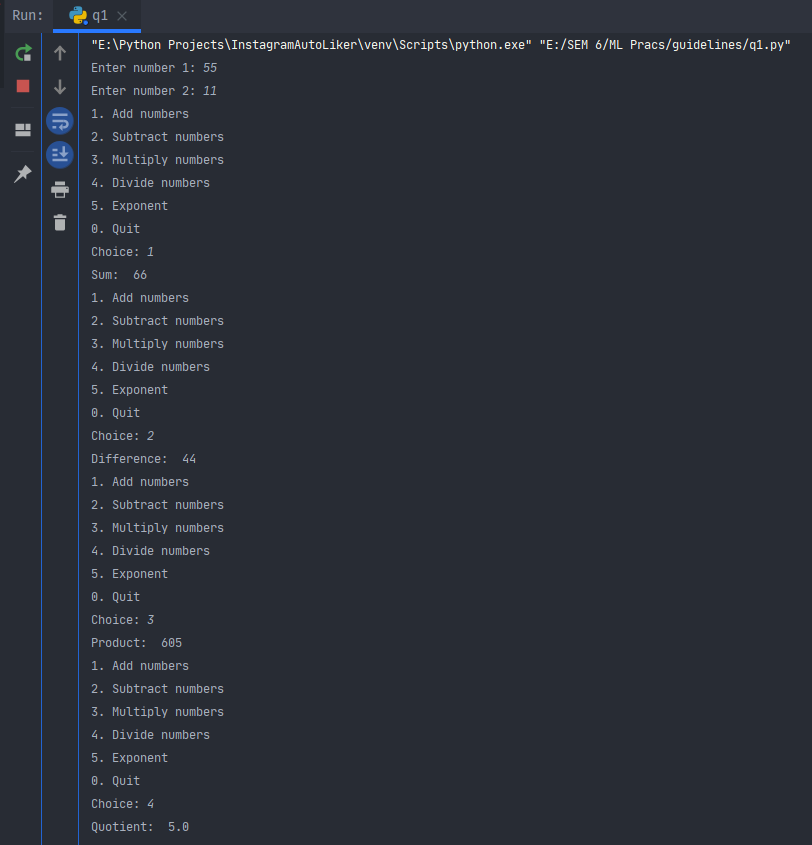
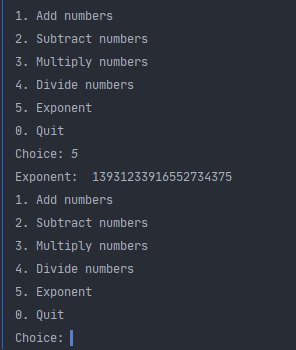
Q1.

# Perform elementary mathematical operations in Octave/MATLAB like addition, multiplication, division and exponentiation.

def getChoice():  
    print("1. Add numbers")  
    print("2. Subtract numbers")  
    print("3. Multiply numbers")  
    print("4. Divide numbers")  
    print("5. Exponent")  
    print("0. Quit")  
    option = int(input("Choice: "))  
    return option  
  
  
n1 = int(input("Enter number 1: "))  
n2 = int(input("Enter number 2: "))  
choice = getChoice()  
  
while choice != 0:  
    if choice == 1:  
        print("Sum: ", n1 + n2)  
    elif choice == 2:  
        print("Difference: ", n1 - n2)  
    elif choice == 3:  
        print("Product: ", n1 \* n2)  
    elif choice == 4:  
        print("Quotient: ", n1 / n2)  
    elif choice == 5:  
        print("Exponent: ", n1 \*\* n2)  
    choice = getChoice()

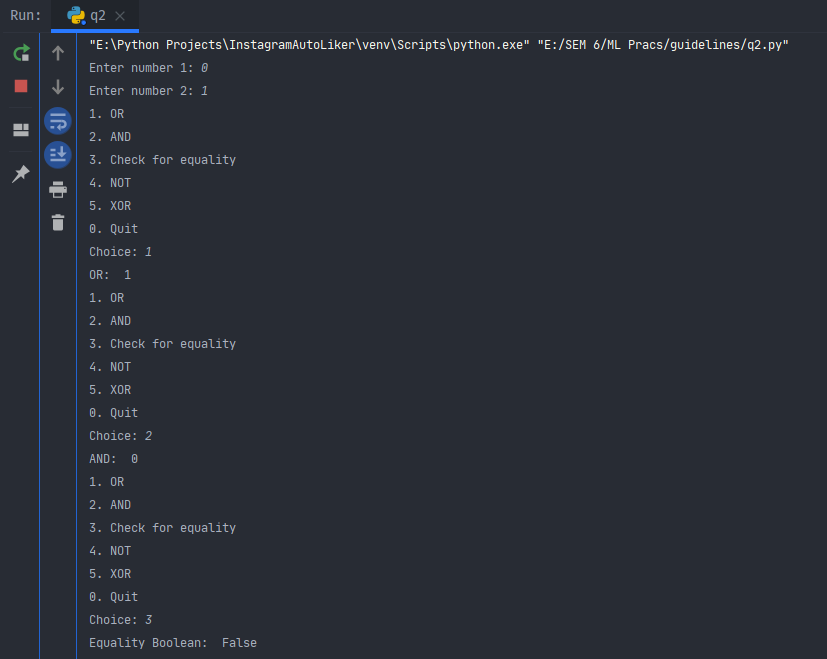


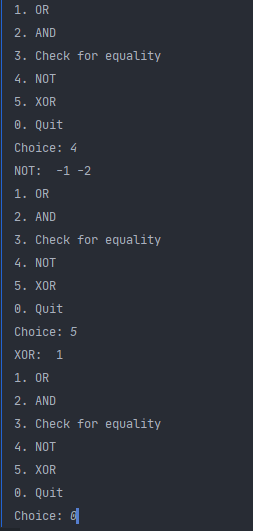


Q2

# Perform elementary logical operations in Octave/MATLAB (like OR, AND, Checking for Equality, NOT, XOR).

def getChoice():  
    print("1. OR")  
    print("2. AND")  
    print("3. Check for equality")  
    print("4. NOT")  
    print("5. XOR")  
    print("0. Quit")  
    option = int(input("Choice: "))  
    return option  
  
  
n1 = int(input("Enter number 1: "))  
n2 = int(input("Enter number 2: "))  
choice = getChoice()  
  
while choice != 0:  
    if choice == 1:  
        print("OR: ", n1 | n2)  
    elif choice == 2:  
        print("AND: ", n1 & n2)  
    elif choice == 3:  
        print("Equality Boolean: ", n1 == n2)  
    elif choice == 4:  
        print("NOT: ", ~n1, ~n2)  
    elif choice == 5:  
        print("XOR: ", n1 ^ n2)  
    choice = getChoice()

