```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
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, classification_report, precision_recall_fscore_support
import warnings
import os
```

```
# Load the dataset
a1 = pd.read_excel("/content/case_study1.xlsx")
a2 = pd.read_excel("/content/case_study2.xlsx")
```

```
df1 = a1.copy()
df2 = a2.copy()
```

```
df1
```

	PROSPECTID	Total_TL	Tot_Closed_TL	Tot_Active_TL	Total_TL_opened_L6M	Tot_TL_closed_L6M	pct_
0	1	5	4	1	0	0	
1	2	1	0	1	0	0	
2	3	8	0	8	1	0	
3	4	1	0	1	1	0	
4	5	3	2	1	0	0	
...	...	...	...	...	...	...	
51331	51332	3	0	3	1	0	
51332	51333	4	2	2	0	1	
51333	51334	2	1	1	1	1	
51334	51335	2	1	1	0	0	
51335	51336	1	0	1	0	0	

51296 rows × 26 columns

```
df1.shape
```

```
(51296, 26)
```

```
df2
```

	PROSPECTID	time_since_recent_payment	num_times_delinquent	max_recent_level_of_deliq	num_deli
0	1	549	11	29	
1	2	47	0	0	
2	3	302	9	25	
4	5	583	0	0	
5	6	245	14	270	
...	...	...	...	...	
51331	51332	15	2	24	
51332	51333	57	0	0	
51333	51334	32	0	0	
51334	51335	58	0	0	
51335	51336	74	0	0	

42066 rows × 54 columns

```
df2.shape
```

```
(42066, 54)
```

```
# Remove nulls
df1 = df1.loc[df1['Age_Oldest_TL'] != -99999]
```

```
columns_to_be_removed = []

for i in df2.columns:
    if df2.loc[df2[i] == -99999].shape[0] > 10000:
        columns_to_be_removed .append(i)
```

```
df2 = df2.drop(columns_to_be_removed, axis =1)
```

```
for i in df2.columns:
    df2 = df2.loc[ df2[i] != -99999 ]
```

```
# Checking common column names
for i in list(df1.columns):
    if i in list(df2.columns):
        print (i)
```

```
PROSPECTID
```

```
# Merge the two dataframes, inner join so that no nulls are present
df = pd. merge ( df1, df2, how ='inner', left_on = ['PROSPECTID'], right_on = ['PROSPECTID'] )
```

```
df
```

```
df.shape
```

```
(42064, 43)
```

```
# check how many columns are categorical
for i in df.columns:
    if df[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']:
    chi2, pval, _, _ = chi2_contingency(pd.crosstab(df[i], df['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
```

```
df.shape
```

```
(42064, 43)
```

```
# VIF for numerical columns
numeric_columns = []
for i in df.columns:
    if df[i].dtype != 'object' and i not in ['PROSPECTID','Approved_Flag']:
```

```
            numeric_columns.append(i)
```

```
# VIF sequentially check

vif_data = df[numeric_columns]
total_columns = vif_data.shape[1]
columns_to_be_kept = []
column_index = 0
```

```
for i in range (0,total_columns):

    vif_value = variance_inflation_factor(vif_data, column_index)
    print (column_index,'---',vif_value)


    if vif_value <= 6:
        columns_to_be_kept.append( numeric_columns[i] )
        column_index = column_index+1

    else:
        vif_data = vif_data.drop([ numeric_columns[i] ] , axis=1)
```

```
6 --- 30.595522588100053
6 --- 4.3843464059655854
7 --- 3.064658415523423
8 --- 2.898639771299253
9 --- 4.377876915347322
10 --- 2.207853583695844
11 --- 4.916914200506861
12 --- 5.214702030064725
13 --- 3.3861625024231476
14 --- 7.840583309478997
14 --- 5.255034641721438
/usr/local/lib/python3.12/dist-packages/statsmodels/stats/outliers_influence.py:197: RuntimeWarning: d
  vif = 1. / (1. - r_squared_i)
15 --- inf
```

```
29 --- 17.006562234161628
29 --- 10.730485153719197
29 --- 2.3538497522950275
30 --- 22.104855915136433
30 --- 2.7971639638512915
31 --- 3.424171203217697
32 --- 10.175021454450922
32 --- 6.408710354561292
32 --- 1.001151196262563
33 --- 3.069197305397274
34 --- 2.8091261600643707
35 --- 20.249538381980678
35 --- 15.864576541593745
35 --- 1.8331649740532152
36 --- 1.5680839909542044
37 --- 1.9307572353811688
38 --- 4.331265056645249
39 --- 9.390334396150173
```

```
df.shape
```

```
(42064, 43)
```

```
# check Anova for columns_to_be_kept

from scipy.stats import f_oneway

columns_to_be_kept_numerical = []

for i in columns_to_be_kept:
    a = list(df[i])
    b = list(df['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_statistic, p_value = f_oneway(group_P1, group_P2, group_P3, group_P4)

    if p_value <= 0.05:
        columns_to_be_kept_numerical.append(i)




# feature selection is done for cat and num features
```

```
df.shape
```

```
(42064, 43)
```

```
# listing all the final features
features = columns_to_be_kept_numerical + ['MARITALSTATUS', 'EDUCATION', 'GENDER', 'last_prod_enq2', '
df = df[features + ['Approved_Flag']]
```

```
# Label encoding for the categorical features
['MARITALSTATUS', 'EDUCATION', 'GENDER' , 'last_prod_enq2' ,'first_prod_enq2']
```

