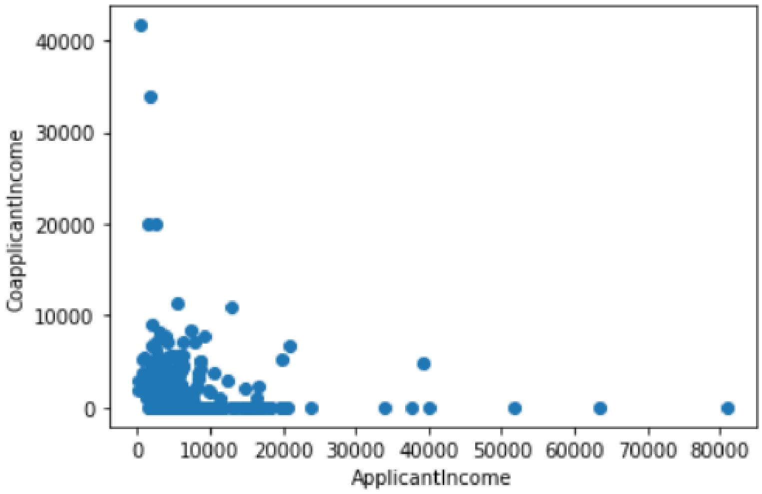



```
sns.histplot(df['Gender'])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f37f265f650>

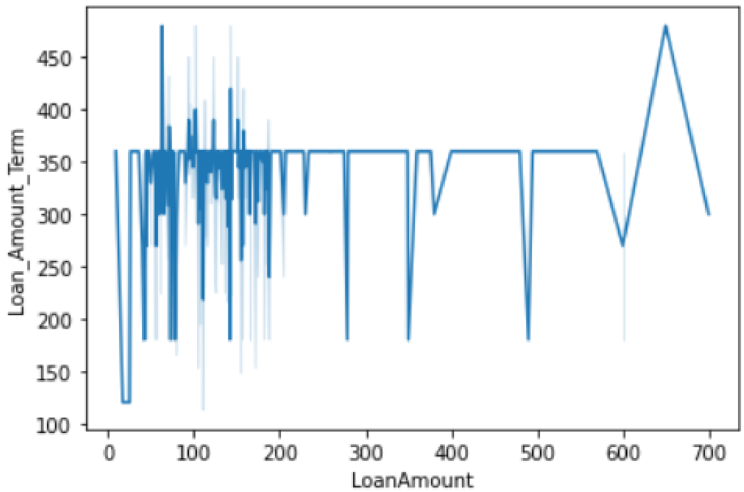


```
plt.scatter(df['ApplicantIncome'],df['CoapplicantIncome'])
plt.xlabel('ApplicantIncome')
plt.ylabel('CoapplicantIncome')
plt.show()
```



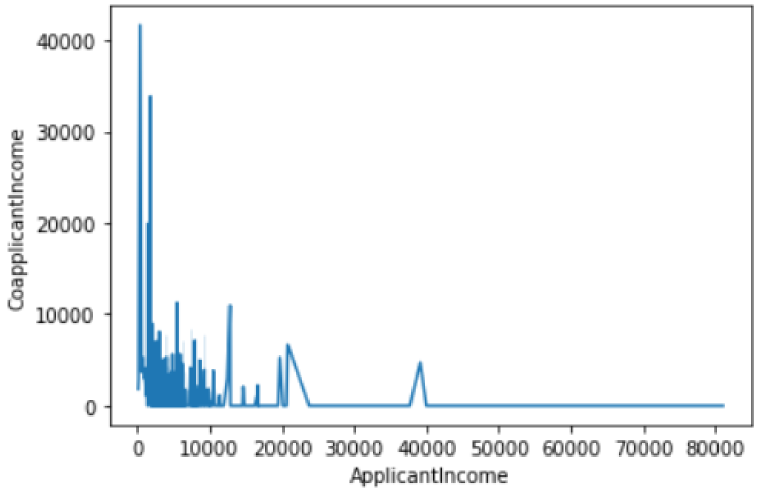

```
sns.lineplot(df['LoanAmount'],df['Loan_Amount_Term'])
plt.xlabel('LoanAmount')
plt.ylabel('Loan_Amount_Term')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarni
ng: Pass the following variables as keyword args: x, y. From version 0.12, th
e only valid positional argument will be `data`, and passing other arguments
without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



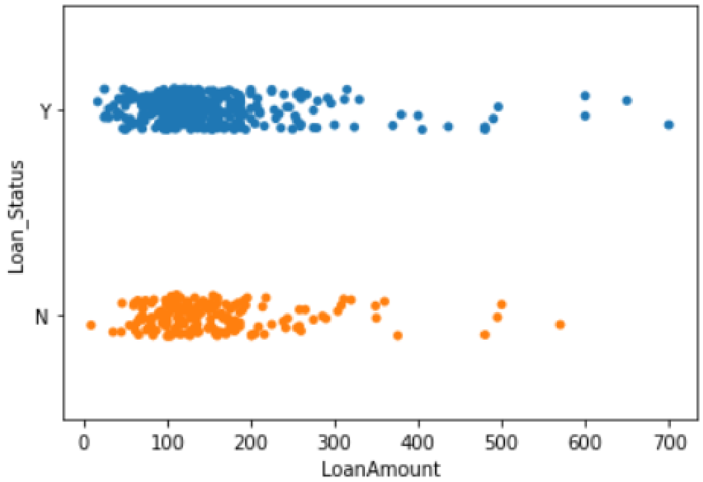
```
sns.lineplot(df['ApplicantIncome'],df['CoapplicantIncome'])
plt.xlabel('ApplicantIncome')
plt.ylabel('CoapplicantIncome')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarni
ng: Pass the following variables as keyword args: x, y. From version 0.12, th
e only valid positional argument will be `data`, and passing other arguments
without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



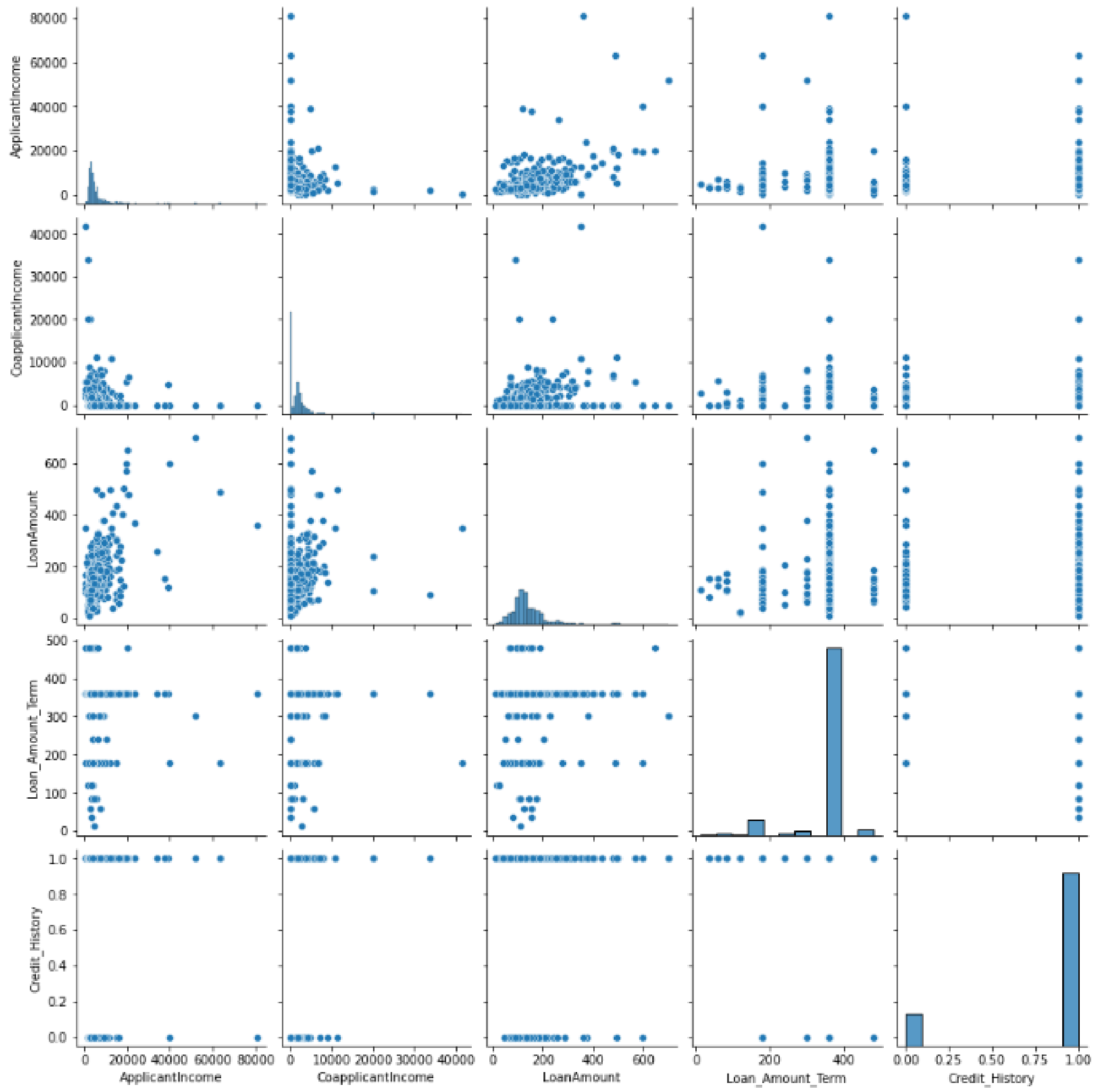

