```
from google.colab import drive
import zipfile
import os

drive.mount('/content/drive')

zip_file_path = '/content/drive/MyDrive/HEART.zip'

extraction_path = '/content/extracted_files'

if not os.path.exists(extraction_path):
    os.makedirs(extraction_path)

try:
    with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
        zip_ref.extractall(extraction_path)
    print(f"Successfully extracted '{zip_file_path}' to '{extraction_path}'")
except FileNotFoundError:
    print(f"Error: Zip file not found at '{zip_file_path}'")
except zipfile.BadZipFile:
    print(f"Error: Invalid zip file at '{zip_file_path}'")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
Mounted at /content/drive
Successfully extracted '/content/drive/MyDrive/HEART.zip' to '/content/extracted_files'
```

```
import pandas as pd
import numpy as np
import os
from natsort import natsorted
import joblib


NORMAL_=[]
MI_=[]
PMI_=[]
HB_=[]

normal = '/content/drive/MyDrive/NORMAL'
abnormal = '/content/drive/MyDrive/ABNORMAL'
MI = '/content/drive/MyDrive/MI'
MI_history = '/content/drive/MyDrive/HIS_MI'

Types_ECG = {'normal':normal,'Abnormal_hear_beat':abnormal,'MI':MI,'History_MI':MI_history}

for types,folder in Types_ECG.items():
  for files in os.listdir(folder):
    if types=='normal':
      NORMAL_.append(files)
    elif types=='Abnormal_hear_beat':
      HB_.append(files)
    elif types=='MI':
      MI_.append(files)
    elif types=='History_MI':
      PMI_.append(files)


NORMAL_ = natsorted(NORMAL_)
NORMAL_
```

```
['scaled_data_1D_1.csv',
 'scaled_data_1D_2.csv',
 'scaled_data_1D_3.csv',
 'scaled_data_1D_4.csv',
 'scaled_data_1D_5.csv',
 'scaled_data_1D_6.csv',
 'scaled_data_1D_7.csv',
 'scaled_data_1D_8.csv',
 'scaled_data_1D_9.csv',
 'scaled_data_1D_10.csv',
 'scaled_data_1D_11.csv',
 'scaled_data_1D_12.csv',
 'scaled_data_1D_13.csv']
```

```
MI_ = natsorted(MI_)
MI_
```

```
['scaled_data_1D_1.csv',
 'scaled_data_1D_2.csv',
 'scaled_data_1D_3.csv',
 'scaled_data_1D_4.csv',
 'scaled_data_1D_5.csv',
 'scaled_data_1D_6.csv',
 'scaled_data_1D_7.csv',
 'scaled_data_1D_8.csv',
 'scaled_data_1D_9.csv',
 'scaled_data_1D_10.csv',
 'scaled_data_1D_11.csv',
 'scaled_data_1D_12.csv',
 'scaled_data_1D_13.csv']
```

```
PMI_ = natsorted(PMI_)
PMI_
```

```
['scaled_data_1D_1.csv',
 'scaled_data_1D_2.csv',
 'scaled_data_1D_3.csv',
 'scaled_data_1D_4.csv',
 'scaled_data_1D_5.csv',
 'scaled_data_1D_6.csv',
 'scaled_data_1D_7.csv',
 'scaled_data_1D_8.csv',
 'scaled_data_1D_9.csv',
 'scaled_data_1D_10.csv',
 'scaled_data_1D_11.csv',
 'scaled_data_1D_12.csv',
 'scaled_data_1D_13.csv']
```

```
HB_ = natsorted(HB_)
HB_
```

```
['scaled_data_1D_1.csv',
 'scaled_data_1D_2.csv',
 'scaled_data_1D_3.csv',
 'scaled_data_1D_4.csv',
 'scaled_data_1D_5.csv',
 'scaled_data_1D_6.csv',
 'scaled_data_1D_7.csv',
 'scaled_data_1D_8.csv',
 'scaled_data_1D_9.csv',
 'scaled_data_1D_10.csv',
 'scaled_data_1D_11.csv',
 'scaled_data_1D_12.csv',
 'scaled_data_1D_13.csv']
```

```
for x in range(len(MI_)):
  df1=pd.read_csv('/content/drive/MyDrive/NORMAL/{}'.format(NORMAL_[x]))
  df2=pd.read_csv('/content/drive/MyDrive/ABNORMAL/{}'.format(HB_[x]))
  df3=pd.read_csv('/content/drive/MyDrive/MI/{}'.format(MI_[x]))
  df4=pd.read_csv('/content/drive/MyDrive/HIS_MI/{}'.format(PMI_[x]))
  final_df = pd.concat([df1,df2,df3,df4],ignore_index=True)
  final_df.to_csv('/content/drive/MyDrive/CSV/Combined_IDLead_{}.csv'.format(x+1))
```

```
df=pd.read_csv('/content/drive/MyDrive/CSV/Combined_IDLead_4.csv')
df['Target'].unique()
```

```
array(['No', 'HB', 'MI', 'PM'], dtype=object)
```

```
df.drop(columns=['Unnamed: 0'],inplace=True)
```

```
encode_target_label = df.groupby('Target').ngroup().rename("target").to_frame()
test_final  = df.merge(encode_target_label, left_index=True, right_index=True)
test_final.drop(columns=['Target'],inplace=True)
test_final
```

	0	1	2	3	4	5	6	7	8	9	...	246	247	248	
0	0.809451	0.824043	0.842080	0.822459	0.757465	0.685397	0.619183	0.541821	0.465460	0.385040	...	0.494504	0.570954	0.650062	0.7
1	0.569824	0.615569	0.666696	0.703014	0.735202	0.771098	0.804154	0.813072	0.814730	0.823930	...	0.856130	0.848951	0.840789	0.8
2	0.113263	0.159469	0.227619	0.273481	0.313428	0.299528	0.245614	0.187663	0.143794	0.122099	...	0.324555	0.280572	0.312502	0.2
3	0.784627	0.765097	0.748083	0.728984	0.698198	0.649436	0.623723	0.608578	0.566427	0.531256	...	0.663801	0.645689	0.681868	0.7
4	0.614967	0.599387	0.586633	0.568064	0.539558	0.496940	0.477266	0.467393	0.430404	0.402909	...	0.467256	0.499971	0.536083	0.5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
923	0.739798	0.712922	0.682578	0.667172	0.685945	0.694153	0.645519	0.564976	0.478419	0.393603	...	0.519409	0.606635	0.684209	0.7
924	0.513742	0.497542	0.476405	0.458170	0.438768	0.428422	0.399765	0.374438	0.349900	0.381332	...	0.546212	0.521361	0.486381	0.4
925	0.239724	0.247544	0.230555	0.216509	0.210634	0.205003	0.189393	0.166423	0.163551	0.170094	...	0.218799	0.226693	0.247584	0.2
926	0.491290	0.437395	0.381380	0.325960	0.267678	0.209876	0.155466	0.097997	0.039994	0.000000	...	0.950250	0.907176	0.852748	0.7
927	0.365163	0.352763	0.327322	0.301394	0.281359	0.249260	0.250355	0.300558	0.328623	0.347447	...	0.157060	0.227316	0.295442	0.3

928 rows × 256 columns

```
print(df[['Target']].drop_duplicates().reset_index(drop=True).assign(target=df.groupby('Target').ngroup().drop_duplicates().values))
```

```
  Target  target
0     No       2
1     HB       0
2     MI       1
3     PM       3
```

```
from sklearn.decomposition import PCA

#do PCA and choose componeents as 100
pca = PCA(n_components=100)
x_pca = pca.fit_transform(test_final.iloc[:,0:-1])
x_pca = pd.DataFrame(x_pca)

# Calculate the variance explained by priciple components
explained_variance = pca.explained_variance_ratio_
print('Variance of each component:', pca.explained_variance_ratio_)
print('\n Total Variance Explained:', round(sum(list(pca.explained_variance_ratio_))*100, 2))

#store the new pca generated dimensions in a dataframe
pca_df = pd.DataFrame(data = x_pca)
target = pd.Series(test_final['target'], name='target')
result_df = pd.concat([pca_df, target], axis=1)
result_df
```

```
Variance of each component: [2.47383899e-01 7.06266239e-02 6.46556727e-02 5.42745624e-02
 4.42117606e-02 3.89855236e-02 3.61031950e-02 3.11917688e-02
 2.97125080e-02 2.90713327e-02 2.81223481e-02 2.40783593e-02
 2.35063914e-02 2.07103856e-02 1.87666877e-02 1.75301881e-02
 1.62130428e-02 1.48549850e-02 1.47066025e-02 1.32201516e-02
 1.29178107e-02 1.12377976e-02 1.08458727e-02 1.01911226e-02
 9.01032961e-03 8.39162818e-03 8.21643491e-03 7.64256409e-03
 7.08929403e-03 6.81122704e-03 6.11049827e-03 5.75685104e-03
 5.26557110e-03 5.06347590e-03 4.62761179e-03 4.03446793e-03
 3.29060626e-03 3.00940598e-03 2.82916608e-03 2.48988720e-03
 2.29757591e-03 2.14845804e-03 1.74841408e-03 1.47761985e-03
 1.42566553e-03 1.32940698e-03 1.16904532e-03 1.06724804e-03
 1.01764263e-03 9.32938580e-04 7.89112240e-04 7.22612867e-04
 6.60897492e-04 6.08530972e-04 5.58634173e-04 5.38883822e-04
 5.01680002e-04 4.68429557e-04 4.41321579e-04 3.86414604e-04
 3.61083183e-04 3.40804400e-04 3.20526806e-04 3.00580342e-04
 2.86759690e-04 2.67899357e-04 2.65671643e-04 2.38348816e-04
 2.24004149e-04 2.15604901e-04 2.05955729e-04 1.96817742e-04
 1.78089650e-04 1.70220377e-04 1.62985597e-04 1.54478281e-04
 1.46554575e-04 1.41905817e-04 1.29836973e-04 1.27182124e-04
 1.21259027e-04 1.14843028e-04 1.11723043e-04 1.07270749e-04
 9.92896288e-05 9.60118757e-05 9.10975511e-05 8.42723960e-05
 8.01018575e-05 7.76268811e-05 7.14811020e-05 7.09390042e-05
 6.90557282e-05 6.35894859e-05 5.93705829e-05 5.51404867e-05
 5.22386029e-05 5.15851606e-05 4.86325749e-05 4.39445768e-05]

Total Variance Explained: 99.91
```