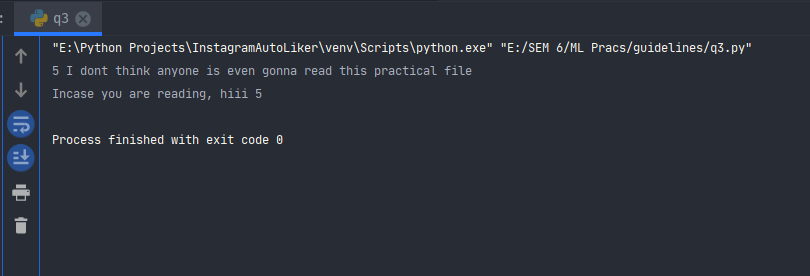




Q3

# Create, initialize and display simple variables and simple strings and use simple formatting for variable.

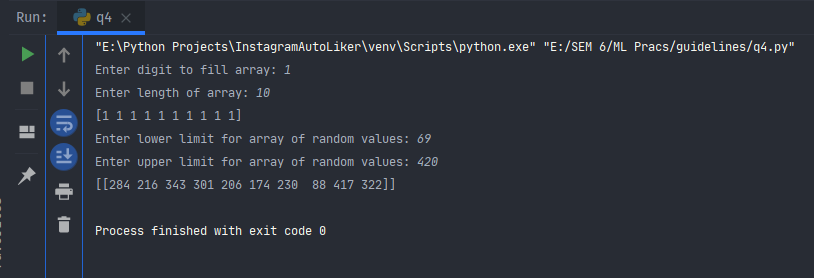
x = 5  
myString = "Hello World"   
myFormattedString = f"Hello World {x}"  
  
print(x, myString)  
print(myFormattedString)



Q4

# Create/Define single dimension / multi-dimension arrays, and arrays with specific values like array of all ones, all zeros, array with random values within a range, or a diagonal matrix.

import numpy as np  
  
d = int(input("Enter digit to fill array: "))  
n = int(input("Enter length of array: "))  
arr = np.full((1, n), d).flatten()  
print(arr)  
  
l = int(input("Enter lower limit for array of random values: "))  
u = int(input("Enter upper limit for array of random values: "))  
  
arr = np.random.randint(l, u, size=(1, n))  
print(arr)



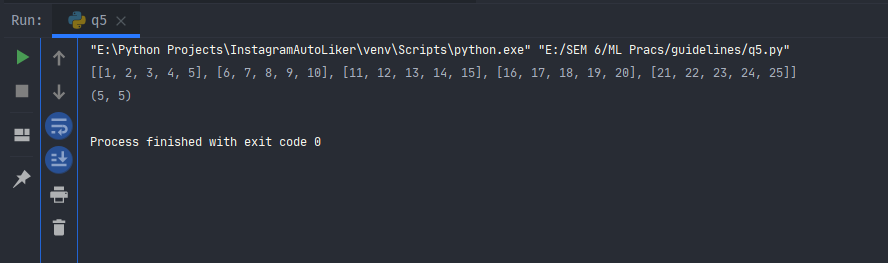
Q5

# Use command to compute the size of a matrix, size/length of a particular row/column, load data from a text file, store matrix data to a text file, finding out variables and their features in the current scope.

import numpy as np  
  
  
with open('input.txt', 'r') as f:  
    l = [[int(num) for num in line.split(',')] for line in f]  
print(l)  
  
print(np.shape(l))

Input.txt

1,2,3,4,5  
6,7,8,9,10  
11,12,13,14,15  
16,17,18,19,20  
21,22,23,24,25

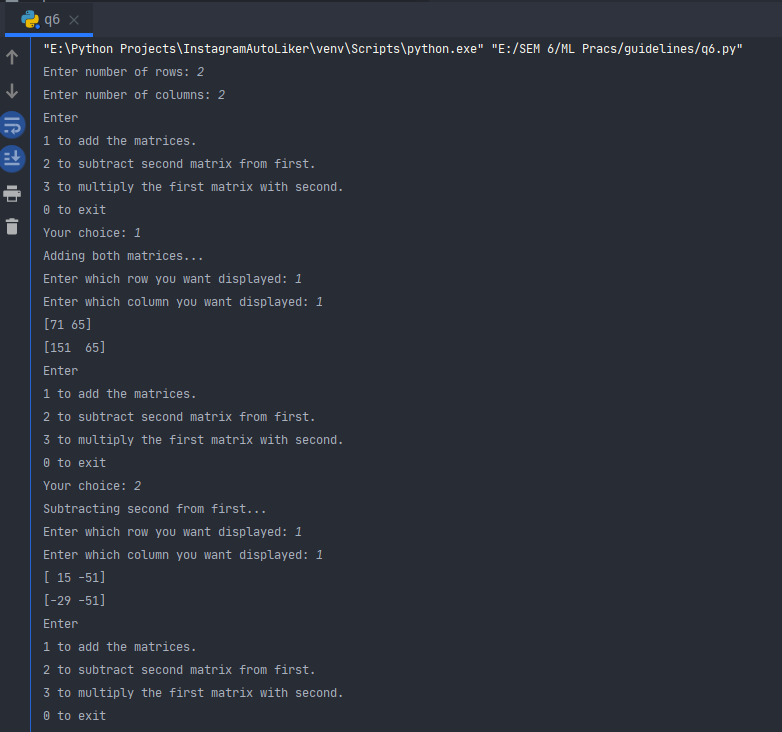


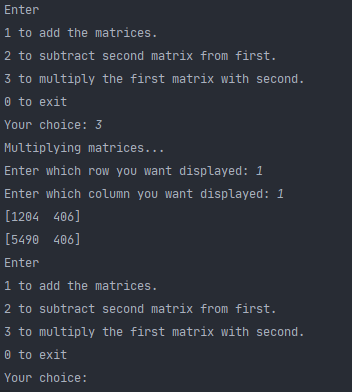
Q6

# Perform basic operations on matrices (like addition, subtraction, multiplication) and display specific rows or columns of the matrix.

