The goal of this project is to develop a machine learning model for predicting customer churn in a telco company. Customer churn refers to the phenomenon where customers terminate their relationship with a company and switch to a competitor or discontinue using the service altogether. By analyzing various customer attributes and service usage patterns, we aim to create a classification model that can accurately predict whether a customer is likely to churn or not.

```
import pandas as pd
import numpy as np
import lightgbm as lgb
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest  #for fearure selecti
from sklearn.feature_selection import mutual_info_classif     #for fear
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,ConfusionMatrixDispla
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier,StackingClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from matplotlib import pyplot as plt
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE,RandomOverSampler
from sklearn.neural_network import MLPClassifier
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecur
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	`
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	`
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	`
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	
...	...	...	...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	Yes	DSL	`
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic	
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No phone service	DSL	`
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	Yes	Fiber optic	
7042	3186-AJIEK	Male	0	No	No	66	Yes	No	Fiber optic	`

7043 rows × 21 columns

The dataset used for this project is a sample dataset of Telco Customer Churn, inspired by the original dataset "Telco customer churn (11.1.3+)" from IBM Business Analytics Community. It has been cleaned and aggregated for the purpose of telco customer churn analysis and prediction. The dataset is obtained from Kaggle.com.

```
df=pd.read_csv("/content/drive/MyDrive/data_luminar/WA_Fn-UseC_-Telco-Customer-Churn.csv")
df
```

The objective of this project is to build a classification model that can predict customer churn based on the given set of features. By analyzing the historical data and customer behavior patterns, the model will learn to identify the factors that contribute to churn and make accurate predictions for new customers. The ultimate aim is to assist the telco company in taking proactive measures to retain customers and improve customer satisfaction.

```
print("------------Visualizing the class Imbalance----------")
y=df["Churn"]
print(sns.countplot(data=df,y="Churn"))
df["Churn"].value_counts()
```

```
------------Visualizing the class Imbalance----------
Axes(0.125,0.11;0.775x0.77)
No     5174
Yes    1869
Name: Churn, dtype: int64
```


```
print(df.isna().sum())
df.dtypes
```

```
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

```
customerID           object
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object
```

Data Preprocessing: Perform data cleaning, handle missing values, and preprocess the categorical and numerical features as required.

```
df=df.drop(["customerID"],axis=1)
df['MultipleLines'] = df['MultipleLines'].replace('No phone service', 'No')
df
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBa
0	Female	0	Yes	No	1	No	No	DSL	No	
1	Male	0	No	No	34	Yes	No	DSL	Yes	
2	Male	0	No	No	2	Yes	No	DSL	Yes	
3	Male	0	No	No	45	No	No	DSL	Yes	
4	Female	0	No	No	2	Yes	No	Fiber optic	No	
...	...	...	...	...	...	...	...	...	...	
7038	Male	0	Yes	Yes	24	Yes	Yes	DSL	Yes	
7039	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic	No	
7040	Female	0	Yes	Yes	11	No	No	DSL	Yes	

```
le=LabelEncoder()
lst=["gender","Partner","OnlineSecurity","MonthlyCharges",
"Dependents",
"PhoneService",
"MultipleLines",
"InternetService",
"OnlineBackup",
"DeviceProtection",
"TechSupport",
"StreamingTV",
"StreamingMovies",
```

```
"Contract",
"PaperlessBilling","PaymentMethod","TotalCharges","Churn"]
for i in lst:
  df[i]=le.fit_transform(df[i])
df.dtypes
```

```
gender              int64
SeniorCitizen       int64
Partner             int64
Dependents          int64
tenure              int64
PhoneService        int64
MultipleLines       int64
InternetService     int64
OnlineSecurity      int64
OnlineBackup        int64
DeviceProtection    int64
TechSupport         int64
StreamingTV         int64
StreamingMovies     int64
Contract            int64
PaperlessBilling    int64
PaymentMethod       int64
MonthlyCharges      int64
TotalCharges        int64
Churn               int64
dtype: object
```

```
df.corr()
```

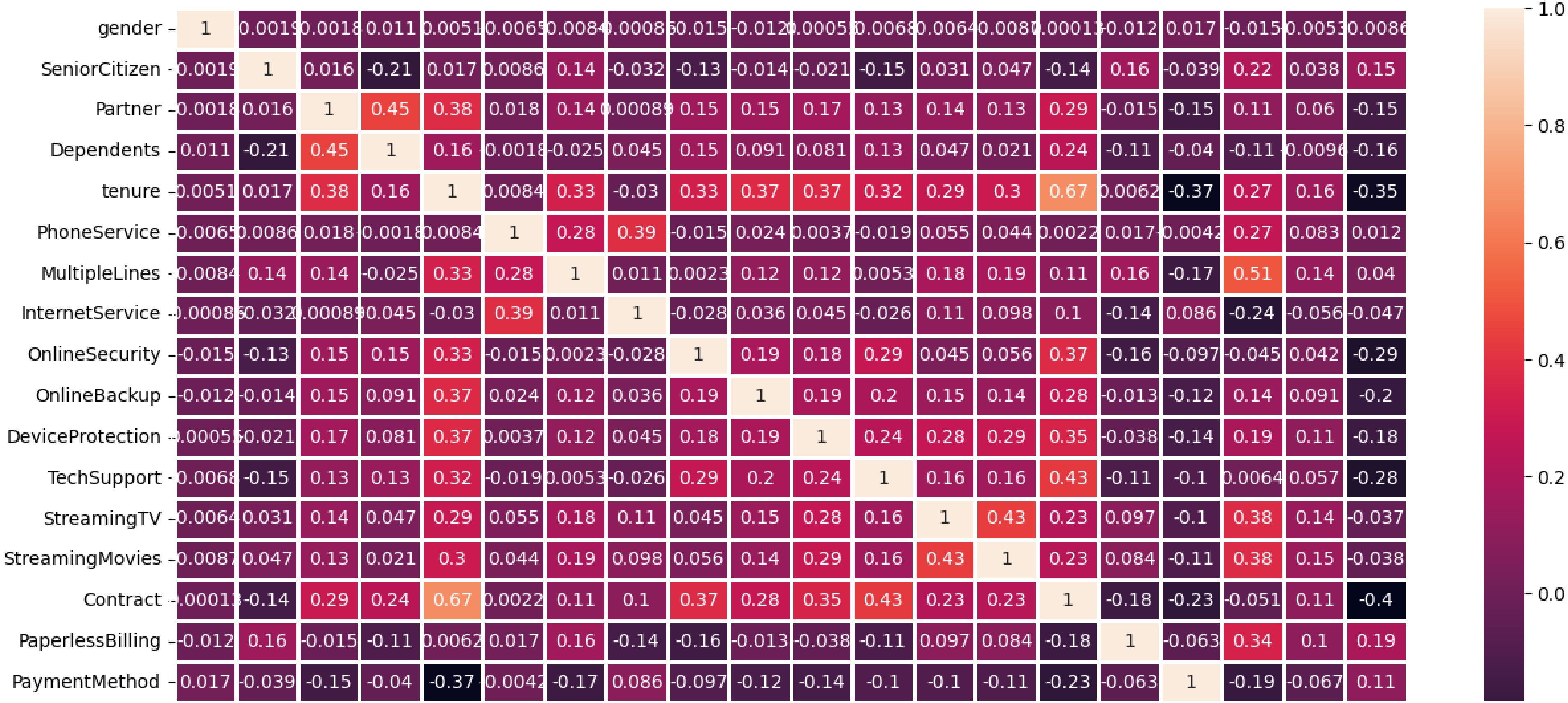
	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineS
gender	1.000000	-0.001874	-0.001808	0.010517	0.005106	-0.006488	-0.008414	-0.000863	-(
SeniorCitizen	-0.001874	1.000000	0.016479	-0.211185	0.016567	0.008576	0.142948	-0.032310	-(
Partner	-0.001808	0.016479	1.000000	0.452676	0.379697	0.017706	0.142057	0.000891	(
Dependents	0.010517	-0.211185	0.452676	1.000000	0.159712	-0.001762	-0.024526	0.044590	(
tenure	0.005106	0.016567	0.379697	0.159712	1.000000	0.008448	0.331941	-0.030359	(
PhoneService	-0.006488	0.008576	0.017706	-0.001762	0.008448	1.000000	0.279690	0.387436	-(
MultipleLines	-0.008414	0.142948	0.142057	-0.024526	0.331941	0.279690	1.000000	0.011124	(
InternetService	-0.000863	-0.032310	0.000891	0.044590	-0.030359	0.387436	0.011124	1.000000	-(
OnlineSecurity	-0.015017	-0.128221	0.150828	0.152166	0.325468	-0.015198	0.002306	-0.028416	
OnlineBackup	-0.012057	-0.013632	0.153130	0.091015	0.370876	0.024105	0.119886	0.036138	(
DeviceProtection	0.000549	-0.021398	0.166330	0.080537	0.371105	0.003727	0.118577	0.044944	(
TechSupport	-0.006825	-0.151268	0.126733	0.133524	0.322942	-0.019158	0.005275	-0.026047	(
StreamingTV	-0.006421	0.030776	0.137341	0.046885	0.289373	0.055353	0.184681	0.107417	(
StreamingMovies	-0.008743	0.047266	0.129574	0.021321	0.296866	0.043870	0.186907	0.098350	(
Contract	0.000126	-0.142554	0.294806	0.243187	0.671607	0.002247	0.107114	0.099721	(
PaperlessBilling	-0.011754	0.156530	-0.014877	-0.111377	0.006152	0.016505	0.163530	-0.138625	-(
PaymentMethod	0.017352	0.038551	0.154798	0.040292	0.370436	0.004184	0.171026	0.086140	[illegible]

Exploratory Data Analysis: Analyze the dataset to gain insights into the distribution of features and their relationship with churn.

```
import warnings
print("----------finds potential relationships between variables and to understand the strength of these relationshi
plt.figure(figsize=[15,8])
```

```
warnings.filterwarnings("ignore")
plt.show(sns.heatmap(df.corr(),annot=True,linewidth=1))
```

```
----------finds potential relationships between variables and to understand the strength of these relationships.-----------
```




```
df=df.drop(["gender"],axis=1)
```

```
df.dtypes
```

```
SeniorCitizen       int64
Partner             int64
Dependents          int64
tenure              int64
PhoneService        int64
```

```
MultipleLines       int64
InternetService     int64
OnlineSecurity      int64
OnlineBackup        int64
DeviceProtection    int64
TechSupport         int64
StreamingTV         int64
StreamingMovies     int64
Contract            int64
PaperlessBilling    int64
PaymentMethod       int64
MonthlyCharges      int64
TotalCharges        int64
Churn               int64
dtype: object
```

```
X=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

```
#train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=341,test_size=0.3)
print(X_train.shape[0]," rows is present in X_train")
```

```
4930  rows is present in X_train
```

```
 # #feature selection
# bestfeatures=SelectKBest(score_func=mutual_info_classif,k=19)
# print("------------feature selection using the Information Gain method----------")
# scores = mutual_info_classif(X_train, y_train)
# # Create a list of feature names
# num_features = X_train.shape[1]
# feature_names = []
# for i in range(num_features):
```

```
#     feature_names.append(f'Feature_{i}')

# df_scores = pd.DataFrame({'Feature':feature_names , 'Score': scores})
# df_scores = df_scores.sort_values('Score', ascending=False)
# print(df_scores)
# print("-----------")
# num_columns = X.shape[1]
# column_names = [f'Feature_{i}' for i in range(num_columns)]
# column_name = X.columns[4]
# print(column_name)
```

-----------feature selection using the Information Gain method---------

```
Feature     Score
```

14 Feature_14 0.097724 4 Feature_4 0.071392 8 Feature_8 0.067328 11 Feature_11 0.057248 7 Feature_7 0.053660 16 Feature_16 0.048679 10 Feature_10 0.041474 17 Feature_17 0.039832 9 Feature_9 0.035735 12 Feature_12 0.032219 13 Feature_13 0.028845 15 Feature_15 0.023087 18 Feature_18 0.022439 3 Feature_3 0.015919 2 Feature_2 0.012065 6 Feature_6 0.011901 1 Feature_1 0.001936 0 Feature_0 0.001458 5 Feature_5 0.000346Looking at the provided feature scores,("gender") "Feature_0" has the lowest score of 0.002307. To potentially improve accuracy i consider to remove this particular column. even after removing features with low score accuracy is not improving so i droped this process

```
#Scaling
sc=MinMaxScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

Hyperparameter Tuning: Fine-tune the model's hyperparameters using techniques like grid search or random search to optimize its performance.

```
##HYPER PARAMETER TUNING
# from sklearn import feature_selection
# import warnings
# from sklearn.model_selection import GridSearchCV
# import warnings
# lgb_classifier=MLPClassifier()
# grid_vls={'hidden_layer_sizes': [(100,), (50, 50), (50, 25), (25, 25)],
#     'activation': ['relu', 'tanh'],
#     'solver': ['sgd', 'adam'],
#     'learning_rate': ['constant', 'adaptive']
#      }
# grid=GridSearchCV(estimator=lgb_classifier, param_grid=grid_vls,scoring="accuracy", cv=5,refit=True,return_train_s
# grid.fit(X_train, y_train)
# grid.best_params_
```

Model Selection and Training: Experiment with various classification algorithms such as logistic regression, decision trees, random forests, or gradient boosting, and train the model using appropriate evaluation metrics.

normal method

```

#To execute machine learning models in a more efficient and organized manner,
#a recommended approach involves utilizing a for loop to iteratively run the models.

rf=RandomForestClassifier()
ad=AdaBoostClassifier()
nn = MLPClassifier(activation='tanh',hidden_layer_sizes= (100,),learning_rate="constant",solver="sgd")
```

```
lg=lgb.LGBMClassifier(learning_rate= 0.1,n_estimators=50,num_leaves=20)
lr=LogisticRegression()
gb=GradientBoostingClassifier()
xgb=XGBClassifier()
svc=SVC(C=0.1,gamma=0.01,kernel='linear',random_state=2)
dt=DecisionTreeClassifier(criterion="entropy",max_depth= 5,min_samples_leaf=4,min_samples_split=2)
lst1=[ad,dt,gb,rf,lr,svc,xgb,lg,nn]
ac=[]
model=[]
for i in lst1:
  print('*'*20,i,'*'*20)
  i.fit(X_train,y_train)
  y_pred=i.predict(X_test)
  print(classification_report(y_test,y_pred))

  print(i,ConfusionMatrixDisplay.from_predictions(y_test,y_pred))

  correct_predictions = (y_pred == y_test).sum()
  accuracy = correct_predictions / len(y_test)
  print("Accuracy:", accuracy)
  print("_"*200)
  ac.append(accuracy)
  model.append(i)
```

```
******************** AdaBoostClassifier() ********************
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      1576
           1       0.66      0.55      0.60       537

    accuracy                           0.81      2113
   macro avg       0.76      0.73      0.74      2113
weighted avg       0.81      0.81      0.81      2113