	0	1	2	3	4	5	6	7	8	9	...	91	92	
0	1.424635	0.029036	-1.861231	-0.280254	-1.282921	0.779238	0.521492	-0.425271	-0.882762	0.862998	...	-0.019467	-0.030266	-0.027
1	2.149600	-2.647103	0.631040	-0.286040	0.068644	-0.419576	-0.306482	0.139887	0.298175	0.164916	...	0.029864	-0.039418	0.002
2	-1.989952	-0.211234	-0.294950	0.015123	-0.892934	-0.861594	0.165186	0.157708	-0.848662	-0.016847	...	-0.042334	-0.033888	0.012
3	1.894083	0.322655	1.802446	0.016671	-0.356275	0.291556	0.104656	-0.392384	-0.325418	0.399872	...	0.019769	-0.000062	-0.028
4	0.103415	0.812166	-0.159479	-0.309533	-0.351921	-0.073090	0.254038	-0.338110	0.778043	-0.730006	...	0.011238	0.027643	-0.006
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
923	0.506125	0.193093	0.104441	0.738993	-1.152633	1.964962	1.012941	-0.703392	1.133179	-0.436208	...	0.007672	0.004537	0.021
924	-0.597777	0.610828	0.080245	-0.855721	-0.449857	0.352847	0.057263	-0.564911	0.876534	0.002680	...	0.004982	-0.011177	0.018
925	-4.521993	-0.430066	-0.387740	-0.134175	-0.638040	-0.788012	-0.121930	-0.224364	-0.087962	-0.325539	...	-0.020100	-0.022789	0.004
926	1.209005	-0.437725	-1.089207	-0.210384	-1.574087	0.296617	-0.187524	0.451946	-0.983279	0.102724	...	0.041789	-0.008634	0.010
927	-1.916430	1.674895	-0.212192	0.539636	0.532121	0.222149	-1.490695	0.272030	-0.218884	-0.138174	...	0.017355	-0.030056	-0.009

928 rows × 101 columns

```
result_df
```

	0	1	2	3	4	5	6	7	8	9	...	91	92	
0	1.424635	0.029036	-1.861231	-0.280254	-1.282921	0.779238	0.521492	-0.425271	-0.882762	0.862998	...	-0.019467	-0.030266	-0.027
1	2.149600	-2.647103	0.631040	-0.286040	0.068644	-0.419576	-0.306482	0.139887	0.298175	0.164916	...	0.029864	-0.039418	0.002
2	-1.989952	-0.211234	-0.294950	0.015123	-0.892934	-0.861594	0.165186	0.157708	-0.848662	-0.016847	...	-0.042334	-0.033888	0.012
3	1.894083	0.322655	1.802446	0.016671	-0.356275	0.291556	0.104656	-0.392384	-0.325418	0.399872	...	0.019769	-0.000062	-0.028
4	0.103415	0.812166	-0.159479	-0.309533	-0.351921	-0.073090	0.254038	-0.338110	0.778043	-0.730006	...	0.011238	0.027643	-0.006
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
923	0.506125	0.193093	0.104441	0.738993	-1.152633	1.964962	1.012941	-0.703392	1.133179	-0.436208	...	0.007672	0.004537	0.021
924	-0.597777	0.610828	0.080245	-0.855721	-0.449857	0.352847	0.057263	-0.564911	0.876534	0.002680	...	0.004982	-0.011177	0.018
925	-4.521993	-0.430066	-0.387740	-0.134175	-0.638040	-0.788012	-0.121930	-0.224364	-0.087962	-0.325539	...	-0.020100	-0.022789	0.004
926	1.209005	-0.437725	-1.089207	-0.210384	-1.574087	0.296617	-0.187524	0.451946	-0.983279	0.102724	...	0.041789	-0.008634	0.010
927	-1.916430	1.674895	-0.212192	0.539636	0.532121	0.222149	-1.490695	0.272030	-0.218884	-0.138174	...	0.017355	-0.030056	-0.009

928 rows × 101 columns

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import StratifiedKFold
```

```
from sklearn.metrics import classification_report
import numpy as np

# Input
X = np.array(result_df.iloc[:, :-1])

# Target
y = np.array(result_df.iloc[:, -1])

# Initialize Stratified K-Fold
skf = StratifiedKFold(n_splits=5)

acc = []

# Perform Stratified K-Fold Cross-Validation
for i, (train_index, test_index) in enumerate(skf.split(X, y)):
    print(f"Fold {i+1}:")
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    knn = KNeighborsClassifier()
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)

    # Calculate accuracy
    accuracy = knn.score(X_test, y_test)
    acc.append(accuracy)

    print(f"Accuracy: {accuracy}")
    print(classification_report(y_test, y_pred))

# Print the average accuracy across all folds
print(f"Mean Accuracy: {np.mean(acc)}")
```

```
Fold 1:
Accuracy: 0.7473118279569892
              precision    recall  f1-score   support

           0       0.69      0.53      0.60        47
           1       0.80      1.00      0.89        48
           2       0.70      0.74      0.72        57
           3       0.80      0.71      0.75        34

    accuracy                           0.75       186
   macro avg       0.75      0.74      0.74       186
weighted avg       0.74      0.75      0.74       186

Fold 2:
Accuracy: 0.7204301075268817
              precision    recall  f1-score   support

           0       0.74      0.55      0.63        47
           1       0.81      1.00      0.90        47
           2       0.65      0.79      0.71        57
           3       0.67      0.46      0.54        35

    accuracy                           0.72       186
   macro avg       0.72      0.70      0.70       186
weighted avg       0.72      0.72      0.71       186

Fold 3:
Accuracy: 0.7473118279569892
              precision    recall  f1-score   support

           0       0.72      0.61      0.66        46
           1       0.89      1.00      0.94        48
           2       0.70      0.79      0.74        57
           3       0.62      0.51      0.56        35

    accuracy                           0.75       186
   macro avg       0.73      0.73      0.73       186
weighted avg       0.74      0.75      0.74       186

Fold 4:
Accuracy: 0.7675675675675676
              precision    recall  f1-score   support

           0       0.82      0.59      0.68        46
           1       0.81      1.00      0.90        48
           2       0.74      0.89      0.81        57
```

```
                3       0.67      0.47      0.55        34

         accuracy                           0.77       185
        macro avg       0.76      0.74      0.74       185
     weighted avg       0.76      0.77      0.75       185

     Fold 5:
     Accuracy: 0.7621621621621621
                   precision    recall  f1-score   support

                0       0.84      0.57      0.68        47
                1       0.79      1.00      0.88        48
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
import numpy as np

# Input
X = np.array(result_df.iloc[:, :-1])

# Target
y = np.array(result_df.iloc[:, -1])

# Initialize Stratified K-Fold
skf = StratifiedKFold(n_splits=5)

acc = []

# Perform Stratified K-Fold Cross-Validation
for i, (train_index, test_index) in enumerate(skf.split(X, y)):
    print(f"Fold {i+1}:")
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    lr = LogisticRegression()
    lr.fit(X_train, y_train)
    y_pred = lr.predict(X_test)

    # Calculate accuracy
    accuracy = lr.score(X_test, y_test)
    acc.append(accuracy)

    print(f"Accuracy: {accuracy}")
    print(classification_report(y_test, y_pred))

# Print the average accuracy across all folds
print(f"Mean Accuracy: {np.mean(acc)}")
```

```
Fold 1:
Accuracy: 0.5967741935483871
              precision    recall  f1-score   support

           0       0.54      0.32      0.40        47
           1       0.64      0.85      0.73        48
           2       0.65      0.68      0.67        57
           3       0.47      0.47      0.47        34

    accuracy                           0.60       186
   macro avg       0.57      0.58      0.57       186
weighted avg       0.59      0.60      0.58       186

Fold 2:
Accuracy: 0.6129032258064516
              precision    recall  f1-score   support

           0       0.47      0.38      0.42        47
           1       0.65      0.96      0.78        47
           2       0.62      0.61      0.62        57
           3       0.70      0.46      0.55        35

    accuracy                           0.61       186
   macro avg       0.61      0.60      0.59       186
weighted avg       0.61      0.61      0.60       186

Fold 3:
Accuracy: 0.5967741935483871
              precision    recall  f1-score   support

           0       0.67      0.48      0.56        46
           1       0.70      0.77      0.73        48
```

```
                2       0.56      0.68      0.61        57
                3       0.43      0.37      0.40        35

            accuracy                           0.60       186
           macro avg       0.59      0.58      0.58       186
        weighted avg       0.60      0.60      0.59       186

        Fold 4:
        Accuracy: 0.6108108108108108
                      precision    recall  f1-score   support

                0       0.53      0.41      0.46        46
                1       0.66      0.83      0.73        48
                2       0.62      0.72      0.67        57
                3       0.59      0.38      0.46        34

            accuracy                           0.61       185
           macro avg       0.60      0.59      0.58       185
        weighted avg       0.60      0.61      0.60       185

        Fold 5:
        Accuracy: 0.6432432432432432
                      precision    recall  f1-score   support