import numpy as np  
  
  
def getChoice():  
    c = int(input("Enter "  
                  "\n1 to add the matrices."  
                  "\n2 to subtract second matrix from

first."  
                  "\n3 to multiply the first matrix with second."  
                  "\n0 to exit"  
                  "\nYour choice: "))  
    return c  
  
  
def nthRowCol():  
    r = int(input("Enter which row you want displayed: "))  
    c = int(input("Enter which column you want displayed: "))  
    return r, c  
  
  
m = int(input("Enter number of rows: "))  
n = int(input("Enter number of columns: "))  
  
mat1 = np.random.randint(1, 100, size=(m, n))  
mat2 = np.random.randint(1, 100, size=(m, n))  
choice = getChoice()  
while choice != 0:  
    if choice == 1:  
        print("Adding both matrices...")  
        result = np.add(mat1, mat2)  
        row, col = nthRowCol()  
        print(result[row])  
        print(result[:, col])  
    elif choice == 2:  
        print("Subtracting second from first...")  
        result = np.subtract(mat1, mat2)  
        row, col = nthRowCol()  
        print(result[row])  
        print(result[:, col])  
    elif choice == 3:  
        print("Multiplying matrices...")  
        result = np.multiply(mat1, mat2)  
        row, col = nthRowCol()  
        print(result[row])  
        print(result[:, col])  
    elif choice == 0:  
        print("End of program.")  
    else:  
        print("Invalid option chosen.")  
    choice = getChoice()

