```
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
import numpy as np
import seaborn as sns

from sklearn.model_selection import KFold, train_test_split,
cross_val_score, StratifiedKFold, cross_validate, cross_val_predict
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.feature_selection import SelectKBest, f_classif, chi2
from sklearn.metrics import confusion_matrix, classification_report,
log_loss, hinge_loss, make_scorer
from sklearn.model_selection import GridSearchCV
from pandas.plotting import table

data = pd.read_csv('Machine learning.csv')
data.head()
```

# DATA PROCESSING

```
data.info()

data.describe()
```

# Scaling and Encoding

```
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
norm = MinMaxScaler()
encode = LabelEncoder()

data['Location'] = encode.fit_transform(data['Location'])
data.head()

Data_x = data.iloc[:,:-1]
Data_x
col = list(Data_x.keys())
Data_tf = pd.DataFrame(data=norm.fit_transform(Data_x), columns=col)
Data_tf['Bioturbation Index'] = data.iloc[:,-1]
Data_tf.head()
```

# Feature selection using SelectKBest

```
fe_selection = SelectKBest(score_func=f_classif,
k=8).fit_transform(Data_tf.iloc[:,:-1],Data_tf.iloc[:,-1])
fe_selection.shape
feat_select = pd.DataFrame(fe_selection)
feat_select.head(5)

feat_select = SelectKBest(score_func=f_classif,
k=5).fit(data.iloc[:,:-1],data.iloc[:,-1])
param = pd.DataFrame()
param['features'] = data.iloc[:,:-1].columns
param['f_score'] = feat_select.scores_
param['P_values'] = feat_select.pvalues_
param['Features_bool'] = feat_select.get_support()
param= param.sort_values(by='f_score', ascending=False)
param = param.round(5)
param

plt.figure(figsize=(13,7))
plt.bar(param['features'], param['P_values'])
plt.ylabel('P-Values')
plt.xticks(rotation = 45)
plt.title('Features P-values')
plt.savefig('P-Values')

plt.figure(figsize=(15,8))
plt.bar(param['features'],1 - param['P_values'])
plt.xticks(rotation = 45)
plt.ylabel('P-Values')
plt.title('Features {1-} P-values')
plt.savefig('1-(P-Values)')

New_data = Data_tf.drop(['Sample Length','Particle Volume','Dry
Weight','Sample Volume','Pore Volume','Bioturbation Index'], axis=1)
New_data['Bioturbation Index'] = Data_tf.iloc[:,-1]
New_data.head()

corr = New_data.corr()
plt.figure(figsize= (11,8))
sns.heatmap(corr, annot=True, cbar=True)

#sns.pairplot(New_data,hue='Bioturbation Index', palette= ['red',
'green','blue','violet','purple'])
```

# Data splitting, Cross_validation and Retraining

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

```
X_train, X_test, y_train, y_test = train_test_split(X, y ,
test_size=0.2, random_state=42)
print('X_train shape = ', X_train.shape )
print('X_test shape = ', X_test.shape )
print('y_train shape = ', y_train.shape )
print('y_test shape = ', y_test.shape )

Cl_svc = SVC()
Cl_lda = LinearDiscriminantAnalysis()
models = [ Cl_svc, Cl_lda]

splits = StratifiedKFold(n_splits=5, shuffle=True)
SVCcv_scr = cross_validate(Cl_svc, X_train, y_train, cv=splits,
return_estimator=True, return_train_score=True)

ldacv_scr = cross_validate(Cl_lda, X_train, y_train, cv=splits,
return_estimator=True, return_train_score=True)

scores = pd.DataFrame({'SVC': (1-SVCcv_scr['train_score']),
                       'LDA': (1-ldacv_scr['train_score'])})
scores

print('SVC: ',SVCcv_scr['train_score'].mean())
print('LDA: ',ldacv_scr['train_score'].mean())

k_neighbors =  range(3,11,2)
for k in k_neighbors:
    Cl_knn = KNeighborsClassifier(n_neighbors=k)
    knncv_scr = cross_validate(Cl_knn, X_train, y_train, cv=splits,
return_estimator=True, return_train_score=True)
    knnscore = pd.DataFrame({k: knncv_scr['train_score']})
    print(knnscore)

CRS_pred = SVCcv_scr['estimator'][0].predict(X_train)
CRS_report = classification_report(y_train,CRS_pred,
target_names=["Class 0","Class 1","Class 2","Class 3","Class 4","Class
5","Class 6"])
print(CRS_report)

CRS_pred = SVCcv_scr['estimator'][0].predict(X_train)
CRS_report = confusion_matrix(y_train,CRS_pred)
print(CRS_report)

CRK_pred = knncv_scr['estimator'][0].predict(X_train)
CRK_report = confusion_matrix(y_train,CRK_pred)
print(CRK_report)

CRL_pred = ldacv_scr['estimator'][0].predict(X_train)
CRL_report = confusion_matrix(y_train,CRL_pred)
print(CRL_report)
```