                0       0.59      0.43      0.49        47
                1       0.73      1.00      0.84        48
```

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import StratifiedKFold

cv=SVC()

X =np.array (result_df.iloc[:,:-1])
y=np.array(result_df.iloc[:,-1])

acc=[]
skf = StratifiedKFold(n_splits=5)

for i, (train_index, test_index) in enumerate(skf.split(X, y)):
    print(f"Fold {i}:")
    X_train, X_test, y_train, y_test = X[train_index],X[test_index],y[train_index],y[test_index] #train_test_split(X,y,test_size=0.2,random_
    cv.fit(X_train,y_train)
    y_pred = cv.predict(X_test)
    SVM_Accuracy = cv.score(X_test, y_test)
    SVM_Accuracy=cv.score(X_test, y_test)
    print("Accuracy: {}".format(SVM_Accuracy))
    print(classification_report(y_test, y_pred))
    acc.append(SVM_Accuracy)
print(np.mean(acc))
```

```
Fold 0:
Accuracy: 0.8763440860215054
              precision    recall  f1-score   support

           0       0.81      0.83      0.82        47
           1       1.00      1.00      1.00        48
           2       0.83      0.88      0.85        57
           3       0.87      0.76      0.81        34

    accuracy                           0.88       186
   macro avg       0.88      0.87      0.87       186
weighted avg       0.88      0.88      0.88       186

Fold 1:
Accuracy: 0.8333333333333334
              precision    recall  f1-score   support

           0       0.83      0.72      0.77        47
           1       0.92      1.00      0.96        47
           2       0.77      0.89      0.83        57
           3       0.82      0.66      0.73        35

    accuracy                           0.83       186
   macro avg       0.84      0.82      0.82       186
weighted avg       0.83      0.83      0.83       186

Fold 2:
Accuracy: 0.8225806451612904
```

```
                precision    recall  f1-score   support

             0       0.73      0.83      0.78        46
             1       0.96      1.00      0.98        48
             2       0.80      0.82      0.81        57
             3       0.80      0.57      0.67        35

      accuracy                           0.82       186
     macro avg       0.82      0.81      0.81       186
  weighted avg       0.82      0.82      0.82       186

  Fold 3:
  Accuracy: 0.8378378378378378
                precision    recall  f1-score   support

             0       0.77      0.78      0.77        46
             1       0.92      1.00      0.96        48
             2       0.81      0.95      0.87        57
             3       0.89      0.50      0.64        34

      accuracy                           0.84       185
     macro avg       0.85      0.81      0.81       185
  weighted avg       0.84      0.84      0.83       185

  Fold 4:
  Accuracy: 0.827027027027027
                precision    recall  f1-score   support

             0       0.79      0.66      0.72        47
             1       0.96      1.00      0.98        48
```

```
location= '/content/drive/MyDrive/CSV/'
for files in natsorted(os.listdir(location)):
  if files.endswith(".csv"):
    if files!='Combined_IDLead_4.csv':
      df=pd.read_csv('/content/drive/MyDrive/CSV/{}'.format(files))
      df.drop(columns=['Unnamed: 0'],inplace=True)
      test_final=pd.concat([test_final,df],axis=1,ignore_index=True)
      test_final.drop(columns=test_final.columns[-1],axis=1,inplace=True)

#drop the target column
test_final.drop(columns=[255],axis=1,inplace=True)
test_final
```

	0	1	2	3	4	5	6	7	8	9	...	3306	3307	3308	
0	0.809451	0.824043	0.842080	0.822459	0.757465	0.685397	0.619183	0.541821	0.465460	0.385040	...	0.666583	0.844202	0.819383	0.8
1	0.569824	0.615569	0.666696	0.703014	0.735202	0.771098	0.804154	0.813072	0.814730	0.823930	...	0.805113	0.746419	0.782025	0.8
2	0.113263	0.159469	0.227619	0.273481	0.313428	0.299528	0.245614	0.187663	0.143794	0.122099	...	0.522917	0.482057	0.488037	0.4
3	0.784627	0.765097	0.748083	0.728984	0.698198	0.649436	0.623723	0.608578	0.566427	0.531256	...	0.713676	0.934875	0.923509	0.9
4	0.614967	0.599387	0.586633	0.568064	0.539558	0.496940	0.477266	0.467393	0.430404	0.402909	...	0.771135	0.778679	0.850710	0.9
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
923	0.739798	0.712922	0.682578	0.667172	0.685945	0.694153	0.645519	0.564976	0.478419	0.393603	...	0.834507	0.773230	0.826538	0.9
924	0.513742	0.497542	0.476405	0.458170	0.438768	0.428422	0.399765	0.374438	0.349900	0.381332	...	0.891928	0.882263	0.900776	0.8
925	0.239724	0.247544	0.230555	0.216509	0.210634	0.205003	0.189393	0.166423	0.163551	0.170094	...	0.806931	0.868186	0.831040	0.4
926	0.491290	0.437395	0.381380	0.325960	0.267678	0.209876	0.155466	0.097997	0.039994	0.000000	...	0.518886	0.638386	0.796492	0.6
927	0.365163	0.352763	0.327322	0.301394	0.281359	0.249260	0.250355	0.300558	0.328623	0.347447	...	0.870166	0.570748	0.298058	0.3

928 rows × 3315 columns

```
test_final.to_csv('final_1D.csv',header=False,index=False)
```

```
test_final.to_csv('/content/drive/MyDrive/CSVs/final_1D.csv', index=False)
```

```
from sklearn.decomposition import PCA

#do PCA and choose componeents as 400
pca = PCA(n_components=400)
x_pca = pca.fit_transform(test_final)
x_pca = pd.DataFrame(x_pca)
```

```
# Calculate the variance explained by priciple components
explained_variance = pca.explained_variance_ratio_
print('Variance of each component:', pca.explained_variance_ratio_)
print('\n Total Variance Explained:', round(sum(list(pca.explained_variance_ratio_))*100, 2))

#store the new pca generated dimensions in a dataframe
#store the new pca generated dimensions in a dataframe
pca_df = pd.DataFrame(data = x_pca)
target = pd.Series(result_df.iloc[:,-1], name='target')
final_result_df = pd.concat([pca_df, target], axis=1)
final_result_df
```

```
Variance of each component: [8.47910621e-02 4.45917558e-02 3.55638892e-02 2.82720839e-02
 2.45129942e-02 2.20102797e-02 2.05139530e-02 1.92712515e-02
 1.85309094e-02 1.71224115e-02 1.54964008e-02 1.44723381e-02
 1.41403370e-02 1.36981215e-02 1.29782042e-02 1.27405997e-02
 1.23024618e-02 1.21048586e-02 1.12503356e-02 1.09911599e-02
 1.03668546e-02 1.00879313e-02 9.80296678e-03 9.61252654e-03
 9.55727254e-03 9.25905124e-03 8.69332393e-03 8.35792057e-03
 8.19215779e-03 8.13898880e-03 7.79153573e-03 7.70430909e-03
 7.42940607e-03 7.11338193e-03 6.93819233e-03 6.74428697e-03
 6.57367087e-03 6.43548987e-03 6.12684047e-03 6.03912488e-03
 5.88961308e-03 5.87096871e-03 5.71460811e-03 5.63269506e-03
 5.50061254e-03 5.40445702e-03 5.28285212e-03 5.12307642e-03
 4.87884107e-03 4.79603526e-03 4.73945248e-03 4.65607864e-03
 4.47521521e-03 4.41402043e-03 4.26613296e-03 4.22002085e-03
 4.15338171e-03 4.05484603e-03 3.99580411e-03 3.93212376e-03
 3.89044768e-03 3.81763711e-03 3.71299043e-03 3.69104682e-03
 3.61968185e-03 3.55244776e-03 3.45275654e-03 3.42110056e-03
 3.31315626e-03 3.27204685e-03 3.20805670e-03 3.16680229e-03
 3.05951759e-03 3.04275763e-03 3.01044249e-03 2.94991081e-03
 2.87747105e-03 2.84045739e-03 2.82036667e-03 2.76204904e-03
 2.73749230e-03 2.71966015e-03 2.65249618e-03 2.62713621e-03
 2.56376042e-03 2.51985139e-03 2.45622282e-03 2.44607507e-03
 2.40045235e-03 2.37282550e-03 2.34986903e-03 2.32294245e-03
 2.28728291e-03 2.24632806e-03 2.23508238e-03 2.19249710e-03
 2.17345320e-03 2.13938511e-03 2.12666372e-03 2.05599948e-03
 2.02684407e-03 1.98697315e-03 1.97515547e-03 1.96236325e-03
 1.94208258e-03 1.90252436e-03 1.86407214e-03 1.84550268e-03
 1.82900566e-03 1.78642933e-03 1.76845127e-03 1.73387446e-03
 1.71379385e-03 1.70478059e-03 1.68356497e-03 1.67181438e-03
 1.64735063e-03 1.62086937e-03 1.61422070e-03 1.56870163e-03
 1.56171037e-03 1.54600240e-03 1.51809520e-03 1.51289773e-03
 1.48467985e-03 1.47563977e-03 1.45119857e-03 1.43093505e-03
 1.41483263e-03 1.40795688e-03 1.39692756e-03 1.37330875e-03
 1.36073190e-03 1.34090556e-03 1.33046786e-03 1.32218336e-03
 1.27922775e-03 1.25542540e-03 1.24452415e-03 1.23776069e-03
 1.22655485e-03 1.21043236e-03 1.19579889e-03 1.18928382e-03
 1.17865415e-03 1.17039755e-03 1.16083526e-03 1.14458414e-03
 1.13109161e-03 1.11983349e-03 1.11163560e-03 1.10354416e-03
 1.09478199e-03 1.08318295e-03 1.06166936e-03 1.05658922e-03
 1.04551634e-03 1.02688112e-03 1.02281213e-03 1.01340365e-03
 1.00372376e-03 9.97484066e-04 9.84370332e-04 9.77749512e-04
 9.72121033e-04 9.58022005e-04 9.52831532e-04 9.26209060e-04
 9.12528011e-04 9.05303077e-04 9.03518526e-04 8.83687421e-04
 8.80922311e-04 8.60050094e-04 8.51142602e-04 8.41027189e-04
 8.31282373e-04 8.28447306e-04 8.18193034e-04 8.16053602e-04
 8.06590320e-04 7.97834077e-04 7.94976677e-04 7.82039532e-04
 7.72765086e-04 7.62199704e-04 7.58270385e-04 7.48412297e-04
 7.42466514e-04 7.27144647e-04 7.21275165e-04 7.16388978e-04
 7.09949839e-04 7.08326171e-04 7.04056975e-04 6.92407357e-04
 6.83199866e-04 6.74549666e-04 6.64221595e-04 6.60061203e-04
 6.52555741e-04 6.47364060e-04 6.45208407e-04 6.38090722e-04
 6.28035498e-04 6.21894046e-04 6.16485753e-04 6.10636143e-04
 5.99388700e-04 5.88984720e-04 5.83006487e-04 5.80996525e-04
 5.79133003e-04 5.73913073e-04 5.72731092e-04 5.67244578e-04
 5.59813048e-04 5.51705636e-04 5.42258598e-04 5.41270269e-04
 5.37602629e-04 5.36272659e-04 5.28344942e-04 5.26038073e-04
 5.23383409e-04 5.20560862e-04 5.14984167e-04 5.08129806e-04
```

```
final_result_df.to_csv("pca_final.csv")
```

```
# prompt: final_result_df.to_csv("pca_final.csv")  can you help me out to save this directly to my drive content/mydrive/CSVs

final_result_df.to_csv('/content/drive/MyDrive/CSVs/pca_final.csv', index=False)
```

```
 3.59827340e-04 3.57453169e-04 3.51528980e-04 3.47889963e-04
 3.45413076e-04 3.43008141e-04 3.42145029e-04 3.36675089e-04
 3.34244149e-04 3.30528872e-04 3.27938680e-04 3.25234588e-04
```

