# GOVERNMENT ARTS & SCIENCE COLLEGE WOMEN BARGUR

III BCA SHIFT –[1]

PERSONAL LOAN APPROVAL

1.INTRODUCTION

1.1 OVERVIEW

Generally, loan prediction involves the leander looking at various background information about the applicant and deciding whether the bank should great the loan.

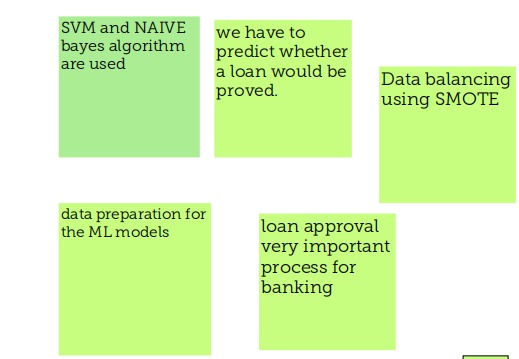
1.2 PURPOSE

THE prediction model not only helps the applicant but also helps the bank by minimizing the risk and reducing the numbers of defaulters.

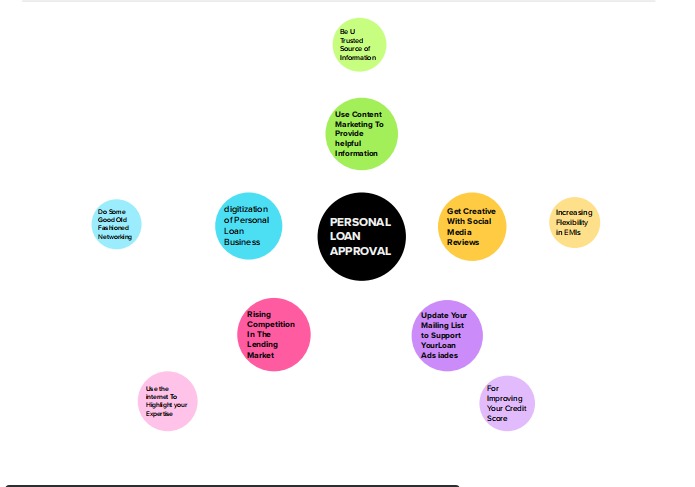
It’s done by predicting if the loan can be given to the person on the basis of various parameter like credit score, income, age, martial status etc…….