AdaBoostClassifier() <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7bdb86a606a0>
Accuracy: 0.8140085186938003
```

```
******************** DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=4) ********************
              precision    recall  f1-score   support

           0       0.84      0.89      0.86      1576
           1       0.61      0.48      0.54       537

    accuracy                           0.79      2113
   macro avg       0.72      0.69      0.70      2113
weighted avg       0.78      0.79      0.78      2113

DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=4) <sklearn.metrics._plot.confusion_matrix.ConfusionM
Accuracy: 0.7893989588263133
```

```
******************** GradientBoostingClassifier() ********************
              precision    recall  f1-score   support

           0       0.86      0.91      0.88      1576
           1       0.67      0.55      0.61       537

    accuracy                           0.82      2113
   macro avg       0.76      0.73      0.74      2113
weighted avg       0.81      0.82      0.81      2113

GradientBoostingClassifier() <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7bdb8695d9c0>
Accuracy: 0.8168480832938949
```

```
******************** RandomForestClassifier() ********************
              precision    recall  f1-score   support
```

```
           0       0.84      0.90      0.87      1576
           1       0.64      0.50      0.56       537

    accuracy                           0.80      2113
   macro avg       0.74      0.70      0.72      2113
weighted avg       0.79      0.80      0.79      2113

RandomForestClassifier() <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7bdb869c4760>
Accuracy: 0.8021769995267393
```

```
******************** LogisticRegression() ********************
              precision    recall  f1-score   support

           0       0.85      0.90      0.88      1576
           1       0.65      0.55      0.60       537

    accuracy                           0.81      2113
   macro avg       0.75      0.72      0.74      2113
weighted avg       0.80      0.81      0.80      2113

LogisticRegression() <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7bdb86824f70>
Accuracy: 0.8097491717936584
```

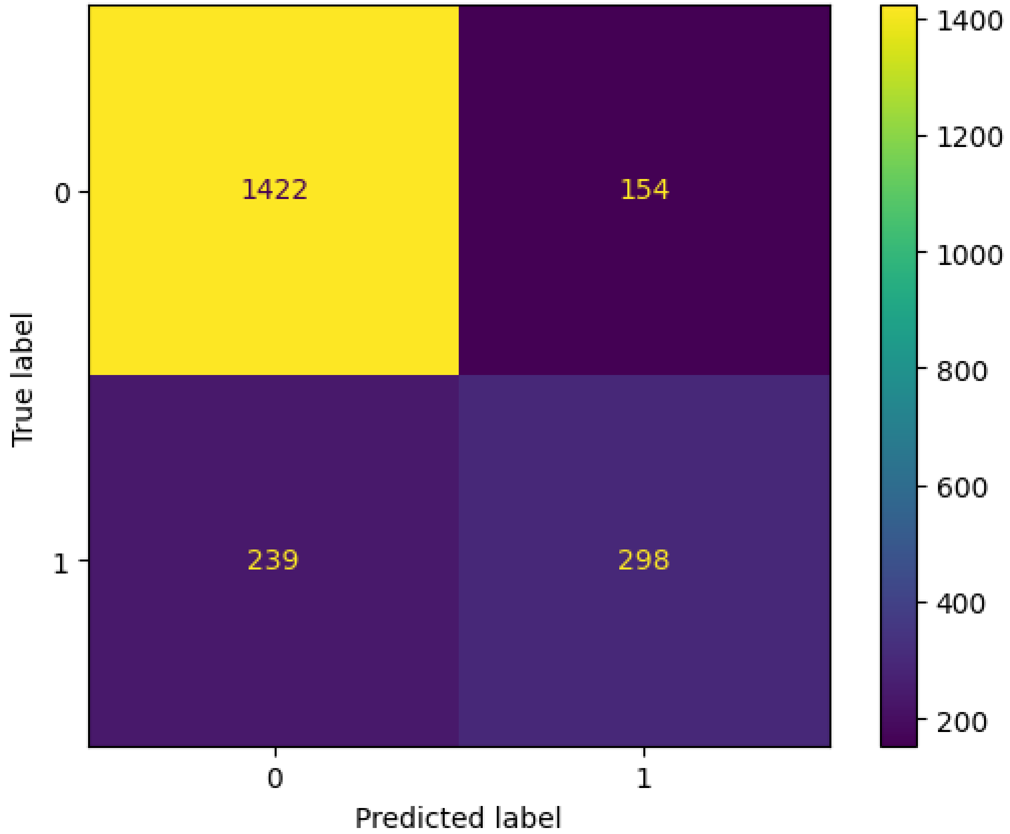
```
******************** SVC(C=0.1, gamma=0.01, kernel='linear', random_state=2) ********************
              precision    recall  f1-score   support

           0       0.85      0.90      0.87      1576
           1       0.65      0.54      0.59       537

    accuracy                           0.81      2113
   macro avg       0.75      0.72      0.73      2113
weighted avg       0.80      0.81      0.80      2113

SVC(C=0.1, gamma=0.01, kernel='linear', random_state=2) <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object a
Accuracy: 0.8078561287269286
```

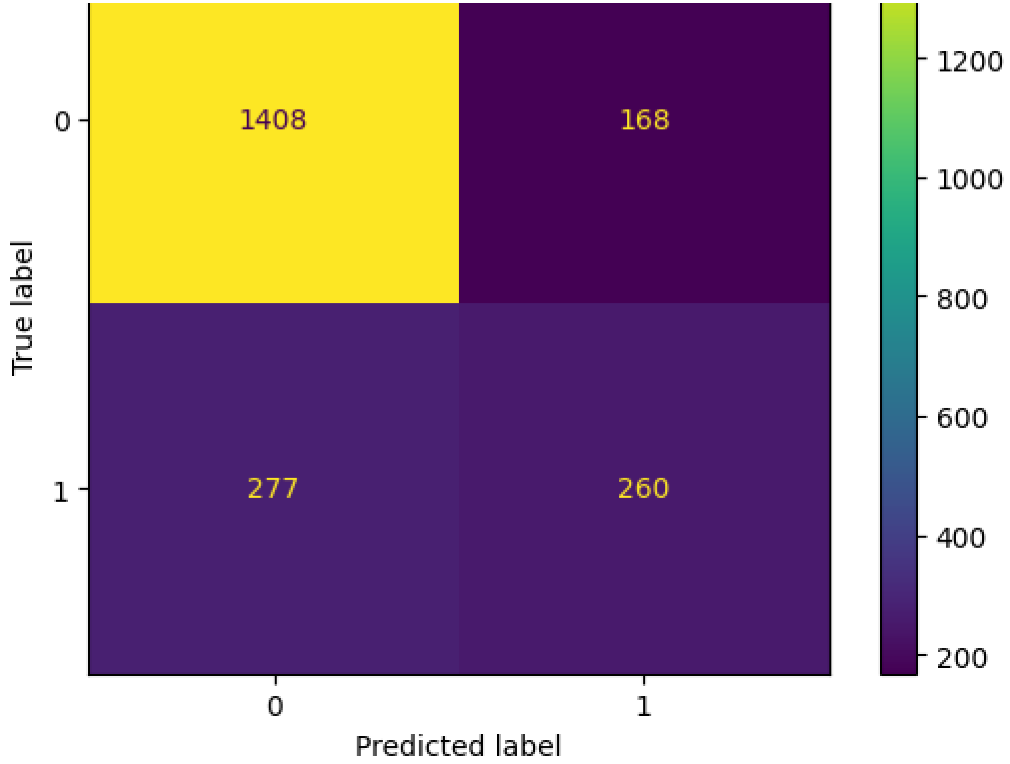
```
******************** XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
```

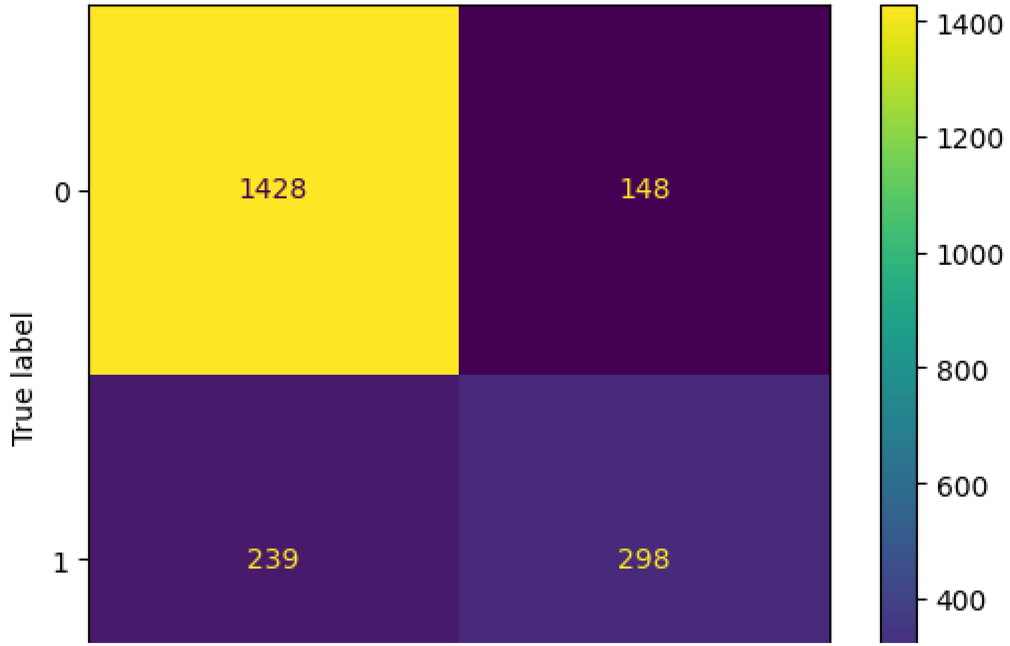
```
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...) ********************
              precision    recall  f1-score   support

           0       0.85      0.88      0.87      1576
           1       0.62      0.54      0.58       537

    accuracy                           0.80      2113
   macro avg       0.73      0.71      0.72      2113
weighted avg       0.79      0.80      0.79      2113

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...) <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at
Accuracy: 0.7974443918599148
```

```
******************** LGBMClassifier(n_estimators=50, num_leaves=20) ********************
              precision    recall  f1-score   support

           0       0.85      0.90      0.88      1576
           1       0.66      0.54      0.60       537

    accuracy                           0.81      2113
   macro avg       0.76      0.72      0.74      2113
weighted avg       0.80      0.81      0.81      2113

LGBMClassifier(n_estimators=50, num_leaves=20) <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7bdb8
Accuracy: 0.812588736393753
```

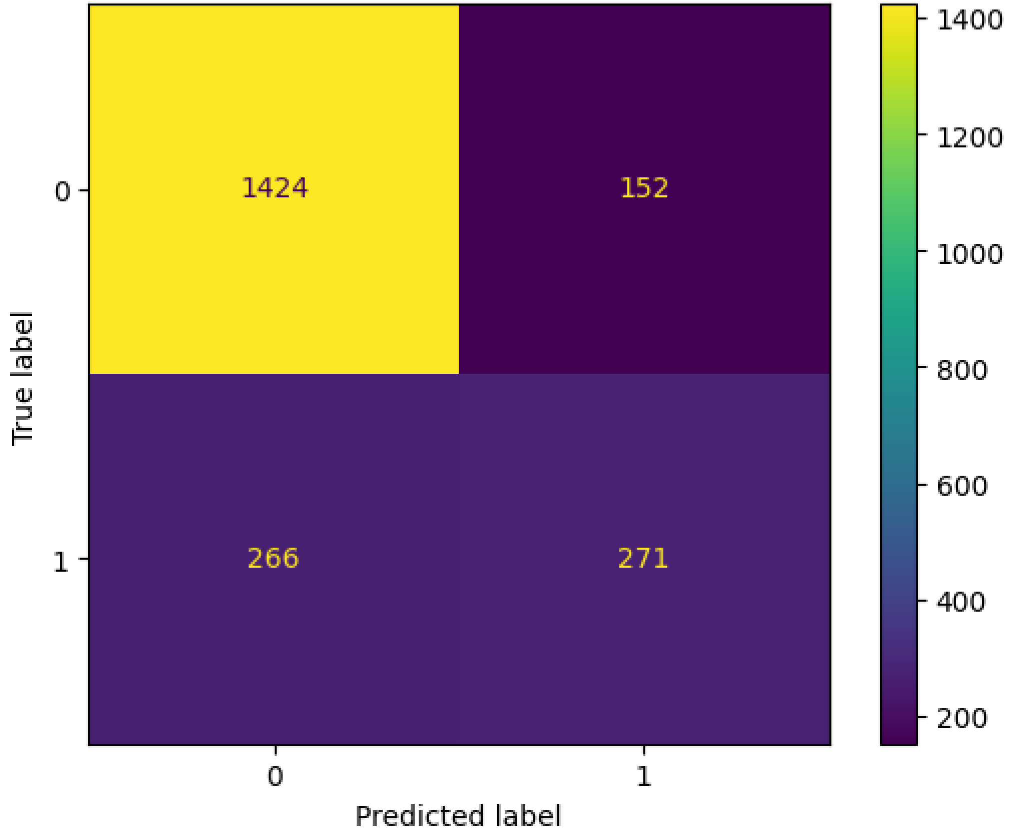
```
******************** MLPClassifier(activation='tanh', solver='sgd') ********************
              precision    recall  f1-score   support

           0       0.85      0.90      0.87      1576
           1       0.64      0.52      0.57       537

    accuracy                           0.80      2113
   macro avg       0.74      0.71      0.72      2113
weighted avg       0.79      0.80      0.80      2113