Double-click (or enter) to edit

KNN

```
3.23827462e-04 3.17658992e-04 3.15609806e-04 3.13956006e-04
3.11024355e-04 3.09651276e-04 3.06160637e-04 3.02869806e-04
2.98412261e-04 2.95671392e-04 2.95161017e-04 2.94402003e-04
2.91163196e-04 2.90023503e-04 2.85046943e-04 2.82166755e-04
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Input
X = np.array(final_result_df.iloc[:, :-1])
# Target
y = np.array(final_result_df.iloc[:, -1])

# Initialize Stratified K-Fold
skf = StratifiedKFold(n_splits=5)

acc = []
all_y_test = []
all_y_pred = []
report_dicts = []

# Perform Stratified K-Fold Cross-Validation
for i, (train_index, test_index) in enumerate(skf.split(X, y)):
    print(f"Fold {i+1}:")
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Initialize and Train KNN
    knn = KNeighborsClassifier(n_neighbors=5)  # Adjust as needed
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    acc.append(accuracy)

    # Store predictions and reports
    all_y_test.extend(y_test)
    all_y_pred.extend(y_pred)
    report = classification_report(y_test, y_pred, output_dict=True)
    report_dicts.append(report)

    print(f"Accuracy for Fold {i+1}: {accuracy:.2f}")
    print(classification_report(y_test, y_pred))

# Calculate Mean Accuracy
print(f"\nOverall Mean Accuracy: {np.mean(acc):.2f}")

# Calculate Average Classification Report
def average_reports(report_dicts):
    avg_report = {}
    for key in report_dicts[0].keys():
        if key == 'accuracy':
            avg_report[key] = np.mean([report['accuracy'] for report in report_dicts])
        else:
            avg_report[key] = {}
            for metric in report_dicts[0][key]:
                # Average only numeric values (ignoring non-numeric keys like 'support')
                if isinstance(report_dicts[0][key][metric], (int, float)):
                    avg_report[key][metric] = np.mean([report[key][metric] for report in report_dicts])
    return avg_report

avg_report = average_reports(report_dicts)
print("\nAverage Classification Report:")
print(pd.DataFrame(avg_report).transpose())

# Generate and Plot Confusion Matrix
conf_matrix = confusion_matrix(all_y_test, all_y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Overall Confusion Matrix')
```

```
plt.show()
```

```
Fold 1:
Accuracy for Fold 1: 0.79
              precision    recall  f1-score   support

           0       0.75      0.70      0.73        47
           1       0.94      1.00      0.97        48
           2       0.71      0.88      0.79        57
           3       0.76      0.47      0.58        34

    accuracy                           0.79       186
   macro avg       0.79      0.76      0.77       186
weighted avg       0.79      0.79      0.78       186

Fold 2:
Accuracy for Fold 2: 0.76
              precision    recall  f1-score   support

           0       0.79      0.72      0.76        47
           1       0.94      1.00      0.97        47
           2       0.63      0.86      0.73        57
           3       0.80      0.34      0.48        35

    accuracy                           0.76       186
   macro avg       0.79      0.73      0.73       186
weighted avg       0.78      0.76      0.75       186

Fold 3:
Accuracy for Fold 3: 0.82
              precision    recall  f1-score   support

           0       0.85      0.76      0.80        46
           1       0.89      1.00      0.94        48
           2       0.75      0.95      0.84        57
           3       0.79      0.43      0.56        35

    accuracy                           0.82       186
   macro avg       0.82      0.78      0.78       186
weighted avg       0.82      0.82      0.80       186

Fold 4:
Accuracy for Fold 4: 0.76
              precision    recall  f1-score   support

           0       0.84      0.70      0.76        46
           1       0.84      1.00      0.91        48
           2       0.70      0.84      0.76        57
           3       0.62      0.38      0.47        34

    accuracy                           0.76       185
   macro avg       0.75      0.73      0.73       185
weighted avg       0.76      0.76      0.75       185

Fold 5:
Accuracy for Fold 5: 0.74
              precision    recall  f1-score   support

           0       0.68      0.55      0.61        47
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import *
import numpy as np


# Input
X = np.array(final_result_df.iloc[:, :-1])
# Target
y = np.array(final_result_df.iloc[:, -1])

# Initialize Stratified K-Fold
skf = StratifiedKFold(n_splits=5)

acc = []

f = open("/content/drive/MyDrive/Results/results.txt","a")

# Perform Stratified K-Fold Cross-Validation
for i, (train_index, test_index) in enumerate(skf.split(X, y)):
    print(f"Fold {i+1}:")
```

```
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        knn = KNeighborsClassifier()
        knn.fit(X_train, y_train)
        y_pred = knn.predict(X_test)
        y_pred = [round(value) for value in y_pred]

        # Calculate accuracy
        accuracy = knn.score(X_test, y_test)
        acc.append(accuracy)

        print(f"Accuracy: {accuracy}")
        print(classification_report(y_test, y_pred))
        f1 = f1_score(y_test, y_pred,average='weighted')
        pre = precision_score(y_test, y_pred,average='weighted')
        rec = recall_score(y_test, y_pred,average='weighted')
        f.write("KNN, "+str(i+1)+", "+str(accuracy)+", "+str(f1)+", "+str(pre)+", "+str(rec)+"\n")

    f.close()

    # Print the average accuracy across all folds
    print(f"Mean Accuracy: {np.mean(acc)}")
```

```
Fold 1:
Accuracy: 0.7903225806451613
              precision    recall  f1-score   support

           0       0.75      0.70      0.73        47
           1       0.94      1.00      0.97        48
           2       0.71      0.88      0.79        57
           3       0.76      0.47      0.58        34

    accuracy                           0.79       186
   macro avg       0.79      0.76      0.77       186
weighted avg       0.79      0.79      0.78       186

Fold 2:
Accuracy: 0.7634408602150538
              precision    recall  f1-score   support

           0       0.80      0.74      0.77        47
           1       0.94      1.00      0.97        47
           2       0.63      0.84      0.72        57
           3       0.75      0.34      0.47        35

    accuracy                           0.76       186
   macro avg       0.78      0.73      0.73       186
weighted avg       0.77      0.76      0.75       186

Fold 3:
Accuracy: 0.8225806451612904
              precision    recall  f1-score   support

           0       0.86      0.78      0.82        46
           1       0.91      1.00      0.95        48
           2       0.75      0.95      0.84        57
           3       0.79      0.43      0.56        35

    accuracy                           0.82       186
   macro avg       0.83      0.79      0.79       186
weighted avg       0.82      0.82      0.81       186

Fold 4:
Accuracy: 0.7621621621621621
              precision    recall  f1-score   support

           0       0.84      0.70      0.76        46
           1       0.84      1.00      0.91        48
           2       0.70      0.84      0.76        57
           3       0.62      0.38      0.47        34

    accuracy                           0.76       185
   macro avg       0.75      0.73      0.73       185
weighted avg       0.76      0.76      0.75       185

Fold 5:
Accuracy: 0.7405405405405405
              precision    recall  f1-score   support

           0       0.68      0.55      0.61        47
```

```
           1       0.92      1.00      0.96        48
```

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Input
X = np.array(final_result_df.iloc[:, :-1])
# Target
y = np.array(final_result_df.iloc[:, -1])

# Initialize Stratified K-Fold
skf = StratifiedKFold(n_splits=5)

acc = []
all_y_test = []
all_y_pred = []
report_dicts = []

# Perform Stratified K-Fold Cross-Validation
for i, (train_index, test_index) in enumerate(skf.split(X, y)):
    print(f"Fold {i+1}:")
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Initialize and Train Logistic Regression
    lr = LogisticRegression(max_iter=1000, random_state=42)
    lr.fit(X_train, y_train)
    y_pred = lr.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    acc.append(accuracy)

    # Store predictions and reports
    all_y_test.extend(y_test)
    all_y_pred.extend(y_pred)
    report = classification_report(y_test, y_pred, output_dict=True)
    report_dicts.append(report)

    print(f"Accuracy for Fold {i+1}: {accuracy:.2f}")
    print(classification_report(y_test, y_pred))

# Calculate Mean Accuracy
print(f"\nOverall Mean Accuracy: {np.mean(acc):.2f}")

# Calculate Average Classification Report
def average_reports(report_dicts):
    avg_report = {}
    for key in report_dicts[0].keys():
        if key == 'accuracy':
            avg_report[key] = np.mean([report['accuracy'] for report in report_dicts])
        else:
            avg_report[key] = {}
            for metric in report_dicts[0][key]:
                if isinstance(report_dicts[0][key][metric], (int, float)):
                    avg_report[key][metric] = np.mean([report[key][metric] for report in report_dicts])
    return avg_report

avg_report = average_reports(report_dicts)
print("\nAverage Classification Report:")
print(pd.DataFrame(avg_report).transpose())