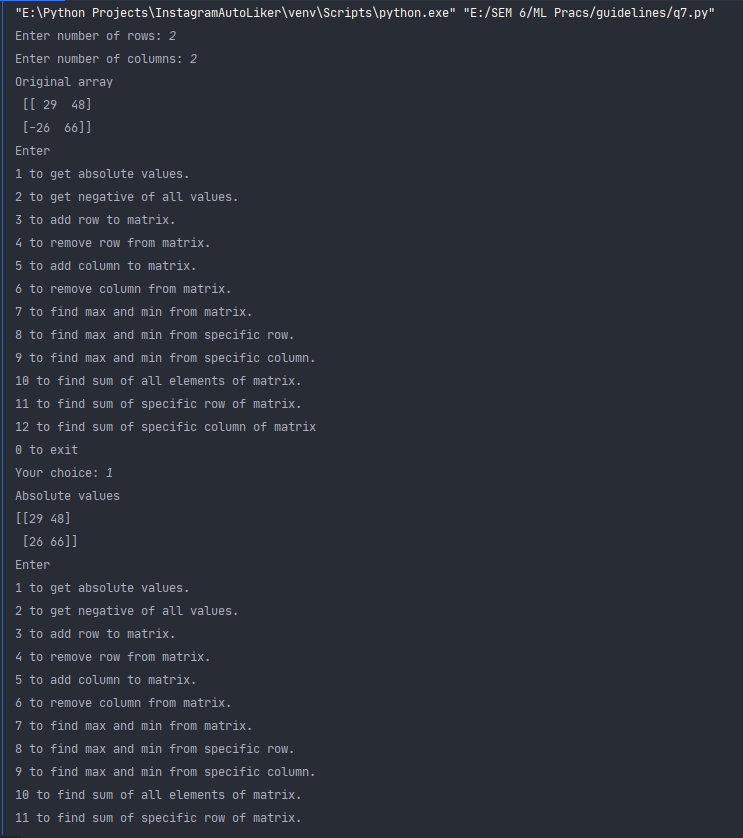


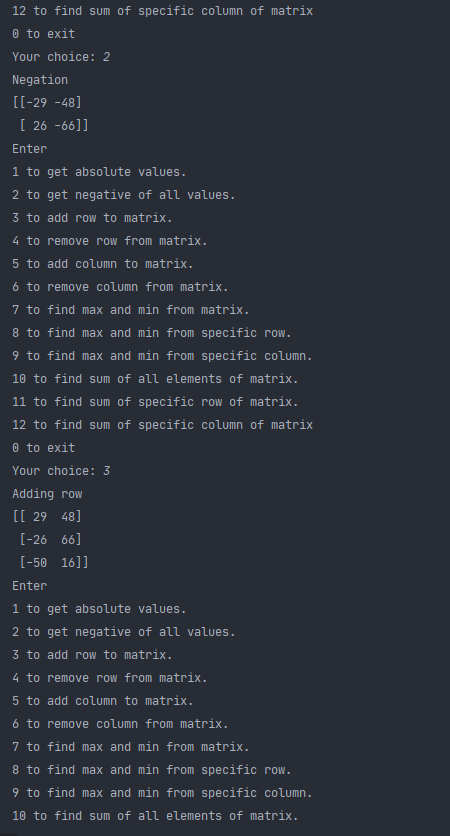


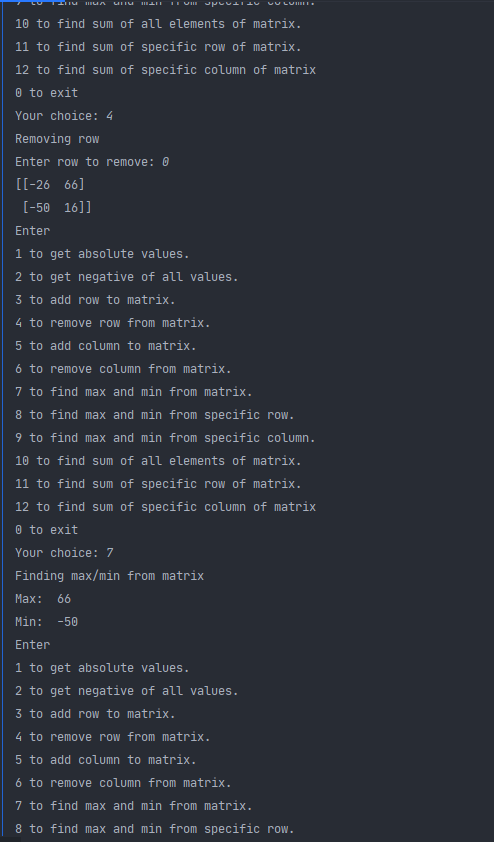
Q7

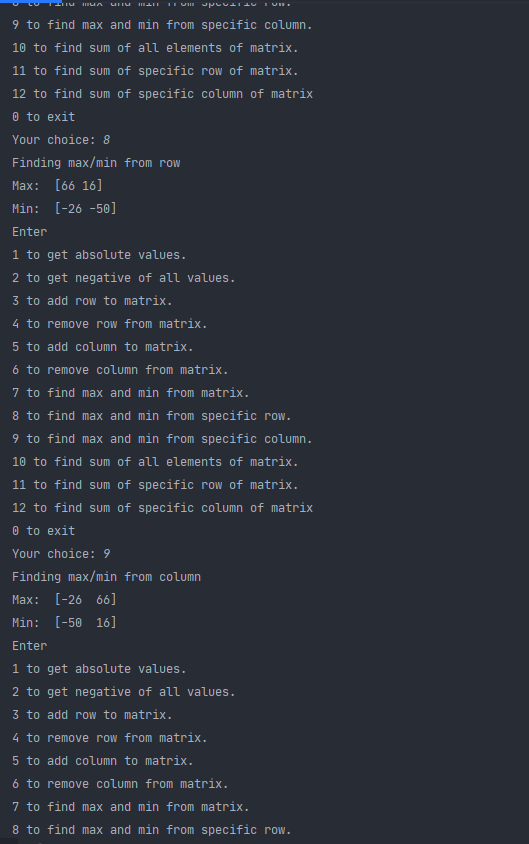
# Perform other matrix operations like converting matrix data to absolute values, taking the negative of matrix values, adding/removing rows/columns from a matrix, finding the maximum or minimum values in a matrix or in a row/column, and finding the sum of some/all elements in a matrix.

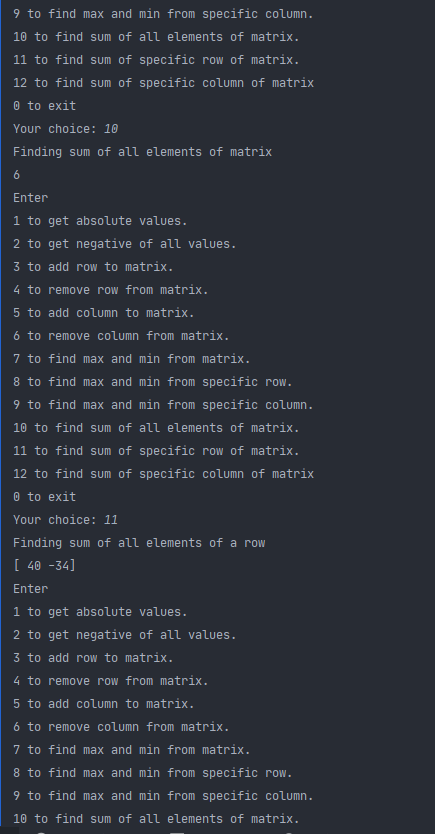
import numpy as np  
  
  
def getChoice():  
    c = int(input("Enter "  
                  "\n1 to get absolute values."  
                  "\n2 to get negative of all values."  
                  "\n3 to add row to matrix."  
                  "\n4 to remove row from matrix."  
                  "\n5 to add column to matrix."  
                  "\n6 to remove column from matrix."  
                  "\n7 to find max and min from matrix."  
                  "\n8 to find max and min from specific row."  
                  "\n9 to find max and min from specific column."  
                  "\n10 to find sum of all elements of matrix."  
                  "\n11 to find sum of specific row of matrix."  
                  "\n12 to find sum of specific column of matrix"  
                  "\n0 to exit"  
                  "\nYour choice: "))  
    return c  
  
  
m = int(input("Enter number of rows: "))  
n = int(input("Enter number of columns: "))  
  
mat = np.random.randint(-100, 100, size=(m, n))  
print("Original array\n", mat)  
choice = getChoice()  
while choice != 0:  
    if choice == 1:  
        print("Absolute values")  
        print(np.abs(mat))  
    elif choice == 2:  
        print("Negation")  
        print(np.negative(mat))  
    elif choice == 3:  
        print("Adding row")  
        mat = np.append(mat, [np.random.randint(-100, 100, size=(1, n)).flatten()], axis=0)  
        print(mat)  
    elif choice == 4:  
        print("Removing row")  
        r = int(input("Enter row to remove: "))  
        if 0 <= r <= m:  
            mat = np.delete(mat, r, axis=0)  
            print(mat)  
        else:  
            print("Index is outside matrix range.")  
    elif choice == 5:  
        print("Adding column")  
        mat = np.append(mat, [np.random.randint(-100, 100, size=(m, 1)).flatten()], axis=1)  
        print(mat)  
    elif choice == 6:  
        print("Removing column")  
        c = int(input("Enter column to remove: "))  
        if 0 <= c <= m:  
            mat = np.delete(mat, c, axis=1)  
            print(mat)  
        else:  
            print("Index is outside matrix range.")  
    elif choice == 7:  
        print("Finding max/min from matrix")  
        print("Max: ", np.max(mat))  
        print("Min: ", np.min(mat))  
    elif choice == 8:  
        print("Finding max/min from row")  
        print("Max: ", np.amax(mat, axis=1))  
        print("Min: ", np.amin(mat, axis=1))  
    elif choice == 9:  
        print("Finding max/min from column")  
        print("Max: ", np.amax(mat, axis=0))  
        print("Min: ", np.amin(mat, axis=0))  
    elif choice == 10:  
        print("Finding sum of all elements of matrix")  
        print(np.sum(mat))  
    elif choice == 11:  
        print("Finding sum of all elements of a row")  
        print(np.sum(mat, axis=1))  
    elif choice == 12:  
        print("Finding sum of all elements of a column")  
        print(np.sum(mat, axis=0))  
    elif choice == 0:  
        print("End of program.")  
    else:  
        print("Invalid option chosen.")  
    choice = getChoice()

