Consumer Credit Worthiness, Milestone : 3 ,Feature Engineering

```
# Import neccessary Libraries

import numpy as np
import pandas as pd
import seaborn as sbn
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder

#  Model selection libraries
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score,GridSearchCV

# Ml models
from sklearn.linear_model import LinearRegression,Lasso,Ridge
#Lasso (least absolute shrinkage and selection operator),add L1 
penalty,L1 it is the sum of absolute value of the beta coefficient
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import 
RandomForestRegressor,AdaBoostRegressor,GradientBoostingRegressor
import xgboost
from xgboost import XGBRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score


# Model evaluation libraries
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.metrics import accuracy_score, precision_score, 
recall_score, f1_score, roc_auc_score, roc_curve, auc
from sklearn.metrics import confusion_matrix

# Decision tree visualization
from sklearn.tree import DecisionTreeClassifier, export_graphviz
import graphviz
from PIL import Image

# Importing Dataset

train=pd.read_excel("Consumer creditworthiness train data.xlsx")
train.head()
```

```
   Loan_ID Gender Married Dependents     Education Self_Employed  \
0   294853   Male      No          0      Graduate            No
```

```
1   162883   Male     Yes          1      Graduate            No
2   620668   Male     Yes          0      Graduate           Yes
3   295747   Male     Yes          0  Not Graduate            No
4   133390   Male      No          0      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0          1316025                0.0      250000             360.0
1          1031175           339300.0      256000             360.0
2           675000                0.0      132000             360.0
3           581175           530550.0      240000             360.0
4          1350000                0.0      282000             360.0

   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
```

Things to do in milestone 3

1. Encoding
2. Missing value encoding
3. Scaling
4. Adding new features

```
train.shape
```

```
(521, 13)
```

Here 13 columns and 521 rows are there. Loan_ID is not an important feature,so we can drop it

```
train.drop(columns=["Loan_ID"],inplace=True)
```

```
train.head()
```

```
  Gender Married Dependents     Education Self_Employed
ApplicantIncome  \
0   Male      No          0      Graduate            No
1316025
1   Male     Yes          1      Graduate            No
1031175
2   Male     Yes          0      Graduate           Yes
675000
3   Male     Yes          0  Not Graduate            No
581175
4   Male      No          0      Graduate            No
1350000

   CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
0                0.0      250000             360.0             1.0
```

```
1           339300.0      256000              360.0              1.0
2                0.0      132000              360.0              1.0
3           530550.0      240000              360.0              1.0
4                0.0      282000              360.0              1.0

  Property_Area Loan_Status
0         Urban           Y
1         Rural           N
2         Urban           Y
3         Urban           Y
4         Urban           Y
```

```
train.shape
```

```
(521, 12)
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 521 entries, 0 to 520
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Gender             511 non-null    object
 1   Married            518 non-null    object
 2   Dependents         508 non-null    object
 3   Education          521 non-null    object
 4   Self_Employed      494 non-null    object
 5   ApplicantIncome    521 non-null    int64
 6   CoapplicantIncome  521 non-null    float64
 7   LoanAmount         521 non-null    int64
 8   Loan_Amount_Term   507 non-null    float64
 9   Credit_History     478 non-null    float64
 10  Property_Area      521 non-null    object
 11  Loan_Status        521 non-null    object
dtypes: float64(3), int64(2), object(7)
memory usage: 49.0+ KB
```

Performing Outlier Treatement

```
def replace_Outlier(train,col,method="Quartile",strategy="Median"):
    col_data = train[col]
    if method == "Quartile":
    ## Using quartile to calculate IQR
        q1 = col_data.quantile(0.25)
        q2 = col_data.quantile(0.50)
        q3 = col_data.quantile(0.75)
        IQR = q3-q1
        LW = q1-1.5*IQR
        UW = q3+1.5*IQR

    ## Using SD
```

```
    elif method == "Standard Deviation":
        mean = col_data.mean()
        std = col_data.std()
        LW = mean - 2*std
        UW = mean + 2*std
    else:
        print("Pass a correct method")

    ## Printing all the Outliers

    Outliers = train.loc[(col_data < LW) |(col_data > UW)]
    Outlier_density = round(len(Outliers)/len(my_df),2)*100

    if len(Outliers) == 0:
        print(f'Feature {col} doesnot have outliers')
        print("\n")
    else:
        print(f"Feature {col} has Outliers")
        print("\n")
        print(f'Total no of Outliers in {col} are {len(Outliers)}')
        print("\n")
        print(f'Outlier percentage in {col} is {Outlier_density}% ')
        print("\n")
        display(train[(col_data < LW) | (col_data > UW) ])

    ## Replacing Outliers:

    if strategy == "Median" :
        train.loc[(col_data < LW) | (col_data > UW) , col] = q2
    elif strategy == "Mean" :
        train.loc[(col_data < LW) | (col_data > UW) , col] = mean
    else:
        print("Pass a correst strategy")

    return train
```

Now let us divide categories as object and non object type

```
# Categorical type
cat_cols = train.dtypes=="object"
cat_cols=list(cat_cols[cat_cols].index)
cat_cols=cat_cols + ["Credit_History"]
#cat_cols=cat_cols + ["Income"]

# Numerical type
num_cols = train.dtypes!="object"
```

```
num_cols=list(num_cols[num_cols].index)
num_cols.remove("Credit_History")
```

Missing value imputation

```
train.isnull().sum()
```

```
Gender               10
Married               3
Dependents           13
Education             0
Self_Employed        27
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term     14
Credit_History       43
Property_Area         0
Loan_Status           0
dtype: int64
```

We are using simpleimputer to impute missing values.For object type datatypes we can use strategy as "most_frequent" and for numerical datatypes we can use"median".

```
train[cat_cols]=SimpleImputer(strategy="most_frequent").fit_transform(
train[cat_cols])
train[num_cols]=SimpleImputer(strategy="median").fit_transform(train[n
um_cols])
```

```
train.isnull().sum()
```

```
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
```

Adding new features

```
train.head()
```

```
  Gender Married Dependents     Education Self_Employed
ApplicantIncome  \
0   Male      No          0      Graduate            No
1316025.0
1   Male     Yes          1      Graduate            No
```

```
1031175.0
2   Male     Yes          0      Graduate          Yes
675000.0
3   Male     Yes          0  Not Graduate           No
581175.0
4   Male      No          0      Graduate           No
1350000.0

   CoapplicantIncome  LoanAmount  Loan_Amount_Term Credit_History  \
0                0.0    250000.0             360.0            1.0
1           339300.0    256000.0             360.0            1.0
2                0.0    132000.0             360.0            1.0
3           530550.0    240000.0             360.0            1.0
4                0.0    282000.0             360.0            1.0

  Property_Area Loan_Status
0         Urban           Y
1         Rural           N
2         Urban           Y
3         Urban           Y
4         Urban           Y
```

```
train.columns
```

```
Index(['Gender', 'Married', 'Dependents', 'Education',
'Self_Employed',
       'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area',
'Loan_Status'],
      dtype='object')
```

From milestone 1 analysis,the misssing features than can be analysed based on business logic are capacity,capital,charachter,conditions,age,experience,credit score,employment history,purpose of loan and surplus income

But from our given data,we can analyse EMI,total income of family

```
## Family income can be calculated as the sum of income of applicant
and co applicant

train["Family_income"]= train["ApplicantIncome"]
+train["CoapplicantIncome"]
```

EMI is the every month interest.For EMI we have to calculate loan amount per year and we have divide it by 12 to get monthly interest.Therfore first we have to calculate loan amount per year.