MLPClassifier(activation='tanh', solver='sgd') <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7bdb8
Accuracy: 0.8045433033601515
```

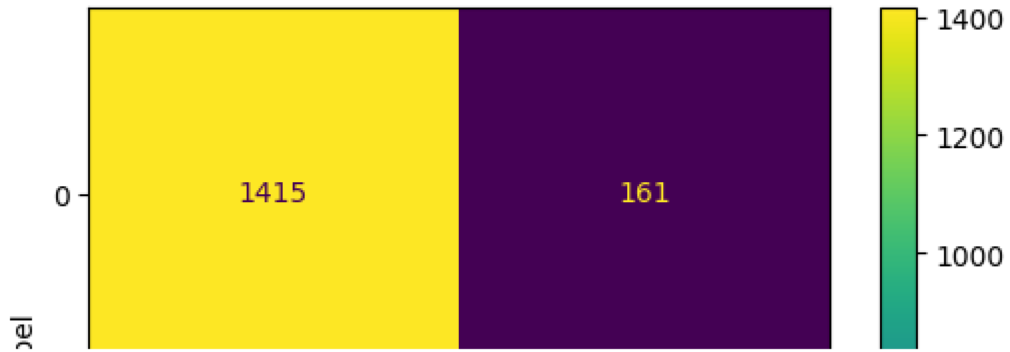


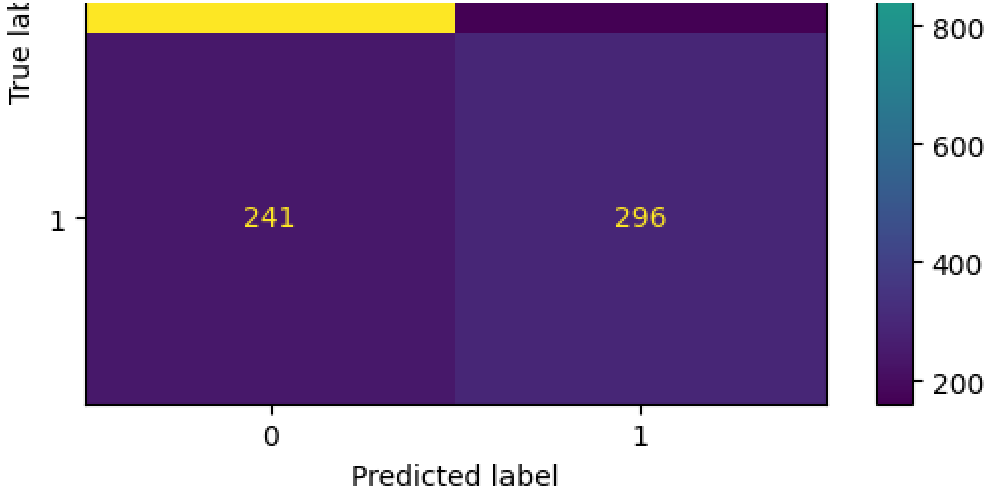

1408
168
277
260
True label
Predicted label
0
1
0
1
1200
1000
800
600
400
200

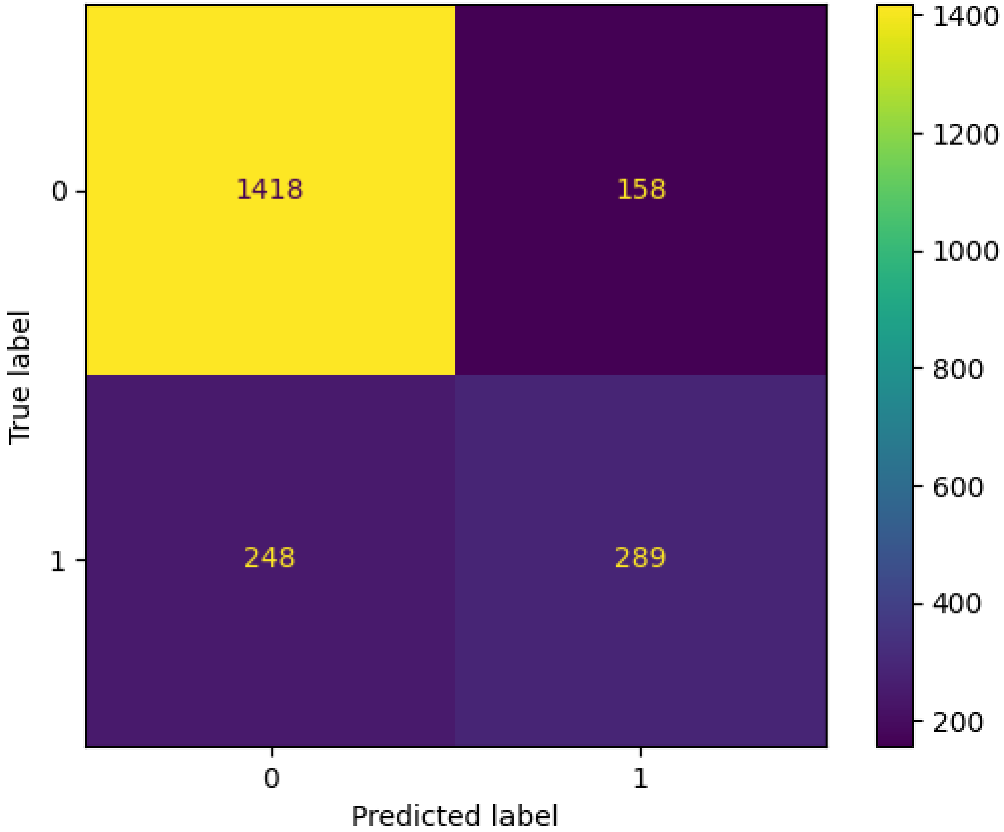

1428
148
239
298
True label
0
1
1400
1200
1000
800
600
400

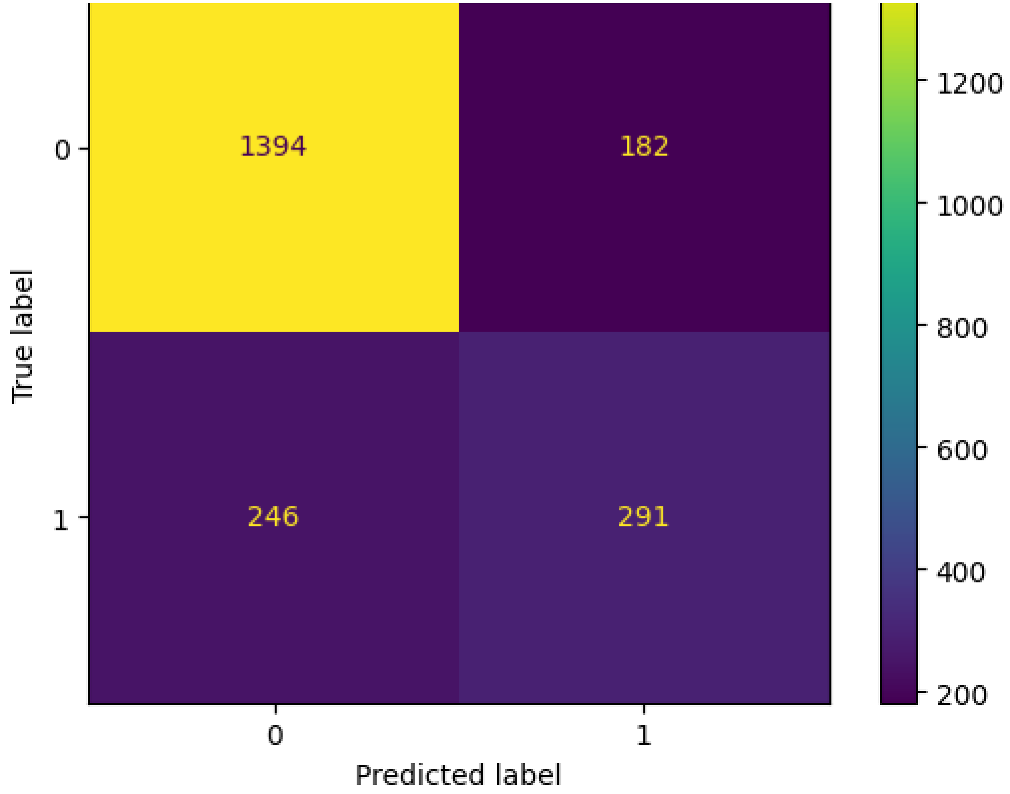

200
0
1
Predicted label

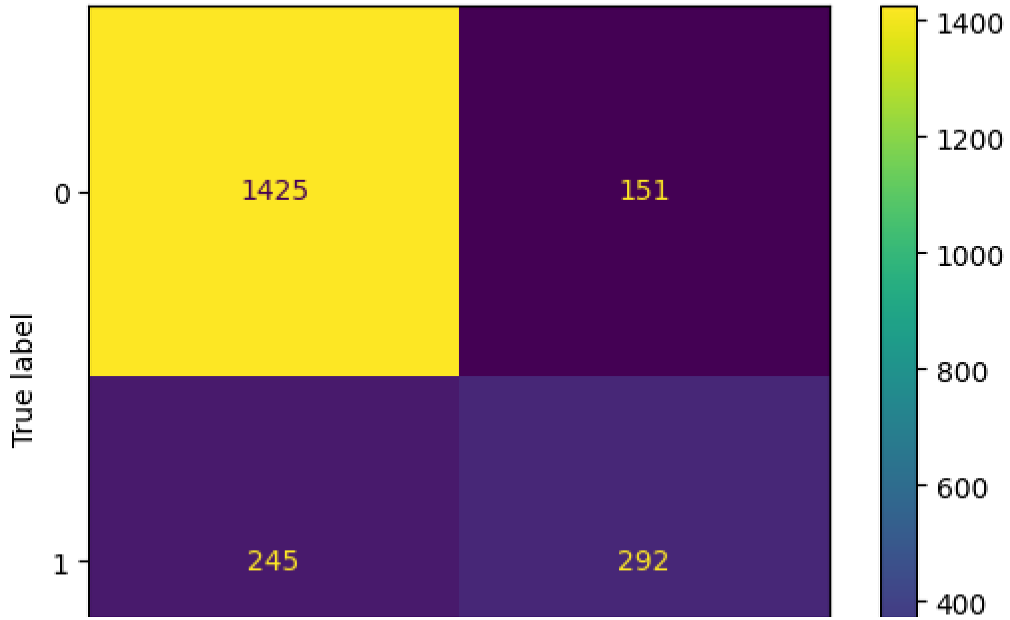

True label
0
1
1424
152
266
271
1400
1200
1000
800
600
400
200
0
1
Predicted label

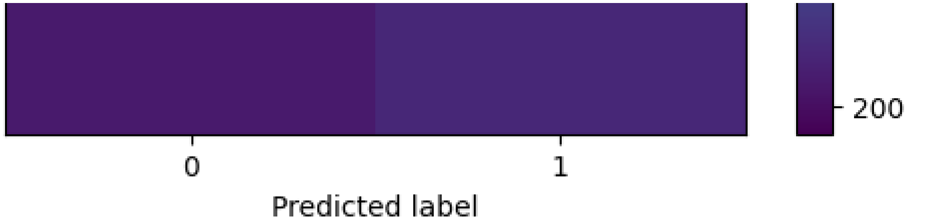

0
1415
161
1400
1200
1000

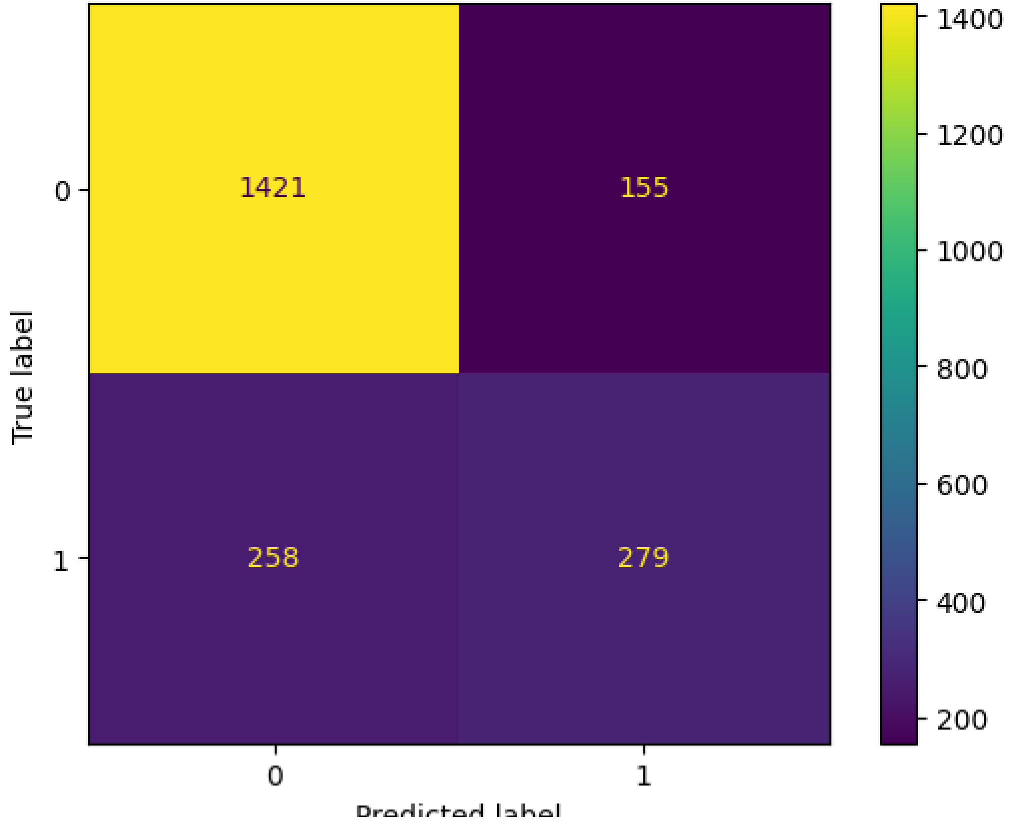

True lat
1
241
296
0
1
Predicted label
800
600
400
200


True label
0
1
1418
158
248
289
0
1
Predicted label
1400
1200
1000
800
600
400
200


1394
182
246
291
0
1
True label
0
1
Predicted label
1200
1000
800
600
400
200


1425
151
245
292
0
1
True label
1400
1200
1000
800
600
400

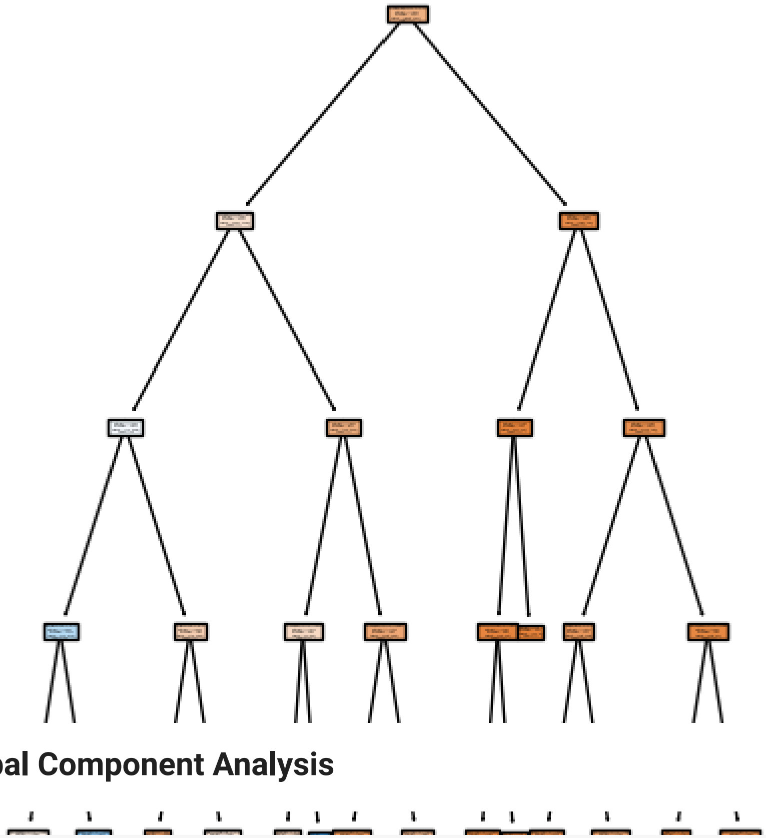




DECISION TREE

