**Machine Learning Practical Back-up**

***Vaishnavi Jha***

***2018355***

***18066570048  
B.Sc. (Hons.) Computer Science***

***2018-2021***

**GUIDELINES QUESTIONS**

1. **Perform elementary mathematical operations in Python like addition, multiplication, division and exponentiation.**

**Program:**

x=int(input("Enter number 1: "))

y=int(input("Enter number 2: "))

print("Results after elementary maths operations:------")

print("Addition: ",x+y)

print("Subtraction: ",x-y)

if(y!=0):

print("Division: ",x/y)

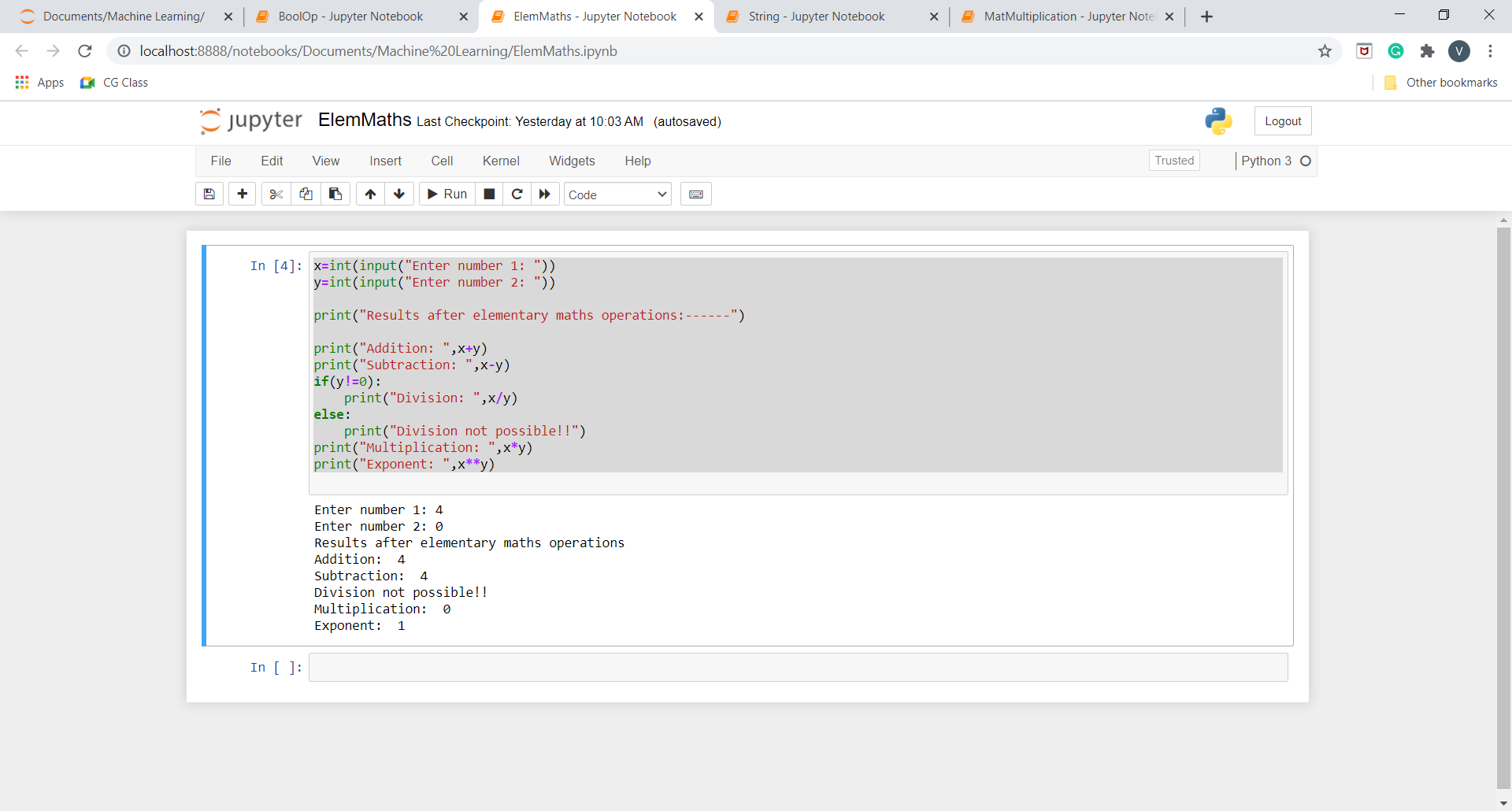
else:

print("Division not possible!!")

print("Multiplication: ",x\*y)

print("Exponent: ",x\*\*y)

**Output:**



1. **Perform elementary logical operations in Python (like OR, AND, Checking for Equality, NOT, XOR).**

**Program:**

x=int(input("Enter number 1: "))

y=int(input("Enter number 2: "))

print("Results after boolean operations:------")

print("AND :",x and y)

print("OR :",x or y)

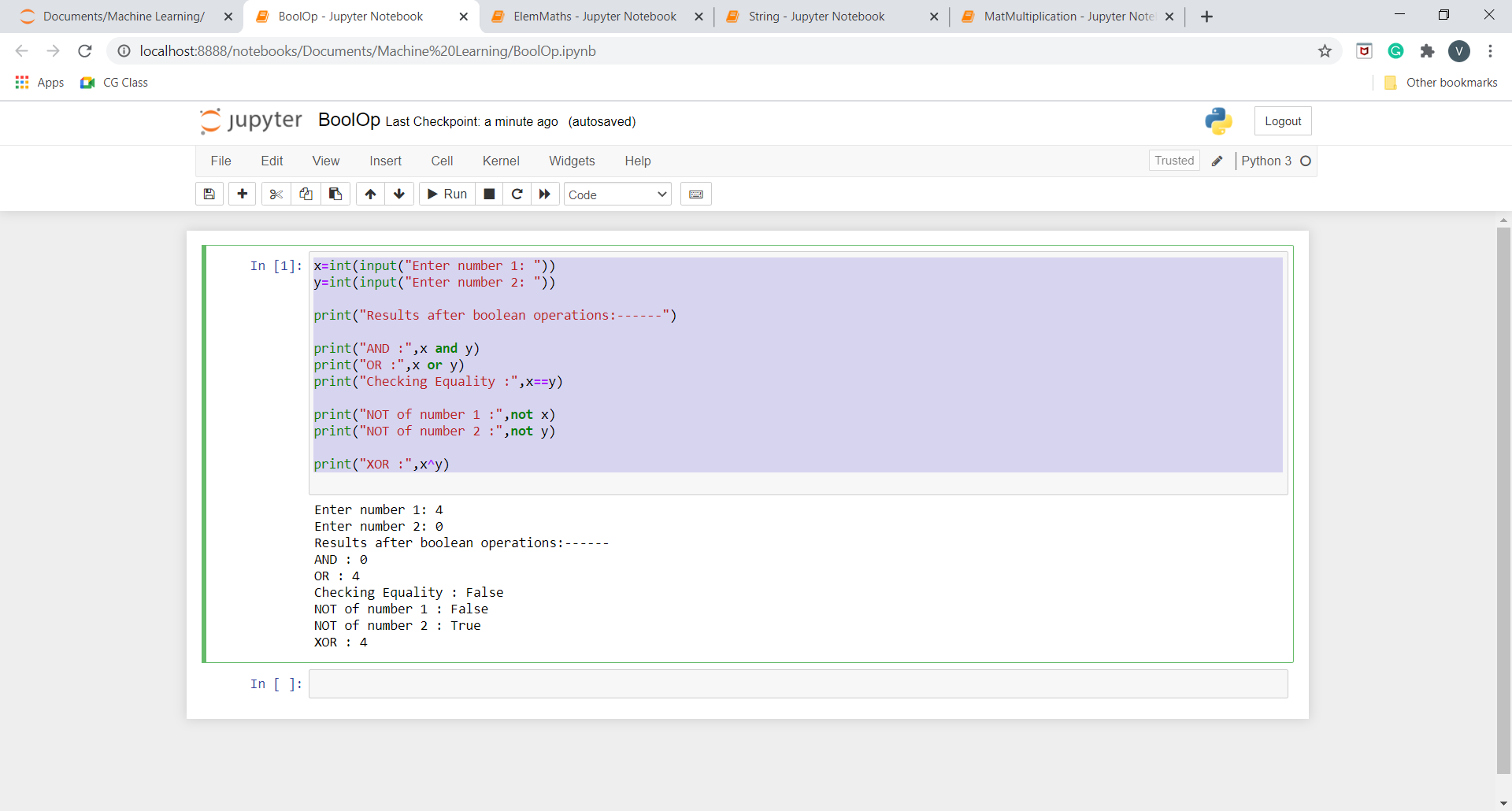
print("Checking Equality :",x==y)

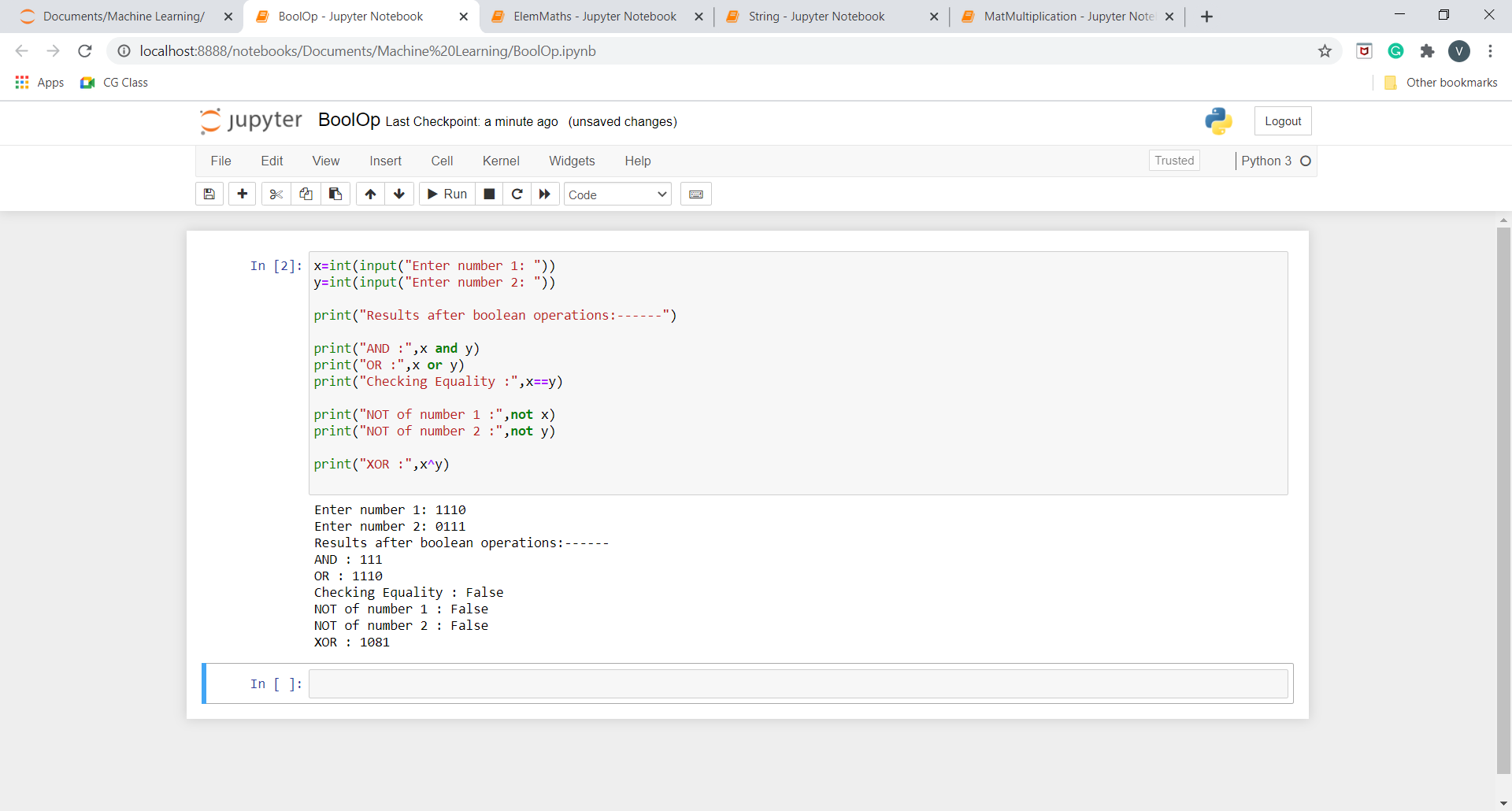
print("NOT of number 1 :",not x)

print("NOT of number 2 :",not y)

print("XOR :",x^y)

**Output:**





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

**Program:**

x= 2/3

z=16

y= """ vaishnavi jha is great.

Hail mogambo"""

print("String Formatting---------------------")

print(x)

print(y.split())

print(y.splitlines())

print(y.strip())

print(y.upper())

print(y.lower())

print(y.capitalize())

print(y.title())

print("\_".join(y.split()))

print()

print("Number Formatting----------------------")

print(int(x))

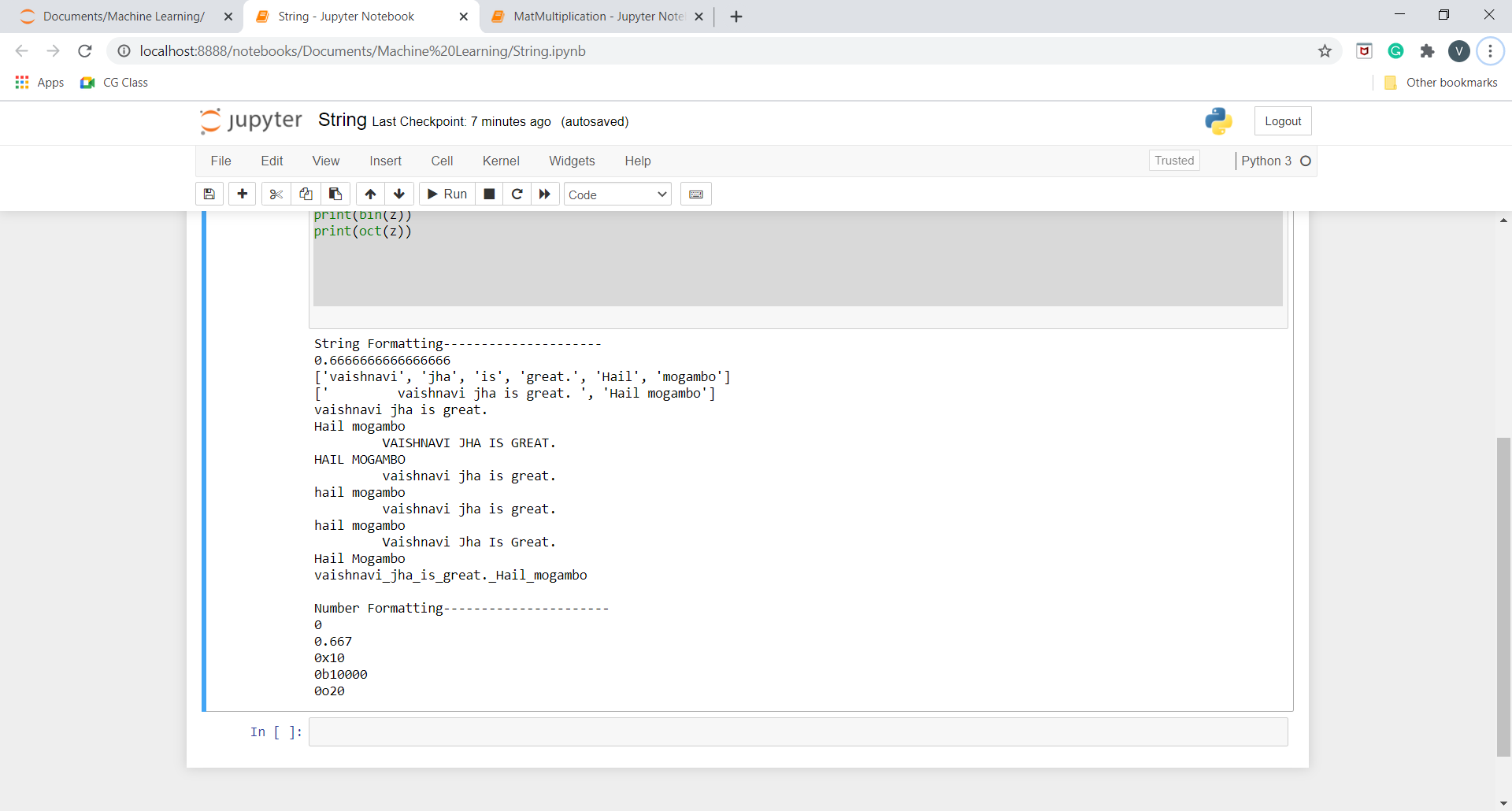
print("%0.3f"%(x))

print(hex(z))

print(bin(z))

print(oct(z))

**Output:**



1. **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.**

**Program:**

import numpy as np

print('Single Dimensional Array :')

print('---------------------------------------------------------------------------')

print('With all zeroes')

print(np.zeros(5,int))

print()

print('With all ones')

print(np.ones(5,int))

print()

print('With all random integers')

print(np.random.randint(1,10,5))

print()

print('With values within a range')

print(np.arange(1,10))

print()

print('Multi Dimensional Array :')

print('---------------------------------------------------------------------------')

print('With all zeroes')

print(np.zeros((4,3),int))

print()

print('With all ones')

print(np.ones((4,3),int))

print()

print('With all random integers')

print(np.random.randint(1,10,(4,3)))

print()

print('With values within a range')

print(np.arange(1,10).reshape(3,3))

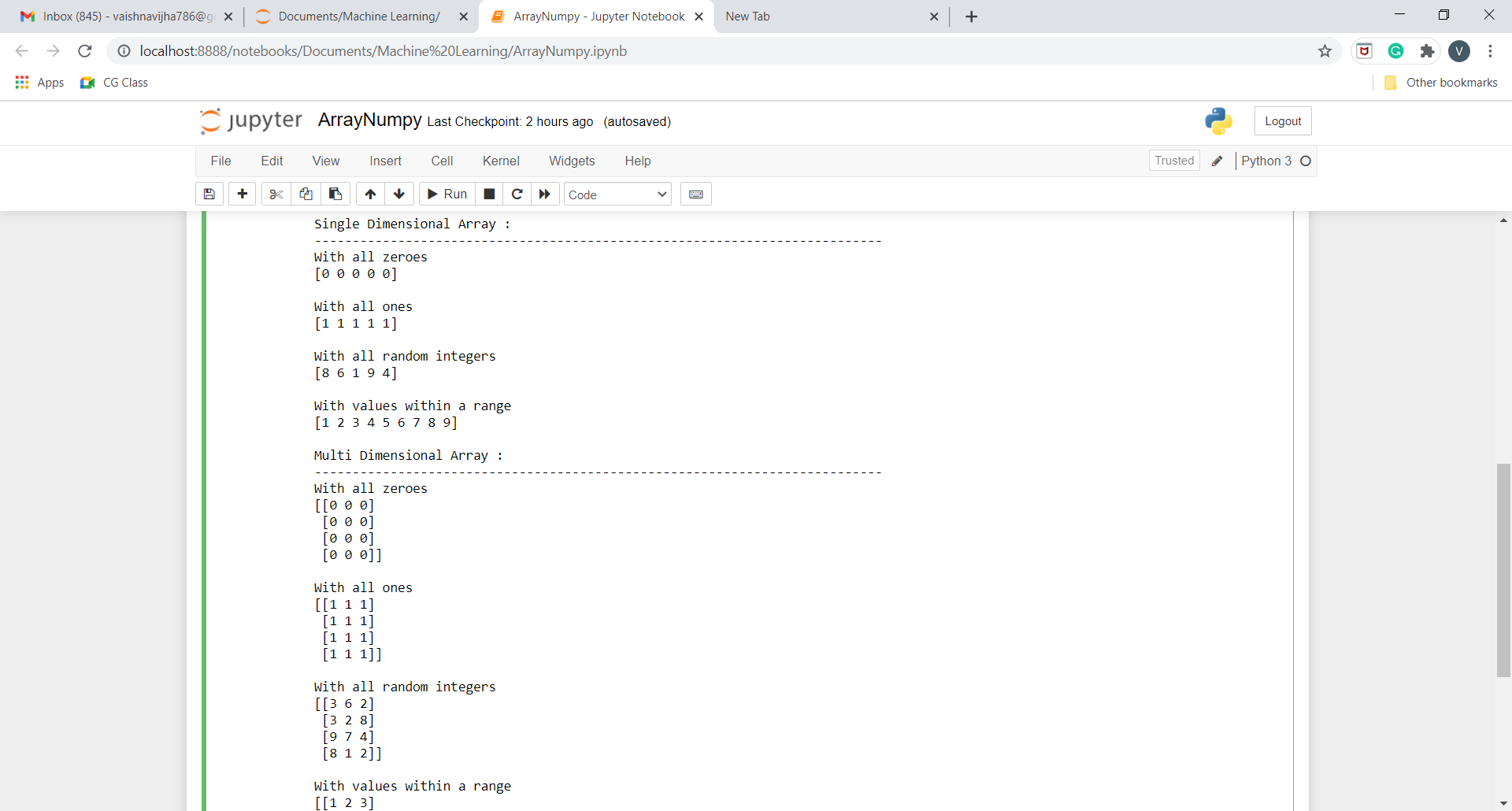
print()

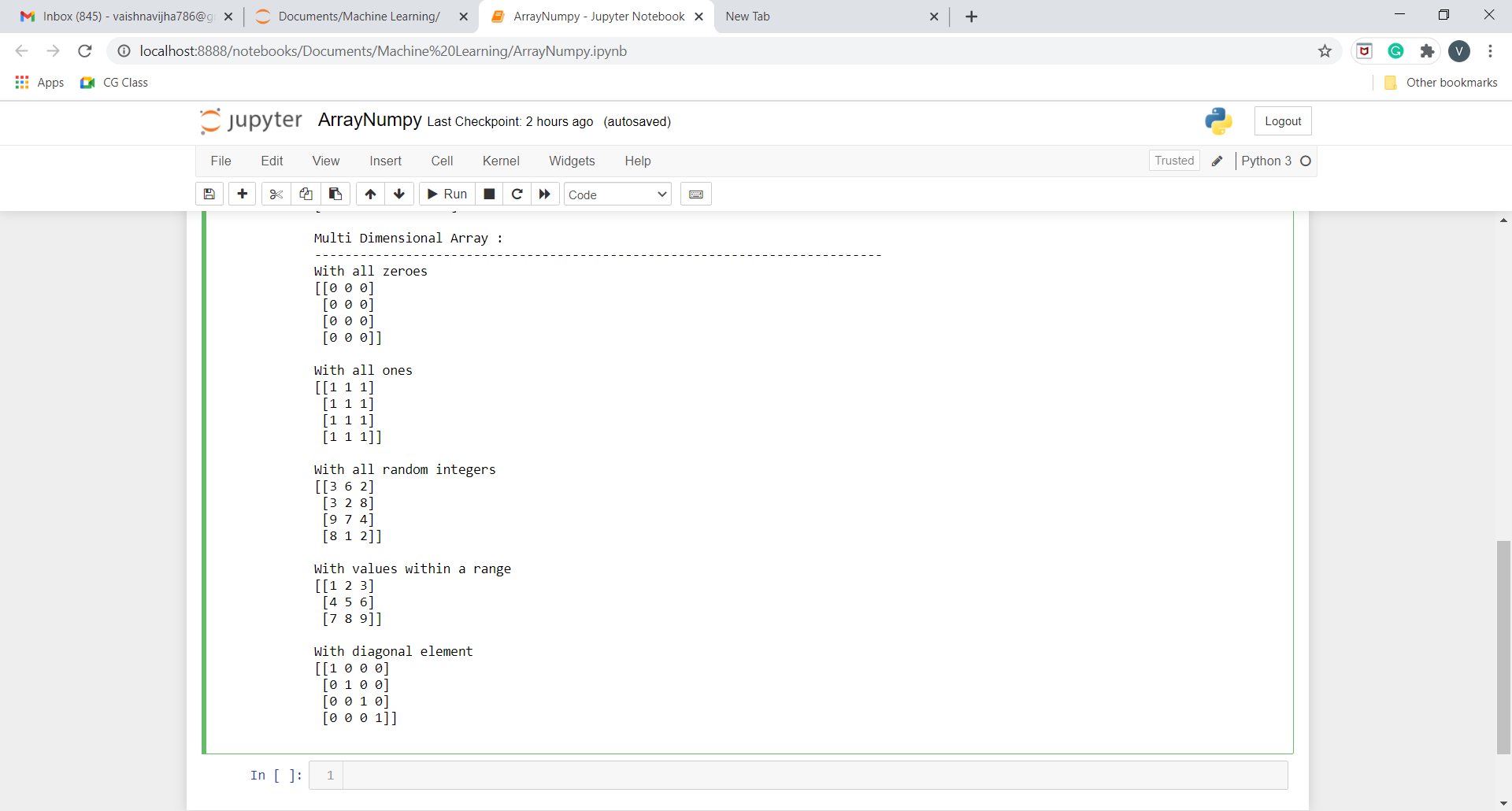
print('With diagonal element')

print(np.eye(4,None,0,int))

print()

**Output:**





1. **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.**

**Program:**

import numpy as np

arr=np.random.randn(4,3)

print('Array:')

print('---------------------------------------------------------------------------')

print(arr)

print()

print('Size of Array:')

print('---------------------------------------------------------------------------')

print(arr.size)

print()

print('Size of particular row (row 2):')

print('---------------------------------------------------------------------------')

print(len(arr[:,2]))

print()

print('Size of particular column (column 2):')

print('---------------------------------------------------------------------------')

print(len(arr[2,:]))

print()

print('Taking input from file: ')

print('---------------------------------------------------------------------------')

with open('numpy2text.txt', 'r') as f:

l =np.loadtxt(f)

print(l)

print()

