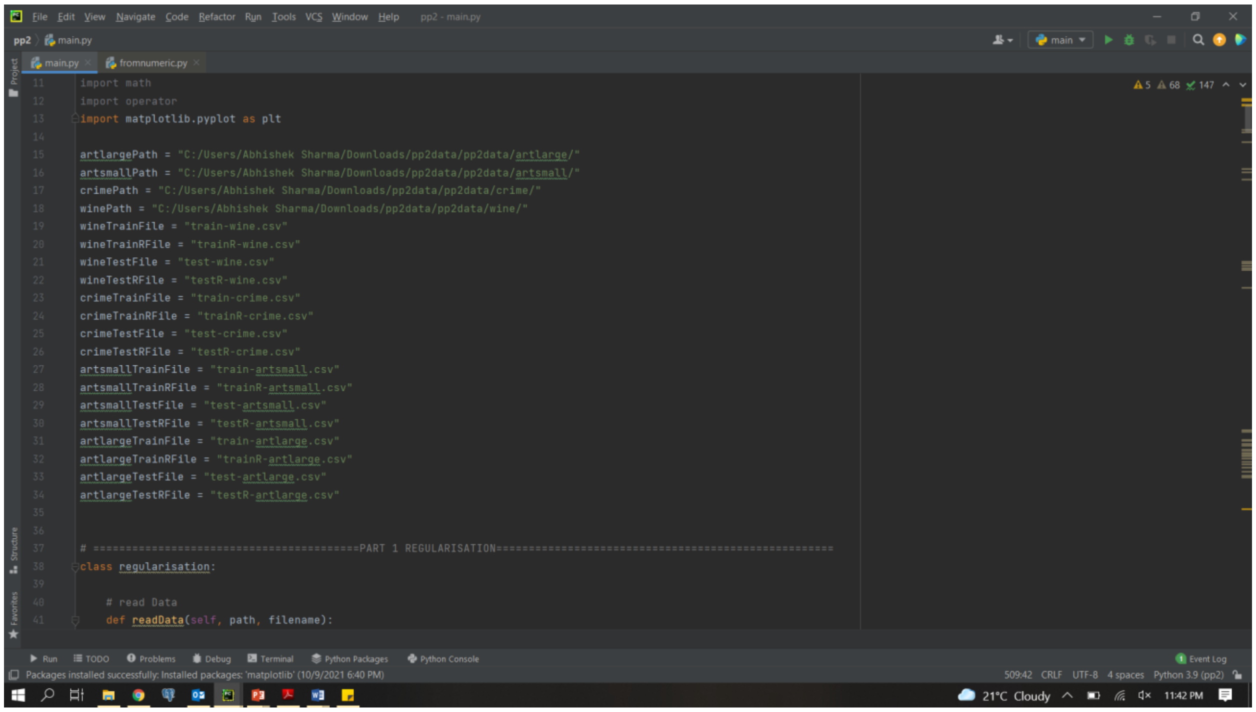
ReadMe:

Info about the code:

1. **The code is divided into three sections: Part 1: Regularisation, Part 2: Cross Fold and part 3: Bayesian Model Selection.**
2. **In each part, the code is further divided into 4 parts according to each of the 4 data sets.**
3. **Out of these 4 parts of code, one is left uncommented and the remaining three are commented.**

Steps to run the code:

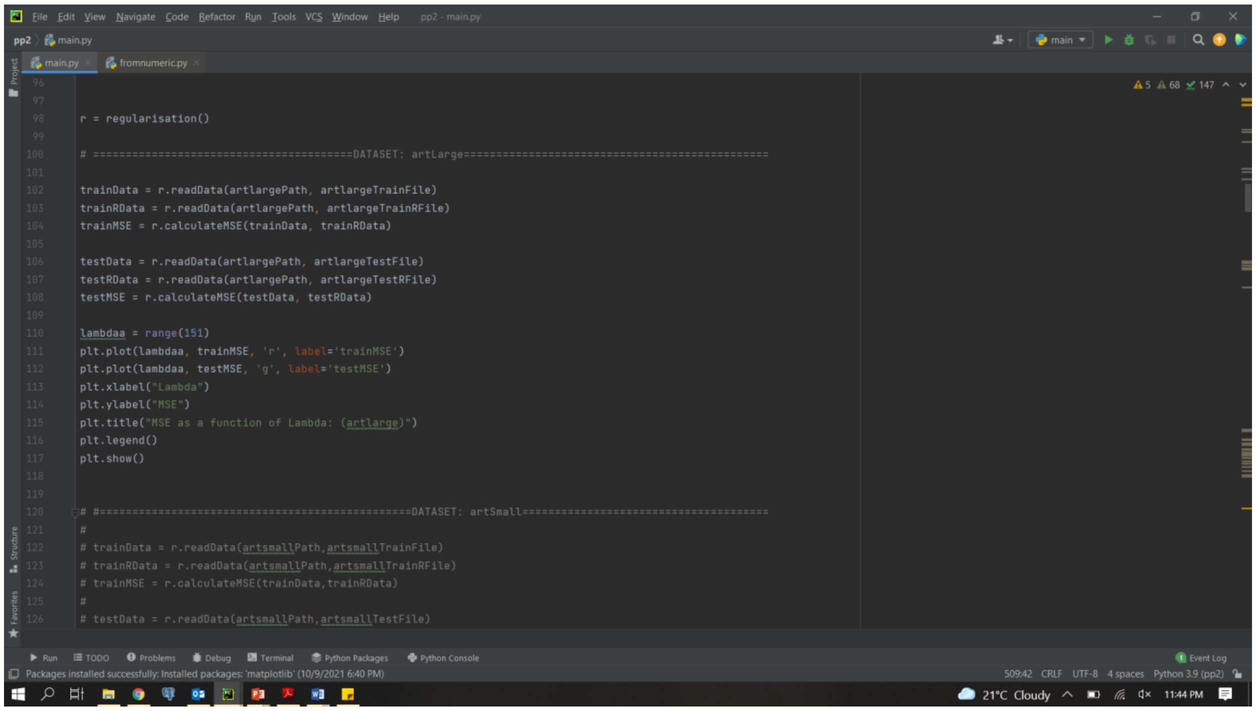
1. **Change the path values in the path variables according to the new location where the code will be run. The variables are divided into the base path and the respective file names of the datasets.**



```python
import math
import operator
import matplotlib.pyplot as plt

artlargePath = "C:/Users/Abhishek Sharma/Downloads/pp2data/pp2data/artlarge/"
artsmallPath = "C:/Users/Abhishek Sharma/Downloads/pp2data/pp2data/artsmall/"
crimePath = "C:/Users/Abhishek Sharma/Downloads/pp2data/pp2data/crime/"
winePath = "C:/Users/Abhishek Sharma/Downloads/pp2data/pp2data/wine/"
wineTrainFile = "train-wine.csv"
wineTrainRFile = "trainR-wine.csv"
wineTestFile = "test-wine.csv"
wineTestRFile = "testR-wine.csv"
crimeTrainFile = "train-crime.csv"
crimeTrainRFile = "trainR-crime.csv"
crimeTestFile = "test-crime.csv"
crimeTestRFile = "testR-crime.csv"
artsmallTrainFile = "train-artsmall.csv"
artsmallTrainRFile = "trainR-artsmall.csv"
artsmallTestFile = "test-artsmall.csv"
artsmallTestRFile = "testR-artsmall.csv"
artlargeTrainFile = "train-artlarge.csv"
artlargeTrainRFile = "trainR-artlarge.csv"
artlargeTestFile = "test-artlarge.csv"
artlargeTestRFile = "testR-artlarge.csv"


# ========================================PART 1 REGULARISATION====================================================
class regularisation:

    # read Data
    def readData(self, path, filename):
```

```
Packages installed successfully: Installed packages: 'matplotlib' (10/9/2021 6:40 PM)
```