```
#DECISION TREE
plt.figure(figsize=(5,8))
plt.title("Customer_Churn_Decision_Tree")
plt.show(tree.plot_tree(dt,filled=True,
feature_names=["gender",
```

```
"SeniorCitizen",
"Partner",
"Dependents",
"tenure",
"PhoneService",
"MultipleLines",
"InternetService",
"OnlineSecurity",
"OnlineBackup",
"DeviceProtection",
"TechSupport",
"StreamingTV",
"StreamingMovies",
"Contract",
"PaperlessBilling",
"PaymentMethod",
"MonthlyCharges",
"TotalCharges",
"Churn"],
class_names=["no","yes"]
))
```



Principal Component Analysis

```
pca=PCA(n_components=3,random_state=3)
X_trainm=pca.fit_transform(X_train)
X_testm=pca.transform(X_test)
pca.explained_variance_ratio_
print(X_trainm.shape)
print(X_testm.shape)
```

```
(4930, 3)
(2113, 3)
```

```
rf=RandomForestClassifier(n_estimators=500,n_jobs=-1,max_depth=9,oob_score=True,random_state=25,max_samples=0.25,min
ad=AdaBoostClassifier()
nn = MLPClassifier(activation='tanh',hidden_layer_sizes= (100,),learning_rate="constant",solver="sgd")
lg=lgb.LGBMClassifier(learning_rate= 0.1,n_estimators=50,num_leaves=20)
lr=LogisticRegression(C=0.5,class_weight=None,dual=False,fit_intercept=True,max_iter=100,solver="saga",random_state=
gb=GradientBoostingClassifier()
xgb=XGBClassifier(learning_rate=0.01,max_depth=4,n_estimators=150)
svc=SVC(C=0.1,gamma=0.01,kernel='linear',random_state=2)
dt=DecisionTreeClassifier(criterion="entropy",max_depth= 5,min_samples_leaf=4,min_samples_split=2)
lst1=[dt,gb,rf,lr,svc,xgb,lg,nn]
acp=[]
modelp=[]
for p in lst1:
  print('*'*20,p,'*'*20)
  p.fit(X_trainm,y_train)
  y_pred=p.predict(X_testm)

  print(classification_report(y_test,y_pred))
  print(p,ConfusionMatrixDisplay.from_predictions(y_test,y_pred))

  correct_predictions = (y_pred == y_test).sum()
  accuracy = correct_predictions / len(y_test)
  print("_"*200)
  acp.append(accuracy)
  modelp.append(p)
```

```
******************* DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=4) *******************
              precision    recall  f1-score   support

           0       0.80      0.93      0.86      1576
           1       0.61      0.32      0.42       537

    accuracy                           0.77      2113
   macro avg       0.70      0.62      0.64      2113
weighted avg       0.75      0.77      0.75      2113

DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=4) <sklearn.metrics._plot.confusion_matrix.ConfusionM
```

```
******************* GradientBoostingClassifier() *******************
              precision    recall  f1-score   support

           0       0.83      0.89      0.86      1576
           1       0.59      0.47      0.53       537

    accuracy                           0.78      2113
   macro avg       0.71      0.68      0.69      2113
weighted avg       0.77      0.78      0.77      2113

GradientBoostingClassifier() <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7bdb86705090>
```

```
******************* RandomForestClassifier(max_depth=9, max_samples=0.25, min_samples_leaf=3,
                       n_estimators=500, n_jobs=-1, oob_score=True,
                       random_state=25) *******************
              precision    recall  f1-score   support

           0       0.82      0.91      0.86      1576
           1       0.60      0.42      0.49       537

    accuracy                           0.78      2113
   macro avg       0.71      0.66      0.68      2113
weighted avg       0.76      0.78      0.77      2113

RandomForestClassifier(max_depth=9, max_samples=0.25, min_samples_leaf=3,
                       n_estimators=500, n_jobs=-1, oob_score=True,
                       random_state=25) <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7bdb864d5b70
```

```
******************* LogisticRegression(C=0.5, random_state=15, solver='saga') *******************
```

```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86      1576
           1       0.59      0.42      0.49       537

    accuracy                           0.78      2113
   macro avg       0.70      0.66      0.67      2113
weighted avg       0.76      0.78      0.76      2113

LogisticRegression(C=0.5, random_state=15, solver='saga') <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object
```

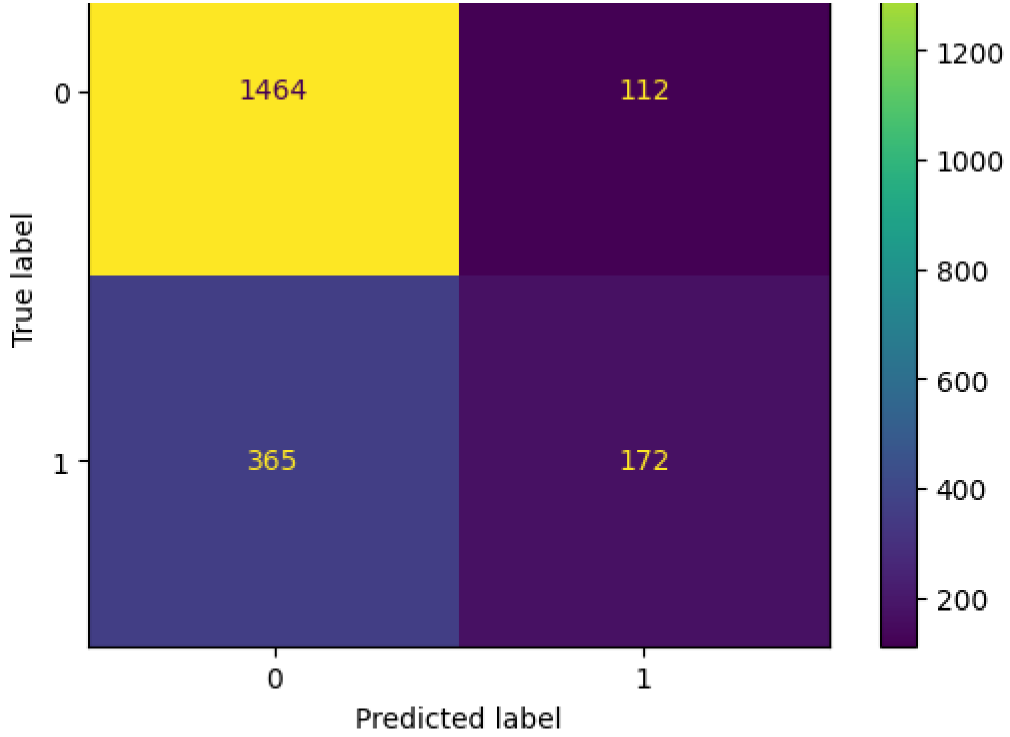
```
******************** SVC(C=0.1, gamma=0.01, kernel='linear', random_state=2) ********************
              precision    recall  f1-score   support

           0       0.82      0.89      0.86      1576
           1       0.58      0.44      0.50       537

    accuracy                           0.78      2113
   macro avg       0.70      0.67      0.68      2113
weighted avg       0.76      0.78      0.77      2113

SVC(C=0.1, gamma=0.01, kernel='linear', random_state=2) <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object a
```

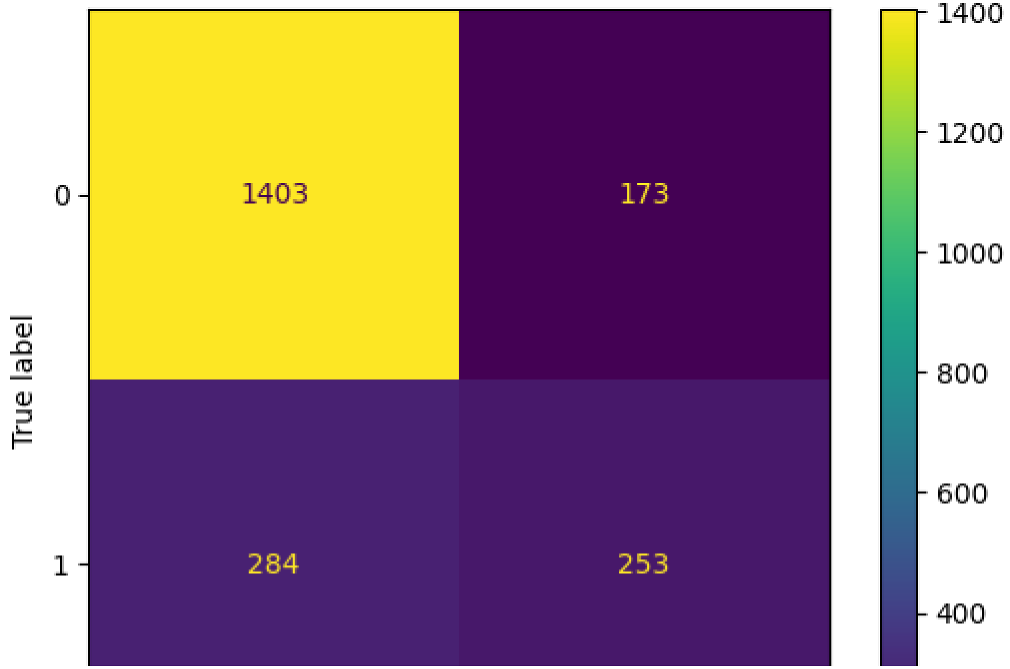
```
******************** XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.01, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=4, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=150, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...) ********************
              precision    recall  f1-score   support

           0       0.81      0.91      0.86      1576
           1       0.60      0.39      0.47       537

    accuracy                           0.78      2113
   macro avg       0.71      0.65      0.67      2113
```

```
weighted avg       0.76      0.78      0.76      2113

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.01, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=4, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=150, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...) <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at
```

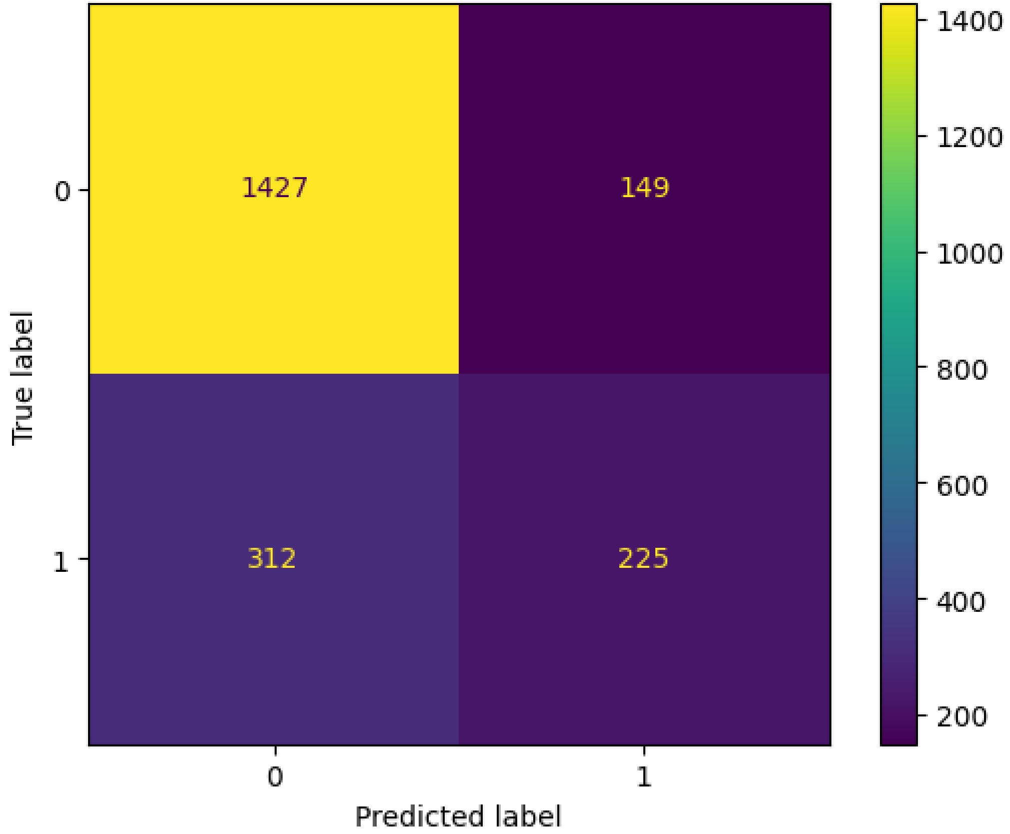
```
******************** LGBMClassifier(n_estimators=50, num_leaves=20) ********************
              precision    recall  f1-score   support

           0       0.82      0.90      0.86      1576
           1       0.60      0.43      0.50       537

    accuracy                           0.78      2113
   macro avg       0.71      0.67      0.68      2113
weighted avg       0.77      0.78      0.77      2113

LGBMClassifier(n_estimators=50, num_leaves=20) <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7bdb8
```

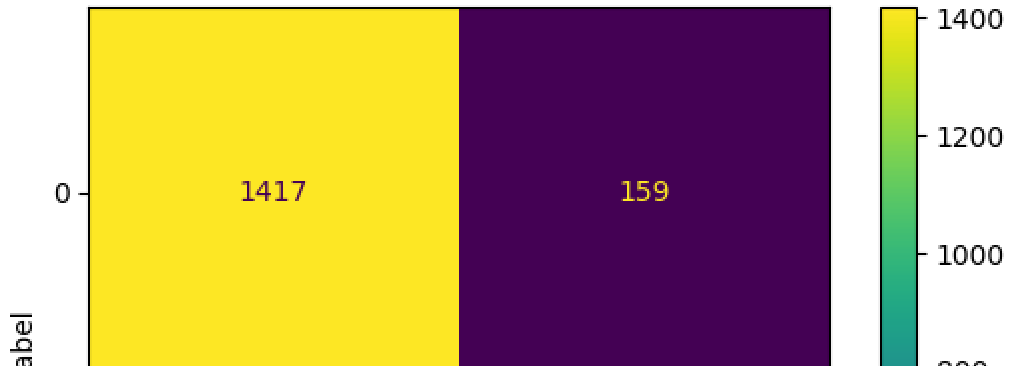
```
******************** MLPClassifier(activation='tanh', solver='sgd') ********************
              precision    recall  f1-score   support

           0       0.82      0.90      0.86      1576
           1       0.58      0.41      0.48       537

    accuracy                           0.78      2113
   macro avg       0.70      0.65      0.67      2113
weighted avg       0.76      0.78      0.76      2113