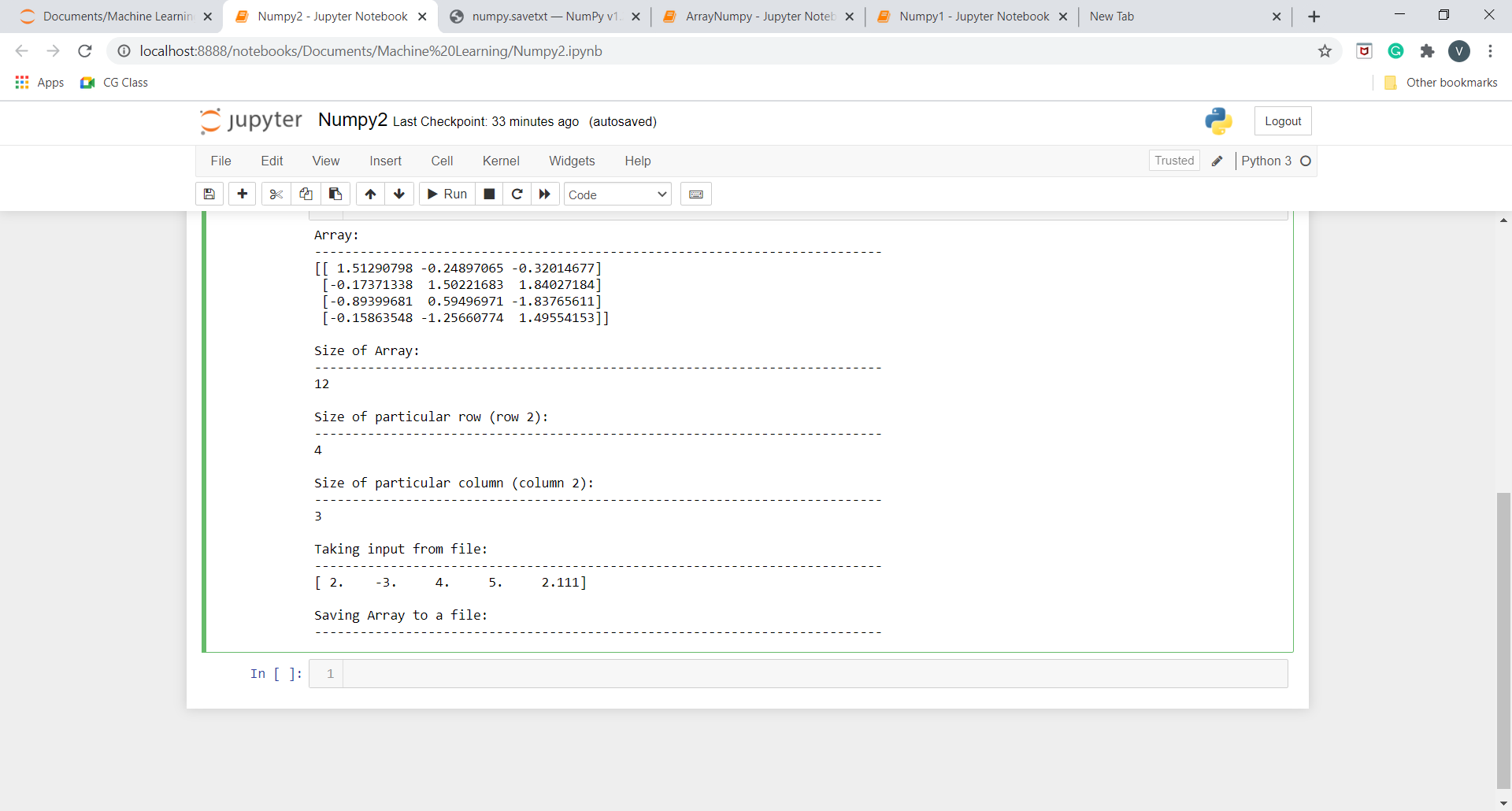
print('Saving Array to a file: ')

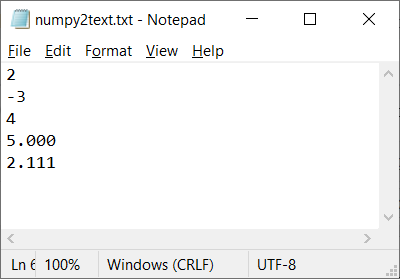
print('---------------------------------------------------------------------------')

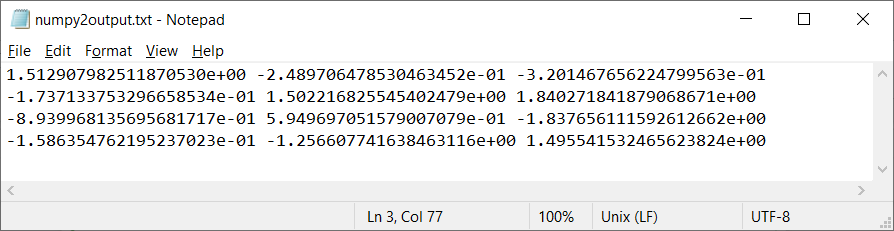
with open('numpy2output.txt', 'wb') as f:

np.savetxt(f,arr)

**Output:**







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

**Program:**

import numpy as np

mat1= np.random.randint(1,10,(3,3))

mat2= np.random.randint(1,10,(3,3))

print("Matrix 1:")

print(mat1)

print("Matrix 2:")

print(mat2)

print()

print('Addition of Two matrices')

print('---------------------------------------------------------------------------')

mat3=mat1+mat2

print(mat3)

print()

print('Subtraction of Two matrices')

print('---------------------------------------------------------------------------')

mat3=mat1+mat2

print(mat3)

print()

print('Multiplication of Two matrices')

print('---------------------------------------------------------------------------')

mat3=mat1\*mat2

print(mat3)

print()

mat1= np.random.randint(1,10,(4,3))

mat2= np.random.randint(1,10,(3,2))

print("Matrix 1:")

print(mat1)

print("Matrix 2:")

print(mat2)

print()

print('Element-by-Element Multiplication of Two matrices')

print('---------------------------------------------------------------------------')

mat3=mat1.dot(mat2)

print(mat3)

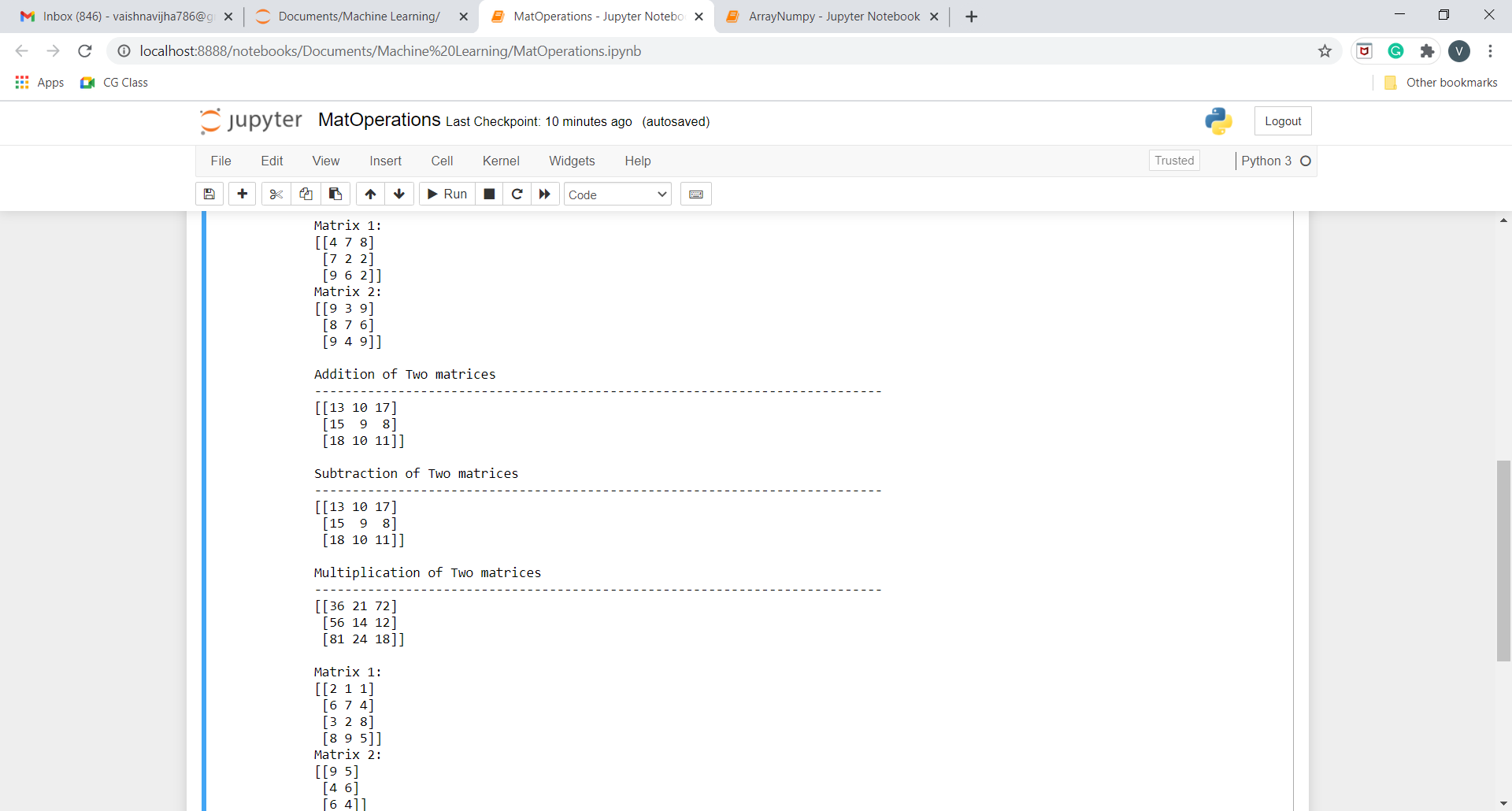
print()

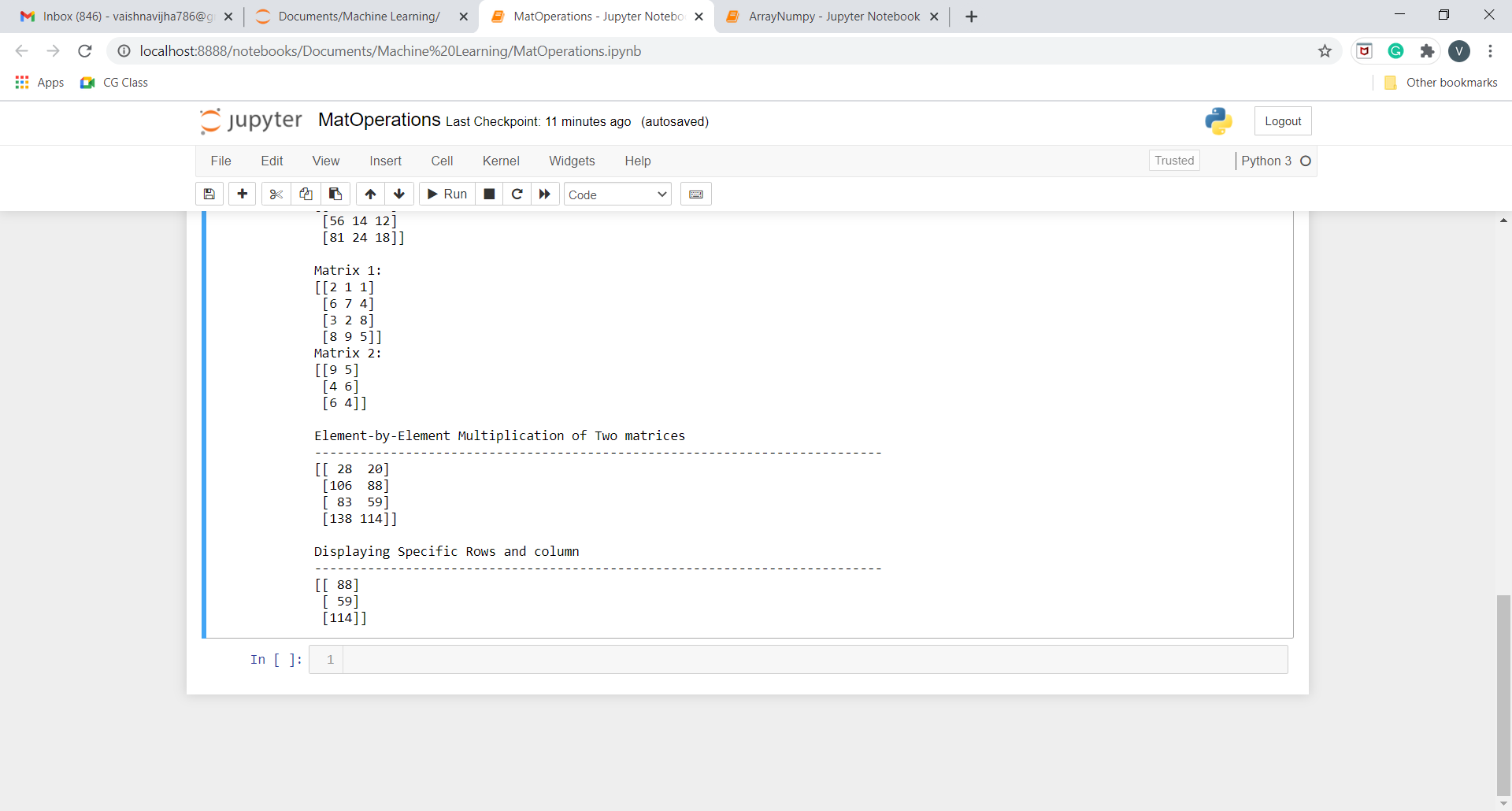
print('Displaying Specific Rows and column')

print('---------------------------------------------------------------------------')

print(mat3[1:,1:])

**Output:**





1. **Perform other matrix operations like converting matrix data to absolute values, taking the negative of matrix values, additing/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.**

**Program:**

import numpy as np

arr=np.random.randn(4,3)

print('Array:')

print('---------------------------------------------------------------------------')

print(arr)

print()

print('Absolute Value of the Array')

print('---------------------------------------------------------------------------')

print(np.absolute(arr))

print()

print('Negative Value of the Array')

print('---------------------------------------------------------------------------')

print(np.negative(arr))

print()

print('Array after deleting row 2')

print('---------------------------------------------------------------------------')

print(np.delete(arr,2,0))

print()

print('Array after deleting column 2')

print('---------------------------------------------------------------------------')

print(np.delete(arr,2,1))

print()

print('Maximum of Array')

print('---------------------------------------------------------------------------')

print(np.max(arr))

print()

print('Minimum of Array')

print('---------------------------------------------------------------------------')

print(np.min(arr))

print()

print('Sum of all elements of Array')

print('---------------------------------------------------------------------------')

print(np.sum(arr))

print()

print('Sum of all elements of Array row-wise')

print('---------------------------------------------------------------------------')

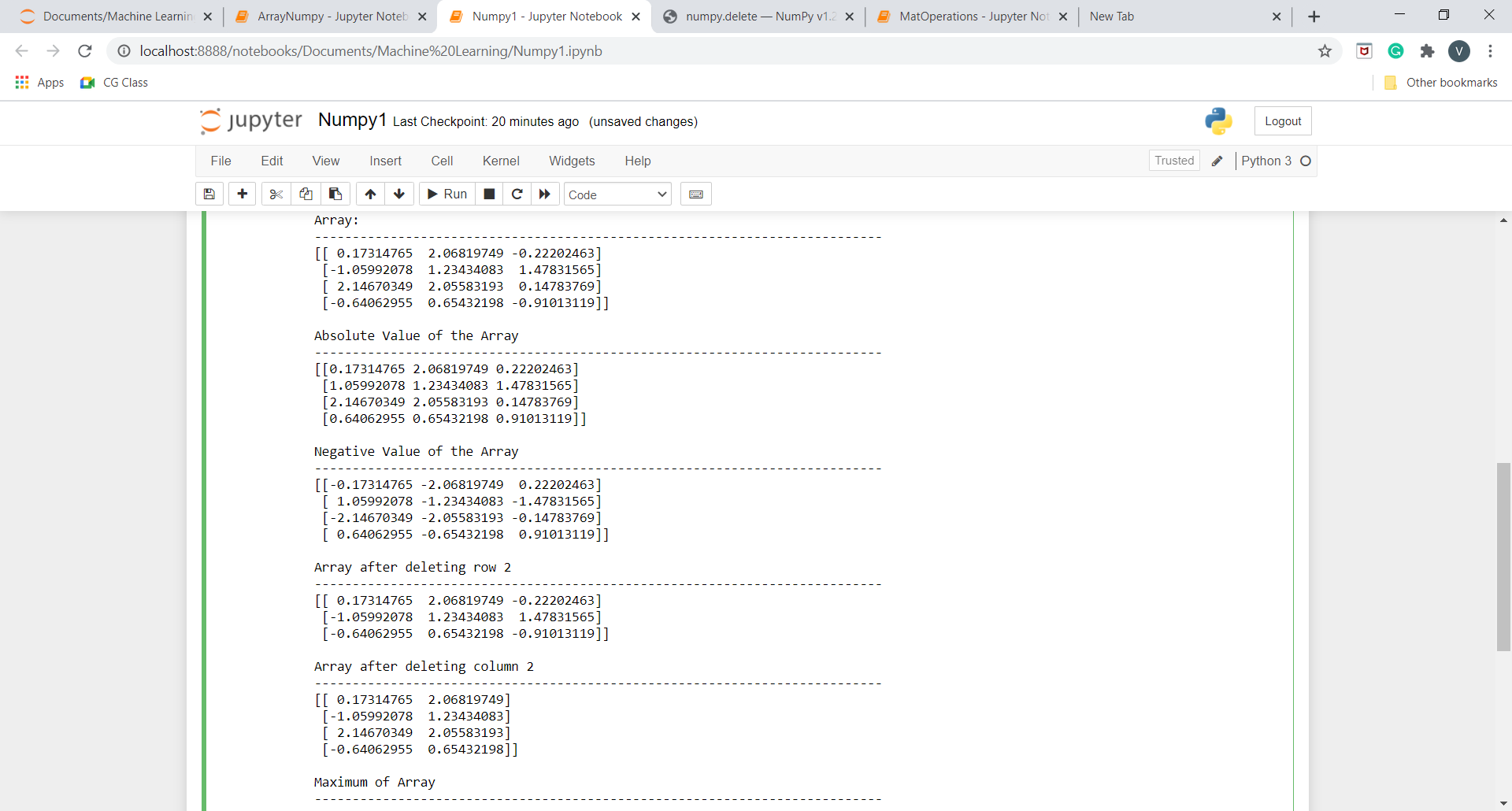
print(np.sum(arr,0))

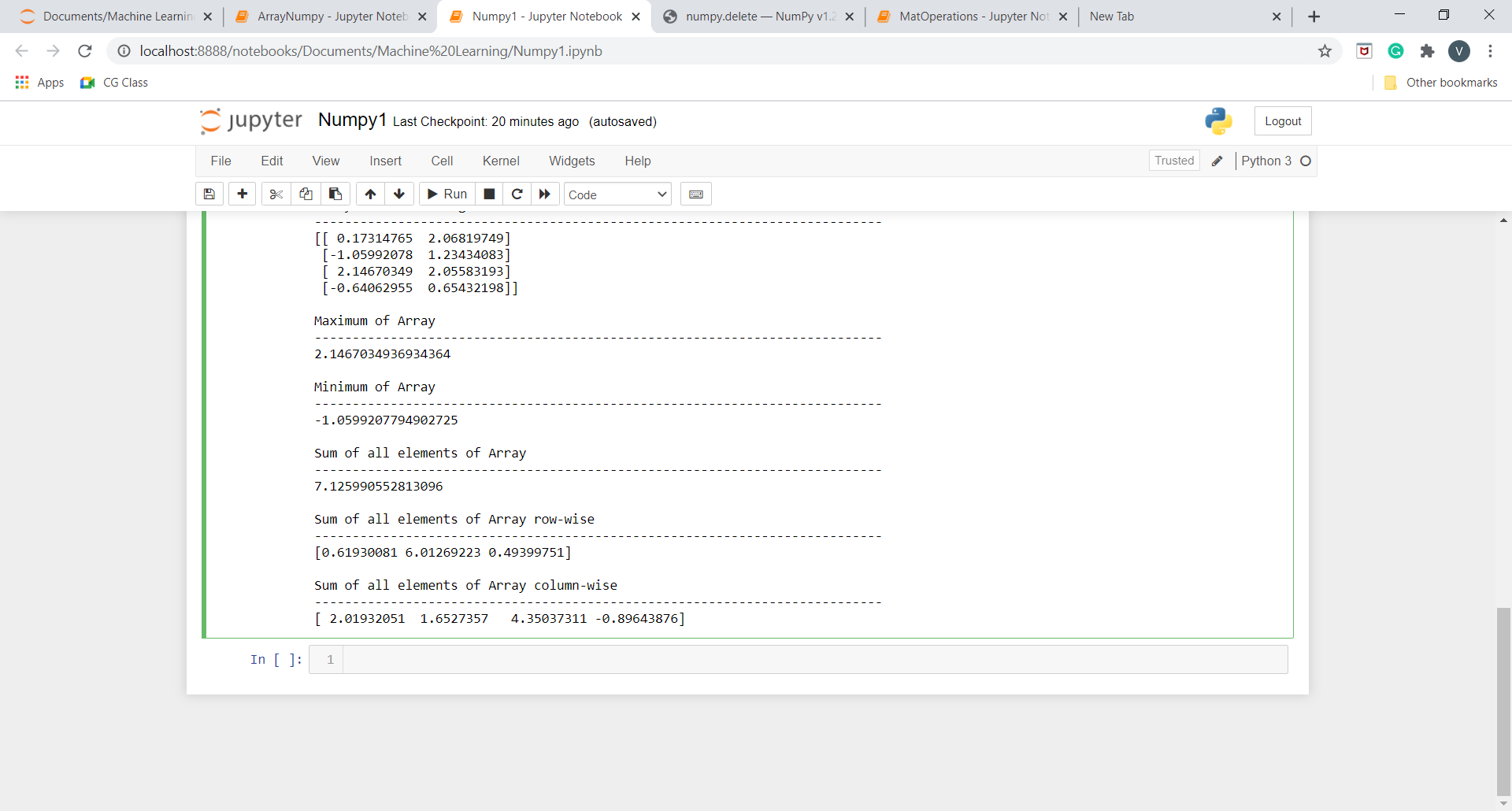
print()

print('Sum of all elements of Array column-wise')

print('---------------------------------------------------------------------------')

print(np.sum(arr,1))  
  
**Output:**





1. **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.**
2. **Generate different subplots from a given plot and color plot data.**

**Program:**

#!/usr/bin/env python

# coding: utf-8

# In[2]:

import matplotlib.pyplot as plt

import numpy as np

# In[15]:

x=np.arange(40)

# In[16]:

x

# In[17]:

plt.plot(x)

plt.show()

# In[18]:

y=np.sin(x)

plt.plot(x)

plt.plot(y)

plt.show()

# In[19]:

y=np.cos(x)

plt.plot(x,color='green')

plt.plot(y)

plt.show()

# In[20]:

x=np.array([1,2,3,4,5])

y=x\*\*2

plt.scatter(x,y,color="orange", label="squares of number", marker="o") #color, label, marker are optional arguments

plt.legend()

plt.xlabel("numbers")

plt.ylabel("squares")

plt.show()

# In[21]:

x=np.array([1,2,3,4,5])

plt.plot(x,x\*\*2,label="squares of number", marker="o")

plt.plot(x,x\*\*3,label="cubes of number", marker="o")

plt.legend()

plt.xlabel("numbers")

plt.title("Squares and Cubes")

plt.show()

# In[22]:

numbers=np.random.randint(0,10,5)

indices=np.arange(5)

indices=indices+2015

numbers2=np.random.randint(0,10,5)

indices2=np.arange(5)

indices2=indices2+2015

print(numbers)

print(numbers2)

# In[23]:

plt.bar(indices,numbers,0.25,color="red",label="rainfall")

plt.bar(indices2+0.25,numbers2,0.25,color="green",label="humidity")

plt.legend()

plt.show()

# In[4]:

labels=["english","hindi","maths","science","social science","computers"]

values=[90,80,40,73,78,43]

plt.pie(values,labels=labels,radius=1)

plt.show()

plt.pie(values,labels=labels,radius=1,explode=(0,0,1,0,0,0),shadow=True)

plt.show()

# In[24]:

u=5

sigma=2

vals=u+sigma\*np.random.randn(1000)

print(vals.shape)

plt.hist(vals,50)

plt.show()

# In[26]:

plt.figure(figsize=(8,8))

plt.subplot(221)

plt.plot(x,x\*\*2, label="2018")

plt.plot(x,x\*\*3, label="2017")

plt.subplot(222)

plt.hist(vals,50)

plt.subplot(223)

plt.pie(values,labels=labels,radius=1)

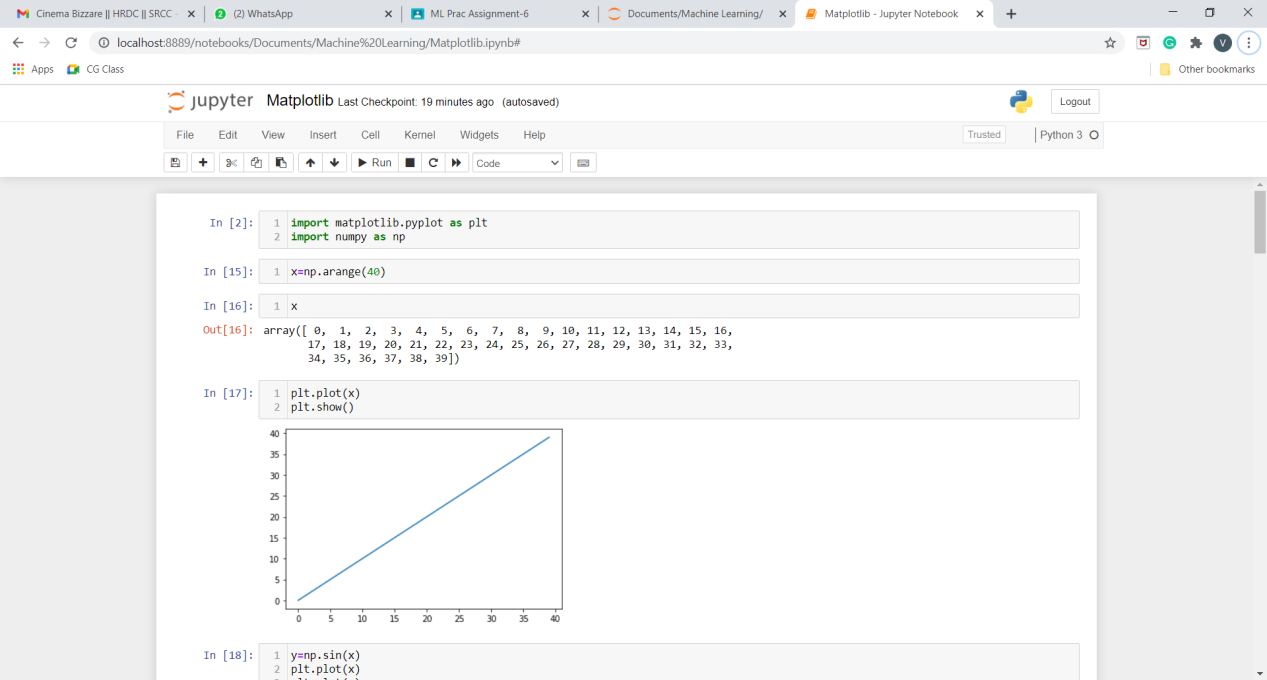
plt.subplot(224)

