INTRODUCTION

A loan is the core business part of banks. The main portion the bank's profit is directly come from the profit earned from the loans. Though bank approves loan after a regress process of verification and testimonial but still there's no surety whether the chosen hopeful is the right hopeful or not. This process takes fresh time while doing it manually. We can prophesy whether that particular hopeful 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.

Overview

Bank employees check the details of applicant manually and give the loan to eligible applicant. Checking the details of all applicants takes lot of time. The artificial neural network model for predict the credit risk of a bank. The Feed- forward back propagation neural network is used to forecast the credit default. The method in which two or more classifiers are combined together to produce a ensemble model for the better prediction. They used the bagging and boosting techniques and then used random forest technique. The process of classifiers is to improve the performance of the data and it gives better efficiency. In this work, the authors describe various ensemble techniques for binary classification and also for multi class classification. The new technique that is described by the authors for ensemble is COB which gives effective performance of classification but it also compromised with noise and outlier data of classification. Finally they concluded that the ensemble based algorithm improves the results for training data set.

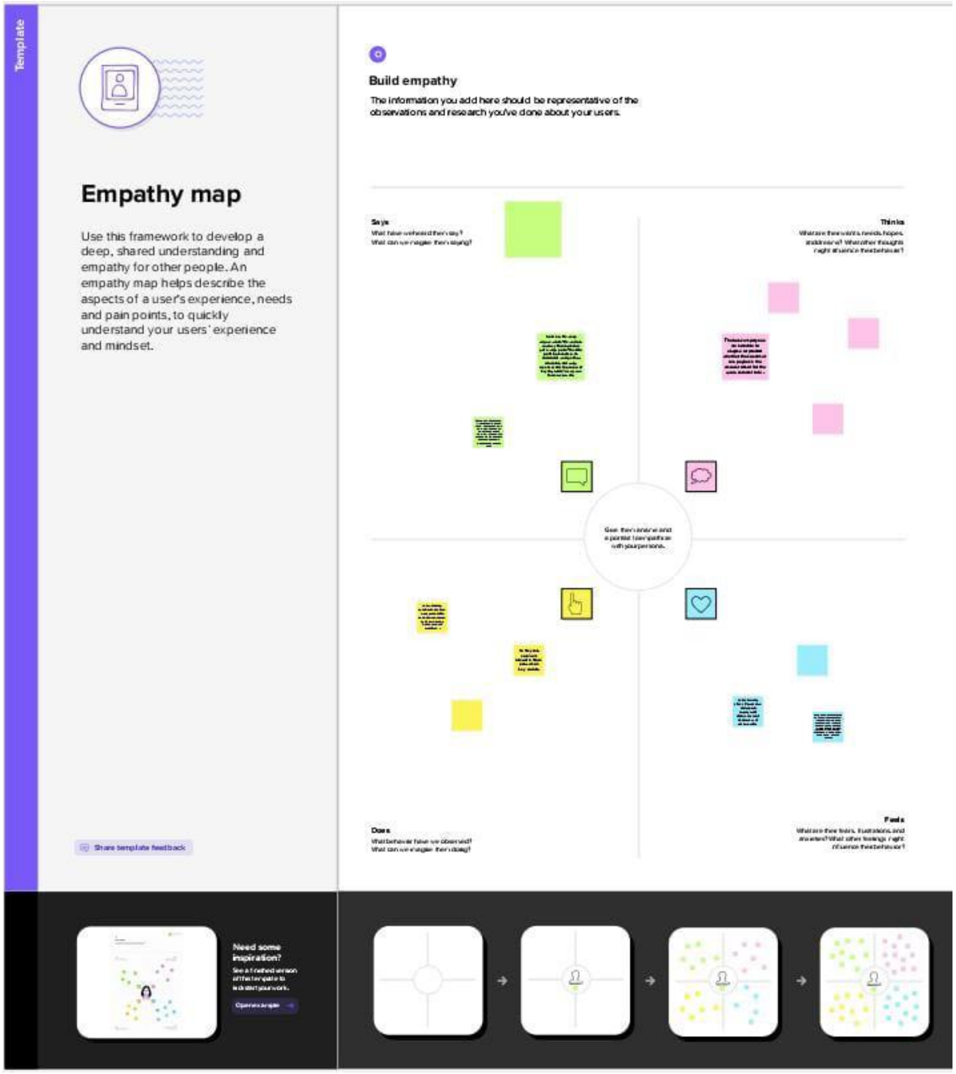
Purpose

When you fill out a loan application, you may come across a section that asks you to specify the purpose of the loan. Some lenders do this to give you the right product. They may also use your loan objective to assess risk and specify loan terms.