MLPClassifier(activation='tanh', solver='sgd') <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7bdb8
```

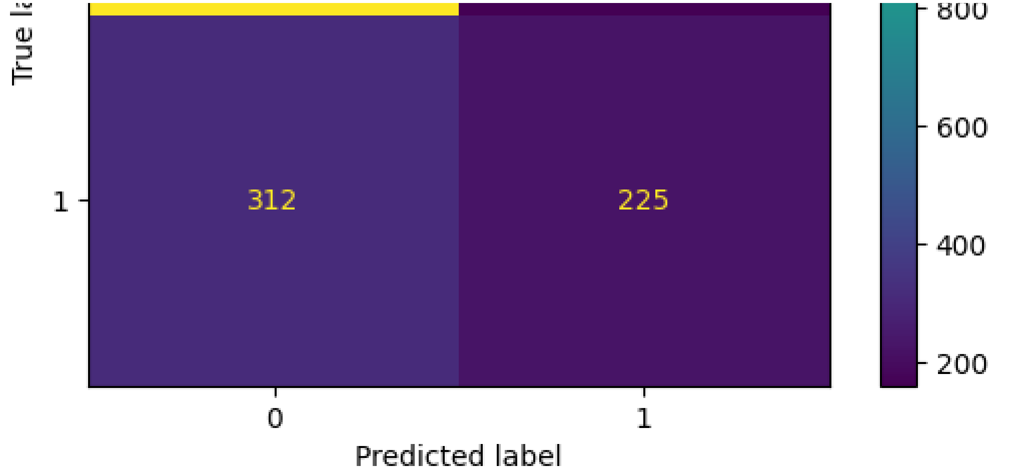

1464
112
365
172
0
1
True label
0
1
Predicted label
1200
1000
800
600
400
200

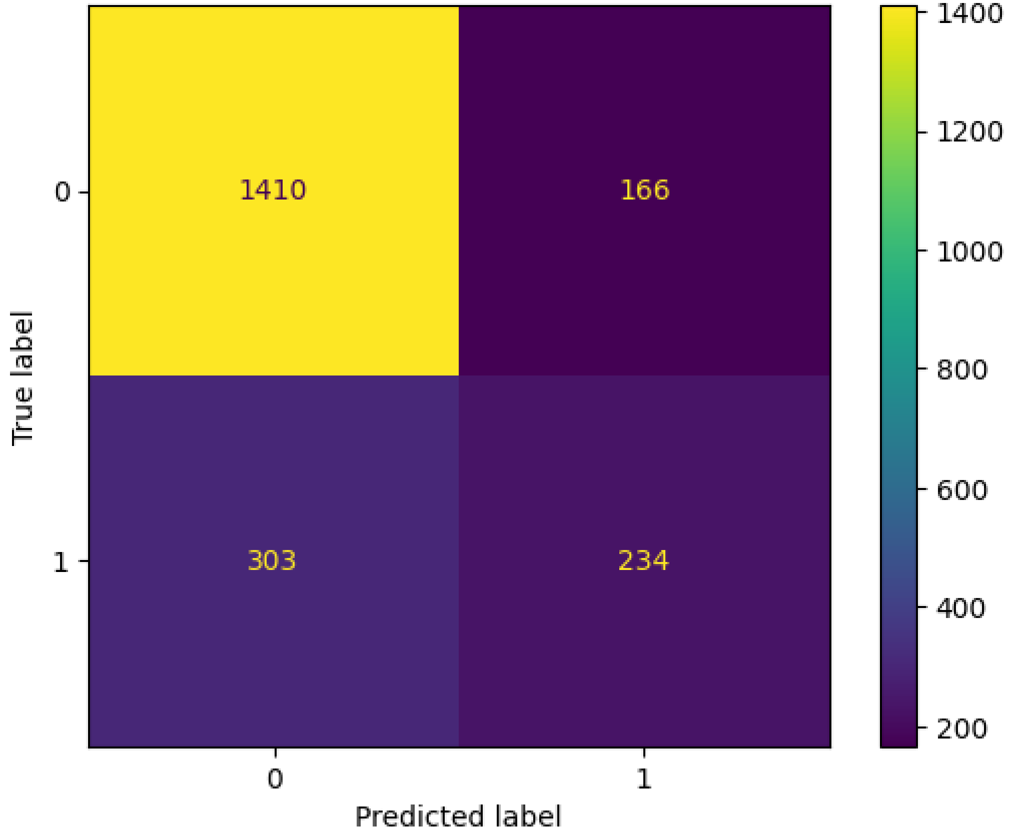

1403
173
284
253
0
1
True label
1400
1200
1000
800
600
400


200
0
1
Predicted label

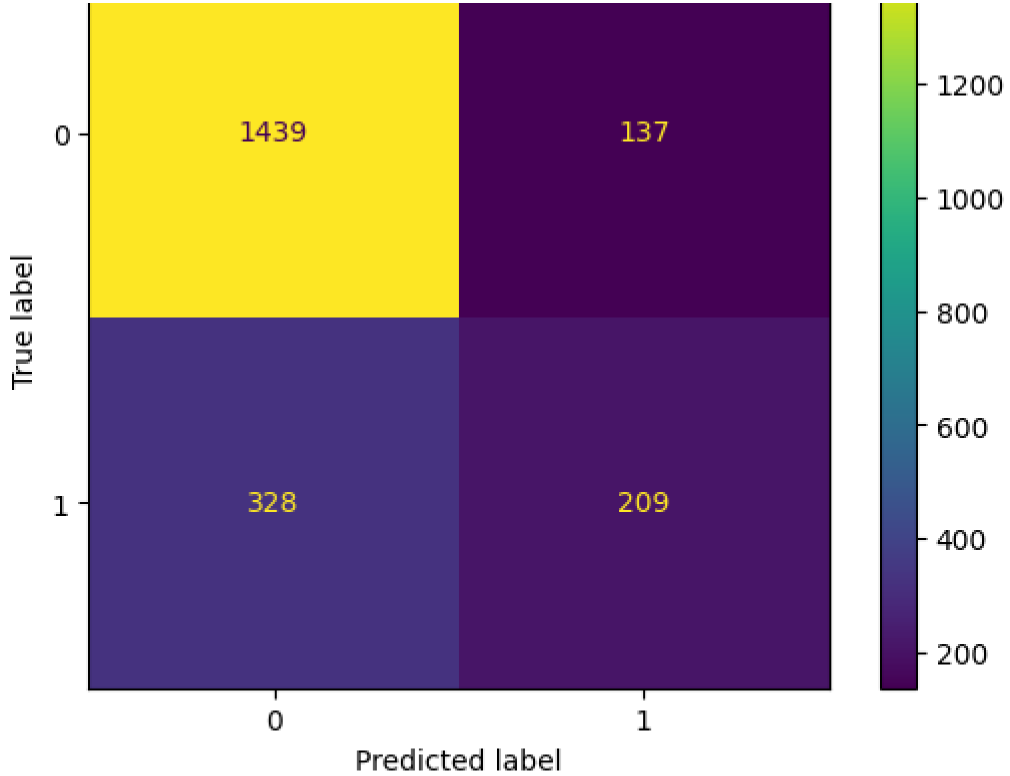

1427
149
312
225
True label
0
1
Predicted label
1400
1200
1000
800
600
400
200

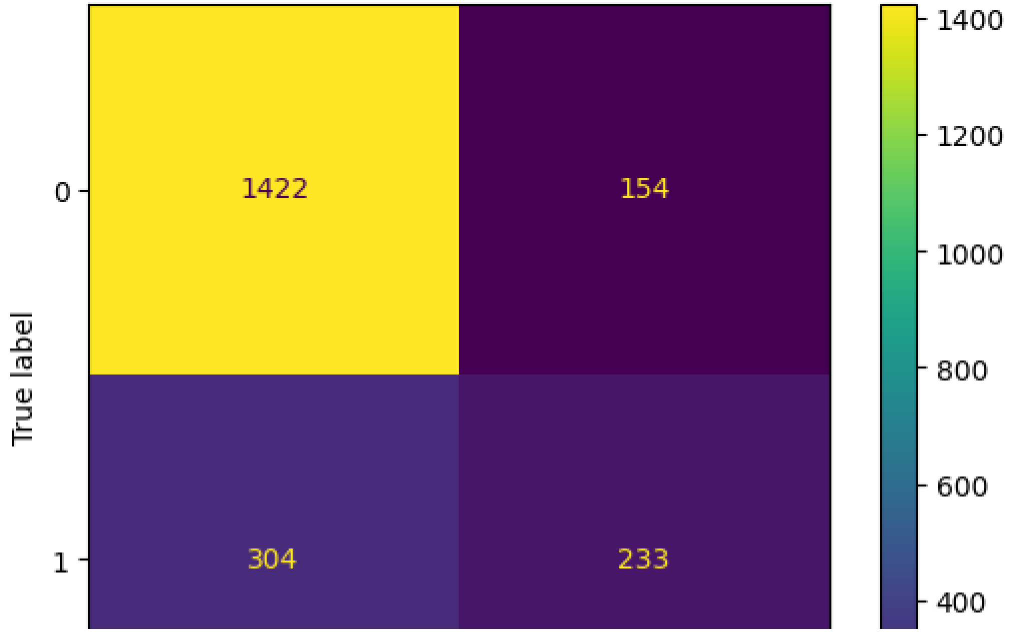

1417
159
0
1400
1200
1000

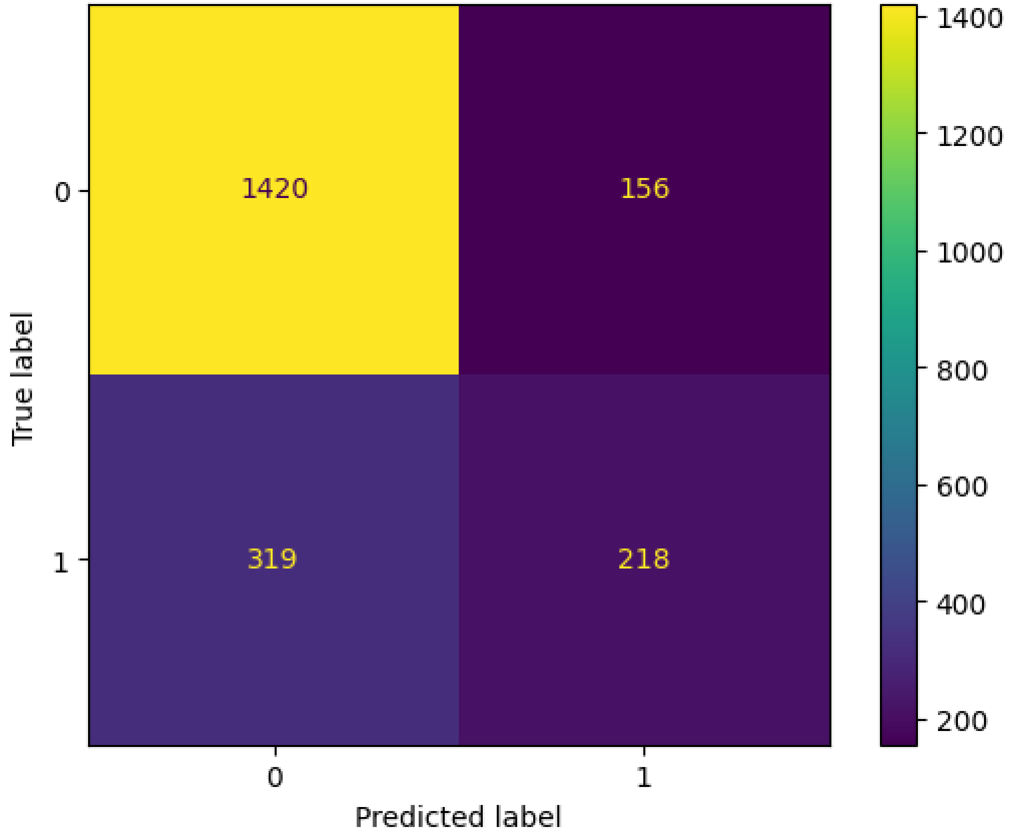

True la
1
312
225
0
1
Predicted label
800
600
400
200


True label
0
1
1410
166
303
234
0
1
Predicted label
1400
1200
1000
800
600
400
200


1400


1439
137
328
209
0
1
True label
0
1
Predicted label
1200
1000
800
600
400
200


1422
154
304
233
0
1
True label
1400
1200
1000
800
600
400



SMOTE SAMPLING

```
sm=SMOTE(random_state=589)
X_train_o, y_train_o=sm.fit_resample(X_train,y_train)
#Finding best random state.
```

```
import random

random_numbers = random.sample(range(801,900),25)

best_random_state=None
best_accuracy=0.0
for value in random_numbers:
  X_train1,X_test1,y_train1,y_test1=train_test_split(X,y,test_size=0.3,random_state=value)
  sm=SMOTE()
  X_train, y_train=sm.fit_resample(X_train,y_train)
  for clf in lst1:
      clf.fit(X_train,y_train)
      accuracy=clf.score(X_test,y_test)
  if accuracy>0.80028395646000:   #best_accuracy   #589
      best_accuracy=accuracy
      best_random_state=value
      print("Best Random State:", best_random_state)
      print("Best Accuracy:", best_accuracy)
```

```
##feature selection
# from sklearn import feature_selection
# import warnings
# from sklearn.model_selection import GridSearchCV
# import warnings
# rf = XGBClassifier()
# grid_vls={'max_depth': [3, 5, 7],
#     'learning_rate': [0.1, 0.01, 0.001],
#     'n_estimators': [100, 200, 300],
#     'subsample': [0.8, 1.0],
#     'colsample_bytree': [0.8, 1.0]
#      }
# grid=GridSearchCV(estimator=rf, param_grid=grid_vls,scoring="accuracy", cv=5,refit=True,return_train_score=True)
```

```
# grid.fit(X_train_o, y_train_o)
# grid.best_params_
```

```


rf=RandomForestClassifier(max_depth=None,min_samples_leaf=1,min_samples_split=2,n_estimators=300)
ad=AdaBoostClassifier()

lg=lgb.LGBMClassifier(learning_rate= 0.1,n_estimators=50,num_leaves=20)
lr=LogisticRegression()
gb=GradientBoostingClassifier(learning_rate=0.05,max_depth= 5,n_estimators=150)
xgb=XGBClassifier(colsample_bytree=0.8,learning_rate= 0.1,max_depth=7,n_estimators=100,subsample=0.8)
svc=SVC()
dt=DecisionTreeClassifier(criterion="entropy",max_depth= 5,min_samples_leaf=4,min_samples_split=2)
nn = MLPClassifier(activation='tanh',hidden_layer_sizes= (100,),learning_rate="constant",solver="sgd")
lst1=[ad,dt,gb,rf,lr,svc,xgb,lg,nn]
ac_o=[]
model_o=[]
for j in lst1:
  print('*'*20,j,'*'*20)
  j.fit(X_train_o,y_train_o)
  y_pred=j.predict(X_test)

  print(classification_report(y_test,y_pred))
  print(j,ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
  print("_"*200)
  correct_predictions = (y_pred == y_test).sum()
  accuracy = correct_predictions / len(y_test)

  ac_o.append(accuracy)
  # ac_o.append(j.accuracy_score(y_test,y_pred))
  # model_o.append(j)
```