```
sns.stripplot(x=df["LoanAmount"],y=df["Loan_Status"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f37f23fc190>
```



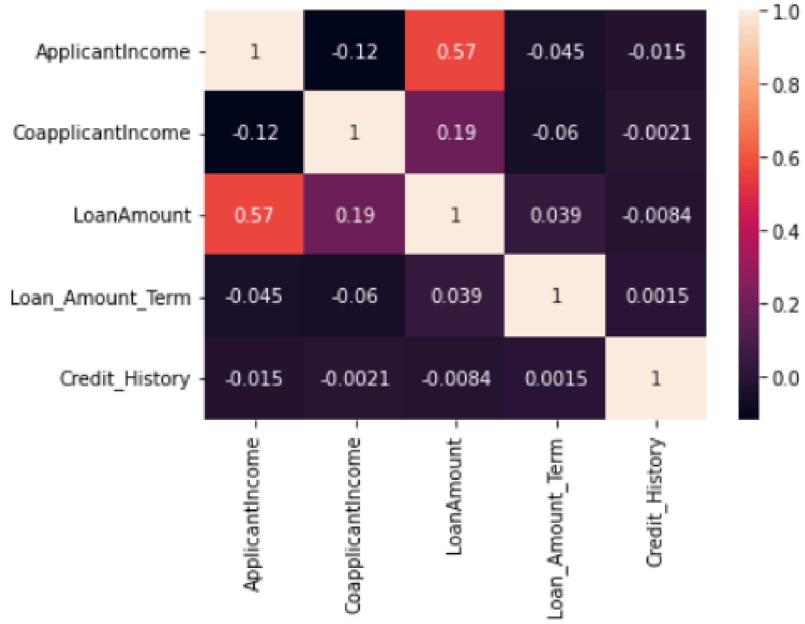
```
sns.pairplot(df)
```

<seaborn.axisgrid.PairGrid at 0x7f37f53bc990>



```
sns.heatmap(df.corr(),annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f37f07929d0>
```



TREATING NULL VALUES:

```
df.isnull().sum().sort_values(ascending=False)
```

```
Credit_History       50
Self_Employed        32
LoanAmount           22
Dependents           15
Loan_Amount_Term     14
Gender               13
Married               3
Loan_ID               0
Education             0
ApplicantIncome       0
CoapplicantIncome     0
Property_Area         0
Loan_Status           0
dtype: int64
```

```python
null_cols = ['Credit_History', 'Self_Employed', 'LoanAmount',
             'Dependents', 'Loan_Amount_Term', 'Gender', 'Married']
for col in null_cols:
    print(f"{col}:\n{df[col].value_counts()}\n","-"*50)
    df[col] = df[col].fillna(
    df[col].dropna().mode().values[0] )
print(f"After filling null values\n",'#'*50)
df.isnull().sum().sort_values(ascending=False)
```

```
Credit_History:
1.0    475
0.0     89
Name: Credit_History, dtype: int64
 --------------------------------------------------
Self_Employed:
No     500
Yes     82
Name: Self_Employed, dtype: int64
 --------------------------------------------------
LoanAmount:
120.0    20
110.0    17
100.0    15
160.0    12
187.0    12
         ..
240.0     1
214.0     1
59.0      1
166.0     1
253.0     1
Name: LoanAmount, Length: 203, dtype: int64
 --------------------------------------------------
Dependents:
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
 --------------------------------------------------
Loan_Amount_Term:
360.0    512
180.0     44
480.0     15
300.0     13
240.0      4
84.0       4
120.0      3
60.0       2
36.0       2
12.0       1
Name: Loan_Amount_Term, dtype: int64
 --------------------------------------------------
Gender:
Male      489
Female    112
Name: Gender, dtype: int64
 --------------------------------------------------
Married:
Yes    398
No     213
Name: Married, dtype: int64
 --------------------------------------------------
After filling null values
 ##################################################

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
```

ENCODING:

```
le=LabelEncoder()

df['Gender']=le.fit_transform(df['Gender'])
df['Married']=le.fit_transform(df['Married'])
df['Dependents']=le.fit_transform(df['Dependents'])
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'])
```

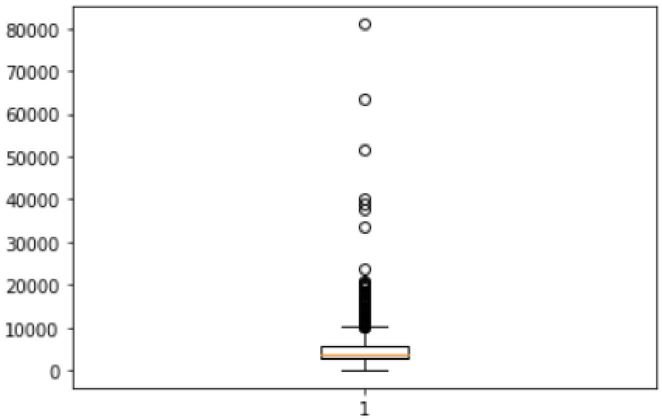
```
numeric_data = df.select_dtypes(include=[np.number])
categorical_data = df.select_dtypes(exclude=[np.number])
print("Number of numerical variables: ", numeric_data.shape[1])
print("Number of categorical variables: ", categorical_data.shape[1])
```

```
Number of numerical variables:  12
Number of categorical variables:  1
```

TREATING OUTLIERS:

```
plt.boxplot(df["ApplicantIncome"])
```

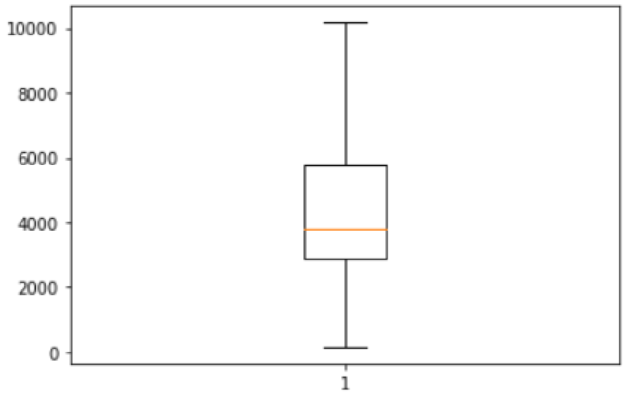
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f37ee4dcd50>,
  <matplotlib.lines.Line2D at 0x7f37ee4e12d0>],
 'caps': [<matplotlib.lines.Line2D at 0x7f37ee4e1810>,
  <matplotlib.lines.Line2D at 0x7f37ee4e1d50>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f37ee4dc810>],
 'medians': [<matplotlib.lines.Line2D at 0x7f37ee4e9310>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f37ee4e9850>],
 'means': []}
```



```
Quan1 = df['ApplicantIncome'].quantile(0.25)
Quan3 = df['ApplicantIncome'].quantile(0.75)
IQR = Quan3-Quan1
lower_limit = Quan1 - (1.5*IQR)
upper_limit = Quan3 + (1.5*IQR)
df["ApplicantIncome"] = np.where(df["ApplicantIncome"]> upper_limit,
                                 upper_limit,
                        np.where(df["ApplicantIncome"]< lower_limit,
                                 lower_limit,
                          df["ApplicantIncome"]))
```