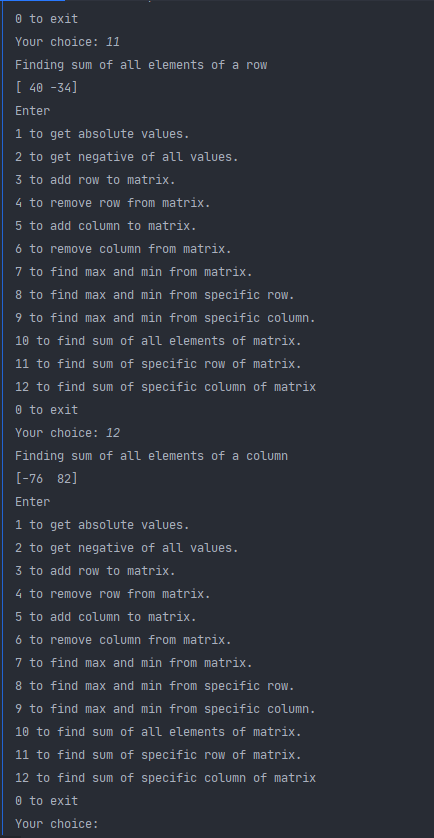












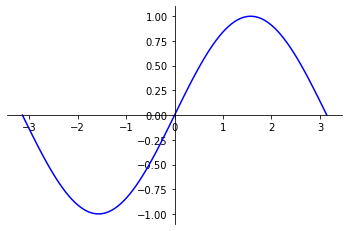
Q8

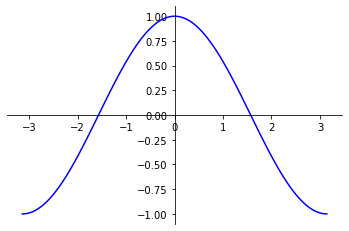
# Create various type of plots/charts like histograms, plot based on sine/cosine function based on data from a matrix. Further label different axes in a plot and data in a plot.

from matplotlib import pyplot as plt  
import numpy as np  
  
  
a = np.array([22, 87, 5, 43, 56, 73, 55, 54, 11, 20, 51, 5, 79, 31, 27])  
  
fig, ax = plt.subplots(figsize=(10, 7))  
ax.hist(a, bins=[0, 25, 50, 75, 100])  
ax.set\_xlabel("x")  
ax.set\_ylabel("y")  
  
plt.show()

x = np.linspace(-np.pi, np.pi, 100)  
  
y = np.sin(x)  
  
fig = plt.figure()  
ax = fig.add\_subplot(1, 1, 1)  
ax.spines['left'].set\_position('center')  
ax.spines['bottom'].set\_position('center')  
ax.spines['right'].set\_color('none')  
ax.spines['top'].set\_color('none')  
ax.xaxis.set\_ticks\_position('bottom')  
ax.yaxis.set\_ticks\_position('left')  
  
plt.plot(x, y, 'b-')  
  
  
x = np.linspace(-np.pi, np.pi, 100)  
  
y = np.cos(x)  
  
fig = plt.figure()  
ax = fig.add\_subplot(1, 1, 1)  
ax.spines['left'].set\_position('center')  
ax.spines['bottom'].set\_position('center')  
ax.spines['right'].set\_color('none')  
ax.spines['top'].set\_color('none')  
ax.xaxis.set\_ticks\_position('bottom')  
ax.yaxis.set\_ticks\_position('left')  
  
plt.plot(x, y, 'b-')  
  
plt.show()







Q9

# Generate different subplots from a given plot and color plot data.

import pandas as pd

import matplotlib.pyplot as plt

df = pd.read\_csv("company\_sales\_data.csv")

profitList = df['total\_profit'].tolist()

monthList = df['month\_number'].tolist()

plt.plot(monthList, profitList, label='Profit data of last year', color='r', marker='o', markerfacecolor='k', linestyle='--', linewidth=3)

plt.xlabel('Month Number')

plt.ylabel('Total profit')

plt.legend(loc="lower right")

plt.title('Company Sales data of last year')

plt.xticks(monthList)

plt.yticks([100000, 200000, 300000, 400000, 500000])

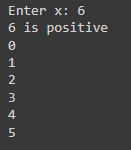
plt.show()



Q10

# Use conditional statements and different type of loops based on simple example/s.

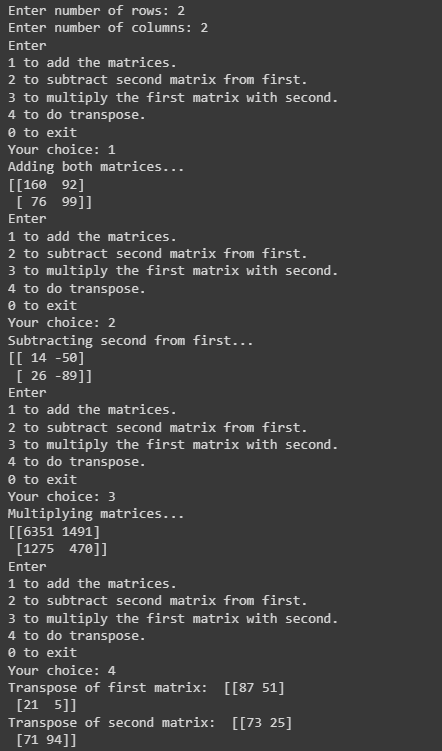
x = int(input("Enter x: "))  
  
if x > 0:  
    print(f"{x} is positive")  
elif x < 0:  
    print(f"{x} is negative")  
else:  
    print(f"{x} is zero")  
  
for i in range(0, x):  
    print(f"{i} ")