```
****************** AdaBoostClassifier() ********************
              precision    recall  f1-score   support

           0       0.91      0.79      0.84      1576
           1       0.55      0.76      0.64       537

    accuracy                           0.78      2113
   macro avg       0.73      0.78      0.74      2113
eighted avg       0.82      0.78      0.79      2113

daBoostClassifier() <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7bdb80a92e90>
```

```
****************** DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=4) ********************
              precision    recall  f1-score   support

           0       0.90      0.76      0.82      1576
           1       0.51      0.76      0.61       537

    accuracy                           0.76      2113
   macro avg       0.71      0.76      0.72      2113
eighted avg       0.80      0.76      0.77      2113

ecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=4) <sklearn.metrics._plot.confusion_matrix.ConfusionMat
```

```
****************** GradientBoostingClassifier(learning_rate=0.05, max_depth=5, n_estimators=150) ********************
              precision    recall  f1-score   support

           0       0.88      0.83      0.86      1576
           1       0.58      0.67      0.62       537

    accuracy                           0.79      2113
   macro avg       0.73      0.75      0.74      2113
eighted avg       0.80      0.79      0.80      2113

radientBoostingClassifier(learning_rate=0.05, max_depth=5, n_estimators=150) <sklearn.metrics._plot.confusion_matrix.ConfusionMa
```

```
****************** RandomForestClassifier(n_estimators=300) ********************
              precision    recall  f1-score   support

           0       0.87      0.86      0.86      1576
           1       0.60      0.62      0.61       537
```

```
    accuracy                           0.80      2113
   macro avg       0.73      0.74      0.74      2113
eighted avg       0.80      0.80      0.80      2113

andomForestClassifier(n_estimators=300) <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7bdb866b04f0>
```

```
******************* LogisticRegression() ********************
              precision    recall  f1-score   support

           0       0.91      0.75      0.82      1576
           1       0.52      0.79      0.62       537

    accuracy                           0.76      2113
   macro avg       0.71      0.77      0.72      2113
eighted avg       0.81      0.76      0.77      2113

ogisticRegression() <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7bdb86288250>
```

```
******************* SVC() ********************
              precision    recall  f1-score   support

           0       0.89      0.78      0.83      1576
           1       0.53      0.73      0.61       537

    accuracy                           0.76      2113
   macro avg       0.71      0.75      0.72      2113
eighted avg       0.80      0.76      0.78      2113

VC() <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7bdb866489d0>
```

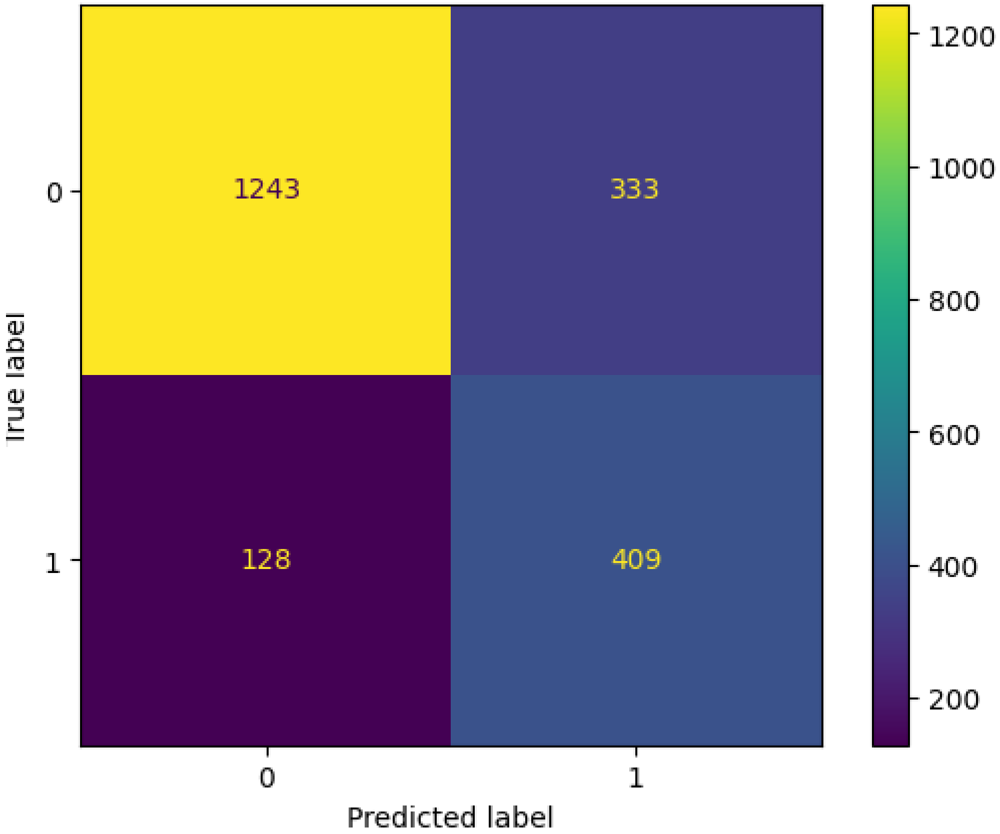
```
******************* XGBClassifier(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.8, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=7, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
```

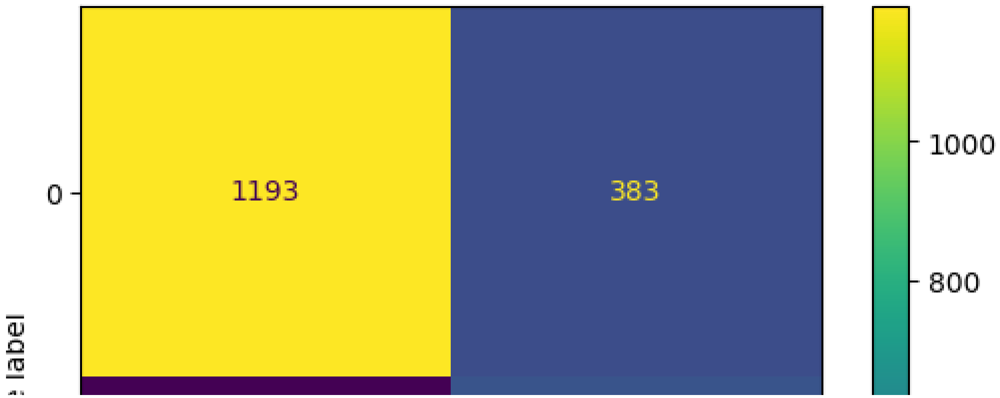
```
              predictor=None, random_state=None, ...) ********************
              precision    recall  f1-score   support

           0       0.88      0.84      0.86      1576
           1       0.59      0.65      0.62       537

    accuracy                           0.80      2113
   macro avg       0.73      0.75      0.74      2113
eighted avg       0.80      0.80      0.80      2113

GBClassifier(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.8, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=7, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...) <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x
```

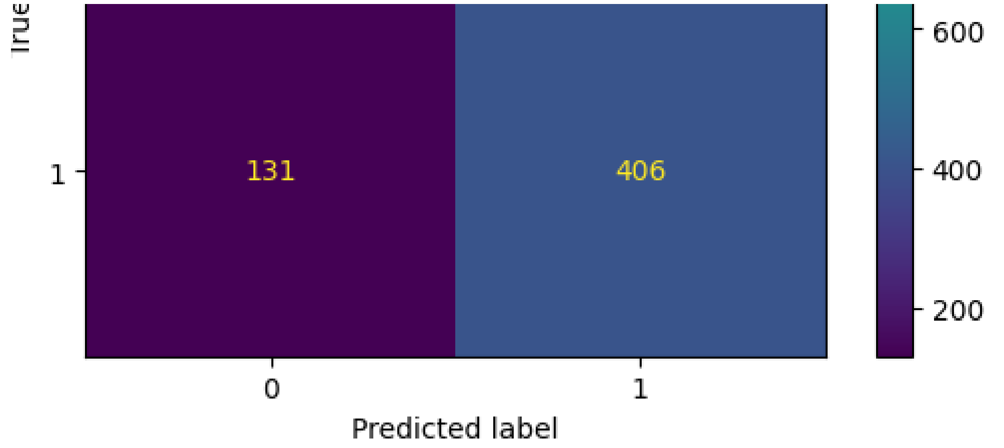
```
******************* LGBMClassifier(n_estimators=50, num_leaves=20) ********************
              precision    recall  f1-score   support

           0       0.88      0.84      0.86      1576
           1       0.58      0.67      0.62       537

    accuracy                           0.79      2113
   macro avg       0.73      0.75      0.74      2113
eighted avg       0.80      0.79      0.80      2113

GBMClassifier(n_estimators=50, num_leaves=20) <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7bdb866
```

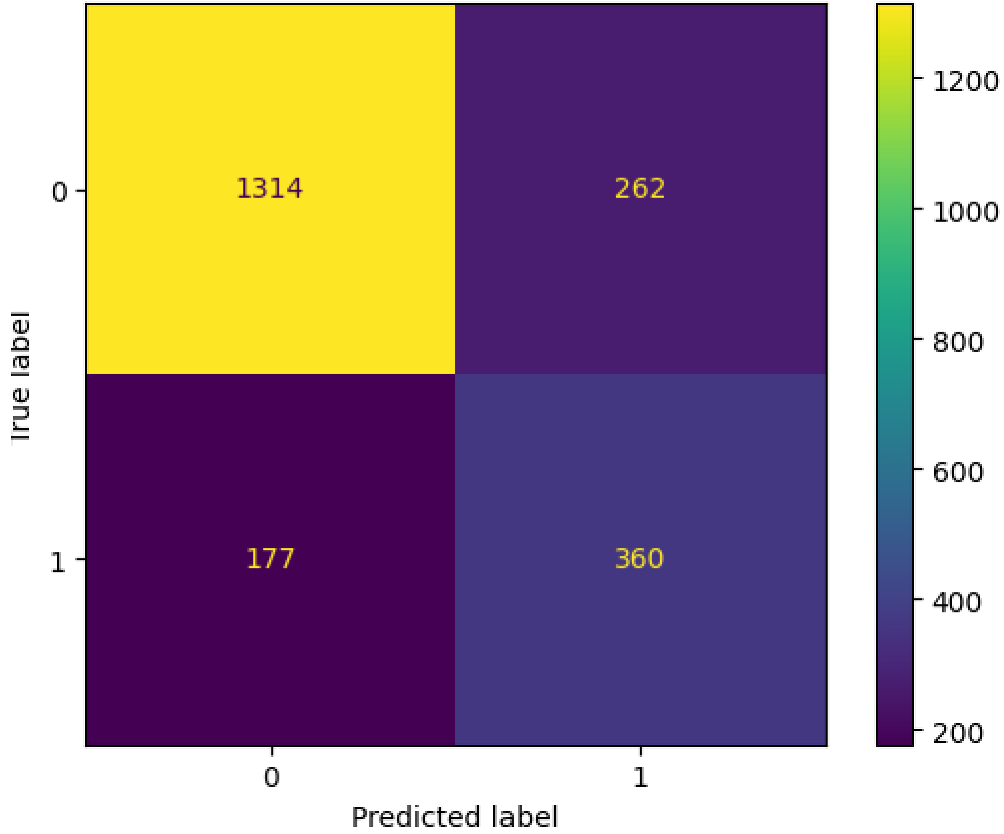
```
******************* MLPClassifier(activation='tanh', solver='sgd') ********************
              precision    recall  f1-score   support

           0       0.91      0.74      0.82      1576
           1       0.51      0.80      0.62       537

    accuracy                           0.76      2113
```

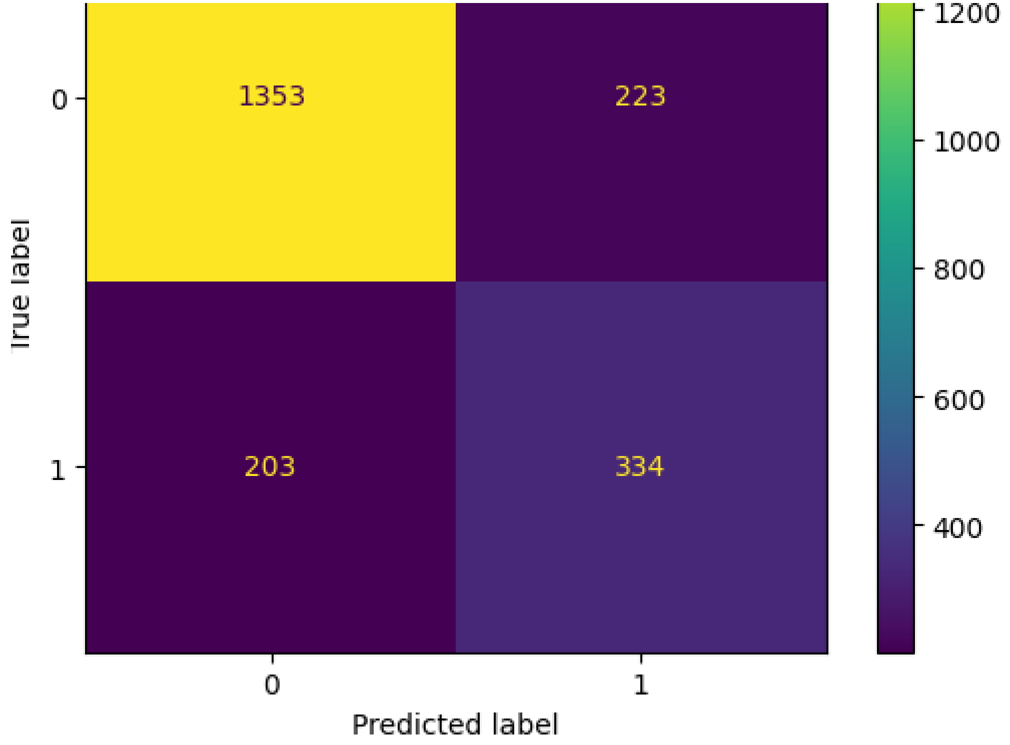
```
 macro avg       0.71      0.77      0.72      2113
eighted avg       0.81      0.76      0.77      2113

