There is some wordings in Indonesian because the main aim of the study was for thesis.

Install library

```
pip install prince
```

```
pip install kmodes
```

```
!pip install catboost
```

```
pip install scikit-optimize
```

```
import os; os.kill(os.getpid(),9)
```

Import packages

```
import numpy  as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams
import seaborn as sns
import missingno as msno
from numpy import mean, std
from sklearn import cluster
from sklearn.cluster import KMeans
from kmodes.kmodes import KModes
from kmodes import kprototypes
from sklearn.preprocessing import OneHotEncoder
from sklearn.decomposition import PCA
from sklearn.preprocessing import KBinsDiscretizer
import prince
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from xgboost import plot_importance, plot_tree
from sklearn.metrics import classification_report, mean_squared_error, f1_score, accuracy_score, silhouette_score
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold, StratifiedKFold
from catboost import CatBoostClassifier, Pool
from sklearn.feature_selection import SelectKBest, chi2
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTEN, SMOTENC
from imblearn.under_sampling import RandomUnderSampler
import lightgbm
from lightgbm import LGBMClassifier
import time
import skopt
from skopt import BayesSearchCV
from skopt.space import Real, Integer, Categorical
```

Export data

```
data = pd.read_csv('train.csv')
data = data.drop(columns='Id')
x = data.drop(columns='Response')
y = data['Response']
data.tail()
```

```
plt.figure(figsize=(9, 5))
ax = sns.countplot(data=data, x='Response')
for label in ax.containers:
    ax.bar_label(label)
plt.xlabel('Level Risiko')
plt.ylabel('Jumlah');
```

Descriptive Analysis

```
data["Response"] = data["Response"].replace([1,2,3,4,5,6,7,8],[0,1,2,2,2,3,4,5])
```

```
plt.figure(figsize=(9, 5))
ax = sns.countplot(data=data, x='Response')
for label in ax.containers:
    ax.bar_label(label)
plt.xlabel('Level Risiko')
plt.ylabel('Jumlah');
```

```
data['Response'].value_counts()
```

```
sns.boxplot(data=data, x='Response', y='Ins_Age',palette='Set1').set_title("Boxplot Persebaran Level Risiko Terhadap Usia")
```

```
sns.boxplot(data=data, x='Response', y='Ht', palette='Set2').set_title("Boxplot Persebaran Level Risiko Terhadap Tinggi Ba
```

```
sns.boxplot(data=data, x='Response', y='Wt', palette='Set3').set_title("Boxplot Persebaran Level Risiko Terhadap Berat Bad
```

```
sns.boxplot(data=data, x='Response', y='BMI', palette='Set1').set_title("Boxplot Persebaran Level Risiko Terhadap BMI")
```

Missing value imputation

```
data = data.drop(columns=['Family_Hist_2','Family_Hist_3','Family_Hist_5','Medical_History_10','Medical_History_15','Medic
```

```
data.tail()
```

```
ave = ['Employment_Info_1','Employment_Info_4','Employment_Info_6','Insurance_History_5','Family_Hist_4']
for column in ave:
    data[column].fillna(value=data[column].mean(), inplace=True)
```

```
data['Medical_History_1'].fillna(value=round(data['Medical_History_1'].median()), inplace=True)
```

```
data['Medical_History_1']
```

Conversion of numerical feature into categorical

```
data['Product_Info_4'] = pd.cut(data.Product_Info_4, bins=3,labels=['1','2','3'], include_lowest=True)
data['Employment_Info_1'] = pd.cut(data.Employment_Info_1, bins=3,labels=['1','2','3'], include_lowest=True)
data['Employment_Info_4'] = pd.cut(data.Employment_Info_4, bins=3,labels=['1','2','3'], include_lowest=True)
data['Employment_Info_6'] = pd.cut(data.Employment_Info_6, bins=3,labels=['1','2','3'], include_lowest=True)
data['Insurance_History_5'] = pd.cut(data.Insurance_History_5, bins=3,labels=['1','2','3'], include_lowest=True)
data['Family_Hist_4'] = pd.cut(data.Family_Hist_4, bins=3,labels=['1','2','3'], include_lowest=True)
data['Ins_Age'] = pd.cut(data.Ins_Age, bins=3,labels=['1','2','3'], include_lowest=True)
data['Ht'] = pd.cut(data.Ht, bins=3,labels=['1','2','3'], include_lowest=True)
data['Wt'] = pd.cut(data.Wt, bins=3,labels=['1','2','3'], include_lowest=True)
data['BMI'] = pd.cut(data.BMI, bins=3,labels=['1','2','3'], include_lowest=True)
data['Medical_History_1'] = pd.cut(data.Medical_History_1, bins=3,labels=['1','2','3'], include_lowest=True)
```

