```
"""
Developing a Machine Learning Algorithm to Predict Daily Functioning in a Population
of Adolescents Living With Chronic Pain

Psychology Honors Thesis
Max Kramer, Oberlin College Class of 2020

This code is designed to be run after the R markdown file generates the dataset
"""

#
# EXTERNAL LIBRARY IMPORTATION
#

import sys
import csv
from math import sqrt
import pandas as pd
import numpy as np
import matplotlib.pyplot
from sklearn import preprocessing, metrics, model_selection

#
# MODELS
#

from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression

#
# Helper Functions
#

def readin(path, verbose=False): # takes verbose flag
    dataset = pd.read_csv(path) # read csv from supplied filepath
    if verbose:  # for diagnostics
        print('dataset contains {} instances and {} attributes'.format(dataset.shape[0], dataset.shape[1] - 1))
    return dataset

def instanceFormat(dataset,k,n):
    X = []
    y = []
    participants = np.arange(1,101)
    for participant in participants:
        for label_day in range(k+n,29):
            label = (dataset.label)[(dataset.participant == participant) & (dataset.day == label_day)].to_numpy()[0]
            y.append(label)
            inst = []
            for day_offset in range(k-1,-1,-1):
                attribute_day = label_day - n - day_offset
                original_row = dataset[(dataset.participant == participant) & (dataset.day == attribute_day)].to_numpy()[0].tolist()
                if day_offset == 0 and n == 0:
                    inst.extend(original_row[3:])
                else:
                    inst.extend(original_row[2:])
            X.append(inst)
    return np.asarray(X),np.asarray(y)
```

```
def split(X, y, train_percent, seed, verbose=False):  # Split dataset
    X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, train_size=train_percent,
                                                        random_state=seed)  # creates test and train set
    if verbose:  # for diagnostics
        print('training set contains {} instances'.format(X_train.shape[0]))
        print('test set contains {} instances'.format(X_test.shape[0]))
        print('split complete')
    return X_test.shape[0], X_train, X_test, y_train, y_test

def ConfidenceInterval(acc, testset_size):  # generate 95% CI with Bonferroni Correction for 4 comparisons per dataset
    CI = 2.39 * sqrt((acc * (1 - acc)) / testset_size)
    return CI

############## CLASSIFICATION MODELS ################

#
# ASSUMPTIONS TEST: Naive Bayes
#

def NaiveBayes(k,N,testset_size, dataset, X_train, X_test, y_train, y_test):
    clf = GaussianNB()
    clf.fit(X_train, y_train)
    acc = clf.score(X_test, y_test)
    predicted = clf.predict(X_test)
    summary = metrics.classification_report(y_test, predicted)
    conmat = metrics.confusion_matrix(y_test, predicted)
    with open('results_NB_{}_{}.csv'.format(k,N), mode='w') as csvout:
        writer = csv.writer(csvout, delimiter=',')
        writer.writerows(conmat)
    CI = ConfidenceInterval(acc, testset_size)
    print('Naive Bayes: {} accuracy  95% CI : [{} , {}]'.format('%.3f' % acc, '%.3f' % (acc - CI), '%.3f' % (acc + CI)))
    print()
    print(summary)
    return acc

#
# BASELINE: Logistic Regression
#

def LR(k,N,testset_size, dataset, X_train, X_test, y_train, y_test,seed):
    clf = LogisticRegression(random_state=seed)
    clf.fit(X_train, y_train)
    acc = clf.score(X_test, y_test)
    predicted = clf.predict(X_test)
    summary = metrics.classification_report(y_test, predicted)
    conmat = metrics.confusion_matrix(y_test, predicted)
    with open('results_LR_{}_{}.csv'.format(k,N), mode='w') as csvout:
        writer = csv.writer(csvout, delimiter=',')
        writer.writerows(conmat)
    CI = ConfidenceInterval(acc, testset_size)
    print('LR: {} accuracy  95% CI : [{} , {}]'.format('%.3f' % acc, '%.3f' % (acc - CI), '%.3f' % (acc + CI)))
    print()
    #debug = 1
        #if debug:
            #print(clf.intercept_)
            #print(clf.coef_[0].size)
            #print(clf.coef_)
            #print()
    print(summary)
    return acc