.PClassifier(activation='tanh', solver='sgd') <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7bdb864
```

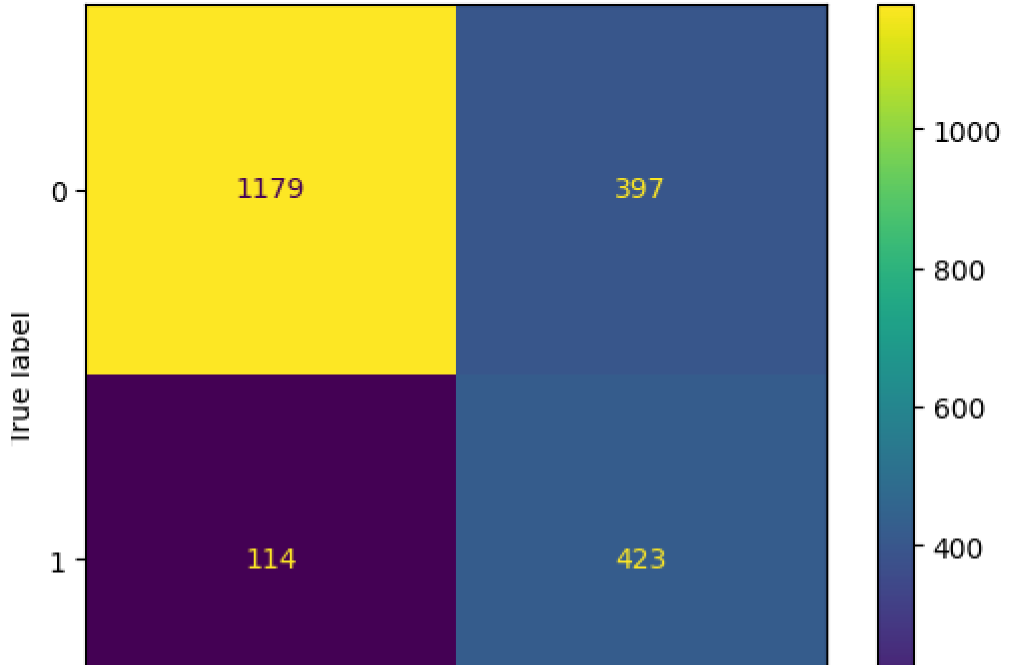




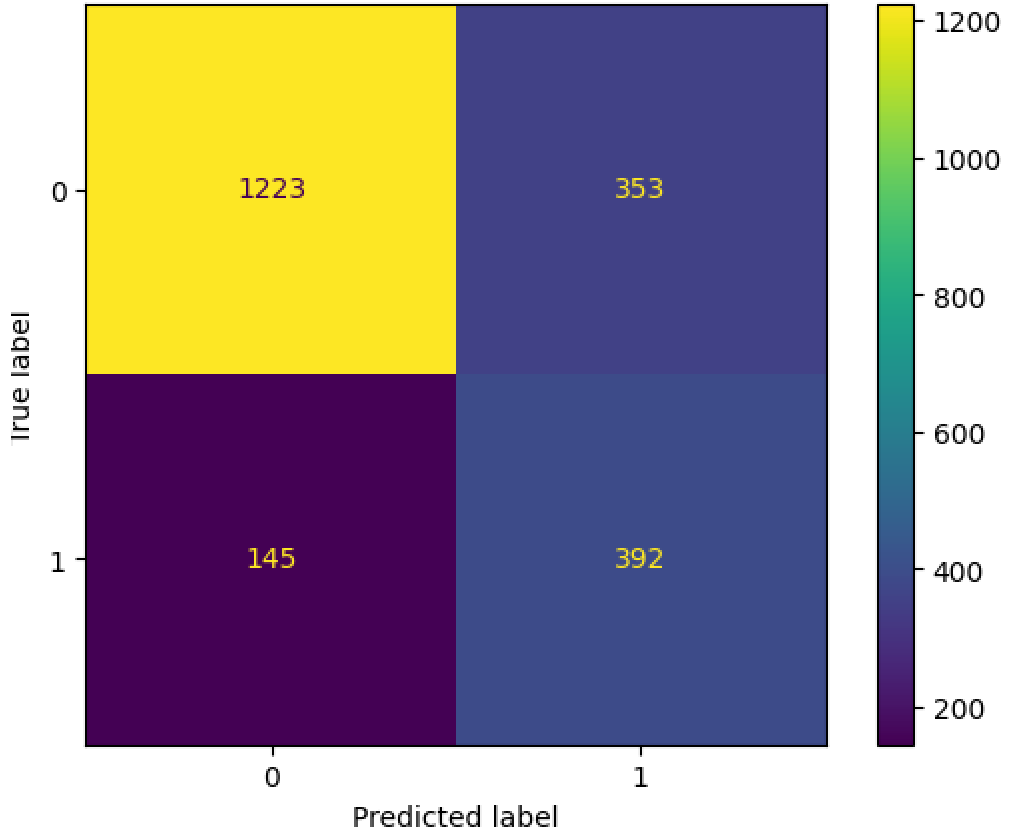

True
1
131
406
0
1
Predicted label
600
400
200

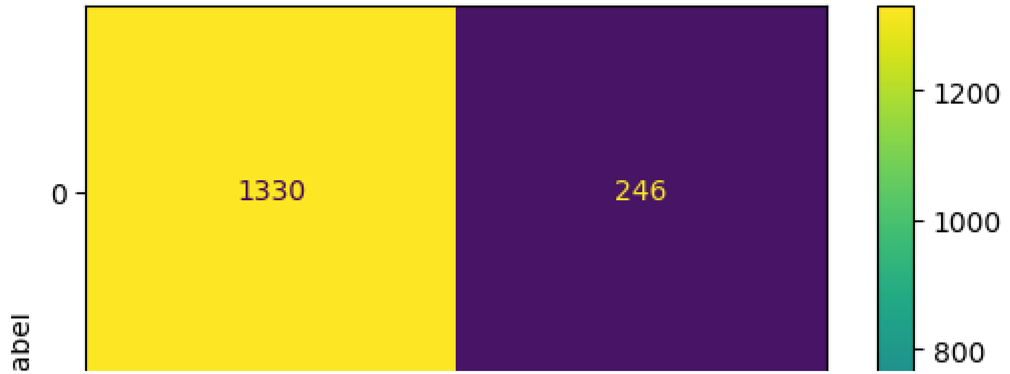

0
1
1314
262
177
360
True label
0
1
Predicted label
1200
1000
800
600
400
200

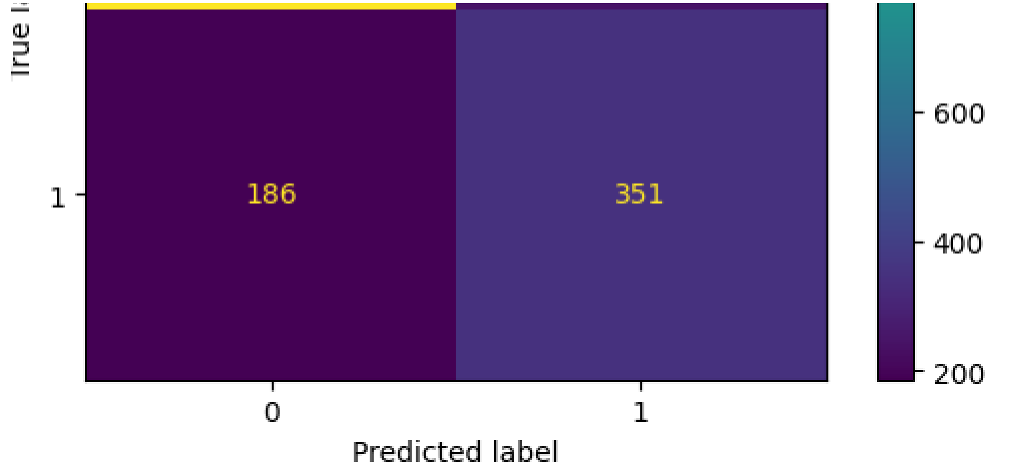

1353
223
203
334
0
1
True label
0
1
Predicted label
1200
1000
800
600
400

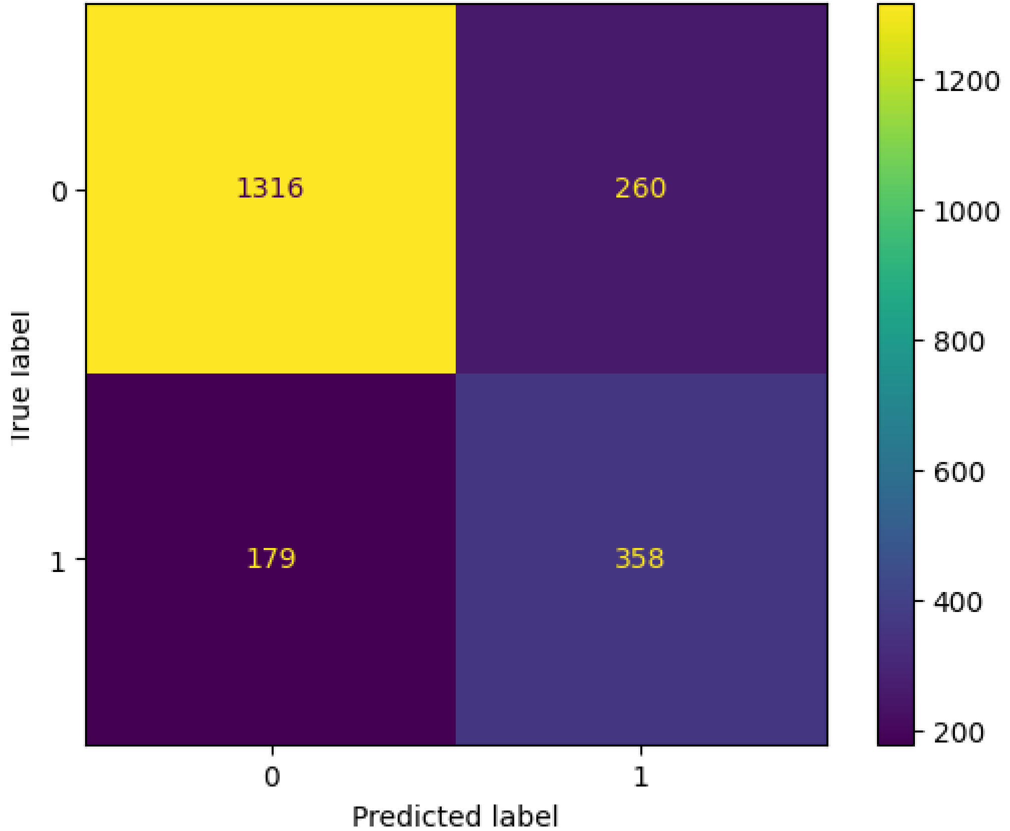

1179
397
114
423
0
1
True label
1000
800
600
400

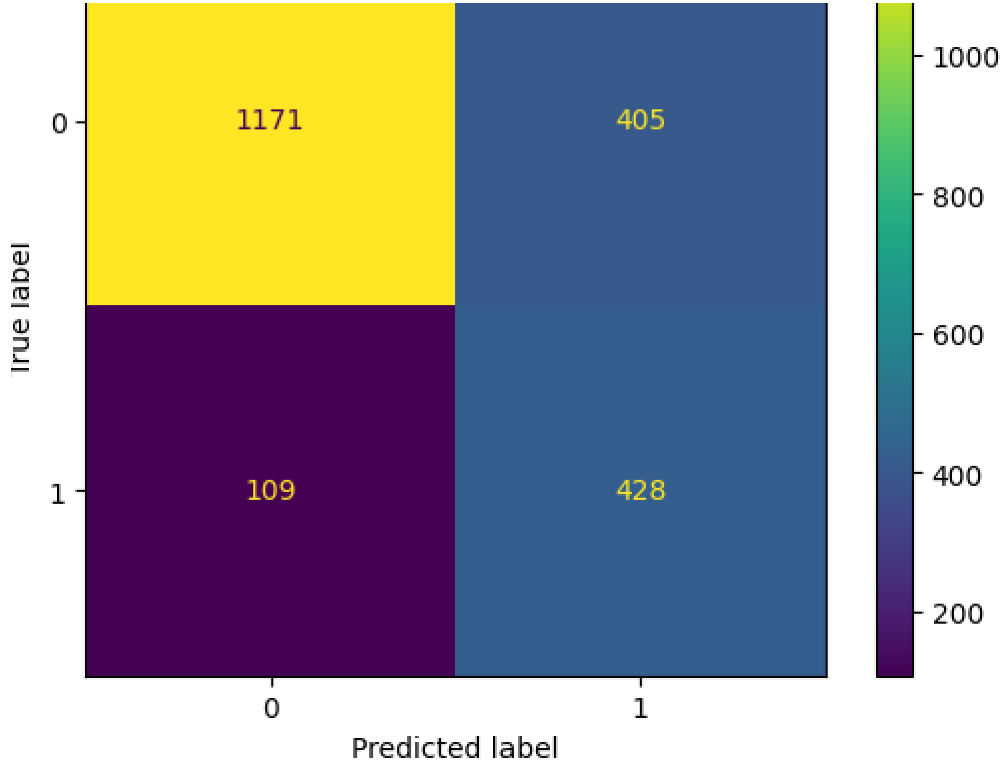

200
0
1
Predicted label


True label
0
1
1223
353
145
392
0
1
Predicted label
1200
1000
800
600
400
200


abel
0
1330
246
1200
1000
800


1
186
351
600
400
200
0
1
Predicted label


True label
0
1
1316
260
179
358
1200
1000
800
600
400
200
0
1
Predicted label




```
#RandomOverSampler
oversampler = RandomOverSampler()
X_train_resampled, y_train_resampled = oversampler.fit_resample(X, y)


rf=RandomForestClassifier(max_depth=None,min_samples_leaf=1,min_samples_split=2,n_estimators=300)
ad=AdaBoostClassifier()

lg=lgb.LGBMClassifier(learning_rate= 0.1,n_estimators=50,num_leaves=20)
```

```
lr=LogisticRegression()
gb=GradientBoostingClassifier(learning_rate=0.05,max_depth= 5,n_estimators=150)
xgb=XGBClassifier(colsample_bytree=0.8,learning_rate= 0.1,max_depth=7,n_estimators=100,subsample=0.8)
svc=SVC()
dt=DecisionTreeClassifier(criterion="entropy",max_depth= 5,min_samples_leaf=4,min_samples_split=2)
nn = MLPClassifier(activation='tanh',hidden_layer_sizes= (100,),learning_rate="constant",solver="sgd")
lst1=[ad,dt,gb,rf,lr,svc,xgb,lg,nn]
ac_ro=[]
model_ro=[]
for j in lst1:
  print('*'*20,j,'*'*20)
  j.fit(X_train_resampled,y_train_resampled)
  y_pred=j.predict(X_test)

  print(classification_report(y_test,y_pred))
  #print(j,ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
  print("_"*200)
  correct_predictions = (y_pred == y_test).sum()
  accuracy = correct_predictions / len(y_test)

  ac_ro.append(accuracy)
  model_ro.append(j)
```

```
weighted avg       0.78      0.50      0.20      2113
```

over_sampling with PCA - it's accuracy is very low

UNDER_SAMPILNG - gives comparatively less accuracy than over sampling.