Combining data with one-hot encoding

```
#one-hot encoding
encoder = OneHotEncoder(handle_unknown='ignore')

#perform one-hot encoding
cat = data[['Product_Info_4','Employment_Info_1','Employment_Info_4','Employment_Info_6','Insurance_History_5','Family_Hist
num = data.drop(['Product_Info_4','Employment_Info_1','Employment_Info_4','Employment_Info_6','Insurance_History_5','Family

encoder_data = encoder.fit(cat)
column_name = encoder.get_feature_names_out(['Product_Info_4','Employment_Info_1','Employment_Info_4','Employment_Info_6',
encoder_data2 = pd.DataFrame(encoder.fit_transform(cat).toarray(), columns=column_name)

#merge one-hot encoded columns back with original DataFrame
final_data = num.join(encoder_data2)

#view final df
print(final_data)
```

```
data2 = pd.DataFrame(final_data)
```

```
data2 = data2.astype('category')
```

```
data2.tail()
```

Feature selection with chi-squared

```
X_select = data2.drop(columns='Response')
y_select = data2['Response']
```

```
sf = SelectKBest(chi2, k='all')
sf_fit = sf.fit(X_select, y_select)
for i in range(len(sf_fit.scores_)):
    print(' %s: %f' % (X_select.columns[i], sf_fit.scores_[i]))
```

```
chi_scores = chi2(X_select,y_select)
```

```
p_values = pd.Series(chi_scores[1])
chiselection = []
for i in range (0,910):
  if p_values[i]<=0.05:
    chiselection.append(X_select.columns[i])
```

```
len(chiselection)
```

```
data31 = data2[chiselection]
data3 = pd.concat([data31, data2['Response']], axis=1)
```

```
print(data3)
```

K-fold cross validation formula

```
def evaluate_model(X, y, model):
  # define evaluation procedure
  cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=1, random_state=42)
  # evaluate model
  scores = cross_val_score(model, X, y, scoring='f1_weighted', cv=cv, n_jobs=-1)
  return scores
```