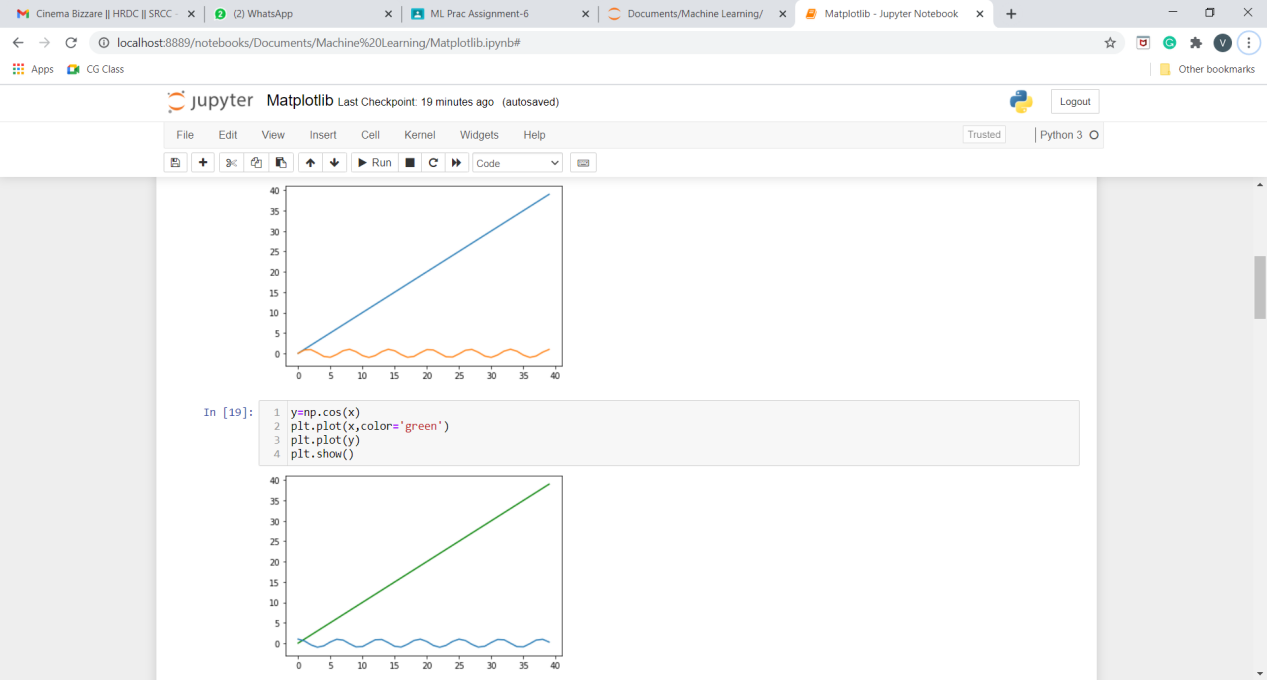
plt.pie(values,labels=labels,radius=1,explode=(0,0,1,1,0,1),shadow=True)

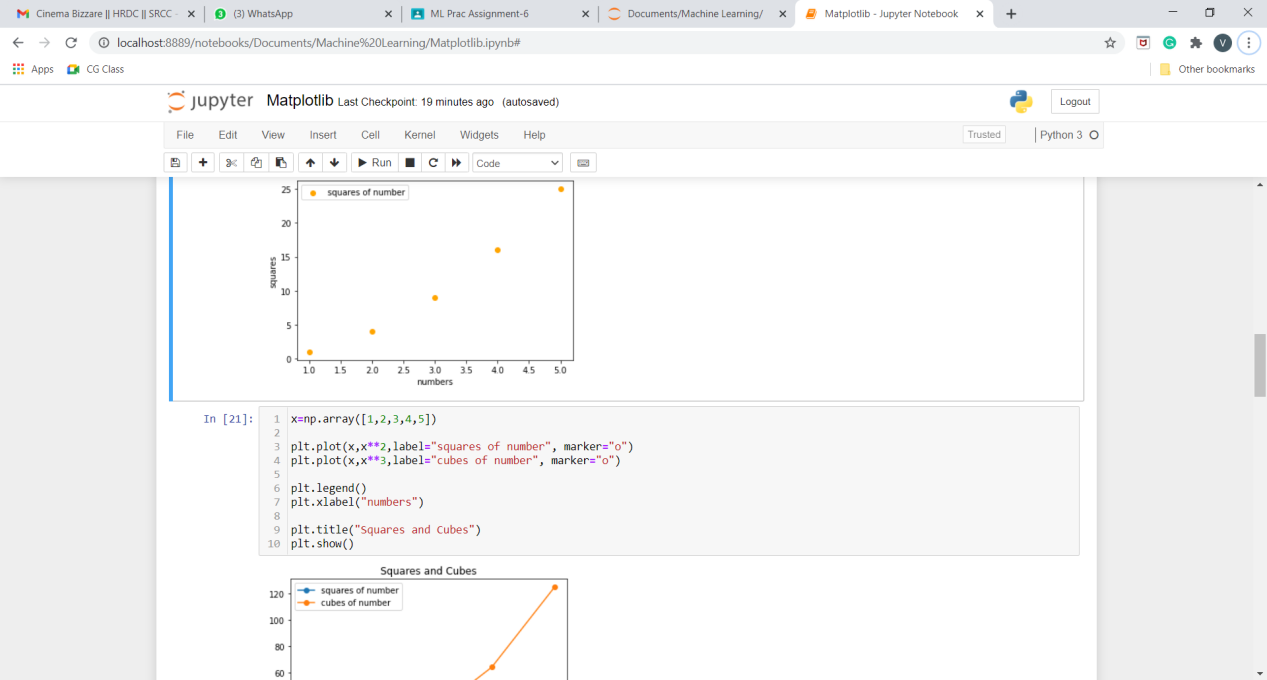
plt.show()

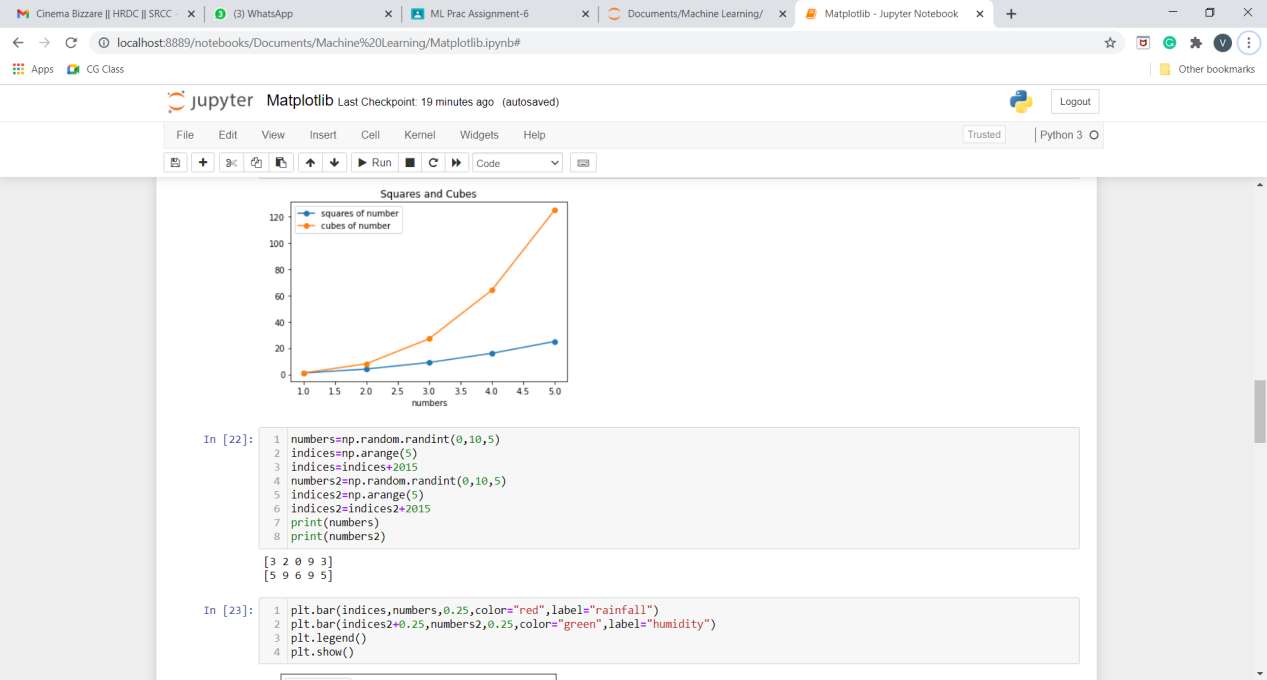
# In[ ]:

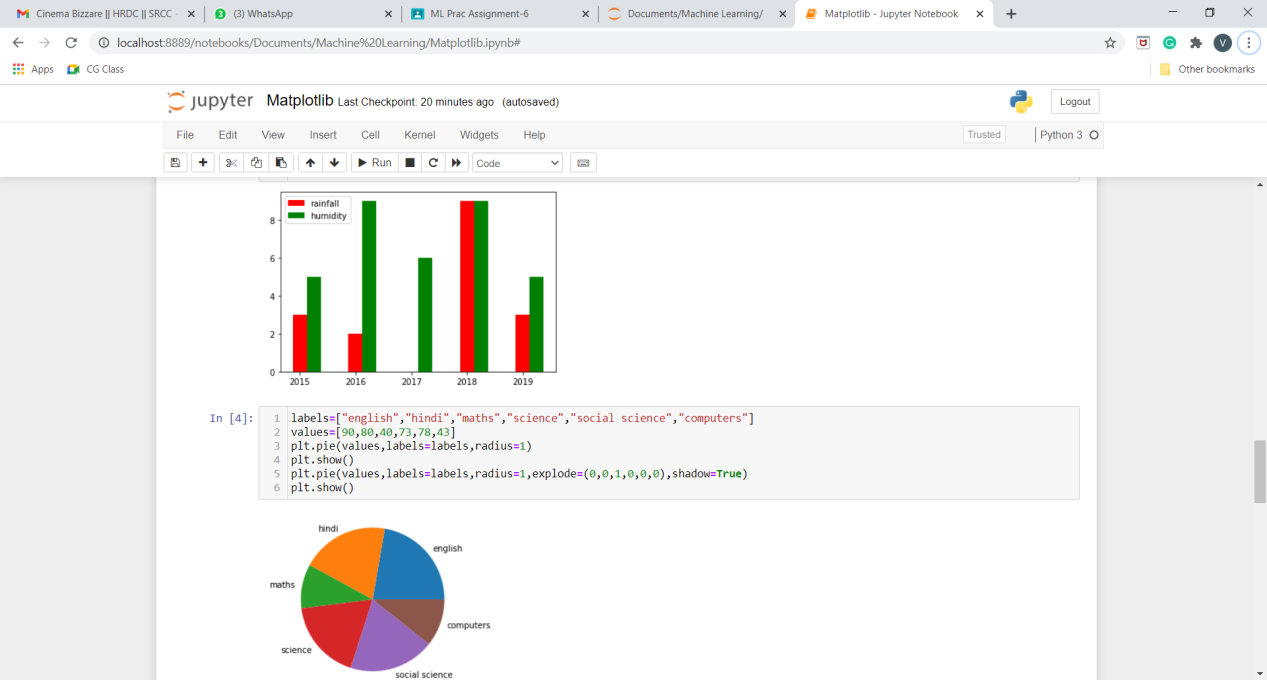
**Output:**

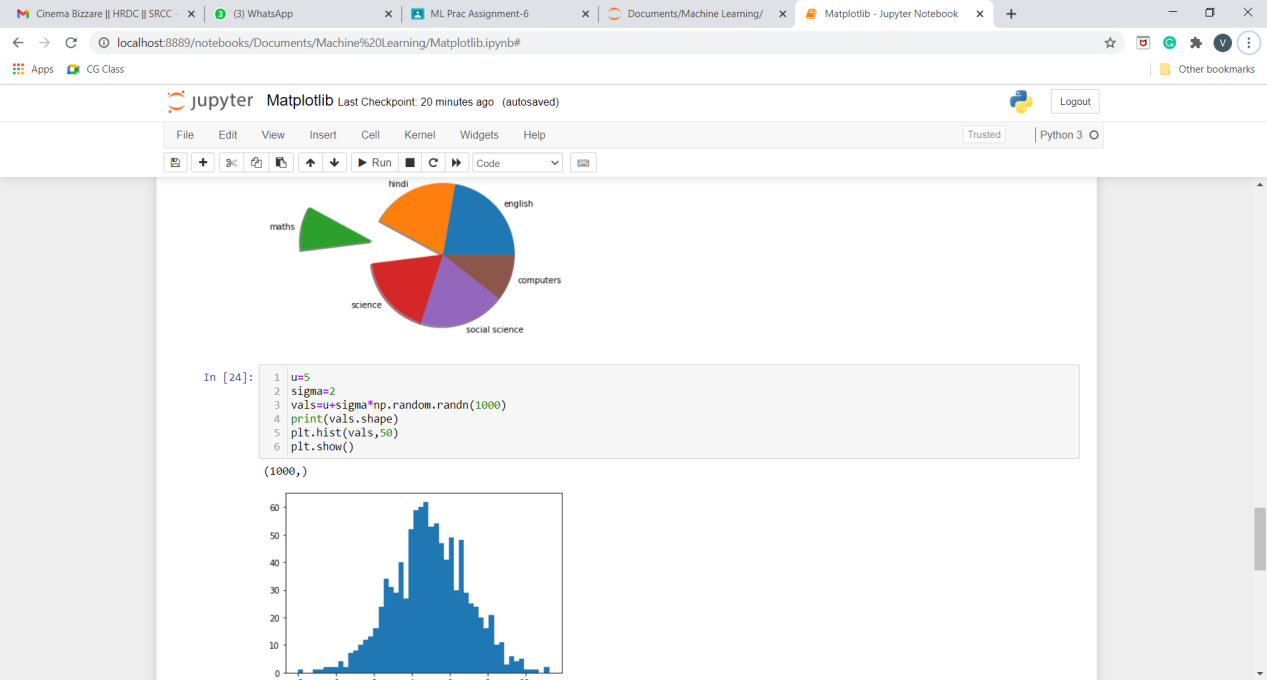


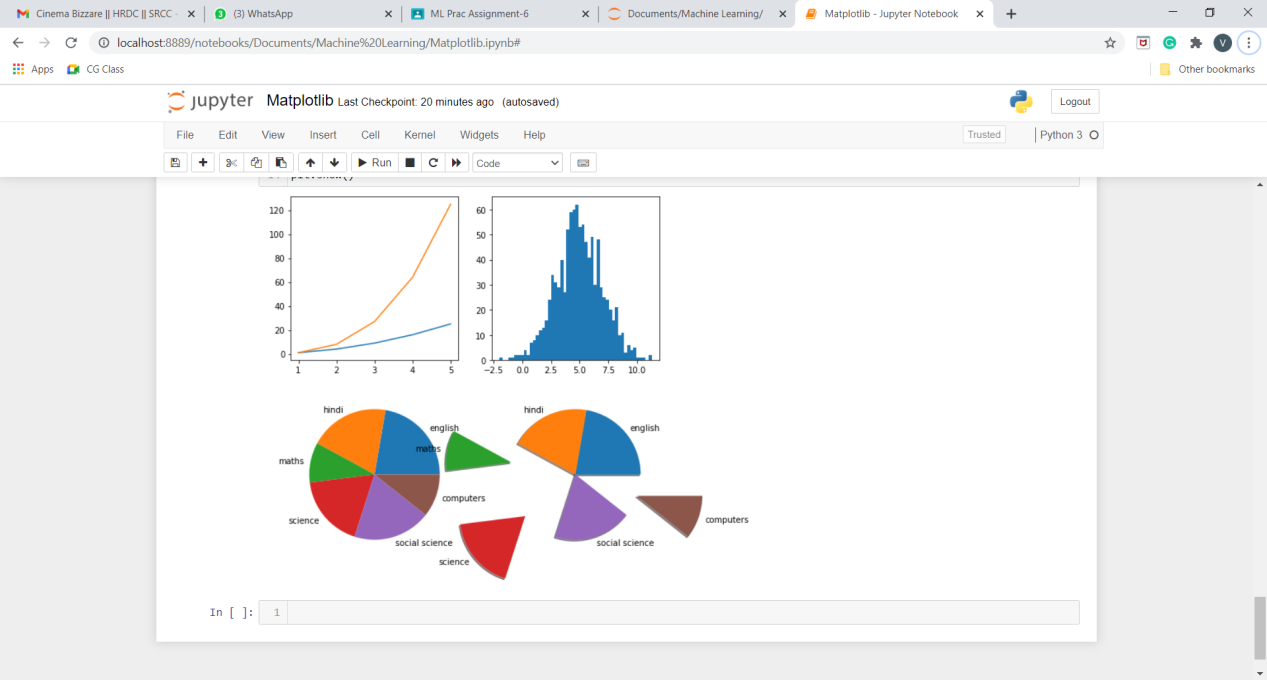












1. **Use conditional statements and different type of loops based on simple example/s. (Perform matrix multiplication using nested loops)**

**Program:**

A = [[1, 2, 3],

[4, 5, 6],

[7, 8, 9]]

B = [[10, 11, 12, 13],

[14, 15, 16, 17],

[18, 19, 20, 21]]

result = [[0, 0, 0, 0],

[0, 0, 0, 0],

[0, 0, 0, 0]]

for i in range(len(A)):

for j in range(len(B[0])):

for k in range(len(B)):

result[i][j] += A[i][k] \* B[k][j]

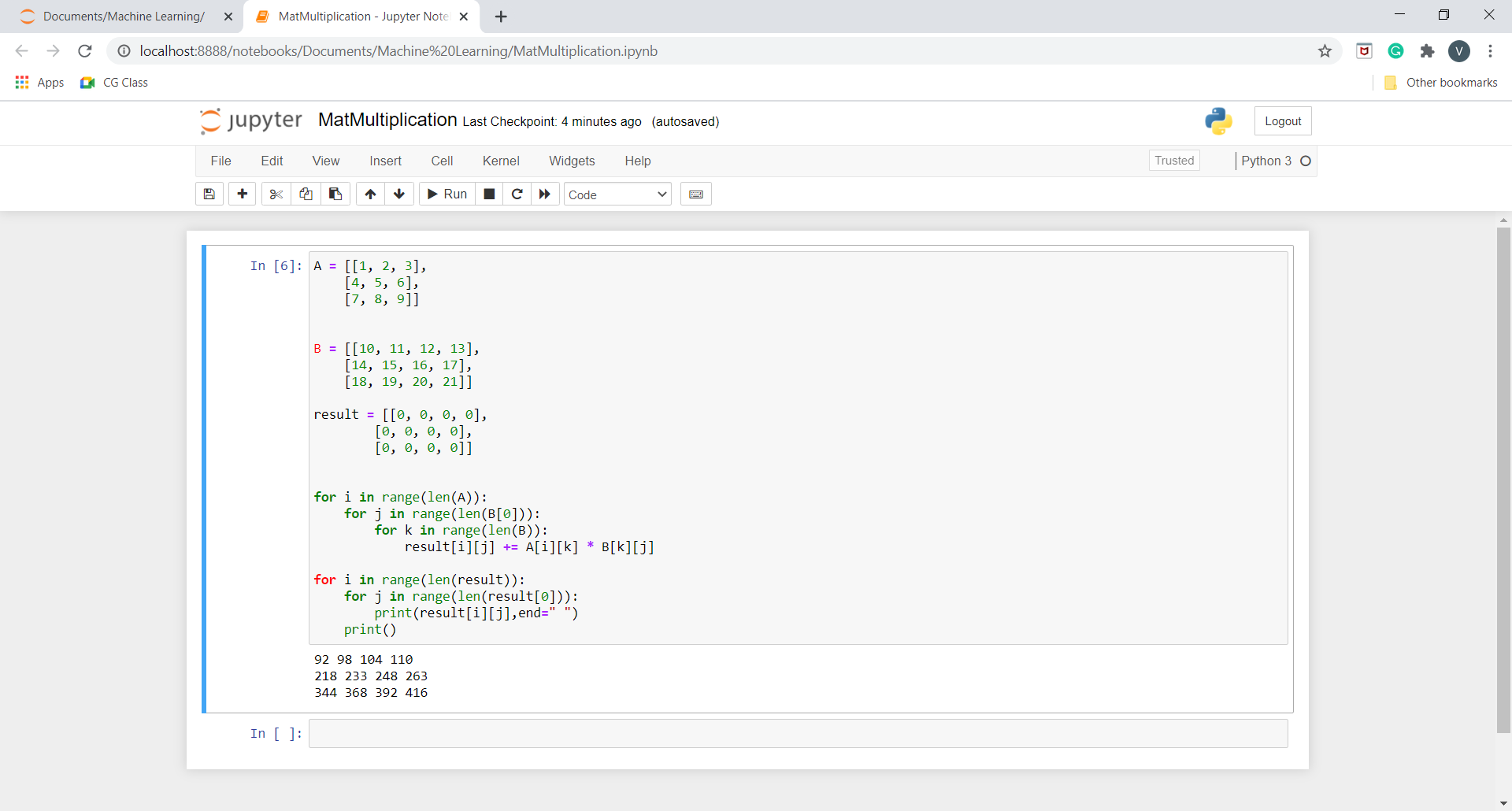
for i in range(len(result)):

for j in range(len(result[0])):

print(result[i][j],end=" ")

print()

**Output:**



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

**Program:**

1. **Plotting of Data:**

**Commands:**

>>> import matplotlib.pyplot as plt

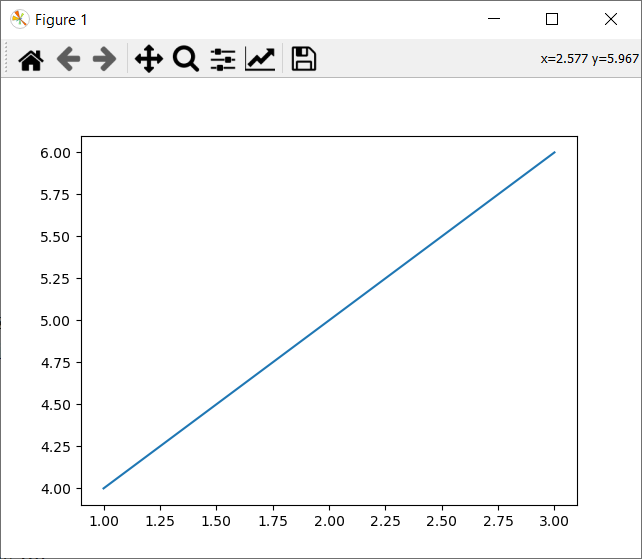
>>> plt.plot([1,2,3],[4,5,6])

[<matplotlib.lines.Line2D object at 0x00000224330A0790>]

>>> plt.draw()

>>> plt.show()

**Output:**



1. **Vectorization:**

**Commands:**

>>> import numpy as np

>>> #converting a tuple into a vector

>>> #using numpy methods to convert into a array

>>> #1-D array

>>> type((1,2,3,4,5))

<class 'tuple'>

>>> x=np.array((1,2,3,4,5))

>>> x

array([1, 2, 3, 4, 5])

>>> type(x)

<class 'numpy.ndarray'>

>>> #2-D array

>>> y=np.random.randn(3,4)

>>> y

array([[ 0.73970731, -0.11279598, 0.26243823, -0.77851299],

[-0.53945554, -0.95227891, -0.41367667, 0.08030448],

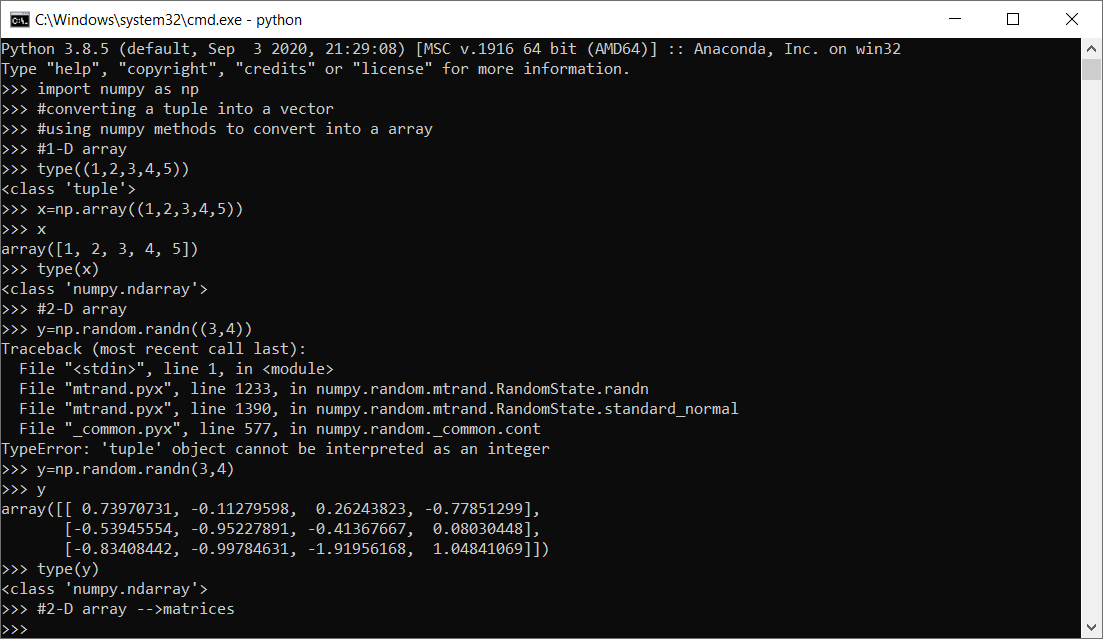
[-0.83408442, -0.99784631, -1.91956168, 1.04841069]])