# Optimization with Gridsearchcv

```
gdModel = KNeighborsClassifier()
params = [{ 'n_neighbors':[3,5,7,9],
            'weights' : ['uniform', 'distance'],
            'algorithm' : ['auto', 'ball_tree', 'kd_tree', 'brute'],
             'leaf_size' : [5,10,15,10]
          }]
Gdkmodel = GridSearchCV(estimator=gdModel, param_grid=params,
scoring='accuracy', return_train_score=True, cv=splits)
Gdkmodel.fit(X_train, y_train)
print(Gdkmodel.best_params_)
print(Gdkmodel.best_score_)

gdsModel = SVC()
params = [{'C': [0.1, 1, 10, 100, 1000],
           'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
           'kernel': ['rbf', 'linear']
          }]
Gdsmodel = GridSearchCV(estimator=gdsModel, param_grid=params,
scoring='accuracy', return_train_score=True, cv=splits)
Gdsmodel.fit(X_train, y_train)
print(Gdsmodel.best_params_)
print(Gdsmodel.best_score_)

gdlModel =  LinearDiscriminantAnalysis()
grid = dict()
grid['solver'] = ['svd', 'eigen','lsqr']
grid['shrinkage'] = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,]
Gdlmodel = GridSearchCV(gdlModel, param_grid = grid,
scoring='accuracy', return_train_score=True, cv=splits)
Gdlmodel.fit(X_train, y_train)
print('Best Prams:', Gdlmodel.best_params_)
print('Best Score:', Gdlmodel.best_score_)

from sklearn.metrics import accuracy_score

for k in range(3,11,2):#'algorithm': 'auto', 'leaf_size': 5,
'n_neighbors': 3, 'weights': 'uniform'
    cl_knn1 =
KNeighborsClassifier(n_neighbors=k,algorithm='auto',leaf_size=5,
weights='uniform')
    cl_knn1.fit(X_train,y_train)
    knn_pred = cl_knn1.predict(X_train)
    knn_prob = cl_knn1.predict_proba(X_test)
    print(cl_knn1.score(X_train, y_train))

C= [1,10,100,1000]
for c in C:
    cl_svm = SVC(C=c, gamma=0.01,kernel='rbf')
    cl_svm.fit(X_train,y_train)
    svm_pred = cl_svm.predict(X_train)
    print(cl_svm.score(X_train, y_train))
```

# TRAINING AND TESTING OF BEST HYPERPARAMETERS

```
#Best Prams: {'shrinkage': 0.9, 'solver': 'eigen'}
cl_lda = LinearDiscriminantAnalysis(shrinkage=0.9, solver='eigen')
cl_lda.fit(X_train,y_train)
lda_pred = cl_lda.predict(X_train)
lda_prob = cl_lda.predict_proba(X_test)
print(cl_lda.score(X_train, y_train))
Ldatest_prediction = cl_lda.predict(X_test)

svm_train = SVC(C=100, kernel='rbf', gamma=1)
svm_train.fit(X_train,y_train)
print(svm_train.score(X_train,y_train))
Svmtest_prediction = svm_train.predict(X_test)

knn_train = KNeighborsClassifier(n_neighbors=3)
knn_train.fit(X_train, y_train)
print(knn_train.score(X_train, y_train))
Knntest_prediction = knn_train.predict(X_test)

y_test = list(y_test)
outcomes = pd.DataFrame(data=y_test, columns=['Actual value'])
outcomes['KNN_PRED'] = Knntest_prediction
outcomes['LDA_PRED'] = Ldatest_prediction
outcomes['SVM_PRED'] = Svmtest_prediction
outcomes

# TESTING OF DATASETS
print('KNN Accuracy: ', accuracy_score(y_test, Knntest_prediction))
print('SVM Accuracy: ', accuracy_score(y_test, Svmtest_prediction))
print('LDA Accuracy: ', accuracy_score(y_test, Ldatest_prediction))
```