Condition Evaluation: 6 Risk Levels

```
n = [150, 100, 50]
eta = [0.05, 0.1, 0.15]
```

```
X = data2.drop(columns='Response')
y = data2['Response']
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=42, test_size=0.2)
X1 = Xtrain.astype('int')
y1 = ytrain.astype('int')
cat_var = list(X1.columns)
```

```
X_2 = data3.drop(columns='Response')
y_2 = data3['Response']
Xtrain2, Xtest2, ytrain2, ytest2 = train_test_split(X_2, y_2, random_state=42, test_size=0.2)
X_21 = Xtrain2.astype('int')
y_21 = ytrain2.astype('int')
cat_var2 = list(X_21.columns)
```

```
ytrain2.value_counts()
```

```
ytrain.value_counts()
```

```
for i in range(0,3):
  model_1_ada = AdaBoostClassifier(algorithm="SAMME", n_estimators=n[i], learning_rate = eta[i])
  model_1_xg = XGBClassifier(booster='gbtree', objective='binary:logistic', tree_method='hist', enable_categorical = True,
                             n_estimators=n[i], learning_rate = eta[i])
  model_1_lgb = LGBMClassifier(boosting_type= 'gbdt', objective= 'multiclassova', data_sample_strategy= 'goss',
                               num_class= 6, top_rate= 0.5, other_rate= 0.5, n_estimators=n[i], learning_rate = eta[i])
  model_1_cat = CatBoostClassifier(boosting_type='Ordered', loss_function='MultiClassOneVsAll', cat_features = cat_var,
                                   classes_count=6, n_estimators=n[i], learning_rate=eta[i])
  stada = time.time()
  score_1_ada = evaluate_model(Xtrain, ytrain, model_1_ada)
```

```
    etada = time.time()
    stxg = time.time()
    score_1_xg = evaluate_model(Xtrain, ytrain, model_1_xg)
    etxg = time.time()
    stlgb = time.time()
    score_1_lgb = evaluate_model(Xtrain, ytrain, model_1_lgb)
    etlgb = time.time()
    stcat = time.time()
    score_1_cat = evaluate_model(X1, y1, model_1_cat)
    etcat = time.time()
    print('Untuk n: ',n[i],' dan eta: ',eta[i])
    print('Mean Ada: %.4f (%.4f)' % (mean(score_1_ada), std(score_1_ada)))
    print('Execution time Ada:', etada-stada, 'seconds')
    print('Mean XG: %.4f (%.4f)' % (mean(score_1_xg), std(score_1_xg)))
    print('Execution time XG:', etxg-stxg, 'seconds')
    print('Mean LGB: %.4f (%.4f)' % (mean(score_1_lgb), std(score_1_lgb)))
    print('Execution time LGB:', etlgb-stlgb, 'seconds')
    print('Mean cat: %.4f (%.4f)' % (mean(score_1_cat), std(score_1_cat)))
    print('Execution time Cat:', etcat-stcat, 'seconds')
```

```
import matplotlib.pyplot as plt
x = ["n:150, eta:0.05", "n:100, eta:0.1", "n:50, eta:0.15"]
ada = [0.2767, 0.2776, 0.2766]
stdada = [0.0033, 0.0031, 0.0033]
xg = [0.4643, 0.4666, 0.4638]
stdxg = [0.0018, 0.0029, 0.0024]
lgb = [0.4713, 0.4721, 0.4710]
stdlgb = [0.0016, 0.0024, 0.0032]
cat = [0.4418, 0.4488, 0.4414]
stdcat = [0.0022, 0.0031, 0.0018]

#plt.figure(figsize=(8, 6))
plt.xlabel("Kombinasi jumlah iterasi dan learning rate")
plt.ylabel("Weighted F1")
plt.ylim(0.2500, 0.5000)
plt.yticks([0.25,0.275,0.30,0.325,0.35,0.375,0.4,0.425,0.45,0.475,0.50])

plt.errorbar(x, ada, yerr=stdada, fmt='.', capsize=4, color='g', label='Multi-class AdaBoost', ls='none')
plt.errorbar(x, xg, yerr=stdxg, fmt='.', capsize=4, color='r', label='XGBoost', ls='none')
plt.errorbar(x, lgb, yerr=stdlgb, fmt='.', capsize=4, color='b', label='LightGBM', ls='none')
plt.errorbar(x, cat, yerr=stdcat, fmt='.', capsize=4, color='k',  label='CatBoost', ls='none')
plt.grid()
plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
plt.show()
```

```
import matplotlib.pyplot as plt
x = ["n:150, eta:0.05", "n:100, eta:0.1", "n:50, eta:0.15"]
xg = [0.4643, 0.4666, 0.4638]
stdxg = [0.0018, 0.0029, 0.0024]
lgb = [0.4713, 0.4721, 0.4710]
stdlgb = [0.0016, 0.0024, 0.0032]
cat = [0.4418, 0.4488, 0.4414]
stdcat = [0.0022, 0.0031, 0.0018]

#plt.figure(figsize=(8, 6))
plt.xlabel("Kombinasi jumlah iterasi dan learning rate")
plt.ylabel("Weighted F1")
plt.ylim(0.4300, 0.4900)

plt.errorbar(x, xg, yerr=stdxg, fmt='.', capsize=4, color='r', label='XGBoost', ls='none')
plt.errorbar(x, lgb, yerr=stdlgb, fmt='.', capsize=4, color='b', label='LightGBM', ls='none')
plt.errorbar(x, cat, yerr=stdcat, fmt='.', capsize=4, color='k',  label='CatBoost', ls='none')
plt.grid()
plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
plt.show()
```