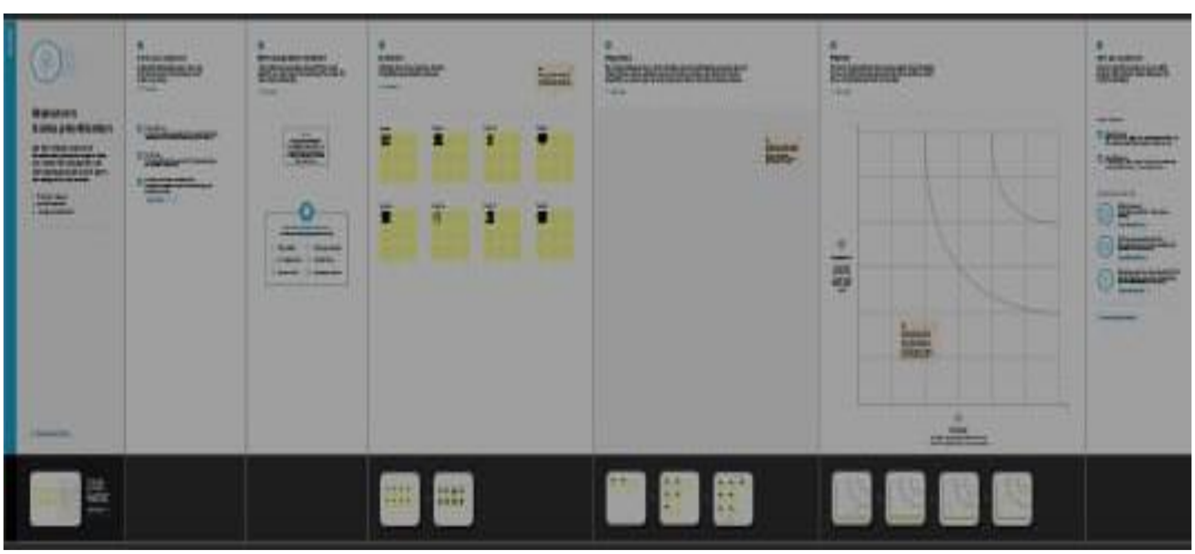
There are several reasons you might consider taking out a Personal Loan. Most people have something special on their minds when they decide to borrow money. Three out of every four people considering taking out a Personal Loan say the decision is driven by a specific upcoming need or life event.

Although each person considers himself and his personal needs unique, it turns out that the reasons for taking out a **personal loan** fall into this list which is as follows:

EMPATHY MAP



Ideation & Brainstorming map



RESULT

The analytical process started from data cleaning and processing, Missing value imputation with mice package, then exploratory analysis and finally model building and evaluation. The best accuracy on public test set is 0.811. This brings some of the following insights about approval. Applicants with Credit history not passing fails to get approved, Probably because that they have a probability of a not paying back. Most of the Time, Applicants with high income sanctioning low amount is to more likely get approved which make sense, more likely to pay back their loans. Some basic characteristic gender and marital status seems not to be taken into consideration by the company

ADVANTAGES

Accuracy—one of the primary benefits of using machine learning for credit scoring is **its accuracy**. Unlike human manual processing, ML-based models are automated and less likely to make mistakes. This means that loan processing becomes not only faster but more accurate, too, cutting costs on the whole.

DISADVANTAGES

The disadvantage of this model is that **it emphasize different weights to each factor** but in real life sometime loan can be approved on the basis of single strong factor only, which is not possible through this system. Loan Prediction is very helpful for employee of banks as well as for the applicant also.

APPLICATIONS

An in principal approval loan is **a loan which indicates whether bank can potentially lend the amount to the borrower**. In principal approval is availed in the following scenarios: If you are planing to buy a new home. If you have found a property and need an indication that we may be able to lend you the amount you need.