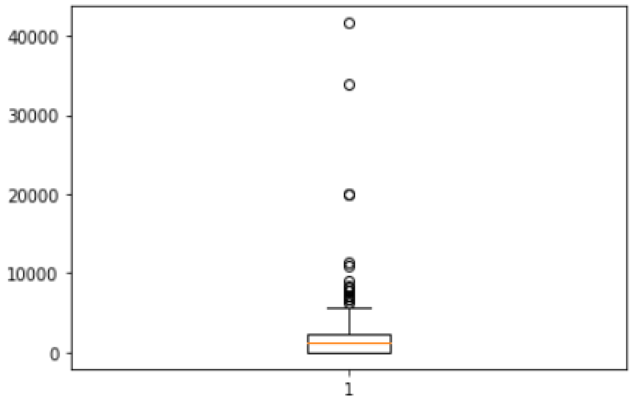
```
plt.boxplot(df["ApplicantIncome"])
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f37ee4d8310>,
  <matplotlib.lines.Line2D at 0x7f37ee4d8850>],
 'caps': [<matplotlib.lines.Line2D at 0x7f37ee4d8d50>,
  <matplotlib.lines.Line2D at 0x7f37ee45e2d0>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f37ee4d1e10>],
 'medians': [<matplotlib.lines.Line2D at 0x7f37ee45e850>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f37ee45ed90>],
 'means': []}
```




```
plt.boxplot(df["CoapplicantIncome"])
```

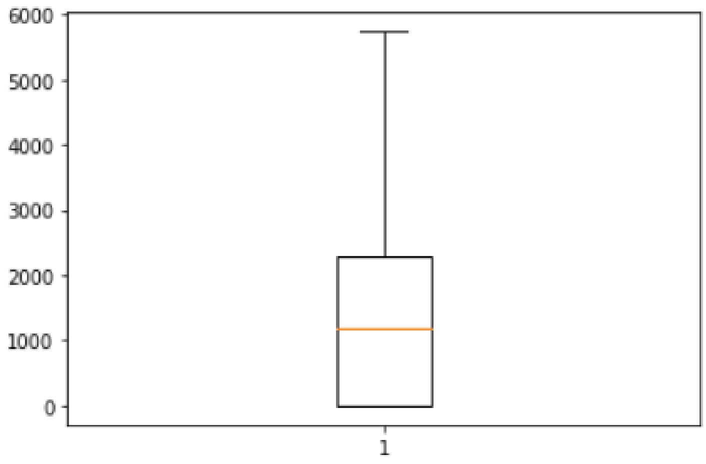
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f37ee443ed0>,
  <matplotlib.lines.Line2D at 0x7f37ee448450>],
 'caps': [<matplotlib.lines.Line2D at 0x7f37ee448990>,
  <matplotlib.lines.Line2D at 0x7f37ee448ed0>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f37ee443a10>],
 'medians': [<matplotlib.lines.Line2D at 0x7f37ee451490>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f37ee4519d0>],
 'means': []}
```



```python
Quan1 = df['CoapplicantIncome'].quantile(0.25)
Quan3 = df['CoapplicantIncome'].quantile(0.75)
IQR = Quan3-Quan1
lower_limit = Quan1 - (1.5*IQR)
upper_limit = Quan3 + (1.5*IQR)
df["CoapplicantIncome"] = np.where(df["CoapplicantIncome"]>
                                   upper_limit, upper_limit,
                        np.where(df["CoapplicantIncome"]< lower_limit,
                                 lower_limit,
                          df["CoapplicantIncome"]))
```

```python
plt.boxplot(df["CoapplicantIncome"])
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f37ee3bf3d0>,
  <matplotlib.lines.Line2D at 0x7f37ee3bf910>],
 'caps': [<matplotlib.lines.Line2D at 0x7f37ee3bfe50>,
  <matplotlib.lines.Line2D at 0x7f37ee3c73d0>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f37ee3b7ed0>],
 'medians': [<matplotlib.lines.Line2D at 0x7f37ee3c7950>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f37ee3c7e90>],
 'means': []}
```



BUSINESS LOAN PREDICTION:

SPLITTING DEPENDENT AND INDEPENDENT VARIABLES:

```python
X = df.drop("Loan_ID",axis=1)
X = X.drop("Loan_Status",axis=1)
Y = df["Loan_Status"]
```

```python
print(X)
```

```
     Gender  Married  Dependents  Education  Self_Employed  ApplicantIncome  \
0         1        0           0          0              0           5849.0   
1         1        1           1          0              0           4583.0   
2         1        1           0          0              1           3000.0   
3         1        1           0          1              0           2583.0   
4         1        0           0          0              0           6000.0   
..      ...      ...         ...        ...            ...              ...   
609       0        0           0          0              0           2900.0   
610       1        1           3          0              0           4106.0   
611       1        1           1          0              0           8072.0   
612       1        1           2          0              0           7583.0   
613       0        0           0          0              1           4583.0   

     CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
0                  0.0       120.0             360.0             1.0   
1               1508.0       128.0             360.0             1.0   
2                  0.0        66.0             360.0             1.0   
3               2358.0       120.0             360.0             1.0   
4                  0.0       141.0             360.0             1.0   
..                 ...         ...               ...             ...   
609                0.0        71.0             360.0             1.0   
610                0.0        40.0             180.0             1.0   
611              240.0       253.0             360.0             1.0   
612                0.0       187.0             360.0             1.0   
613                0.0       133.0             360.0             0.0   

     Property_Area  
0                2  
1                0  
2                2  
3                2  
4                2  
..             ...  
609              0  
610              0  
611              2  
612              2  
613              1  

[614 rows x 11 columns]
```

```python
print(Y)
```

```
0      1
1      0
2      1
3      1
4      1
      ..
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status, Length: 614, dtype: int64
```

SCALING:

```
from sklearn.preprocessing import StandardScaler
```

```
scale = StandardScaler()
st_scale = scale.fit_transform(X)
st_scale
```

```
array([[ 0.49716393, -0.87458735, -0.30275919,  0.2732313 ,  0.41173269,
         1.22329839],
       [-0.0137667 ,  0.05439458, -0.20764834,  0.2732313 ,  0.41173269,
        -1.31851281],
       [-0.65263178, -0.87458735, -0.94475737,  0.2732313 ,  0.41173269,
         1.22329839],
       ...,
       [ 1.39431937, -0.72673876,  1.27845856,  0.2732313 ,  0.41173269,
         1.22329839],
       [ 1.19696939, -0.87458735,  0.49379411,  0.2732313 ,  0.41173269,
         1.22329839],
       [-0.0137667 , -0.87458735, -0.14820407,  0.2732313 , -2.42876026,
        -0.04760721]])
```

```
X_scaled  = pd.DataFrame(st_scale, columns = X.columns)
```

SPLITTING THE DATASET:

```
from sklearn.model_selection import train_test_split
train_X,test_X,train_y,test_y = train_test_split(X_scaled, Y,
                    test_size = 0.2, random_state = 0)
```

```
print("Shape of Training X :",train_X.shape)
print("Shape of Validation X :",test_X.shape)
```

```
Shape of Training X : (491, 6)
Shape of Validation X : (123, 6)
```

```
print("Shape of Training y :",train_y.shape)
print("Shape of Validation y :",test_y.shape)
```

```
Shape of Training y : (491,)
Shape of Validation y : (123,)
```

MODELS:

LOGISTIC REGRESSION:

```
# normal log reg model

logmodel = LogisticRegression()
logmodel.fit(train_X , train_y)
# testig accuracy
pred_t = logmodel.predict(train_X)
acc_t = accuracy_score(train_y , pred_t)*100
print("training accuracy is ",acc_t)
pred_l = logmodel.predict(test_X)
acc_l = accuracy_score(test_y , pred_l)*100
print("testing accuracy is ",acc_l)
# acc_l
```

```
training accuracy is  80.41958041958041
testing accuracy is  82.16216216216216
```

RANDOM FOREST:

```
# random forest improved
random_forest_imp = RandomForestClassifier(n_estimators= 800,max_depth = 5,
criterion = "entropy", oob_score = True, verbose = 1,n_jobs = -1,random_state =1
random_forest_imp.fit(train_X, train_y)
pred_rft = random_forest_imp.predict(train_X)
acc_rft = accuracy_score(train_y , pred_rft)*100
print("training accuracy is ",acc_rft)
pred_rfl = random_forest_imp.predict(test_X)
acc_rfl = accuracy_score(test_y , pred_rfl)*100
print("testing accuracy is ",acc_rfl)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 2 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 196 tasks      | elapsed:    0.6s
[Parallel(n_jobs=-1)]: Done 446 tasks      | elapsed:    1.1s
[Parallel(n_jobs=-1)]: Done 796 tasks      | elapsed:    1.8s
[Parallel(n_jobs=-1)]: Done 800 out of 800 | elapsed:    1.8s finished
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.0s
[Parallel(n_jobs=2)]: Done 196 tasks      | elapsed:    0.1s
[Parallel(n_jobs=2)]: Done 446 tasks      | elapsed:    0.2s
[Parallel(n_jobs=2)]: Done 796 tasks      | elapsed:    0.3s
[Parallel(n_jobs=2)]: Done 800 out of 800 | elapsed:    0.3s finished
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.0s
[Parallel(n_jobs=2)]: Done 196 tasks      | elapsed:    0.0s
[Parallel(n_jobs=2)]: Done 446 tasks      | elapsed:    0.1s
```

