Credit Risk Modelling

Importing necessary libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import chi2_contingency
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split
from sklearn.ensemble import  RandomForestClassifier
from sklearn.metrics import accuracy_score,r2_score,classification_report,confusion_matrix,precision_score,recall_score,f1_score
import warnings
import os
import time
```

```
print('Program is running...')
print()
start_time=time.time()
```

```
Program is running...
```

Load the dataset

```
data_1=pd.read_csv('/content/case_study1.xlsx - case_study1.csv')
data_2=pd.read_csv('/content/case_study2.xlsx - case_study2.csv')
```

```
data_1.head()
```

	PROSPECTID	Total_TL	Tot_Closed_TL	Tot_Active_TL	Total_TL_opened_L6M	Tot_TL_closed_L6M	pct_tl_open_L6M	pct_tl_closed_L6
0	1	5	4	1	0	0	0.000	0.
1	2	1	0	1	0	0	0.000	0.
2	3	8	0	8	1	0	0.125	0.
3	4	1	0	1	1	0	1.000	0.
4	5	3	2	1	0	0	0.000	0.

5 rows × 26 columns

```
data_2.head()
```

	PROSPECTID	time_since_recent_payment	time_since_first_deliquency	time_since_recent_deliquency	num_times_delinquent	max_c
0	1	549	35	15	11	
1	2	47	-99999	-99999	0	
2	3	302	11	3	9	
3	4	-99999	-99999	-99999	0	
4	5	583	-99999	-99999	0	

5 rows × 62 columns

```
data_1.describe()
```

	PROSPECTID	Total_TL	Tot_Closed_TL	Tot_Active_TL	Total_TL_opened_L6M	Tot_TL_closed_L6M	pct_tl_open_L6M	pct_tl_
count	51336.000000	51336.000000	51336.000000	51336.000000	51336.000000	51336.000000	51336.000000	51
mean	25668.500000	4.858598	2.770415	2.088184	0.736754	0.428919	0.184574	
std	14819.571046	7.177116	5.941680	2.290774	1.296717	0.989972	0.297414	
min	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	12834.750000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	
50%	25668.500000	2.000000	1.000000	1.000000	0.000000	0.000000	0.000000	
75%	38502.250000	5.000000	3.000000	3.000000	1.000000	1.000000	0.308000	
max	51336.000000	235.000000	216.000000	47.000000	27.000000	19.000000	1.000000	

8 rows × 26 columns

```
data_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51336 entries, 0 to 51335
Data columns (total 26 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   PROSPECTID            51336 non-null  int64
 1   Total_TL              51336 non-null  int64
 2   Tot_Closed_TL         51336 non-null  int64
 3   Tot_Active_TL         51336 non-null  int64
 4   Total_TL_opened_L6M   51336 non-null  int64
 5   Tot_TL_closed_L6M     51336 non-null  int64
 6   pct_tl_open_L6M       51336 non-null  float64
 7   pct_tl_closed_L6M     51336 non-null  float64
 8   pct_active_tl         51336 non-null  float64
 9   pct_closed_tl         51336 non-null  float64
 10  Total_TL_opened_L12M  51336 non-null  int64
 11  Tot_TL_closed_L12M    51336 non-null  int64
 12  pct_tl_open_L12M      51336 non-null  float64
 13  pct_tl_closed_L12M    51336 non-null  float64
 14  Tot_Missed_Pmnt       51336 non-null  int64
 15  Auto_TL               51336 non-null  int64
 16  CC_TL                 51336 non-null  int64
 17  Consumer_TL           51336 non-null  int64
 18  Gold_TL               51336 non-null  int64
 19  Home_TL               51336 non-null  int64
 20  PL_TL                 51336 non-null  int64
 21  Secured_TL            51336 non-null  int64
 22  Unsecured_TL          51336 non-null  int64
 23  Other_TL              51336 non-null  int64
 24  Age_Oldest_TL         51336 non-null  int64
```

```
 25  Age_Newest_TL         51336 non-null  int64
dtypes: float64(6), int64(20)
memory usage: 10.2 MB
```

```
#remove nulls (those rows are removed in which age_oldest_tl is -99999)
data_1=data_1.loc[data_1['Age_Oldest_TL']!=-99999]
```

```
data_1.shape
```

```
(51296, 26)
```

```
#now for deleting the null values for data_2
col_rem=[]
for i in data_2.columns:
  if data_2.loc[data_2[i]==-99999].shape[0]>10000:
    col_rem.append(i)
```

```
col_rem
```

```
['time_since_first_deliquency',
 'time_since_recent_deliquency',
 'max_delinquency_level',
 'max_deliq_6mts',
 'max_deliq_12mts',
 'CC_utilization',
 'PL_utilization',
 'max_unsec_exposure_inPct']
```

```
data_2=data_2.drop(columns=col_rem,axis=1)
```

```
data_2.shape
```

```
(51336, 54)
```

```
for i in data_2.columns:
  data_2=data_2.loc[data_2[i]!=-99999]
```

```
data_2.shape
```

```
(42066, 54)
```

```
data_2.isnull().sum()
```

```
PROSPECTID                    0
time_since_recent_payment     0
num_times_delinquent          0
max_recent_level_of_deliq     0
num_deliq_6mts                0
num_deliq_12mts               0
num_deliq_6_12mts             0
num_times_30p_dpd             0
num_times_60p_dpd             0
num_std                       0
num_std_6mts                  0
num_std_12mts                 0
num_sub                       0
num_sub_6mts                  0
num_sub_12mts                 0
num_dbt                       0
num_dbt_6mts                  0
num_dbt_12mts                 0
num_lss                       0
num_lss_6mts                  0
num_lss_12mts                 0
recent_level_of_deliq         0
tot_enq                       0
CC_enq                        0
CC_enq_L6m                    0
CC_enq_L12m                   0
PL_enq                        0
PL_enq_L6m                    0
PL_enq_L12m                   0
time_since_recent_enq         0
enq_L12m                      0
enq_L6m                       0
enq_L3m                       0
MARITALSTATUS                 0
EDUCATION                     0
AGE                           0
GENDER                        0
NETMONTHLYINCOME              0
Time_With_Curr_Empr           0
pct_of_active_TLs_ever        0
pct_opened_TLs_L6m_of_L12m    0
pct_currentBal_all_TL         0
CC_Flag                       0
PL_Flag                       0
pct_PL_enq_L6m_of_L12m        0
pct_CC_enq_L6m_of_L12m        0
pct_PL_enq_L6m_of_ever        0
pct_CC_enq_L6m_of_ever        0
HL_Flag                       0
GL_Flag                       0
last_prod_enq2                0
```