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

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

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

```
df
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42064 entries, 0 to 42063
Data columns (total 43 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   pct_tl_open_L6M            42064 non-null  float64
 1   pct_tl_closed_L6M          42064 non-null  float64
 2   Tot_TL_closed_L12M         42064 non-null  int64
 3   pct_tl_closed_L12M         42064 non-null  float64
 4   Tot_Missed_Pmnt            42064 non-null  int64
 5   CC_TL                      42064 non-null  int64
 6   Home_TL                    42064 non-null  int64
 7   PL_TL                      42064 non-null  int64
 8   Secured_TL                 42064 non-null  int64
 9   Unsecured_TL               42064 non-null  int64
 10  Other_TL                   42064 non-null  int64
 11  Age_Oldest_TL              42064 non-null  int64
 12  Age_Newest_TL              42064 non-null  int64
 13  time_since_recent_payment  42064 non-null  int64
 14  max_recent_level_of_deliq  42064 non-null  int64
 15  num_deliq_6_12mts          42064 non-null  int64
 16  num_times_60p_dpd          42064 non-null  int64
 17  num_std_12mts              42064 non-null  int64
 18  num_sub                    42064 non-null  int64
 19  num_sub_6mts               42064 non-null  int64
 20  num_sub_12mts              42064 non-null  int64
 21  num_dbt                    42064 non-null  int64
 22  num_dbt_12mts              42064 non-null  int64
 23  num_lss                    42064 non-null  int64
 24  recent_level_of_deliq      42064 non-null  int64
 25  CC_enq_L12m                42064 non-null  int64
 26  PL_enq_L12m                42064 non-null  int64
 27  time_since_recent_enq      42064 non-null  int64
 28  enq_L3m                    42064 non-null  int64
 29  NETMONTHLYINCOME           42064 non-null  int64
```

```
 30  Time_With_Curr_Empr        42064 non-null  int64
 31  CC_Flag                    42064 non-null  int64
 32  PL_Flag                    42064 non-null  int64
 33  pct_PL_enq_L6m_of_ever     42064 non-null  float64
 34  pct_CC_enq_L6m_of_ever     42064 non-null  float64
 35  HL_Flag                    42064 non-null  int64
 36  GL_Flag                    42064 non-null  int64
 37  MARITALSTATUS              42064 non-null  object
 38  EDUCATION                  42064 non-null  int64
 39  GENDER                     42064 non-null  object
 40  last_prod_enq2             42064 non-null  object
 41  first_prod_enq2            42064 non-null  object
 42  Approved_Flag              42064 non-null  object
dtypes: float64(5), int64(33), object(5)
memory usage: 13.8+ MB
```

```
df_encoded = pd.get_dummies(df, columns=['MARITALSTATUS','GENDER', 'last_prod_enq2' ,'first_prod_enq2'
```