CONCLUSION

The developed model automates the method of determining the applicant's creditworthiness. It focuses on information containing the main points of the loan applicants. In this system NAIVE BAYES Classification model is used. In Machine Learning, NAIVE BAYES classification analysis is one of the supervised learning algorithms, which is dependent on BAYES theorem and used to solve classification problems. Hence, it is good for predicting the right result in the current world scenario and also help the bank to give the money in the right hands and also help the people in getting loan in a much faster way. The main advantage of this system is, it gives more accuracy.

FUTURE SCOPE

The system is trained on old training dataset in future software can be made such that new testing data should also take part in training data after some fix time.

APPENDIX

Importing the libraries

```python
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RandomizedSearchCV
import imblearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score
```

Read the Dataset

```python
#importing the dataset which is in csv file
data = pd.read_csv('loan_prediction.csv')
data
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	L
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	3
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	3
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	3
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	3
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	3
...	...	...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	3
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	1
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	3
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	3
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	3

614 rows × 13 columns

Handling missing values

```
data.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
```

```
#finding the sum of null values in each column
data.isnull().sum()
```

```
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
```

```
data['Gender'] = data['Gender'].fillna(data['Gender'].mode()[0])

data['Married'] = data['Married'].fillna(data['Married'].mode()[0])

#replacing + with space for filling the nan values
data['Dependents']=data['Dependents'].str.replace('+','')

data['Dependents'] = data['Dependents'].fillna(data['Dependents'].mode()[0])

data['Self_Employed'] = data['Self_Employed'].fillna(data['Self_Employed'].mode()[0])

data['LoanAmount'] = data['LoanAmount'].fillna(data['LoanAmount'].mode()[0])

data['Loan_Amount_Term'] = data['Loan_Amount_Term'].fillna(data['Loan_Amount_Term'].mode()[0])

data['Credit_History'] = data['Credit_History'].fillna(data['Credit_History'].mode()[0])
```

Handling Categorical Values

```python
#changing the datype of each float column to int
data['Gender']=data['Gender'].astype('int64')
data['Married']=data['Married'].astype('int64')
data['Dependents']=data['Dependents'].astype('int64')
data['Self_Employed']=data['Self_Employed'].astype('int64')
data['CoapplicantIncome']=data['CoapplicantIncome'].astype('int64')
data['LoanAmount']=data['LoanAmount'].astype('int64')
data['Loan_Amount_Term']=data['Loan_Amount_Term'].astype('int64')
data['Credit_History']=data['Credit_History'].astype('int64')
```

Handling Imbalance Data

```python
#Balancing the dataset by using smote
from imblearn.combine import SMOTETomek

smote = SMOTETomek(0.90)
```

```
C:\Users\HP\AppData\Roaming\Python\Python39\site-packages\imblearn\utils\_validation.py:587: FutureWarning: Pass sampling_strategy=0.
keyword args. From version 0.9 passing these as positional arguments will result in an error
  warnings.warn(
```

```python
#dividing the dataset into dependent and independent y and x respectively
y = data['Loan_Status']
x = data.drop(columns=['Loan_Status'],axis=1)

#creating a new x and y variables for the balnced set
x_bal,y_bal = smote.fit_resample(x,y)

#printing the values of y before balancing the data and after
print(y.value_counts())
print(y_bal.value_counts())
```