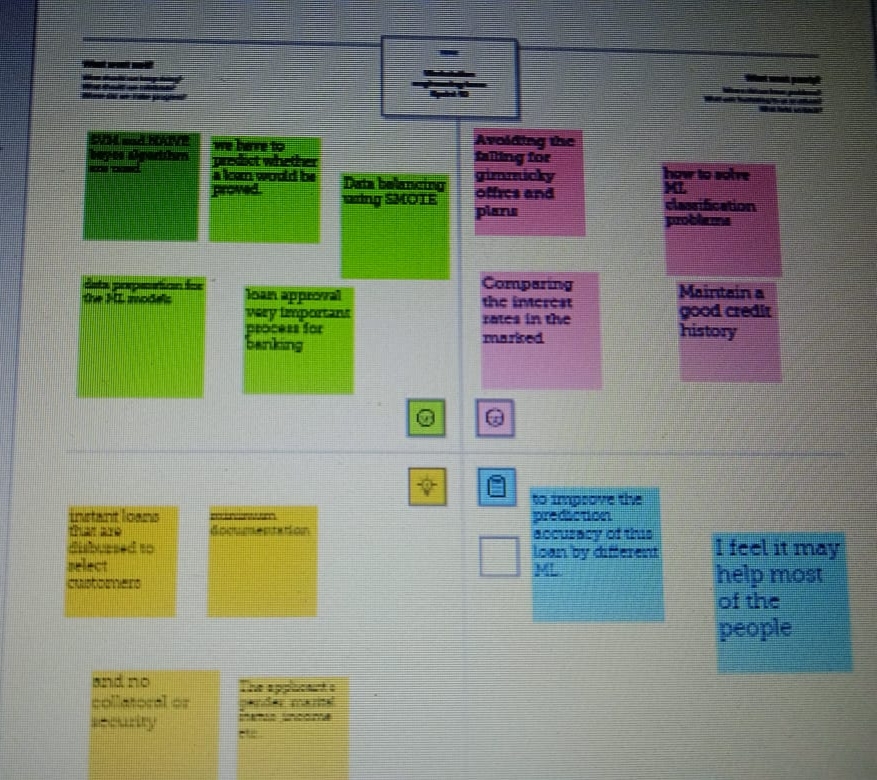
2. PROBLEM DEFINITION & DESIGN THINKING

2.1 EMPATHY MAP

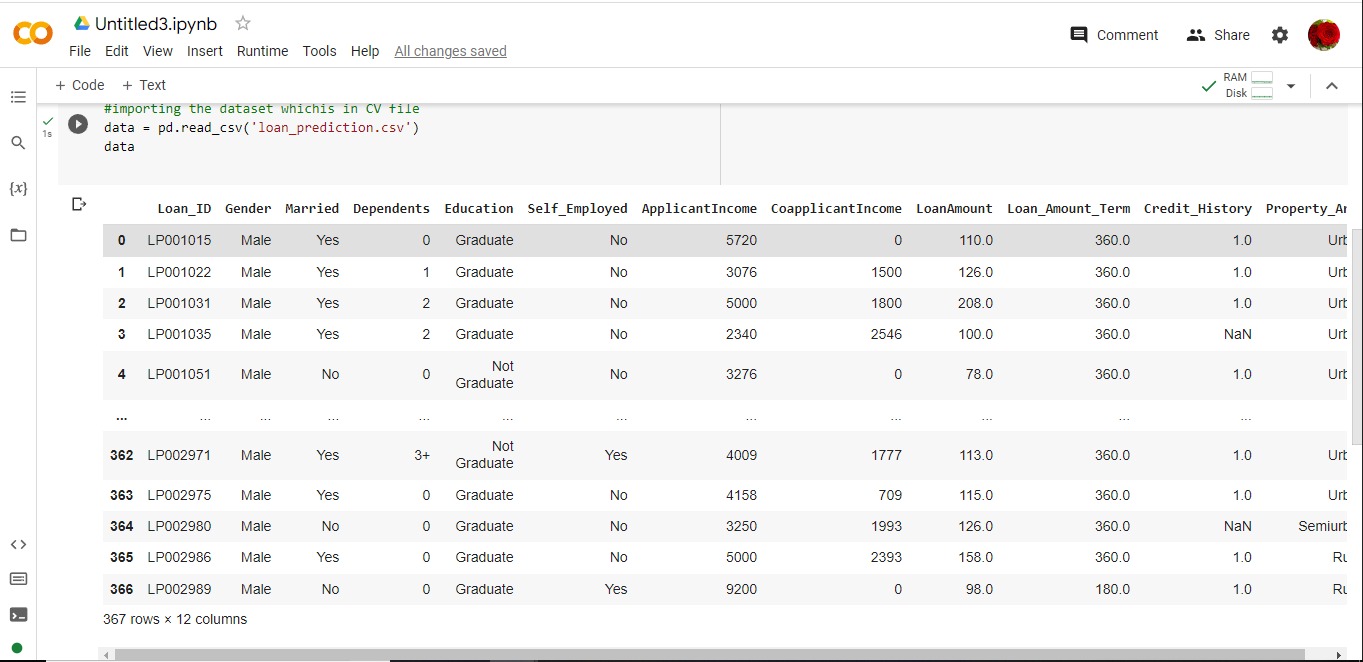
**

2.2 IDEATION & BRAINSTORMING MAP





3.RESULT



4.ADVANTAGES & DISADVANTAGES

ADVANTAGES

* Lower interest rate
* Quick funding
* No security deposits
* Various purpose

DISADVANTAGE

* Interest premiums
* Processing fees
* Fixed payments
* Lower returns on investment

5. APPLICATIONS

* Good customer support
* Doorstep loan service
* Minimum documentation with digital
* Get you an immediate approval
* Saves your valuable time

6. CONCLUSION

A system called loan credibility predication system that helps the organization in making right decision to approve or reject the loan request of the customer.

7.FUTURE SCOPE

When digitalization in almost everything today a personal loan has become accessible to almost everyone.

8. APPENDIX

A .SOURCE CODE

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 ensemble from.sklearn.ensemble import GradientBoostingClassifier,RandomForestClassifier

from sklearn.neighbours 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.metrices import

accurancy\_score,classfication\_report,confusion\_matrix,f1\_score

data = pd.read\_csv('loan\_prediction.csv') data.info()

data.isnull().sum()

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

data['Married'] = data['Married'].fillna(data['Married'].mode()[0]) data['Dependents'] = data['Dependents'].fillna(data['Dependents'].mode()[0]) data['Dependents'] = data['Dependents'].fillna(data['Dependents'].mode()[0]) data['Self\_Employed'] = data['Self\_Employed'].fillna(data['Self\_Employed'].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]) 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')

from imblearn.combine import SMOTETomek smote = SMOTETomek(0.90)

y = data['Loan\_Status']

x = data.drop(coiumns=['Loan\_Status'],axis=1) x\_bal,y\_bal = smote.fit\_resample(x,y) print(y.value\_counts()) print(y\_bal.value\_counts())

data.describe()

plt.figure(figsize=(12.5)) plt.subplot(121)