# Generate and Plot Confusion Matrix
conf_matrix = confusion_matrix(all_y_test, all_y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Overall Confusion Matrix')
```

```
plt.show()
```

```
Fold 1:
Accuracy for Fold 1: 0.86
              precision    recall  f1-score   support

           0       1.00      0.57      0.73        47
           1       0.94      1.00      0.97        48
           2       0.81      0.96      0.88        57
           3       0.75      0.88      0.81        34

    accuracy                           0.86       186
   macro avg       0.88      0.86      0.85       186
weighted avg       0.88      0.86      0.85       186

Fold 2:
Accuracy for Fold 2: 0.89
              precision    recall  f1-score   support

           0       1.00      0.64      0.78        47
           1       0.96      1.00      0.98        47
           2       0.85      0.96      0.90        57
           3       0.79      0.94      0.86        35

    accuracy                           0.89       186
   macro avg       0.90      0.89      0.88       186
weighted avg       0.90      0.89      0.88       186

Fold 3:
Accuracy for Fold 3: 0.90
              precision    recall  f1-score   support

           0       0.94      0.70      0.80        46
           1       0.96      1.00      0.98        48
           2       0.93      0.93      0.93        57
           3       0.78      1.00      0.88        35

    accuracy                           0.90       186
   macro avg       0.90      0.91      0.90       186
weighted avg       0.91      0.90      0.90       186

Fold 4:
Accuracy for Fold 4: 0.90
              precision    recall  f1-score   support

           0       0.89      0.72      0.80        46
           1       0.96      1.00      0.98        48
           2       0.89      0.96      0.92        57
           3       0.83      0.88      0.86        34

    accuracy                           0.90       185
   macro avg       0.89      0.89      0.89       185
weighted avg       0.90      0.90      0.89       185

Fold 5:
Accuracy for Fold 5: 0.90
              precision    recall  f1-score   support

           0       0.94      0.70      0.80        47
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
import numpy as np

# Input
X = np.array(final_result_df.iloc[:, :-1])

# Target
y = np.array(final_result_df.iloc[:, -1])

# Initialize Stratified K-Fold
skf = StratifiedKFold(n_splits=5)

acc = []

f = open("/content/drive/MyDrive/Results/results.txt","a")

# Perform Stratified K-Fold Cross-Validation
for i, (train_index, test_index) in enumerate(skf.split(X, y)):
    print(f"Fold {i+1}:")
```

```
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    lr = LogisticRegression()
    lr.fit(X_train, y_train)
    y_pred = lr.predict(X_test)
    y_pred = [round(value) for value in y_pred]

    # Calculate accuracy
    accuracy = lr.score(X_test, y_test)
    acc.append(accuracy)

    print(f"Accuracy: {accuracy}")
    print(classification_report(y_test, y_pred))
    f1 = f1_score(y_test, y_pred,average='weighted')
    pre = precision_score(y_test, y_pred,average='weighted')
    rec = recall_score(y_test, y_pred,average='weighted')
    f.write("LR, "+str(i+1)+", "+str(accuracy)+", "+str(f1)+", "+str(pre)+", "+str(rec)+"\n")

f.close()

# Print the average accuracy across all folds
print(f"Mean Accuracy: {np.mean(acc)}")
```

Predicted

```
Fold 1:
Accuracy: 0.8440860215053764
              precision    recall  f1-score   support

           0       0.93      0.55      0.69        47
           1       0.94      1.00      0.97        48
           2       0.80      0.96      0.87        57
           3       0.74      0.82      0.78        34

    accuracy                           0.84       186
   macro avg       0.85      0.84      0.83       186
weighted avg       0.86      0.84      0.84       186

Fold 2:
Accuracy: 0.8924731182795699
              precision    recall  f1-score   support

           0       1.00      0.66      0.79        47
           1       0.98      1.00      0.99        47
           2       0.83      0.96      0.89        57
           3       0.80      0.94      0.87        35

    accuracy                           0.89       186
   macro avg       0.90      0.89      0.89       186
weighted avg       0.91      0.89      0.89       186

Fold 3:
Accuracy: 0.8978494623655914
              precision    recall  f1-score   support

           0       0.94      0.67      0.78        46
           1       0.96      1.00      0.98        48
           2       0.93      0.93      0.93        57
           3       0.76      1.00      0.86        35

    accuracy                           0.90       186
   macro avg       0.90      0.90      0.89       186
weighted avg       0.91      0.90      0.89       186

Fold 4:
Accuracy: 0.9135135135135135
              precision    recall  f1-score   support

           0       0.89      0.74      0.81        46
           1       0.96      1.00      0.98        48
           2       0.89      1.00      0.94        57
           3       0.91      0.88      0.90        34

    accuracy                           0.91       185
   macro avg       0.91      0.91      0.91       185
weighted avg       0.91      0.91      0.91       185

Fold 5:
Accuracy: 0.9027027027027027
              precision    recall  f1-score   support

           0       0.94      0.70      0.80        47
           1       0.92      1.00      0.96        48
```

SVM

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import StratifiedKFold

cv=SVC()

X =np.array (final_result_df.iloc[:,:-1])
y=np.array(final_result_df.iloc[:,-1])

acc=[]
skf = StratifiedKFold(n_splits=5)

f = open("/content/drive/MyDrive/Results/results.txt","a")

for i, (train_index, test_index) in enumerate(skf.split(X, y)):
    print(f"Fold {i}:")
    X_train, X_test, y_train, y_test = X[train_index],X[test_index],y[train_index],y[test_index] #train_test_split(X,y,test_size=0.2,random_
    cv.fit(X_train,y_train)
    y_pred = cv.predict(X_test)
    y_pred = [round(value) for value in y_pred]
    SVM_Accuracy = cv.score(X_test, y_test)

    print("Accuracy: {}".format(SVM_Accuracy))
    print(classification_report(y_test, y_pred))
    acc.append(SVM_Accuracy)

    f1 = f1_score(y_test, y_pred,average='weighted')
    pre = precision_score(y_test, y_pred,average='weighted')
    rec = recall_score(y_test, y_pred,average='weighted')
    f.write("SVM, "+str(i+1)+", "+str(accuracy)+", "+str(f1)+", "+str(pre)+", "+str(rec)+"\n")

f.close()
print(f"Mean Accuracy: {np.mean(acc)}")
```

```
Fold 0:
Accuracy: 0.9301075268817204
              precision    recall  f1-score   support

           0       0.91      0.91      0.91        47
           1       1.00      1.00      1.00        48
           2       0.90      0.95      0.92        57
           3       0.90      0.82      0.86        34

    accuracy                           0.93       186
   macro avg       0.93      0.92      0.92       186
weighted avg       0.93      0.93      0.93       186

Fold 1:
Accuracy: 0.9032258064516129
              precision    recall  f1-score   support

           0       0.89      0.87      0.88        47
           1       1.00      1.00      1.00        47
           2       0.83      0.95      0.89        57
           3       0.93      0.74      0.83        35

    accuracy                           0.90       186
   macro avg       0.91      0.89      0.90       186
weighted avg       0.91      0.90      0.90       186

Fold 2:
Accuracy: 0.9516129032258065
              precision    recall  f1-score   support

           0       0.90      0.93      0.91        46
           1       1.00      1.00      1.00        48
           2       0.93      0.95      0.94        57
           3       1.00      0.91      0.96        35

    accuracy                           0.95       186
   macro avg       0.96      0.95      0.95       186
weighted avg       0.95      0.95      0.95       186

Fold 3:
```

```
Accuracy: 0.9675675675675676
              precision    recall  f1-score   support

           0       0.98      0.91      0.94        46
           1       1.00      1.00      1.00        48
           2       0.95      1.00      0.97        57
           3       0.94      0.94      0.94        34

    accuracy                           0.97       185
   macro avg       0.97      0.96      0.96       185
weighted avg       0.97      0.97      0.97       185

Fold 4:
Accuracy: 0.9027027027027027
              precision    recall  f1-score   support

           0       0.90      0.79      0.84        47
           1       1.00      1.00      1.00        48
```

```
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from sklearn.model_selection import StratifiedKFold
import seaborn as sns
import matplotlib.pyplot as plt

# Input
X = np.array(final_result_df.iloc[:, :-1])
y = np.array(final_result_df.iloc[:, -1])

# Initialize Stratified K-Fold
skf = StratifiedKFold(n_splits=5)

# Initialize SVM Classifier
cv = SVC(probability=True)

acc = []
all_y_test = []
all_y_pred = []
report_dicts = []

# Perform Stratified K-Fold Cross-Validation
for i, (train_index, test_index) in enumerate(skf.split(X, y)):
    print(f"Fold {i+1}:")
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train SVM
    cv.fit(X_train, y_train)
    y_pred = cv.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    acc.append(accuracy)

    # Store predictions and reports
    all_y_test.extend(y_test)
    all_y_pred.extend(y_pred)
    report = classification_report(y_test, y_pred, output_dict=True)
    report_dicts.append(report)

    print(f"Accuracy for Fold {i+1}: {accuracy:.2f}")
    print(classification_report(y_test, y_pred))

# Calculate Mean Accuracy
print(f"\nOverall Mean Accuracy: {np.mean(acc):.2f}")

# Calculate Average Classification Report
def average_reports(report_dicts):
    avg_report = {}
    for key in report_dicts[0].keys():
        if key == 'accuracy':
            avg_report[key] = np.mean([report['accuracy'] for report in report_dicts])
        else:
            avg_report[key] = {}
            for metric in report_dicts[0][key]:
                if isinstance(report_dicts[0][key][metric], (int, float)):
                    avg_report[key][metric] = np.mean([report[key][metric] for report in report_dicts])
```

```
    return avg_report

avg_report = average_reports(report_dicts)
print("\nAverage Classification Report:")
print(pd.DataFrame(avg_report).transpose())

# Generate and Plot Overall Confusion Matrix
conf_matrix = confusion_matrix(all_y_test, all_y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Overall Confusion Matrix')
plt.show()
```

```
Fold 1:
Accuracy for Fold 1: 0.93
              precision    recall  f1-score   support

           0       0.91      0.91      0.91        47
           1       1.00      1.00      1.00        48
           2       0.90      0.95      0.92        57
           3       0.90      0.82      0.86        34

    accuracy                           0.93       186
   macro avg       0.93      0.92      0.92       186
weighted avg       0.93      0.93      0.93       186

Fold 2:
Accuracy for Fold 2: 0.90
              precision    recall  f1-score   support

           0       0.89      0.87      0.88        47
           1       1.00      1.00      1.00        47
           2       0.83      0.95      0.89        57
           3       0.93      0.74      0.83        35

    accuracy                           0.90       186
   macro avg       0.91      0.89      0.90       186
weighted avg       0.91      0.90      0.90       186

Fold 3:
Accuracy for Fold 3: 0.95
              precision    recall  f1-score   support

           0       0.90      0.93      0.91        46
           1       1.00      1.00      1.00        48
           2       0.93      0.95      0.94        57
           3       1.00      0.91      0.96        35

    accuracy                           0.95       186
   macro avg       0.96      0.95      0.95       186
weighted avg       0.95      0.95      0.95       186

Fold 4:
Accuracy for Fold 4: 0.97
              precision    recall  f1-score   support

           0       0.98      0.91      0.94        46
           1       1.00      1.00      1.00        48
           2       0.95      1.00      0.97        57
           3       0.94      0.94      0.94        34

    accuracy                           0.97       185
   macro avg       0.97      0.96      0.96       185
weighted avg       0.97      0.97      0.97       185

Fold 5:
Accuracy for Fold 5: 0.90
              precision    recall  f1-score   support

           0       0.90      0.79      0.84        47
           1       1.00      1.00      1.00        48
           2       0.85      1.00      0.92        56
           3       0.87      0.76      0.81        34
```