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

```
---  ------                         --------------  -----
 0   pct_tl_open_L6M                42064 non-null  float64
 1   pct_tl_closed_L6M              42064 non-null  float64
 2   Tot_TL_closed_L12M             42064 non-null  int64
 3   pct_tl_closed_L12M             42064 non-null  float64
 4   Tot_Missed_Pmnt                42064 non-null  int64
 5   CC_TL                          42064 non-null  int64
 6   Home_TL                        42064 non-null  int64
 7   PL_TL                          42064 non-null  int64
 8   Secured_TL                     42064 non-null  int64
 9   Unsecured_TL                   42064 non-null  int64
 10  Other_TL                       42064 non-null  int64
 11  Age_Oldest_TL                  42064 non-null  int64
 12  Age_Newest_TL                  42064 non-null  int64
 13  time_since_recent_payment      42064 non-null  int64
 14  max_recent_level_of_deliq      42064 non-null  int64
 15  num_deliq_6_12mts              42064 non-null  int64
 16  num_times_60p_dpd              42064 non-null  int64
 17  num_std_12mts                  42064 non-null  int64
 18  num_sub                        42064 non-null  int64
 19  num_sub_6mts                   42064 non-null  int64
 20  num_sub_12mts                  42064 non-null  int64
 21  num_dbt                        42064 non-null  int64
 22  num_dbt_12mts                  42064 non-null  int64
 23  num_lss                        42064 non-null  int64
 24  recent_level_of_deliq          42064 non-null  int64
 25  CC_enq_L12m                    42064 non-null  int64
 26  PL_enq_L12m                    42064 non-null  int64
 27  time_since_recent_enq          42064 non-null  int64
 28  enq_L3m                        42064 non-null  int64
 29  NETMONTHLYINCOME               42064 non-null  int64
 30  Time_With_Curr_Empr            42064 non-null  int64
 31  CC_Flag                        42064 non-null  int64
 32  PL_Flag                        42064 non-null  int64
 33  pct_PL_enq_L6m_of_ever         42064 non-null  float64
 34  pct_CC_enq_L6m_of_ever         42064 non-null  float64
 35  HL_Flag                        42064 non-null  int64
```

```
 40  MARITALSTATUS_Single          42064 non-null  bool
 41  GENDER_F                      42064 non-null  bool
 42  GENDER_M                      42064 non-null  bool
 43  last_prod_enq2_AL             42064 non-null  bool
 44  last_prod_enq2_CC             42064 non-null  bool
 45  last_prod_enq2_ConsumerLoan   42064 non-null  bool
 46  last_prod_enq2_HL             42064 non-null  bool
 47  last_prod_enq2_PL             42064 non-null  bool
 48  last_prod_enq2_others         42064 non-null  bool
 49  first_prod_enq2_AL            42064 non-null  bool
 50  first_prod_enq2_CC            42064 non-null  bool
 51  first_prod_enq2_ConsumerLoan  42064 non-null  bool
 52  first_prod_enq2_HL            42064 non-null  bool
 53  first_prod_enq2_PL            42064 non-null  bool
 54  first_prod_enq2_others        42064 non-null  bool
dtypes: bool(16), float64(5), int64(33), object(1)
memory usage: 13.2+ MB
```

```
df_encoded
```

	pct_tl_open_L6M	pct_tl_closed_L6M	Tot_TL_closed_L12M	pct_tl_closed_L12M	Tot_Missed_Pmnt	CC_
0	0.000	0.00	0	0.000	0	
1	0.000	0.00	0	0.000	0	
2	0.125	0.00	0	0.000	1	
3	0.000	0.00	0	0.000	0	
4	0.000	0.00	1	0.167	0	
...	...	...	...	...	...	
42059	0.333	0.00	0	0.000	0	
42060	0.000	0.25	1	0.250	0	
42061	0.500	0.50	1	0.500	0	
42062	0.000	0.00	1	0.500	0	
42063	0.000	0.00	0	0.000	0	