```
first_prod_enq2           0
Credit_Score              0
Approved_Flag             0
dtype: int64
```

```
#checking common column names:
for i in list(data_1.columns):
  if i in list(data_2.columns):
    print(i)
```

```
PROSPECTID
```

```
#merge the two dataframes, inner join so that no nulls are present
data=pd.merge(data_1,data_2,how='inner',left_on=['PROSPECTID'] ,right_on=['PROSPECTID'])
```

```
data
```

	PROSPECTID	Total_TL	Tot_Closed_TL	Tot_Active_TL	Total_TL_opened_L6M	Tot_TL_closed_L6M	pct_tl_open_L6M	pct_tl_close
0	1	5	4	1	0	0	0.000	
1	2	1	0	1	0	0	0.000	
2	3	8	0	8	1	0	0.125	
3	5	3	2	1	0	0	0.000	
4	6	6	5	1	0	0	0.000	
...	...	...	...	...	...	...	...	
42059	51332	3	0	3	1	0	0.333	
42060	51333	4	2	2	0	1	0.000	
42061	51334	2	1	1	1	1	0.500	
42062	51335	2	1	1	0	0	0.000	
42063	51336	1	0	1	0	0	0.000	

42064 rows × 79 columns

```
data.info()
```

```
 25  Age_Newest_TL              42064 non-null  int64
 26  time_since_recent_payment   42064 non-null  int64
 27  num_times_delinquent        42064 non-null  int64
 28  max_recent_level_of_deliq   42064 non-null  int64
 29  num_deliq_6mts              42064 non-null  int64
 30  num_deliq_12mts             42064 non-null  int64
 31  num_deliq_6_12mts           42064 non-null  int64
 32  num_times_30p_dpd           42064 non-null  int64
 33  num_times_60p_dpd           42064 non-null  int64
 34  num_std                     42064 non-null  int64
 35  num_std_6mts                42064 non-null  int64
 36  num_std_12mts               42064 non-null  int64
 37  num_sub                     42064 non-null  int64
 38  num_sub_6mts                42064 non-null  int64
 39  num_sub_12mts               42064 non-null  int64
 40  num_dbt                     42064 non-null  int64
 41  num_dbt_6mts                42064 non-null  int64
 42  num_dbt_12mts               42064 non-null  int64
 43  num_lss                     42064 non-null  int64
 44  num_lss_6mts                42064 non-null  int64
 45  num_lss_12mts               42064 non-null  int64
 46  recent_level_of_deliq       42064 non-null  int64
 47  tot_enq                     42064 non-null  int64
 48  CC_enq                      42064 non-null  int64
 49  CC_enq_L6m                  42064 non-null  int64
 50  CC_enq_L12m                 42064 non-null  int64
```

```
dtypes: float64(13), int64(60), object(6)
memory usage: 25.4+ MB
```

```
data.isna().sum().sum()
```

```
0
```

```
#Separating the type of features:Categorical | Numerical
for i in data.columns:
  if data[i].dtype=='object':
    print(i)
```

```
MARITALSTATUS
EDUCATION
GENDER
last_prod_enq2
first_prod_enq2
Approved_Flag
```

```
#chi-square test
for i in ['MARITALSTATUS','EDUCATION','GENDER','last_prod_enq2','first_prod_enq2','Approved_Flag']:
  chi2,pval,_,_ = chi2_contingency(pd.crosstab(data[i],data['Approved_Flag']))
  print(i,'---',pval)

#since all the categorical features have pval <= 0.05, we will accept all
```

```
MARITALSTATUS --- 3.578180861038862e-233
EDUCATION --- 2.6942265249737532e-30
GENDER --- 1.907936100186563e-05
last_prod_enq2 --- 0.0
first_prod_enq2 --- 7.84997610555419e-287
Approved_Flag --- 0.0
```

```
#VIF for numerical columns
num_cols=[]
for i in data.columns:
  if data[i].dtype!='object' and i not in ['PROSPECTID','Approved_flag']:
    num_cols.append(i)

num_cols
```

```
 'Home_TL',
 'PL_TL',
 'Secured_TL',
 'Unsecured_TL',
 'Other_TL',
 'Age_Oldest_TL',
 'Age_Newest_TL',
 'time_since_recent_payment',
 'num_times_delinquent',
 'max_recent_level_of_deliq',
 'num_deliq_6mts',
 'num_deliq_12mts',
 'num_deliq_6_12mts',
 'num_times_30p_dpd',
 'num_times_60p_dpd',
 'num_std',
 'num_std_6mts',
 'num_std_12mts',
 'num_sub',
 'num_sub_6mts',
 'num_sub_12mts',
 'num_dbt',
 'num_dbt_6mts',
 'num_dbt_12mts',
 'num_lss',
 'num_lss_6mts',
 'num_lss_12mts',
 'recent_level_of_deliq',
 'tot_enq',
 'CC_enq',
 'CC_enq_L6m',
 'CC_enq_L12m',
 'PL_enq',
 'PL_enq_L6m',
 'PL_enq_L12m',
 'time_since_recent_enq',
 'enq_L12m',
 'enq_L6m',
 'enq_L3m',
 'AGE',
```

```
#Multicollinearity vs Correlation
#Predictability of each features by other features
#Correlation is specific to linear relationships between columns(convex function-cannot say whether positively or negatively cor

#VIF sequwntially check
vlf_data=data[num_cols]
total_col=vlf_data.shape[1]
col_kept=[]
col_index=0

for i in range(total_col):
  vif_value=variance_inflation_factor(vlf_data.values,col_index)
  print(col_index,'---',vif_value)
  if vif_value<=6:
    col_kept.append(num_cols[i])
    col_index+=1
  else:
    vif_data=vlf_data.drop(columns=num_cols[i],axis=1)



