```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
df_origin = pd.read_csv("./Data/D_train.csv")
```

```
df = df_origin
df = df.drop(['Unnamed: 0'], axis=1)
a1 = df.iloc[:,np.arange(2,35,3)].apply(np.mean,axis=1)
a2 = df.iloc[:,np.arange(3,36,3)].apply(np.mean,axis=1)
a3 = df.iloc[:,np.arange(4,37,3)].apply(np.mean,axis=1)
res = pd.DataFrame({'X_mean':a1,'Y_mean':a2,'Z_mean':a3})
res.insert(0,'Class',df['Class'])
res.insert(1,'User',df['User'])
res.insert(5,'X_std',df.iloc[:,np.arange(2,35,3)].apply(np.std,axis=1))
res.insert(6,'Y_std',df.iloc[:,np.arange(3,36,3)].apply(np.std,axis=1))
res.insert(7,'Z_std',df.iloc[:,np.arange(4,37,3)].apply(np.std,axis=1))
res.insert(8,'X_min',df.iloc[:,np.arange(2,35,3)].apply(np.min,axis=1))
res.insert(9,'Y_min',df.iloc[:,np.arange(3,36,3)].apply(np.min,axis=1))
res.insert(10,'Z_min',df.iloc[:,np.arange(4,37,3)].apply(np.min,axis=1))
res.insert(11,'X_max',df.iloc[:,np.arange(2,35,3)].apply(np.max,axis=1))
res.insert(12,'Y_max',df.iloc[:,np.arange(3,36,3)].apply(np.max,axis=1))
res.insert(13,'Z_max',df.iloc[:,np.arange(4,37,3)].apply(np.max,axis=1))
res.insert(14,'Null',df.isnull().sum(axis=1))
```

```
k = 0
for i in res['Class'].unique():
    plt.subplot(231+k)
    res[res['Class']==i].loc[:,'Null'].value_counts().plot(kind='bar')
    plt.title("NULL values of Class {}".format(i))
    plt.show()
    k += 1
```

```
for i in res['Class'].unique():
    sns.heatmap(res.loc[res['Class']==i].iloc[:,:4].corr())
    plt.title("Class {}'s corr matrix'".format(i))
    plt.show()
# res.loc[res['User']==2].corr()['Y_max']
# res['Class'].unique()
```

```
res.iloc[:,2:].hist(figsize=(8,8), sharex=True)
plt.show()
res.iloc[:,2:].describe()
```

```
# PCA, check the scree plot and variance sum
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

pca = PCA(n_components=6)
without_std = res.iloc[:,1:].copy()
pca.fit(without_std)
```

```
print("PCA first 6 largest components without standardization")
print(pca.explained_variance_ratio_)
print("The first 6 components adds up to:", pca.explained_variance_ratio_.sum())
scaler = StandardScaler()
scaler.fit(without_std)
with_std = scaler.transform(without_std)
pca.fit(with_std)
print("\nPCA first 6 largest components with standardization")
print(pca.explained_variance_ratio_)
print("The first 6 components adds up to:", pca.explained_variance_ratio_.sum())
# pca.components_
```

```
pca = PCA()
pca.fit(res.iloc[:,2:])
print("The 6 components ratio that adds up to:",
pca.explained_variance_ratio_[:6].sum().round(4))
```

```
cor_index = res.corr()['Class'].abs().sort_values(ascending=False)[:7].index
cor = res[cor_index.tolist()].copy()
cor.insert(7,'User',res['User'])
cor.head()
```

```
# We pick the first 6 eigenvectors
pca = PCA(n_components=6)
res_pca = pca.fit_transform(res.iloc[:,2:])
after_pca = pd.DataFrame(data=res_pca, columns = ['1','2','3','4','5','6'])
after_pca
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import PredefinedSplit
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier

# Pick the params according to the pipeline you construct

params = {
    'svm__C':[0.1,0.5,1,10,50,100,500],
    'svm__gamma':[0.001,0.005,0.01,0.05,0.1,0.5,1,5,10,50],
    'dc__max_depth':[None,1,3,5,7,9],
    'knn__n_neighbors':[1,2,3,4,5,6,7],
    'preprocessing__pca__n_components':[4,5,6],
    'preprocessing__scaler__with_std': [True, False],
    'preprocessing__scaler__with_mean': [True, False],}

preprocessing = Pipeline([('scaler', StandardScaler()),
                         ('pca', PCA()),
                         ('lda', LinearDiscriminantAnalysis())])

motion_pipeline = Pipeline([('preprocessing', preprocessing),
                            ('svm', SVC(kernel='rbf'))])
```

```
X_dropped = res.drop('Class', axis=1)
y_dropped = res['Class']
# get_best_model_and_accuracy(motion_pipeline, params, X_dropped, y_dropped)
cv = res['User'].astype(int)
cv = PredefinedSplit(cv)
gs = GridSearchCV(motion_pipeline,params,cv=cv, error_score=0., n_jobs=-1)
gs.fit(X_dropped, y_dropped)
print("Best Accuracy: {}".format(gs.best_score_))
print("Best Parameters: {}".format(gs.best_params_))
print("Average Time to Fit (s):
{}".format(round(gs.cv_results_['mean_fit_time'].mean(), 3)))
```

```
##########Test section##############
df_test_o = pd.read_csv("./Data/D_test.csv")
df_test_o = df_test_o.drop(['Unnamed: 0'], axis=1)
df_test_o.head()
df_test = df_test_o.copy()
```

```
a1 = df_test.iloc[:,np.arange(2,35,3)].apply(np.mean,axis=1)
a2 = df_test.iloc[:,np.arange(3,36,3)].apply(np.mean,axis=1)
a3 = df_test.iloc[:,np.arange(4,37,3)].apply(np.mean,axis=1)
res_test = pd.DataFrame({'X_mean':a1,'Y_mean':a2,'Z_mean':a3})
res_test.insert(0,'Class',df_test['Class'])
res_test.insert(1,'User',df_test['User'])
res_test.insert(5,'X_std',df_test.iloc[:,np.arange(2,35,3)].apply(np.std,axis=1)
)
res_test.insert(6,'Y_std',df_test.iloc[:,np.arange(3,36,3)].apply(np.std,axis=1)
)
res_test.insert(7,'Z_std',df_test.iloc[:,np.arange(4,37,3)].apply(np.std,axis=1)
)
res_test.insert(8,'X_min',df_test.iloc[:,np.arange(2,35,3)].apply(np.min,axis=1)
)
res_test.insert(9,'Y_min',df_test.iloc[:,np.arange(3,36,3)].apply(np.min,axis=1)
)
res_test.insert(10,'Z_min',df_test.iloc[:,np.arange(4,37,3)].apply(np.min,axis=1
))
res_test.insert(11,'X_max',df_test.iloc[:,np.arange(2,35,3)].apply(np.max,axis=1
))
res_test.insert(12,'Y_max',df_test.iloc[:,np.arange(3,36,3)].apply(np.max,axis=1
))
res_test.insert(13,'Z_max',df_test.iloc[:,np.arange(4,37,3)].apply(np.max,axis=1
))
res_test.insert(14,'Null',df_test.isnull().sum(axis=1))
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

X_test = res_test.drop('Class', axis=1)
y_test = res_test['Class']
yyss = grid.best_estimator_.predict(X_test)
print("The confusion matrix is:\n",confusion_matrix(y_test, yyss))
print("The accuracy on the test set is:", accuracy_score(y_test, yyss))
################################################
```