>>> type(y)

<class 'numpy.ndarray'>

>>> #2-D array -->matrices

>>>



1. **Operations on Vectors:**

**Commands:**

>>> import numpy as np

>>> x=np.array([1,2,3])

>>> y=np.array([9,8,7])

>>> #addition

>>> x+y

array([10, 10, 10])

>>> #subtraction

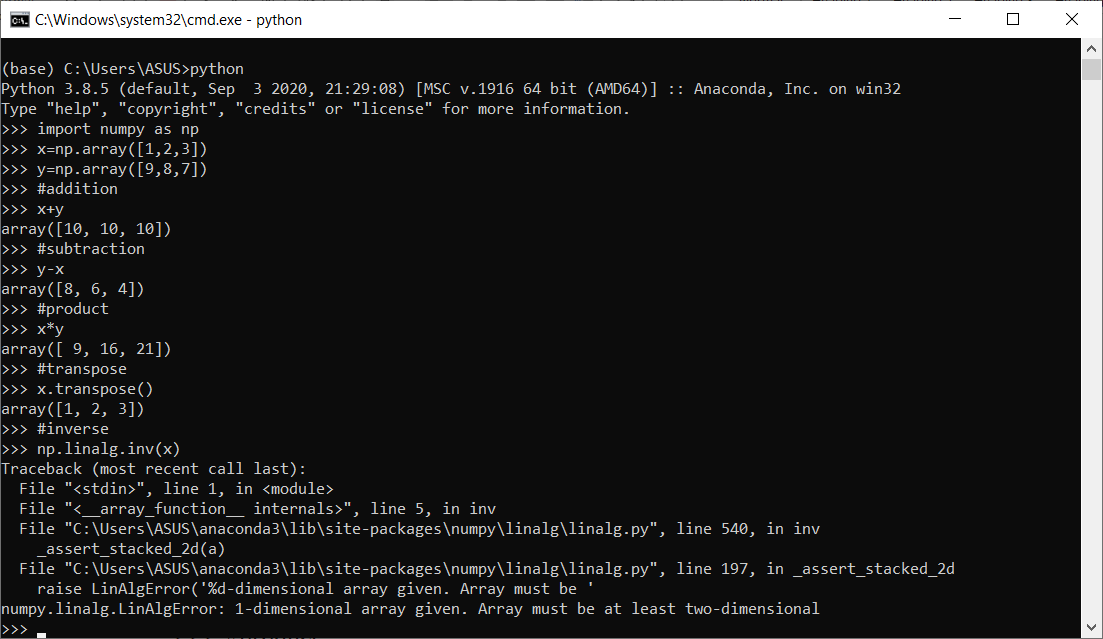
>>> y-x

array([8, 6, 4])

>>> #product

>>> x\*y

array([ 9, 16, 21])



1. **Operations on matrices**

>>> mat1=np.random.randint(1,10,(3,3))

>>> mat2=np.random.randint(1,10,(3,3))

>>> mat1

array([[6, 3, 8],

[3, 4, 5],

[9, 3, 6]])

>>> mat2

array([[3, 7, 6],

[6, 8, 6],

[6, 7, 4]])

>>> #addition

>>> mat1+mat2

array([[ 9, 10, 14],

[ 9, 12, 11],

[15, 10, 10]])

>>> #subtration

>>> mat1-mat2

array([[ 3, -4, 2],

[-3, -4, -1],

[ 3, -4, 2]])

>>> #product

>>> mat1\*mat2

array([[18, 21, 48],

[18, 32, 30],

[54, 21, 24]])

>>> #matrix multipliaction

>>> mat1.dot(mat2)

array([[ 84, 122, 86],

[ 63, 88, 62],

[ 81, 129, 96]])

>>> #transpose

>>> np.transpose(mat1)

array([[6, 3, 9],

[3, 4, 3],

[8, 5, 6]])

>>> np.transpose(mat2)

array([[3, 6, 6],

[7, 8, 7],

[6, 6, 4]])

>>> #inverse

>>> np.linalg.inv(mat1)

array([[-0.11111111, -0.07407407, 0.20987654],

[-0.33333333, 0.44444444, 0.07407407],

[ 0.33333333, -0.11111111, -0.18518519]])

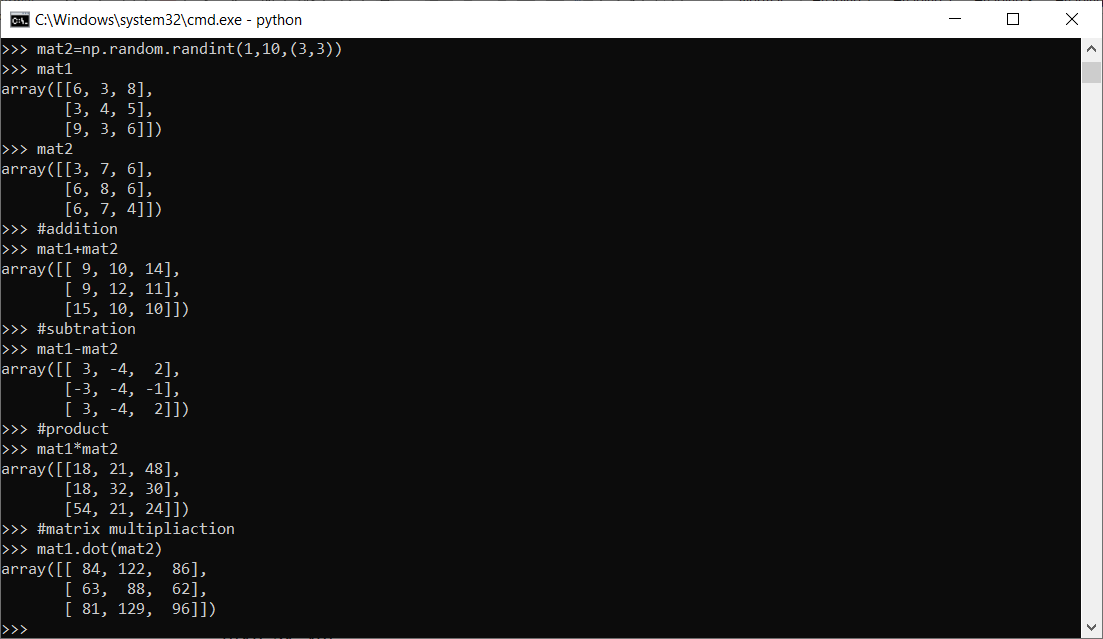
>>> np.linalg.inv(mat2)

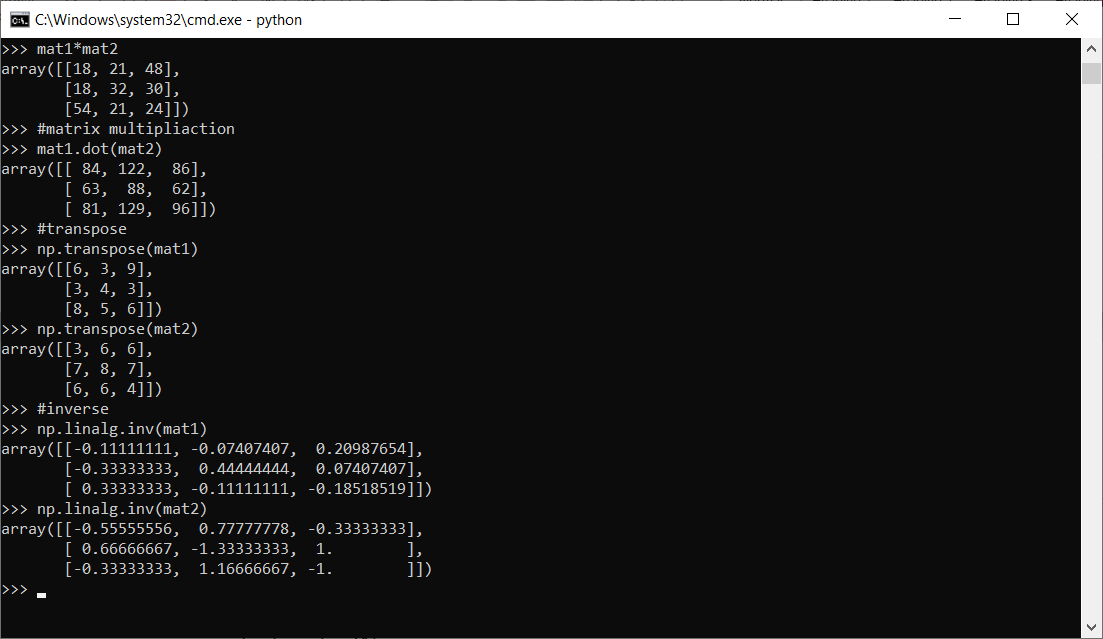
array([[-0.55555556, 0.77777778, -0.33333333],

[ 0.66666667, -1.33333333, 1. ],

[-0.33333333, 1.16666667, -1. ]])

>>>





1. **Implement Simple Linear Regression on Advertising.csv.**

**Hint : Use functions from sklearn.linear\_model, statsmodels.api**

**Program:**

#!/usr/bin/env python

# coding: utf-8

# In[2]:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

import statsmodels.api as sm

# In[3]:

data = pd.read\_csv(r"Advertising.csv")

data

data.columns

# In[18]:

plt.figure(figsize=(16, 8))

plt.scatter(

data['TV'],

data['sales']

)

plt.xlabel("TV ")

plt.ylabel("Sales ")

plt.show()

# In[19]:

X = data['TV'].values.reshape(-1,1)

y = data['sales'].values.reshape(-1,1)

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.3,random\_state=0)

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

reg = LinearRegression()

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

# In[20]:

print(reg.coef\_[0][0])

print(reg.intercept\_[0])

print("The linear model is: Y = {:.5} + {:.5}X".format(reg.intercept\_[0], reg.coef\_[0][0]))

# In[24]:

predictions = reg.predict(X\_test)

plt.figure(figsize=(16, 8))

plt.scatter(

data['TV'],

data['sales']

)

plt.plot(

X\_test,

predictions,

linewidth=2,

color='red'

)

plt.xlabel("TV ")

plt.ylabel("Sales ")

plt.show()

# In[13]:

X=X\_train

y=y\_train

X2 = sm.add\_constant(X)

est = sm.OLS(y, X2)

est2 = est.fit()

print(est2.summary())

# In[14]:

print('Train Score :', reg.score(X\_train,y\_train))

print('Test Score:', reg.score(X\_test,y\_test))

# In[15]:

from sklearn import metrics

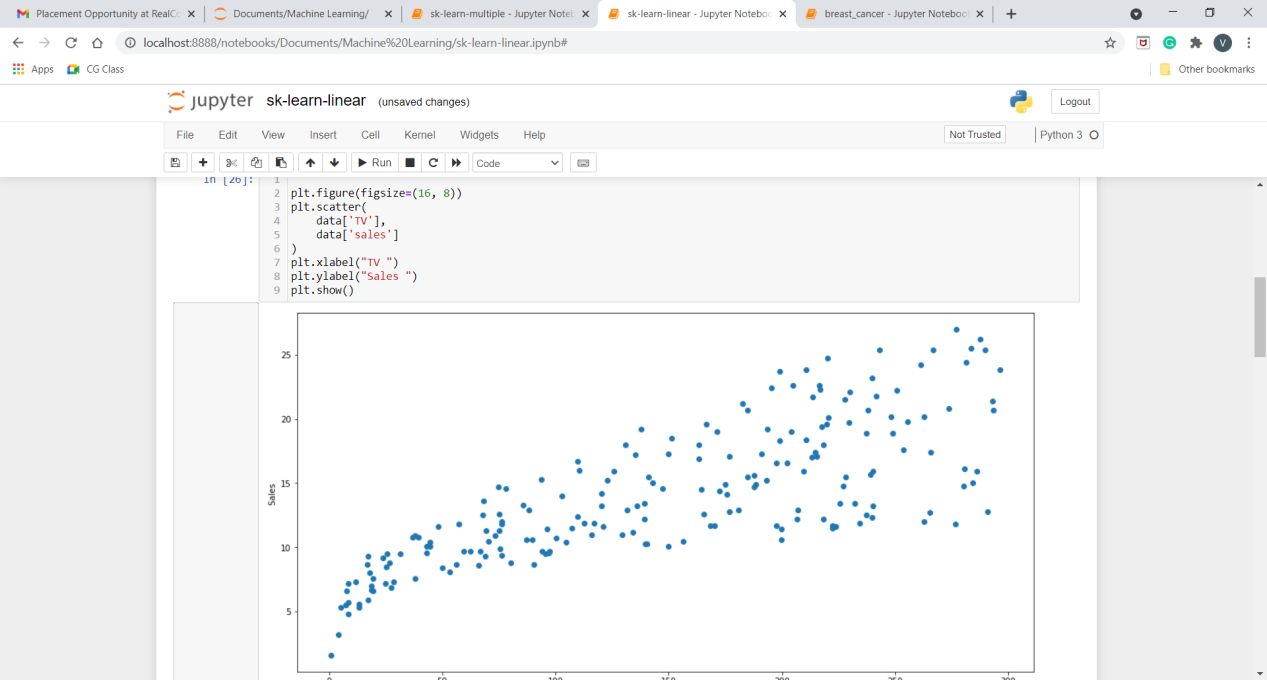
print('MSE :', metrics.mean\_squared\_error(y\_test,predictions))

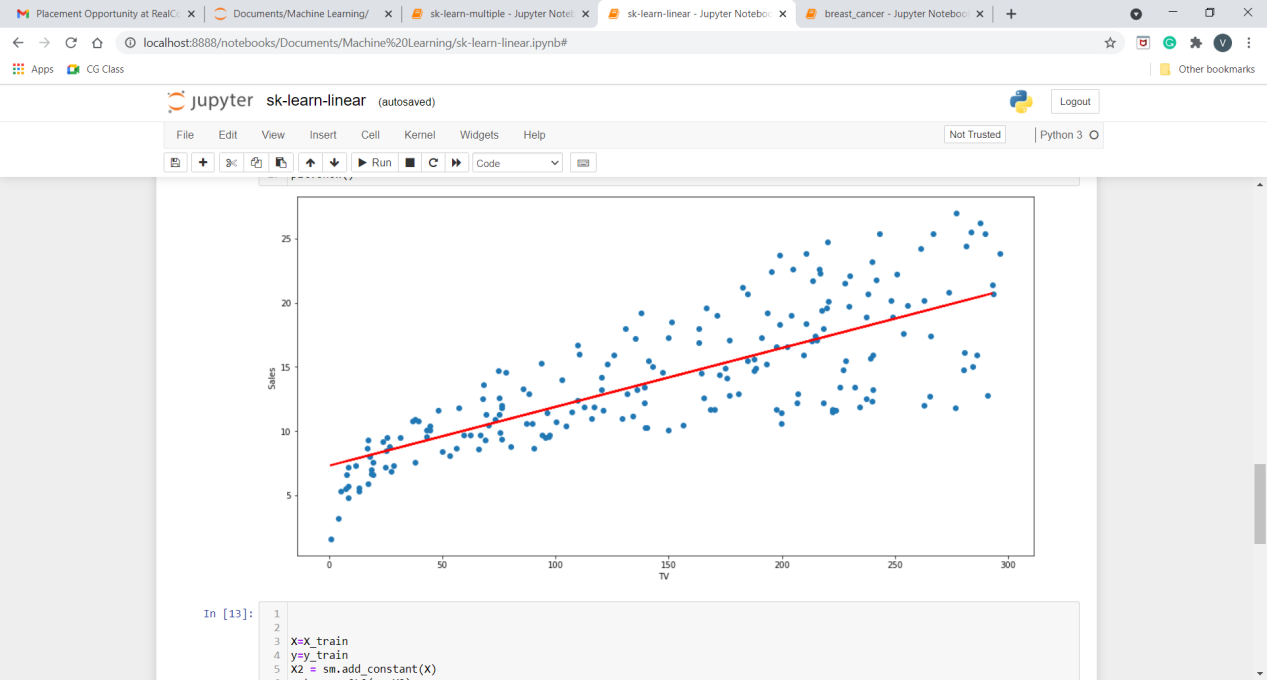
print('RMSE :', np.sqrt(metrics.mean\_squared\_error(y\_test,predictions)))

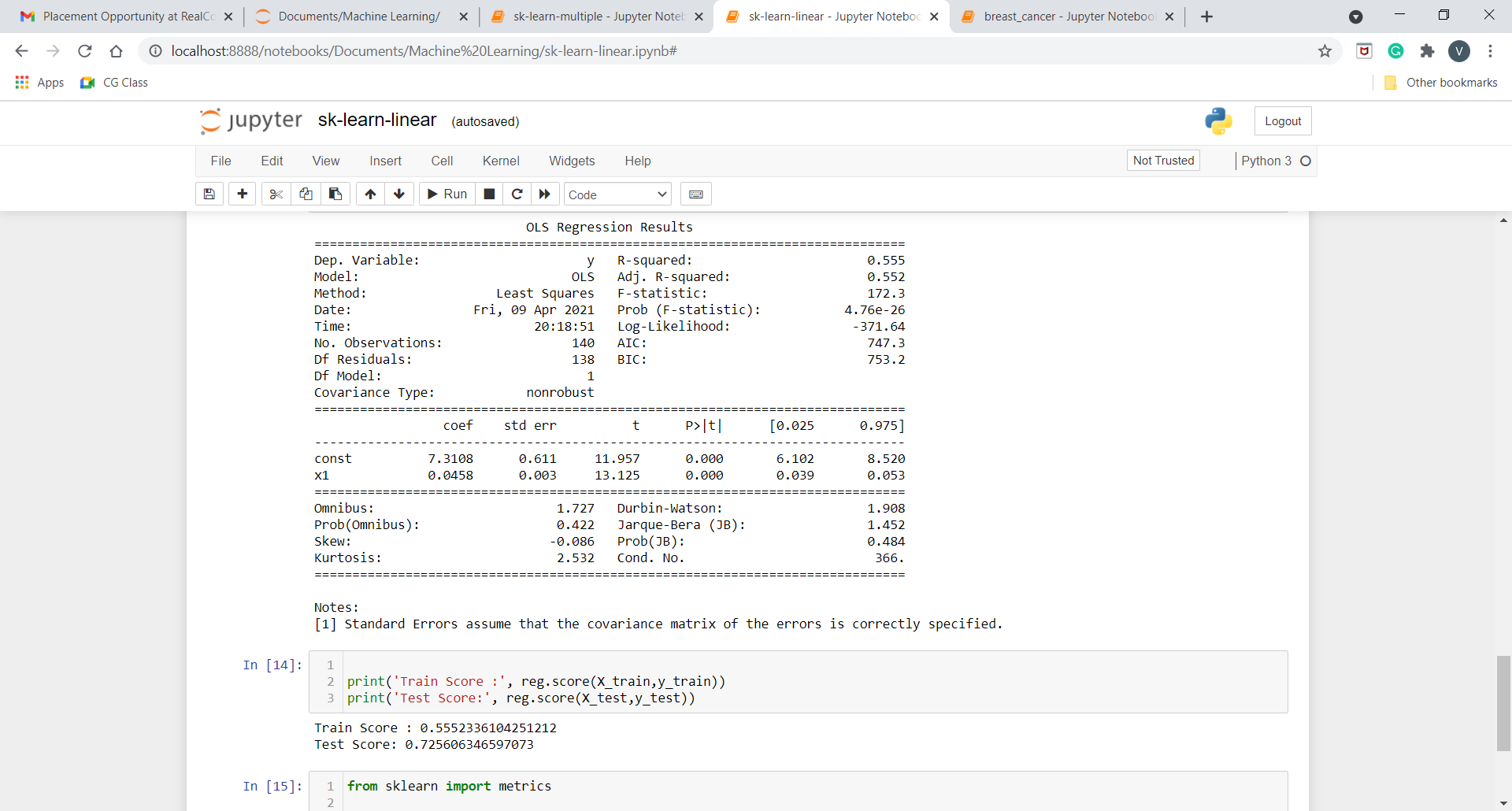
# In[ ]:

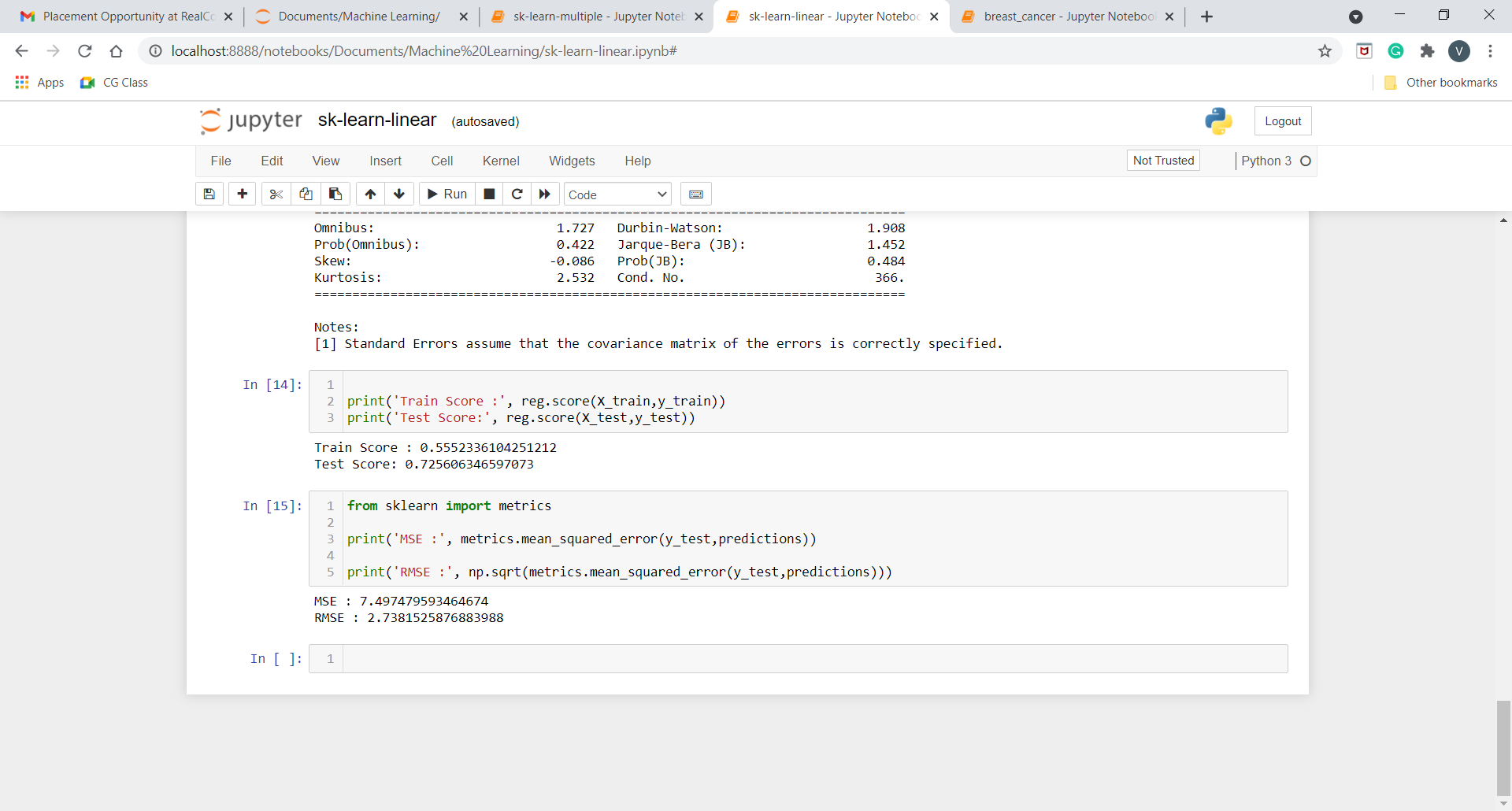
**Output:**











**13. Implement Multiple Linear Regression on Advertising.csv.**

**Hint : Use functions from sklearn.linear\_model, statsmodels.api**

**Program:**

#!/usr/bin/env python

# coding: utf-8

# In[2]:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

import statsmodels.api as sm

# In[6]:

data = pd.read\_csv(r"Advertising.csv")

# In[7]:

Xs = data.drop(['sales', 'Unnamed: 0'], axis=1)

y = data['sales'].values.reshape(-1,1)

reg = LinearRegression()

reg.fit(Xs, y)

print("The linear model is: Y = {:.5} + {:.5}\*TV + {:.5}\*radio + {:.5}\*newspaper".format(reg.intercept\_[0], reg.coef\_[0][0], reg.coef\_[0][1], reg.coef\_[0][2]))

# In[8]:

X = np.column\_stack((data['TV'], data['radio'], data['newspaper']))

y = data['sales']

X2 = sm.add\_constant(X)

est = sm.OLS(y, X2)

est2 = est.fit()

print(est2.summary())

**Output:**



1. **Implement Logistic Regression on built in dataset breast\_cancer from sklearn.datasets.**

**Program:**

#!/usr/bin/env python

# coding: utf-8

# In[45]:

import pandas as pd

from matplotlib import pyplot as plt

from sklearn.datasets import load\_breast\_cancer

# In[46]:

df = load\_breast\_cancer()

#df

# In[47]:

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

# In[48]:

#X\_train, X\_test, y\_train, y\_test = train\_test\_split(df[['age']],df.bought\_insurance,train\_size=0.9)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(df.data,df.target,test\_size=0.2, stratify=df.target,random\_state=42)