  #72 numerical features-->39 after vif
```

```
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
```

```
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
0 --- inf
```

```
#Check Anova for columns_to_be_kept
from scipy.stats import f_oneway
col_to_be_kept_numerical=[]
for i in col_kept:
  a=list(data[i])
  b=list(data['Approved_Flag'])
  group_P1=[value for value,group in zip(a,b) if group=='P1']
  group_P2=[value for value,group in zip(a,b) if group=='P2']
  group_P3=[value for value,group in zip(a,b) if group=='P3']
  group_P4=[value for value,group in zip(a,b) if group=='P4']

  f_statistict,p_value=f_oneway(group_P1,group_P2,group_P3,group_P4)
  if p_value<=0.05:
    col_to_be_kept_numerical.append(i)
```

```
col_to_be_kept_numerical
```

```
[]
```

```
#feature selection is done for cat and num features
# Label encoding for the categorical features
['MARITALSTATUS', 'EDUCATION', 'GENDER' , 'last_prod_enq2' ,'first_prod_enq2']
```

```
['MARITALSTATUS', 'EDUCATION', 'GENDER', 'last_prod_enq2', 'first_prod_enq2']
```

```
data['MARITALSTATUS'].unique()
data['EDUCATION'].unique()
data['GENDER'].unique()
data['last_prod_enq2'].unique()
data['first_prod_enq2'].unique()
```

```
array(['PL', 'ConsumerLoan', 'others', 'AL', 'HL', 'CC'], dtype=object)
```

```
# Ordinal feature -- EDUCATION
# SSC            : 1
# 12TH           : 2
# GRADUATE       : 3
# UNDER GRADUATE : 3
# POST-GRADUATE  : 4
# OTHERS         : 1
# PROFESSIONAL   : 3


# Others has to be verified by the business end user

data.loc[data['EDUCATION'] == 'SSC',['EDUCATION']]              = 1
data.loc[data['EDUCATION'] == '12TH',['EDUCATION']]             = 2
data.loc[data['EDUCATION'] == 'GRADUATE',['EDUCATION']]         = 3
data.loc[data['EDUCATION'] == 'UNDER GRADUATE',['EDUCATION']]   = 3
data.loc[data['EDUCATION'] == 'POST-GRADUATE',['EDUCATION']]    = 4
data.loc[data['EDUCATION'] == 'OTHERS',['EDUCATION']]           = 1
data.loc[data['EDUCATION'] == 'PROFESSIONAL',['EDUCATION']]     = 3

data['EDUCATION'].value_counts()
data['EDUCATION'] = data['EDUCATION'].astype(int)
data.info()
```

```
 43  num_lss                   42064 non-null  int64
 44  num_lss_6mts              42064 non-null  int64
 45  num_lss_12mts             42064 non-null  int64
 46  recent_level_of_deliq     42064 non-null  int64
 47  tot_enq                   42064 non-null  int64
 48  CC_enq                    42064 non-null  int64
 49  CC_enq_L6m                42064 non-null  int64
 50  CC_enq_L12m               42064 non-null  int64
 51  PL_enq                    42064 non-null  int64
 52  PL_enq_L6m                42064 non-null  int64
 53  PL_enq_L12m               42064 non-null  int64
 54  time_since_recent_enq     42064 non-null  int64
 55  enq_L12m                  42064 non-null  int64
 56  enq_L6m                   42064 non-null  int64
 57  enq_L3m                   42064 non-null  int64
 58  MARITALSTATUS             42064 non-null  object
 59  EDUCATION                 42064 non-null  int64
 60  AGE                       42064 non-null  int64
 61  GENDER                    42064 non-null  object
 62  NETMONTHLYINCOME          42064 non-null  int64
 63  Time_With_Curr_Empr       42064 non-null  int64
 64  pct_of_active_TLs_ever    42064 non-null  float64
 65  pct_opened_TLs_L6m_of_L12m  42064 non-null  float64
 66  pct_currentBal_all_TL     42064 non-null  float64
 67  CC_Flag                   42064 non-null  int64
 68  PL_Flag                   42064 non-null  int64
 69  pct_PL_enq_L6m_of_L12m    42064 non-null  float64
 70  pct_CC_enq_L6m_of_L12m    42064 non-null  float64
 71  pct_PL_enq_L6m_of_ever    42064 non-null  float64
 72  pct_CC_enq_L6m_of_ever    42064 non-null  float64
 73  HL_Flag                   42064 non-null  int64
 74  GL_Flag                   42064 non-null  int64
 75  last_prod_enq2            42064 non-null  object
 76  first_prod_enq2           42064 non-null  object
 77  Credit_Score              42064 non-null  int64
 78  Approved_Flag             42064 non-null  object
dtypes: float64(13), int64(61), object(5)
memory usage: 25.4+ MB
```

```
data_encoded = pd.get_dummies(data, columns=['MARITALSTATUS','GENDER', 'last_prod_enq2' ,'first_prod_enq2'])
```