Loan amount per year can be calculated by dividing loan amount by loan amount term. .

```
train["Loan_amount_per_year"]=train["LoanAmount"]/
train["Loan_Amount_Term"]
```

```
train["EMI"]=train["Loan_amount_per_year"]*100000/12
```

We can calculate capacity of customer to repay loan by comparising EMI with Family income .Let us assume that Family income is given monthly as loan term is multiple of 12.

```
train["Loan_repay_capacity"]=(train['EMI']<train["Family_income"]).ast
ype(int)
```

Binning

```
bins=[0,720000,1440000,2800000,5600000,20000000]
group=["very_low","low","average","high","very_high"]
train["Income"]=pd.cut(train["Family_income"],bins,labels=group)
```

Encoding

Encoding are mainly of three type.

1. Label encoding
2. One-Hot encoding
3. Target encoding

1. Label encoding can be used to encode target variables
2. Target encoding can be used if we have many categories
3. One-Hot encoding also known as dummy encoding

LabelEncoder

```
encoder=LabelEncoder()
train["Loan_Status"]=encoder.fit_transform(train["Loan_Status"])
```

```
train.head()
```

```
  Gender Married Dependents     Education Self_Employed
ApplicantIncome  \
0   Male      No          0      Graduate            No
1316025.0
1   Male     Yes          1      Graduate            No
1031175.0
2   Male     Yes          0      Graduate           Yes
675000.0
3   Male     Yes          0  Not Graduate            No
581175.0
4   Male      No          0      Graduate            No
1350000.0

   CoapplicantIncome  LoanAmount  Loan_Amount_Term Credit_History  \
0                0.0    250000.0             360.0            1.0
1           339300.0    256000.0             360.0            1.0
2                0.0    132000.0             360.0            1.0
3           530550.0    240000.0             360.0            1.0
4                0.0    282000.0             360.0            1.0
```

```
  Property_Area  Loan_Status  Family_income  Loan_amount_per_year  \
0         Urban            1      1316025.0            694.444444   
1         Rural            0      1370475.0            711.111111   
2         Urban            1       675000.0            366.666667   
3         Urban            1      1111725.0            666.666667   
4         Urban            1      1350000.0            783.333333   

            EMI  Loan_repay_capacity    Income  
0  5.787037e+06                    0       low  
1  5.925926e+06                    0       low  
2  3.055556e+06                    0  very_low  
3  5.555556e+06                    0       low  
4  6.527778e+06                    0       low  
```

```
train["Education"].unique()
```

```
array(['Graduate', 'Not Graduate'], dtype=object)
```

```
train["Income"].unique()
```

```
['low', 'very_low', 'average', 'high', 'very_high']
Categories (5, object): ['very_low' < 'low' < 'average' < 'high' <
'very_high']
```

```
train["Property_Area"].unique()
```

```
array(['Urban', 'Rural', 'Semiurban'], dtype=object)
```

```
train["Self_Employed"].unique()
```

```
array(['No', 'Yes'], dtype=object)
```

Mapping

```
train['Married']=train["Married"].map({"Yes":1,"No":0})

train['Education']=train["Education"].map({"Graduate":1,"Not
Graduate":0})

train['Income']=train["Income"].map({"low":1,"very_low":0,"average":2,
"high":3,"very_high":4})

train['Gender']=train["Gender"].map({"Male":1,"Female":0})

train['Self_Employed']=train["Self_Employed"].map({"Yes":1,"No":0})
```

Target Encoding

```
#train['Gender']=train.groupby("Gender")
["Loan_Status"].transform("mean")

#train['Education']=train.groupby("Education")
["Loan_Status"].transform("mean")
```

```
train.head()
```

```
   Gender  Married Dependents  Education  Self_Employed  
ApplicantIncome  \
0       1        0          0          1              0     
1316025.0   
1       1        1          1          1              0     
1031175.0   
2       1        1          0          1              1      
675000.0   
3       1        1          0          0              0      
581175.0   
4       1        0          0          1              0     
1350000.0   

   CoapplicantIncome  LoanAmount  Loan_Amount_Term Credit_History  \
0                0.0    250000.0             360.0            1.0   
1           339300.0    256000.0             360.0            1.0   
2                0.0    132000.0             360.0            1.0   
3           530550.0    240000.0             360.0            1.0   
4                0.0    282000.0             360.0            1.0   

  Property_Area  Loan_Status  Family_income  Loan_amount_per_year  \
0         Urban            1      1316025.0            694.444444   
1         Rural            0      1370475.0            711.111111   
2         Urban            1       675000.0            366.666667   
3         Urban            1      1111725.0            666.666667   
4         Urban            1      1350000.0            783.333333   

            EMI  Loan_repay_capacity Income  
0  5.787037e+06                    0      1  
1  5.925926e+06                    0      1  
2  3.055556e+06                    0      0  
3  5.555556e+06                    0      1  
4  6.527778e+06                    0      1  
```

```
train["Family_income"].describe().round(3)
```

```
count    5.210000e+02
mean     1.579006e+06
std      1.474995e+06
min      3.244500e+05
25%      9.373500e+05
50%      1.199925e+06
75%      1.696950e+06
max      1.822500e+07
Name: Family_income, dtype: float64
```

Mapping

```
#train['Income']=train["Income"].map({"very_low":0,"low":1,"average":2
,"high":3,"very_high":4})
```

```
train.head()

   Gender  Married Dependents  Education  Self_Employed  
ApplicantIncome  \
0       1        0          0          1              0         
1316025.0   
1       1        1          1          1              0         
1031175.0   
2       1        1          0          1              1          
675000.0   
3       1        1          0          0              0          
581175.0   
4       1        0          0          1              0         
1350000.0   

   CoapplicantIncome  LoanAmount  Loan_Amount_Term Credit_History  \
0                0.0    250000.0             360.0            1.0   
1           339300.0    256000.0             360.0            1.0   
2                0.0    132000.0             360.0            1.0   
3           530550.0    240000.0             360.0            1.0   
4                0.0    282000.0             360.0            1.0   

  Property_Area  Loan_Status  Family_income  Loan_amount_per_year  \
0         Urban            1      1316025.0            694.444444   
1         Rural            0      1370475.0            711.111111   
2         Urban            1       675000.0            366.666667   
3         Urban            1      1111725.0            666.666667   
4         Urban            1      1350000.0            783.333333   

            EMI  Loan_repay_capacity Income  
0  5.787037e+06                    0      1  
1  5.925926e+06                    0      1  
2  3.055556e+06                    0      0  
3  5.555556e+06                    0      1  
4  6.527778e+06                    0      1  
```