2. **Move to the part which you want to execute and uncomment the code for that particular dataset that you want to test the code on.**



```python


r = regularisation()

# =========================================DATASET: artLarge==============================================

trainData = r.readData(artlargePath, artlargeTrainFile)
trainRData = r.readData(artlargePath, artlargeTrainRFile)
trainMSE = r.calculateMSE(trainData, trainRData)

testData = r.readData(artlargePath, artlargeTestFile)
testRData = r.readData(artlargePath, artlargeTestRFile)
testMSE = r.calculateMSE(testData, testRData)

lambdaa = range(151)
plt.plot(lambdaa, trainMSE, 'r', label='trainMSE')
plt.plot(lambdaa, testMSE, 'g', label='testMSE')
plt.xlabel("Lambda")
plt.ylabel("MSE")
plt.title("MSE as a function of Lambda: (artlarge)")
plt.legend()
plt.show()


# #=============================================DATASET: artSmall=====================================
#
# trainData = r.readData(artsmallPath,artsmallTrainFile)
# trainRData = r.readData(artsmallPath,artsmallTrainRFile)
# trainMSE = r.calculateMSE(trainData,trainRData)
#
# testData = r.readData(artsmallPath,artsmallTestFile)
```

```
Packages installed successfully: Installed packages: 'matplotlib' (10/9/2021 6:40 PM)
```

File Edit View Navigate Code Refactor Run Tools VCS Window Help pp2 - main.py
pp2 main.py
main.py
fromnumeric.py
Run TODO Problems Debug Terminal Python Packages Python Console
Event Log
509:42 CRLF UTF-8 4 spaces Python 3.9 (pp2)
21°C Cloudy 11:44 PM

3. Run the code.

Programming Assignment 2 Report

<u>Part 1: REGULARISATION</u>

The plots of train and test MSE as a function of Lambda for different datasets are given below.

ARTLARGE:

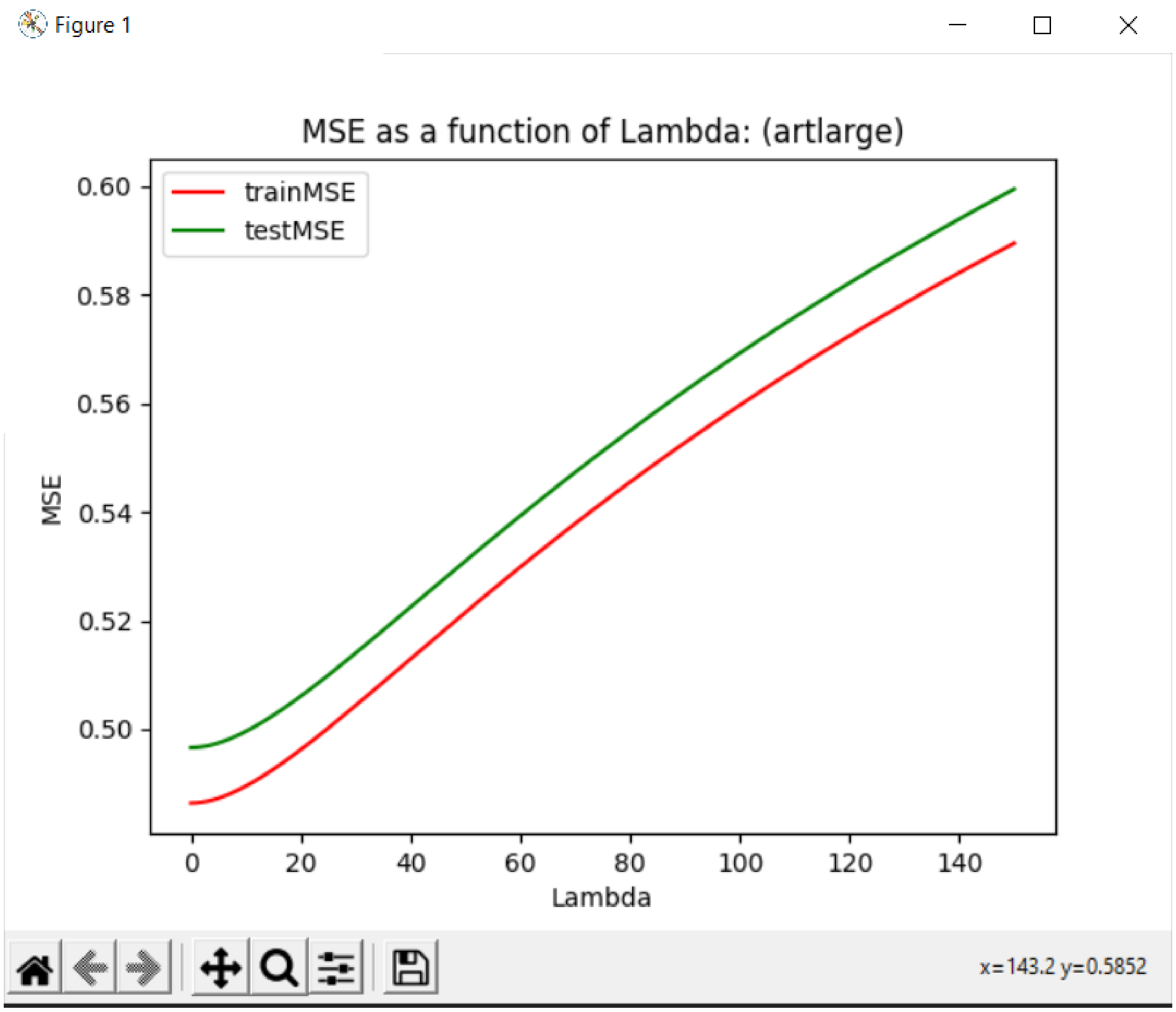


Figure 1
MSE as a function of Lambda: (artlarge)
trainMSE
testMSE
MSE
Lambda
0.60
0.58
0.56
0.54
0.52
0.50
0
20
40
60
80
100
120
140
x=143.2 y=0.5852

ARTSMALL:

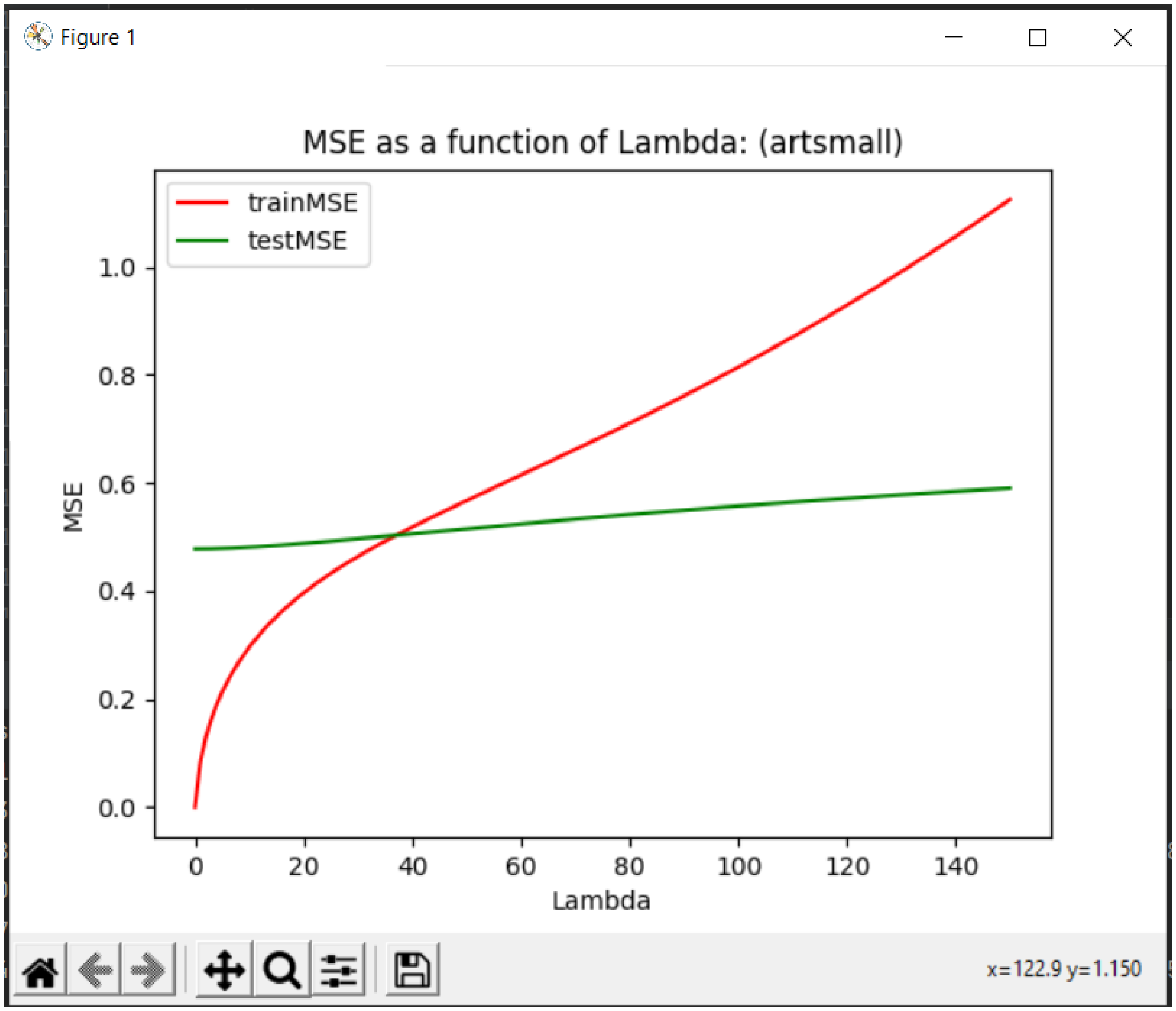


Figure 1
MSE as a function of Lambda: (artsmall)
trainMSE
testMSE
MSE
Lambda
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
120
140
x=122.9 y=1.150

CRIME:

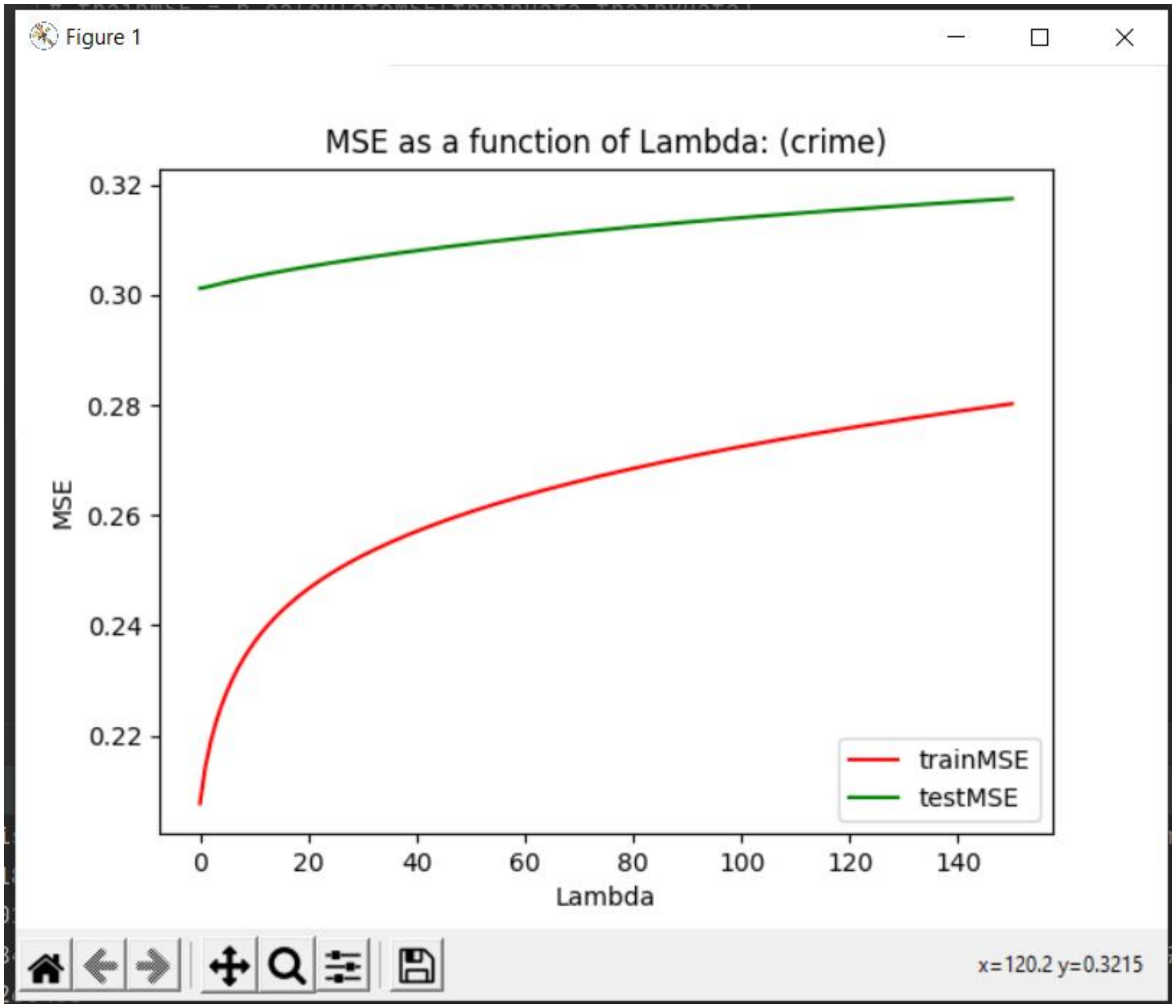
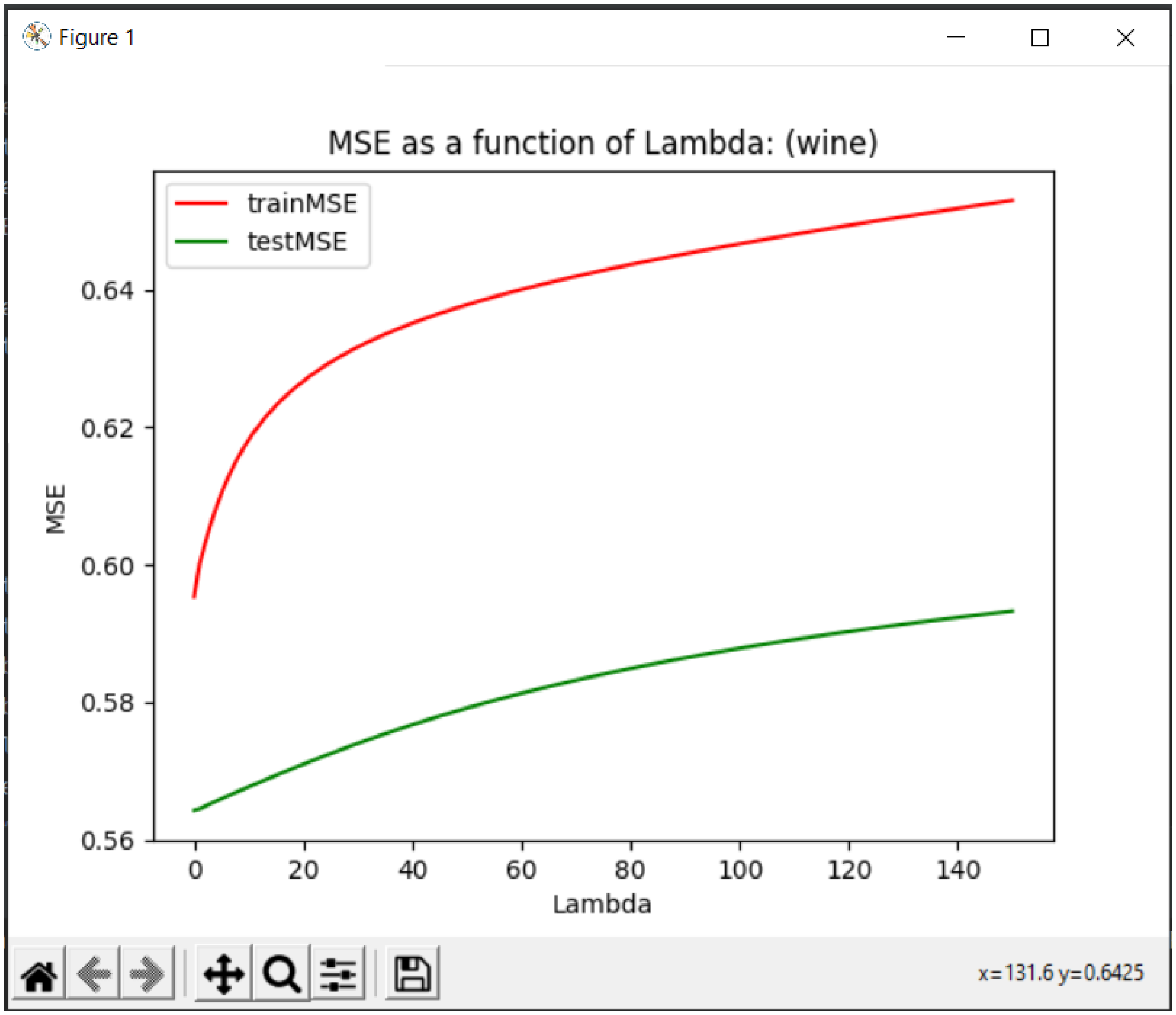