```
1    422
0    192
Name: Loan_Status, dtype: int64
1    351
0    388
Name: Loan_Status, dtype: int64
```

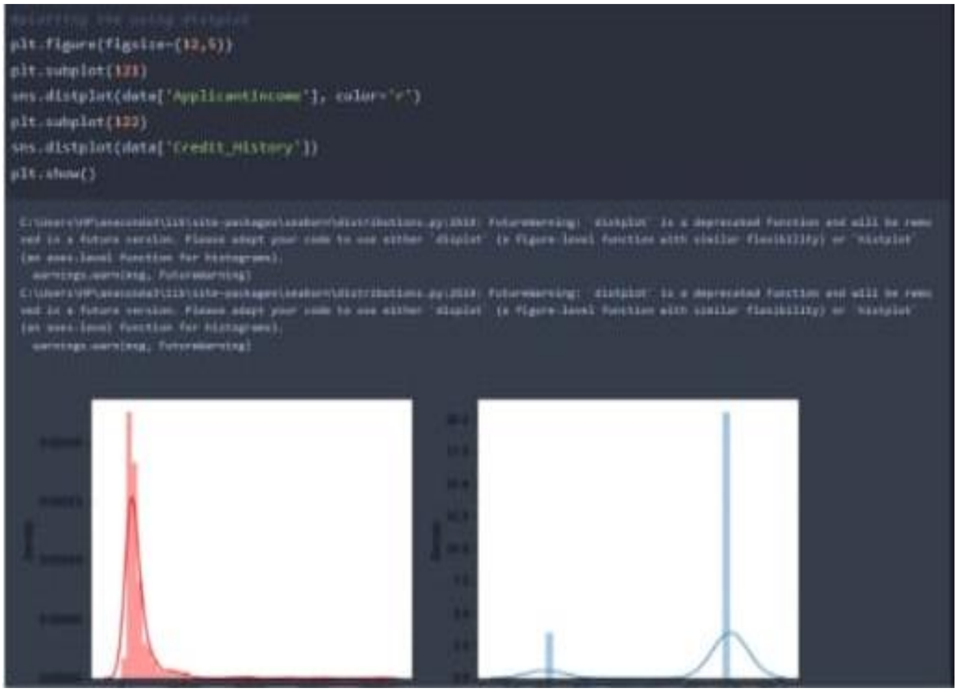
Exploratory Data Analysis

Descriptive statistical

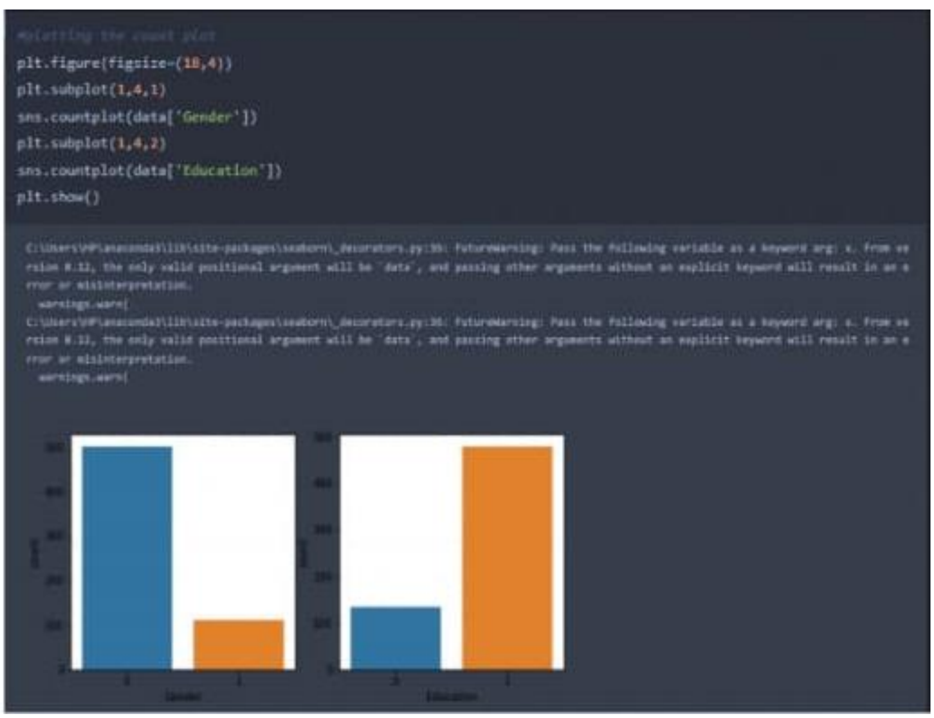
```python
data.describe()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