42064 rows × 55 columns

Machine Learing model fitting

Data processing

```
# 1. Random Forest

y = df_encoded['Approved_Flag']
x = df_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)
y_pred = rf_classifier.predict(x_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print ()
print(f'Accuracy: {accuracy}')
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: 0.7636990372043266

Class p1:
Precision: 0.8370457209847597
Recall: 0.7041420118343196
F1 Score: 0.7648634172469202

Class p2:
Precision: 0.7957519116397621
Recall: 0.9282457879088206
F1 Score: 0.856907593778591

Class p3:
Precision: 0.4423380726698262
Recall: 0.21132075471698114
F1 Score: 0.28600612870275793

Class p4:
Precision: 0.7178502879078695
Recall: 0.7269193391642371
F1 Score: 0.7223563495895703
```

```
# 2. xgboost

import xgboost as xgb
from sklearn.preprocessing import LabelEncoder

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



y = df_encoded['Approved_Flag']
x = df_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: 0.78

Class p1:
Precision: 0.823906083244397
Recall: 0.7613412228796844
F1 Score: 0.7913890312660175

Class p2:
Precision: 0.8255418233924413
Recall: 0.913577799801784
F1 Score: 0.8673315769665035

Class p3:
Precision: 0.4756380510440835
Recall: 0.30943396226415093
F1 Score: 0.37494284407864653

Class p4:
Precision: 0.7342386032977691
Recall: 0.7356656948493683
F1 Score: 0.7349514563106796
```

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


y = df_encoded['Approved_Flag']
x = df_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: 0.71

Class p1:
Precision: 0.7210682492581603
Recall: 0.7189349112426036
F1 Score: 0.72

Class p2:
Precision: 0.8082961814305097
Recall: 0.8265609514370664
F1 Score: 0.8173265386123089

Class p3:
Precision: 0.3394276629570747
Recall: 0.3222641509433962
F1 Score: 0.33062330623306235

Class p4:
Precision: 0.6558375634517767
Recall: 0.6277939747327502
F1 Score: 0.6415094339622641
```

xgboost is giving me best results

We will further finetune it

```
# Apply standard scaler

from sklearn.preprocessing import StandardScaler

columns_to_be_scaled = ['Age_Oldest_TL','Age_Newest_TL','time_since_recent_payment',
'max_recent_level_of_deliq','recent_level_of_deliq',
'time_since_recent_enq','NETMONTHLYINCOME','Time_With_Curr_Empr']

for i in columns_to_be_scaled:
    column_data = df_encoded[i].values.reshape(-1, 1)
    scaler = StandardScaler()
    scaled_column = scaler.fit_transform(column_data)
    df_encoded[i] = scaled_column
```

```
import xgboost as xgb
from sklearn.preprocessing import LabelEncoder

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

```
y = df_encoded['Approved_Flag']
x = df_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(f'Accuracy: {accuracy:.2f}')


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: 0.78
Class p1:
Precision: 0.823906083244397
Recall: 0.7613412228796844
F1 Score: 0.7913890312660175

Class p2:
Precision: 0.8255418233924413
Recall: 0.913577799801784
F1 Score: 0.8673315769665035

Class p3:
Precision: 0.4756380510440835
Recall: 0.30943396226415093
F1 Score: 0.37494284407864653

Class p4:
Precision: 0.7342386032977691
Recall: 0.7356656948493683
F1 Score: 0.7349514563106796
```

No improvement in metrices

```
# 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_job
grid_search.fit(x_train, y_train)

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

# 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}
```

```
Best Hyperparameters: {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 200}
Test Accuracy: 0.7811719957209081
```