Figure 1
MSE as a function of Lambda: (crime)
0.32
0.30
0.28
0.26
0.24
0.22
MSE
0
20
40
60
80
100
120
140
Lambda
trainMSE
testMSE
x=120.2 y=0.3215

WINE:



Discussions:

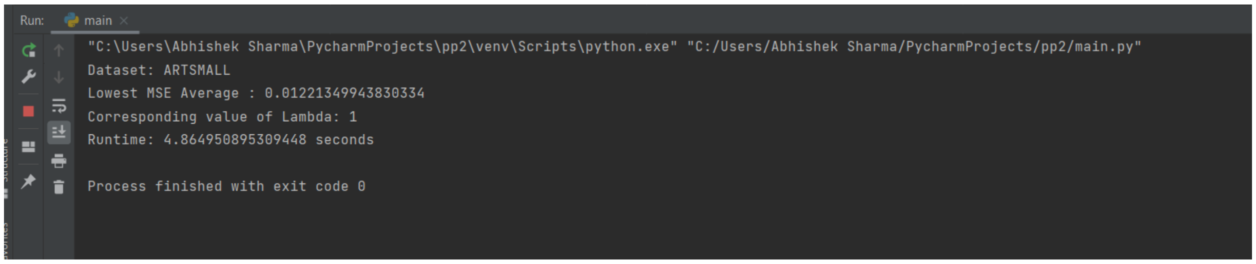
1. Why can’t the training set MSE be used to select λ?
   As can be seen in the plots above, the test set MSE is higher than the training set MSE . This means that the test set data is being Over-fitted which causes the MSE to increase. Therefore the training set MSE should not be used to select λ since it will cause over-fitting in the dataset.

2. How does λ affect error on the test set?
   The test set MSE for each dataset increases on increasing the value of λ, but as can be seen, it doesn’t vary as vastly as the MSE of the train set. Thus, the effects of regularisation are evident.

3. Does this differ for different datasets?
   Yes, the range of MSE is different for different datasets.

PART 2: Model Selection using Cross Validation

DATASET: ARTSMALL

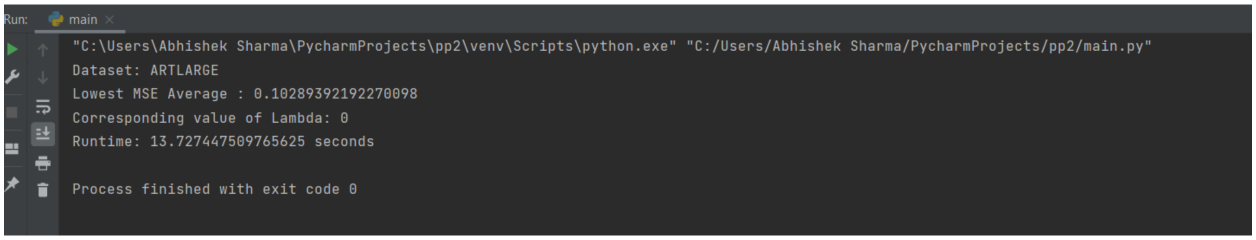


```
"C:\Users\Abhishek Sharma\PycharmProjects\pp2\venv\Scripts\python.exe" "C:/Users/Abhishek Sharma/PycharmProjects/pp2/main.py"
Dataset: ARTSMALL
Lowest MSE Average : 0.01221349943830334
Corresponding value of Lambda: 1
Runtime: 4.864950895309448 seconds

Process finished with exit code 0
```

Lowest MSE: 0.012

Corresponding lamda value: 1

DATASET: ARTLARGE

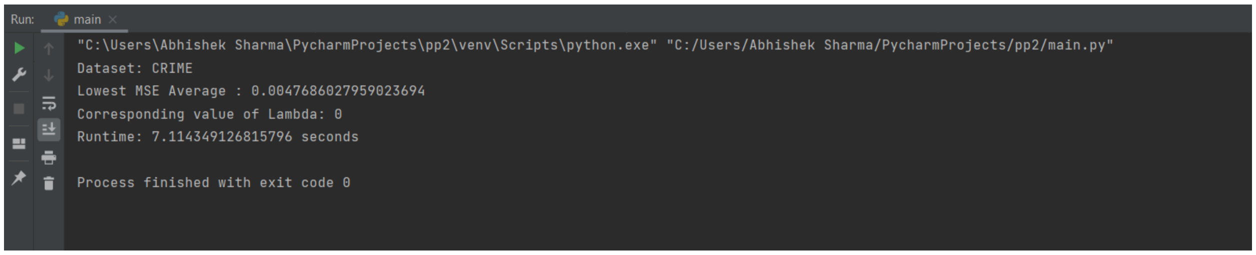


```
"C:\Users\Abhishek Sharma\PycharmProjects\pp2\venv\Scripts\python.exe" "C:/Users/Abhishek Sharma/PycharmProjects/pp2/main.py"
Dataset: ARTLARGE
Lowest MSE Average : 0.10289392192270098
Corresponding value of Lambda: 0
Runtime: 13.727447509765625 seconds

Process finished with exit code 0
```

Lowest MSE: 0.102

Corresponding lamda value: 0

DATASET: CRIME

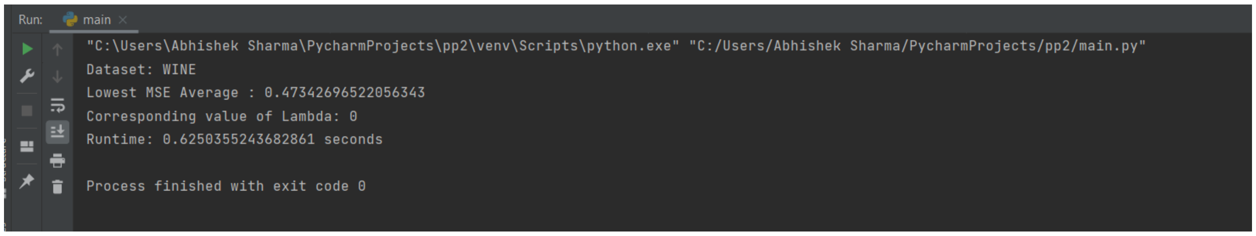


```
"C:\Users\Abhishek Sharma\PycharmProjects\pp2\venv\Scripts\python.exe" "C:/Users/Abhishek Sharma/PycharmProjects/pp2/main.py"
Dataset: CRIME
Lowest MSE Average : 0.0047686027959023694
Corresponding value of Lambda: 0
Runtime: 7.114349126815796 seconds

Process finished with exit code 0
```

Lowest MSE: 0.004

Corresponding lamda value: 0

DATASET: WINE



```
"C:\Users\Abhishek Sharma\PycharmProjects\pp2\venv\Scripts\python.exe" "C:/Users/Abhishek Sharma/PycharmProjects/pp2/main.py"
Dataset: WINE
Lowest MSE Average : 0.47342696522056343
Corresponding value of Lambda: 0
Runtime: 0.6250355243682861 seconds

Process finished with exit code 0
```