```
data_encoded.info()
k = data_encoded.describe()
```

```
 41  num_dbt_6mts                 42064 non-null  int64
 42  num_dbt_12mts                42064 non-null  int64
 43  num_lss                      42064 non-null  int64
 44  num_lss_6mts                 42064 non-null  int64
 45  num_lss_12mts                42064 non-null  int64
 46  recent_level_of_deliq        42064 non-null  int64
 47  tot_enq                      42064 non-null  int64
 48  CC_enq                       42064 non-null  int64
 49  CC_enq_L6m                   42064 non-null  int64
 50  CC_enq_L12m                  42064 non-null  int64
 51  PL_enq                       42064 non-null  int64
 52  PL_enq_L6m                   42064 non-null  int64
 53  PL_enq_L12m                  42064 non-null  int64
 54  time_since_recent_enq        42064 non-null  int64
 55  enq_L12m                     42064 non-null  int64
 56  enq_L6m                      42064 non-null  int64
 57  enq_L3m                      42064 non-null  int64
 58  EDUCATION                    42064 non-null  int64
 59  AGE                          42064 non-null  int64
 60  NETMONTHLYINCOME             42064 non-null  int64
 61  Time_With_Curr_Empr          42064 non-null  int64
 62  pct_of_active_TLs_ever       42064 non-null  float64
 63  pct_opened_TLs_L6m_of_L12m   42064 non-null  float64
 64  pct_currentBal_all_TL        42064 non-null  float64
 65  CC_Flag                      42064 non-null  int64
 66  PL_Flag                      42064 non-null  int64
 67  pct_PL_enq_L6m_of_L12m       42064 non-null  float64
 68  pct_CC_enq_L6m_of_L12m       42064 non-null  float64
 69  pct_PL_enq_L6m_of_ever       42064 non-null  float64
 70  pct_CC_enq_L6m_of_ever       42064 non-null  float64
 71  HL_Flag                      42064 non-null  int64
 72  GL_Flag                      42064 non-null  int64
 73  Credit_Score                 42064 non-null  int64
 74  Approved_Flag                42064 non-null  object
 75  MARITALSTATUS_Married        42064 non-null  bool
 76  MARITALSTATUS_Single         42064 non-null  bool
 77  GENDER_F                     42064 non-null  bool
 78  GENDER_M                     42064 non-null  bool
 79  last_prod_enq2_AL            42064 non-null  bool
 80  last_prod_enq2_CC            42064 non-null  bool
 81  last_prod_enq2_ConsumerLoan  42064 non-null  bool
 82  last_prod_enq2_HL            42064 non-null  bool
 83  last_prod_enq2_PL            42064 non-null  bool
 84  last_prod_enq2_others        42064 non-null  bool
 85  first_prod_enq2_AL           42064 non-null  bool
 86  first_prod_enq2_CC           42064 non-null  bool
 87  first_prod_enq2_ConsumerLoan 42064 non-null  bool
 88  first_prod_enq2_HL           42064 non-null  bool
 89  first_prod_enq2_PL           42064 non-null  bool
 90  first_prod_enq2_others       42064 non-null  bool
dtypes: bool(16), float64(13), int64(61), object(1)
memory usage: 24.7+ MB
```

```
data_encoded
```

	PROSPECTID	Total_TL	Tot_Closed_TL	Tot_Active_TL	Total_TL_opened_L6M	Tot_TL_closed_L6M	pct_tl_open_L6M	pct_tl_closed_L6M	pct_active_tl	pct_closed_tl	...
0	1	5	4	1	0	0	0.000	0.00	0.200	0.800	...
1	2	1	0	1	0	0	0.000	0.00	1.000	0.000	...
2	3	8	0	8	1	0	0.125	0.00	1.000	0.000	...
3	5	3	2	1	0	0	0.000	0.00	0.333	0.667	...
4	6	6	5	1	0	0	0.000	0.00	0.167	0.833	...
...	...	...	...	...	...	...	...	...	...	...	...
42059	51332	3	0	3	1	0	0.333	0.00	1.000	0.000	...
42060	51333	4	2	2	0	1	0.000	0.25	0.500	0.500	...
42061	51334	2	1	1	1	1	0.500	0.50	0.500	0.500	...
42062	51335	2	1	1	0	0	0.000	0.00	0.500	0.500	...
42063	51336	1	0	1	0	0	0.000	0.00	1.000	0.000	...

42064 rows × 91 columns

```
#Machine learning model fitting

y = data_encoded['Approved_Flag']
x = data_encoded. drop ( ['Approved_Flag'], axis = 1 )


x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

rf_classifier = RandomForestClassifier(n_estimators = 200, random_state=42)

rf_classifier.fit(x_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=200, random_state=42)
```

```
y_pred = rf_classifier.predict(x_test)
```

```python
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

# Assuming y_test and y_pred are defined and contain the actual and predicted labels, respectively.
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}\n')

# Calculate precision, recall, and F1 score for each class
precision, recall, f1_score, _ = precision_recall_fscore_support(y_test, y_pred, average=None)

# Print precision, recall, and F1 score for each class
for i, v in enumerate(['p1', 'p2', 'p3', 'p4']):
    print(f"Class {v}:")
    print(f"Precision: {precision[i]}")
    print(f"Recall: {recall[i]}")
    print(f"F1 Score: {f1_score[i]}")
    print()
```

```
Accuracy: 0.9900154522762391

Class p1:
Precision: 0.9465290806754222
Recall: 0.995069033530572
F1 Score: 0.9701923076923077

Class p2:
Precision: 0.9954617205998422
Recall: 1.0
F1 Score: 0.9977256995945812

Class p3:
Precision: 0.9968102073365231
Recall: 0.9433962264150944
F1 Score: 0.9693679720822024

Class p4:
Precision: 1.0
Recall: 0.9961127308066083
F1 Score: 0.9980525803310614
```

```
# 2. xgboost

import xgboost as xgb
from sklearn.preprocessing import LabelEncoder

xgb_classifier = xgb.XGBClassifier(objective='multi:softmax',  num_class=4)



y = data_encoded['Approved_Flag']
x = data_encoded. drop ( ['Approved_Flag'], axis = 1 )


label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)


x_train, x_test, y_train, y_test = train_test_split(x, y_encoded, test_size=0.2, random_state=42)




xgb_classifier.fit(x_train, y_train)
y_pred = xgb_classifier.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
print ()
print(f'Accuracy: {accuracy:.2f}')
print ()

precision, recall, f1_score, _ = precision_recall_fscore_support(y_test, y_pred)

for i, v in enumerate(['p1', 'p2', 'p3', 'p4']):
    print(f"Class {v}:")
    print(f"Precision: {precision[i]}")
    print(f"Recall: {recall[i]}")
    print(f"F1 Score: {f1_score[i]}")
    print()
```