Condition Evaluation: 6 Risk Levels + Feature Selection

```
for i in range(0,3):
  model_2_ada = AdaBoostClassifier(algorithm="SAMME", n_estimators=n[i], learning_rate = eta[i])
  model_2_xg = XGBClassifier(booster='gbtree', objective='binary:logistic', tree_method='hist', enable_categorical = True,
                             n_estimators=n[i], learning_rate = eta[i])
  model_2_lgb = LGBMClassifier(boosting_type= 'gbdt', objective= 'multiclassova', data_sample_strategy= 'goss',
                               num_class= 6, top_rate= 0.5, other_rate= 0.5, n_estimators=n[i], learning_rate = eta[i])
  model_2_cat = CatBoostClassifier(boosting_type='Ordered', loss_function='MultiClassOneVsAll', cat_features = cat_var2,
                                   classes_count=6, n_estimators=n[i], learning_rate=eta[i])
  stada = time.time()
  score_2_ada = evaluate_model(Xtrain2, ytrain2, model_2_ada)
  etada = time.time()
```

```
    stxg = time.time()
    score_2_xg = evaluate_model(Xtrain2, ytrain2, model_2_xg)
    etxg = time.time()
    stlgb = time.time()
    score_2_lgb = evaluate_model(Xtrain2, ytrain2, model_2_lgb)
    etlgb = time.time()
    stcat = time.time()
    score_2_cat = evaluate_model(X_21, y_21, model_2_cat)
    etcat = time.time()
    print('Untuk n: ',n[i],' dan eta: ',eta[i])
    print('Mean Ada with red: %.4f (%.4f)' % (mean(score_2_ada), std(score_2_ada)))
    print('Execution time Ada:', etada-stada, 'seconds')
    print('Mean XG with red: %.4f (%.4f)' % (mean(score_2_xg), std(score_2_xg)))
    print('Execution time XG:', etxg-stxg, 'seconds')
    print('Mean LGB with red: %.4f (%.4f)' % (mean(score_2_lgb), std(score_2_lgb)))
    print('Execution time LGB:', etlgb-stlgb, 'seconds')
    print('Mean cat with red: %.4f (%.4f)' % (mean(score_2_cat), std(score_2_cat)))
    print('Execution time Cat:', etcat-stcat, 'seconds')
```

```
x = ["n:150, eta:0.05", "n:100, eta:0.1", "n:50, eta:0.15"]
ada = [0.2767, 0.2776, 0.2766]
stdada = [0.0033, 0.0031, 0.0033]
xg = [0.4644, 0.4671, 0.4635]
stdxg = [0.0028, 0.0031, 0.0026]
lgb = [0.4711, 0.4705, 0.4707]
stdlgb = [0.0022, 0.0023, 0.0031]
cat = [0.4445, 0.4517, 0.4446]
stdcat = [0.0024, 0.0041, 0.0019]

#plt.figure(figsize=(8, 6))
plt.xlabel("Kombinasi jumlah iterasi dan learning rate")
plt.ylabel("Weighted F1")
plt.ylim(0.2500, 0.5000)
plt.yticks([0.25,0.275,0.30,0.325,0.35,0.375,0.4,0.425,0.45,0.475,0.50])

plt.errorbar(x, ada, yerr=stdada, fmt='.', capsize=4, color='g', label='Multi-class AdaBoost', ls='none')
plt.errorbar(x, xg, yerr=stdxg, fmt='.', capsize=4, color='r', label='XGBoost', ls='none')
plt.errorbar(x, lgb, yerr=stdlgb, fmt='.', capsize=4, color='b', label='LightGBM', ls='none')
plt.errorbar(x, cat, yerr=stdcat, fmt='.', capsize=4, color='k',  label='CatBoost', ls='none')
plt.grid()
plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
plt.show()
```

```
x = ["n:150, eta:0.05", "n:100, eta:0.1", "n:50, eta:0.15"]
xg = [0.4644, 0.4671, 0.4635]
stdxg = [0.0028, 0.0031, 0.0026]
lgb = [0.4711, 0.4705, 0.4707]
stdlgb = [0.0022, 0.0023, 0.0031]
cat = [0.4445, 0.4517, 0.4446]
stdcat = [0.0024, 0.0041, 0.0019]

#plt.figure(figsize=(8, 6))
plt.xlabel("Kombinasi jumlah iterasi dan learning rate")
plt.ylabel("Weighted F1")
plt.ylim(0.4300, 0.4900)

plt.errorbar(x, xg, yerr=stdxg, fmt='.', capsize=4, color='r', label='XGBoost', ls='none')
plt.errorbar(x, lgb, yerr=stdlgb, fmt='.', capsize=4, color='b', label='LightGBM', ls='none')
plt.errorbar(x, cat, yerr=stdcat, fmt='.', capsize=4, color='k',  label='CatBoost', ls='none')
plt.grid()
plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
plt.show()
```