```
training accuracy is  80.88578088578089
testing accuracy is  82.16216216216216
```

```
# random forest
random_forest = RandomForestClassifier(n_estimators= 1000)
random_forest.fit(train_X, train_y)
pred_rf = random_forest.predict(test_X)
acc_rf = accuracy_score(test_y , pred_rf)*100
acc_rf
```

```
79.67479674796748
```

KNN:

```
# knn
knn = KNeighborsClassifier(n_neighbors = 12)
knn.fit(train_X, train_y)
# pred_knn = knn.predict(test_X)
# acc_knn = accuracy_score(test_y , pred_knn)*100
# acc_knn
pred_knnt = knn.predict(train_X)
acc_knnt = accuracy_score(train_y , pred_knnt)*100
print("training accuracy is ",acc_knnt)
pred_knnl = knn.predict(test_X)
acc_knnl = accuracy_score(test_y , pred_knnl)*100
print("testing accuracy is ",acc_knnl)
```

```
training accuracy is  80.1864801864802
testing accuracy is  81.08108108108108
```

XGBOOST:

```
# xgb
xgb = XGBClassifier()
xgb.fit(train_X, train_y)
pred_xgb = xgb.predict(test_X)
acc_xgb = accuracy_score(test_y , pred_xgb)*100
acc_xgb
```

```
83.73983739837398
```

```
# xgb improved
# import xgboost as xgb
# from sklearn.grid_search import GridSearchCV

xgb_model_imp = XGBClassifier()
optimization_dict = {'max_depth': [2,4,6,8,10,12,14,16,18,20,24],
                     'n_estimators': [50,100,200,300,400,500,600]}
model = GridSearchCV(xgb_model_imp, optimization_dict, scoring='accuracy',
                     verbose=1)

model.fit(train_X,train_y)
print(model.best_score_)
print(model.best_params_)
```

```
Fitting 5 folds for each of 77 candidates, totalling 385 fits
0.8041313269493843
{'max_depth': 2, 'n_estimators': 50}
```

```
pred_xt = model.predict(train_X)
acc_xt = accuracy_score(train_y , pred_xt)*100
print("training accuracy is ",acc_xt)
pred_xl = model.predict(test_X)
acc_xl = accuracy_score(test_y , pred_xl)*100
print("testing accuracy is ",acc_xl)
```

```
training accuracy is  80.41958041958041
testing accuracy is  82.16216216216216
```

DECISION TREE:

```
# decision tree
dt =DecisionTreeClassifier(max_features=6 , max_depth=15)
dt.fit(train_X,train_y)
# pred_dt = dt.predict(test_X)
# acc_dt = accuracy_score(test_y , pred_dt)*100
# acc_dt
```

```
70.73170731707317
```

```
# decision tree imp
bagging_clf_dt = BaggingClassifier(DecisionTreeClassifier(), n_estimators=250,
                    max_samples=100, bootstrap=True, random_state=101)
bagging_clf_dt.fit(train_X, train_y)
# y_pred = bagging_clf_dt.predict(test_X)
# print(accuracy_score(test_y, y_pred))
pred_dt = bagging_clf_dt.predict(train_X)
acc_dt = accuracy_score(train_y , pred_dt)*100
print("training accuracy is ",acc_dt)
pred_dl = bagging_clf_dt.predict(test_X)
acc_dl = accuracy_score(test_y , pred_dl)*100
print("testing accuracy is ",acc_dl)
```

```
training accuracy is  82.28438228438229
testing accuracy is  82.16216216216216
```

HOME LOAN PREDICTION:

SPLITTING DEPENDENT AND INDEPENDENT VARIABLES:

```
X = df.drop("Loan_ID",axis=1)
X = X.drop("CoapplicantIncome",axis=1)
X = X.drop("Loan_Status",axis=1)
Y = df["Loan_Status"]
```

```python
print(X)
```

```
     Gender  Married  Dependents  Education  Self_Employed  ApplicantIncome  \
0         1        0           0          0              0           5849.0   
1         1        1           1          0              0           4583.0   
2         1        1           0          0              1           3000.0   
3         1        1           0          1              0           2583.0   
4         1        0           0          0              0           6000.0   
..      ...      ...         ...        ...            ...              ...   
609       0        0           0          0              0           2900.0   
610       1        1           3          0              0           4106.0   
611       1        1           1          0              0           8072.0   
612       1        1           2          0              0           7583.0   
613       0        0           0          0              1           4583.0   

     LoanAmount  Loan_Amount_Term  Credit_History  Property_Area  
0         120.0             360.0             1.0              2  
1         128.0             360.0             1.0              0  
2          66.0             360.0             1.0              2  
3         120.0             360.0             1.0              2  
4         141.0             360.0             1.0              2  
..          ...               ...             ...            ...  
609        71.0             360.0             1.0              0  
610        40.0             180.0             1.0              0  
611       253.0             360.0             1.0              2  
612       187.0             360.0             1.0              2  
613       133.0             360.0             0.0              1  

[614 rows x 10 columns]
```

SCALING:

```python
from sklearn.preprocessing import StandardScaler
```

```python
scale = StandardScaler()
st_scale = scale.fit_transform(X)
st_scale
```

```
array([[ 0.47234264, -1.37208932, -0.73780632, ...,  0.2732313 ,
         0.41173269,  1.22329839],
       [ 0.47234264,  0.72881553,  0.25346957, ...,  0.2732313 ,
         0.41173269, -1.31851281],
       [ 0.47234264,  0.72881553, -0.73780632, ...,  0.2732313 ,
         0.41173269,  1.22329839],
       ...,
       [ 0.47234264,  0.72881553,  0.25346957, ...,  0.2732313 ,
         0.41173269,  1.22329839],
       [ 0.47234264,  0.72881553,  1.24474546, ...,  0.2732313 ,
         0.41173269,  1.22329839],
       [-2.11710719, -1.37208932, -0.73780632, ...,  0.2732313 ,
        -2.42876026, -0.04760721]])
```

```python
X_scaled  = pd.DataFrame(st_scale, columns = X.columns)
```

SPLITTING THE DATASET:

```
from sklearn.model_selection import train_test_split
train_X,test_X,train_y,test_y = train_test_split(X_scaled, Y, test_size = 0.3,
                                                  random_state = 0)
```

MODELS:

LOGISTIC REGRESSION:

```
# log regression
logmodel = LogisticRegression()
logmodel.fit(train_X , train_y)
# pred_l = logmodel.predict(test_X)
# acc_l = accuracy_score(test_y , pred_l)*100
# acc_l
pred_t = logmodel.predict(train_X)
acc_t = accuracy_score(train_y , pred_t)*100
print("training accuracy is ",acc_t)
pred_l = logmodel.predict(test_X)
acc_l = accuracy_score(test_y , pred_l)*100
print("testing accuracy is ",acc_l)
```

```
training accuracy is  80.1864801864802
testing accuracy is  82.70270270270271
```

RANDOM FOREST:

```
# random forest
random_forest = RandomForestClassifier(n_estimators= 100,max_depth = 3,
criterion = "entropy", oob_score = True, verbose = 1,n_jobs = -1,
                                    random_state =1)
random_forest.fit(train_X, train_y)

pred_rft = random_forest.predict(train_X)
acc_rft = accuracy_score(train_y , pred_rft)*100
print("training accuracy is ",acc_rft)
pred_rfl = random_forest.predict(test_X)
acc_rfl = accuracy_score(test_y , pred_rfl)*100
print("testing accuracy is ",acc_rfl)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 2 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:    0.3s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.5s finished
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.1s
[Parallel(n_jobs=2)]: Done 100 out of 100 | elapsed:    0.1s finished
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.1s
[Parallel(n_jobs=2)]: Done 100 out of 100 | elapsed:    0.1s finished
```

```
training accuracy is  80.1864801864802
testing accuracy is  82.70270270270271
```

KNN:

```
# knn
knn = KNeighborsClassifier(n_neighbors = 20)
knn.fit(train_X, train_y)
# pred_knn = knn.predict(test_X)
# acc_knn = accuracy_score(test_y , pred_knn)*100
# acc_knn
pred_knnt = knn.predict(train_X)
acc_knnt = accuracy_score(train_y , pred_knnt)*100
print("training accuracy is ",acc_knnt)
pred_knnl = knn.predict(test_X)
acc_knnl = accuracy_score(test_y , pred_knnl)*100
print("testing accuracy is ",acc_knnl)
```