XG BOOST

```
from sklearn.model_selection import StratifiedKFold
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report
import numpy as np
```

```
# Input
X = np.array(final_result_df.iloc[:, :-1])

# Target
y = np.array(final_result_df.iloc[:, -1])

# Initialize Stratified K-Fold
skf = StratifiedKFold(n_splits=5)

acc = []


f = open("/content/drive/MyDrive/Results/results.txt","a")
# Perform Stratified K-Fold Cross-Validation
for i, (train_index, test_index) in enumerate(skf.split(X, y)):
    print(f"Fold {i+1}:")
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    model = XGBClassifier()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_pred = [round(value) for value in y_pred]

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    acc.append(accuracy)

    print(f"Accuracy: {accuracy}")
    print(classification_report(y_test, y_pred))
    f1 = f1_score(y_test, y_pred,average='weighted')
    pre = precision_score(y_test, y_pred,average='weighted')
    rec = recall_score(y_test, y_pred,average='weighted')
    f.write("XGBoost, "+str(i+1)+", "+str(accuracy)+", "+str(f1)+", "+str(pre)+", "+str(rec)+"\n")

f.close()

# Print the average accuracy across all folds
print(f"Mean Accuracy: {np.mean(acc)}")
```

```
Fold 1:
Accuracy: 0.9139784946236559
              precision    recall  f1-score   support

           0       0.90      0.74      0.81        47
           1       1.00      1.00      1.00        48
           2       0.87      0.96      0.92        57
           3       0.89      0.94      0.91        34

    accuracy                           0.91       186
   macro avg       0.91      0.91      0.91       186
weighted avg       0.91      0.91      0.91       186

Fold 2:
Accuracy: 0.9247311827956989
              precision    recall  f1-score   support

           0       0.91      0.87      0.89        47
           1       1.00      1.00      1.00        47
           2       0.88      1.00      0.93        57
           3       0.93      0.77      0.84        35

    accuracy                           0.92       186
   macro avg       0.93      0.91      0.92       186
weighted avg       0.93      0.92      0.92       186

Fold 3:
Accuracy: 0.9623655913978495
              precision    recall  f1-score   support

           0       1.00      0.89      0.94        46
           1       1.00      1.00      1.00        48
           2       0.92      1.00      0.96        57
           3       0.94      0.94      0.94        35

    accuracy                           0.96       186
   macro avg       0.97      0.96      0.96       186
weighted avg       0.96      0.96      0.96       186
```

```
Fold 4:
Accuracy: 0.918918918918919
              precision    recall  f1-score   support

           0       0.91      0.85      0.88        46
           1       0.96      1.00      0.98        48
           2       0.90      0.93      0.91        57
           3       0.91      0.88      0.90        34

    accuracy                           0.92       185
   macro avg       0.92      0.92      0.92       185
weighted avg       0.92      0.92      0.92       185

Fold 5:
Accuracy: 0.9621621621621622
              precision    recall  f1-score   support

           0       0.92      0.98      0.95        47
```

```
from sklearn.model_selection import StratifiedKFold
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Input
X = np.array(final_result_df.iloc[:, :-1])
y = np.array(final_result_df.iloc[:, -1])

# Initialize Stratified K-Fold
skf = StratifiedKFold(n_splits=5)

acc = []
all_y_test = []
all_y_pred = []
report_dicts = []

# Perform Stratified K-Fold Cross-Validation
for i, (train_index, test_index) in enumerate(skf.split(X, y)):
    print(f"Fold {i+1}:")
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Initialize and Train XGBoost
    model = XGBClassifier()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    acc.append(accuracy)

    # Store predictions and reports
    all_y_test.extend(y_test)
    all_y_pred.extend(y_pred)
    report = classification_report(y_test, y_pred, output_dict=True)
    report_dicts.append(report)

    print(f"Accuracy for Fold {i+1}: {accuracy:.2f}")
    print(classification_report(y_test, y_pred))

# Calculate Mean Accuracy
print(f"\nOverall Mean Accuracy: {np.mean(acc):.2f}")

# Calculate Average Classification Report
def average_reports(report_dicts):
    avg_report = {}
    for key in report_dicts[0].keys():
        if key == 'accuracy':
            avg_report[key] = np.mean([report['accuracy'] for report in report_dicts])
        else:
            avg_report[key] = {}
            for metric in report_dicts[0][key]:
                if isinstance(report_dicts[0][key][metric], (int, float)):
                    avg_report[key][metric] = np.mean([report[key][metric] for report in report_dicts])
    return avg_report
```

```
avg_report = average_reports(report_dicts)
print("\nAverage Classification Report:")
print(pd.DataFrame(avg_report).transpose())

# Generate and Plot Overall Confusion Matrix
conf_matrix = confusion_matrix(all_y_test, all_y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Overall Confusion Matrix')
plt.show()
```

```
Fold 1:
Accuracy for Fold 1: 0.91
              precision    recall  f1-score   support

           0       0.95      0.77      0.85        47
           1       1.00      1.00      1.00        48
           2       0.85      0.96      0.90        57
           3       0.86      0.88      0.87        34

    accuracy                           0.91       186
   macro avg       0.91      0.90      0.90       186
weighted avg       0.91      0.91      0.91       186

Fold 2:
Accuracy for Fold 2: 0.93
              precision    recall  f1-score   support

           0       0.91      0.89      0.90        47
           1       1.00      1.00      1.00        47
           2       0.86      1.00      0.93        57
           3       1.00      0.77      0.87        35

    accuracy                           0.93       186
   macro avg       0.94      0.92      0.93       186
weighted avg       0.94      0.93      0.93       186

Fold 3:
Accuracy for Fold 3: 0.95
              precision    recall  f1-score   support

           0       0.88      0.96      0.92        46
           1       1.00      1.00      1.00        48
           2       0.93      0.93      0.93        57
           3       1.00      0.89      0.94        35

    accuracy                           0.95       186
   macro avg       0.95      0.94      0.95       186
weighted avg       0.95      0.95      0.95       186

Fold 4:
Accuracy for Fold 4: 0.93
              precision    recall  f1-score   support

           0       0.91      0.85      0.88        46
           1       1.00      1.00      1.00        48
           2       0.91      0.93      0.92        57
           3       0.89      0.94      0.91        34

    accuracy                           0.93       185
   macro avg       0.93      0.93      0.93       185
weighted avg       0.93      0.93      0.93       185

Fold 5:
Accuracy for Fold 5: 0.95
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        47
           1       1.00      1.00      1.00        48
           2       0.89      1.00      0.94        56
           3       0.96      0.76      0.85        34
```