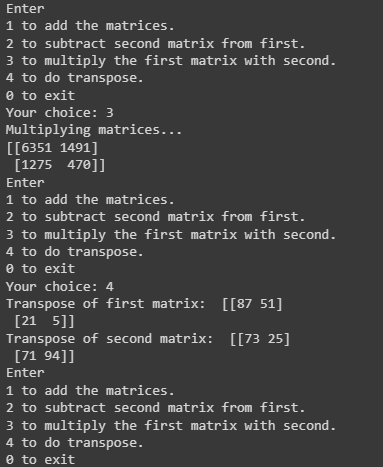


Q11

# 11. Perform vectorized implementation of simple matrix operation like finding the transpose of a matrix, adding, subtracting or multiplying two matrices

import numpy as np  
  
  
def getChoice():  
    c = int(input("Enter "  
                  "\n1 to add the matrices."  
                  "\n2 to subtract second matrix from first."  
                  "\n3 to multiply the first matrix with second."  
                  "\n4 to do transpose."  
                  "\n0 to exit"  
                  "\nYour choice: "))  
    return c  
  
  
m = int(input("Enter number of rows: "))  
n = int(input("Enter number of columns: "))  
  
mat1 = np.random.randint(1, 100, size=(m, n))  
mat2 = np.random.randint(1, 100, size=(m, n))  
choice = getChoice()  
while choice != 0:  
    if choice == 1:  
        print("Adding both matrices...")  
        print(np.add(mat1, mat2))  
    elif choice == 2:  
        print("Subtracting second from first...")  
        print(np.subtract(mat1, mat2))  
    elif choice == 3:  
        print("Multiplying matrices...")  
        print(np.multiply(mat1, mat2))  
    elif choice == 4:  
        print("Transpose of first matrix: ", mat1.T)  
        print("Transpose of second matrix: ", mat2.T)  
    elif choice == 0:  
        print("End of program.")  
    else:  
        print("Invalid option chosen.")  
    choice = getChoice()





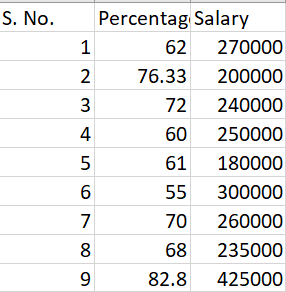
Q12

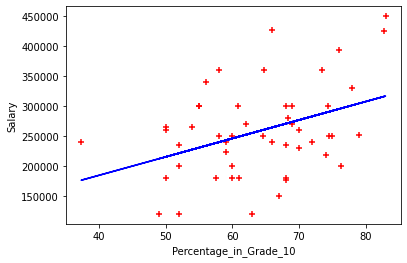
12. Implement Linear Regression problem. For example, based on a dataset comprising of existing set of prices and area/size of the houses, predict the estimated price of a given house.

import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn import linear\_model

data = pd.read\_csv('MBASalary.csv')  
  
reg = linear\_model.LinearRegression()  
reg.fit(data[['Percentage\_in\_Grade\_10']], data.Salary)  
  
plt.xlabel('Percentage\_in\_Grade\_10')  
plt.ylabel('Salary')  
plt.scatter(data.Percentage\_in\_Grade\_10, data.Salary, color='red', marker='+')  
plt.plot(data.Percentage\_in\_Grade\_10, reg.predict(data[['Percentage\_in\_Grade\_10']]), color='blue')  
plt.show()

Sample of the dataset





Q13

# Based on multiple features/variables perform Linear Regression. For example, based on a number of additional features like number of bedrooms, servant room, number of balconies, number of houses of years a house has been built predict the price of a house.

import pandas as pd  
from sklearn import linear\_model  
import math  
  
df = pd.read\_csv("homeprices.csv")  
  
median\_bedrooms = math.floor(df.bedrooms.median())  
df.bedrooms = df.bedrooms.fillna(median\_bedrooms)  
  
reg = linear\_model.LinearRegression()  
reg.fit(df[['area', 'bedrooms', 'age']], df.price)  
  
val1 = reg.predict([[3000, 3, 40]])  
val2 = reg.predict([[2500, 4, 5]])  
  
print(f"For instance with values [3000, 3, 40]: {val1}")  
print(f"For instance with values [2500, 4, 5]: {val2}")



Q14

# Implement a classification/ logistic regression problem. For example based on different features of students data, classify, whether a student is suitable for a particular activity. Based on the available dataset, a student can also implement another classification problem like checking whether an email is spam or not.

import pandas as pd  
from sklearn.linear\_model import LogisticRegression  
from sklearn.model\_selection import train\_test\_split  
  
df = pd.read\_csv("HR\_comma\_sep.csv")  
df\_left = df.loc[df["left"] > 0]  
df\_stayed = df.loc[df["left"] == 0]  
  
dummies = pd.get\_dummies(df.salary)  
merged = pd.concat([df, dummies], axis='columns')  
merged.drop(['salary'], axis='columns')  
  
df["Department"] = df["Department"].astype('category').cat.codes

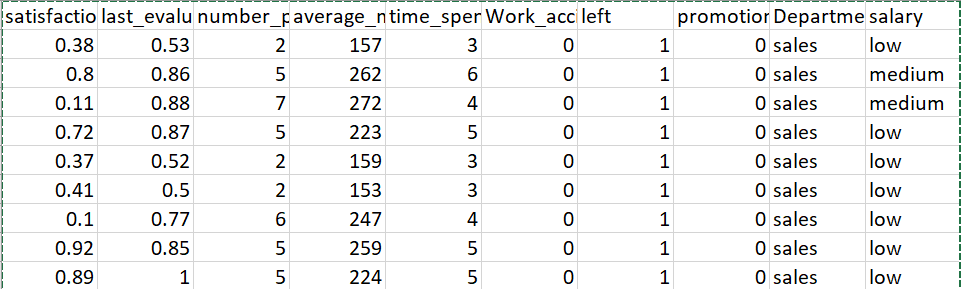
df["salary"] = df["salary"].astype('category').cat.codes

x = df.drop(["left"], axis=1)  
x = x.drop(["last\_evaluation"], axis=1)  
y = df[["left"]].values  
  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.22, random\_state=27)

logReg = LogisticRegression()  
logReg.fit(x\_train, y\_train)  
predictions = logReg.predict(x\_test)  
score = logReg.score(x\_test, y\_test)  
print("Score: ", score)