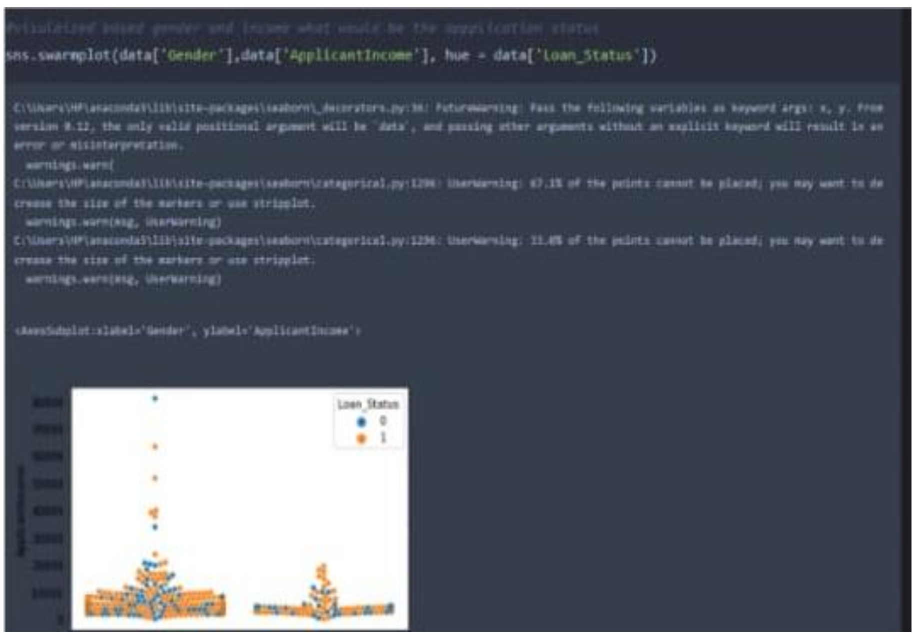
Univariate analysis



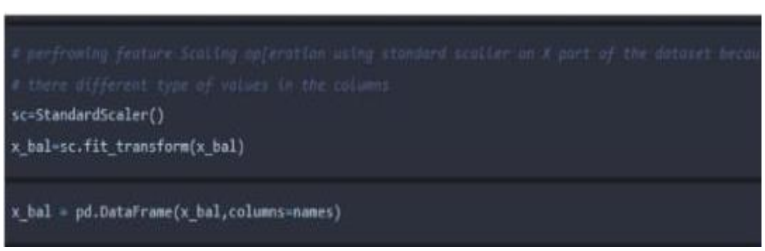
Bivariate analysis



Multivariate analysis



Scaling the Data



Splitting data into train and test

```
#splitting the dataset in train and test on balnmced datasew
X_train, X_test, y_train, y_test = train_test_split(
    x_bal, y_bal, test_size=0.33, random_state=42)
```

Model Building

Decision tree model

```
def decisionTree(x_train, x_test, y_train, y_test)
    dt=DecisionTreeClassifier()
    dt.fit(x_train,y_train)
    yPred = dt.predict(x_test)
    print('***DecisionTreeClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

Random Forest model

```
def randomForest(x_train, x_test, y_train, y_test):
    rf = RandomForestClassifier()
    rf.fit(x_train,y_train)
    yPred = rf.predict(x_test)
    print('***RandomForestClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

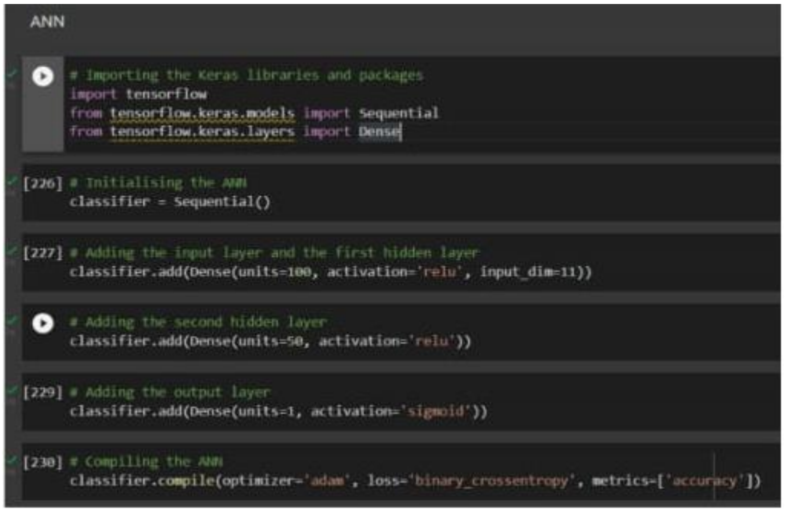
KNN model

```
def KNN(x_train, x_test, y_train, y_test):
    knn = KNeighborsClassifier()
    knn.fit(x_train,y_train)
    yPred = knn.predict(x_test)
    print('***KNeighborsClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

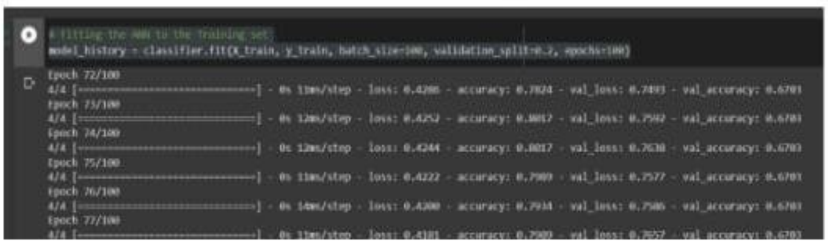
Xgboost model