concatenating

```
for col in cat_cols:
    train=pd.concat([train, pd.get_dummies(train[col], 
drop_first=True,prefix=col)], axis=1)

train.drop(columns=cat_cols,inplace=True)

train.head()

E:\New folder (3)\lib\site-packages\pandas\core\algorithms.py:798: 
FutureWarning: In a future version, the Index constructor will not 
infer numeric dtypes when passed object-dtype sequences (matching 
Series behavior)
  uniques = Index(uniques)
```

```
   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0        1316025.0                0.0    250000.0             360.0
1        1031175.0           339300.0    256000.0             360.0
2         675000.0                0.0    132000.0             360.0
3         581175.0           530550.0    240000.0             360.0
4        1350000.0                0.0    282000.0             360.0

   Family_income  Loan_amount_per_year           EMI
Loan_repay_capacity  \
0      1316025.0            694.444444  5.787037e+06
0
1      1370475.0            711.111111  5.925926e+06
0
2       675000.0            366.666667  3.055556e+06
0
3      1111725.0            666.666667  5.555556e+06
0
4      1350000.0            783.333333  6.527778e+06
0

  Income  Gender_1  Married_1  Dependents_1  Dependents_2
Dependents_3+  \
0      1         1          0             0             0
0
1      1         1          1             1             0
0
2      0         1          1             0             0
0
3      1         1          1             0             0
0
4      1         1          0             0             0
0

   Education_1  Self_Employed_1  Property_Area_Semiurban
Property_Area_Urban  \
0            1                0                        0
1
1            1                0                        0
0
2            1                1                        0
1
3            0                0                        0
1
4            1                0                        0
1

   Loan_Status_1  Credit_History_1.0
0              1                   1
1              0                   1
2              1                   1
```

```
3             1                   1
4             1                   1
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 521 entries, 0 to 520
Data columns (total 20 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   ApplicantIncome          521 non-null    float64
 1   CoapplicantIncome        521 non-null    float64
 2   LoanAmount               521 non-null    float64
 3   Loan_Amount_Term         521 non-null    float64
 4   Family_income            521 non-null    float64
 5   Loan_amount_per_year     521 non-null    float64
 6   EMI                      521 non-null    float64
 7   Loan_repay_capacity      521 non-null    int32
 8   Income                   521 non-null    category
 9   Gender_1                 521 non-null    uint8
 10  Married_1                521 non-null    uint8
 11  Dependents_1             521 non-null    uint8
 12  Dependents_2             521 non-null    uint8
 13  Dependents_3+            521 non-null    uint8
 14  Education_1              521 non-null    uint8
 15  Self_Employed_1          521 non-null    uint8
 16  Property_Area_Semiurban  521 non-null    uint8
 17  Property_Area_Urban      521 non-null    uint8
 18  Loan_Status_1            521 non-null    uint8
 19  Credit_History_1.0       521 non-null    uint8
dtypes: category(1), float64(7), int32(1), uint8(11)
memory usage: 37.0 KB
```

```
train.isnull().sum()
```

```
ApplicantIncome            0
CoapplicantIncome          0
LoanAmount                 0
Loan_Amount_Term           0
Family_income              0
Loan_amount_per_year       0
EMI                        0
Loan_repay_capacity        0
Income                     0
Gender_1                   0
Married_1                  0
Dependents_1               0
Dependents_2               0
Dependents_3+              0
Education_1                0
Self_Employed_1            0
```

```
Property_Area_Semiurban    0
Property_Area_Urban        0
Loan_Status_1              0
Credit_History_1.0         0
dtype: int64
```

```
train["Income"].unique()
```

```
[1, 0, 2, 3, 4]
Categories (5, int64): [0 < 1 < 2 < 3 < 4]
```

```
train["Income"]=train["Income"].astype(int)
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 521 entries, 0 to 520
Data columns (total 20 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   ApplicantIncome          521 non-null    float64
 1   CoapplicantIncome        521 non-null    float64
 2   LoanAmount               521 non-null    float64
 3   Loan_Amount_Term         521 non-null    float64
 4   Family_income            521 non-null    float64
 5   Loan_amount_per_year     521 non-null    float64
 6   EMI                      521 non-null    float64
 7   Loan_repay_capacity      521 non-null    int32  
 8   Income                   521 non-null    int32  
 9   Gender_1                 521 non-null    uint8  
 10  Married_1                521 non-null    uint8  
 11  Dependents_1             521 non-null    uint8  
 12  Dependents_2             521 non-null    uint8  
 13  Dependents_3+            521 non-null    uint8  
 14  Education_1              521 non-null    uint8  
 15  Self_Employed_1          521 non-null    uint8  
 16  Property_Area_Semiurban  521 non-null    uint8  
 17  Property_Area_Urban      521 non-null    uint8  
 18  Loan_Status_1            521 non-null    uint8  
 19  Credit_History_1.0       521 non-null    uint8  
dtypes: float64(7), int32(2), uint8(11)
memory usage: 38.3 KB
```

Scaling

StandardScaler

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
strd_train=scaler.fit_transform(train)
strd_train=pd.DataFrame(strd_train,columns=train.columns)
```

```
strd_train.head()
strd_train.describe().round(3)
```

```
       ApplicantIncome  CoapplicantIncome  LoanAmount
Loan_Amount_Term  \
count          521.000            521.000     521.000
521.000
mean            -0.000             -0.000       0.000
0.000
std              1.001              1.001       1.001
1.001
min             -0.826             -0.725      -1.635            -
5.287
25%             -0.402             -0.725      -0.489
0.260
50%             -0.256             -0.142      -0.174
0.260
75%              0.044              0.349       0.236
0.260
max             11.734              8.611       6.548
2.172

       Family_income  Loan_amount_per_year      EMI
Loan_repay_capacity  \
count        521.000               521.000  521.000
521.000
mean           0.000                -0.000   -0.000
0.000
std            1.001                 1.001    1.001
1.001
min           -0.851                -0.850   -0.850               -
0.225
25%           -0.435                -0.340   -0.340               -
0.225
50%           -0.257                -0.183   -0.183               -
0.225
75%            0.080                 0.066    0.066               -
0.225
max           11.296                16.488   16.488
4.454

        Income  Gender_1  Married_1  Dependents_1  Dependents_2  \
count  521.000   521.000    521.000       521.000       521.000
mean    -0.000     0.000     -0.000         0.000         0.000
std      1.001     1.001      1.001         1.001         1.001
min     -1.653    -2.104     -1.359        -0.438        -0.438
25%     -0.416     0.475     -1.359        -0.438        -0.438
50%     -0.416     0.475      0.736        -0.438        -0.438
75%      0.822     0.475      0.736        -0.438        -0.438
max      3.297     0.475      0.736         2.281         2.281
```

```
       Dependents_3+  Education_1  Self_Employed_1
Property_Area_Semiurban  \
count        521.000      521.000          521.000
521.000
mean          -0.000        0.000            0.000                   -
0.000
std            1.001        1.001            1.001
1.001
min           -0.292       -1.900           -0.384                   -
0.780
25%           -0.292        0.526           -0.384                   -
0.780
50%           -0.292        0.526           -0.384                   -
0.780
75%           -0.292        0.526           -0.384
1.282
max            3.422        0.526            2.603
1.282

       Property_Area_Urban  Loan_Status_1  Credit_History_1.0
count              521.000        521.000             521.000
mean                 0.000         -0.000               0.000
std                  1.001          1.001               1.001
min                 -0.714         -1.482              -2.477
25%                 -0.714         -1.482               0.404
50%                 -0.714          0.675               0.404
75%                  1.400          0.675               0.404
max                  1.400          0.675               0.404
```

strd_train

```
     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term
\
0           0.059804          -0.725361   -0.173623          0.259532

1          -0.136866          -0.021370   -0.138553          0.259532

2          -0.382781          -0.725361   -0.863334          0.259532

3          -0.447561           0.375442   -0.232073          0.259532

4           0.083262          -0.725361    0.013418          0.259532

..               ...                ...         ...               ...