Lowest MSE: 0.47

Corresponding lamda value: 0

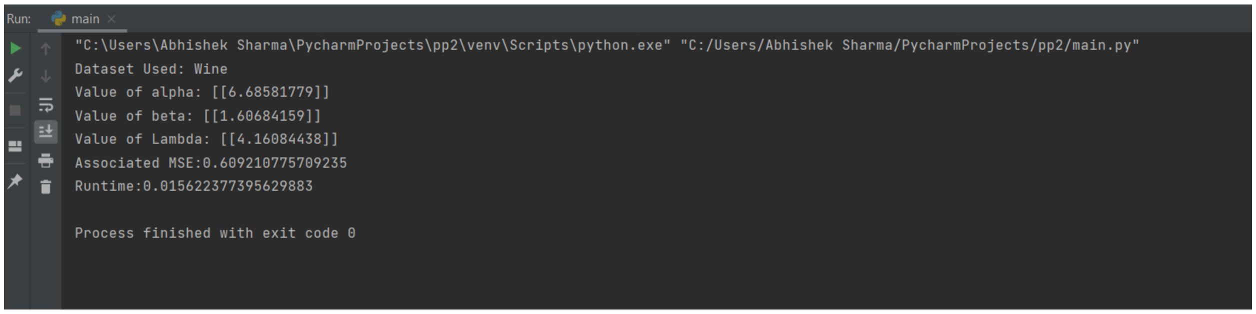
Observations:

The MSE in case of cross validation increases as the value of lambda increases. The lower the value of lambda, the lower the MSE.
However, the runtime increases a lot in case of cross validation. The bigger the dataset size, the runtime is bigger.

Compared to the values of lambda and associated MSE in part 1, the values of lambda and MSE both are lower in case of cross validation.

<u>Part 3: BAYESIAN MODEL SELECTION</u>

WINE:

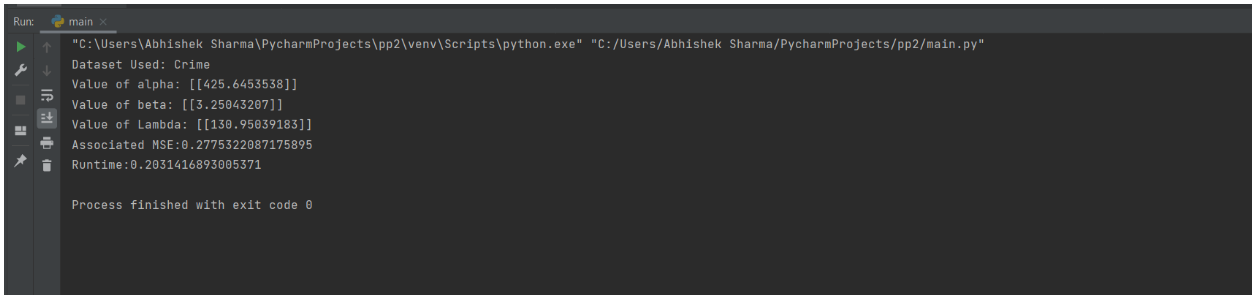


```
"C:\Users\Abhishek Sharma\PycharmProjects\pp2\venv\Scripts\python.exe" "C:/Users/Abhishek Sharma/PycharmProjects/pp2/main.py"
Dataset Used: Wine
Value of alpha: [[6.68581779]]
Value of beta: [[1.60684159]]
Value of Lambda: [[4.16084438]]
Associated MSE:0.609210775709235
Runtime:0.015622377395629883

Process finished with exit code 0
```

The observations are as follows:

For Wine Dataset, the value of alpha : 6.68, beta = 1.60, optimal λ = 4.16, Associated MSE = 0.6 and the runtime is 0.15 seconds.

CRIME:

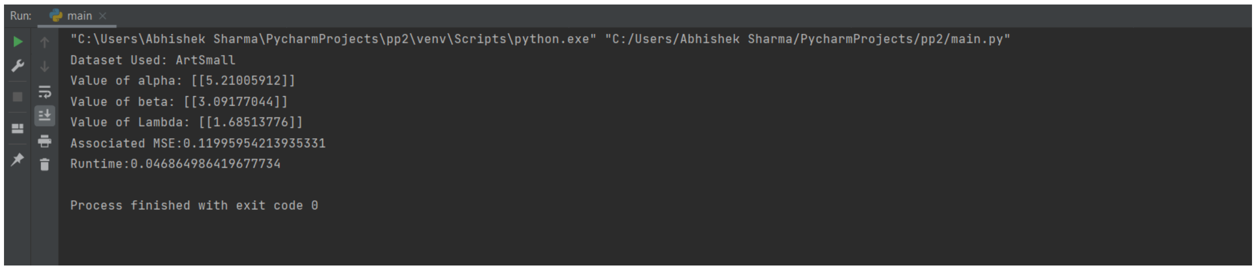


```
"C:\Users\Abhishek Sharma\PycharmProjects\pp2\venv\Scripts\python.exe" "C:/Users/Abhishek Sharma/PycharmProjects/pp2/main.py"
Dataset Used: Crime
Value of alpha: [[425.6453538]]
Value of beta: [[3.25043207]]
Value of Lambda: [[130.95039183]]
Associated MSE:0.2775322087175895
Runtime:0.2031416893005371

Process finished with exit code 0
```

The observations are as follows:

For Crime Dataset, the value of alpha : 425.6, beta = 3.25, optimal λ = 130.9, Associated MSE = 0.27 and the runtime is 0.20 seconds.

ARTSMALL:

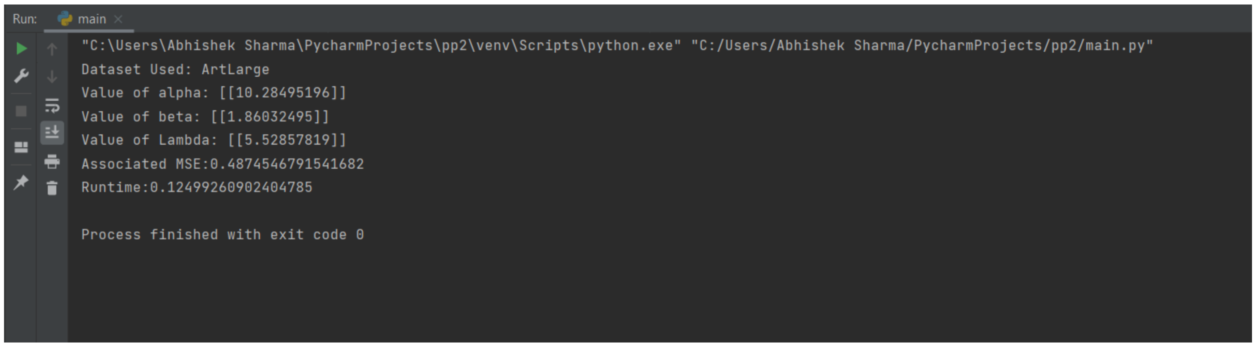


```
"C:\Users\Abhishek Sharma\PycharmProjects\pp2\venv\Scripts\python.exe" "C:/Users/Abhishek Sharma/PycharmProjects/pp2/main.py"
Dataset Used: ArtSmall
Value of alpha: [[5.21005912]]
Value of beta: [[3.09177044]]
Value of Lambda: [[1.68513776]]
Associated MSE:0.11995954213935331
Runtime:0.046864986419677734

Process finished with exit code 0
```

The observations are as follows:

For artsmall Dataset, the value of alpha : 5.21, beta = 3.09, optimal λ = 1.68, Associated MSE = 0.11 and the runtime is 0.04 seconds.

ARTLARGE:



```
"C:\Users\Abhishek Sharma\PycharmProjects\pp2\venv\Scripts\python.exe" "C:/Users/Abhishek Sharma/PycharmProjects/pp2/main.py"
Dataset Used: ArtLarge
Value of alpha: [[10.28495196]]
Value of beta: [[1.86032495]]
Value of Lambda: [[5.52857819]]
Associated MSE:0.4874546791541682
Runtime:0.12499260902404785

Process finished with exit code 0
```

The observations are as follows:

For artlarge Dataset, the value of alpha : 10.28, beta = 1.86, optimal λ = 5.5, Associated MSE = 0.48 and the runtime is 0.12 seconds.