VOTING CLASSIFIER

```
import numpy as np
import random
from sklearn import linear_model, ensemble
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, classification_report, f1_score, precision_score, recall_score
from xgboost import XGBClassifier

# Fix random seeds for reproducibility
np.random.seed(42)
random.seed(42)

# Data Preparation
X = final_result_df.iloc[:, :-1]
y = final_result_df.iloc[:, -1]

# Define classifiers
svm = SVC(probability=True, random_state=42)
knn = KNeighborsClassifier()
rf = ensemble.RandomForestClassifier(random_state=42)
bayes = GaussianNB()
logistic = linear_model.LogisticRegression(random_state=42)
xgb = XGBClassifier(random_state=42)

# Voting Classifier with XGBoost
eclf = VotingClassifier(
    estimators=[
        ('SVM', svm),
        ('knn', knn),
        ('rf', rf),
        ('bayes', bayes),
        ('logistic', logistic),
        ('xgb', xgb)
    ],
    voting='soft'
)

# Stratified K-Fold
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
accuracies = []
f = open("/content/drive/MyDrive/Results/results.txt", "a")

for fold, (train_index, test_index) in enumerate(skf.split(X, y)):
    print(f"Fold {fold + 1}")
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Fit the model
    eclf.fit(X_train, y_train)
    y_pred = eclf.predict(X_test)

    # Evaluate Performance
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)
    print(f"Accuracy: {accuracy:.2f}")
    print(classification_report(y_test, y_pred))

    # Calculate other metrics
    f1 = f1_score(y_test, y_pred, average='weighted')
    pre = precision_score(y_test, y_pred, average='weighted')
    rec = recall_score(y_test, y_pred, average='weighted')

    # Log results
    f.write(f"Voting with XGBoost, {fold+1}, {accuracy}, {f1}, {pre}, {rec}\n")

f.close()

# Print overall average accuracy
print(f"\nOverall Average Accuracy: {np.mean(accuracies):.2f}")
```

```
Fold 1
Accuracy: 0.97
              precision    recall  f1-score   support

           0       0.96      0.94      0.95        47
           1       1.00      1.00      1.00        48
           2       0.97      1.00      0.98        57
           3       0.97      0.94      0.96        34

    accuracy                           0.97       186
   macro avg       0.97      0.97      0.97       186
```

```
weighted avg       0.97      0.97      0.97       186

Fold 2
Accuracy: 0.95
              precision    recall  f1-score   support

           0       0.96      0.94      0.95        47
           1       1.00      1.00      1.00        47
           2       0.92      1.00      0.96        57
           3       0.94      0.83      0.88        35

    accuracy                           0.95       186
   macro avg       0.95      0.94      0.95       186
weighted avg       0.95      0.95      0.95       186

Fold 3
Accuracy: 0.92
              precision    recall  f1-score   support

           0       0.92      0.96      0.94        46
           1       1.00      1.00      1.00        48
           2       0.87      0.96      0.92        57
           3       0.93      0.71      0.81        35

    accuracy                           0.92       186
   macro avg       0.93      0.91      0.91       186
weighted avg       0.93      0.92      0.92       186

Fold 4
Accuracy: 0.95
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        46
           1       1.00      1.00      1.00        48
           2       0.89      1.00      0.94        57
           3       1.00      0.76      0.87        34

    accuracy                           0.95       185
   macro avg       0.96      0.94      0.94       185
weighted avg       0.96      0.95      0.95       185

Fold 5
Accuracy: 0.95
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        47
           1       1.00      1.00      1.00        48
```

```
import numpy as np
import random
from sklearn import linear_model, ensemble
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from xgboost import XGBClassifier
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Fix random seeds for reproducibility
np.random.seed(42)
random.seed(42)

# Data Preparation
X = np.array(final_result_df.iloc[:, :-1])
y = np.array(final_result_df.iloc[:, -1])

# Define classifiers
svm = SVC(probability=True, random_state=42)
knn = KNeighborsClassifier()
rf = ensemble.RandomForestClassifier(random_state=42)
bayes = GaussianNB()
logistic = linear_model.LogisticRegression(random_state=42)
xgb = XGBClassifier(random_state=42)

# Voting Classifier
eclf = VotingClassifier(
    estimators=[
```

```
            ('SVM', svm),
            ('knn', knn),
            ('rf', rf),
            ('bayes', bayes),
            ('logistic', logistic),
            ('xgb', xgb)
        ],
        voting='soft'
    )

    # Stratified K-Fold
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    accuracies = []
    all_y_test = []
    all_y_pred = []
    report_dicts = []

    # Perform Stratified K-Fold Cross-Validation
    for fold, (train_index, test_index) in enumerate(skf.split(X, y)):
        print(f"Fold {fold + 1}")
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        # Fit the model
        eclf.fit(X_train, y_train)
        y_pred = eclf.predict(X_test)

        # Evaluate Performance
        accuracy = accuracy_score(y_test, y_pred)
        accuracies.append(accuracy)
        print(f"Accuracy: {accuracy:.2f}")
        print(classification_report(y_test, y_pred))

        # Store for confusion matrix and average classification report
        all_y_test.extend(y_test)
        all_y_pred.extend(y_pred)
        report = classification_report(y_test, y_pred, output_dict=True)
        report_dicts.append(report)

    # Calculate Average Classification Report
    def average_reports(report_dicts):
        avg_report = {}
        for key in report_dicts[0].keys():
            if key == 'accuracy':
                avg_report[key] = np.mean([report['accuracy'] for report in report_dicts])
            else:
                avg_report[key] = {}
                for metric in report_dicts[0][key]:
                    if isinstance(report_dicts[0][key][metric], (int, float)):
                        avg_report[key][metric] = np.mean([report[key][metric] for report in report_dicts])
        return avg_report

    avg_report = average_reports(report_dicts)
    print("\nAverage Classification Report:")
    print(pd.DataFrame(avg_report).transpose())

    # Print overall average accuracy
    print(f"\nOverall Average Accuracy: {np.mean(accuracies):.2f}")

    # Generate and Plot Confusion Matrix
    conf_matrix = confusion_matrix(all_y_test, all_y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Overall Confusion Matrix')
    plt.show()
```

```
Fold 1
Accuracy: 0.98
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        47
           1       1.00      1.00      1.00        48
           2       1.00      1.00      1.00        57
           3       0.97      0.94      0.96        34

    accuracy                           0.98       186
   macro avg       0.98      0.98      0.98       186
weighted avg       0.98      0.98      0.98       186

Fold 2
Accuracy: 0.96
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        47
           1       1.00      1.00      1.00        47
           2       0.92      1.00      0.96        57
           3       0.97      0.83      0.89        35

    accuracy                           0.96       186
   macro avg       0.96      0.95      0.95       186
weighted avg       0.96      0.96      0.96       186

Fold 3
Accuracy: 0.92
              precision    recall  f1-score   support

           0       0.92      0.96      0.94        46
           1       1.00      1.00      1.00        48
           2       0.87      0.96      0.92        57
           3       0.93      0.71      0.81        35

    accuracy                           0.92       186
   macro avg       0.93      0.91      0.91       186
weighted avg       0.93      0.92      0.92       186

Fold 4
Accuracy: 0.94
              precision    recall  f1-score   support

           0       0.96      0.93      0.95        46
           1       1.00      1.00      1.00        48
           2       0.86      1.00      0.93        57
           3       1.00      0.76      0.87        34

    accuracy                           0.94       185
   macro avg       0.95      0.92      0.93       185
weighted avg       0.95      0.94      0.94       185

Fold 5
Accuracy: 0.96
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        47
           1       1.00      1.00      1.00        48
           2       0.88      1.00      0.93        56
           3       1.00      0.82      0.90        34
```