```
Accuracy: 1.00

Class p1:
Precision: 1.0
Recall: 1.0
F1 Score: 1.0

Class p2:
Precision: 1.0
Recall: 1.0
F1 Score: 1.0

Class p3:
Precision: 1.0
Recall: 1.0
F1 Score: 1.0

Class p4:
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```

```
# 3. Decision Tree
from sklearn.tree import DecisionTreeClassifier


y = data_encoded['Approved_Flag']
x = data_encoded. drop ( ['Approved_Flag'], axis = 1 )

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)


dt_model = DecisionTreeClassifier(max_depth=20, min_samples_split=10)
dt_model.fit(x_train, y_train)
y_pred = dt_model.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
print ()
print(f"Accuracy: {accuracy:.2f}")
print ()

precision, recall, f1_score, _ = precision_recall_fscore_support(y_test, y_pred)

for i, v in enumerate(['p1', 'p2', 'p3', 'p4']):
    print(f"Class {v}:")
    print(f"Precision: {precision[i]}")
    print(f"Recall: {recall[i]}")
    print(f"F1 Score: {f1_score[i]}")
    print()
```

```
Accuracy: 1.00

Class p1:
Precision: 1.0
Recall: 1.0
F1 Score: 1.0

Class p2:
Precision: 1.0
Recall: 1.0
F1 Score: 1.0

Class p3:
Precision: 1.0
```

```
Recall: 1.0
F1 Score: 1.0

Class p4:
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```

```
#Xg boost is giving the highest accuracy so we will pick it and finetune it
```

```
# Hyperparameter tuning in xgboost
from sklearn.model_selection import GridSearchCV
x_train, x_test, y_train, y_test = train_test_split(x, y_encoded, test_size=0.2, random_state=42)

# Define the XGBClassifier with the initial set of hyperparameters
xgb_model = xgb.XGBClassifier(objective='multi:softmax', num_class=4)

# Define the parameter grid for hyperparameter tuning

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
}

grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=3, scoring='accuracy', n_jobs=-1)
grid_search.fit(x_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)
```

```
Best Hyperparameters: {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 50}
```

```
# Evaluate the model with the best hyperparameters on the test set
best_model = grid_search.best_estimator_
accuracy = best_model.score(x_test, y_test)
print("Test Accuracy:", accuracy)

# Best Hyperparameters: {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 200}


# Based on risk appetite of the bank, you will suggest P1,P2,P3,P4 to the business end user
```

```
Test Accuracy: 1.0
```

```
# # Hyperparameter tuning for xgboost (Used in the session)

# # Define the hyperparameter grid
param_grid = {
   'colsample_bytree': [0.1, 0.3, 0.5, 0.7, 0.9],
   'learning_rate'   : [0.001, 0.01, 0.1, 1],
   'max_depth'       : [3, 5, 8, 10],
   'alpha'           : [1, 10, 100],
   'n_estimators'    : [10,50,100]
 }

index = 0

answers_grid = {
     'combination'       :[],
     'train_Accuracy'    :[],
     'test_Accuracy'     :[],
     'colsample_bytree'  :[],
     'learning_rate'     :[],
     'max_depth'         :[],
     'alpha'             :[],
     'n_estimators'      :[]

     }
```

```
# # Loop through each combination of hyperparameters
for colsample_bytree in param_grid['colsample_bytree']:
   for learning_rate in param_grid['learning_rate']:
     for max_depth in param_grid['max_depth']:
       for alpha in param_grid['alpha']:
           for n_estimators in param_grid['n_estimators']:

               index = index + 1

               # Define and train the XGBoost model
               model = xgb.XGBClassifier(objective='multi:softmax',
                                         num_class=4,
                                         colsample_bytree = colsample_bytree,
                                         learning_rate = learning_rate,
                                         max_depth = max_depth,
                                         alpha = alpha,
                                         n_estimators = n_estimators)



               y = data_encoded['Approved_Flag']
               x = data_encoded. drop ( ['Approved_Flag'], axis = 1 )

               label_encoder = LabelEncoder()
               y_encoded = label_encoder.fit_transform(y)


               x_train, x_test, y_train, y_test = train_test_split(x, y_encoded, test_size=0.2, random_state=42)


               model.fit(x_train, y_train)



               # Predict on training and testing sets
               y_pred_train = model.predict(x_train)
               y_pred_test = model.predict(x_test)
```

```
# Calculate train and test results

train_accuracy =  accuracy_score (y_train, y_pred_train)
test_accuracy  =  accuracy_score (y_test , y_pred_test)



# Include into the lists
answers_grid ['combination']   .append(index)
answers_grid ['train_Accuracy']    .append(train_accuracy)
answers_grid ['test_Accuracy']     .append(test_accuracy)
answers_grid ['colsample_bytree']   .append(colsample_bytree)
answers_grid ['learning_rate']      .append(learning_rate)
answers_grid ['max_depth']          .append(max_depth)
answers_grid ['alpha']              .append(alpha)
answers_grid ['n_estimators']       .append(n_estimators)


# Print results for this combination
print(f"Combination {index}")
print(f"colsample_bytree: {colsample_bytree}, learning_rate: {learning_rate}, max_depth: {max_depth}, alpha: {alph
print(f"Train Accuracy: {train_accuracy:.2f}")
print(f"Test Accuracy : {test_accuracy :.2f}")
print("-" * 30)
```

```
Combination 721
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 3, alpha: 1, n_estimators: 10
Train Accuracy: 0.75
Test Accuracy : 0.74
------------------------------
Combination 722
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 3, alpha: 1, n_estimators: 50
Train Accuracy: 0.67
Test Accuracy : 0.66
------------------------------
Combination 723
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 3, alpha: 1, n_estimators: 100
Train Accuracy: 0.67
Test Accuracy : 0.66
------------------------------
Combination 724
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 3, alpha: 10, n_estimators: 10
Train Accuracy: 0.75
Test Accuracy : 0.74
------------------------------
Combination 725
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 3, alpha: 10, n_estimators: 50
Train Accuracy: 0.66
Test Accuracy : 0.66
------------------------------
Combination 726
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 3, alpha: 10, n_estimators: 100
Train Accuracy: 0.67
Test Accuracy : 0.66
------------------------------
Combination 727
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 3, alpha: 100, n_estimators: 10
Train Accuracy: 0.74
Test Accuracy : 0.74
------------------------------
Combination 728
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 3, alpha: 100, n_estimators: 50
Train Accuracy: 0.66
Test Accuracy : 0.65
------------------------------
Combination 729
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 3, alpha: 100, n_estimators: 100
Train Accuracy: 0.66
Test Accuracy : 0.65
------------------------------
Combination 730
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 5, alpha: 1, n_estimators: 10
Train Accuracy: 0.78
Test Accuracy : 0.77
------------------------------
Combination 731
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 5, alpha: 1, n_estimators: 50
Train Accuracy: 0.68
Test Accuracy : 0.67
```