Sample of the dataset



Q15

# Use some function for regularization of dataset based on problem 14.

import warnings

warnings.filterwarnings('ignore')

import matplotlib.pyplot as plt

import numpy as np

def grade(h, i):

return -2.5 \* h + i

from mpl\_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(16,9))

ax = fig.add\_subplot(111, projection='3d')

h = np.array([2, 4]) # hours on CSGO from 0 to 10

i = np.array([85, 100]) # IQ from 70 to 130

grades = grade(h, i)

ax.plot(h, i, grades)

ax.scatter([2, 4],[85,100], [80, 90], s=100, c='red') # plotting our sample points

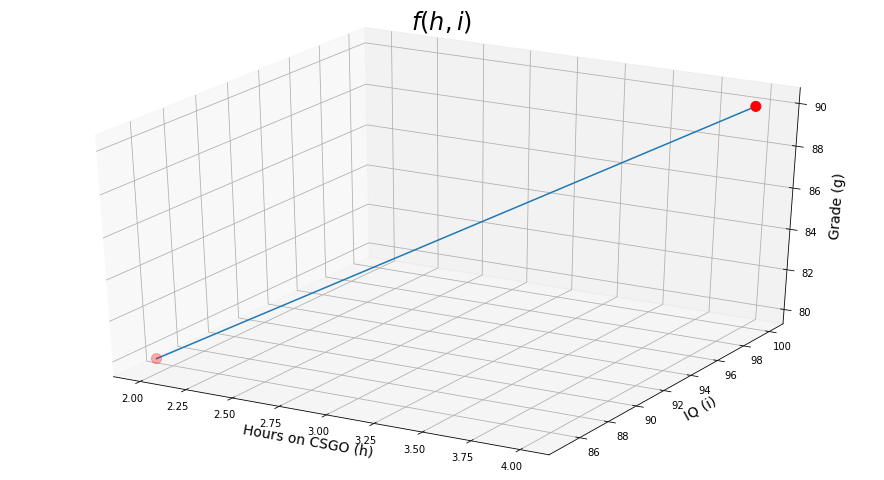
ax.set\_xlabel("Hours on CSGO (h)", fontsize=14)

ax.set\_ylabel("IQ (i)", fontsize=14)

ax.set\_zlabel("Grade (g)", fontsize=14)

plt.title(r"$f(h,i)$", fontsize=24)

plt.show()



from sklearn.datasets import load\_boston

*# loading the data*

X, y= load\_boston(return\_X\_y=True) *# we want both features matrix X, and labels vector y*

X.shape *# the dataset has 506 houses with 13 features (or predictors) for a house price in boston*



import warnings

warnings.filterwarnings('ignore')

from sklearn.linear\_model import LinearRegression

# Initialize the model

lr = LinearRegression()

# training the model

# we pass in the features as well as the labels we want to map to (remember the CGSO and IQ = GPA example?)

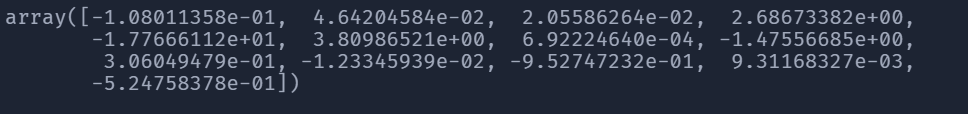
lr.fit(X, y)

# we can now use the model for predictions! We will just give the same predictors

predictions = lr.predict(X)

*# here are the coefficients*

lr.coef\_



*# the intercept*

lr.intercept\_



from sklearn.linear\_model import ElasticNet, Lasso, Ridge

from sklearn.metrics import mean\_squared\_error  *# we will use MSE for evaluation*

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

# we set aside 20% of the data for testing, and use the remaining 80% for training

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

def plot\_errors(lambdas, train\_errors, test\_errors, title):

    plt.figure(figsize=(16, 9))

    plt.plot(lambdas, train\_errors, label="train")

    plt.plot(lambdas, test\_errors, label="test")

    plt.xlabel("$\\lambda$", fontsize=14)

    plt.ylabel("MSE", fontsize=14)

    plt.title(title, fontsize=20)

    plt.legend(fontsize=14)

    plt.show()

def evaluate\_model(Model, lambdas):

training\_errors = [] # we will store the error on the training set, for using each different lambda

testing\_errors = [] # and the error on the testing set

for l in lambdas:

# in sklearn, they refer to lambda as alpha, the name is different in different literature

# Model will be either Lasso, Ridge or ElasticNet

model = Model(alpha=l, max\_iter=1000) # we allow max number of iterations until the model converges

model.fit(X\_train, y\_train)

training\_predictions = model.predict(X\_train)

training\_mse = mean\_squared\_error(y\_train, training\_predictions)

training\_errors.append(training\_mse)

testing\_predictions = model.predict(X\_test)

testing\_mse = mean\_squared\_error(y\_test, testing\_predictions)

testing\_errors.append(testing\_mse)

return training\_errors, testing\_errors

import warnings

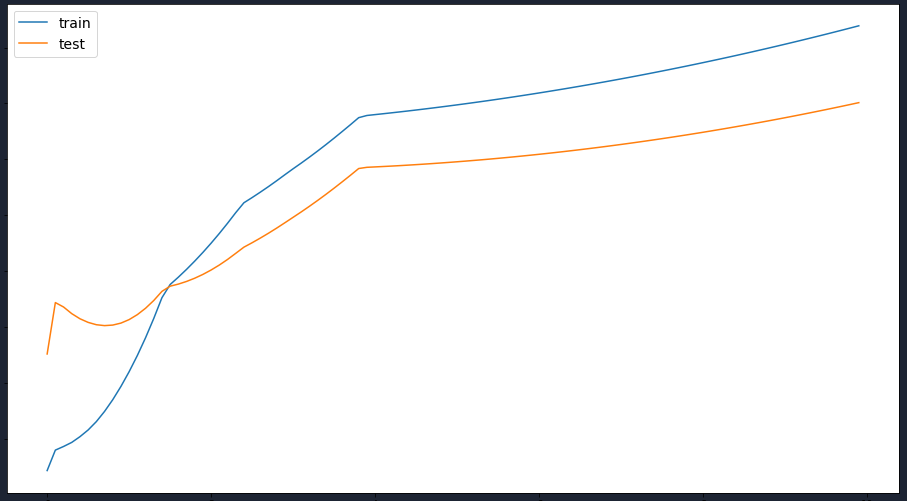
warnings.filterwarnings('ignore')