PART 4: DISCUSSION OF RESULTS

Q1. How do the two model selection methods compare in terms of effective λ, test set MSE and run time?

The two models vary with each other in terms of lambda, test set MSE and run time as follows:

In terms of lambda:

Values of lambda in cross fold model remain low compared to other model.
However, lambda has higher values in Bayesian Model Selection.

In terms of MSE:

Test set MSE values are also lower for cross fold method as compared to those in the Bayesian Model Selection.

This shows that cross fold method is better at prevention of the issue of over fitting as is evident from the values of MSE from both the models.

In terms of Runtime:

The runtime of cross fold model goes a lot higher than that in the Bayesian model. This is because the whole process of cross fold validation, though results in lower values of MSE, take large amount of time.

Q2. Do the results suggest conditions where one method is preferable to the other?

The results from this assignment suggest that both the models, namely the cross fold validation model and the Bayesian model selection give low MSE and lambda values when tested on test data. (The values being lower in the cross fold model).

The difference, however lies in the runtime of both the models. For large datasets, the cross fold model can take up large amounts of time which may sometimes result in huge costs. The runtime of the Bayesian Model selection stays below a few mili seconds for the given datasets and therefore results in faster results.

So, we can say that in conditions where getting results FAST is the priority, the Bayesian Model selection may be the better choice.

However, if the priority is to get as accurate results as possible, with minimum MSEs, then the better option would be to go for Cross fold Validation model.