```
Test Accuracy : 0.67
------------------------------
Combination 732
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 5, alpha: 1, n_estimators: 100
Train Accuracy: 0.69
Test Accuracy : 0.68
------------------------------
Combination 733
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 5, alpha: 10, n_estimators: 10
Train Accuracy: 0.78
Test Accuracy : 0.77
------------------------------
Combination 734
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 5, alpha: 10, n_estimators: 50
Train Accuracy: 0.68
Test Accuracy : 0.67
------------------------------
Combination 735
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 5, alpha: 10, n_estimators: 100
Train Accuracy: 0.68
Test Accuracy : 0.67
------------------------------
Combination 736
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 5, alpha: 100, n_estimators: 10
Train Accuracy: 0.75
Test Accuracy : 0.74
------------------------------
Combination 737
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 5, alpha: 100, n_estimators: 50
Train Accuracy: 0.66
Test Accuracy : 0.65
------------------------------
Combination 738
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 5, alpha: 100, n_estimators: 100
Train Accuracy: 0.66
Test Accuracy : 0.65
------------------------------
Combination 739
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 8, alpha: 1, n_estimators: 10
Train Accuracy: 0.80
Test Accuracy : 0.78
------------------------------
Combination 740
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 8, alpha: 1, n_estimators: 50
Train Accuracy: 0.71
Test Accuracy : 0.68
------------------------------
Combination 741
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 8, alpha: 1, n_estimators: 100
Train Accuracy: 0.71
Test Accuracy : 0.69
------------------------------
Combination 742
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 8, alpha: 10, n_estimators: 10
Train Accuracy: 0.79
```

```
Test Accuracy : 0.78
------------------------------
Combination 743
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 8, alpha: 10, n_estimators: 50
Train Accuracy: 0.69
Test Accuracy : 0.68
------------------------------
Combination 744
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 8, alpha: 10, n_estimators: 100
Train Accuracy: 0.69
Test Accuracy : 0.68
------------------------------
Combination 745
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 8, alpha: 100, n_estimators: 10
Train Accuracy: 0.75
Test Accuracy : 0.74
------------------------------
Combination 746
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 8, alpha: 100, n_estimators: 50
Train Accuracy: 0.66
Test Accuracy : 0.65
------------------------------
Combination 747
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 8, alpha: 100, n_estimators: 100
Train Accuracy: 0.66
Test Accuracy : 0.65
------------------------------
Combination 748
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 10, alpha: 1, n_estimators: 10
Train Accuracy: 0.82
Test Accuracy : 0.79
------------------------------
Combination 749
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 10, alpha: 1, n_estimators: 50
Train Accuracy: 0.72
Test Accuracy : 0.69
------------------------------
Combination 750
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 10, alpha: 1, n_estimators: 100
Train Accuracy: 0.72
Test Accuracy : 0.69
------------------------------
Combination 751
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 10, alpha: 10, n_estimators: 10
Train Accuracy: 0.79
Test Accuracy : 0.78
------------------------------
Combination 752
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 10, alpha: 10, n_estimators: 50
Train Accuracy: 0.69
Test Accuracy : 0.68
------------------------------
Combination 753
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 10, alpha: 10, n_estimators: 100
```

```
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 10, alpha: 10, n_estimators: 100
Train Accuracy: 0.69
Test Accuracy : 0.68
------------------------------
Combination 754
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 10, alpha: 100, n_estimators: 10
Train Accuracy: 0.75
Test Accuracy : 0.74
------------------------------
Combination 755
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 10, alpha: 100, n_estimators: 50
Train Accuracy: 0.66
Test Accuracy : 0.65
------------------------------
Combination 756
colsample_bytree: 0.1, learning_rate: 0.001, max_depth: 10, alpha: 100, n_estimators: 100
Train Accuracy: 0.66
Test Accuracy : 0.65
------------------------------
Combination 757
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 3, alpha: 1, n_estimators: 10
Train Accuracy: 0.76
Test Accuracy : 0.75
------------------------------
Combination 758
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 3, alpha: 1, n_estimators: 50
Train Accuracy: 0.68
Test Accuracy : 0.68
------------------------------
Combination 759
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 3, alpha: 1, n_estimators: 100
Train Accuracy: 0.71
Test Accuracy : 0.70
------------------------------
Combination 760
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 3, alpha: 10, n_estimators: 10
Train Accuracy: 0.76
Test Accuracy : 0.75
------------------------------
Combination 761
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 3, alpha: 10, n_estimators: 50
Train Accuracy: 0.69
Test Accuracy : 0.68
------------------------------
Combination 762
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 3, alpha: 10, n_estimators: 100
Train Accuracy: 0.71
Test Accuracy : 0.70
------------------------------
Combination 763
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 3, alpha: 100, n_estimators: 10
Train Accuracy: 0.75
Test Accuracy : 0.74
------------------------------
Combination 764
```

```
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 3, alpha: 100, n_estimators: 50
Train Accuracy: 0.68
Test Accuracy : 0.67
------------------------------
Combination 765
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 3, alpha: 100, n_estimators: 100
Train Accuracy: 0.70
Test Accuracy : 0.69
------------------------------
Combination 766
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 5, alpha: 1, n_estimators: 10
Train Accuracy: 0.79
Test Accuracy : 0.78
------------------------------
Combination 767
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 5, alpha: 1, n_estimators: 50
Train Accuracy: 0.71
Test Accuracy : 0.70
------------------------------
Combination 768
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 5, alpha: 1, n_estimators: 100
Train Accuracy: 0.73
Test Accuracy : 0.72
------------------------------
Combination 769
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 5, alpha: 10, n_estimators: 10
Train Accuracy: 0.79
Test Accuracy : 0.78
------------------------------
Combination 770
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 5, alpha: 10, n_estimators: 50
Train Accuracy: 0.70
Test Accuracy : 0.69
------------------------------
Combination 771
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 5, alpha: 10, n_estimators: 100
Train Accuracy: 0.72
Test Accuracy : 0.71
------------------------------
Combination 772
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 5, alpha: 100, n_estimators: 10
Train Accuracy: 0.76
Test Accuracy : 0.75
------------------------------
Combination 773
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 5, alpha: 100, n_estimators: 50
Train Accuracy: 0.68
Test Accuracy : 0.67
------------------------------
Combination 774
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 5, alpha: 100, n_estimators: 100
Train Accuracy: 0.70
Test Accuracy : 0.69
------------------------------
```