```
training accuracy is  79.95337995337995
testing accuracy is  81.62162162162161
```

DECISION TREE:

```
# decision tree imp
bagging_clf = BaggingClassifier(DecisionTreeClassifier(), n_estimators=250,
max_samples=100, bootstrap=True, random_state=101)
bagging_clf.fit(train_X, train_y)
# y_pred = bagging_clf.predict(test_X)
# print(accuracy_score(test_y, y_pred))
pred_dt = bagging_clf.predict(train_X)
acc_dt = accuracy_score(train_y , pred_dt)*100
print("training accuracy is ",acc_dt)
pred_dl = bagging_clf.predict(test_X)
acc_dl = accuracy_score(test_y , pred_dl)*100
print("testing accuracy is ",acc_dl)
```

```
training accuracy is  83.91608391608392
testing accuracy is  79.45945945945945
```

XGBOOST:

```
# xgb improved
# import xgboost as xgb
# from sklearn.grid_search import GridSearchCV

xgb_model_imp = XGBClassifier()
optimization_dict = {'max_depth': [2,4,6,8,10,12,14,16,18,20,24],
                     'n_estimators': [50,100,200,300,400,500,600]}

model = GridSearchCV(xgb_model_imp, optimization_dict, scoring='accuracy',
                     verbose=1)

model.fit(train_X,train_y)
print(model.best_score_)
print(model.best_params_)
```

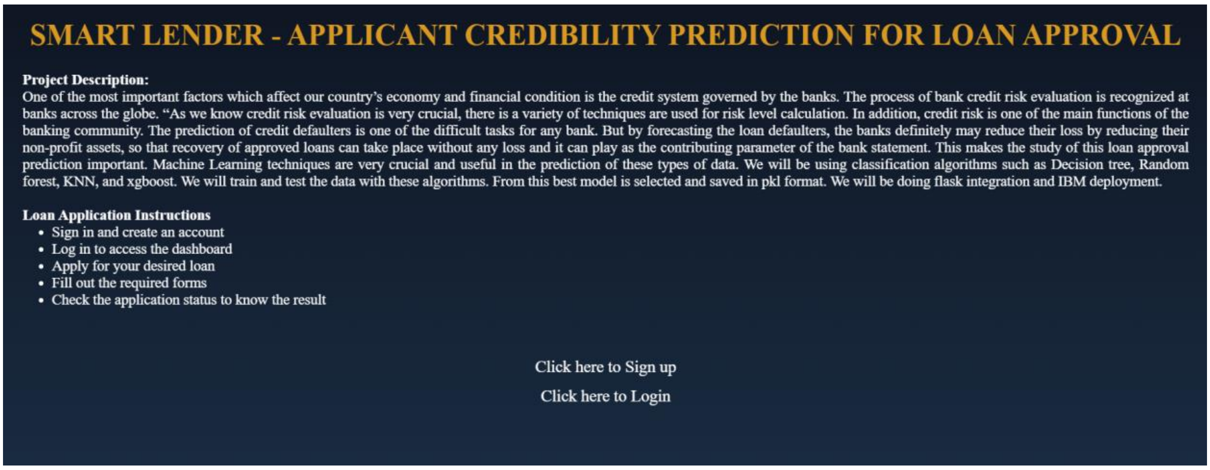
```
Fitting 5 folds for each of 77 candidates, totalling 385 fits
0.799562243502052
{'max_depth': 2, 'n_estimators': 50}
```