516        -0.533313           0.036518   -0.313903          2.172065

517        -0.371286           0.114947   -0.197003          0.259532
```

```
518       -0.121331          0.168633    0.527779          0.259532

519       -0.320642         -0.725361   -0.524323          0.259532

520       -0.508302          0.087870   -1.108824          0.259532


     Family_income  Loan_amount_per_year       EMI  
Loan_repay_capacity  \
0        -0.178464             -0.199043 -0.199043            -
0.224507
1        -0.141513             -0.183423 -0.183423            -
0.224507
2        -0.613476             -0.506236 -0.506236            -
0.224507
3        -0.317106             -0.225076 -0.225076            -
0.224507
4        -0.155408             -0.115736 -0.115736            -
0.224507
..             ...                   ...       ...                    ..
.
516      -0.512243             -0.408611 -0.408611            -
0.224507
517      -0.327336             -0.209456 -0.209456            -
0.224507
518      -0.064100              0.113357  0.113357            -
0.224507
519      -0.552401             -0.355243 -0.355243            -
0.224507
520      -0.470864             -0.615576 -0.615576            -
0.224507

       Income  Gender_1  Married_1  Dependents_1  Dependents_2  
Dependents_3+  \
0   -0.415662  0.475271  -1.359042     -0.438429     -0.438429      -
0.292261
1   -0.415662  0.475271   0.735813      2.280873     -0.438429      -
0.292261
2   -1.653146  0.475271   0.735813     -0.438429     -0.438429      -
0.292261
3   -0.415662  0.475271   0.735813     -0.438429     -0.438429      -
0.292261
4   -0.415662  0.475271  -1.359042     -0.438429     -0.438429      -
0.292261
..        ...       ...        ...           ...           ...            
...
516 -0.415662 -2.104064   0.735813     -0.438429      2.280873      -
0.292261
517 -0.415662  0.475271   0.735813     -0.438429     -0.438429      -
0.292261
```

```
518  0.821823  0.475271  -1.359042     -0.438429     -0.438429      -
0.292261
519 -0.415662 -2.104064  -1.359042     -0.438429     -0.438429      -
0.292261
520 -0.415662  0.475271   0.735813     -0.438429      2.280873      -
0.292261

     Education_1  Self_Employed_1  Property_Area_Semiurban  \
0       0.526271        -0.384158                -0.779759
1       0.526271        -0.384158                -0.779759
2       0.526271         2.603098                -0.779759
3      -1.900163        -0.384158                -0.779759
4       0.526271        -0.384158                -0.779759
..           ...              ...                      ...
516     0.526271        -0.384158                 1.282447
517    -1.900163        -0.384158                 1.282447
518     0.526271        -0.384158                 1.282447
519    -1.900163        -0.384158                -0.779759
520    -1.900163        -0.384158                 1.282447

     Property_Area_Urban  Loan_Status_1  Credit_History_1.0
0               1.400081       0.674765            0.403666
1              -0.714244      -1.481998            0.403666
2               1.400081       0.674765            0.403666
3               1.400081       0.674765            0.403666
4               1.400081       0.674765            0.403666
..                   ...            ...                 ...
516            -0.714244       0.674765            0.403666
517            -0.714244      -1.481998           -2.477294
518            -0.714244      -1.481998            0.403666
519            -0.714244      -1.481998            0.403666
520            -0.714244       0.674765            0.403666

[521 rows x 20 columns]
```

MinMaxScaler

```python
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
strd_train1=scaler.fit_transform(train)
strd_train1=pd.DataFrame(strd_train1,columns=train.columns)
strd_train1.head()
strd_train1.describe().round(3)
```

```
       ApplicantIncome  CoapplicantIncome  LoanAmount
Loan_Amount_Term  \
count          521.000            521.000     521.000
521.000
mean             0.066              0.078       0.200
0.709
std              0.080              0.107       0.122
```

```
0.134
min               0.000              0.000       0.000
0.000
25%               0.034              0.000       0.140
0.744
50%               0.045              0.062       0.179
0.744
75%               0.069              0.115       0.229
0.744
max               1.000              1.000       1.000
1.000

       Family_income  Loan_amount_per_year      EMI
Loan_repay_capacity  \
count        521.000               521.000  521.000
521.000
mean           0.070                 0.049    0.049
0.048
std            0.082                 0.058    0.058
0.214
min            0.000                 0.000    0.000
0.000
25%            0.034                 0.029    0.029
0.000
50%            0.049                 0.038    0.038
0.000
75%            0.077                 0.053    0.053
0.000
max            1.000                 1.000    1.000
1.000

        Income  Gender_1  Married_1  Dependents_1  Dependents_2  \
count  521.000   521.000    521.000       521.000       521.000
mean     0.334     0.816      0.649         0.161         0.161
std      0.202     0.388      0.478         0.368         0.368
min      0.000     0.000      0.000         0.000         0.000
25%      0.250     1.000      0.000         0.000         0.000
50%      0.250     1.000      1.000         0.000         0.000
75%      0.500     1.000      1.000         0.000         0.000
max      1.000     1.000      1.000         1.000         1.000

       Dependents_3+  Education_1  Self_Employed_1
Property_Area_Semiurban  \
count        521.000      521.000          521.000
521.000
mean           0.079        0.783            0.129
0.378
std            0.270        0.413            0.335
0.485
min            0.000        0.000            0.000
```

```
0.000
25%            0.000        1.000           0.000
0.000
50%            0.000        1.000           0.000
0.000
75%            0.000        1.000           0.000
1.000
max            1.000        1.000           1.000
1.000

       Property_Area_Urban  Loan_Status_1  Credit_History_1.0
count              521.000        521.000             521.000
mean                 0.338          0.687               0.860
std                  0.473          0.464               0.347
min                  0.000          0.000               0.000
25%                  0.000          0.000               1.000
50%                  0.000          1.000               1.000
75%                  1.000          1.000               1.000
max                  1.000          1.000               1.000
```

strd_train1

```
     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term
\
0           0.070489            0.00000    0.178571           0.74359

1           0.054830            0.07540    0.182857           0.74359

2           0.035250            0.00000    0.094286           0.74359

3           0.030093            0.11790    0.171429           0.74359

4           0.072356            0.00000    0.201429           0.74359

..               ...                ...         ...               ...

516         0.023265            0.08160    0.161429           1.00000

517         0.036166            0.09000    0.175714           0.74359

518         0.056067            0.09575    0.264286           0.74359

519         0.040198            0.00000    0.135714           0.74359

520         0.025257            0.08710    0.064286           0.74359


     Family_income  Loan_amount_per_year       EMI
Loan_repay_capacity  \
```

```
0         0.055394              0.037538  0.037538
0.0
1         0.058435              0.038438  0.038438
0.0
2         0.019583              0.019820  0.019820
0.0
3         0.043980              0.036036  0.036036
0.0
4         0.057292              0.042342  0.042342
0.0
..             ...                   ...       ...                    ..
.
516       0.027917              0.025450  0.025450
0.0
517       0.043138              0.036937  0.036937
0.0
518       0.064808              0.055556  0.055556
0.0
519       0.024611              0.028529  0.028529
0.0
520       0.031323              0.013514  0.013514
0.0

     Income  Gender_1  Married_1  Dependents_1  Dependents_2
Dependents_3+  \
0      0.25       1.0        0.0           0.0           0.0
0.0
1      0.25       1.0        1.0           1.0           0.0
0.0
2      0.00       1.0        1.0           0.0           0.0
0.0
3      0.25       1.0        1.0           0.0           0.0
0.0
4      0.25       1.0        0.0           0.0           0.0
0.0
..      ...       ...        ...           ...           ...
...
516    0.25       0.0        1.0           0.0           1.0
0.0
517    0.25       1.0        1.0           0.0           0.0
0.0
518    0.50       1.0        0.0           0.0           0.0
0.0
519    0.25       0.0        0.0           0.0           0.0
0.0
520    0.25       1.0        1.0           0.0           1.0
0.0

     Education_1  Self_Employed_1  Property_Area_Semiurban  \
0            1.0              0.0                      0.0
```