sns.distplot(data['ApplicantIncome'],color='r) plt.subplot(122) sns.displot(data['credit\_History'])

plt.show()

plt.figure(figsize=((18,24)) plt.subplot(1,4,1) sns.countplot(data['Gender']) plt.subplot(1,4,2) sns.countplot(data['Education']) plt.show()

plt.figure(figsize=(20,5)) plt.subplot(131)

sns.countplot(data['Married'], hue=data['Gender']) plt.subplot(132)

sns.countplot(data['self\_Employed'], hue=data['Education']) plt.figure(figsize=((18,24))

plt.subplot(133)

sns.countplot(data['Property\_Area'], hue=data['Loan\_Amount\_Term'])

sns.swarmplot(data['Gender'],data['ApplicantIncome'], hue=data['Loan\_Status'])

sc=StandardScalar() x\_bal=sc.fit\_transform(x\_bal)

x\_bal pd.DataFrame(x\_bal,columns=names) x\_train,x\_test,y\_train,y\_test = train\_test\_split( x\_bal,y\_bal,test\_size=0.33,random\_state=42)

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))

def randomForest(x\_train,x\_test,y\_train,y\_test): rf = RondomForestClassifier() rf.fit(x\_train,y\_train)

yPred = rf.predict(x\_test) print('\*\*\*RondomForestClassifier \*\*\*') print('Confusion matrix') print(confusion\_matrix(y\_test,yPred)) print('Classification report') print(classification\_report(y\_test,yPred))

def KNN(x\_train,x\_test,y\_train,y\_test): knn = KNeighboursClassfier() knn.fit(x\_train,y\_train)

yPred = knn.predict(x\_test) print('\*\*\*KNeighboursClassifier\*\*\*') print('Confusion matrix') print(confusion\_matrix(y\_test,yPred)) print('Classification report') print(classification\_report(y\_test,yPred))

def xgboost(x\_train,x\_test,y\_train,y\_test): xg= GradientBoostingClassfier() 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))

import tensortflow

from tensortflow.keras.models import sequential from tensortflow.keras.layers import Dense

classifier = Sequential() Classifier.add(Dence(units=100,activation='relu',input\_dim=11)) classifier.add (Dence(units=50,activation='relu') classifier.add (Dence(units=1,activation='sigmoid')

classifier.compile(optimizer='adam',loss='binary\_crossentropy',matrices=[' accurancy'])

model\_history = classifier.fit(x\_train,y\_train,batch\_size=100,validation\_split=0.2,epochs= 100)

dtr.predict([[1,1,0,1,1,4276,1542,145,240,0,1]])

rfr.predict([[1,1,0,1,1,4276,1542,145,240,0,1]])

knn.predict([[1,1,0,1,1,4276,1542,145,240,0,1]])

xgb.predict([[1,1,0,1,1,4276,1542,145,240,0,1]])

\_ classfier.save("loan.h5)

y\_pred = classifier.predict(x\_test)

1. y\_pred
2. y\_pred = (y\_pred > 0.5) y\_pred

[244] def predict\_exit(sample\_value): sample\_value = np.array(sample\_value) sample\_value=sample\_value.reshape(1,-1) sample\_value = sc.transform(sample\_value)

return classifier.predict(sample\_value) 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 of loan Approval.')

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 of loan Approval.') 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) yPred = classifier.predict(x\_test)

print(accurancy\_score(y\_pred,y\_test) print("ANN Model")

print('Confusion matrix') print(confusion\_matrix(y\_test,yPred)) print('Classification report') print(classification\_report(y\_test,yPred))

rf = RandomForestClassifier() rf.fit(x\_train,y\_train)

ypred = rf.predict(x\_test) f1\_score(yPred,y\_test,average='weighted') cv = cross\_val\_score(rf,x,y,cv=5) np.mean(cv)

pickle.dump(model,open('rdf.pk1','wb'))

from flask import Flask,render\_template,request import numpy as np

import pickle

app = Flask(\_name\_)

model = pickle.load(open(r'rdf.pkl','rb')) scale = pickle.load(open(r'scale.pkl','rb'))

@app.route('/') def home():

return render\_template('home.html')

@app.route('/submit',methods=["POST","GET"]) def submit():

# reading the inputs given by the user input\_feature=[int(x) for x in request.form.values() ] #input\_feature = np.transponse(input\_feature) input\_feature=[np.array(input\_feature) ] print(input\_feature)

names = ['Gender','Married','Dependents','Education','Self\_Employed',ApplicantInco me','CoapplicantIncome','LoadAmount\_Term','Credit\_History','Property\_Area'

]

data = pandas.DataFrame(input\_feature,column=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 Apporved") else:

return render\_template("output.html"result ="Loan will Not be Apporved")

#showing the prediction results in a UI

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)