```python
pred_xt = model.predict(train_X)
acc_xt = accuracy_score(train_y , pred_xt)*100
print("training accuracy is ",acc_xt)
pred_xl = model.predict(test_X)
acc_xl = accuracy_score(test_y , pred_xl)*100
print("testing accuracy is ",acc_xl)
```

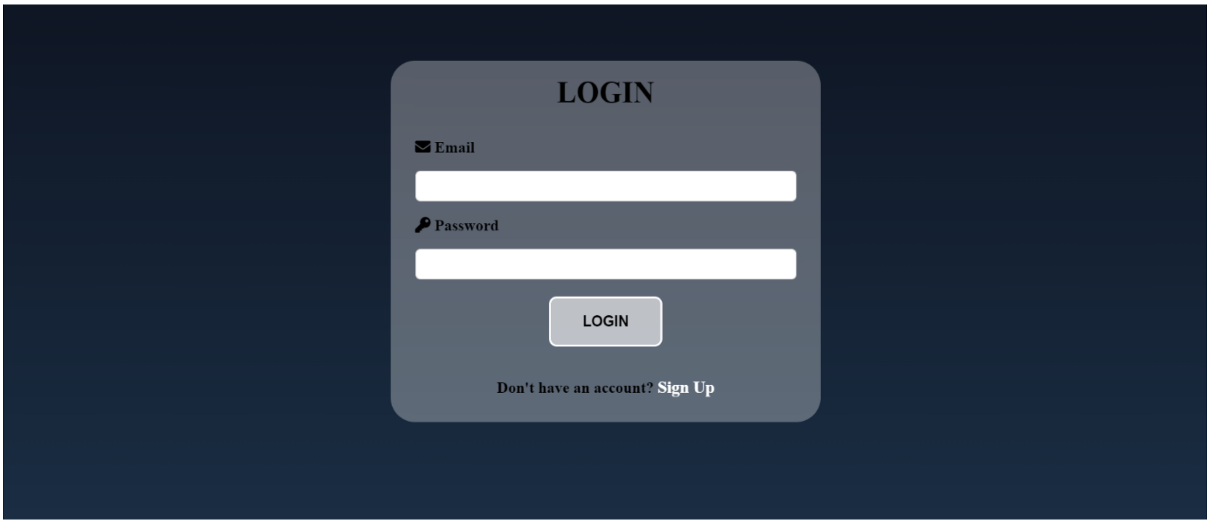
```
training accuracy is  80.88578088578089
testing accuracy is  82.16216216216216
```

7.2 WEBSITE:

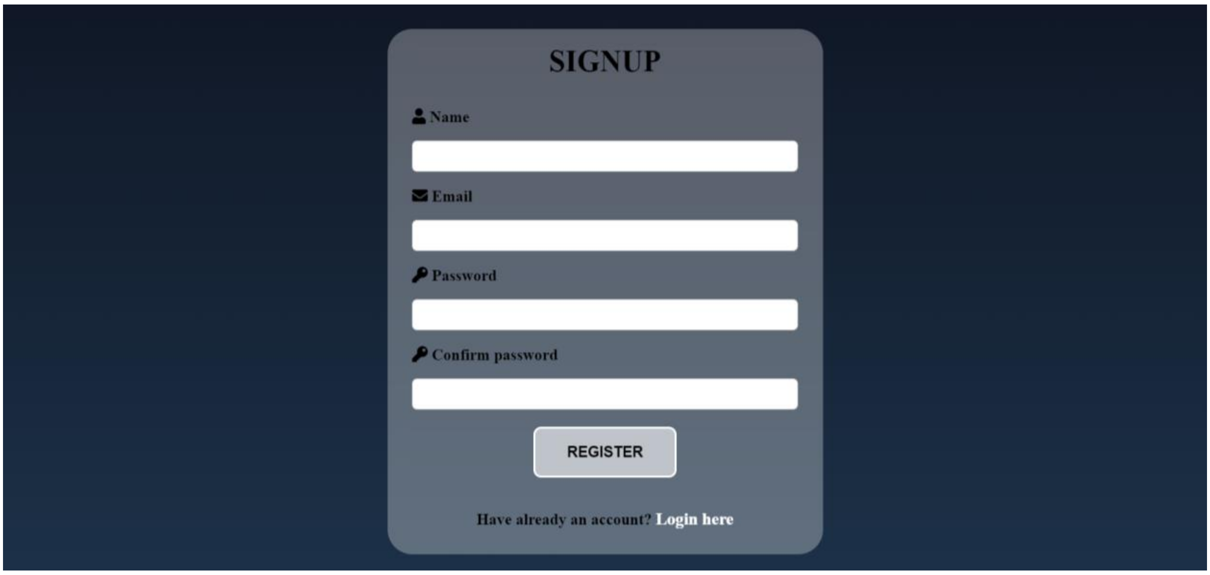
Home Page



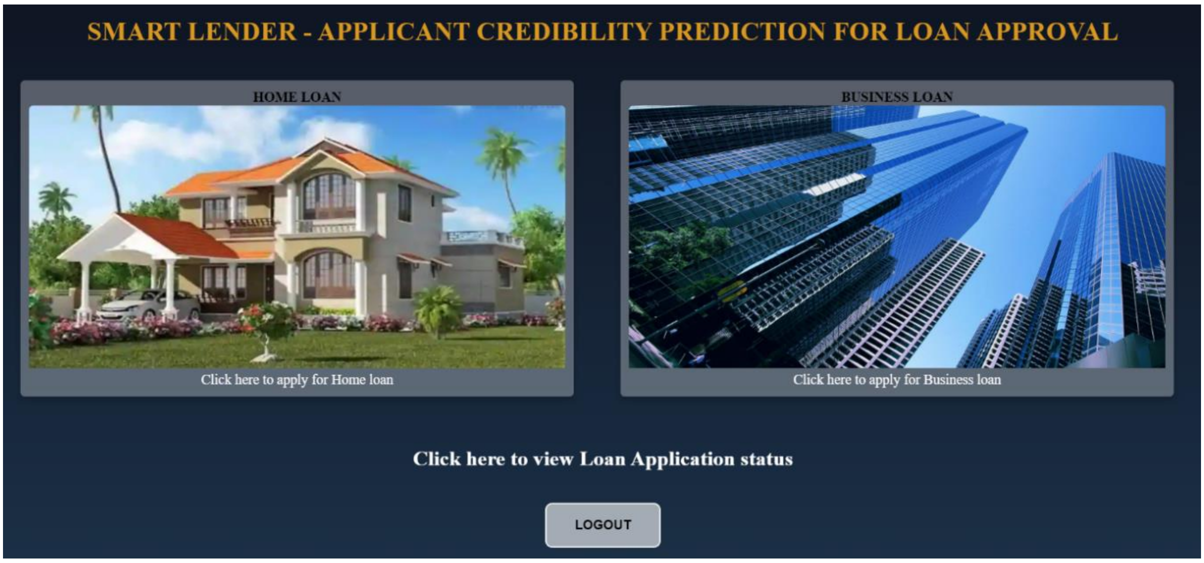
Login Page



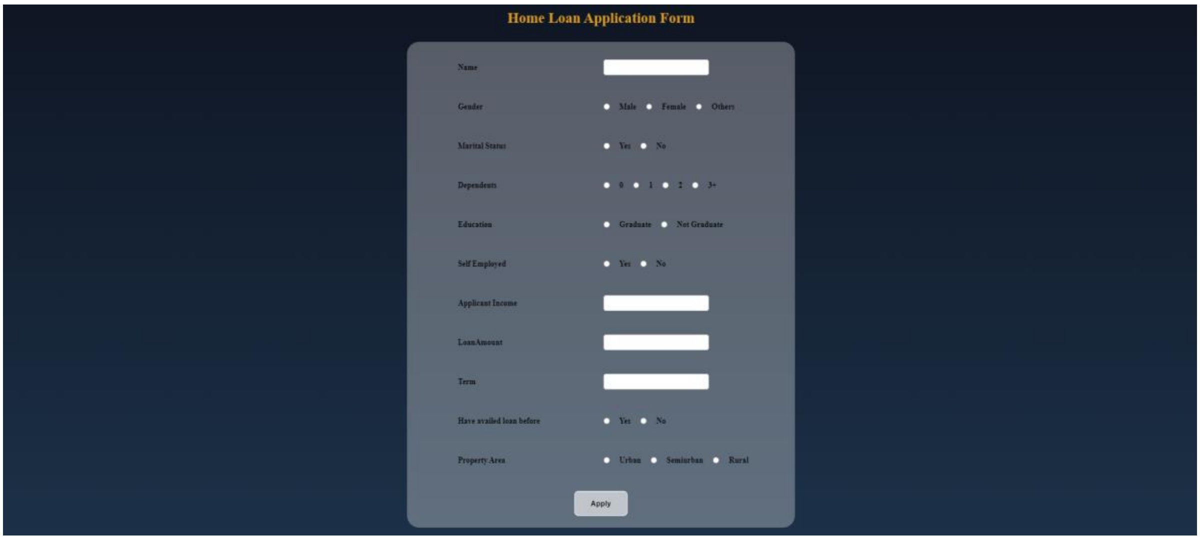
Signup Page



Dashboard

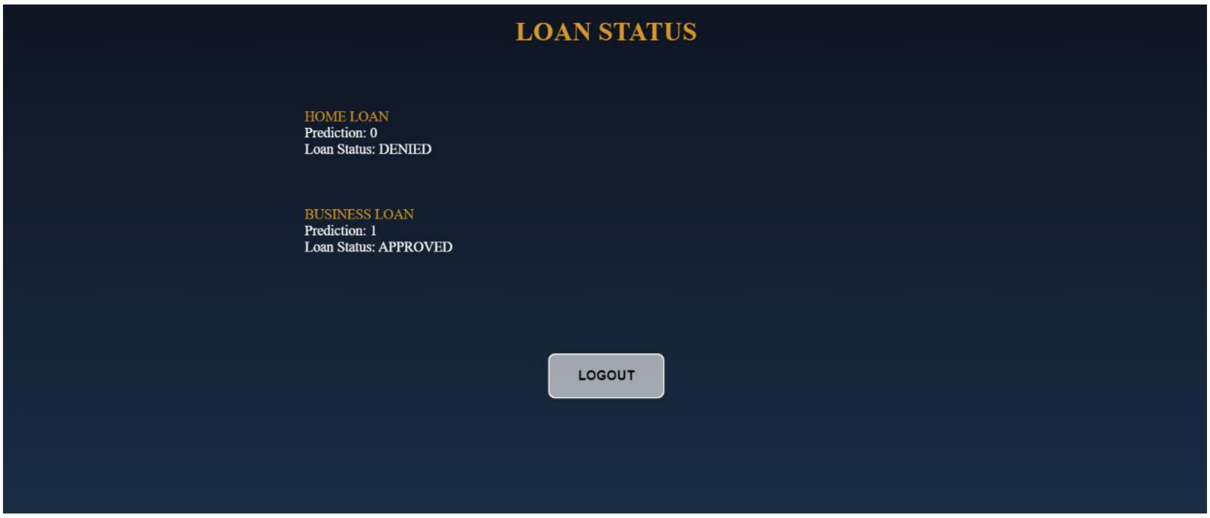


Home Loan Application Form

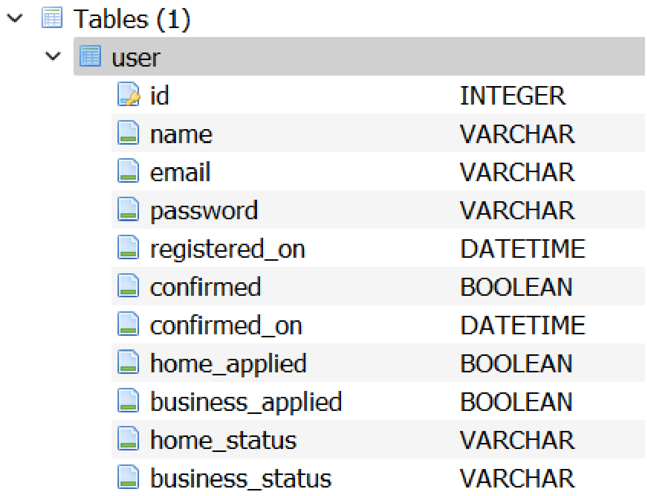


Business Loan Application Form

Loan Status



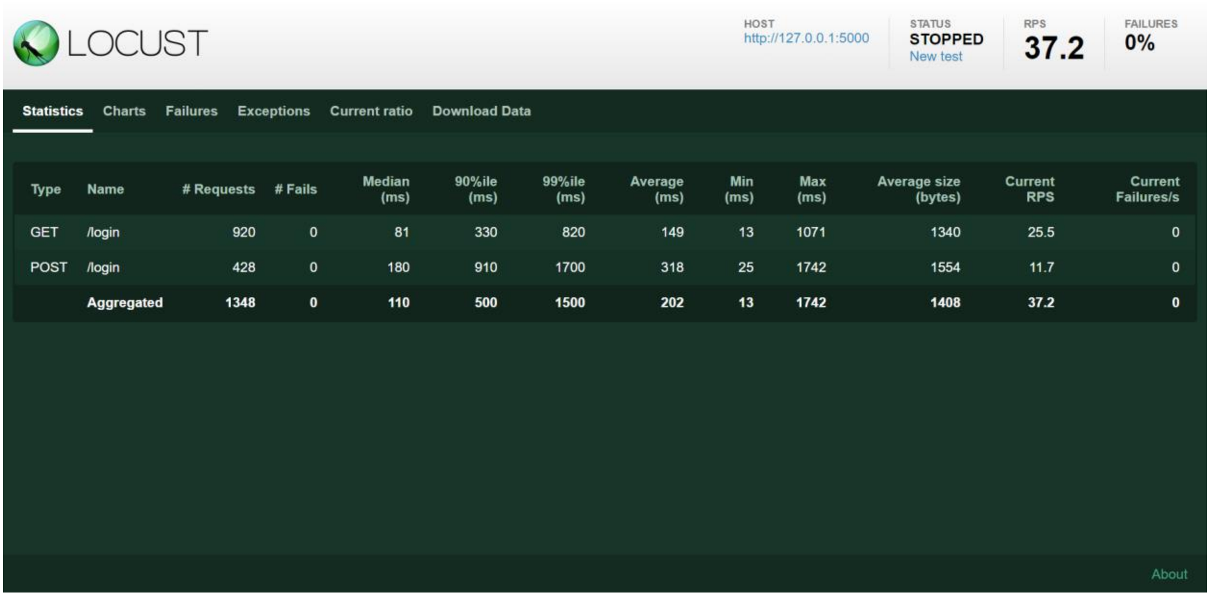
7.3 Database Schema



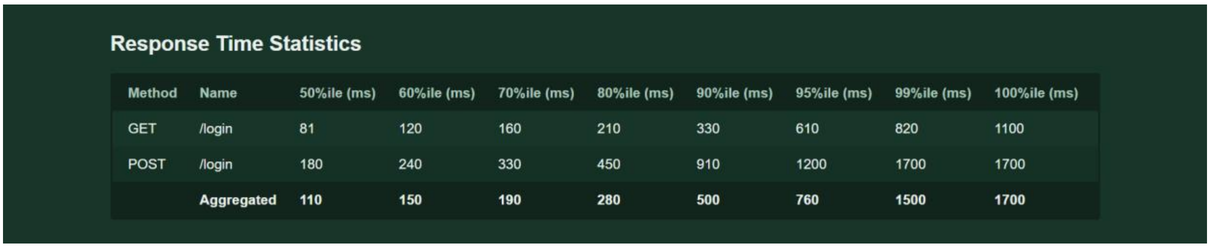
CHAPTER 8

TESTING

8.1 Locust



Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/login	920	0	81	330	820	149	13	1071	1340	25.5	0
POST	/login	428	0	180	910	1700	318	25	1742	1554	11.7	0
	Aggregated	**1348**	**0**	**110**	**500**	**1500**	**202**	**13**	**1742**	**1408**	**37.2**	**0**



Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/login	81	120	160	210	330	610	820	1100
POST	/login	180	240	330	450	910	1200	1700	1700
	Aggregated	**110**	**150**	**190**	**280**	**500**	**760**	**1500**	**1700**

8.2 User Acceptance Testing

Defect Analysis

This report shows the number of resolved or closed bugs at each severity level, and how they were resolved

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	8	3	2	2	15
Duplicate	1	0	1	0	2
External	2	2	0	0	4
Fixed	9	2	3	13	27
Not Reproduced	0	0	1	0	1
Skipped	0	0	0	1	1
Won't Fix	0	4	1	1	6
Totals	20	11	8	17	56

Test Case Analysis

This report shows the number of test cases that have passed, failed, and untested

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	9	0	0	9
Client Application	45	0	0	45
Security	2	0	0	2
Outsource Shipping	2	0	0	2
Exception Reporting	10	0	0	10
Final Report Output	3	0	0	3
Version Control	2	0	0	2

CHAPTER 9

RESULTS

9.1 Performance Metrics

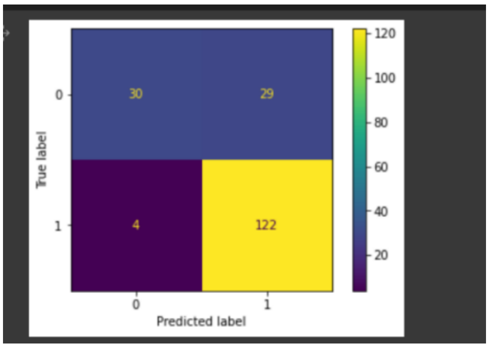
<table>
<tr><th>S.No.</th><th>Parameter</th><th>Values</th><th>Screenshot</th></tr>
<tr><td>1.</td><td>Metrics</td><td><u>BUSINESS LOAN PREDICTION</u>

LOGISTIC REGRESSION
Training Accuracy - 80.41%
Testing Accuracy - 82.16%</td><td><u>BUSINESS LOAN PREDICTION</u>

LOGISTIC REGRESSION

training accuracy is 80.41958041958041
testing accuracy is 82.16216216216216

<table>
<tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr>
<tr><td>0</td><td>0.88</td><td>0.51</td><td>0.65</td><td>59</td></tr>
<tr><td>1</td><td>0.81</td><td>0.97</td><td>0.88</td><td>126</td></tr>
<tr><td>accuracy</td><td></td><td></td><td>0.82</td><td>185</td></tr>
<tr><td>macro avg</td><td>0.85</td><td>0.74</td><td>0.76</td><td>185</td></tr>
<tr><td>weighted avg</td><td>0.83</td><td>0.82</td><td>0.81</td><td>185</td></tr>
</table>


</td></tr>
</table>

RANDOM FOREST

Training Accuracy - 80.88%

Testing Accuracy - 82.16%

KNN

Training Accuracy - 80.18%

Testing Accuracy - 81.08%

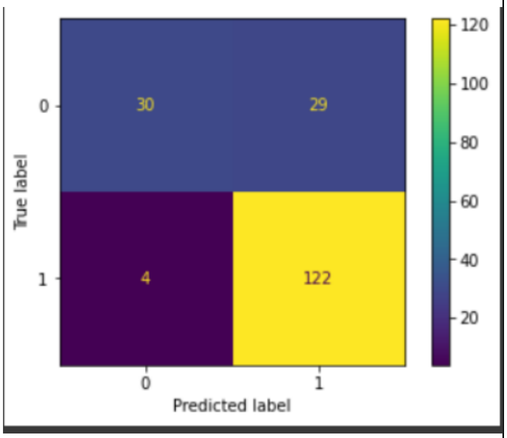
RANDOM FOREST