```
1              1.0              0.0                        0.0
2              1.0              1.0                        0.0
3              0.0              0.0                        0.0
4              1.0              0.0                        0.0
..             ...              ...                        ...
516            1.0              0.0                        1.0
517            0.0              0.0                        1.0
518            1.0              0.0                        1.0
519            0.0              0.0                        0.0
520            0.0              0.0                        1.0

     Property_Area_Urban  Loan_Status_1  Credit_History_1.0
0                    1.0            1.0                 1.0
1                    0.0            0.0                 1.0
2                    1.0            1.0                 1.0
3                    1.0            1.0                 1.0
4                    1.0            1.0                 1.0
..                   ...            ...                 ...
516                  0.0            1.0                 1.0
517                  0.0            0.0                 0.0
518                  0.0            0.0                 1.0
519                  0.0            0.0                 1.0
520                  0.0            1.0                 1.0

[521 rows x 20 columns]
```

Milestone:4

Things to do:

1. Make 5 classification models
2. check train and test accuracy
3. check for overfitting and underfitting,if it is there,do hyperparameter tuning
4. Make evaluation matrix
5. select the top 3 model and justify it.

Classification is a supervised learning task in machine learning where the goal is to predict the class label of a given input data point. There are several classification algorithms in ML, some of which are listed below:

Decision Tree: It is a tree-based model that splits the data based on the most significant attribute that maximizes the information gain. The decision tree is easy to interpret and suitable for handling both categorical and numerical data.

Random Forest: It is an ensemble learning algorithm that combines multiple decision trees to make a final prediction. Random forests reduce overfitting and improve accuracy by combining multiple decision trees.

Support Vector Machines (SVM): It is a linear classification algorithm that finds the best separating hyperplane between classes. SVM is suitable for handling high-dimensional data and works well when there is a clear margin of separation between the classes.

Naive Bayes: It is a probabilistic classifier that is based on Bayes' theorem. Naive Bayes assumes that the features are independent of each other, which makes it suitable for handling large datasets with many features.

K-Nearest Neighbors (KNN): It is a lazy learning algorithm that uses a distance metric to classify data points based on the k-nearest neighbors in the training data. KNN is simple to implement and works well with small datasets.

Logistic Regression: It is a linear classification algorithm that models the probability of the class label given the input features. Logistic regression is a popular algorithm for binary classification problems.

```
strd_train1
```

```
     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term
\
0           0.070489            0.00000    0.178571           0.74359

1           0.054830            0.07540    0.182857           0.74359

2           0.035250            0.00000    0.094286           0.74359

3           0.030093            0.11790    0.171429           0.74359

4           0.072356            0.00000    0.201429           0.74359

..               ...                ...         ...               ...

516         0.023265            0.08160    0.161429           1.00000

517         0.036166            0.09000    0.175714           0.74359

518         0.056067            0.09575    0.264286           0.74359

519         0.040198            0.00000    0.135714           0.74359

520         0.025257            0.08710    0.064286           0.74359


     Family_income  Loan_amount_per_year       EMI
Loan_repay_capacity  \
0         0.055394              0.037538  0.037538
0.0
1         0.058435              0.038438  0.038438
0.0
```

```
2         0.019583              0.019820  0.019820
0.0
3         0.043980              0.036036  0.036036
0.0
4         0.057292              0.042342  0.042342
0.0
..             ...                   ...       ...                  ..
.
516       0.027917              0.025450  0.025450
0.0
517       0.043138              0.036937  0.036937
0.0
518       0.064808              0.055556  0.055556
0.0
519       0.024611              0.028529  0.028529
0.0
520       0.031323              0.013514  0.013514
0.0

     Income  Gender_1  Married_1  Dependents_1  Dependents_2
Dependents_3+  \
0      0.25       1.0        0.0           0.0           0.0
0.0
1      0.25       1.0        1.0           1.0           0.0
0.0
2      0.00       1.0        1.0           0.0           0.0
0.0
3      0.25       1.0        1.0           0.0           0.0
0.0
4      0.25       1.0        0.0           0.0           0.0
0.0
..      ...       ...        ...           ...           ...
...
516    0.25       0.0        1.0           0.0           1.0
0.0
517    0.25       1.0        1.0           0.0           0.0
0.0
518    0.50       1.0        0.0           0.0           0.0
0.0
519    0.25       0.0        0.0           0.0           0.0
0.0
520    0.25       1.0        1.0           0.0           1.0
0.0

     Education_1  Self_Employed_1  Property_Area_Semiurban  \
0            1.0              0.0                      0.0
1            1.0              0.0                      0.0
2            1.0              1.0                      0.0
3            0.0              0.0                      0.0
4            1.0              0.0                      0.0
```

```
..          ...              ...                        ...
516         1.0              0.0                        1.0
517         0.0              0.0                        1.0
518         1.0              0.0                        1.0
519         0.0              0.0                        0.0
520         0.0              0.0                        1.0

     Property_Area_Urban  Loan_Status_1  Credit_History_1.0
0                    1.0            1.0                 1.0
1                    0.0            0.0                 1.0
2                    1.0            1.0                 1.0
3                    1.0            1.0                 1.0
4                    1.0            1.0                 1.0
..                   ...            ...                 ...
516                  0.0            1.0                 1.0
517                  0.0            0.0                 0.0
518                  0.0            0.0                 1.0
519                  0.0            0.0                 1.0
520                  0.0            1.0                 1.0

[521 rows x 20 columns]
```

Target variable : Loan_Status_1

Train_Test Split

```
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier


# Warning filter library
import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split

X = strd_train1.drop('Loan_Status_1', axis=1)
y = strd_train1.Loan_Status_1

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

y_test
```

```
507    0.0
93     1.0
6      1.0
245    0.0
```

```
90     1.0
      ...
508    1.0
24     0.0
17     0.0
247    1.0
66     0.0
Name: Loan_Status_1, Length: 157, dtype: float64
```

```python
def train_and_test_split(data,t_col,testsize = 0.3,randomstate = 3):
    x = data.drop(t_col,axis = 1)
    y = data[t_col]
    return train_test_split(x,y,test_size = testsize,random_state =
randomstate)

def model_builder(model_name,model,data,t_col,train):
    x_train,x_test,y_train,y_test = train_and_test_split(data,t_col)
    model.fit(x_train,y_train)
    if train==True:
        y_pred = model.predict(X_train)
        accuracy = accuracy_score(y_train, y_pred)
        precision = precision_score(y_train, y_pred)
        recall = recall_score(y_train, y_pred)
        f1 = f1_score(y_train, y_pred)
        auc_roc = roc_auc_score(y_train, y_pred)
        cm = confusion_matrix(y_train, y_pred)
       # print("Accuracy_train",accuracy_train)
        print("Model_Name:",model_name)
        print("\n")
        print("Train Data Report")
        print("\n")
        print("Accuracy     :", accuracy)
        print("Precision    :", precision)
        print("Recall       :", recall)
        print("F1 score     :", f1)
        print("AUC-ROC score:", auc_roc)
        #print("Confusion Matrix:",cm)

        print ("Confusion Matrix : \n", cm)



    elif train==False:
        y_pred = model.predict(X_test)
        accuracy_test = accuracy_score(y_test, y_pred)
        precision_test = precision_score(y_test, y_pred)
        recall_test = recall_score(y_test, y_pred)
        f1_test = f1_score(y_test, y_pred)
        auc_roc_test = roc_auc_score(y_test, y_pred)
        cm_test = confusion_matrix(y_test, y_pred)
```