# In[49]:

X\_test.shape

# In[50]:

Y\_test.shape

# In[51]:

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(C=0.1)

# In[53]:

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

# In[54]:

#X\_test

# In[41]:

y\_predicted = model.predict(X\_test)

# In[42]:

print('Accuracy on the training subset: {:.3f}'.format(model.score(X\_train, Y\_train)))

print('Accuracy on the test subset: {:.3f}'.format(model.score(X\_test, Y\_test)))

# In[43]:

#X\_test

# In[44]:

from sklearn.metrics import confusion\_matrix

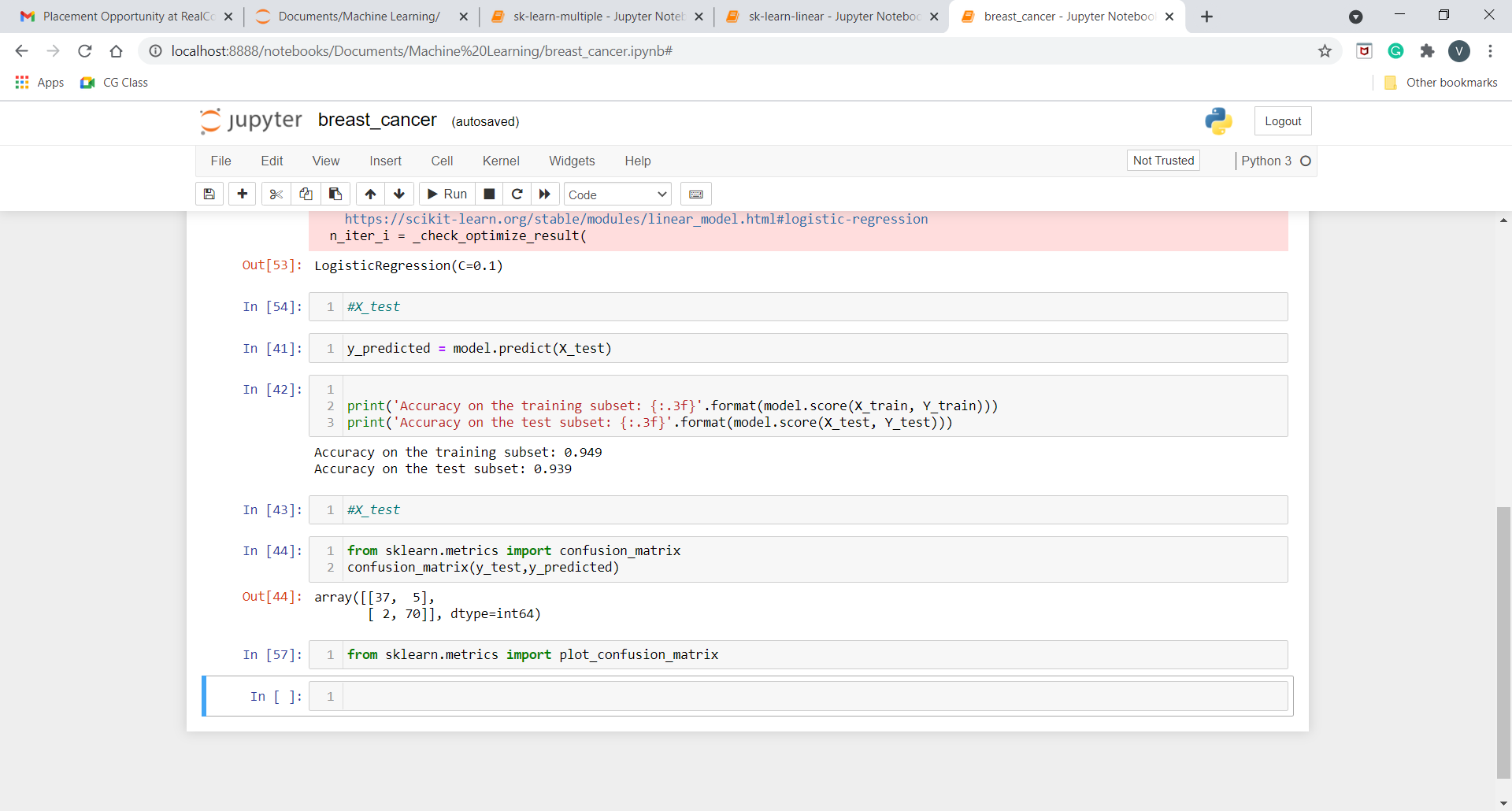
confusion\_matrix(y\_test,y\_predicted)

# In[57]:

from sklearn.metrics import plot\_confusion\_matrix

# In[ ]:

**Output:**



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

**Program:**

#!/usr/bin/env python

# coding: utf-8

# In[1]:

import pandas as pd

from matplotlib import pyplot as plt

get\_ipython().run\_line\_magic('matplotlib', 'inline')

from sklearn.datasets import load\_breast\_cancer

# In[2]:

df = load\_breast\_cancer()

#df

# In[3]:

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

# In[4]:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.data,df.target,test\_size=0.2, stratify=df.target,random\_state=42)

# In[5]:

X\_test.shape

# In[6]:

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(C=100)

# In[7]:

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

# In[8]:

model.coef\_

# In[9]:

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(C=1)

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

model.coef\_

# In[10]:

### Coefficent's Value (decrease) if C value (decrease)

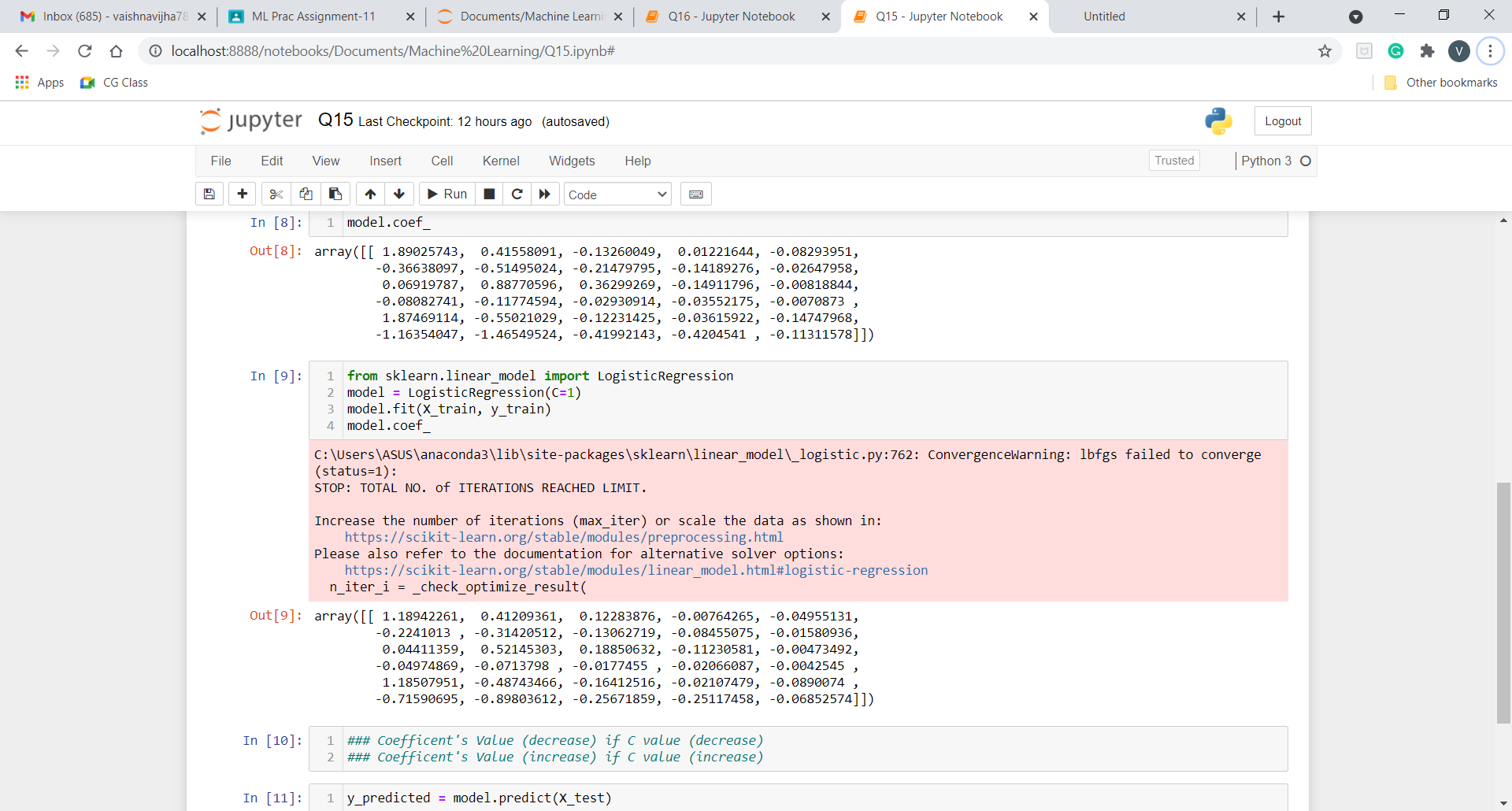
### Coefficent's Value (increase) if C value (increase)

# In[11]:

y\_predicted = model.predict(X\_test)

# In[ ]:

**Output:**



1. **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.**

**Program:**

#!/usr/bin/env python

# coding: utf-8

# In[1]:

from numpy import\*

import pandas as pd

from sklearn.datasets import load\_breast\_cancer

from sklearn.neural\_network import MLPClassifier

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

# In[2]:

df = load\_breast\_cancer()

# In[3]:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.data,df.target,test\_size=0.2,random\_state=4)

y\_test

# In[4]:

nn=MLPClassifier(activation='logistic',solver='sgd',hidden\_layer\_sizes=(10,15),random\_state=1)

# In[5]:

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

# In[6]:

pred=nn.predict(X\_test)

pred

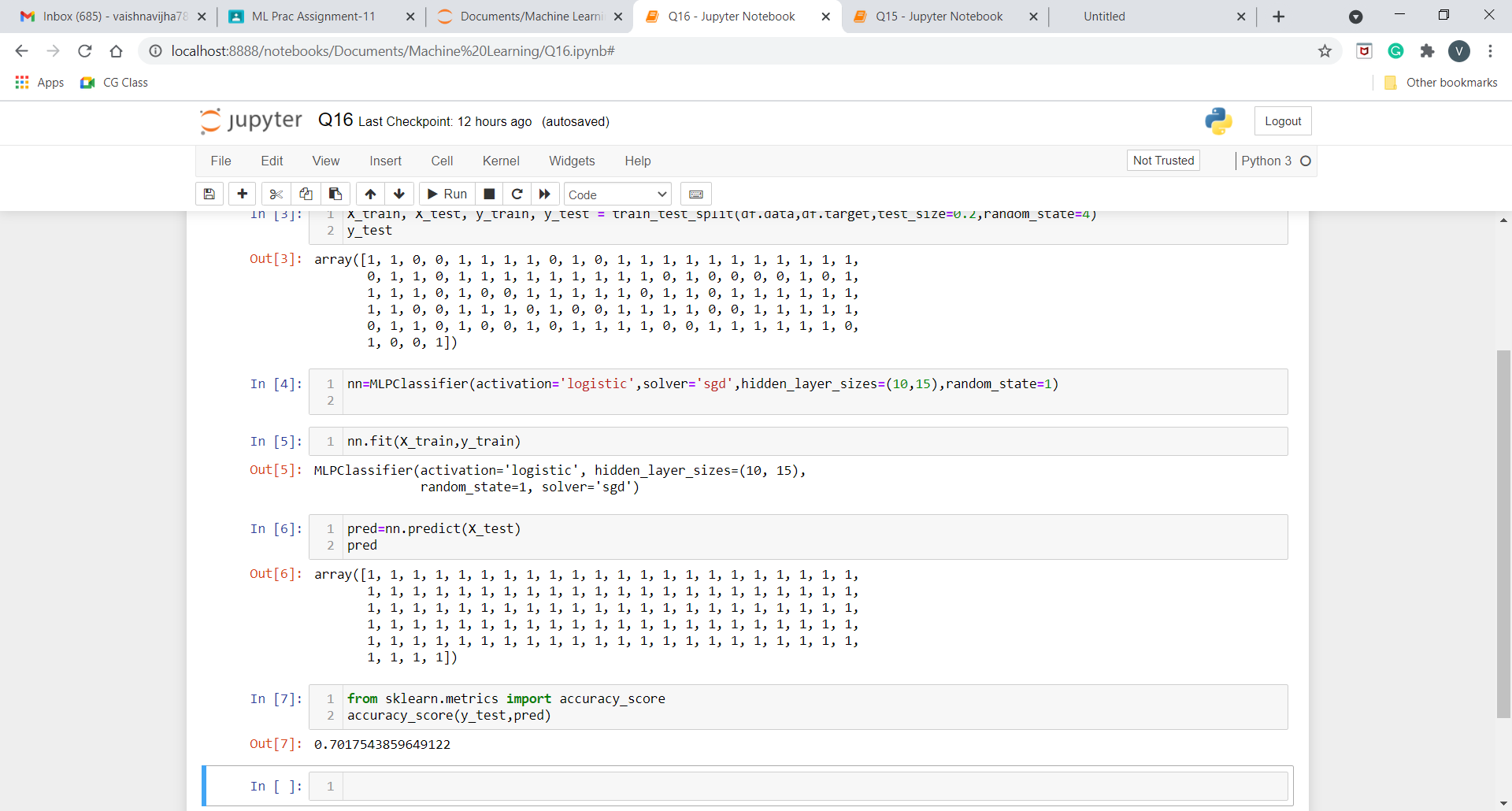
# In[7]:

from sklearn.metrics import accuracy\_score

accuracy\_score(y\_test,pred)

# In[ ]:

**Output:**



**EXTRA QUESTIONS**

1. **Write a python program which takes a number input from user and check if that is Armstrong number or not.**

**Program:**

n=int(input("Enter a number: "))

x=n

cube=0

while(x!=0):

a=x%10

cube=cube+a\*\*3

x=x//10

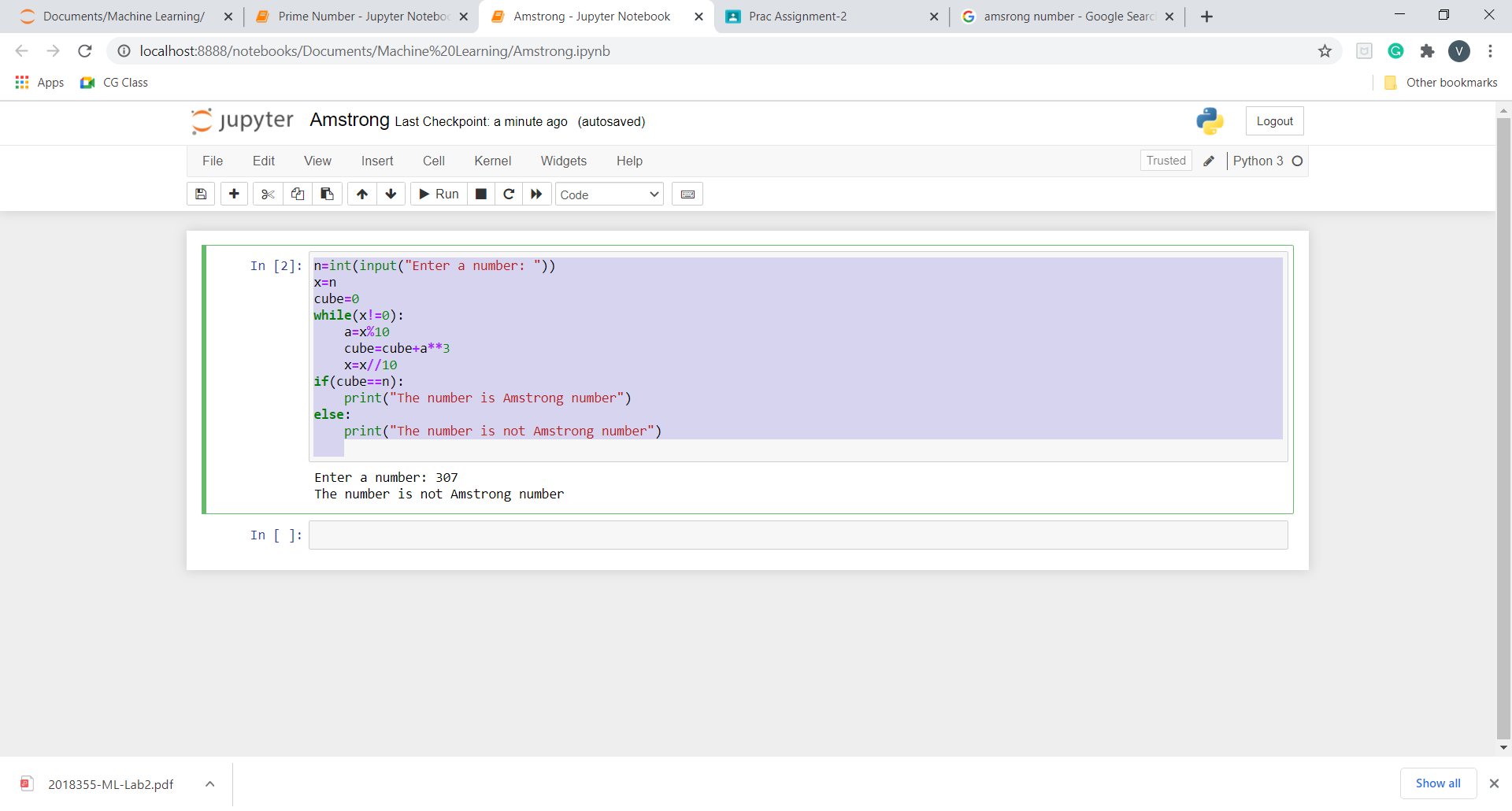
if(cube==n):

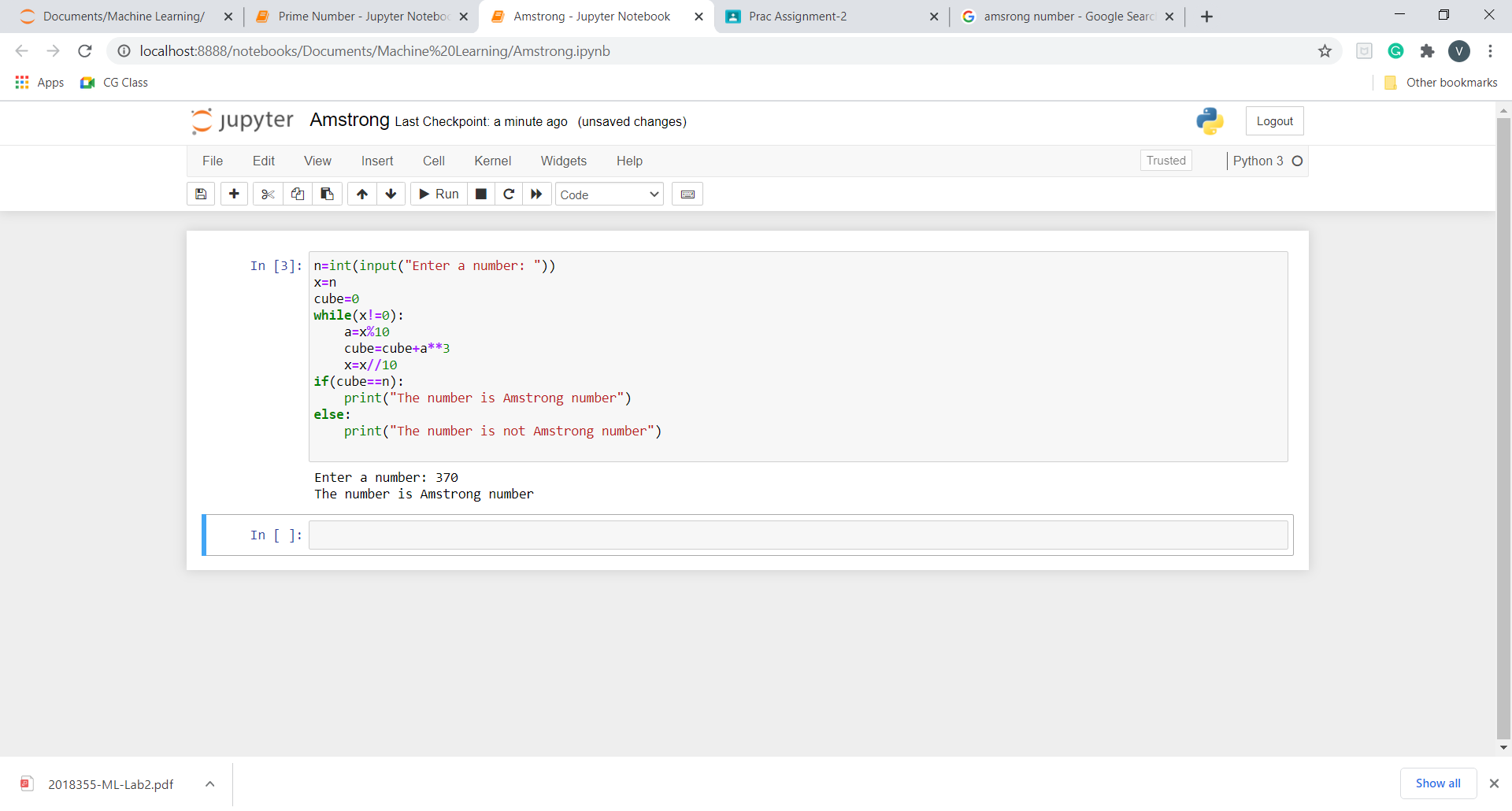
print("The number is Amstrong number")

else:

print("The number is not Amstrong number")

**Output:**





1. **Write a python program which takes a number input from user and print all prime numbers from 2 till that number.**

**Program:**

def checkPrime(n):

for i in range(2, n):

if (n%i==0):

return False

return True

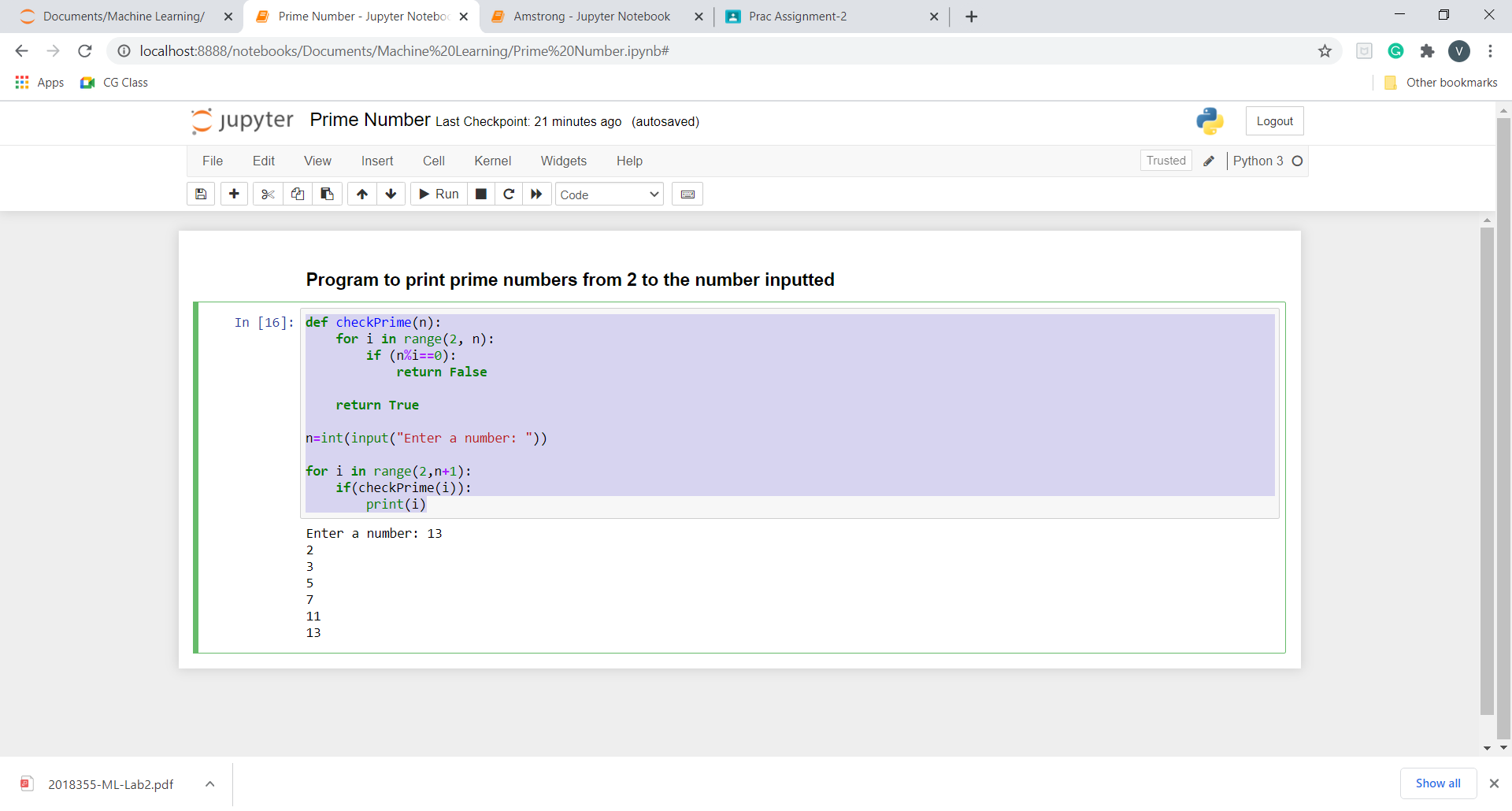
n=int(input("Enter a number: "))

for i in range(2,n+1):

if(checkPrime(i)):

print(i)

**Output:**



1. **Write a python program which takes a string input from user and check if that is palindrome or not.**