Cluster Evaluation

```
#ELBOW METHOD K MODES
cost=[]
K = range(1,10)
for num_clusters in K :
 kmode = KModes(n_clusters=num_clusters, init = "random", n_init = 5, verbose=1)
 kmode.fit_predict(data3.drop(columns='Response'))
 cost.append(kmode.cost_)

plt.plot(K,cost,'bx-')
plt.xlabel('Jumlah klaster')
plt.ylabel('Cost')
plt.show()
```

```
#silhoutte score
```

```
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]
silhouette_avg = []
for num_clusters in range_n_clusters:

 # initialise
 kmeans = KMeans(n_clusters=num_clusters)
 kmeans.fit(data3.drop(columns='Response'))
 cluster_labels = kmeans.labels_

 silhouette_avg.append(silhouette_score(data3.drop(columns='Response'), cluster_labels))

plt.plot(range_n_clusters,silhouette_avg,'bx-')
plt.xlabel('Jumlah klaster')
plt.ylabel('Silhouette score')
plt.show()
```

```
Traindata = pd.concat([Xtrain2, ytrain2], axis=1)
Traindata.insert(298, 'Type', 'train')
Testdata = pd.concat([Xtest2, ytest2], axis=1)
Testdata.insert(298, 'Type', 'test')
Forcluster = pd.concat([Traindata,Testdata], ignore_index=False)
```

```
Forcluster
```

```
#KModes for clustering
kmode = KModes(n_clusters=2, init = "random", n_init = 5, verbose=1)
clusters = kmode.fit_predict(Forcluster.drop(columns=['Response','Type']))
```

```
dataclust2 = Forcluster
dataclust2['label'] = clusters
dataclust2
```

```
cluster1 = dataclust2.loc[dataclust2['label'] == 0]
cluster2 = dataclust2.loc[dataclust2['label'] == 1]
cluster1['Response'].value_counts()
```

```
cluster2['Response'].value_counts()
```

```
for i in range (0,2):
  klaster = dataclust2.loc[(dataclust2['label'] == i) & (dataclust2['Type'] == 'train')]
  X_c = klaster.drop(columns=['Response','label','Type'])
  y_c = klaster['Response']
  X_c1 = X_c.astype('int')
  y_c1 = y_c.astype('int')
  cat_varc = list(X_c1.columns)
  print('Untuk klaster ',i, ' dengan reduksi')
  for j in range (0,3):
    model_2_ada = AdaBoostClassifier(algorithm="SAMME", n_estimators=n[j], learning_rate = eta[j])
    model_2_xg = XGBClassifier(booster='gbtree', objective='binary:logistic', tree_method='hist', enable_categorical = True
                         n_estimators=n[j], learning_rate = eta[j])
    model_2_lgb = LGBMClassifier(boosting_type= 'gbdt', objective= 'multiclassova', data_sample_strategy= 'goss',
                          num_class= 6, top_rate= 0.5, other_rate= 0.5, n_estimators=n[j], learning_rate = eta[j])
    model_2_cat = CatBoostClassifier(boosting_type='Ordered', loss_function='MultiClassOneVsAll', cat_features = cat_varc,
                              classes_count=6, n_estimators=n[j], learning_rate=eta[j])
    stada = time.time()
    score_c_ada = evaluate_model(X_c, y_c, model_2_ada)
    etada = time.time()
    stxg = time.time()
    score_c_xg = evaluate_model(X_c, y_c, model_2_xg)
    etxg = time.time()
    stlgb = time.time()
    score_c_lgb = evaluate_model(X_c, y_c, model_2_lgb)
    etlgb = time.time()
    stcat = time.time()
    score_c_cat = evaluate_model(X_c1, y_c1, model_2_cat)
    etcat = time.time()
    print('Untuk n: ',n[j],' dan eta: ',eta[j])
    print('Mean Ada: %.4f (%.4f)' % (mean(score_c_ada), std(score_c_ada)))
    print('Execution time Ada:', etada-stada, 'seconds')
    print('Mean XG: %.4f (%.4f)' % (mean(score_c_xg), std(score_c_xg)))
    print('Execution time XG:', etxg-stxg, 'seconds')
    print('Mean LGB: %.4f (%.4f)' % (mean(score_c_lgb), std(score_c_lgb)))
    print('Execution time LGB:', etlgb-stlgb, 'seconds')
    print('Mean cat: %.4f (%.4f)' % (mean(score_c_cat), std(score_c_cat)))
    print('Execution time Cat:', etcat-stcat, 'seconds')
```

```
x = ['1','2','3']
ada = [0.2767, 0.2776, 0.2766]
ada1 = [0.2921, 0.2959, 0.3047]
ada2 = [0.2697, 0.2728, 0.2695]
xg = [0.4644, 0.4671, 0.4635]
xg1 = [0.4532, 0.4542, 0.4529]
xg2 = [0.4617, 0.4636, 0.4633]
lgb = [0.4711, 0.4705, 0.4707]
lgb1 = [0.4600, 0.4604, 0.4603]
lgb2 = [0.4699, 0.4729, 0.4713]
cat = [0.4445, 0.4517, 0.4446]
cat1 = [0.4270, 0.4365, 0.4262]
cat2 = [0.4483, 0.4556, 0.4476]

fig, axs = plt.subplots(2, 2, figsize=(9,7))
axs[0, 0].errorbar(x, ada, color='c', marker='o', label='All', ls='none')
axs[0, 0].errorbar(x, ada1, color='m', marker='o', label='Klaster 1', ls='none')
axs[0, 0].errorbar(x, ada2, color='y', marker='o', label='Klaster 2', ls='none')
axs[0, 0].set_title('Multi-class AdaBoost')
axs[0, 0].grid()
axs[0, 1].errorbar(x, xg, color='c', marker='o', label='All', ls='none')
axs[0, 1].errorbar(x, xg1, color='m', marker='o', label='Klaster 1', ls='none')
axs[0, 1].errorbar(x, xg2, color='y', marker='o', label='Klaster 2', ls='none')
axs[0, 1].set_title('XGBoost')
axs[0, 1].grid()
axs[1, 0].errorbar(x, lgb, color='c', marker='o', label='All', ls='none')
axs[1, 0].errorbar(x, lgb1, color='m', marker='o', label='Klaster 1', ls='none')
axs[1, 0].errorbar(x, lgb2, color='y', marker='o', label='Klaster 2', ls='none')
axs[1, 0].set_title('LightGBM')
axs[1, 0].grid()
axs[1, 1].errorbar(x, cat, color='c', marker='o', label='All', ls='none')
axs[1, 1].errorbar(x, cat1, color='m', marker='o', label='Klaster 1', ls='none')
axs[1, 1].errorbar(x, cat2, color='y', marker='o', label='Klaster 2', ls='none')
axs[1, 1].set_title('CatBoost')
axs[1, 1].grid()

plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
```