# let's generate different values for lambda from 0 (no-regularization) and (10 too much regularization)

lambdas = np.arange(0, 10, step=0.1)

lasso\_train, lasso\_test = evaluate\_model(Lasso, lambdas)

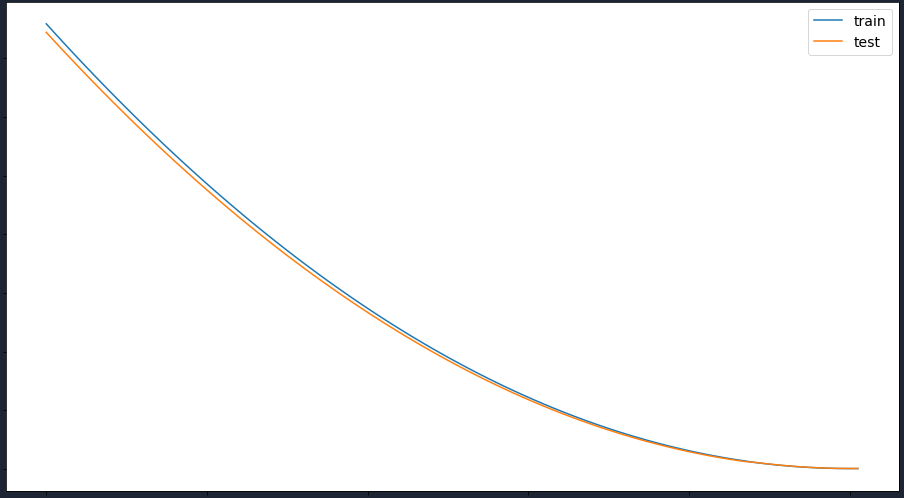
plot\_errors(lambdas, lasso\_train, lasso\_test, "Lasso")



lambdas = np.arange(-10, 0.2, step=0.1)

lasso\_train, lasso\_test = evaluate\_model(Lasso, lambdas)

plot\_errors(lambdas, lasso\_train, lasso\_test, "Lasso")



Q16

# Use some function for neural networks, like Stochastic Gradient Descent or backpropagation - algorithm to predict the value of a variable based on the dataset of problem 14.

from numpy import\*

import pandas as pd

from sklearn.neural\_network import MLPClassifier

from sklearn.model\_selection import train\_test\_split

import warnings

warnings.filterwarnings('ignore')

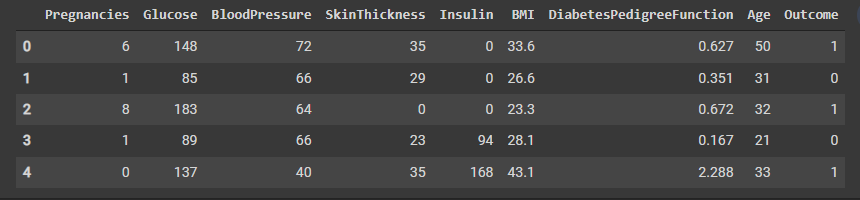
from google.colab import files

uploaded = files.upload()

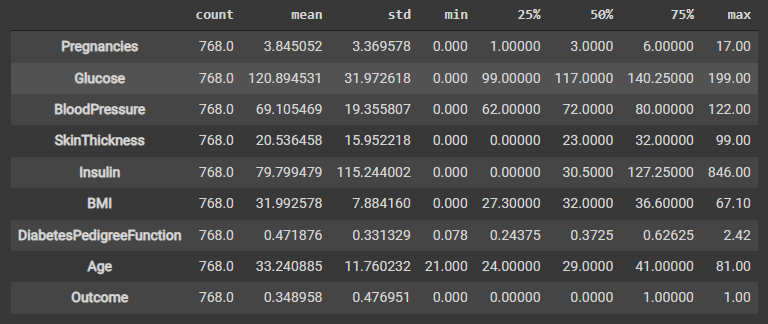
import io

df = pd.read\_csv(io.StringIO(uploaded['diabetes.csv'].decode('utf-8')))

df.head()



df.describe().transpose()



df.shape



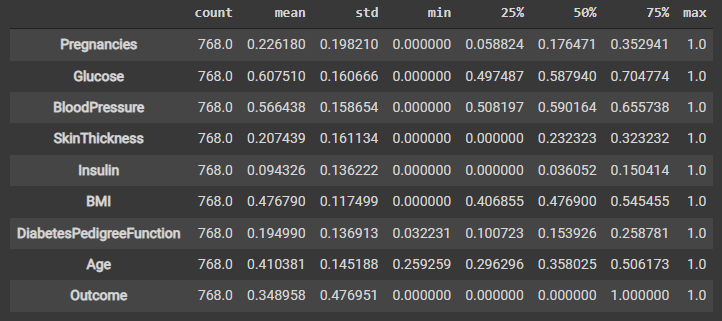
#Creating Arrays for the Features and the Response Variable

target\_column = ['Outcome']

predictors = list(set(list(df.columns))-set(target\_column))

df[predictors] = df[predictors]/df[predictors].max()

df.describe().transpose()



#Creating the Training and Test Datasets

X = df[predictors].values

y = df[target\_column].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=40)

print(X\_train.shape); print(X\_test.shape)



mlp = MLPClassifier(hidden\_layer\_sizes=(8,8), activation='relu', solver='sgd', max\_iter=500)

mlp.fit(X\_train,y\_train)



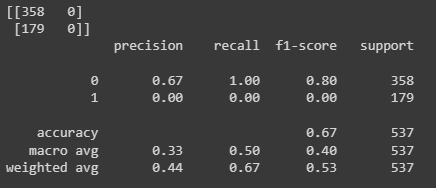
predict\_train = mlp.predict(X\_train)

predict\_test = mlp.predict(X\_test)

from sklearn.metrics import classification\_report,confusion\_matrix

print(confusion\_matrix(y\_train,predict\_train))

print(classification\_report(y\_train,predict\_train))



#The next step is to evaluate the performance of the model on the test data that is done with the lines of code below.

print(confusion\_matrix(y\_test,predict\_test))

print(classification\_report(y\_test,predict\_test))