```
Combination 775
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 8, alpha: 1, n_estimators: 10
Train Accuracy: 0.81
Test Accuracy : 0.79
------------------------------
Combination 776
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 8, alpha: 1, n_estimators: 50
Train Accuracy: 0.73
Test Accuracy : 0.71
------------------------------
Combination 777
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 8, alpha: 1, n_estimators: 100
Train Accuracy: 0.76
Test Accuracy : 0.73
------------------------------
Combination 778
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 8, alpha: 10, n_estimators: 10
Train Accuracy: 0.80
Test Accuracy : 0.78
------------------------------
Combination 779
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 8, alpha: 10, n_estimators: 50
Train Accuracy: 0.71
Test Accuracy : 0.70
------------------------------
Combination 780
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 8, alpha: 10, n_estimators: 100
Train Accuracy: 0.74
Test Accuracy : 0.72
------------------------------
Combination 781
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 8, alpha: 100, n_estimators: 10
Train Accuracy: 0.76
Test Accuracy : 0.75
------------------------------
Combination 782
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 8, alpha: 100, n_estimators: 50
Train Accuracy: 0.68
Test Accuracy : 0.67
------------------------------
Combination 783
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 8, alpha: 100, n_estimators: 100
Train Accuracy: 0.70
Test Accuracy : 0.69
------------------------------
Combination 784
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 10, alpha: 1, n_estimators: 10
Train Accuracy: 0.83
Test Accuracy : 0.79
------------------------------
Combination 785
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 10, alpha: 1, n_estimators: 50
Train Accuracy: 0.75
Test Accuracy : 0.71
```

```
------------------------------
Combination 786
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 10, alpha: 1, n_estimators: 100
Train Accuracy: 0.78
Test Accuracy : 0.73
------------------------------
Combination 787
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 10, alpha: 10, n_estimators: 10
Train Accuracy: 0.80
Test Accuracy : 0.79
------------------------------
Combination 788
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 10, alpha: 10, n_estimators: 50
Train Accuracy: 0.72
Test Accuracy : 0.70
------------------------------
Combination 789
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 10, alpha: 10, n_estimators: 100
Train Accuracy: 0.74
Test Accuracy : 0.72
------------------------------
Combination 790
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 10, alpha: 100, n_estimators: 10
Train Accuracy: 0.76
Test Accuracy : 0.75
------------------------------
Combination 791
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 10, alpha: 100, n_estimators: 50
Train Accuracy: 0.68
Test Accuracy : 0.67
------------------------------
Combination 792
colsample_bytree: 0.1, learning_rate: 0.01, max_depth: 10, alpha: 100, n_estimators: 100
Train Accuracy: 0.70
Test Accuracy : 0.69
------------------------------
Combination 793
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 3, alpha: 1, n_estimators: 10
Train Accuracy: 0.86
Test Accuracy : 0.86
------------------------------
Combination 794
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 3, alpha: 1, n_estimators: 50
Train Accuracy: 0.95
Test Accuracy : 0.94
------------------------------
Combination 795
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 3, alpha: 1, n_estimators: 100
Train Accuracy: 0.99
Test Accuracy : 0.99
------------------------------
Combination 796
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 3, alpha: 10, n_estimators: 10
Train Accuracy: 0.86
```

```
Test Accuracy : 0.86
------------------------------
Combination 797
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 3, alpha: 10, n_estimators: 50
Train Accuracy: 0.95
Test Accuracy : 0.94
------------------------------
Combination 798
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 3, alpha: 10, n_estimators: 100
Train Accuracy: 0.99
Test Accuracy : 0.99
------------------------------
Combination 799
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 3, alpha: 100, n_estimators: 10
Train Accuracy: 0.85
Test Accuracy : 0.85
------------------------------
Combination 800
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 3, alpha: 100, n_estimators: 50
Train Accuracy: 0.93
Test Accuracy : 0.93
------------------------------
Combination 801
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 3, alpha: 100, n_estimators: 100
Train Accuracy: 0.98
Test Accuracy : 0.98
------------------------------
Combination 802
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 5, alpha: 1, n_estimators: 10
Train Accuracy: 0.88
Test Accuracy : 0.87
------------------------------
Combination 803
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 5, alpha: 1, n_estimators: 50
Train Accuracy: 0.95
Test Accuracy : 0.94
------------------------------
Combination 804
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 5, alpha: 1, n_estimators: 100
Train Accuracy: 0.99
Test Accuracy : 0.99
------------------------------
Combination 805
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 5, alpha: 10, n_estimators: 10
Train Accuracy: 0.88
Test Accuracy : 0.87
------------------------------
Combination 806
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 5, alpha: 10, n_estimators: 50
Train Accuracy: 0.95
Test Accuracy : 0.94
------------------------------
Combination 807
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 5, alpha: 10, n_estimators: 100
```

```
Train Accuracy: 0.99
Test Accuracy : 0.99
------------------------------
Combination 808
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 5, alpha: 100, n_estimators: 10
Train Accuracy: 0.86
Test Accuracy : 0.85
------------------------------
Combination 809
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 5, alpha: 100, n_estimators: 50
Train Accuracy: 0.93
Test Accuracy : 0.93
------------------------------
Combination 810
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 5, alpha: 100, n_estimators: 100
Train Accuracy: 0.98
Test Accuracy : 0.98
------------------------------
Combination 811
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 8, alpha: 1, n_estimators: 10
Train Accuracy: 0.90
Test Accuracy : 0.88
------------------------------
Combination 812
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 8, alpha: 1, n_estimators: 50
Train Accuracy: 0.96
Test Accuracy : 0.94
------------------------------
Combination 813
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 8, alpha: 1, n_estimators: 100
Train Accuracy: 1.00
Test Accuracy : 0.99
------------------------------
Combination 814
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 8, alpha: 10, n_estimators: 10
Train Accuracy: 0.88
Test Accuracy : 0.87
------------------------------
Combination 815
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 8, alpha: 10, n_estimators: 50
Train Accuracy: 0.95
Test Accuracy : 0.94
------------------------------
Combination 816
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 8, alpha: 10, n_estimators: 100
Train Accuracy: 0.99
Test Accuracy : 0.99
------------------------------
Combination 817
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 8, alpha: 100, n_estimators: 10
Train Accuracy: 0.86
Test Accuracy : 0.85
------------------------------
Combination 818
```