Optimised Model

```
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=1, random_state=42)
```

```
lgbbase = LGBMClassifier(boosting_type= 'gbdt', objective= 'multiclassova', data_sample_strategy= 'goss', num_class= 6, ve
```

```
stlgbf = time.time()
opt = BayesSearchCV(lgbbase,
                     {
                         "learning_rate": Real(0.01,0.3),
                         "n_estimators": Integer(20,300),
                         "max_depth": Integer(3, 13),
                         "num_leaves": Integer(20, 200),
                         "min_child_samples": Integer(7, 75),
                         "colsample_bytree": Real(0.25, 1),
                         "reg_alpha": Real(0, 1),
                         "reg_lambda": Real(0, 1),
                         "min_split_gain": Real(0, 0.5),
                         "top_rate": Real(0.4, 0.6),
                         "other_rate": Real(0.1, 0.4)
                     },
                     n_iter = 50,
                     cv = cv,
                     n_jobs = -1,
                     scoring = "f1_weighted",
                     random_state = 42,
                     verbose = 0
                    )

opt.fit(Xtrain2, ytrain2)
etlgbf = time.time()
print("Best Score is: ", opt.best_score_, "\n")
print("Best Parameters: ", opt.best_params_, "\n")
lgbfinal = opt.best_estimator_
lgbfinal
print('Execution time LGB:', etlgbf-stlgbf, 'seconds')
```