```
def xgboost(x_train, x_test, y_train, y_test):
    xg = GradientBoostingClassifier()
    xg.fit(x_train,y_train)
    yPred = xg.predict(x_test)
    print('***GradientBoostingClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

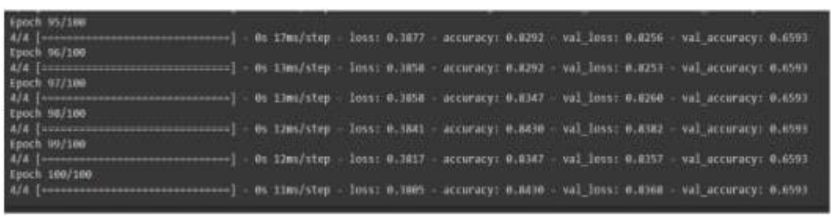
ANN model



```
ANN

# Importing the Keras libraries and packages
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

[226] # Initialising the ANN
classifier = Sequential()

[227] # Adding the input layer and the first hidden layer
classifier.add(Dense(units=100, activation='relu', input_dim=11))

# Adding the second hidden layer
classifier.add(Dense(units=50, activation='relu'))

[229] # Adding the output layer
classifier.add(Dense(units=1, activation='sigmoid'))

[230] # Compiling the ANN
classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```



```
# Fitting the ANN to the Training set
model_history = classifier.fit(X_train, y_train, batch_size=100, validation_split=0.2, epochs=100)

Epoch 72/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4286 - accuracy: 0.7824 - val_loss: 0.7493 - val_accuracy: 0.6703
Epoch 73/100
4/4 [==============================] - 0s 12ms/step - loss: 0.4252 - accuracy: 0.8017 - val_loss: 0.7592 - val_accuracy: 0.6703
Epoch 74/100
4/4 [==============================] - 0s 12ms/step - loss: 0.4244 - accuracy: 0.8017 - val_loss: 0.7638 - val_accuracy: 0.6703
Epoch 75/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4222 - accuracy: 0.7909 - val_loss: 0.7577 - val_accuracy: 0.6703
Epoch 76/100
4/4 [==============================] - 0s 14ms/step - loss: 0.4200 - accuracy: 0.7934 - val_loss: 0.7586 - val_accuracy: 0.6703
Epoch 77/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4181 - accuracy: 0.7909 - val_loss: 0.7657 - val_accuracy: 0.6703
```



```
Epoch 95/100
4/4 [==============================] - 0s 17ms/step - loss: 0.3877 - accuracy: 0.8292 - val_loss: 0.8256 - val_accuracy: 0.6593
Epoch 96/100
4/4 [==============================] - 0s 13ms/step - loss: 0.3858 - accuracy: 0.8292 - val_loss: 0.8253 - val_accuracy: 0.6593
Epoch 97/100
4/4 [==============================] - 0s 13ms/step - loss: 0.3858 - accuracy: 0.8347 - val_loss: 0.8260 - val_accuracy: 0.6593
Epoch 98/100
4/4 [==============================] - 0s 12ms/step - loss: 0.3841 - accuracy: 0.8430 - val_loss: 0.8382 - val_accuracy: 0.6593
Epoch 99/100
4/4 [==============================] - 0s 12ms/step - loss: 0.3817 - accuracy: 0.8347 - val_loss: 0.8357 - val_accuracy: 0.6593
Epoch 100/100
4/4 [==============================] - 0s 11ms/step - loss: 0.3805 - accuracy: 0.8430 - val_loss: 0.8368 - val_accuracy: 0.6593
```

Testing the model

```
[147] #Gender Married Dependents  Education Self_Employed ApplicantIncome CoapplicantIncome LoanAmount  Loan_Amount_Term  Credit_History  Property_Area
      dtr.predict([[1,1, 0, 1, 1, 4276, 1542,145, 240, 0,1]])