CODE:

```
# import necessary libraries
import csv
import time
from random import randrange
import pandas as pd

import matplotlib as matplotlib
import numpy as np
import statistics

import math
import operator
import matplotlib.pyplot as plt

artlargePath = "C:/Users/Abhishek Sharma/Downloads/pp2data/pp2data/artlarge/"
artsmallPath = "C:/Users/Abhishek Sharma/Downloads/pp2data/pp2data/artsmall/"
crimePath = "C:/Users/Abhishek Sharma/Downloads/pp2data/pp2data/crime/"
winePath = "C:/Users/Abhishek Sharma/Downloads/pp2data/pp2data/wine/"
wineTrainFile = "train-wine.csv"
wineTrainRFile = "trainR-wine.csv"
wineTestFile = "test-wine.csv"
wineTestRFile = "testR-wine.csv"
crimeTrainFile = "train-crime.csv"
crimeTrainRFile = "trainR-crime.csv"
crimeTestFile = "test-crime.csv"
crimeTestRFile = "testR-crime.csv"
artsmallTrainFile = "train-artsmall.csv"
artsmallTrainRFile = "trainR-artsmall.csv"
artsmallTestFile = "test-artsmall.csv"
artsmallTestRFile = "testR-artsmall.csv"
artlargeTrainFile = "train-artlarge.csv"
artlargeTrainRFile = "trainR-artlarge.csv"
```

```
artlargeTestFile = "test-artlarge.csv"
artlargeTestRFile = "testR-artlarge.csv"


# =========================================PART 1
REGULARISATION=====================================================
class regularisation:

    # read Data
    def readData(self, path, filename):
        file = open(path + filename)
        csvReader = csv.reader(file)
        data = []
        for row in csvReader:
            data.append(row)
        file.close()
        return data

    # function to calculate MSE
    def calculateMSE(self, DataArray, rDataArray):
        # Train Data Array (phi)
        trainDataArray = np.array(DataArray)  # creates a np array of the
data set

        # print('phi shape: ' + str(trainDataArray.shape))
        trainDataArray = trainDataArray.astype(
            np.float64)  # converting elements into type float for matrix
multiplication

        # Data Array Transposed (phi transpose)
        trainDataArrayTranspose = np.array(trainDataArray.T)
        # print('phi transpose shape: ' + str(trainDataArrayTranspose.shape))
        trainDataArrayTranspose = trainDataArrayTranspose.astype(
            np.float64)  # converting elements into type float for matrix
multiplication

        # TrainR Data Array (t)
        trainRDataArray = np.array(rDataArray)
        # print('t shape: ' + str(trainRDataArray.shape))
        trainRDataArray = trainRDataArray.astype(
            np.float64)  # converting elements into type float for matrix
multiplication

        # (phi transpose * t)
        phi_t = trainDataArrayTranspose @ trainRDataArray  # '@' = matrix
mult
        # print('phi_t shape: ' + str(phi_t.shape))

        # # matrix Multiplication (phi * phi transpose)
        matrixMultiplyArray = trainDataArrayTranspose @ trainDataArray
        # print('matrixMultiplyArray shape: ' +
str(matrixMultiplyArray.shape))

        eyeSize = len(matrixMultiplyArray)  # to calculate the size of the
identity matrix
        I = np.eye(eyeSize)  # Identity Matrix
        # print('I shape: ' + str(I.shape))
        W = []
        for l in range(151):  # l =lambda Calculating w for each lambda and
appending to List W
            wMatrix = I * l + matrixMultiplyArray
            wMatrixInverse = np.linalg.inv(wMatrix)
            w = wMatrixInverse @ phi_t
            W.append(w)

        # calculate MSE

        MSE = []
        for w in W:
            SE = ((trainRDataArray - trainDataArray @ w) ** 2)  # SE: square
error
            MSE.append(np.mean(SE))  # Mean of square Error
        return MSE


r = regularisation()

# ========================================DATASET:
artLarge=======================================================

trainData = r.readData(artlargePath, artlargeTrainFile)
trainRData = r.readData(artlargePath, artlargeTrainRFile)
trainMSE = r.calculateMSE(trainData, trainRData)

testData = r.readData(artlargePath, artlargeTestFile)
testRData = r.readData(artlargePath, artlargeTestRFile)
testMSE = r.calculateMSE(testData, testRData)

lambdaa = range(151)
plt.plot(lambdaa, trainMSE, 'r', label='trainMSE')
plt.plot(lambdaa, testMSE, 'g', label='testMSE')
plt.xlabel("Lambda")
plt.ylabel("MSE")
plt.title("MSE as a function of Lambda: (artlarge)")
plt.legend()
plt.show()


# #===================================================DATASET:
artSmall==============================================
#
# trainData = r.readData(artsmallPath,artsmallTrainFile)
# trainRData = r.readData(artsmallPath,artsmallTrainRFile)
# trainMSE = r.calculateMSE(trainData,trainRData)
#
# testData = r.readData(artsmallPath,artsmallTestFile)
# testRData = r.readData(artsmallPath,artsmallTestRFile)
# testMSE = r.calculateMSE(testData,testRData)
#
# lambdaa = range(151)
# plt.plot(lambdaa,trainMSE,'r',label='trainMSE')
# plt.plot(lambdaa,testMSE,'g',label = 'testMSE')
# plt.xlabel("Lambda")
# plt.ylabel("MSE")
# plt.title("MSE as a function of Lambda: (artsmall)")
# plt.legend()
# plt.show()
#
# # ============================================DATASET:
crime============================================
#
# trainData = r.readData(crimePath,crimeTrainFile)
# trainRData = r.readData(crimePath,crimeTrainRFile)
# trainMSE = r.calculateMSE(trainData,trainRData)
#
# testData = r.readData(crimePath,crimeTestFile)
# testRData = r.readData(crimePath,crimeTestRFile)
# testMSE = r.calculateMSE(testData,testRData)
#
# lambdaa = range(151)
# plt.plot(lambdaa,trainMSE,'r',label='trainMSE')
# plt.plot(lambdaa,testMSE,'g',label = 'testMSE')
# plt.xlabel("Lambda")
# plt.ylabel("MSE")
# plt.title("MSE as a function of Lambda: (crime)")
# plt.legend()
# plt.show()
#
#
# # ============================================DATASET:
wine======================================
#
# trainData = r.readData(winePath,wineTrainFile)
# trainRData = r.readData(winePath,wineTrainRFile)
# trainMSE = r.calculateMSE(trainData,trainRData)
#
# testData = r.readData(winePath,wineTestFile)
# testRData = r.readData(winePath,wineTestRFile)
# testMSE = r.calculateMSE(testData,testRData)
#
# lambdaa = range(151)
# plt.plot(lambdaa,trainMSE,'r',label='trainMSE')
# plt.plot(lambdaa,testMSE,'g',label = 'testMSE')
# plt.xlabel("Lambda")
# plt.ylabel("MSE")
# plt.title("MSE as a function of Lambda: (wine)")
# plt.legend()
# plt.show()


## ==============================================PART 2 K-
fold=========================================================

class kFold:

    def cross_validation_split(self, dataset, folds):
        dataset_split = list()
        dataset_copy = list(dataset)
        fold_size = int(len(dataset) / folds)
        for i in range(folds):
            fold = list()
            while len(fold) < fold_size:
                index = randrange(len(dataset_copy))
                fold.append(dataset_copy.pop(index))
            dataset_split.append(fold)
        return dataset_split

    # read Data
    def readData(self, path, filename):
        file = open(path + filename)
        csvReader = csv.reader(file)
        data = []
        for row in csvReader:
            data.append(row)
        file.close()
        return data

    def calculateMSE(self, DataArray, rDataArray, l):
        # Train Data Array (phi)
        trainDataArray = np.array(DataArray)

        # print('phi shape: ' + str(trainDataArray.shape))
        trainDataArray = trainDataArray.astype(
            np.float64)  # converting elements into type float for matrix
multiplication

        # Data Array Transposed (phi transpose)
        trainDataArrayTranspose = np.array(trainDataArray.T)
        # print('phi transpose shape: ' + str(trainDataArrayTranspose.shape))
        trainDataArrayTranspose = trainDataArrayTranspose.astype(
            np.float64)  # converting elements into type float for matrix
multiplication

        # TrainR Data Array (t)
        trainRDataArray = np.array(rDataArray)
        # print('t shape: ' + str(trainRDataArray.shape))
        trainRDataArray = trainRDataArray.astype(
            np.float64)  # converting elements into type float for matrix
multiplication

        # (phi transpose * t)
        phi_t = trainDataArrayTranspose @ trainRDataArray  # '@' = matrix
mult
        # print('phi_t shape: ' + str(phi_t.shape))

        # # matrix Multiplication (phi * phi trans)
        matrixMultiplyArray = trainDataArrayTranspose @ trainDataArray
        # print('matrixMultiplyArray shape: ' +
str(matrixMultiplyArray.shape))

        eyeSize = len(matrixMultiplyArray)
        I = np.eye(eyeSize)  # Identity Matrix
        # print('I shape: ' + str(I.shape))
        W = []

        wMatrix = I * (l + 1) + matrixMultiplyArray
        wMatrixInverse = np.linalg.inv(wMatrix)
        w = wMatrixInverse @ phi_t
        # calculate SE
        SE = ((trainRDataArray - trainDataArray @ w) ** 2)  # square error
        MSE = (np.mean(SE))

        return MSE


k = kFold()

start = time.time()

# ============================DATASET:
ARTSMALL===========================================

totalTrainData = np.array(k.readData(artsmallPath, artsmallTrainFile))
splittedTrainData = np.array_split(totalTrainData, 10)

totalTrainRData = np.array(k.readData(artsmallPath, artsmallTrainRFile))
splittedTrainRData = np.array_split(totalTrainRData, 10)

MSEAverageList = []
lambdaList = []

# loop for splitting the dataset and calculating MSE
for l in range(150):
    MSEList = []
    for i in range(10):
        train = splittedTrainData[i]
        test = splittedTrainData[-i]

        trainR = splittedTrainRData[i]
        testR = splittedTrainRData[-i]

        MSE = k.calculateMSE(train, trainR, l)
        MSEList.append(MSE)

    MSEAverage = statistics.mean(MSEList)
    MSEAverageList.append(MSEAverage)
    lambdaList.append(l)

# now find the lowest MSE Average and its corresponding lambda
lowestMSEAverage = min(MSEAverageList)
index = MSEAverageList.index(lowestMSEAverage)
lowestLambda = lambdaList[index]

end = time.time()

print("Dataset: ARTSMALL")
print("Lowest MSE Average : " + str(lowestMSEAverage))
print("Corresponding value of Lambda: " + str(lowestLambda))
print("Runtime: " + str(end - start) + " seconds")


# #============================DATASET:
ARTLARGE===========================================