```
        # print("Accuracy_train",accuracy_train)
        print("\n")
        print("Test Data Report:")
        print("\n")
        print("Accuracy     :", accuracy_test)
        print("Precision    :", precision_test)
        print("Recall       :", recall_test)
        print("F1 score     :", f1_test)
        print("AUC-ROC score:", auc_roc_test)
        #print("Confusion Matrix:",cm)
        print ("Confusion Matrix : \n", cm_test)




        # result = [accuracy,precision,recall,f1,auc_roc,cm]
    # return result



model_builder(model_name = 'LogisticRegression',model =
LogisticRegression(),data = strd_train1,t_col =
'Loan_Status_1',train=True)
model_builder(model_name = 'LogisticRegression',model =
LogisticRegression(),data = strd_train1,t_col =
'Loan_Status_1',train=False)
print("\n")
model_builder(model_name = 'DecisionTreeClassifier',model =
DecisionTreeClassifier(),data = strd_train1,t_col =
'Loan_Status_1',train=True)
model_builder(model_name = 'DecisionTreeClassifier',model =
DecisionTreeClassifier(),data = strd_train1,t_col =
'Loan_Status_1',train=False)
print("\n")
model_builder(model_name = 'SVC',model = SVC(),data =
strd_train1,t_col = 'Loan_Status_1',train=True)
model_builder(model_name = 'SVC',model = SVC(),data =
strd_train1,t_col = 'Loan_Status_1',train=False)
print("\n")
model_builder(model_name = 'RandomForestClassifier',model =
RandomForestClassifier(),data = strd_train1,t_col =
'Loan_Status_1',train=True)
model_builder(model_name = 'RandomForestClassifier',model =
RandomForestClassifier(),data = strd_train1,t_col =
'Loan_Status_1',train=False)
print("\n")
model_builder(model_name = 'KNeighborsClassifier',model =
KNeighborsClassifier(),data = strd_train1,t_col =
'Loan_Status_1',train=True)
model_builder(model_name = 'KNeighborsClassifier',model =
```

```
KNeighborsClassifier(),data = strd_train1,t_col =
'Loan_Status_1',train=False)
print("\n")
model_builder(model_name = 'XGBClassifier',model =
XGBClassifier(),data = strd_train1,t_col = 'Loan_Status_1',train=True)
model_builder(model_name = 'XGBClassifier',model =
XGBClassifier(),data = strd_train1,t_col =
'Loan_Status_1',train=False)
```

```
Model_Name: LogisticRegression


Train Data Report


Accuracy     : 0.6401098901098901
Precision    : 0.6845425867507886
Recall       : 0.875
F1 score     : 0.7681415929203539
AUC-ROC score: 0.5064655172413793
Confusion Matrix :
 [[ 16 100]
 [ 31 217]]


Test Data Report:


Accuracy     : 0.6815286624203821
Precision    : 0.7307692307692307
Recall       : 0.8636363636363636
F1 score     : 0.7916666666666666
AUC-ROC score: 0.559477756286267
Confusion Matrix :
 [[12 35]
 [15 95]]


Model_Name: DecisionTreeClassifier


Train Data Report


Accuracy     : 0.5384615384615384
Precision    : 0.6538461538461539
Recall       : 0.6854838709677419
F1 score     : 0.6692913385826771
```

```
AUC-ROC score: 0.4548109010011123
Confusion Matrix :
 [[ 26  90]
 [ 78 170]]


Test Data Report:


Accuracy     : 0.6305732484076433
Precision    : 0.75
Recall       : 0.7090909090909091
F1 score     : 0.7289719626168225
AUC-ROC score: 0.5779497098646035
Confusion Matrix :
 [[21 26]
 [32 78]]


Model_Name: SVC


Train Data Report


Accuracy     : 0.6428571428571429
Precision    : 0.6855345911949685
Recall       : 0.8790322580645161
F1 score     : 0.7703180212014133
AUC-ROC score: 0.5084816462736373
Confusion Matrix :
 [[ 16 100]
 [ 30 218]]


Test Data Report:


Accuracy     : 0.6815286624203821
Precision    : 0.7307692307692307
Recall       : 0.8636363636363636
F1 score     : 0.7916666666666666
AUC-ROC score: 0.559477756286267
Confusion Matrix :
 [[12 35]
 [15 95]]


Model_Name: RandomForestClassifier
```

```
Train Data Report


Accuracy     : 0.5741758241758241
Precision    : 0.6715867158671587
Recall       : 0.7338709677419355
F1 score     : 0.7013487475915221
AUC-ROC score: 0.4833147942157954
Confusion Matrix :
 [[ 27  89]
 [ 66 182]]


Test Data Report:


Accuracy     : 0.6305732484076433
Precision    : 0.7452830188679245
Recall       : 0.7181818181818181
F1 score     : 0.7314814814814815
AUC-ROC score: 0.5718568665377176
Confusion Matrix :
 [[20 27]
 [31 79]]


Model_Name: KNeighborsClassifier


Train Data Report


Accuracy     : 0.6043956043956044
Precision    : 0.6733333333333333
Recall       : 0.8145161290322581
F1 score     : 0.7372262773722629
AUC-ROC score: 0.4848442714126807
Confusion Matrix :
 [[ 18  98]
 [ 46 202]]


Test Data Report:


Accuracy     : 0.6496815286624203
```

```
Precision    : 0.72
Recall       : 0.8181818181818182
F1 score     : 0.7659574468085107
AUC-ROC score: 0.5367504835589942
Confusion Matrix :
 [[12 35]
 [20 90]]


Model_Name: XGBClassifier


Train Data Report


Accuracy     : 0.5604395604395604
Precision    : 0.6654135338345865
Recall       : 0.7137096774193549
F1 score     : 0.688715953307393
AUC-ROC score: 0.473234149054505
Confusion Matrix :
 [[ 27  89]
 [ 71 177]]


Test Data Report:


Accuracy     : 0.6242038216560509
Precision    : 0.7524752475247525
Recall       : 0.6909090909090909
F1 score     : 0.7203791469194313
AUC-ROC score: 0.5794970986460348
Confusion Matrix :
 [[22 25]
 [34 76]]
```

Cross Validation Test

```
def K_fold_CV(x,y,fold = 10):
    score_las = cross_val_score(Lasso(),x,y,cv = fold)
    score_rd = cross_val_score(Ridge(),x,y,cv = fold)
    score_dtr = cross_val_score(DecisionTreeClassifier(),x,y,cv =
fold)
    score_svc = cross_val_score(SVC(),x,y,cv = fold)
    score_rf = cross_val_score(RandomForestClassifier(),x,y,cv = fold)
    score_knn = cross_val_score(KNeighborsClassifier(),x,y,cv = fold)
    score_xgb = cross_val_score(XGBClassifier(),x,y,cv = fold)

    model_name =
```

```
["LogisticRegression","DecisionTreeClassifier","SVC","RandomForestClas
sifier","KNeighborsClassifier","XGBClassifier"]

    scores = 
[score_las,score_rd,score_dtr,score_svc,score_rf,score_knn,score_xgb]
    result = []
    for i in range(len(model_name)):
        score_mean = np.mean(scores[i])
        score_std = np.std(scores[i])
        m_name = model_name[i]
        temp = [m_name,score_mean,score_std]
        result.append(temp)
    k_fold_df = pd.DataFrame(result,columns=["Model Name","CV 
accuracy"," CV Std"])
    return k_fold_df.sort_values("CV accuracy",ascending=False)
```

```
K_fold_CV(strd_train1.drop("Loan_Status_1",axis = 
1),strd_train1["Loan_Status_1"])
```