      /usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature names
        warnings.warn(
      array([0])

[149] #Gender Married Dependents  Education Self_Employed ApplicantIncome CoapplicantIncome LoanAmount  Loan_Amount_Term  Credit_History  Property_Area
      rfr.predict([[1,1, 0, 1, 1, 4276, 1542,145, 240, 0,1]])

      /usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
        warnings.warn(
      array([1])

      #Gender Married Dependents  Education Self_Employed ApplicantIncome CoapplicantIncome LoanAmount  Loan_Amount_Term  Credit_History  Property_Area
      knn.predict([[1,1, 0, 1, 1, 4276, 1542,145, 240, 0,1]])

      /usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but KNeighborsClassifier was fitted with feature names
        warnings.warn(
      array([1])

[153] #Gender Married Dependents  Education Self_Employed ApplicantIncome CoapplicantIncome LoanAmount  Loan_Amount_Term  Credit_History  Property_Area
      xgb.predict([[1,1, 0, 1, 1, 4276, 1542,145, 240, 0,1]])

      /usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but GradientBoostingClassifier was fitted with feature names
        warnings.warn(
      array([1])
```

```
classifier.save("loan.h5")

# Predicting the Test set results
y_pred = classifier.predict(X_test)

8/8 [==============================] - 0s 2ms/ste

[237] y_pred
          [0.03911224],
          [0.5707451 ],
          [0.9951428 ],
```

```
[238] y_pred = (y_pred > 0.5)
      y_pred
          [False],
          [ True],
          [ True],
          [ True],
```

```python
[244] def predict_exit(sample_value):

    # Convert list to numpy array
    sample_value = np.array(sample_value)

    # Reshape because sample_value contains only 1 record
    sample_value = sample_value.reshape(1, -1)

    # Feature Scaling
    sample_value = sc.transform(sample_value)

    return classifier.predict(sample_value)
```

```python
# Predictions
# Value order 'CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary','France','Germany','Spain','Female','Male'.
sample_value = [[1,1, 0, 1, 1, 4276, 1542,145, 240, 0,1]]
if predict_exit(sample_value)>0.5:
    print('Prediction: High chance of Loan Approval!')
else:
    print('Prediction: Low chance Loan Approval.')
```

```
1/1 [==============================] - 0s 18ms/step
Prediction: Low chance Loan Approval.
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

```python
# Predictions
# Value order 'CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary','France','Germany','Spain','Female','Male'.
sample_value = [[1,0, 1, 1, 1, 45, 14,45, 240, 1,1]]
if predict_exit(sample_value)>0.5:
    print('Prediction: High chance of Loan Approval!')
else:
    print('Prediction: Low chance of Loan Approval.')
```

```
1/1 [==============================] - 0s 50ms/step
Prediction: High chance of Loan Approval!
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

Performance testing &Hyperparameter Tuning

Compare the model

```python
def compareModel(X_train,X_test,y_train,y_test):
  decisionTree(X_train,X_test,y_train,y_test)
  print('-'*100)
  RandomForest(X_train,X_test,y_train,y_test)
  print('-'*100)
  XGB(X_train,X_test,y_train,y_test)
  print('-'*100)
  KNN(X_train,X_test,y_train,y_test)
  print('-'*100)
```

```
compareModel(X_train,X_test,y_train,y_test)
```

```
1.0
0.7822222222222223
Decision Tree
Confusion_Matrix
[[83 24]
 [25 93]]
Classification Report
              precision    recall  f1-score   support

           0       0.77      0.78      0.77       107
           1       0.79      0.79      0.79       118

    accuracy                           0.78       225
   macro avg       0.78      0.78      0.78       225
weighted avg       0.78      0.78      0.78       225

------------------------------------------------------------
```

```
------------------------------------------------------------
1.0
0.8088888888888889
Random Forest
Confusion_Matrix
[[ 78  29]
 [ 14 104]]
Classification Report
              precision    recall  f1-score   support

           0       0.85      0.73      0.78       107
           1       0.78      0.88      0.83       118

    accuracy                           0.81       225
   macro avg       0.81      0.81      0.81       225
weighted avg       0.81      0.81      0.81       225
```