```
training accuracy is  80.88578088578089
testing accuracy is  82.16216216216216
```

```
              precision    recall  f1-score   support

           0       0.88      0.51      0.65        59
           1       0.81      0.97      0.88       126

    accuracy                           0.82       185
   macro avg       0.85      0.74      0.76       185
weighted avg       0.83      0.82      0.81       185
```



KNN

```
training accuracy is  80.1864801864802
testing accuracy is  81.08108108108108
```

```
              precision    recall  f1-score   support

           0       0.83      0.51      0.63        59
           1       0.81      0.95      0.87       126

    accuracy                           0.81       185
   macro avg       0.82      0.73      0.75       185
weighted avg       0.81      0.81      0.80       185
```

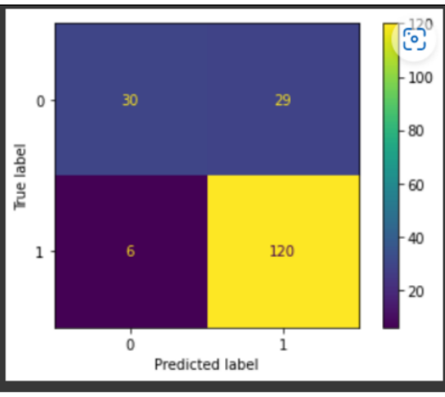
<table>
<tr>
<td></td>
<td></td>
<td>

XGB

Training Accuracy - 80.41%

Testing Accuracy - 82.16%

</td>
<td>



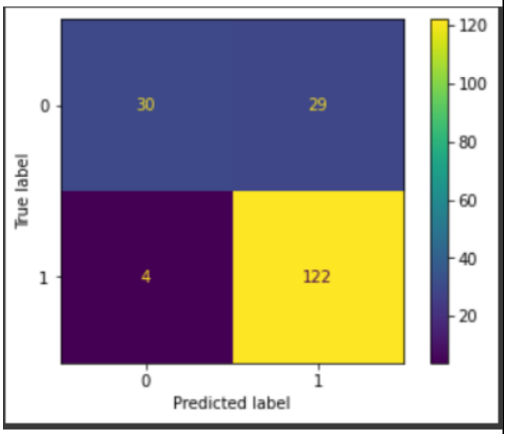
XGB

```
training accuracy is  80.41958041958041
testing accuracy is  82.16216216216216
```