```
               Model Name  CV accuracy    CV Std
3  RandomForestClassifier     0.804245  0.038127
4    KNeighborsClassifier     0.779318  0.040255
5           XGBClassifier     0.754282  0.034406
2                     SVC     0.717779  0.035971
1  DecisionTreeClassifier     0.269316  0.111372
0      LogisticRegression    -0.009444  0.009316
```

Hyperparameter tuning

```
"""""def tuning(X,y,fold = 10):
    #parameter grids
    param_las = {"alpha":[1e-15,1e-13,1e-11,1e-9,1e-7,1e-5,1e-3,1e-
1,0,1,2,3,4,5,6,7,8,9,10,20,30,40,50,60,70,80,90,100,200,300,400,500]}
    param_knn = {"n_neighbors" : 
[1,2,3,4,5,6,7,8,9,10,20,30,40,50,60,70,80,90,100]}
    param_dtr = {"max_depth" : [3,5,7,9,10,12,14,16],"max_features" : 
["auto","log2",'sqrt',2,3,4,5,6]}
    param_svc = {"gamma" : ["scale",'auto'],"C" : [0.5,1]}
    param_rf = {"max_depth" : [3,5,7,9,10,12,14,16],"max_features" : 
["auto","log2","sqrt",2,3,4,5,6] }
    param_xgb = {"eta" : [0.1,0.2,0.3,0.4,0.5],"max_depth": 
[3,5,7,9,10,12,14,16],"gamma" : 
[0,1,2,3,4,5,6,7,8,9,10,20,30,40,50,60,70,80,90,100,200,300,400,500],"
reg_lambda" :[0,1] }
```

```
    # Hyper-paameter tuning

    tune_las = GridSearchCV(Lasso(),param_las,cv = fold)
    tune_rid = GridSearchCV(Ridge(),param_las,cv = fold)
    tune_knn = GridSearchCV(KNeighborsClassifier(),param_knn,cv =
fold)
    tune_dtr = GridSearchCV(DecisionTreeClassifier(),param_dtr,cv =
fold)
    tune_svc = GridSearchCV(SVC(),param_svc,cv = fold)
    tune_rf = GridSearchCV(RandomForestClassifier(),param_rf,cv =
fold)
    tune_xgb = GridSearchCV(XGBClassifier(),param_xgb,cv = fold)


    # Fitting X,y

    tune_las.fit(X,y)
    tune_rid.fit(X,y)
    tune_knn.fit(X,y)
    tune_dtr.fit(X,y)
    tune_svc.fit(X,y)
    tune_rf.fit(X,y)
    tune_xgb.fit(X,y)


    tune = [tune_las,tune_rid,tune_knn,tune_dtr,tune_svc,tune_rf]
    models =
["Lasso","Ridge","KNeighborsClassifier","DecisionTreeClassifier","SVC"
,"RandomForestClassifier","XGBClassifier"]
    for i in range(len(tune)):
        print("Models : ",models[i])
        print("Best Parameter :",tune[i].best_params_)
        """"""""""


  Input In [57]
    """"""""""

^
SyntaxError: EOL while scanning string literal


#tuning(strd_train1.drop("Loan_Status_1",axis =
1),strd_train1["Loan_Status_1"])
```

Cross-Validation post hyper-parameters tuning

```
def cv_post_hpt(x,y,fold = 10):
    #score_lr = cross_val_score(LogisticRegression(),x,y,cv = fold)
    score_ls = cross_val_score(Lasso(alpha = 0.001),x,y,cv = fold)
```

```
    score_rd = cross_val_score(Ridge(alpha = 7),x,y,cv = fold)
    score_dtr = cross_val_score(DecisionTreeClassifier(max_depth =
5),x,y,cv = fold)
    score_svc = cross_val_score(SVC(C = 0.5),x,y,cv = fold)
    score_rf = cross_val_score(RandomForestClassifier(max_depth =
3),x,y,cv = fold)
    score_knn = cross_val_score(KNeighborsClassifier(n_neighbors =
5),x,y,cv = fold)
    score_xgb = cross_val_score(XGBClassifier(eta = 0.2, gamma= 4,
max_depth= 9, reg_lambda= 1),x,y,cv = fold)



    model_name =
["Lasso","Ridge","DecisionTreeClassifier","SVC","RandomForestClassifie
r","KNeighborsClassifier","XGBClassifier"]

    scores =
[score_ls,score_rd,score_dtr,score_svc,score_rf,score_knn,score_xgb]
    result = []
    for i in range(len(model_name)):
        score_mean = np.mean(scores[i])
        score_std = np.std(scores[i])
        m_name = model_name[i]
        temp = [m_name,score_mean,score_std]
        result.append(temp)
    k_fold_df = pd.DataFrame(result,columns=["Model Name","CV
accuracy"," CV Std"])
    return k_fold_df.sort_values("CV accuracy",ascending=False)



cv_post_hpt(strd_train1.drop("Loan_Status_1",axis =
1),strd_train1["Loan_Status_1"])
```

	Model Name	CV accuracy	CV Std
6	XGBClassifier	0.806168	0.037765
3	SVC	0.804245	0.038127
4	RandomForestClassifier	0.800399	0.033414
2	DecisionTreeClassifier	0.769775	0.051735
5	KNeighborsClassifier	0.754282	0.034406
0	Lasso	0.276899	0.111448
1	Ridge	0.275199	0.100805

```
 """""""""" def tuning(X,y,fold = 10):
    param_xgb = {"eta" : [0.1,0.2,0.3,0.4,0.5],"max_depth":
[3,5,7,9,10,12,14,16],"gamma" :
[0,1,2,3,4,5,6,7,8,9,10,20,30,40,50,60,70,80,90,100,200,300,400,500],"
reg_lambda" :[0,1] }
    tune_xgb = GridSearchCV(XGBClassifier(),param_xgb,cv = fold)
```

```
    tune_xgb.fit(X,y)
    tune = [tune_xgb]
    models = ["XGBClassifier"]
    for i in range(len(tune)):
        print("Models : ",models[i])
        print("Best Parameter :",tune[i].best_params_)"""""""""""
```

```
#tuning(strd_train1.drop("Loan_Status_1",axis =
1),strd_train1["Loan_Status_1"])
```

Testing Data preprocessing

```
strd_test1=pd.read_csv("strd_test1.csv")
strd_test1
```

```
    ApplicantIncome  CoapplicantIncome  LoanAmount
Loan_Amount_Term  \
0          0.109293           0.000000    0.091667          0.729730

1          0.275907           0.034176    0.166667          0.729730

2          0.395007           0.171983    0.800000          0.729730

3          0.223621           0.050088    0.000000          0.729730

4          0.895951           0.000000    0.666667          0.729730

..              ...                ...         ...               ...