**Program:**

str=input("Enter a string : ")

isPalindrome=True

l=len(str)-1

i=0

while(i<l):

if str[i]!=str[l]:

isPalindrome=False

break

i=i+1

l=l-1

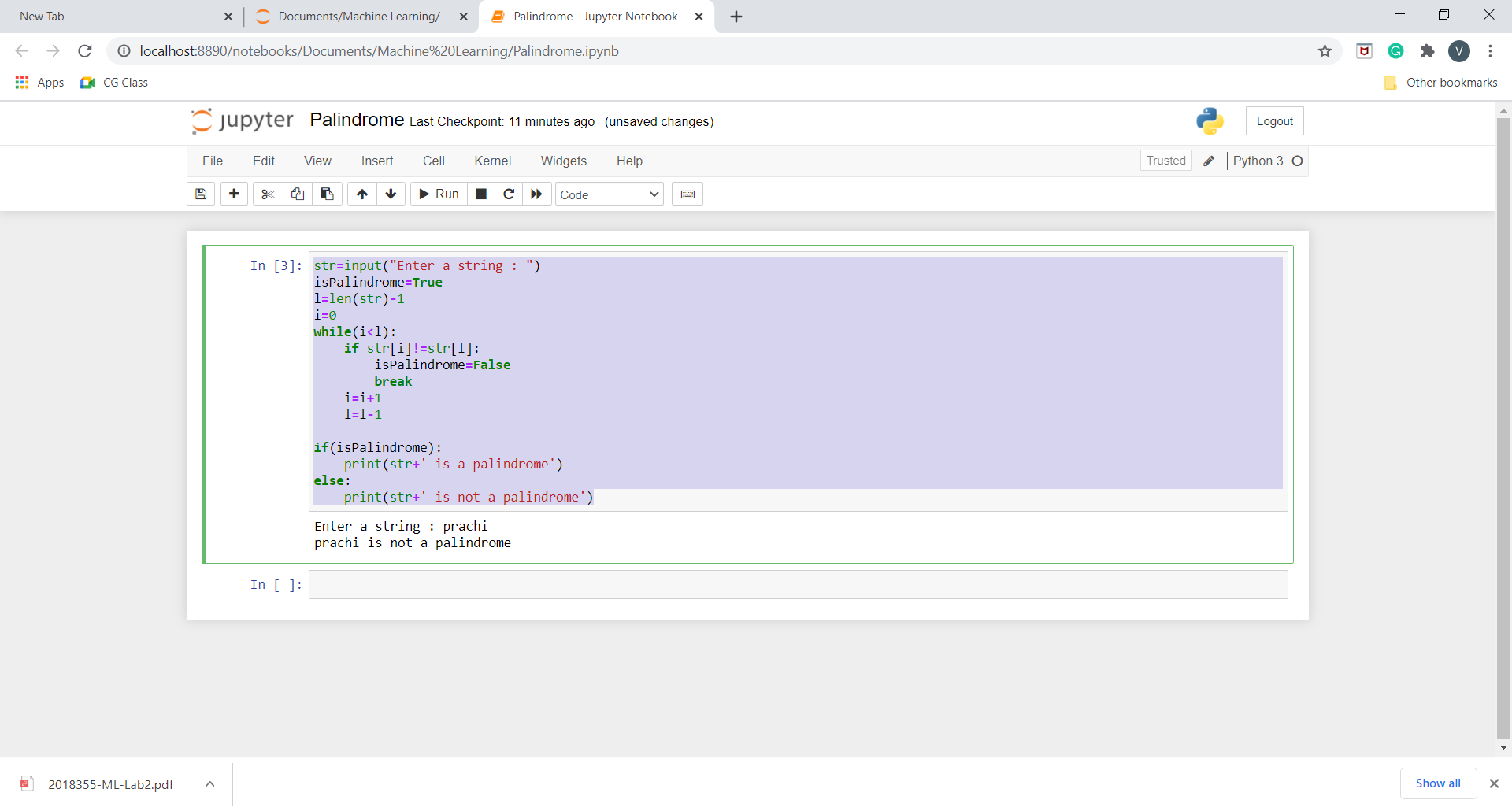
if(isPalindrome):

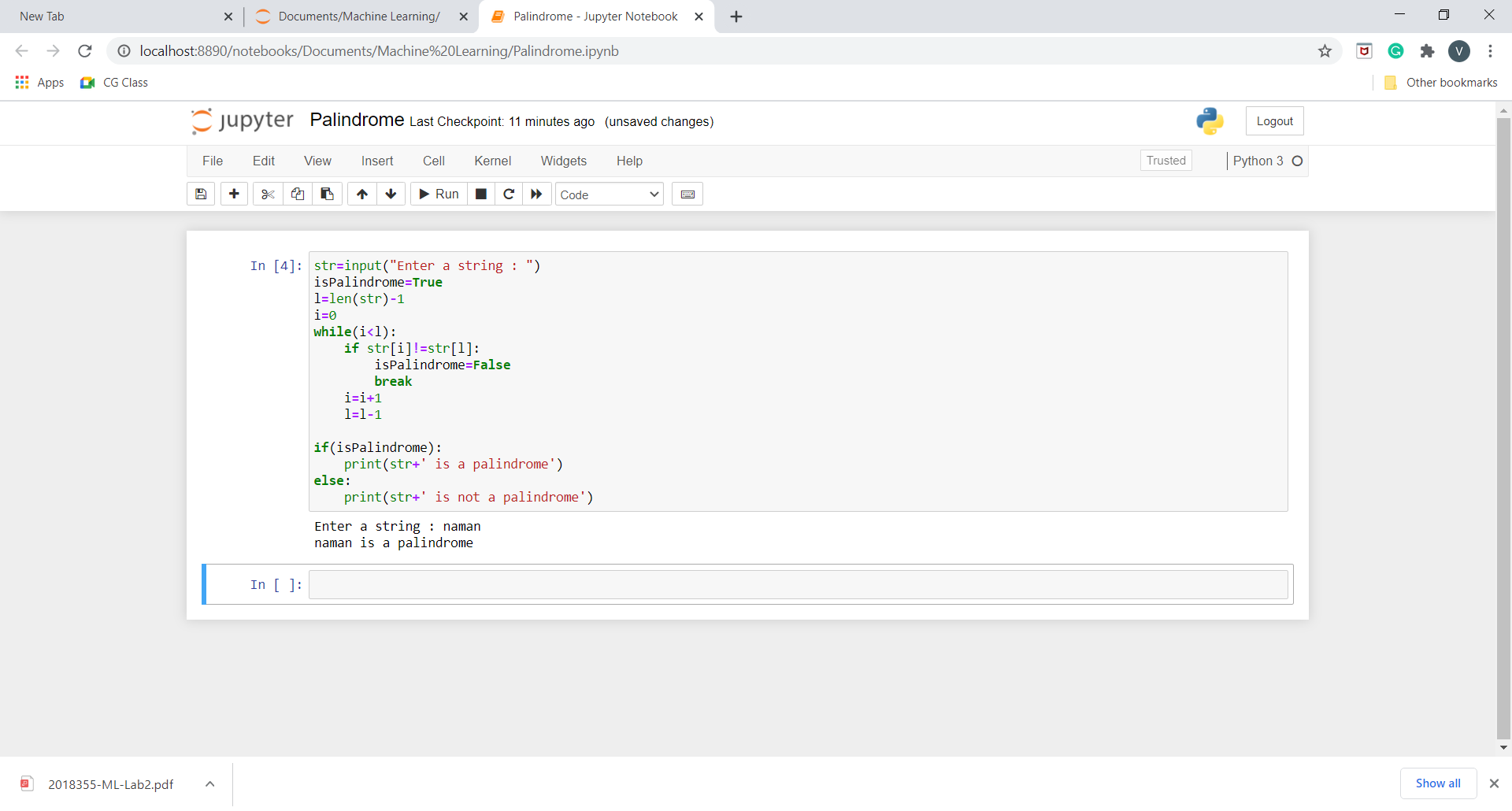
print(str+' is a palindrome')

else:

print(str+' is not a palindrome')

**Output:**





1. **Write a program that takes a list of values as input parameter an returns another list without any duplicates.**

**Program:**

n= int(input("Enter number of elements: "))

oldList=[int(input("Enter element number: ")) for i in range(n)]

print("You entered the following elements ",oldList)

newList=[]

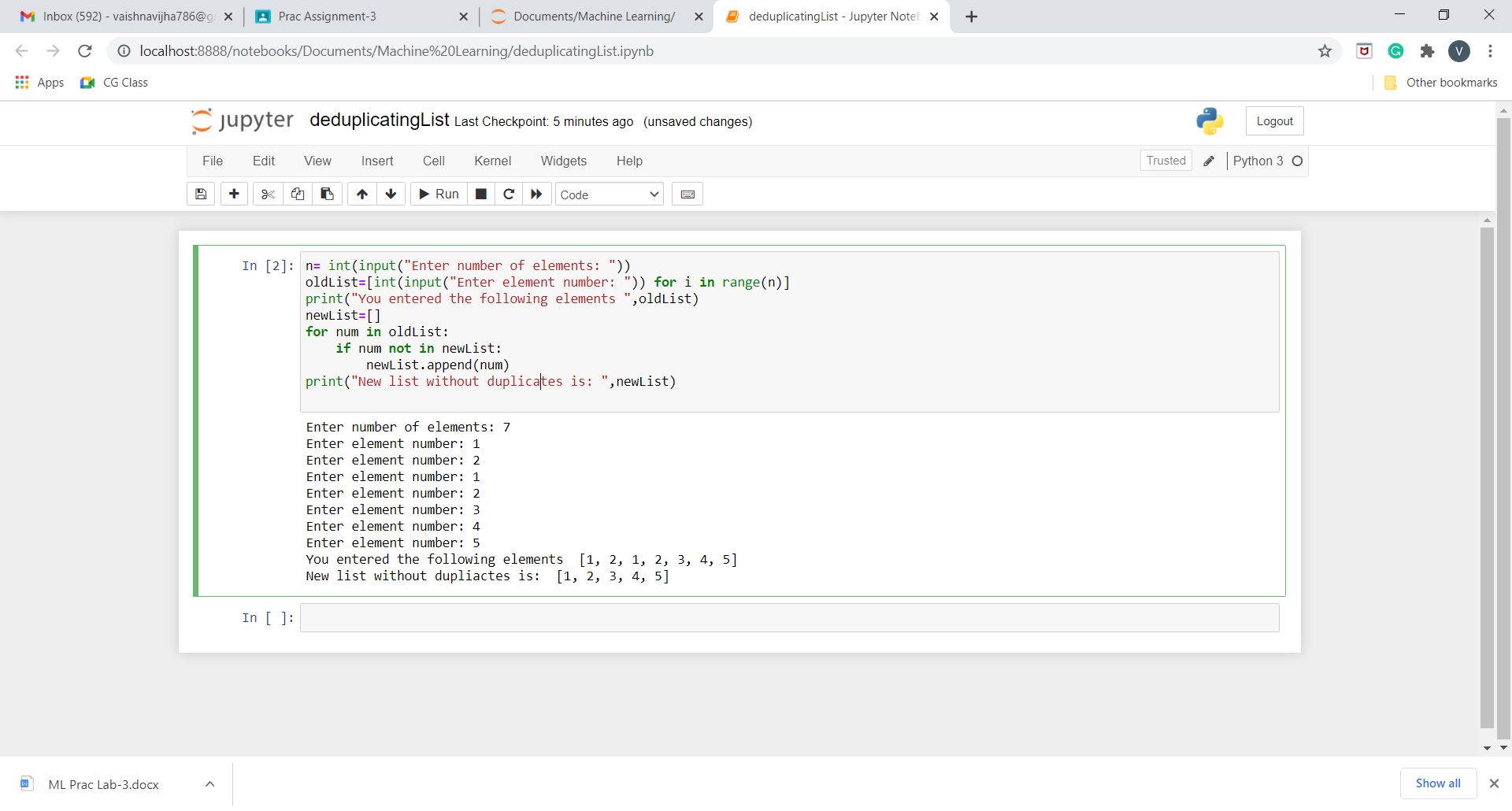
for num in oldList:

if num not in newList:

newList.append(num)

print("New list without duplicates is: ",newList)

**Output:**



1. **Write a program that takes a sequence as input from the user and computes the frequency of each letter. Use a variable of dictionary type to maintain count.**

**Program:**

str=input("Enter a string: ")

freq = { }

for i in str:

if i in freq:

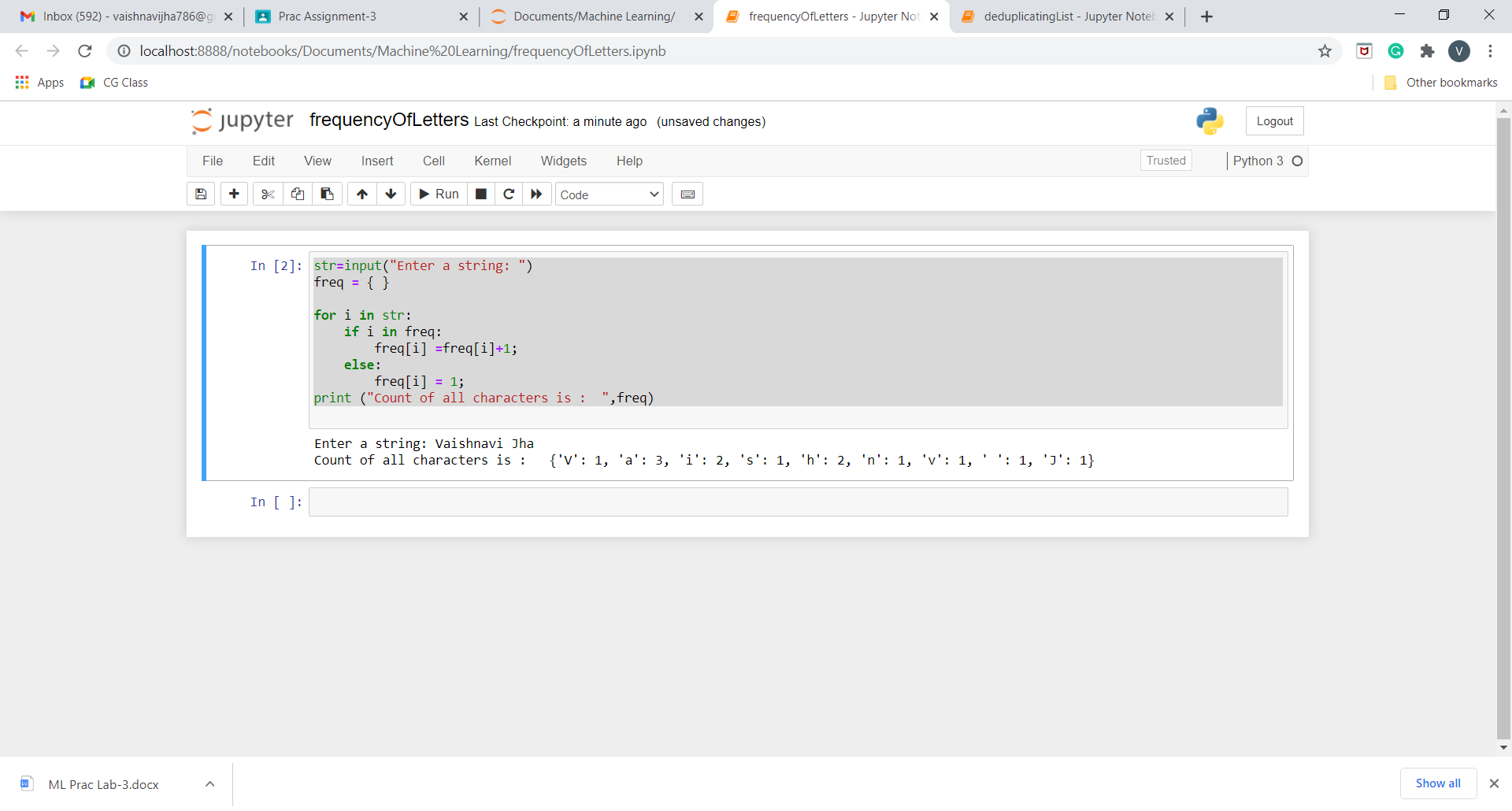
freq[i] =freq[i]+1;

else:

freq[i] = 1;

print ("Count of all characters is : ",freq)

**Output:**



1. **Write a program that takes a list of numbers as input from the user and produces the corresponding cumulative list where each element in the list at index I is the sum of elements at index j<=i.**

**Program:**

n= int(input("Enter number of elements: "))

lst=[int(input("Enter element number: ")) for i in range(n)]

print(lst)

sumList = []

sumList = [sum(lst[0:x]) for x in range(1, len(lst)+1)]

print(sumList)

**Output:**



1. **Implement Simple Linear Regression using analytical method and depict model on scatter data plot.**
2. **Take x=[1,2,4], y=[2,3,6]**

**Program:**

#!/usr/bin/env python

# coding: utf-8

# In[1]:

import numpy as np

import matplotlib.pyplot as plt

# In[2]:

x=np.array([1,2,4])

y=np.array([2,3,6])

# In[3]:

x

# In[4]:

y

# In[5]:

x\_mean=np.mean(x)

print(x\_mean)

y\_mean=np.mean(y)

print(y\_mean)

n=x.size

print(n)

# In[6]:

ss\_xy=np.sum(x\*y) - n\*x\_mean\*y\_mean

print(ss\_xy)

ss\_xx=np.sum(x\*x) - n\*x\_mean\*x\_mean

print(ss\_xx)

b1=ss\_xy/ss\_xx

print(b1)

b0=y\_mean-b1\*x\_mean

print(b0)

# In[7]:

plt.scatter(x,y)

y\_pred=b0+b1\*x

print(y\_pred)

plt.plot(x,y\_pred,color='red')

plt.xlabel("x")

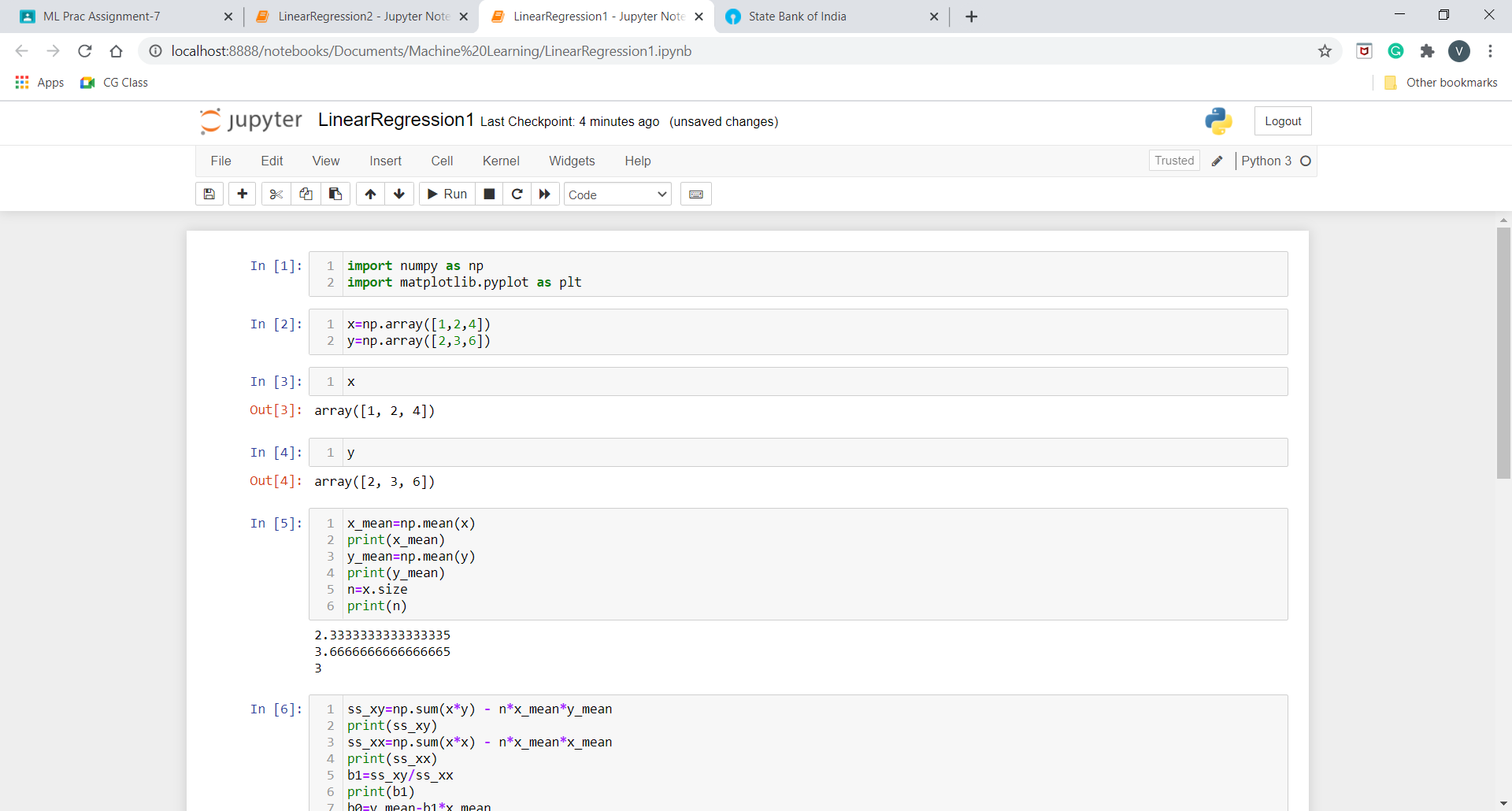
plt.ylabel("y")

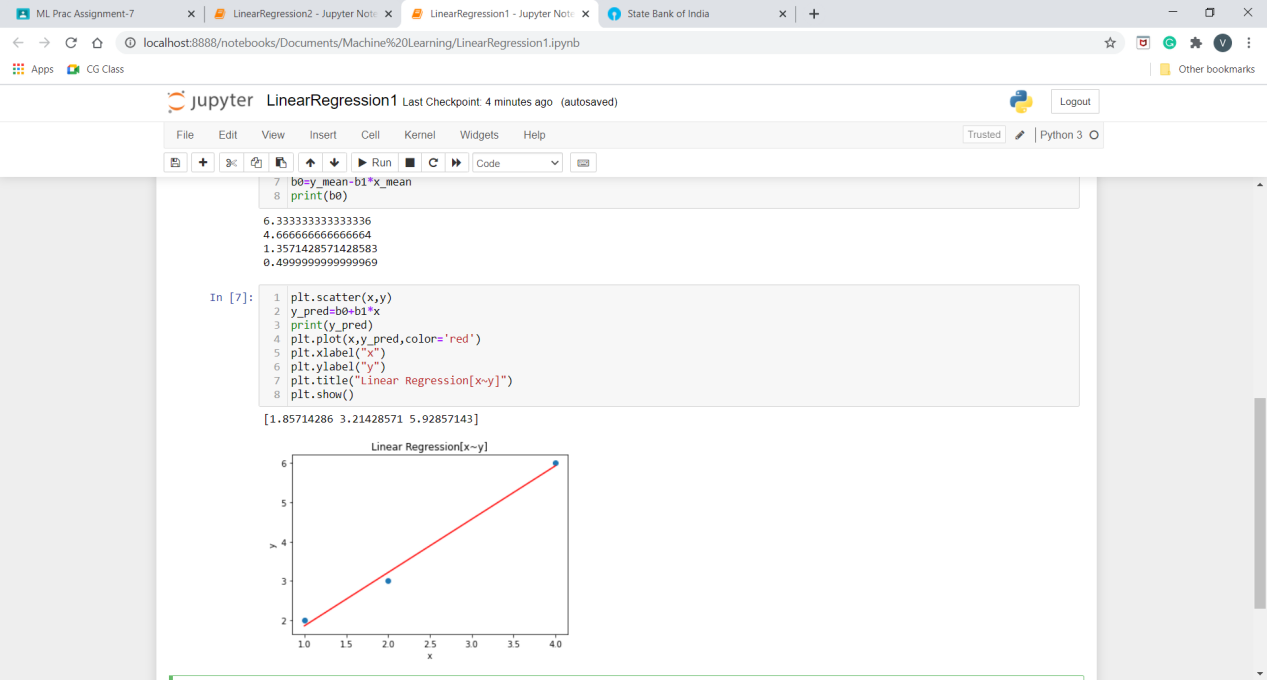
plt.title("Linear Regression[x~y]")

plt.show()

# In[ ]:

**Output:**





1. **Regress Sales~Radio from Advertisingdata.csv**

**Program:**

#!/usr/bin/env python

# coding: utf-8

# In[1]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# In[2]:

col\_list = ["TV", "radio","newspaper","sales"]

data = pd.read\_csv("Advertising.csv", usecols=col\_list)

# In[3]:

data["sales"] #y-axis

# In[7]:

data["radio"] #x-axis

# In[5]:

x\_mean=np.mean(data["radio"])

print(x\_mean)

y\_mean=np.mean(data["sales"])

print(y\_mean)

n=data["sales"].size

print(n)

# In[6]:

ss\_xy=np.sum(data["radio"]\*data["sales"]) - n\*x\_mean\*y\_mean

print(ss\_xy)

ss\_xx=np.sum(data["radio"]\*data["radio"]) - n\*x\_mean\*x\_mean

print(ss\_xx)

b1=ss\_xy/ss\_xx

print(b1)

b0=y\_mean-b1\*x\_mean

print(b0)

# In[9]:

plt.scatter(data["radio"],data["sales"])

y\_pred=b0+b1\*data["radio"]

print(y\_pred)

plt.plot(data["radio"],y\_pred,color='red')

plt.xlabel("radio")