```
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 8, alpha: 100, n_estimators: 50
Train Accuracy: 0.93
Test Accuracy : 0.93
------------------------------
Combination 819
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 8, alpha: 100, n_estimators: 100
Train Accuracy: 0.98
Test Accuracy : 0.98
------------------------------
Combination 820
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 10, alpha: 1, n_estimators: 10
Train Accuracy: 0.91
Test Accuracy : 0.87
------------------------------
Combination 821
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 10, alpha: 1, n_estimators: 50
Train Accuracy: 0.97
Test Accuracy : 0.93
------------------------------
Combination 822
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 10, alpha: 1, n_estimators: 100
Train Accuracy: 1.00
Test Accuracy : 0.98
------------------------------
Combination 823
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 10, alpha: 10, n_estimators: 10
Train Accuracy: 0.88
Test Accuracy : 0.88
------------------------------
Combination 824
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 10, alpha: 10, n_estimators: 50
Train Accuracy: 0.95
Test Accuracy : 0.94
------------------------------
Combination 825
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 10, alpha: 10, n_estimators: 100
Train Accuracy: 0.99
Test Accuracy : 0.99
------------------------------
Combination 826
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 10, alpha: 100, n_estimators: 10
Train Accuracy: 0.86
Test Accuracy : 0.85
------------------------------
Combination 827
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 10, alpha: 100, n_estimators: 50
Train Accuracy: 0.93
Test Accuracy : 0.93
------------------------------
Combination 828
colsample_bytree: 0.1, learning_rate: 0.1, max_depth: 10, alpha: 100, n_estimators: 100
Train Accuracy: 0.98
Test Accuracy : 0.98
------------------------------
```

```
Combination 829
colsample_bytree: 0.1, learning_rate: 1, max_depth: 3, alpha: 1, n_estimators: 10
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 830
colsample_bytree: 0.1, learning_rate: 1, max_depth: 3, alpha: 1, n_estimators: 50
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 831
colsample_bytree: 0.1, learning_rate: 1, max_depth: 3, alpha: 1, n_estimators: 100
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 832
colsample_bytree: 0.1, learning_rate: 1, max_depth: 3, alpha: 10, n_estimators: 10
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 833
colsample_bytree: 0.1, learning_rate: 1, max_depth: 3, alpha: 10, n_estimators: 50
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 834
colsample_bytree: 0.1, learning_rate: 1, max_depth: 3, alpha: 10, n_estimators: 100
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 835
colsample_bytree: 0.1, learning_rate: 1, max_depth: 3, alpha: 100, n_estimators: 10
Train Accuracy: 1.00
Test Accuracy : 0.99
------------------------------
Combination 836
colsample_bytree: 0.1, learning_rate: 1, max_depth: 3, alpha: 100, n_estimators: 50
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 837
colsample_bytree: 0.1, learning_rate: 1, max_depth: 3, alpha: 100, n_estimators: 100
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 838
colsample_bytree: 0.1, learning_rate: 1, max_depth: 5, alpha: 1, n_estimators: 10
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 839
colsample_bytree: 0.1, learning_rate: 1, max_depth: 5, alpha: 1, n_estimators: 50
Train Accuracy: 1.00
Test Accuracy : 1.00
```

```
------------------------------
Combination 840
colsample_bytree: 0.1, learning_rate: 1, max_depth: 5, alpha: 1, n_estimators: 100
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 841
colsample_bytree: 0.1, learning_rate: 1, max_depth: 5, alpha: 10, n_estimators: 10
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 842
colsample_bytree: 0.1, learning_rate: 1, max_depth: 5, alpha: 10, n_estimators: 50
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 843
colsample_bytree: 0.1, learning_rate: 1, max_depth: 5, alpha: 10, n_estimators: 100
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 844
colsample_bytree: 0.1, learning_rate: 1, max_depth: 5, alpha: 100, n_estimators: 10
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 845
colsample_bytree: 0.1, learning_rate: 1, max_depth: 5, alpha: 100, n_estimators: 50
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 846
colsample_bytree: 0.1, learning_rate: 1, max_depth: 5, alpha: 100, n_estimators: 100
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 847
colsample_bytree: 0.1, learning_rate: 1, max_depth: 8, alpha: 1, n_estimators: 10
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 848
colsample_bytree: 0.1, learning_rate: 1, max_depth: 8, alpha: 1, n_estimators: 50
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 849
colsample_bytree: 0.1, learning_rate: 1, max_depth: 8, alpha: 1, n_estimators: 100
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 850
colsample_bytree: 0.1, learning_rate: 1, max_depth: 8, alpha: 10, n_estimators: 10
Train Accuracy: 1.00
```

```
Test Accuracy : 1.00
------------------------------
Combination 851
colsample_bytree: 0.1, learning_rate: 1, max_depth: 8, alpha: 10, n_estimators: 50
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 852
colsample_bytree: 0.1, learning_rate: 1, max_depth: 8, alpha: 10, n_estimators: 100
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 853
colsample_bytree: 0.1, learning_rate: 1, max_depth: 8, alpha: 100, n_estimators: 10
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 854
colsample_bytree: 0.1, learning_rate: 1, max_depth: 8, alpha: 100, n_estimators: 50
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 855
colsample_bytree: 0.1, learning_rate: 1, max_depth: 8, alpha: 100, n_estimators: 100
Train Accuracy: 1.00
Test Accuracy : 1.00
------------------------------
Combination 856
colsample_bytree: 0.1, learning_rate: 1, max_depth: 10, alpha: 1, n_estimators: 10
Train Accuracy: 1.00
```