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

```
# 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 = df_encoded['Approved_Flag']
x = df_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


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
print(f"Train Accuracy: {train_accuracy:.2f}")
print(f"Test Accuracy : {test_accuracy :.2f}")
print("-" * 30)
```

```
colsample_bytree: 0.9, learning_rate: 0.01, max_depth: 8, alpha: 1, n_estimators: 50
Train Accuracy: 0.80
Test Accuracy : 0.77
------------------------------
Combination 633
colsample_bytree: 0.9, learning_rate: 0.01, max_depth: 8, alpha: 1, n_estimators: 100
Train Accuracy: 0.81
Test Accuracy : 0.77
------------------------------
Combination 634
colsample_bytree: 0.9, learning_rate: 0.01, max_depth: 8, alpha: 10, n_estimators: 10
Train Accuracy: 0.78
Test Accuracy : 0.76
------------------------------
Combination 635
colsample_bytree: 0.9, learning_rate: 0.01, max_depth: 8, alpha: 10, n_estimators: 50
Train Accuracy: 0.79
Test Accuracy : 0.77
------------------------------
Combination 636
colsample_bytree: 0.9, learning_rate: 0.01, max_depth: 8, alpha: 10, n_estimators: 100
Train Accuracy: 0.79
Test Accuracy : 0.77
------------------------------
Combination 637
colsample_bytree: 0.9, learning_rate: 0.01, max_depth: 8, alpha: 100, n_estimators: 10
Train Accuracy: 0.75
Test Accuracy : 0.74
------------------------------
Combination 638
colsample_bytree: 0.9, learning_rate: 0.01, max_depth: 8, alpha: 100, n_estimators: 50
Train Accuracy: 0.75
Test Accuracy : 0.74
------------------------------
Combination 639
colsample_bytree: 0.9, learning_rate: 0.01, max_depth: 8, alpha: 100, n_estimators: 100
Train Accuracy: 0.75
Test Accuracy : 0.74
------------------------------
Combination 640
colsample_bytree: 0.9, learning_rate: 0.01, max_depth: 10, alpha: 1, n_estimators: 10
Train Accuracy: 0.83
Test Accuracy : 0.77
------------------------------
Combination 641
colsample_bytree: 0.9, learning_rate: 0.01, max_depth: 10, alpha: 1, n_estimators: 50
Train Accuracy: 0.83
Test Accuracy : 0.77
------------------------------
Combination 642
```

```
# Find the best row based on highest test accuracy
best_row = results_df.loc[results_df['test_Accuracy'].idxmax()]

print("Best Hyperparameter Combination")
print(best_row)
```

```
Best Hyperparameter Combination
combination         398.000000
train_Accuracy        0.807762
test_Accuracy         0.783430
colsample_bytree      0.500000
learning_rate         1.000000
max_depth             3.000000
alpha                 1.000000
n_estimators         50.000000
```

```
Name: 397, dtype: float64
```

```
import pandas as pd

# Convert dictionary to DataFrame
results_df = pd.DataFrame(answers_grid)

# Save as CSV
results_df.to_csv("xgboost_hyperparameter_results.csv", index=False)

# Save as Excel (optional)
results_df.to_excel("xgboost_hyperparameter_results.xlsx", index=False)

# Display the top few rows
results_df.head()
```