#
```

```
# totalTrainData = np.array(k.readData(artlargePath, artlargeTrainFile))
# splittedTrainData = np.array_split(totalTrainData, 10)
#
# totalTrainRData = np.array(k.readData(artlargePath, artlargeTrainRFile))
# splittedTrainRData = np.array_split(totalTrainRData, 10)
#
# MSEAverageList = []
# lambdaList = []
#
# # loop for splitting the dataset and calculating MSE
# for l in range(150):
#     MSEList = []
#     for i in range(10):
#         train = splittedTrainData[i]
#         test = splittedTrainData[-i]
#
#         trainR = splittedTrainRData[i]
#         testR = splittedTrainRData[-i]
#
#         MSE = k.calculateMSE(train, trainR, l)
#         MSEList.append(MSE)
#
#     MSEAverage = statistics.mean(MSEList)
#     MSEAverageList.append(MSEAverage)
#     lambdaList.append(l)
#
# # now find the lowest MSE Average and its corresponding lambda
# lowestMSEAverage = min(MSEAverageList)
# index = MSEAverageList.index(lowestMSEAverage)
# lowestLambda = lambdaList[index]
#
# end = time.time()
#
# print("Dataset: ARTLARGE")
# print("Lowest MSE Average : " + str(lowestMSEAverage))
# print("Corresponding value of Lambda: " + str(lowestLambda))
# print("Runtime: "+str(end-start)+" seconds")

# #=====================================DATASET:
WINE============================================================
#
# totalTrainData = np.array(k.readData(winePath, wineTrainFile))
# splittedTrainData = np.array_split(totalTrainData, 10)
#
# totalTrainRData = np.array(k.readData(winePath, wineTrainRFile))
# splittedTrainRData = np.array_split(totalTrainRData, 10)
#
# MSEAverageList = []
# lambdaList = []
#
# # loop for splitting the dataset and calculating MSE
# for l in range(150):
#     MSEList = []
#     for i in range(10):
#         train = splittedTrainData[i]
#         test = splittedTrainData[-i]
#
```

```
#          trainR = splittedTrainRData[i]
#          testR = splittedTrainRData[-i]
#
#          MSE = k.calculateMSE(train, trainR, l)
#          MSEList.append(MSE)
#
#      MSEAverage = statistics.mean(MSEList)
#      MSEAverageList.append(MSEAverage)
#      lambdaList.append(l)
#
# # now find the lowest MSE Average and its corresponding lambda
# lowestMSEAverage = min(MSEAverageList)
# index = MSEAverageList.index(lowestMSEAverage)
# lowestLambda = lambdaList[index]
#
# end = time.time()
#
# print("Dataset: WINE")
# print("Lowest MSE Average : " + str(lowestMSEAverage))
# print("Corresponding value of Lambda: " + str(lowestLambda))
# print("Runtime: "+str(end-start)+" seconds")
#
# #=======================================DATASET:
CRIME=========================================================
#
# totalTrainData = np.array(k.readData(crimePath, crimeTrainFile))
# splittedTrainData = np.array_split(totalTrainData, 10)
#
# totalTrainRData = np.array(k.readData(crimePath, crimeTrainRFile))
# splittedTrainRData = np.array_split(totalTrainRData, 10)
#
# MSEAverageList = []
# lambdaList = []
#
# # loop for splitting the dataset and calculating MSE
# for l in range(150):
#      MSEList = []
#      for i in range(10):
#          train = splittedTrainData[i]
#          test = splittedTrainData[-i]
#
#          trainR = splittedTrainRData[i]
#          testR = splittedTrainRData[-i]
#
#          MSE = k.calculateMSE(train, trainR, l)
#          MSEList.append(MSE)
#
#      MSEAverage = statistics.mean(MSEList)
#      MSEAverageList.append(MSEAverage)
#      lambdaList.append(l)
#
# # now find the lowest MSE Average and its corresponding lambda
# lowestMSEAverage = min(MSEAverageList)
# index = MSEAverageList.index(lowestMSEAverage)
# lowestLambda = lambdaList[index]
#
# end = time.time()
```

```
#
# print("Dataset: CRIME")
# print("Lowest MSE Average : " + str(lowestMSEAverage))
# print("Corresponding value of Lambda: " + str(lowestLambda))
# print("Runtime: "+str(end-start)+" seconds")


# ==============================================PART
3============================================================

class bayes:
    # read Data
    def readData(self, path, filename):
        file = open(path + filename)
        csvReader = csv.reader(file)
        data = []
        for row in csvReader:
            data.append(row)
        file.close()
        return data

    def calculateBphiTphi(self, beta, phiTphi):  # Computing Beta *
(PhiTranspose * Phi)
        BphiTphi = beta * phiTphi
        return BphiTphi

    def calculateGamma(self, BphiTphi, alpha):  # Computing Gamma for
calculating alpha and beta
        eig = np.linalg.eigvals(BphiTphi)
        gamma = sum([e / (e + alpha) for e in eig])

        return gamma

    def calculateMn(self, B, Sn, phiT_t):  # Computing Mn
        Mn = B * (Sn @ phiT_t)
        return Mn

    def calculateSn(self, alpha, beta, phi, phiTphi):  # Computing Sn
        I = np.identity(np.shape(phi)[1])
        Sn_Inverse = alpha * I + beta * phiTphi
        Sn = np.linalg.inv(Sn_Inverse)
        return Sn

    def calculateAlpha(self, gamma, mn):  # Computing Alpha
        mnT = mn.T
        mnTmn = mnT @ mn
        alpha = gamma / mnTmn
        return alpha

    def calculateBeta(self, phi, t, Mn, gamma):  # Computing Beta
        beta_row_list = []
        N = np.shape(phi)[0]
        for i in range(0, phi.shape[0]):
            phi_i = phi[i]
            t_i = t[i]
            beta_row_value = np.square(t_i - (Mn.T @ phi_i))
            beta_row_list.append(beta_row_value)
        beta = 1 / ((1 / (N - gamma)) * sum(beta_row_list))

        return beta

    def calculateMSE(self, phi, Mn, t):  # Calculating MSE
        SE = (phi @ Mn - t) ** 2
        MSE = np.mean(SE)
        return MSE


b = bayes()
start = time.time()

# ============================================DATASET:
WINE=========================================================

dataarray = b.readData(winePath, wineTrainFile)
t = np.array(b.readData(winePath, wineTrainRFile))
t = t.astype(np.float64)
phi = np.array(dataarray)
phi = phi.astype(np.float64)
phiTranspose = phi.T
phiTranspose = phiTranspose.astype(np.float64)
phiTphi = phiTranspose @ phi
phiT_t = phiTranspose @ t

alpha = 5.0
beta = 1.0

oldAlpha = 0
while alpha - oldAlpha > 0.0001:
    Sn = b.calculateSn(alpha, beta, phi, phiTphi)
    Mn = b.calculateMn(beta, Sn, phiT_t)
    BphiTphi = b.calculateBphiTphi(beta, phiTphi)
    gamma = b.calculateGamma(BphiTphi, alpha)
    oldAlpha = alpha
    alpha = b.calculateAlpha(gamma, Mn)
    beta = b.calculateBeta(phi, t, Mn, gamma)

lambdaa = alpha / beta

MSE = b.calculateMSE(phi, Mn, t)

end = time.time()

print("Dataset Used: Wine")
print("Value of alpha: " + str(alpha))
print("Value of beta: " + str(beta))
print("Value of Lambda: " + str(lambdaa))
print("Associated MSE:" + str(MSE))
print("Runtime:" + str(end - start))

# #================================================DATASET:
CRIME=========================================
#
# dataarray = b.readData(crimePath, crimeTrainFile)
# t = np.array(b.readData(crimePath, crimeTrainRFile))
# t = t.astype(np.float64)
# phi = np.array(dataarray)
# phi = phi.astype(np.float64)
# phiTranspose = phi.T
# phiTranspose = phiTranspose.astype(np.float64)
# phiTphi = phiTranspose @ phi
# phiT_t = phiTranspose @ t
#
# alpha = 5.0
# beta = 1.0
#
# oldAlpha = 0
# while alpha-oldAlpha > 0.0001:
#     Sn = b.calculateSn(alpha, beta, phi, phiTphi)
#     Mn = b.calculateMn(beta, Sn, phiT_t)
#     BphiTphi = b.calculateBphiTphi(beta, phiTphi)
#     gamma = b.calculateGamma(BphiTphi, alpha)
#     oldAlpha = alpha
#     alpha = b.calculateAlpha(gamma, Mn)
#     beta = b.calculateBeta(phi, t, Mn, gamma)
#
# lambdaa = alpha/beta
#
# MSE = b.calculateMSE(phi,Mn,t)
#
# end = time.time()
# print("Dataset Used: Crime")
# print("Value of alpha: " + str(alpha))
# print("Value of beta: " + str(beta))
# print("Value of Lambda: " + str(lambdaa))
# print("Associated MSE:" + str(MSE))
# print("Runtime:" + str(end-start))
#
#
# #==================================================DATASET:
ARTSMALL==================================================
#
# dataarray = b.readData(artsmallPath, artsmallTrainFile)
# t = np.array(b.readData(artsmallPath, artsmallTrainRFile))
# t = t.astype(np.float64)
# phi = np.array(dataarray)
# phi = phi.astype(np.float64)
# phiTranspose = phi.T
# phiTranspose = phiTranspose.astype(np.float64)
# phiTphi = phiTranspose @ phi
# phiT_t = phiTranspose @ t
#
# alpha = 5.0
# beta = 1.0
#
# oldAlpha = 0
# while alpha-oldAlpha > 0.0001:
#     Sn = b.calculateSn(alpha, beta, phi, phiTphi)
#     Mn = b.calculateMn(beta, Sn, phiT_t)
#     BphiTphi = b.calculateBphiTphi(beta, phiTphi)
#     gamma = b.calculateGamma(BphiTphi, alpha)
#     oldAlpha = alpha
#     alpha = b.calculateAlpha(gamma, Mn)
#     beta = b.calculateBeta(phi, t, Mn, gamma)
#
# lambdaa = alpha/beta
#
# MSE = b.calculateMSE(phi,Mn,t)
#
# end = time.time()
# print("Dataset Used: ArtSmall")
# print("Value of alpha: " + str(alpha))
# print("Value of beta: " + str(beta))
# print("Value of Lambda: " + str(lambdaa))
# print("Associated MSE:" + str(MSE))
# print("Runtime:" + str(end-start))
#
#
# #==========================================================DATASET:
ARTLARGE==============================================
#
# dataarray = b.readData(artlargePath, artlargeTrainFile)
# t = np.array(b.readData(artlargePath, artlargeTrainRFile))
# t = t.astype(np.float64)
# phi = np.array(dataarray)
# phi = phi.astype(np.float64)
# phiTranspose = phi.T
# phiTranspose = phiTranspose.astype(np.float64)
# phiTphi = phiTranspose @ phi
# phiT_t = phiTranspose @ t
#
# alpha = 5.0
# beta = 1.0
#
# oldAlpha = 0
# while alpha - oldAlpha > 0.0001:
#     Sn = b.calculateSn(alpha, beta, phi, phiTphi)
#     Mn = b.calculateMn(beta, Sn, phiT_t)
#     BphiTphi = b.calculateBphiTphi(beta, phiTphi)
#     gamma = b.calculateGamma(BphiTphi, alpha)
#     oldAlpha = alpha
#     alpha = b.calculateAlpha(gamma, Mn)
#     beta = b.calculateBeta(phi, t, Mn, gamma)
#
# lambdaa = alpha / beta
#
# MSE = b.calculateMSE(phi, Mn, t)
# end = time.time()
# print("Dataset Used: ArtLarge")
# print("Value of alpha: " + str(alpha))
# print("Value of beta: " + str(beta))
# print("Value of Lambda: " + str(lambdaa))
# print("Associated MSE:" + str(MSE))
# print("Runtime:" + str(end - start))
#
```