```
from imblearn.under_sampling import RandomUnderSampler
ra=RandomUnderSampler(random_state=1)
X_train_u, y_train_u=ra.fit_resample(X_train,y_train)
```

```


rf=RandomForestClassifier(n_estimators=500,n_jobs=-1,max_depth=9,oob_score=True,random_state=25,max_samples=0.25,min
ad=AdaBoostClassifier()

lg=lgb.LGBMClassifier()
lr=LogisticRegression()
gb=GradientBoostingClassifier()
xgb=XGBClassifier(learning_rate=0.01,max_depth=4,n_estimators=150)
svc=SVC(C=0.1,gamma=0.01,kernel='linear',random_state=2)
dt=DecisionTreeClassifier(criterion="entropy",max_depth= 5,min_samples_leaf=4,min_samples_split=2)
nn = MLPClassifier(activation='tanh',hidden_layer_sizes= (100,),learning_rate="constant",solver="sgd")
lst1=[ad,dt,gb,rf,lr,svc,xgb,lg,nn]
acus=[]
modelus=[]
for k in lst1:
  print('*'*20,k,'*'*20)
  k.fit(X_train_u,y_train_u)
  y_pred=k.predict(X_test)

  print(classification_report(y_test,y_pred))
```

```
print("_"*200)
correct_predictions = (y_pred == y_test).sum()
accuracy = correct_predictions / len(y_test)

acus.append(accuracy)

modelus.append(k)
```

```
    accuracy                           0.76      2113
   macro avg       0.71      0.76      0.72      2113
weighted avg       0.81      0.76      0.77      2113


******************** LGBMClassifier() ********************
              precision    recall  f1-score   support

           0       0.87      0.85      0.86      1576
           1       0.59      0.64      0.61       537

    accuracy                           0.80      2113
   macro avg       0.73      0.74      0.74      2113
weighted avg       0.80      0.80      0.80      2113


******************** MLPClassifier(activation='tanh', solver='sgd') ********************
              precision    recall  f1-score   support

           0       0.92      0.74      0.82      1576
           1       0.52      0.81      0.63       537

    accuracy                           0.76      2113
   macro avg       0.72      0.77      0.72      2113
weighted avg       0.82      0.76      0.77      2113
```

pca under sampling

```
#pca under sampling
pca=PCA(n_components=3,random_state=3)
X_trainm=pca.fit_transform(X_train)
X_testm=pca.transform(X_test)
pca.explained_variance_ratio_
```

```
array([0.21219839, 0.13569808, 0.08801288])
```

```
rf=RandomForestClassifier(n_estimators=500,n_jobs=-1,max_depth=9,oob_score=True,random_state=25,max_samples=0.25,min
ad=AdaBoostClassifier()

lg=lgb.LGBMClassifier(learning_rate= 0.1,n_estimators=50,num_leaves=20)
lr=LogisticRegression()
#gb=GradientBoostingClassifier()
xgb=XGBClassifier(learning_rate=0.01,max_depth=4,n_estimators=150)
svc=SVC(C=0.1,gamma=0.01,kernel='linear',random_state=2)
dt=DecisionTreeClassifier(criterion="entropy",max_depth= 5,min_samples_leaf=4,min_samples_split=2)
nn = MLPClassifier(activation='tanh',hidden_layer_sizes= (100,),learning_rate="constant",solver="sgd")
lst1=[ad,dt,gb,rf,lr,svc,xgb,lg,nn]
acup=[]
modelup=[]
for up in lst1:
  print('*'*20,up,'*'*20)
  up.fit(X_trainm,y_train)
  y_pred=up.predict(X_testm)

  print(classification_report(y_test,y_pred))
  print("_"*200)
  correct_predictions = (y_pred == y_test).sum()
  accuracy = correct_predictions / len(y_test)

  acup.append(accuracy)

  modelup.append(up)
```

```
MLPClassifier(activation='tanh', solver='sgd')
              precision    recall  f1-score   support

           0       0.89      0.71      0.79      1576
           1       0.47      0.75      0.58       537

    accuracy                           0.72      2113
   macro avg       0.68      0.73      0.68      2113
weighted avg       0.79      0.72      0.74      2113
```

```
print("*"*20,"stacking method","*"*20)
lst2=[("knn",KNeighborsClassifier()),("nb",GaussianNB()),("ad",AdaBoostClassifier()),("dt",DecisionTreeClassifier())
stc=StackingClassifier(estimators=lst2,final_estimator=RandomForestClassifier())
stc.fit(X_train,y_train)
y_pred_stc=stc.predict(X_test)
print(classification_report(y_test,y_pred_stc))
ac_s=[]
model_s=[]
correct_predictions = (y_pred_stc == y_test).sum()
accuracy_stc = correct_predictions / len(y_test)
print("_"*200)

ac_s.append(accuracy_stc)
model_s.append(stc)
print(ac_s)
```

```
******************** stacking method ********************
              precision    recall  f1-score   support

           0       0.86      0.87      0.86      1576
           1       0.60      0.57      0.58       537

    accuracy                           0.79      2113
   macro avg       0.73      0.72      0.72      2113
weighted avg       0.79      0.79      0.79      2113
```

```
[0.7941315664931378]
```

```
from imblearn.combine import SMOTEENN
ou=SMOTEENN(random_state=1)
X_train_ou,y_train_ou=ou.fit_resample (X_train,y_train)
rf=RandomForestClassifier(n_estimators=500,n_jobs=-1,max_depth=9,oob_score=True,random_state=25,max_samples=0.25,min
ad=AdaBoostClassifier()

lg=lgb.LGBMClassifier(learning_rate= 0.1,n_estimators=50,num_leaves=20)
lr=LogisticRegression()
gb=GradientBoostingClassifier()
xgb=XGBClassifier(learning_rate=0.01,max_depth=4,n_estimators=150)
svc=SVC(C=0.1,gamma=0.01,kernel='linear',random_state=2)
dt=DecisionTreeClassifier(criterion="entropy",max_depth= 5,min_samples_leaf=4,min_samples_split=2)
nn = MLPClassifier(activation='tanh',hidden_layer_sizes= (100,),learning_rate="constant",solver="sgd")
lst1=[ad,dt,gb,rf,lr,svc,xgb,lg,nn]
acou=[]
modelou=[]
for up in lst1:
  print('*'*20,up,'*'*20)
  up.fit(X_train_ou,y_train_ou)
  y_pred=up.predict(X_test)

  print(classification_report(y_test,y_pred))
  print("_"*200)
  correct_predictions = (y_pred == y_test).sum()
  accuracy = correct_predictions / len(y_test)

  acou.append(accuracy)

  modelou.append(ou)
```

```
MLPClassifier(activation='tanh', solver='sgd')
              precision    recall  f1-score   support

           0       0.93      0.66      0.77      1576
           1       0.46      0.85      0.60       537

    accuracy                           0.71      2113
   macro avg       0.69      0.76      0.68      2113
weighted avg       0.81      0.71      0.73      2113
```

```
print("Highest accuracy in each method :-")
highAC_in_up=0.0
for m in acup:
    if m>highAC_in_up:
      highAC_in_up=m

highAC_in_ou=0.0
for m in acou:
    if m>highAC_in_ou:
      highAC_in_ou=m

highAC_in_normal=0.0
for m in ac:
    if m>highAC_in_normal:
      highAC_in_normal=m


highAC_in_us=0.0
for m in acus:
    if m>highAC_in_us:
      highAC_in_us=m
```

```
highAC_in_ro=0.0
for m in ac_ro:
    if m>highAC_in_ro:
      highAC_in_ro=m



highAC_in_o=0.0
for m in ac_o:
    if m>highAC_in_o:
      highAC_in_o=m



highAC_in_acp=0.0
for m in acp:
    if m>highAC_in_acp:
      highAC_in_acp=m



accuracy_stc
highAC_lst=pd.DataFrame({"undersampling_pca":highAC_in_up,"SMOTEENN":highAC_in_ou,"RandomOverSampling":highAC_in_ro,
print(highAC_lst)
```

```
Highest accuracy in each method :-
   undersampling_pca  SMOTEENN  RandomOverSampling    normal  undersampling  \
0           0.729295  0.752958            0.769049  0.816848       0.795078

      smote      pca  stacking--
0  0.798391  0.78372    0.794132
```

A consolidated table was created to compare the accuracy of different algorithms and preprocessing techniques. The table was styled to highlight the highest accuracy value using red color and a gradient color map.

```
accuracy_SMOTEENN = pd.Series(acou, name='accuracy')
algorithm_SMOTEENN = pd.Series(modelou, name='algorithm')
normal = pd.concat([accuracy_SMOTEENN, algorithm_SMOTEENN], axis=1)

accuracy_normal = pd.Series(ac, name='accuracy')
algorithm_normal = pd.Series(model, name='algorithm')
normal = pd.concat([accuracy_normal, algorithm_normal], axis=1)

accuracy_RandomOverSampler = pd.Series(ac_ro, name='accuracy')
algorithm_RandomOverSampler = pd.Series(model_ro, name='algorithm')
RandomOverSampler = pd.concat([accuracy_RandomOverSampler, algorithm_RandomOverSampler], axis=1)

accuracy_pca = pd.Series(acp, name='accuracy')
algorithm_pca = pd.Series(modelp, name='algorithm')
pca = pd.concat([accuracy_pca, algorithm_pca], axis=1)


accuracy_under_sampling_pca = pd.Series(acup, name='accuracy')
algorithm_under_sampling_pca = pd.Series(modelup, name='algorithm')
under_sampling_pca = pd.concat([accuracy_under_sampling_pca,algorithm_under_sampling_pca], axis=1)

accuracy_over_smote = pd.Series(ac_o, name='accuracy')
algorithm_smote = pd.Series(model_o, name='algorithm')
smote = pd.concat([accuracy_over_smote, algorithm_smote], axis=1)

accuracy_under_sampling = pd.Series(acus, name='accuracy')
algorithm_under_sampling = pd.Series(modelus, name='algorithm')
under_sampling = pd.concat([accuracy_under_sampling, algorithm_under_sampling], axis=1)


aggregate_df=pd.concat([normal,pca,RandomOverSampler,under_sampling_pca,smote,under_sampling])
aggregate_df = aggregate_df.reset_index(drop=True)   #becouse of difference in index number order highlight_max() di
```

```
aggregate_df_styled = aggregate_df.style.highlight_max(subset=['accuracy'], color="red")
aggregate_df_styled = aggregate_df_styled.background_gradient(subset=['accuracy'], cmap='RdYlGn', vmax=1.0, vmin=0.0



print(" accuracy obtained through stacking",accuracy_stc)

aggregate_df_styled
```

```
accuracy obtained through stacking 0.7941315664931378
```

	accuracy	algorithm
0	0.814009	AdaBoostClassifier()
1	0.789399	DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=4)
2	0.816848	GradientBoostingClassifier()
3	0.802177	RandomForestClassifier()
4	0.809749	LogisticRegression()
5	0.807856	SVC(C=0.1, gamma=0.01, kernel='linear', random_state=2)
6	0.797444	XGBClassifier(base_score=None, booster=None, callbacks=None, colsample_bylevel=None, colsample_bynode=None, colsample_bytree=None, early_stopping_rounds=None, enable_categorical=False, eval_metric=None, feature_types=None, gamma=None, gpu_id=None, grow_policy=None, importance_type=None, interaction_constraints=None, learning_rate=None, max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None, max_delta_step=None, max_depth=None, max_leaves=None, min_child_weight=None, missing=nan, monotone_constraints=None, n_estimators=100, n_jobs=None, num_parallel_tree=None, predictor=None, random_state=None, ...)
7	0.812589	LGBMClassifier(n_estimators=50, num_leaves=20)
8	0.804543	MLPClassifier(activation='tanh', solver='sgd')
9	0.774255	DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=4)
10	0.783720	GradientBoostingClassifier()
11	0.781827	RandomForestClassifier(max_depth=9, max_samples=0.25, min_samples_leaf=3, n_estimators=500, n_jobs=-1, oob_score=True, random_state=25)
12	0.777094	LogisticRegression(C=0.5, random_state=15, solver='saga')
13	0.778041	SVC(C=0.1, gamma=0.01, kernel='linear', random_state=2)
14	0.779934	XGBClassifier(base_score=None, booster=None, callbacks=None, colsample_bylevel=None, colsample_bynode=None, colsample_bytree=None, early_stopping_rounds=None, enable_categorical=False, eval_metric=None, feature_types=None, gamma=None, gpu_id=None, grow_policy=None, importance_type=None, interaction_constraints=None, learning_rate=0.01, max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None, max_delta_step=None, max_depth=4, max_leaves=None, min_child_weight=None, missing=nan, monotone_constraints=None, n_estimators=150, n_jobs=None, num_parallel_tree=None, predictor=None, random_state=None, ...)

15	0.783247	LGBMClassifier(n_estimators=50, num_leaves=20)
16	0.775201	MLPClassifier(activation='tanh', solver='sgd')
17	0.745859	AdaBoostClassifier()
18	0.478467	DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=4)
19	0.678183	GradientBoostingClassifier(learning_rate=0.05, max_depth=5, n_estimators=150)
20	0.604827	RandomForestClassifier(n_estimators=300)
21	0.769049	LogisticRegression()
22	0.745859	SVC()
23	0.761003	XGBClassifier(base_score=None, booster=None, callbacks=None, colsample_bylevel=None, colsample_bynode=None, colsample_bytree=0.8, early_stopping_rounds=None, enable_categorical=False, eval_metric=None, feature_types=None, gamma=None, gpu_id=None, grow_policy=None, importance_type=None, interaction_constraints=None, learning_rate=0.1, max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None, max_delta_step=None, max_depth=7, max_leaves=None, min_child_weight=None, missing=nan, monotone_constraints=None, n_estimators=100, n_jobs=None, num_parallel_tree=None, predictor=None, random_state=None, ...)
24	0.718410	LGBMClassifier(n_estimators=50, num_leaves=20)
25	0.301467	MLPClassifier(activation='tanh', solver='sgd')
26	0.726455	AdaBoostClassifier()
27	0.703739	DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=4)
28	0.722196	GradientBoostingClassifier()
29	0.729295	RandomForestClassifier(max_depth=9, max_samples=0.25, min_samples_leaf=3, n_estimators=500, n_jobs=-1, oob_score=True, random_state=25)
30	0.723142	LogisticRegression()
31	0.718410	SVC(C=0.1, gamma=0.01, kernel='linear', random_state=2)
32	0.722669	XGBClassifier(base_score=None, booster=None, callbacks=None, colsample_bylevel=None, colsample_bynode=None, colsample_bytree=None, early_stopping_rounds=None, enable_categorical=False, eval_metric=None, feature_types=None, gamma=None, gpu_id=None, grow_policy=None, importance_type=None, interaction_constraints=None, learning_rate=0.01, max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None, max_delta_step=None, max_depth=4, max_leaves=None,

		min_child_weight=None, missing=nan, monotone_constraints=None, n_estimators=150, n_jobs=None, num_parallel_tree=None, predictor=None, random_state=None, ...)
33	0.726455	LGBMClassifier(n_estimators=50, num_leaves=20)
34	0.720303	MLPClassifier(activation='tanh', solver='sgd')
35	0.781827	nan
36	0.756744	nan
37	0.792239	nan
38	0.798391	nan
39	0.758164	nan
40	0.764316	nan
41	0.795551	nan