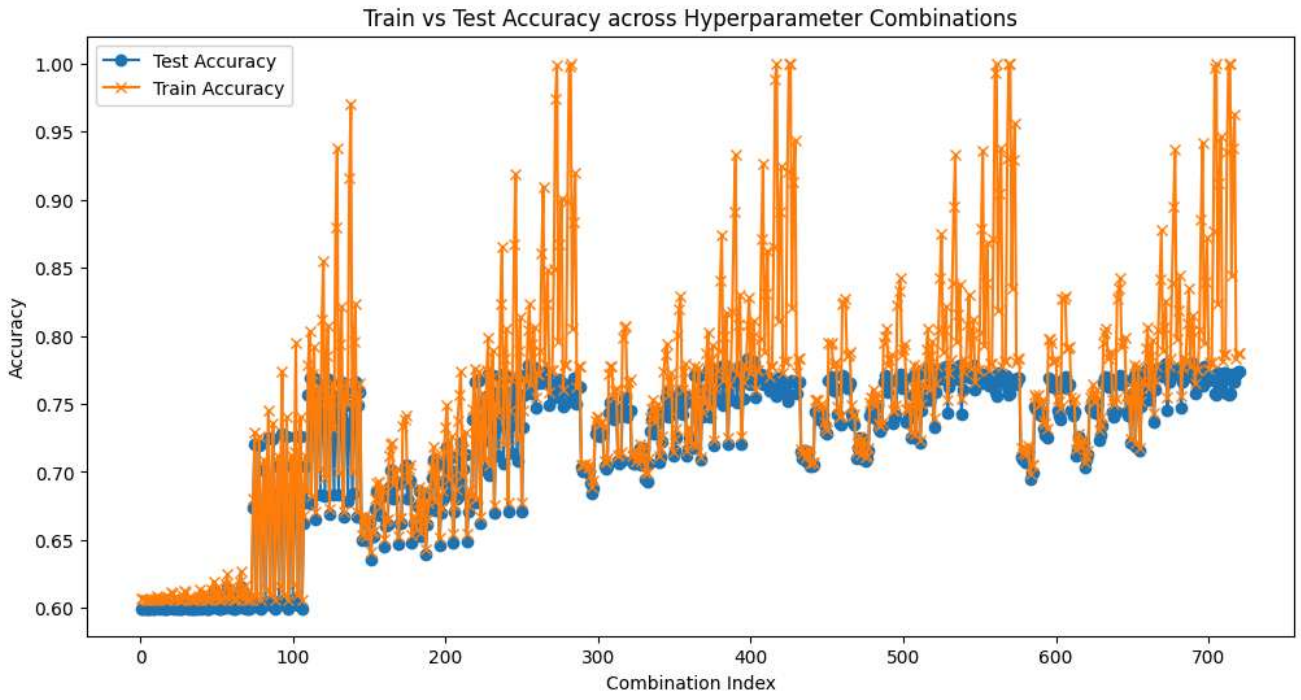
	combination	train_Accuracy	test_Accuracy	colsample_bytree	learning_rate	max_depth	alpha	n_esti
0	1	0.606609	0.599667	0.1	0.001	3	1	
1	2	0.606609	0.599667	0.1	0.001	3	1	
2	3	0.606520	0.599667	0.1	0.001	3	1	
3	4	0.606460	0.599667	0.1	0.001	3	10	
4	5	0.606520	0.599667	0.1	0.001	3	10	

Next steps: Generate code with `results_df` View recommended plots New interactive sheet

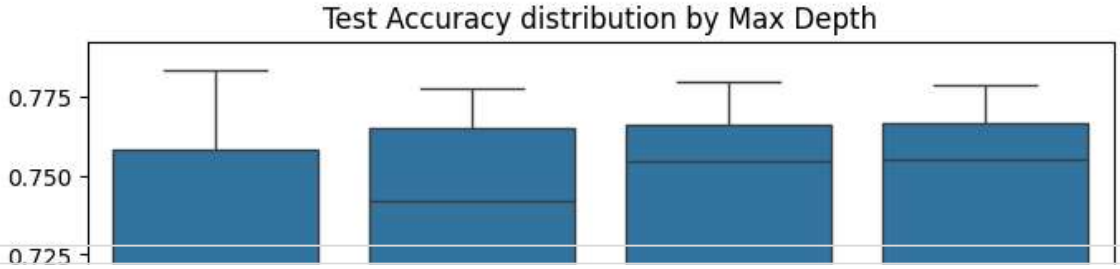
```
from google.colab import files
files.download("xgboost_hyperparameter_results.csv")
```

```
import pandas as pd
from google.colab import files

# Convert dictionary to DataFrame
results_df = pd.DataFrame(answers_grid)

# Find the row with the highest test accuracy
best_index = results_df['test_Accuracy'].idxmax()

# Apply highlight style
def highlight_best(row):
    return ['background-color: yellow' if row.name == best_index else '' for _ in row]

styled_df = results_df.style.apply(highlight_best, axis=1)

# Save styled DataFrame to Excel
excel_file = "xgboost_hyperparameter_results.xlsx"
styled_df.to_excel(excel_file, index=False, engine='openpyxl')

# Download to your computer
files.download(excel_file)
```