XGBoost Optimised Model

```
xgbbase = XGBClassifier(booster='gbtree', objective='binary:logistic', tree_method='hist', enable_categorical = True)
```

```
stxgf = time.time()
opt2 = BayesSearchCV(xgbbase,
                     {
                         "learning_rate": Real(0.01,0.3),
                         "n_estimators": Integer(20,300),
                         "max_depth": Integer(3, 13),
                         "max_leaves": Integer(20, 200),
                         "colsample_bytree": Real(0.25, 1),
                         "subsample": Real(0.25, 1),
                         "reg_alpha": Real(0, 1),
                         "reg_lambda": Real(0, 1),
                         "gamma": Real(0, 0.5)
                     },
                     n_iter = 50,
                     cv = cv,
                     n_jobs = -1,
                     scoring = "f1_weighted",
                     random_state = 42,
                     verbose = 0
                    )

opt2.fit(Xtrain2, ytrain2)
etxgf = time.time()
print("Best Score is: ", opt2.best_score_, "\n")
print("Best Parameters: ", opt2.best_params_, "\n")
xgfinal = opt2.best_estimator_
xgfinal
print('Execution time XG:', etxgf-stxgf, 'seconds')
```

Holdout Testing

```
#LightGBM
lightgbm_model = LGBMClassifier(boosting_type= 'gbdt', objective= 'multiclassova', data_sample_strategy= 'goss', num_class
                                colsample_bytree= 0.6618759946739794, learning_rate= 0.10673149877796027, max_depth= 13, m
                                min_split_gain= 0.5, n_estimators= 171, num_leaves= 20, other_rate= 0.4, reg_alpha= 1.0, r
                                top_rate= 0.5227311646552896)
fitlgb = lightgbm_model.fit(Xtrain2,ytrain2)
y_pred_lgb = fitlgb.predict(Xtest2)
print(classification_report(ytest2, y_pred_lgb))
```

```
lightgbm.plot_importance(fitlgb, importance_type="gain", figsize=(7,6), title="LightGBM Feature Importance (Gain)", max_nu
plt.show()
```

```
#XGBoost
xgboost_model = XGBClassifier(booster='gbtree', objective='binary:logistic', tree_method='hist', enable_categorical = True
                              gamma= 0.0, learning_rate= 0.11256616601357686, max_depth= 7, max_leaves= 200, n_estimators=
                              reg_alpha= 0.5296958737760243, reg_lambda= 0.792820600601485, subsample= 1.0)
fitxgb = xgboost_model.fit(Xtrain2,ytrain2)
y_pred_xgb = fitxgb.predict(Xtest2)
print(classification_report(ytest2, y_pred_xgb))
```

```
plot_importance(fitxgb, importance_type="total_gain" ,title="XGBoost Feature Importance (Gain)", max_num_features=10)
plt.show()
```