```
              precision    recall  f1-score   support

           0       0.88      0.51      0.65        59
           1       0.81      0.97      0.88       126

    accuracy                           0.82       185
   macro avg       0.85      0.74      0.76       185
weighted avg       0.83      0.82      0.81       185
```



</td>
</tr>
</table>

DECISION TREE

Training Accuracy - 82.28%

Testing Accuracy - 82.16%

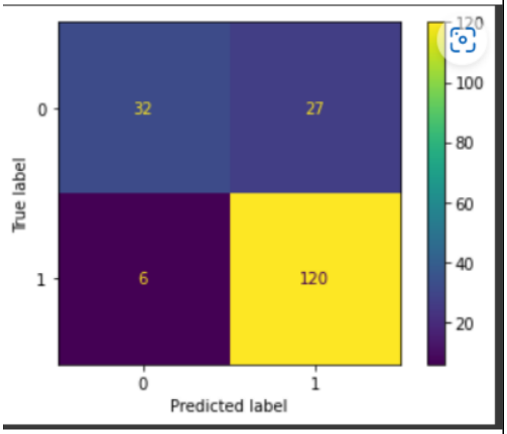
DECISION TREE

```
training accuracy is  82.28438228438229
testing accuracy is  82.16216216216216
```

```
              precision    recall  f1-score   support

           0       0.84      0.54      0.66        59
           1       0.82      0.95      0.88       126

    accuracy                           0.82       185
   macro avg       0.83      0.75      0.77       185
weighted avg       0.82      0.82      0.81       185
```



<u>HOME LOAN PREDICTION</u>

LOGISTIC REGRESSION

Training Accuracy - 80.18%

Testing Accuracy - 82.70%

<u>HOME LOAN PREDICTION:</u>

LOGISTIC REGRESSION

```
training accuracy is  80.1864801864802
testing accuracy is  82.70270270270271
```

```
              precision    recall  f1-score   support

           0       0.91      0.41      0.57        51
           1       0.81      0.99      0.89       134

    accuracy                           0.83       185
   macro avg       0.86      0.70      0.73       185
weighted avg       0.84      0.83      0.80       185
```

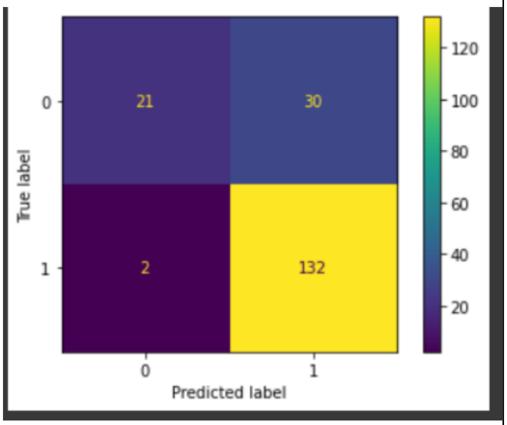
<table>
<tr>
<td></td>
<td></td>
<td>

RANDOM FOREST

Training Accuracy - 80.18%

Testing Accuracy - 82.70%

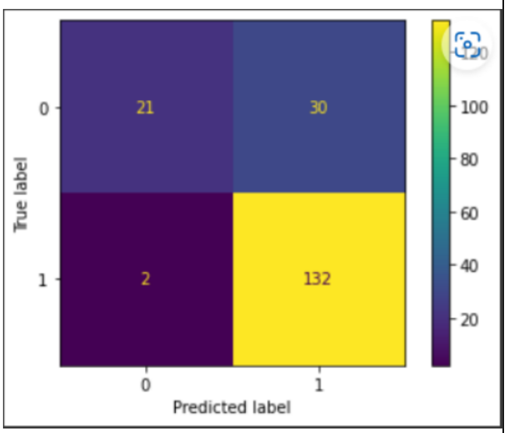
</td>
<td>



RANDOM FOREST

```
training accuracy is  80.1864801864802
testing accuracy is  82.70270270270271
```

	precision	recall	f1-score	support
0	0.91	0.41	0.57	51
1	0.81	0.99	0.89	134
accuracy			0.83	185
macro avg	0.86	0.70	0.73	185
weighted avg	0.84	0.83	0.80	185



</td>
</tr>
</table>

KNN

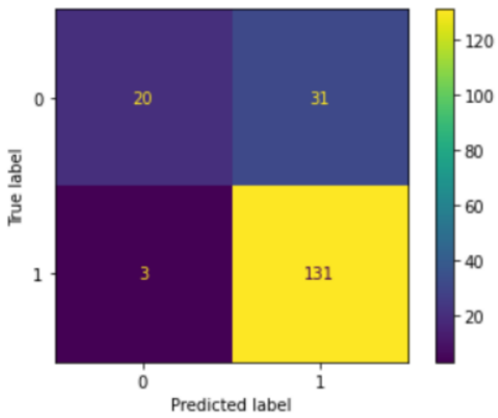
Training Accuracy - 79.95%

Testing Accuracy - 81.62%

KNN

```
training accuracy is  79.95337995337995
testing accuracy is  81.62162162162161
```

	precision	recall	f1-score	support
0	0.87	0.39	0.54	51
1	0.81	0.98	0.89	134
accuracy			0.82	185
macro avg	0.84	0.68	0.71	185
weighted avg	0.83	0.82	0.79	185



XGB

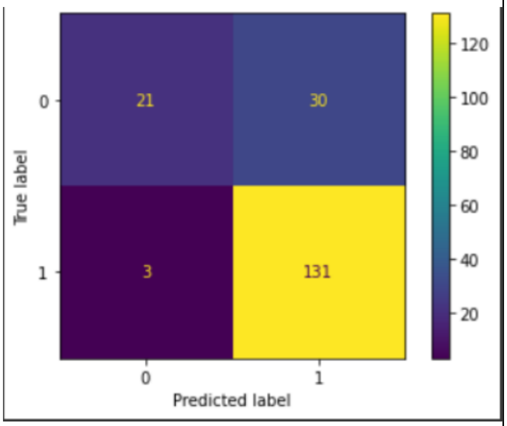
Training Accuracy - 80.88%

Testing Accuracy - 82.16%

XGB

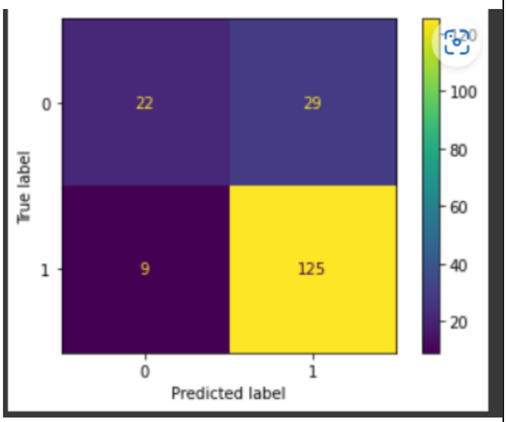
```
training accuracy is  80.88578088578089
testing accuracy is  82.16216216216216
```

	precision	recall	f1-score	support
0	0.88	0.41	0.56	51
1	0.81	0.98	0.89	134
accuracy			0.82	185
macro avg	0.84	0.69	0.72	185
weighted avg	0.83	0.82	0.80	185

<table>
<tr>
<td></td>
<td></td>
<td>
DECISION TREE
Training Accuracy - 83.91%
Testing Accuracy - 79.45%
</td>
<td>


DECISION TREE
<pre>
training accuracy is 83.91608391608392
testing accuracy is 79.45945945945945
</pre>
<pre>
              precision   recall f1-score   support

           0   0.71   0.43   0.54   51
           1   0.81   0.93   0.87   134

    accuracy   0.79   185
   macro avg   0.76   0.68   0.70   185
weighted avg   0.78   0.79   0.78   185
</pre>


</td>
</tr>
</table>

CHAPTER 10

ADVANTAGES & DISADVANTAGES

Advantages:

1) Improved customer experience
   - Being available 24/7, mobile banking is great for those who are not always able to visit the actual facility during its working hours.

2) Time efficiency
   - Using this banking app we can check whether the user is eligible for the loan or not and never worry about the possibility of physical check.

3) Eliminates middlemen, makes loan approval a hassle-free process

Disadvantages:

1) The drawback of this model is that it takes into consideration many attributes but in real life sometimes the loan application can also be approved on a single strong attribute, which will not be possible using this system.

2) The model predicts Business and Home loan alone but On given more data, the model can be able to predict other types of loan too

CHAPTER 11

CONCLUSION

Therefore, it can be said with certainty that all of the models are quite effective and produce better results. It operates properly and satisfies all bankers' requirements. This technology calculates the outcome correctly and precisely. It accurately predicts whether a loan application or customer will be accepted or denied.

CHAPTER 12

FUTURE SCOPE

This app can be extended for other types of loan like education loan etc. So, in the near future the software could be made more secure, reliable and dynamic weight adjustment. In near future this module of prediction can be integrate with the module of automated processing system.

CHAPTER 13

APPENDIX

Github Link:

https://github.com/IBM-EPBL/IBM-Project-2070-1658425806

Demo Link:

https://drive.google.com/file/d/1ZQwt4g6vGl4KkELREJb3XR6UnaxyDU7_/view?usp=share_link