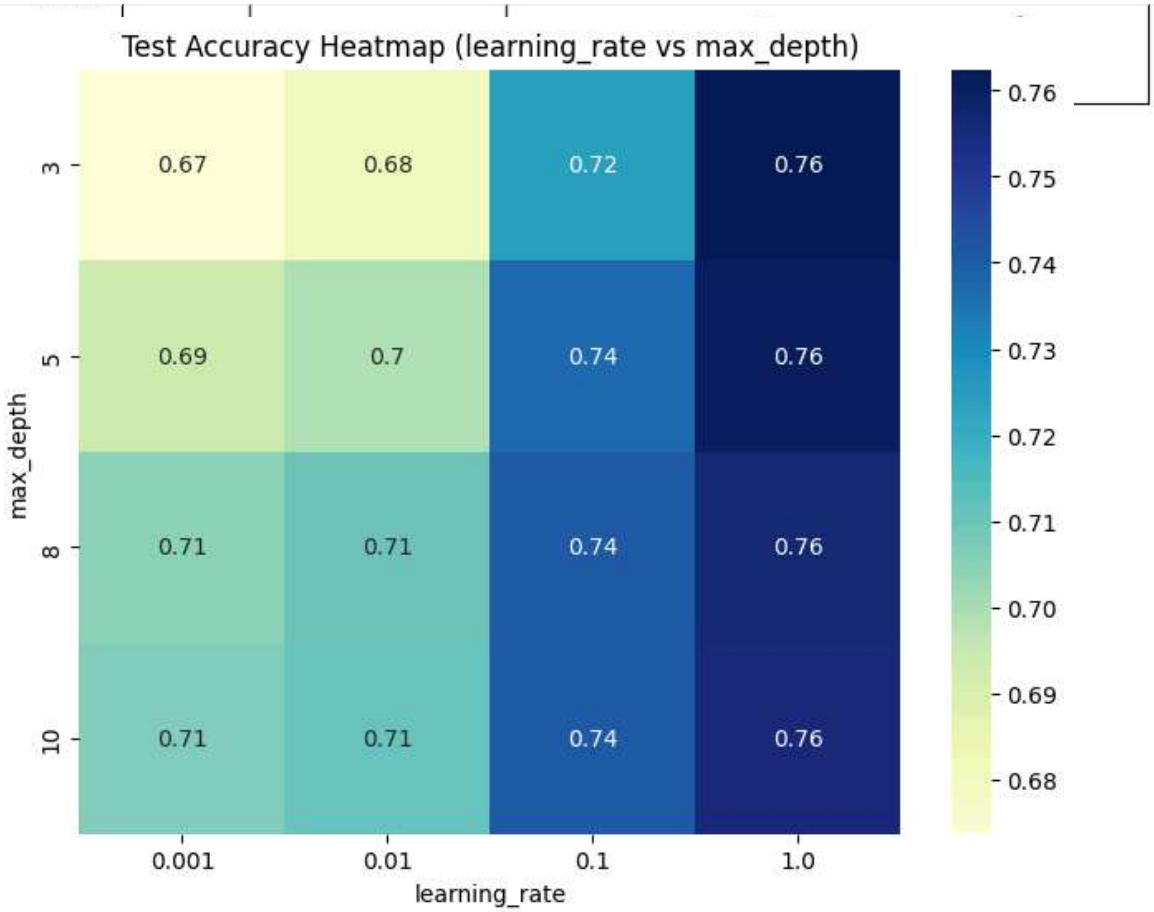
```
import matplotlib.pyplot as plt

plt.figure(figsize=(12,6))
plt.plot(results_df['combination'], results_df['test_Accuracy'], marker='o', label='Test Accuracy')
plt.plot(results_df['combination'], results_df['train_Accuracy'], marker='x', label='Train Accuracy')
plt.xlabel("Combination Index")
plt.ylabel("Accuracy")
plt.title("Train vs Test Accuracy across Hyperparameter Combinations")
plt.legend()
plt.show()
```