stacking classifier (base model is SVM)

```
import numpy as np
import random
from sklearn import linear_model, ensemble
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report

# Fix random seeds for reproducibility
np.random.seed(42)
random.seed(42)

# Data Preparation
X = final_result_df.iloc[:, :-1]
y = final_result_df.iloc[:, -1]

# Define base classifiers with random_state set
svm = SVC(probability=True, random_state=42)
```

```
knn = KNeighborsClassifier()
rf = ensemble.RandomForestClassifier(random_state=42)
bayes = GaussianNB()
logistic = linear_model.LogisticRegression(random_state=42)
xgb = XGBClassifier(random_state=42)

# Stacking Classifier
stacking_clf = StackingClassifier(
    estimators=[
        ('logistic', logistic),
        ('knn', knn),
        ('rf', rf),
        ('bayes', bayes),
        ('xgb', xgb)
    ],
    final_estimator=svm,
    cv=5
)

# Perform Stratified K-Fold Cross-Validation
skf = StratifiedKFold(n_splits=5)
acc = []

f = open("/content/drive/MyDrive/Results/results.txt","a")
for i, (train_index, test_index) in enumerate(skf.split(X, y)):
    print(f"Fold {i+1}")
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    stacking_clf.fit(X_train, y_train)
    y_pred = stacking_clf.predict(X_test)
    y_pred = [round(value) for value in y_pred]

    accuracy = stacking_clf.score(X_test, y_test)
    acc.append(accuracy)

    print(f"Accuracy: {accuracy:.2f}")
    print(classification_report(y_test, y_pred))
    f1 = f1_score(y_test, y_pred,average='weighted')
    pre = precision_score(y_test, y_pred,average='weighted')
    rec = recall_score(y_test, y_pred,average='weighted')
    f.write("Stacking, "+str(i+1)+", "+str(accuracy)+", "+str(f1)+", "+str(pre)+", "+str(rec)+"\n")

f.close()

print(f"Mean Accuracy: {np.mean(acc):.2f}")
```

```
Fold 1
Accuracy: 0.98
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        47
           1       1.00      1.00      1.00        48
           2       0.96      0.96      0.96        57
           3       0.94      0.94      0.94        34

    accuracy                           0.98       186
   macro avg       0.98      0.98      0.98       186
weighted avg       0.98      0.98      0.98       186

Fold 2
Accuracy: 0.99
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        47
           1       1.00      1.00      1.00        47
           2       0.97      1.00      0.98        57
           3       1.00      0.94      0.97        35

    accuracy                           0.99       186
   macro avg       0.99      0.99      0.99       186
weighted avg       0.99      0.99      0.99       186

Fold 3
Accuracy: 0.98
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        46
           1       1.00      1.00      1.00        48
```

```
                2       1.00      0.93      0.96        57
                3       0.90      1.00      0.95        35

           accuracy                           0.98       186
          macro avg       0.97      0.98      0.98       186
       weighted avg       0.98      0.98      0.98       186

       Fold 4
       Accuracy: 0.99
                    precision    recall  f1-score   support

                0       1.00      1.00      1.00        46
                1       1.00      1.00      1.00        48
                2       1.00      0.96      0.98        57
                3       0.94      1.00      0.97        34

           accuracy                           0.99       185
          macro avg       0.99      0.99      0.99       185
       weighted avg       0.99      0.99      0.99       185

       Fold 5
       Accuracy: 0.98
                    precision    recall  f1-score   support

                0       1.00      1.00      1.00        47
                1       1.00      1.00      1.00        48
```

```
import numpy as np
import random
from sklearn import linear_model, ensemble
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Fix random seeds for reproducibility
np.random.seed(42)
random.seed(42)

# Data Preparation
X = np.array(final_result_df.iloc[:, :-1])
y = np.array(final_result_df.iloc[:, -1])

# Define base classifiers with random_state set
svm = SVC(probability=True, random_state=42)
knn = KNeighborsClassifier()
rf = ensemble.RandomForestClassifier(random_state=42)
bayes = GaussianNB()
logistic = linear_model.LogisticRegression(random_state=42)
xgb = XGBClassifier(random_state=42)

# Stacking Classifier
stacking_clf = StackingClassifier(
    estimators=[
        ('logistic', logistic),
        ('knn', knn),
        ('rf', rf),
        ('bayes', bayes),
        ('xgb', xgb)
    ],
    final_estimator=svm,
    cv=5
)

# Perform Stratified K-Fold Cross-Validation
skf = StratifiedKFold(n_splits=5)
acc = []
all_y_test = []
all_y_pred = []
report_dicts = []

for i, (train_index, test_index) in enumerate(skf.split(X, y)):
    print(f"Fold {i+1}")
```

```
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        # Fit and Predict
        stacking_clf.fit(X_train, y_train)
        y_pred = stacking_clf.predict(X_test)

        # Calculate accuracy
        accuracy = accuracy_score(y_test, y_pred)
        acc.append(accuracy)

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

        # Store for overall analysis
        all_y_test.extend(y_test)
        all_y_pred.extend(y_pred)
        report = classification_report(y_test, y_pred, output_dict=True)
        report_dicts.append(report)

    # Calculate Average Classification Report
    def average_reports(report_dicts):
        avg_report = {}
        for key in report_dicts[0].keys():
            if key == 'accuracy':
                avg_report[key] = np.mean([report['accuracy'] for report in report_dicts])
            else:
                avg_report[key] = {}
                for metric in report_dicts[0][key]:
                    if isinstance(report_dicts[0][key][metric], (int, float)):
                        avg_report[key][metric] = np.mean([report[key][metric] for report in report_dicts])
        return avg_report

    avg_report = average_reports(report_dicts)
    print("\nAverage Classification Report:")
    print(pd.DataFrame(avg_report).transpose())

    # Print overall average accuracy
    print(f"\nOverall Average Accuracy: {np.mean(acc):.2f}")

    # Generate and Plot Confusion Matrix
    conf_matrix = confusion_matrix(all_y_test, all_y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Overall Confusion Matrix')
    plt.show()
```

```
Fold 1
Accuracy: 0.98
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        47
           1       1.00      1.00      1.00        48
           2       0.96      0.96      0.96        57
           3       0.94      0.94      0.94        34

    accuracy                           0.98       186
   macro avg       0.98      0.98      0.98       186
weighted avg       0.98      0.98      0.98       186

Fold 2
Accuracy: 0.98
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        47
           1       1.00      1.00      1.00        47
           2       0.93      1.00      0.97        57
           3       1.00      0.89      0.94        35

    accuracy                           0.98       186
   macro avg       0.98      0.97      0.98       186
weighted avg       0.98      0.98      0.98       186

Fold 3
Accuracy: 0.98
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        46
           1       1.00      1.00      1.00        48
           2       1.00      0.93      0.96        57
           3       0.90      1.00      0.95        35

    accuracy                           0.98       186
   macro avg       0.97      0.98      0.98       186
weighted avg       0.98      0.98      0.98       186

Fold 4
Accuracy: 0.98
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        46
           1       1.00      1.00      1.00        48
           2       1.00      0.93      0.96        57
           3       0.89      1.00      0.94        34

    accuracy                           0.98       185
   macro avg       0.97      0.98      0.98       185
weighted avg       0.98      0.98      0.98       185

Fold 5
Accuracy: 0.98
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        47
           1       1.00      1.00      1.00        48
           2       0.95      1.00      0.97        56
           3       1.00      0.88      0.94        34
```

stacking classifier (base model is XGBoost)

```
import numpy as np
import random
from sklearn import linear_model, ensemble
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report

# Fix random seeds for reproducibility
np.random.seed(42)
random.seed(42)

# Data Preparation
X = final_result_df.iloc[:, :-1]
y = final_result_df.iloc[:, -1]

# Define base classifiers with random_state set
svm = SVC(probability=True, random_state=42)
```

```
knn = KNeighborsClassifier()
rf = ensemble.RandomForestClassifier(random_state=42)
bayes = GaussianNB()
logistic = linear_model.LogisticRegression(random_state=42)
xgb = XGBClassifier(random_state=42)

# Stacking Classifier
stacking_clf = StackingClassifier(
    estimators=[
        ('SVM', svm),
        ('knn', knn),
        ('rf', rf),
        ('bayes', bayes),
        ('logistic', logistic),
    ],
    final_estimator=xgb,
    cv=5
)

# Perform Stratified K-Fold Cross-Validation
skf = StratifiedKFold(n_splits=5)
acc = []

f = open("/content/drive/MyDrive/Results/results.txt","a")
for i, (train_index, test_index) in enumerate(skf.split(X, y)):
    print(f"Fold {i+1}")
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    stacking_clf.fit(X_train, y_train)
    y_pred = stacking_clf.predict(X_test)
    y_pred = [round(value) for value in y_pred]

    accuracy = stacking_clf.score(X_test, y_test)
    acc.append(accuracy)

    print(f"Accuracy: {accuracy:.2f}")
    print(classification_report(y_test, y_pred))
    f1 = f1_score(y_test, y_pred,average='weighted')
    pre = precision_score(y_test, y_pred,average='weighted')
    rec = recall_score(y_test, y_pred,average='weighted')
    f.write("Stacking(XGB), "+str(i+1)+", "+str(accuracy)+", "+str(f1)+", "+str(pre)+", "+str(rec)+"\n")

f.close()

print(f"Mean Accuracy: {np.mean(acc):.2f}")
```

```
Fold 1
Accuracy: 0.98
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        47
           1       1.00      1.00      1.00        48
           2       0.96      0.96      0.96        57
           3       0.94      0.94      0.94        34

    accuracy                           0.98       186
   macro avg       0.98      0.98      0.98       186
weighted avg       0.98      0.98      0.98       186

Fold 2
Accuracy: 0.96
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        47
           1       1.00      1.00      1.00        47
           2       0.90      0.96      0.93        57
           3       0.94      0.83      0.88        35

    accuracy                           0.96       186
   macro avg       0.96      0.95      0.95       186
weighted avg       0.96      0.96      0.96       186

Fold 3
Accuracy: 0.99
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        46
           1       1.00      1.00      1.00        48
```

```
                2       1.00      0.96      0.98        57
                3       0.95      1.00      0.97        35

           accuracy                           0.99       186
          macro avg       0.99      0.99      0.99       186
       weighted avg       0.99      0.99      0.99       186

       Fold 4
       Accuracy: 0.99
                    precision    recall  f1-score   support

                0       1.00      1.00      1.00        46
                1       1.00      1.00      1.00        48
                2       1.00      0.96      0.98        57
                3       0.94      1.00      0.97        34

           accuracy                           0.99       185
          macro avg       0.99      0.99      0.99       185
       weighted avg       0.99      0.99      0.99       185

       Fold 5
       Accuracy: 0.97
                    precision    recall  f1-score   support

                0       1.00      0.98      0.99        47
                1       1.00      1.00      1.00        48
```

```
import numpy as np
import random
from sklearn import linear_model, ensemble
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Fix random seeds for reproducibility
np.random.seed(42)
random.seed(42)

# Data Preparation
X = np.array(final_result_df.iloc[:, :-1])
y = np.array(final_result_df.iloc[:, -1])

# Define base classifiers with random_state set
svm = SVC(probability=True, random_state=42)
knn = KNeighborsClassifier()
rf = ensemble.RandomForestClassifier(random_state=42)
bayes = GaussianNB()
logistic = linear_model.LogisticRegression(random_state=42)
xgb = XGBClassifier(random_state=42)

# Stacking Classifier
stacking_clf = StackingClassifier(
    estimators=[
        ('SVM', svm),
        ('knn', knn),
        ('rf', rf),
        ('bayes', bayes),
        ('logistic', logistic),
    ],
    final_estimator=xgb,
    cv=5
)

# Perform Stratified K-Fold Cross-Validation
skf = StratifiedKFold(n_splits=5)
acc = []
all_y_test = []
all_y_pred = []
report_dicts = []

for i, (train_index, test_index) in enumerate(skf.split(X, y)):
    print(f"Fold {i+1}")
```

```
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        # Fit and Predict
        stacking_clf.fit(X_train, y_train)
        y_pred = stacking_clf.predict(X_test)

        # Calculate accuracy
        accuracy = accuracy_score(y_test, y_pred)
        acc.append(accuracy)

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

        # Store for overall analysis
        all_y_test.extend(y_test)
        all_y_pred.extend(y_pred)
        report = classification_report(y_test, y_pred, output_dict=True)
        report_dicts.append(report)

    # Calculate Average Classification Report
    def average_reports(report_dicts):
        avg_report = {}
        for key in report_dicts[0].keys():
            if key == 'accuracy':
                avg_report[key] = np.mean([report['accuracy'] for report in report_dicts])
            else:
                avg_report[key] = {}
                for metric in report_dicts[0][key]:
                    if isinstance(report_dicts[0][key][metric], (int, float)):
                        avg_report[key][metric] = np.mean([report[key][metric] for report in report_dicts])
        return avg_report

    avg_report = average_reports(report_dicts)
    print("\nAverage Classification Report:")
    print(pd.DataFrame(avg_report).transpose())

    # Print overall average accuracy
    print(f"\nOverall Average Accuracy: {np.mean(acc):.2f}")

    # Generate and Plot Confusion Matrix
    conf_matrix = confusion_matrix(all_y_test, all_y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Overall Confusion Matrix')
    plt.show()
```

```
Fold 1
Accuracy: 0.99
              precision    recall  f1-score   support
```