#
# Decision Tree
#

def DecisionTree(k,N,testset_size, dataset, X_train, X_test, y_train, y_test, seed):
```

```
    clf = DecisionTreeClassifier(random_state=seed)
    clf.fit(X_train, y_train)  # fit model to data
    acc = clf.score(X_test, y_test)
    predicted = clf.predict(X_test)
    summary = metrics.classification_report(y_test, predicted)
    conmat = metrics.confusion_matrix(y_test, predicted)
    with open('results_DT_{}_{}.csv'.format(k,N), mode='w') as csvout:
        writer = csv.writer(csvout, delimiter=',')
        writer.writerows(conmat)
    CI = ConfidenceInterval(acc, testset_size)
    print(
        'Decision Tree: {} accuracy  95% CI : [{} , {}]'.format('%.3f' % acc, '%.3f' % (acc - CI), '%.3f' % (acc + CI)))
    print()
    print(summary)
    return acc

#
# Random Forest
#

def RandomForest(k,N,testset_size, D, dataset, X_train, X_test, y_train, y_test, seed):
    clf = RandomForestClassifier(n_estimators=100, random_state=seed)
    clf.fit(X_train, y_train)  # fit model to data
    acc = clf.score(X_test, y_test)
    predicted = clf.predict(X_test)
    summary = metrics.classification_report(y_test, predicted)
    conmat = metrics.confusion_matrix(y_test, predicted)
    with open('results_Forest_{}_{}.csv'.format(k,N), mode='w') as csvout:
        writer = csv.writer(csvout, delimiter=',')
        writer.writerows(conmat)
    CI = ConfidenceInterval(acc, testset_size)
    print(
        'Random Forest: {} accuracy  95% CI : [{} , {}]'.format('%.3f' % acc, '%.3f' % (acc - CI), '%.3f' % (acc + CI)))
    if k == 1 and N == 0:
        importance = pd.DataFrame(clf.feature_importances_, index = D.columns[3:] ,
columns=['importance']).sort_values('importance', ascending=False)
        #debug = 1
        #if debug:
            #print(importance)
        importance.to_csv('importance_RF.csv')
        print()
    print(summary)
    return acc

#
# Neural Network
#

def shallowNN(k,N,testset_size, dataset, X_train, X_test, y_train, y_test, seed):
    clf = MLPClassifier(hidden_layer_sizes=(150,150),solver='adam', max_iter=5000, learning_rate_init=0.001,
random_state=seed)
    clf.fit(X_train, y_train)
    acc = clf.score(X_test, y_test)
    predicted = clf.predict(X_test)
    summary = metrics.classification_report(y_test, predicted)
    conmat = metrics.confusion_matrix(y_test, predicted)
    with open('results_NN_{}_{}.csv'.format(k,N), mode='w') as csvout:
        writer = csv.writer(csvout, delimiter=',')
        writer.writerows(conmat)
    CI = ConfidenceInterval(acc, testset_size)
    print('Shallow NN: {} accuracy  95% CI : [{} , {}]'.format('%.3f' % acc, '%.3f' % (acc - CI), '%.3f' % (acc + CI)))
    print()
    print(summary)
    return acc

############## REGRESSION MODELS ################
```

```
#
# BASELINE: Linear Regression
#

def LinReg(testset_size, dataset, X_train, X_test, y_train, y_test):
    reg = LinearRegression()
    reg.fit(X_train, y_train)
    R2 = reg.score(X_test, y_test)
    predicted = reg.predict(X_test)
    MSE = metrics.mean_squared_error(y_test, predicted)
    #CI = ConfidenceInterval(acc, testset_size)
    print('Linear Regression: R^2 = {} MSE = {}'.format('%.3f' % R2, '%.3f' % MSE))
    print()
    return R2, MSE

#
# Decision Tree Regressor
#

def DecisionTreereg(testset_size, dataset, X_train, X_test, y_train, y_test, seed):
    reg = DecisionTreeRegressor(random_state=seed)
    reg.fit(X_train, y_train)  # fit model to data
    R2 = reg.score(X_test, y_test)
    predicted = reg.predict(X_test)
    MSE = metrics.mean_squared_error(y_test, predicted)
    #CI = ConfidenceInterval(MSE, testset_size)
    print(
        'Decision Tree: R^2 = {} MSE = {}'.format('%.3f' % R2,'%.3f' % MSE))
    print()
    return R2, MSE

#
# Random Forest Regressor
#

def RandomForestreg(testset_size, dataset, X_train, X_test, y_train, y_test, seed):
    reg = RandomForestRegressor(n_estimators=100, random_state=seed)
    reg.fit(X_train, y_train)  # fit model to data
    R2 = reg.score(X_test, y_test)
    predicted = reg.predict(X_test)
    MSE = metrics.mean_squared_error(y_test, predicted)