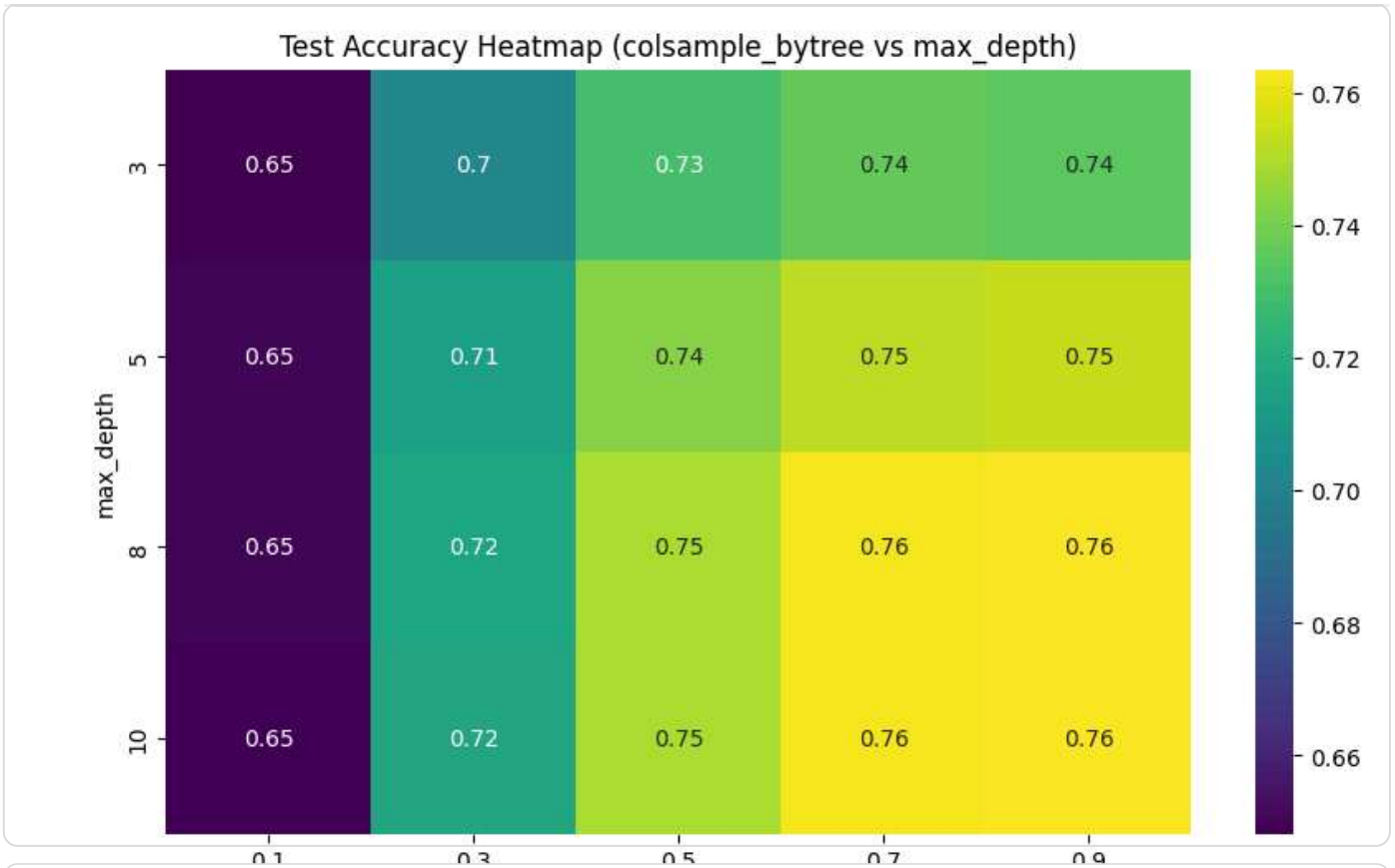

```
import seaborn as sns

plt.figure(figsize=(8,5))
sns.boxplot(x='max_depth', y='test_Accuracy', data=results_df)
plt.title("Test Accuracy distribution by Max Depth")
plt.show()
```




```
pivot = results_df.pivot_table(values='test_Accuracy',
                               index='max_depth',
                               columns='learning_rate')

plt.figure(figsize=(8,6))
sns.heatmap(pivot, annot=True, cmap="YlGnBu")
plt.title("Test Accuracy Heatmap (learning_rate vs max_depth)")
plt.show()
```




```
pivot2 = results_df.pivot_table(values='test_Accuracy',
                                index='max_depth',
                                columns='colsample_bytree')

plt.figure(figsize=(10,6))
sns.heatmap(pivot2, annot=True, cmap="viridis")
plt.title("Test Accuracy Heatmap (colsample_bytree vs max_depth)")
plt.show()
```



Start coding or generate with AI.