```
0.933920704845815
0.8222222222222222
XGBoost
Confusion_Matrix
[[ 78  29]
 [ 11 107]]
Classification Report
              precision    recall  f1-score   support

           0       0.88      0.73      0.80       107
           1       0.79      0.91      0.84       118

    accuracy                           0.82       225
   macro avg       0.83      0.82      0.82       225
weighted avg       0.83      0.82      0.82       225
```

```
0.7665198237885462
0.6666666666666666
KNN
Confusion_Matrix
[[60 47]
 [28 90]]
Classification Report
              precision    recall  f1-score   support

           0       0.68      0.56      0.62       107
           1       0.66      0.76      0.71       118

    accuracy                           0.67       225
   macro avg       0.67      0.66      0.66       225
weighted avg       0.67      0.67      0.66       225
```

```
yPred = classifier.predict(X_test)
print(accuracy_score(y_pred,y_test))
print("ANN Model")
print("Confusion_Matrix")
print(confusion_matrix(y_test,y_pred))
print("Classification Report")
print(classification_report(y_test,y_pred))
```

```
8/8 [==============================] - 0s 4ms/step
0.6844444444444444
ANN Model
Confusion_Matrix
[[63 44]
 [27 91]]
Classification Report
              precision    recall  f1-score   support

           0       0.70      0.59      0.64       107
           1       0.67      0.77      0.72       118

    accuracy                           0.68       225
   macro avg       0.69      0.68      0.68       225
weighted avg       0.69      0.68      0.68       225
```

Comparing model accuracy before&after applying hyparamater tuning

```python
from sklearn.model_selection import cross_val_score

# Random forest model is selected

rf = RandomForestClassifier()
rf.fit(x_train,y_train)
yPred = rf.predict(x_test)

f1_score(yPred,y_test,average='weighted')
```

```
0.9679166666666668
```

```python
cv = cross_val_score(rf,x,y,cv=5)

np.mean(cv)
```

```
0.985
```

```
0.9691629955947136
0.8222222222222222
Random Forest
Confusion_Matrix
[[ 77  30]
 [ 10 108]]
Classification Report
              precision    recall  f1-score   support

           0       0.89      0.72      0.79       107
           1       0.78      0.92      0.84       118

    accuracy                           0.82       225
   macro avg       0.83      0.82      0.82       225
weighted avg       0.83      0.82      0.82       225

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
```

Model deployment

Save the best model

```python
#saviung the model by using pickle function
pickle.dump(model,open('rdf.pkl','wb'))
```

Build python code

```python
from flask import Flask, render_template, request
import numpy as np
import pickle
```

```python
app = Flask(__name__)
model = pickle.load(open(r'rdf.pkl', 'rb'))
scale = pickle.load(open(r'scale1.pkl','rb'))
```

Render HTML page:

```python
@app.route('/') # rendering the html template
def home():
    return render_template('home.html')
```

```python
@app.route('/submit',methods=["POST","GET"])# route to show the predictions in a web UI
def submit():
    #  reading the inputs given by the user
    input_feature=[int(x) for x in request.form.values() ]
    #input_feature = np.transpose(input_feature)
    input_feature=[np.array(input_feature)]
    print(input_feature)
    names = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'ApplicantIncome',
       'CoapplicantIncome','LoanAmount','Loan_Amount_Term','Credit_History','Property_Area']
    data = pandas.DataFrame(input_feature,columns=names)
    print(data)

    #data_scaled = scale.fit_transform(data)
    #data = pandas.DataFrame(,columns=names)



     # predictions using the loaded model file
    prediction=model.predict(data)
    print(prediction)
    prediction = int(prediction)
    print(type(prediction))

    if (prediction == 0):
       return render_template("output.html",result ="Loan will Not be Approved")
    else:
       return render_template("output.html",result = "Loan will be Approved")
     # showing the prediction results in a UI
```

```python
if __name__=="__main__":

   # app.run(host='0.0.0.0', port=8000,debug=True)    # running the app
   port=int(os.environ.get('PORT',5000))
   app.run(debug=False)
```

Run the web application

```
ase) D:\TheSmartBridge\Projects\2. DrugClassification\Drug c
 Serving Flask app "app" (lazy loading)
 Environment: production
 WARNING: This is a development server. Do not use it in a p
 Use a production WSGI server instead.
 Debug mode: off
 Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```