    #CI = ConfidenceInterval(acc, testset_size)
    print(
        'Random Forest: R^2 = {} MSE = {}'.format('%.3f' % R2, '%.3f' % MSE))
    print()
    return R2, MSE

#
# Neural Network Regressor
#

def shallowNNreg(testset_size, dataset, X_train, X_test, y_train, y_test, seed):
    reg = MLPRegressor(solver='adam', max_iter=1000, alpha=1e-3, random_state=seed)
    reg.fit(X_train, y_train)
    R2 = reg.score(X_test, y_test)
    predicted = reg.predict(X_test)
    MSE = metrics.mean_squared_error(y_test, predicted)
    #CI = ConfidenceInterval(acc, testset_size)
    print('Shallow NN: R^2 = {} MSE = {}'.format('%.3f' % R2, '%.3f' % MSE))
    print()
    return R2, MSE

########### MAIN ############

def main():
```

```
seed = int(sys.argv[1])
LR_table = np.zeros((7,8))
NB_table = np.zeros((7,8))
DT_table = np.zeros((7,8))
RF_table = np.zeros((7,8))
NN_table = np.zeros((7,8))
# Linreg_R2_table = np.zeros((7,8))
# DTreg_R2_table = np.zeros((7,8))
# RFreg_R2_table = np.zeros((7,8))
# NNreg_R2_table = np.zeros((7,8))
# Linreg_MSE_table = np.zeros((7,8))
# DTreg_MSE_table = np.zeros((7,8))
# RFreg_MSE_table = np.zeros((7,8))
# NNreg_MSE_table = np.zeros((7,8))
Dataset = 'Honors'
dataset = readin('./HonorsData.csv')
for k in range(1,8):
    for N in range(8):
        X , y = instanceFormat(dataset,k,N)
        testset_size, X_train, X_test, y_train, y_test = split(X,y,0.85,seed)
        # print('K is {}, N is {}'.format(k,N))
        # R2_LR, MSE_LR = LinReg(testset_size, dataset, X_train, X_test, y_train, y_test)
        # Linreg_R2_table[k-1,N] = R2_LR
        # Linreg_MSE_table[k-1,N] = MSE_LR
        # print('K is {}, N is {}'.format(k,N))
        # R2_DT , MSE_DT =  DecisionTreereg(testset_size, Dataset, X_train, X_test, y_train, y_test, seed)
        # DTreg_R2_table[k-1,N] = R2_DT
        # DTreg_MSE_table[k-1,N] = MSE_DT
        # print('K is {}, N is {}'.format(k,N))
        # R2_RF , MSE_RF = RandomForestreg(testset_size, dataset, X_train, X_test, y_train, y_test, seed)
        # RFreg_R2_table[k-1,N] = R2_RF
        # RFreg_MSE_table[k-1,N] = MSE_RF
        # print('K is {}, N is {}'.format(k,N))
        # R2_NN, MSE_NN = shallowNNreg(testset_size, dataset, X_train, X_test, y_train, y_test, seed)
        # NNreg_R2_table[k-1,N] = R2_NN
        # NNreg_MSE_table[k-1,N] = MSE_NN

        print('K is {}, N is {}'.format(k,N))
        LR_acc = LR(k,N,testset_size, Dataset, X_train, X_test, y_train, y_test,seed)
        LR_table[k-1,N] = LR_acc
        print('K is {}, N is {}'.format(k,N))
        NB_acc = NaiveBayes(k,N,testset_size, Dataset, X_train, X_test, y_train, y_test)
        NB_table[k-1,N] = NB_acc
        print('K is {}, N is {}'.format(k,N))
        DT_acc = DecisionTree(k,N,testset_size, Dataset, X_train, X_test, y_train, y_test, seed)
        DT_table[k-1,N] = DT_acc
        print('K is {}, N is {}'.format(k,N))
        RF_acc = RandomForest(k,N,testset_size, dataset, Dataset, X_train, X_test, y_train, y_test, seed)
        RF_table[k-1,N] = RF_acc
        print('K is {}, N is {}'.format(k,N))
        NN_acc = shallowNN(k,N,testset_size, Dataset, X_train, X_test, y_train, y_test, seed)
        NN_table[k-1,N] = NN_acc
        print("------------------------------")

# np.savetxt('Linreg_R2.csv',Linreg_R2_table,delimiter=',',fmt='%f')
# np.savetxt('Linreg_MSE.csv',Linreg_MSE_table,delimiter=',',fmt='%f')
# np.savetxt('DTreg_R2.csv',DTreg_R2_table,delimiter=',',fmt='%f')
# np.savetxt('DTreg_MSE.csv',DTreg_MSE_table,delimiter=',',fmt='%f')
# np.savetxt('RFreg_R2.csv',RFreg_R2_table,delimiter=',',fmt='%f')
# np.savetxt('RFreg_MSE.csv',RFreg_MSE_table,delimiter=',',fmt='%f')
# np.savetxt('NNreg_R2.csv',NNreg_R2_table,delimiter=',',fmt='%f')
# np.savetxt('NNreg_MSE.csv',NNreg_MSE_table,delimiter=',',fmt='%f')

np.savetxt('LR.csv',LR_table,delimiter=',',fmt='%f')
np.savetxt('NB.csv',NB_table,delimiter=',',fmt='%f')
np.savetxt('DT.csv',DT_table,delimiter=',',fmt='%f')
```

```
    np.savetxt('RF.csv',RF_table,delimiter=',',fmt='%f')
    np.savetxt('NN.csv',NN_table,delimiter=',',fmt='%f')

if __name__ == '__main__':
    main()
```