88         0.130271           0.000000    0.118333          0.729730

89         0.193518           0.000000    0.066667          0.324324

90         0.401510           0.005760    0.421667          0.729730

91         0.375865           0.000000    0.311667          0.729730

92         0.218534           0.000000    0.221667          0.729730


    Family_income  Loan_amount_per_year      EMI  Loan_repay_capacity
Income  \
0        0.015422               0.06875  0.06875                  0.0
0.00
1        0.129866               0.12500  0.12500                  0.0
0.50
2        0.329179               0.60000  0.60000                  0.0
0.75
3        0.121558               0.00000  0.00000                  1.0
0.50
```

```
4       0.388528              0.50000  0.50000                  0.0
0.75
..           ...                  ...      ...                  ...
...
88      0.025371              0.08875  0.08875                  0.0
0.00
89      0.055369              0.10000  0.10000                  0.0
0.25
90      0.159988              0.31625  0.31625                  0.0
0.50
91      0.141855              0.23375  0.23375                  0.0
0.50
92      0.067234              0.16625  0.16625                  0.0
0.25

    Gender_1  Married_1  Dependents_1  Dependents_2  Dependents_3+  \
0        1.0        0.0           0.0           0.0            0.0
1        1.0        1.0           0.0           0.0            1.0
2        1.0        1.0           0.0           1.0            0.0
3        1.0        0.0           0.0           0.0            0.0
4        1.0        1.0           0.0           1.0            0.0
..       ...        ...           ...           ...            ...
88       0.0        0.0           0.0           0.0            0.0
89       1.0        1.0           0.0           0.0            1.0
90       1.0        1.0           1.0           0.0            0.0
91       1.0        1.0           0.0           1.0            0.0
92       0.0        0.0           0.0           0.0            0.0

    Education_1  Self_Employed_1  Property_Area_Semiurban  \
0           1.0              0.0                      1.0
1           1.0              1.0                      0.0
2           1.0              1.0                      0.0
3           1.0              0.0                      1.0
4           1.0              1.0                      0.0
..          ...              ...                      ...
88          1.0              0.0                      0.0
89          1.0              0.0                      0.0
90          1.0              0.0                      0.0
91          1.0              0.0                      0.0
92          1.0              1.0                      1.0

    Property_Area_Urban  Credit_History_1.0
0                   0.0                 1.0
1                   0.0                 1.0
2                   0.0                 1.0
3                   0.0                 1.0
4                   0.0                 1.0
..                  ...                 ...
88                  0.0                 1.0
89                  0.0                 1.0
```

```
90                   1.0                   1.0
91                   1.0                   1.0
92                   0.0                   0.0

[93 rows x 19 columns]
```

```
def model_builder(model_name,model,data,):
    x_train,x_test,y_train,y_test = train_and_test_split(data,t_col)
    model.fit(x_train,y_train)
    y_pred = model.predict(strd_test1)

    # print the predicted values
    print("Predicted values: ", y_pred)

    # Evaluate the performance

   # return result


model_builder(model_name = 'SVC',model = SVC(),data =
strd_train1,t_col = 'Loan_Status_1',train=True)
model_builder(model_name = 'SVC',model = SVC(),data =
strd_train1,t_col = 'Loan_Status_1',train=False)
print("\n")
model_builder(model_name = 'RandomForestClassifier',model =
RandomForestClassifier(),data = strd_train1,t_col =
'Loan_Status_1',train=True)
model_builder(model_name = 'RandomForestClassifier',model =
RandomForestClassifier(),data = strd_train1,t_col =
'Loan_Status_1',train=False)

print("\n")
model_builder(model_name = 'XGBClassifier',model =
XGBClassifier(),data = strd_train1,t_col = 'Loan_Status_1',train=True)
model_builder(model_name = 'XGBClassifier',model =
XGBClassifier(),data = strd_train1,t_col =
'Loan_Status_1',train=False)
```

```
----------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
Input In [63], in <cell line: 1>()
----> 1 model_builder(model_name = 'SVC',model = SVC(),data =
strd_train1,t_col = 'Loan_Status_1',train=True)
      2 model_builder(model_name = 'SVC',model = SVC(),data =
strd_train1,t_col = 'Loan_Status_1',train=False)
      3 print("\n")
```

```
TypeError: model_builder() got an unexpected keyword argument 'train'
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# load dataset
strd_test1 = pd.read_csv('strd_test1.csv')

# split dataset into train and test sets
X_test = strd_test1




# predict the target values for the test set using the trained model
y_pred = SVC().predict( X_test )

# print the predicted values
print("Predicted values: ", y_pred)
```

```
----------------------------------------------------------------------
-----
NotFittedError                            Traceback (most recent call
last)
Input In [72], in <cell line: 15>()
      9 X_test = strd_test1
     14 # predict the target values for the test set using the trained
model
---> 15 y_pred = SVC().predict( X_test )
     17 # print the predicted values
     18 print("Predicted values: ", y_pred)

File E:\New folder (3)\lib\site-packages\sklearn\svm\_base.py:778, in
BaseSVC.predict(self, X)
    761 def predict(self, X):
    762     """Perform classification on samples in X.
    763
    764     For an one-class model, +1 or -1 is returned.
   (...)
    776         Class labels for samples in X.
    777     """
--> 778     check_is_fitted(self)
    779     if self.break_ties and self.decision_function_shape ==
"ovo":
    780         raise ValueError(
    781             "break_ties must be False when
decision_function_shape is 'ovo'"
    782         )
```

```
File E:\New folder (3)\lib\site-packages\sklearn\utils\
validation.py:1222, in check_is_fitted(estimator, attributes, msg,
all_or_any)
   1217     fitted = [
   1218         v for v in vars(estimator) if v.endswith("_") and not
v.startswith("__")
   1219     ]
   1221 if not fitted:
-> 1222     raise NotFittedError(msg % {"name":
type(estimator).__name__})

NotFittedError: This SVC instance is not fitted yet. Call 'fit' with
appropriate arguments before using this estimator.
```

```python
import pandas as pd
from sklearn.linear_model import LogisticRegression

# load dataset
df = pd.read_csv('data.csv')

# train a logistic regression model on the entire dataset
X = df.drop('target', axis=1)
y = df['target']
lr_model = LogisticRegression()
lr_model.fit(X, y)

# predict the target for a new observation
new_observation = [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]]  # example data
predicted_target = lr_model.predict(new_observation)

# print the predicted target
print("Predicted target: ", predicted_target)
```

```python
    x_train,x_test,y_train,y_test =
train_and_test_split(strd_train1,Loan_Status_1)
    model.fit(x_train,y_train)
    y_pred = SVC().predict(strd_test1)
    print ("Predicted value is  : \n", y_pred)




        # result = [accuracy,precision,recall,f1,auc_roc,cm]
   # return result
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [74], in <cell line: 1>()
----> 1 x_train,x_test,y_train,y_test = train_and_test_split(strd_train1,Loan_Status_1)
      2 model.fit(x_train,y_train)
      3 y_pred = SVC().predict(strd_test1)

NameError: name 'Loan_Status_1' is not defined
```

```python
def predict1(model_name,model,data,t_col):
    x_train,x_test,y_train,y_test = train_and_test_split(data,t_col)
    model.fit(x_train,y_train)
    y_pred1 = model.predict(strd_test1)
    print("Model_name  :",model_name)
    print("prdicted value is :",y_pred1)




predict1(model_name = 'SVC',model = SVC(),data = strd_train1,t_col = 'Loan_Status_1')


print("\n")

predict1(model_name = 'RandomForestClassifier',model = RandomForestClassifier(),data = strd_train1,t_col = 'Loan_Status_1')


print("\n")
predict1(model_name = 'XGBClassifier',model = XGBClassifier(),data = strd_train1,t_col = 'Loan_Status_1')
```

```
Model_name  : SVC
prdicted value is : [1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1.
 1. 1. 1. 0. 1. 0. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1.
 0. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1.
 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]


Model_name  : RandomForestClassifier
```

```
prdicted value is : [1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1.
1. 1. 1. 1. 0. 1. 1. 1.
 1. 1. 1. 0. 1. 0. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 0.
0.
 0. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0. 1. 1.
0.
 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]


Model_name  : XGBClassifier
prdicted value is : [1 1 1 1 1 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1
1 1 0 1 0 1 1 0 0 1 1 1
 1 1 1 0 1 1 0 1 1 0 1 0 1 0 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 0
1 1
 1 1 0 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 0]
```