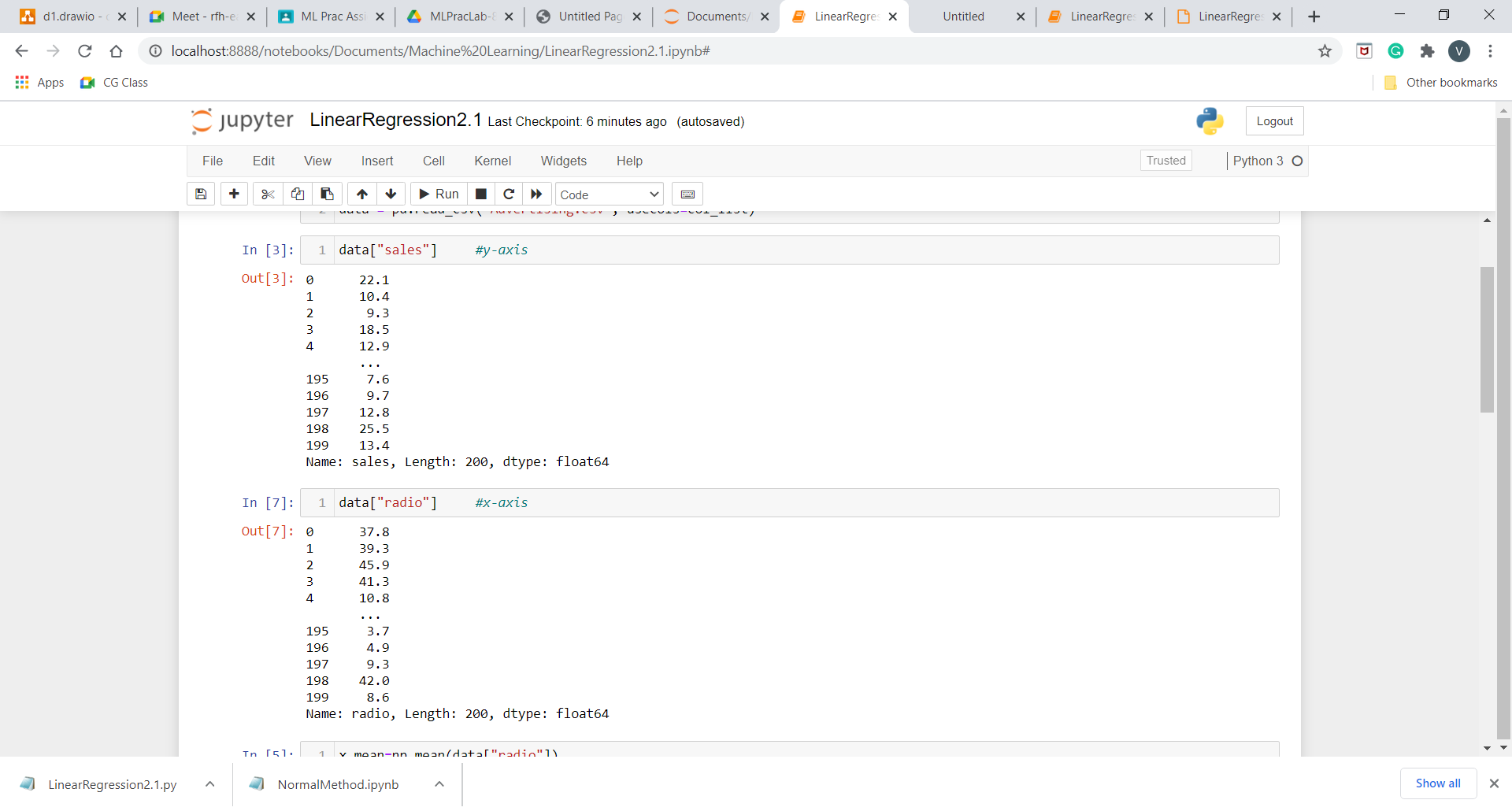
plt.ylabel("sales")

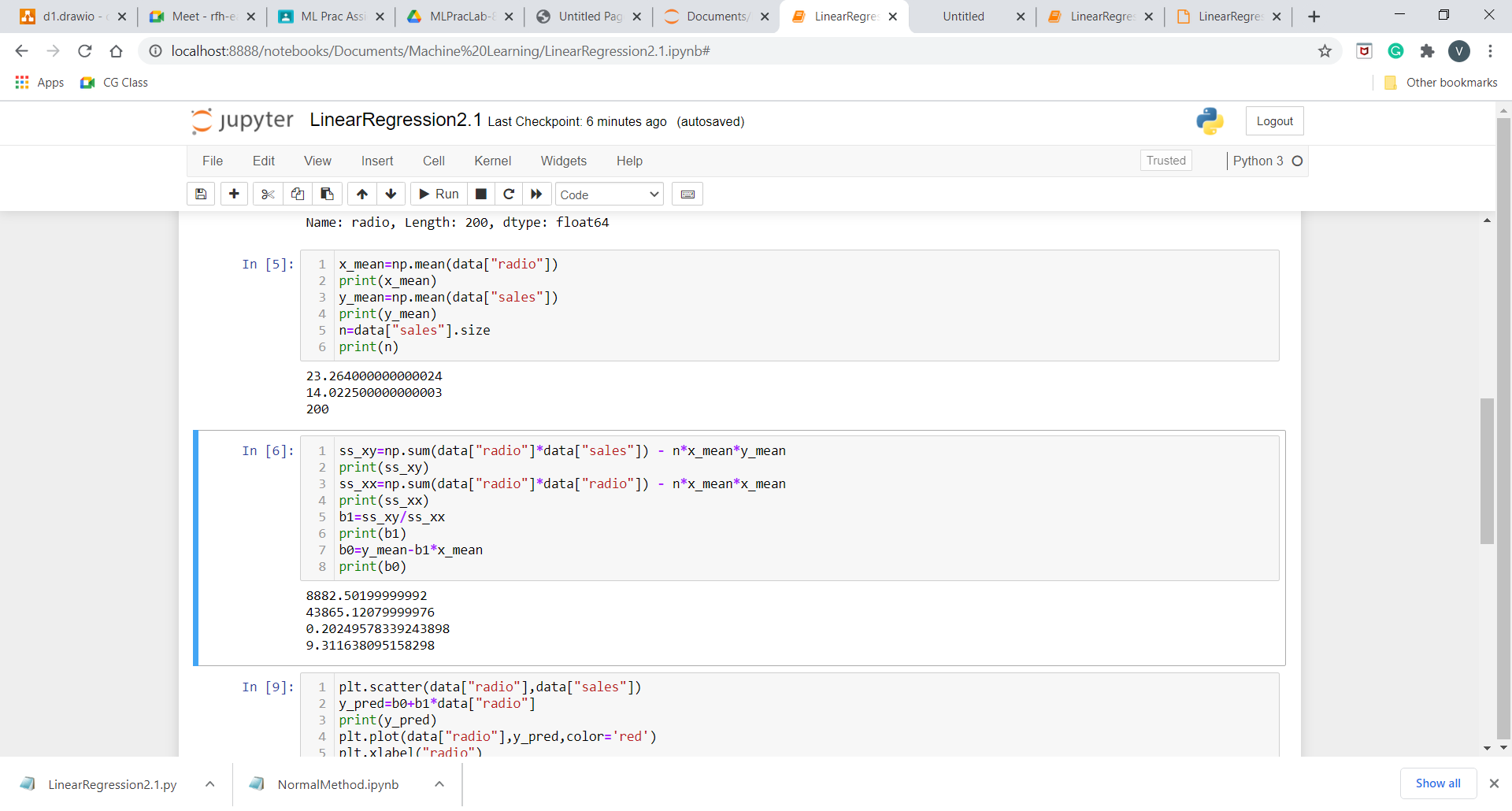
plt.title("Linear Regression[Sales~Radio]")

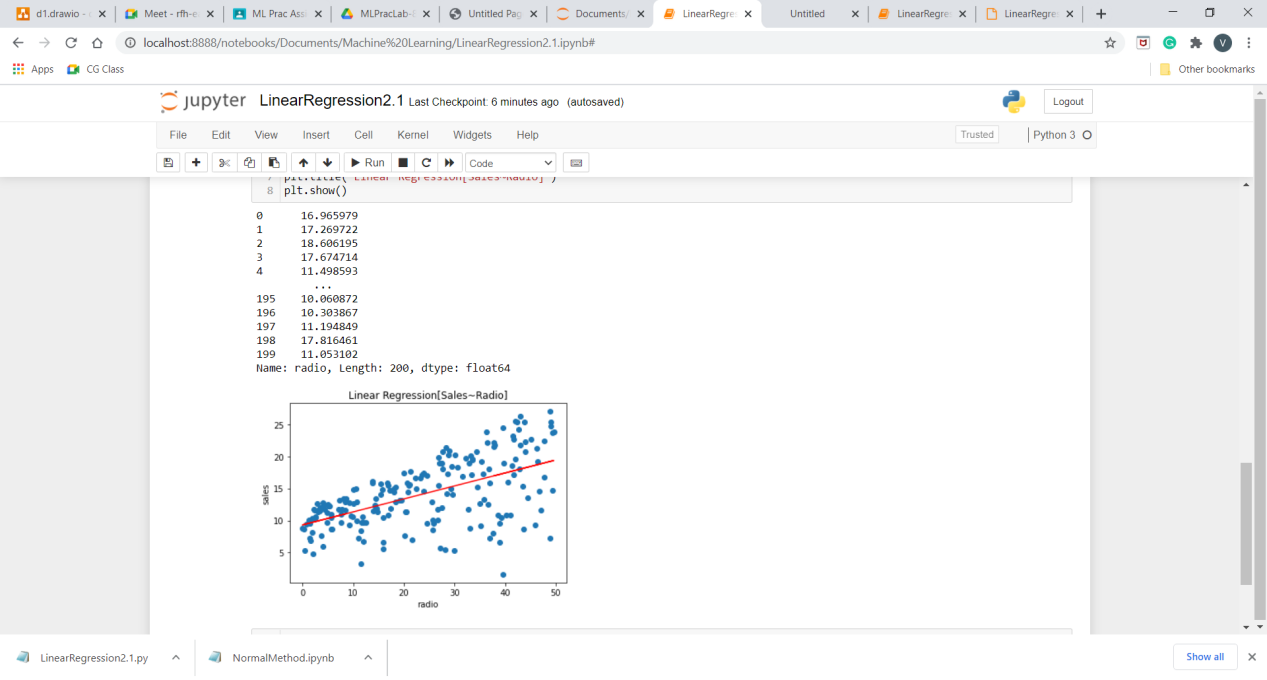
plt.show()

# In[ ]:

**Output:**







**8. Implement Linear Regression using Normal Equation method and depict model on scatter data plot. Also calculate SSE, R 2 .**

**A) Regress Sales~Radio from Advertisingdata.csv**

**Program:**

#!/usr/bin/env python

# coding: utf-8

# In[3]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# In[4]:

col\_list = ["TV", "radio","newspaper","sales"]

data = pd.read\_csv("Advertising.csv", usecols=col\_list)

# In[5]:

data["sales"] #y-axis

# In[6]:

data["radio"] #x-axis

# In[7]:

x\_mean=np.mean(data["radio"])

print(x\_mean)

y\_mean=np.mean(data["sales"])

print(y\_mean)

n=data["sales"].size

print(n)

# In[8]:

ss\_xy=np.sum(data["radio"]\*data["sales"]) - n\*x\_mean\*y\_mean

print(ss\_xy)

ss\_xx=np.sum(data["radio"]\*data["radio"]) - n\*x\_mean\*x\_mean

print(ss\_xx)

b1=ss\_xy/ss\_xx

print(b1)

b0=y\_mean-b1\*x\_mean

print(b0)

# In[9]:

plt.scatter(data["radio"],data["sales"])

y\_pred=b0+b1\*data["radio"]

print(y\_pred)

plt.plot(data["radio"],y\_pred,color='red')

plt.xlabel("radio")

plt.ylabel("sales")

plt.title("Linear Regression[Sales~Radio]")

plt.show()

# In[12]:

e=data["sales"]-y\_pred

e

# In[21]:

RSS=np.sum(e\*\*2)

# In[22]:

RSS

# In[23]:

MSE=RSS/n

# In[24]:

MSE

# In[25]:

SST=np.sum((data["sales"]-y\_mean)\*\*2)

SST

# In[26]:

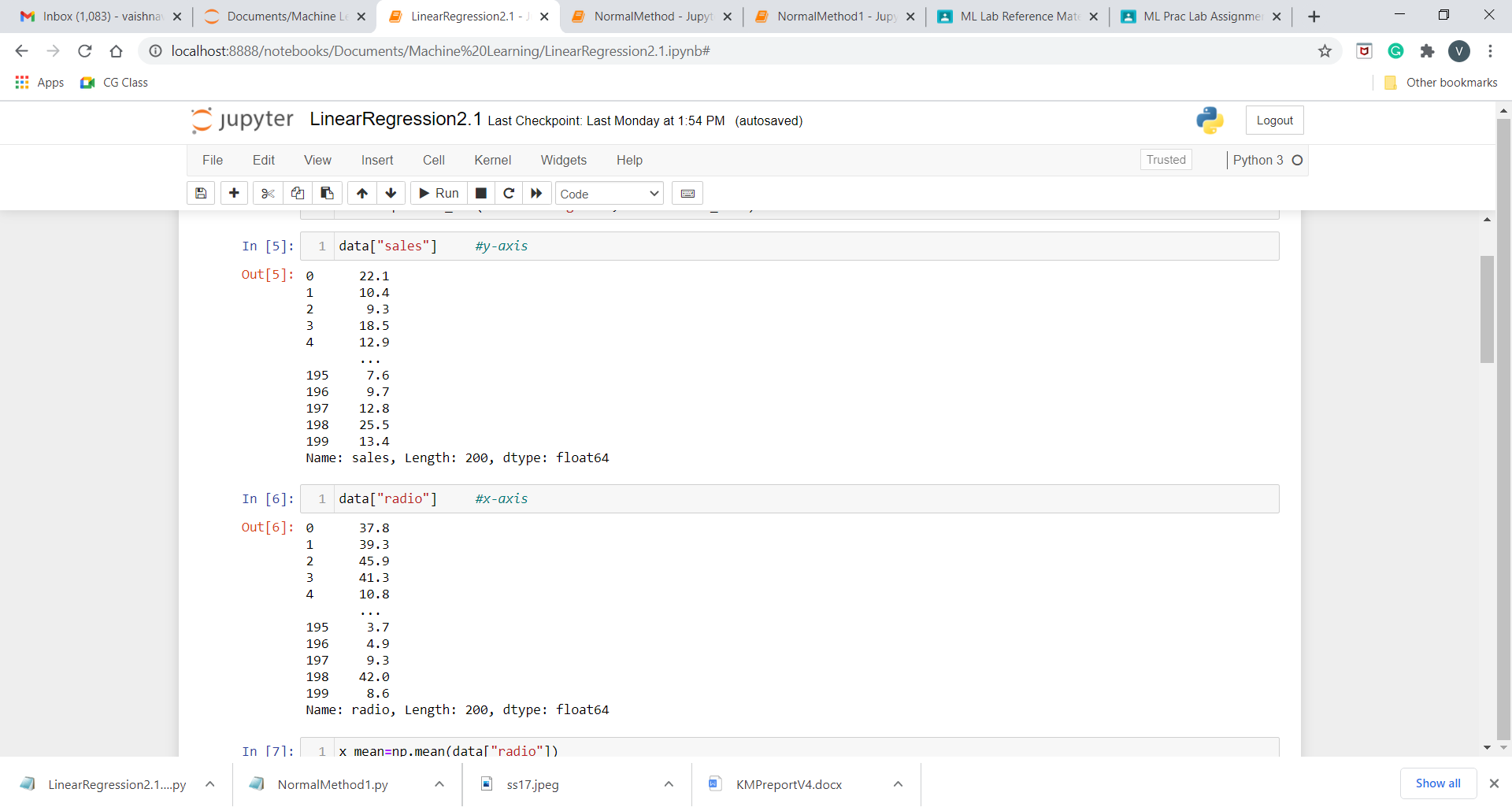
R2=1-(RSS/SST)

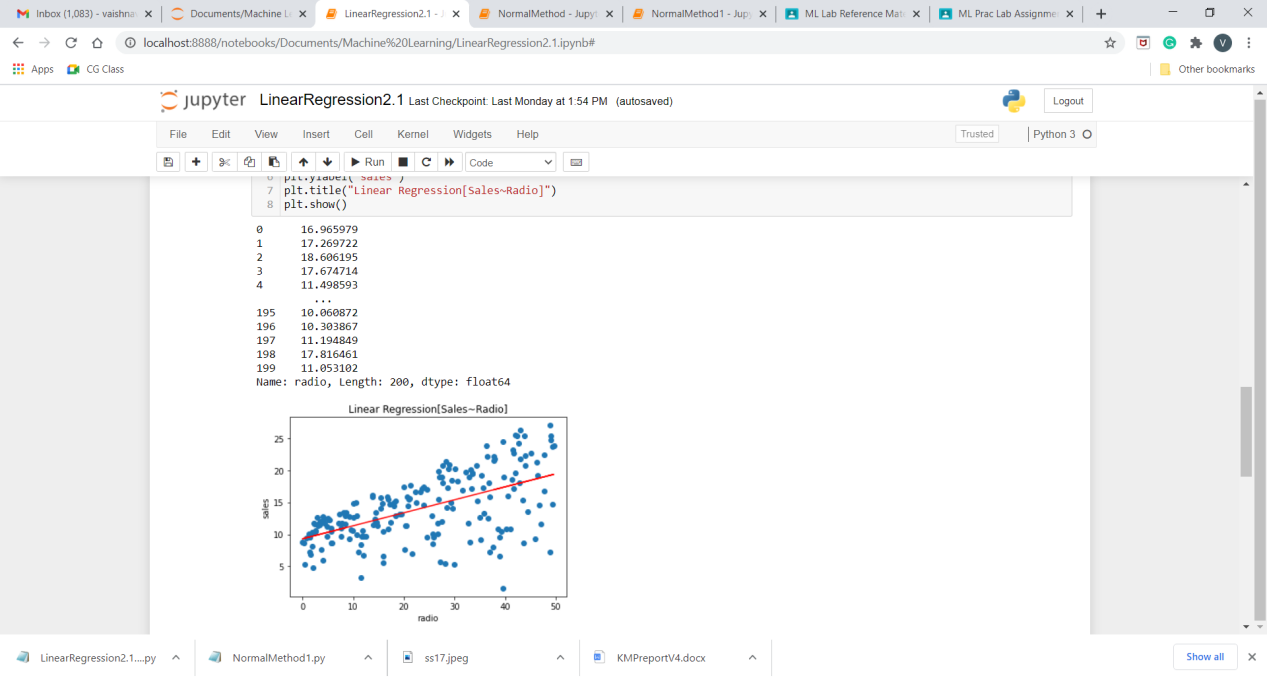
# In[27]:

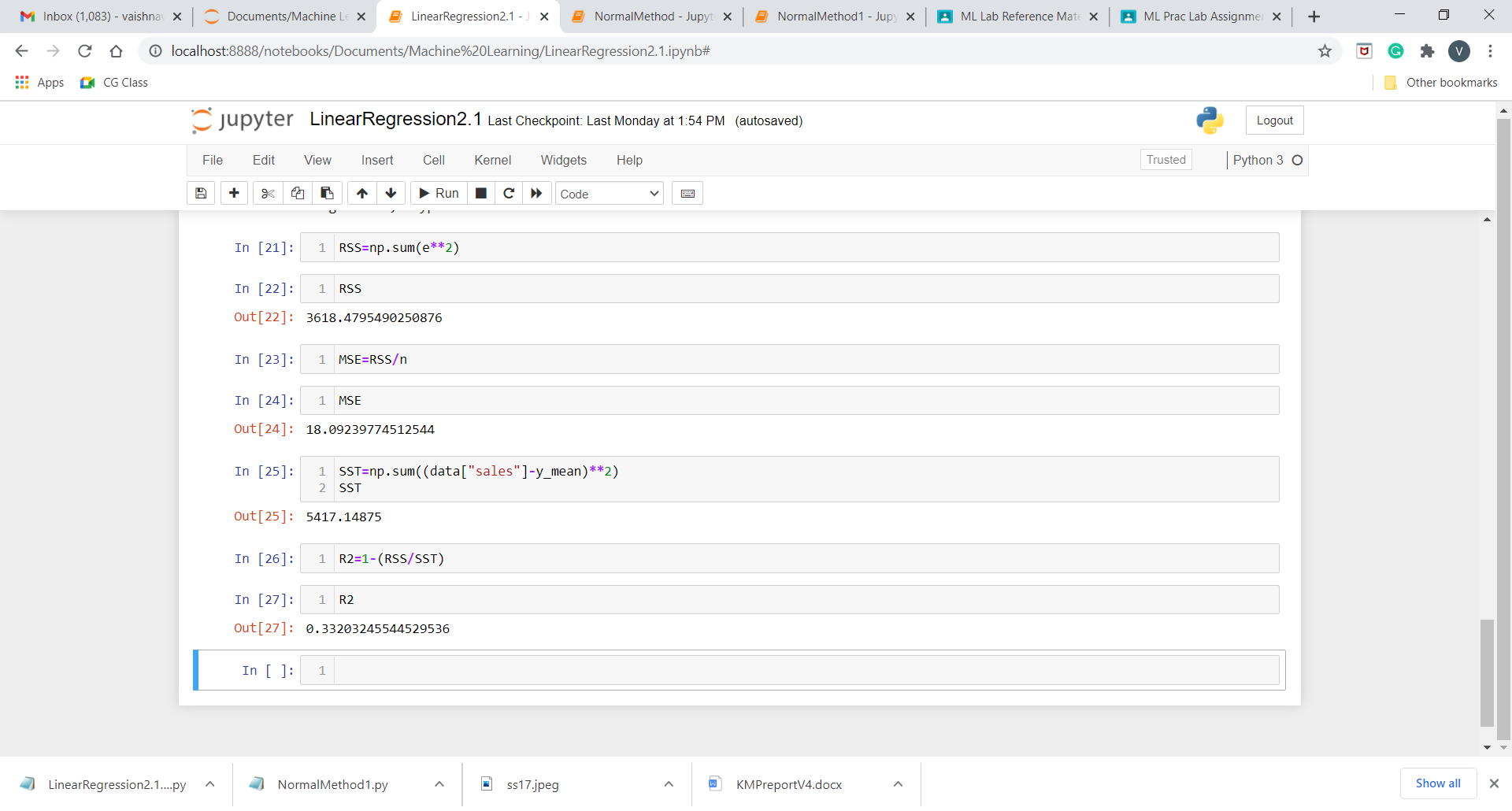
R2

# In[ ]:

**Output:**







1. **Regress Sales~Radio, TV, Newspaper from Advertisingdata.csv**

#!/usr/bin/env python

# coding: utf-8

# In[2]:

import pandas as pd

import numpy as np

import statistics

import math

import statsmodels.api as sm

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import axes3d

ds=pd.read\_csv("Advertising.csv")

# In[3]:

x1=ds.TV.values

#print(x1.shape)

x1=x1.reshape(200,1)

#print(x1.shape)

x2=ds.radio.values

x2=x2.reshape(200,1)

x3=ds.newspaper.values

x3=x3.reshape(200,1)

x0=np.ones([200,1])

X=np.column\_stack((x0,x1,x2,x3))

print(X)

y=ds.sales.values

Y=y.reshape(200,1)

# In[4]:

Xt=np.transpose(X)

print(Xt.shape)

XtX=np.matmul(Xt,X)

print(XtX.shape)

XtX\_inv=np.linalg.inv(XtX)

print(XtX\_inv.shape)

prod=np.matmul(XtX\_inv,Xt)

theta=np.matmul(prod,Y)

theta

# In[8]:

y\_pred=np.matmul(X,theta)

# In[64]:

e=Y-y\_pred

e.shape

# In[71]:

e2=e\*\*2

e2

# In[95]:

RSS=np.sum(e2)

RSS

# In[97]:

MSE=np.mean(e2)

MSE

# In[104]:

y\_mean=np.mean(Y)

y\_mean

# In[105]:

Y-y\_mean

# In[102]:

SST=np.sum((Y-y\_mean)\*\*2)

SST

# In[98]:

fig = plt.figure()

ax = plt.axes(projection='3d')

ax.scatter3D(x1, x2, y\_pred);

ax.set\_xlabel('x1')

ax.set\_ylabel('x2')

ax.set\_zlabel('y\_pred');

# In[99]:

a = np.column\_stack((x0,x1))

b = np.column\_stack((x0,x2))

c = np.column\_stack((x0,y))

# In[92]:

fig = plt.figure(num=1, clear=True)

ax = fig.add\_subplot(1, 1, 1, projection='3d')

ax.plot\_surface(a, b, c)

ax.scatter3D(x1, x2, Y,color='orange');

ax.set(xlabel='x1', ylabel='x2', zlabel='y')

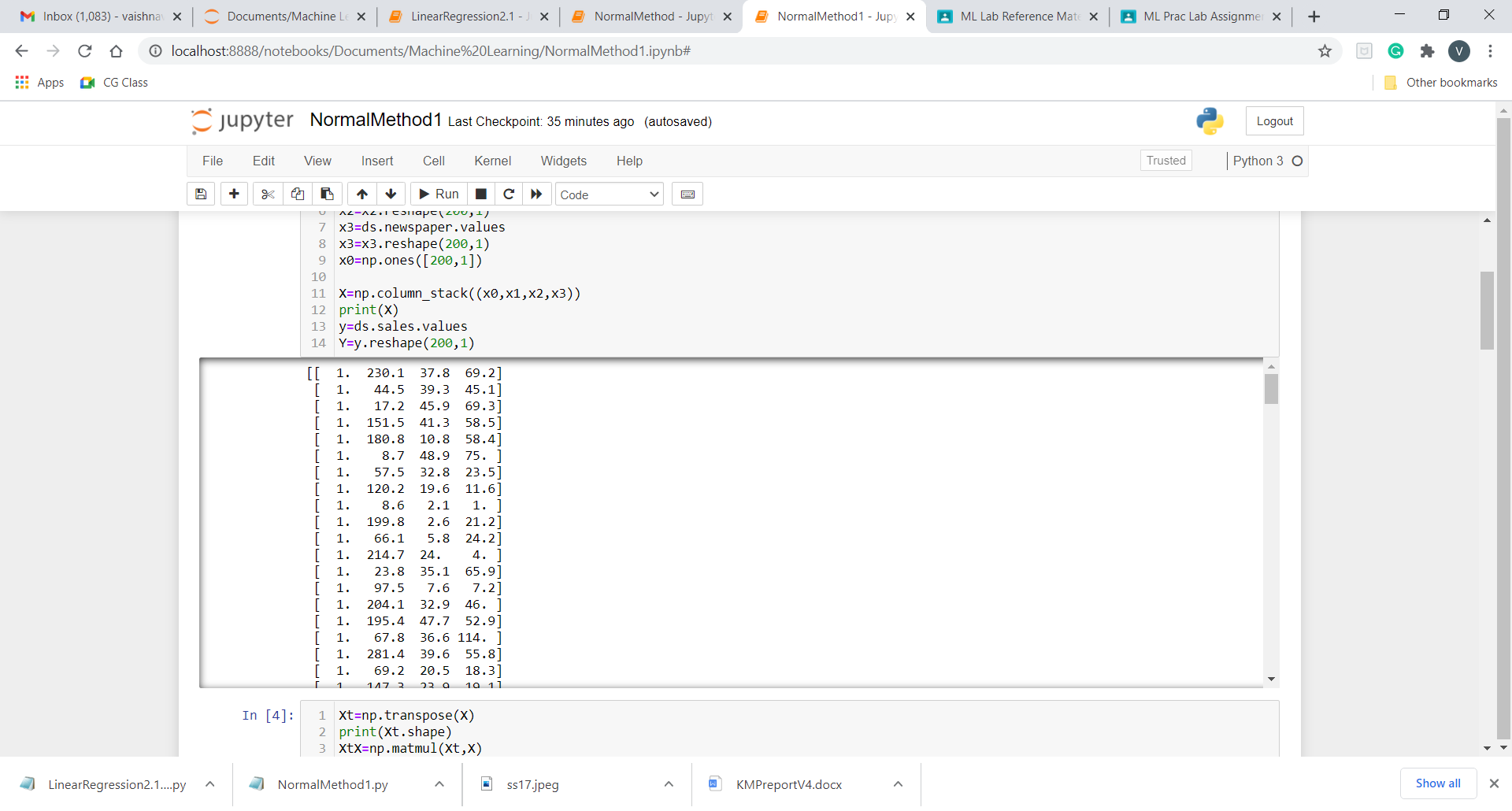
# In[110]:

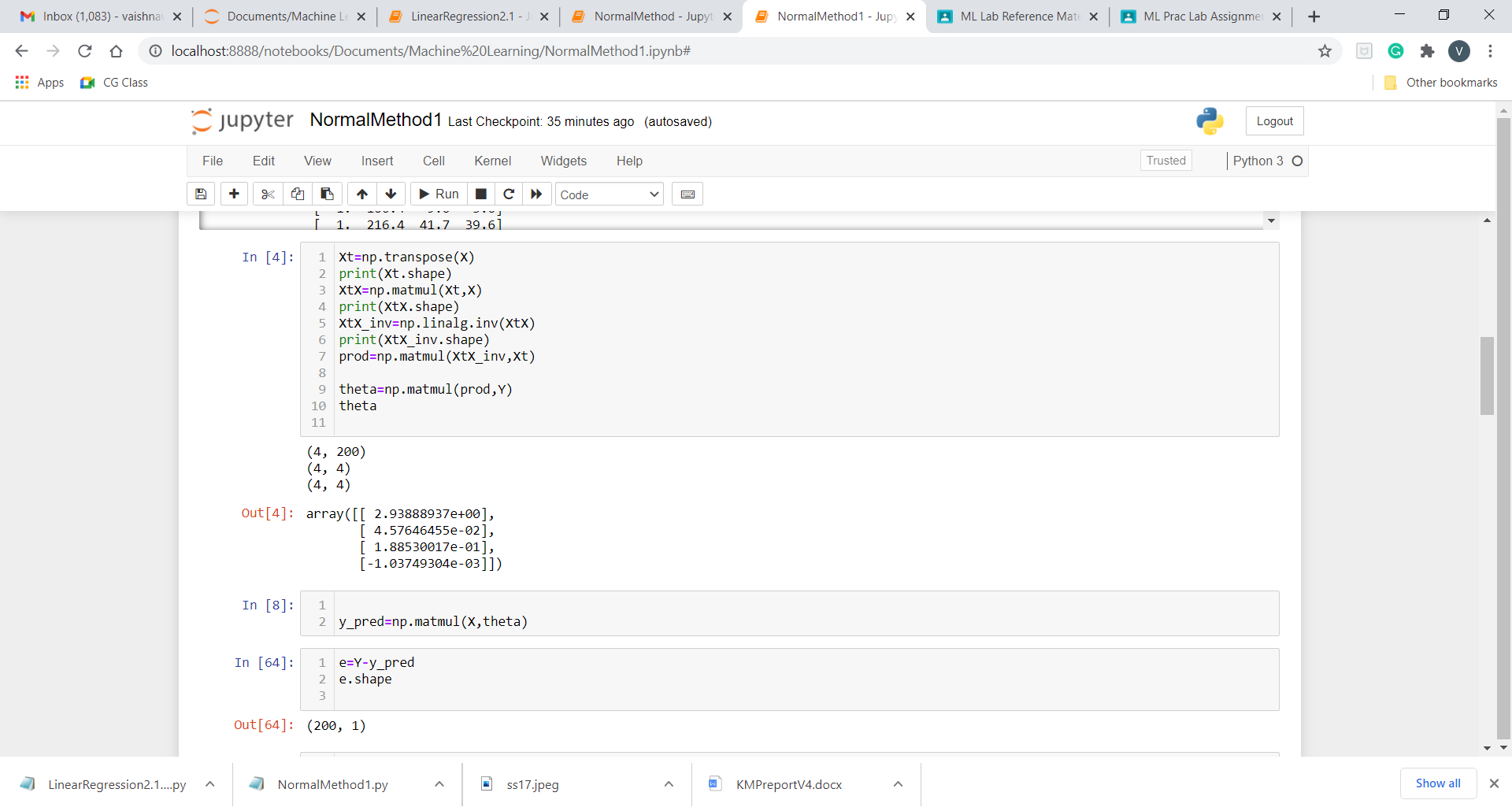
R2=1-(RSS/SST)

R2

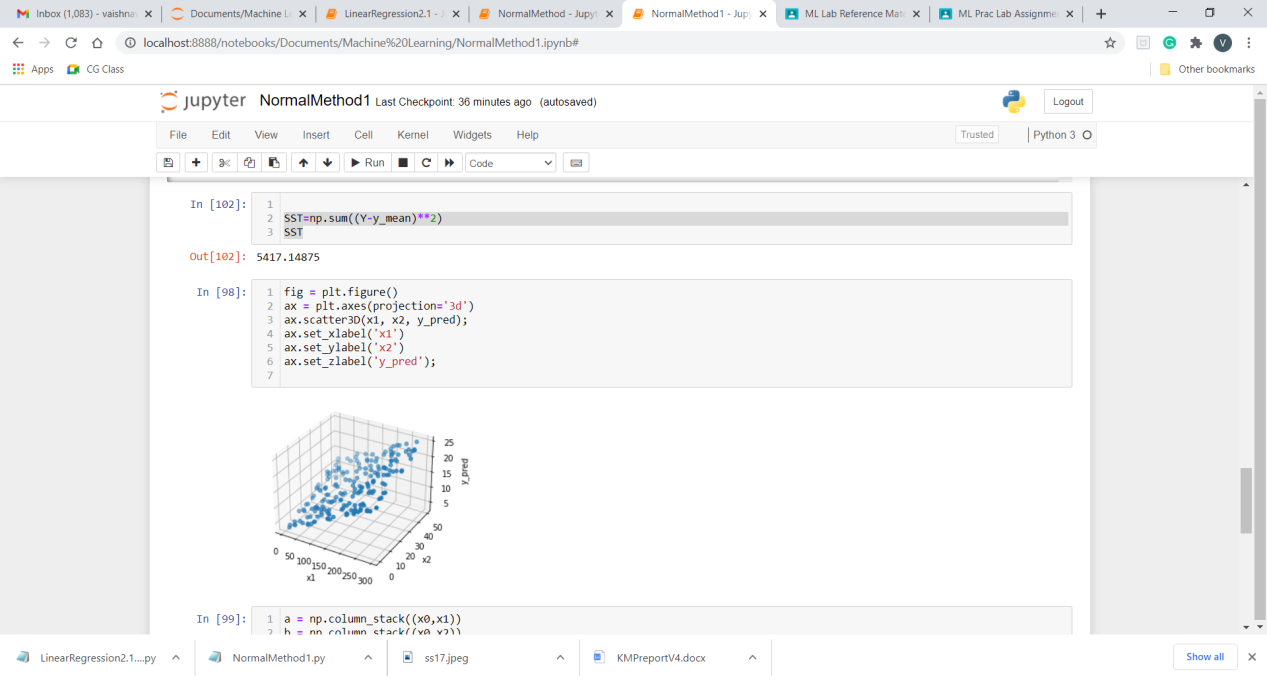
# In[ ]:

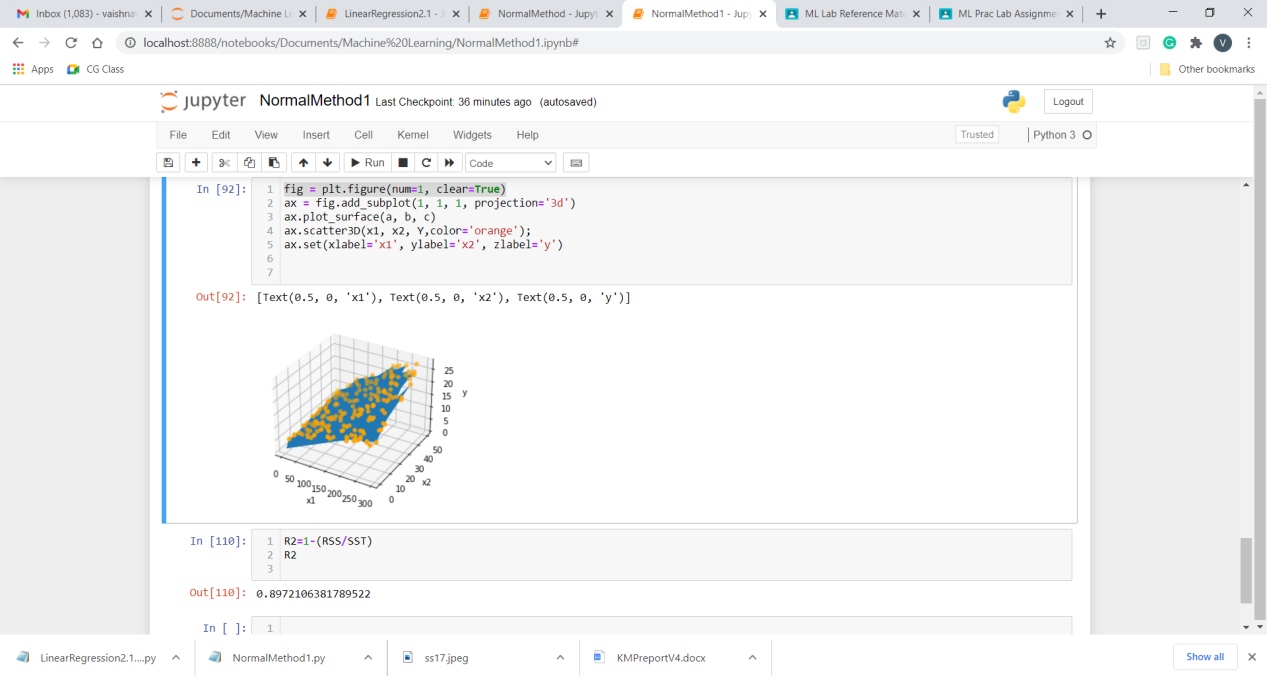
**Output:**











**9. Implement Simple Linear Regressions Sales~Radio, Sales~Newspaper, Sales~TV using Gradient Descent and depict model on scatter data plot. Also calculate SSE, R2 (Use Advertising Dataset).**

**Bonus Question: Implement Multiple Linear Regression using Gradient Descent Method on Advertising Dataset.**

1. **Sales~Radio**

**Program:**

#!/usr/bin/env python

# coding: utf-8

# In[95]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# In[96]:

col\_list = ["TV", "radio","newspaper","sales"]

data = pd.read\_csv("Advertising.csv", usecols=col\_list)

# In[97]:

data["sales"] #y-axis

# In[98]:

data["radio"] #x-axis

# In[99]:

x1=data.radio.values

x1=x1.reshape(200,1)

x0=np.ones([200,1])

X=np.column\_stack((x0,x1))

print(X.shape)

y=data.sales.values

y=y.reshape(200,1)

theta=np.random.randn(2,1)

theta

# In[100]:

def cal\_cost(theta,X,y):

m=len(y)

y\_pred=X.dot(theta)

cost=np.sum((np.square(y\_pred-y)))/(2\*m)

return cost

# In[101]:

def gradient\_descent(X,y,theta,alpha=0.01,iterations=100):

m=len(y)

cost\_history=np.zeros(iterations)

theta\_history=np.zeros((iterations,2))

for it in range(iterations):

prediction=np.dot(X,theta)

theta=theta-(1/m)\*alpha\*(X.T.dot((prediction-y)))

theta\_history[it,:]=theta.T

cost\_history[it]=cal\_cost(theta,X,y)

return theta,cost\_history,theta\_history

# In[102]:

theta,cost\_history,theta\_history=gradient\_descent(X,y,theta,0.0006,100000)

print(cost\_history.shape)

iter1=np.arange(0,100000)

plt.scatter(iter1,cost\_history,color='red')

plt.xlabel("iterations")

plt.ylabel("cost")

plt.title("GradientDescent[Sales~Radio]")

plt.show()

print(theta)

# In[103]:

plt.plot(iter1,cost\_history,color='red')

plt.xlabel("iterations")

plt.ylabel("cost")

plt.title("GradientDescent[Sales~Radio]")

plt.show()

# In[104]:

y\_pred=X.dot(theta)

RSS=np.sum((np.square(y\_pred-y)))

# In[105]:

RSS

# In[106]:

y\_mean=np.mean(y)

SST=np.sum((y-y\_mean)\*\*2)

SST

# In[107]:

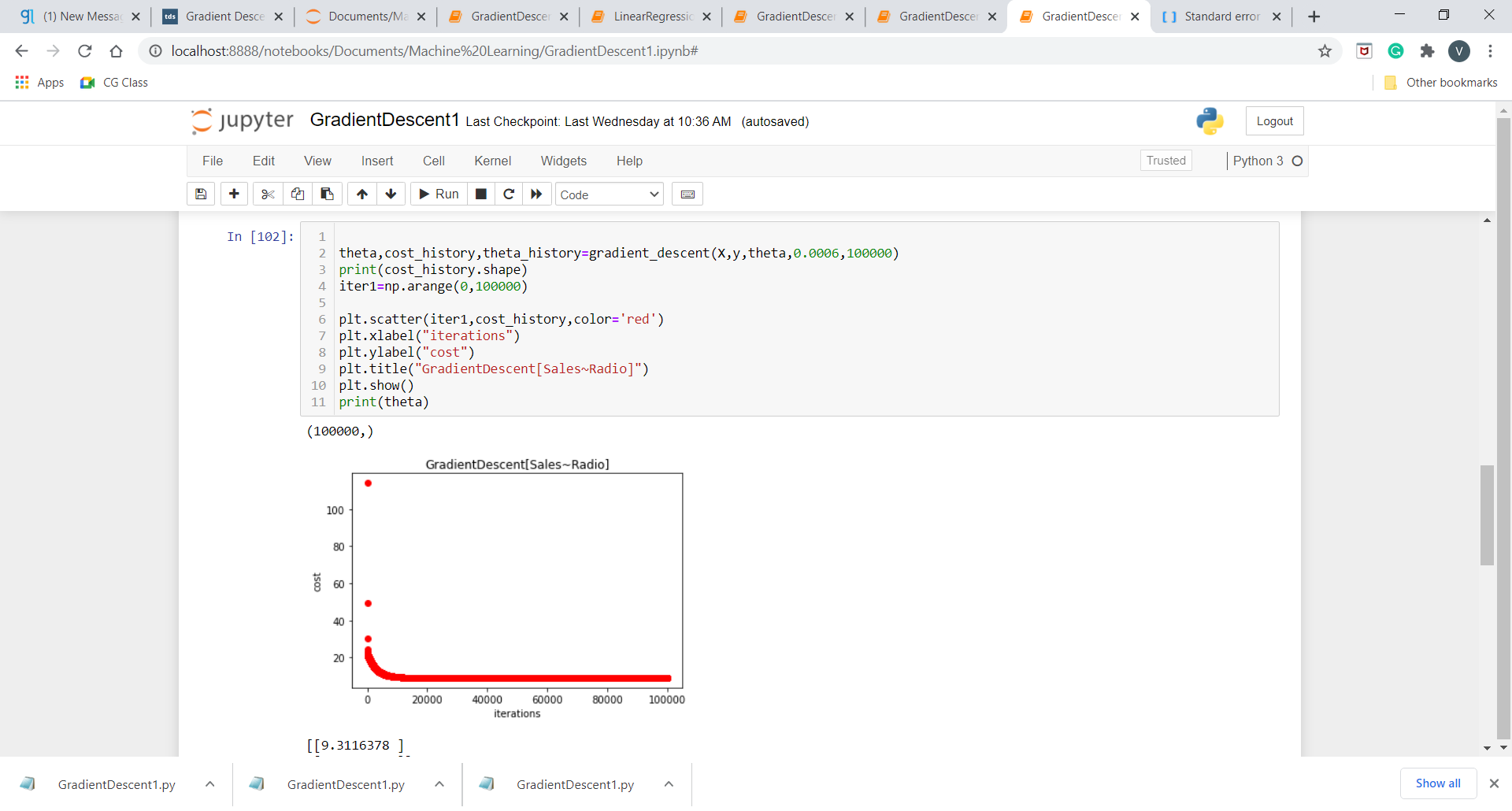
R2=1-(RSS/SST)

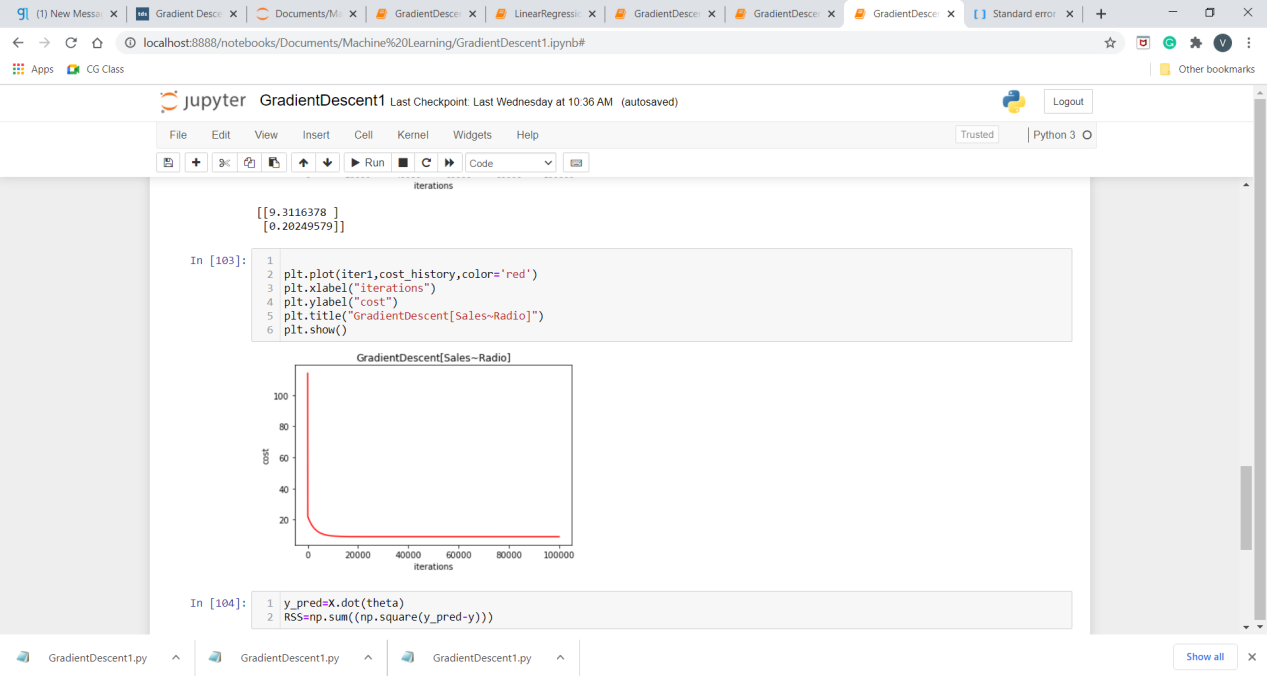
# In[108]:

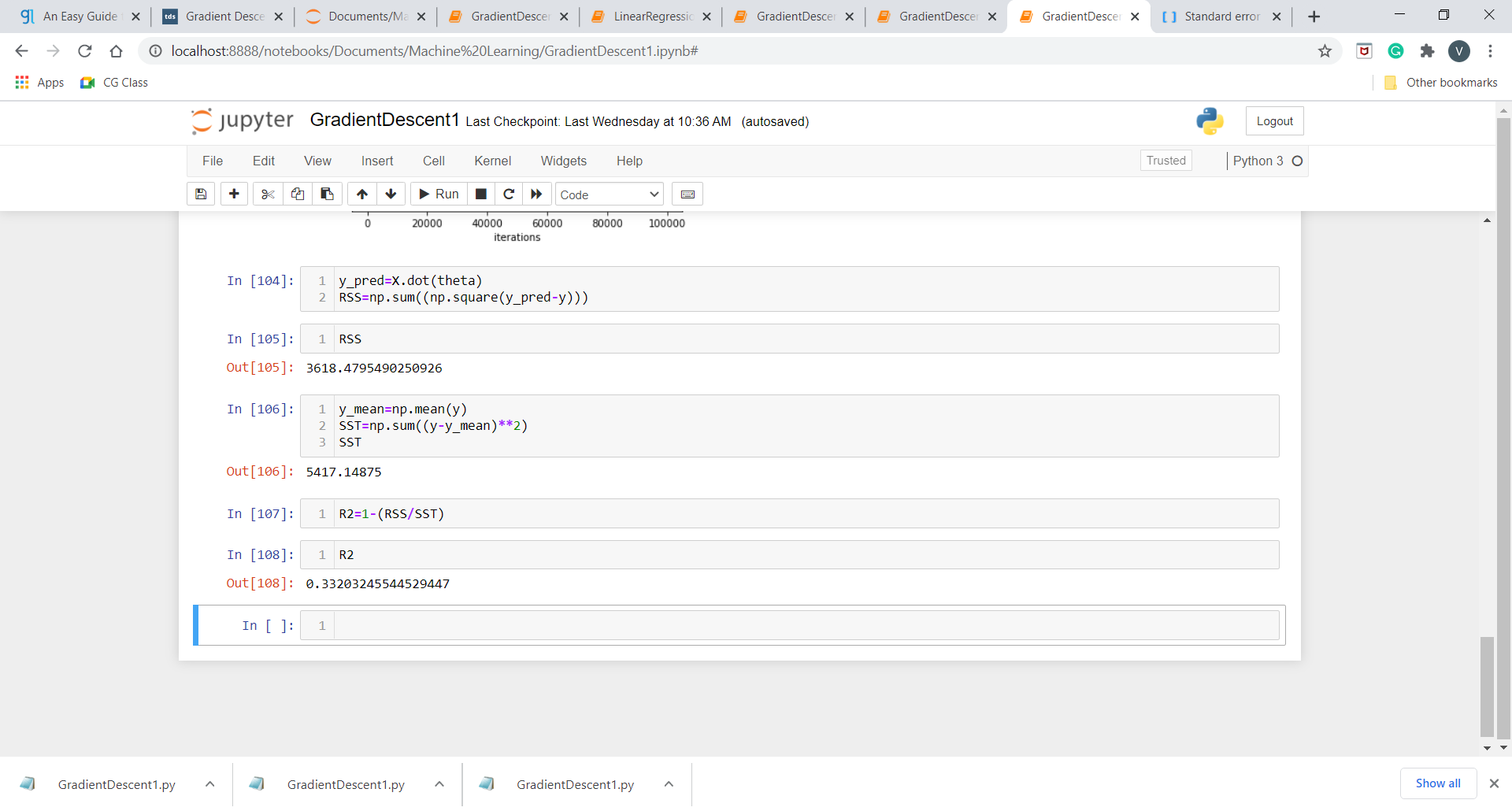
R2

# In[ ]:

**Output:**







1. **Sales~TV**

**Program:**

#!/usr/bin/env python

# coding: utf-8

# In[64]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# In[65]:

col\_list = ["TV", "radio","newspaper","sales"]

data = pd.read\_csv("Advertising.csv", usecols=col\_list)

# In[66]:

data["sales"] #y-axis

# In[67]:

data["TV"] #x-axis

# In[68]:

x1=data.TV.values

x1=x1.reshape(200,1)

x0=np.ones([200,1])

X=np.column\_stack((x0,x1))

print(X.shape)

y=data.sales.values

y=y.reshape(200,1)

theta=np.random.randn(2,1)

theta

# In[69]:

def cal\_cost(theta,X,y):

m=len(y)

y\_pred=X.dot(theta)

cost=np.sum((np.square(y\_pred-y)))/(2\*m)

return cost

# In[70]:

def gradient\_descent(X,y,theta,alpha=0.01,iterations=100):

m=len(y)

cost\_history=np.zeros(iterations)

theta\_history=np.zeros((iterations,2))

for it in range(iterations):

prediction=np.dot(X,theta)

theta=theta-(1/m)\*alpha\*(X.T.dot((prediction-y)))

theta\_history[it,:]=theta.T

cost\_history[it]=cal\_cost(theta,X,y)

return theta,cost\_history,theta\_history

# In[81]:

theta,cost\_history,theta\_history=gradient\_descent(X,y,theta,0.000005,5000000)

print(cost\_history.shape)

iter1=np.arange(0,5000000)

plt.scatter(iter1,cost\_history,color='red')

plt.xlabel("iterations")

plt.ylabel("cost")

plt.title("GradientDescent[Sales~TV]")

plt.show()

# In[82]:

plt.plot(iter1,cost\_history,color='red')

plt.xlabel("iterations")

plt.ylabel("cost")

plt.title("GradientDescent[Sales~TV]")

plt.show()

# In[83]:

print(theta[0][0])

print(theta[1][0])

# In[84]:

y\_pred=X.dot(theta)

RSS=np.sum((np.square(y\_pred-y)))

RSS

# In[85]:

y\_mean=np.mean(y)

SST=np.sum((y-y\_mean)\*\*2)

SST

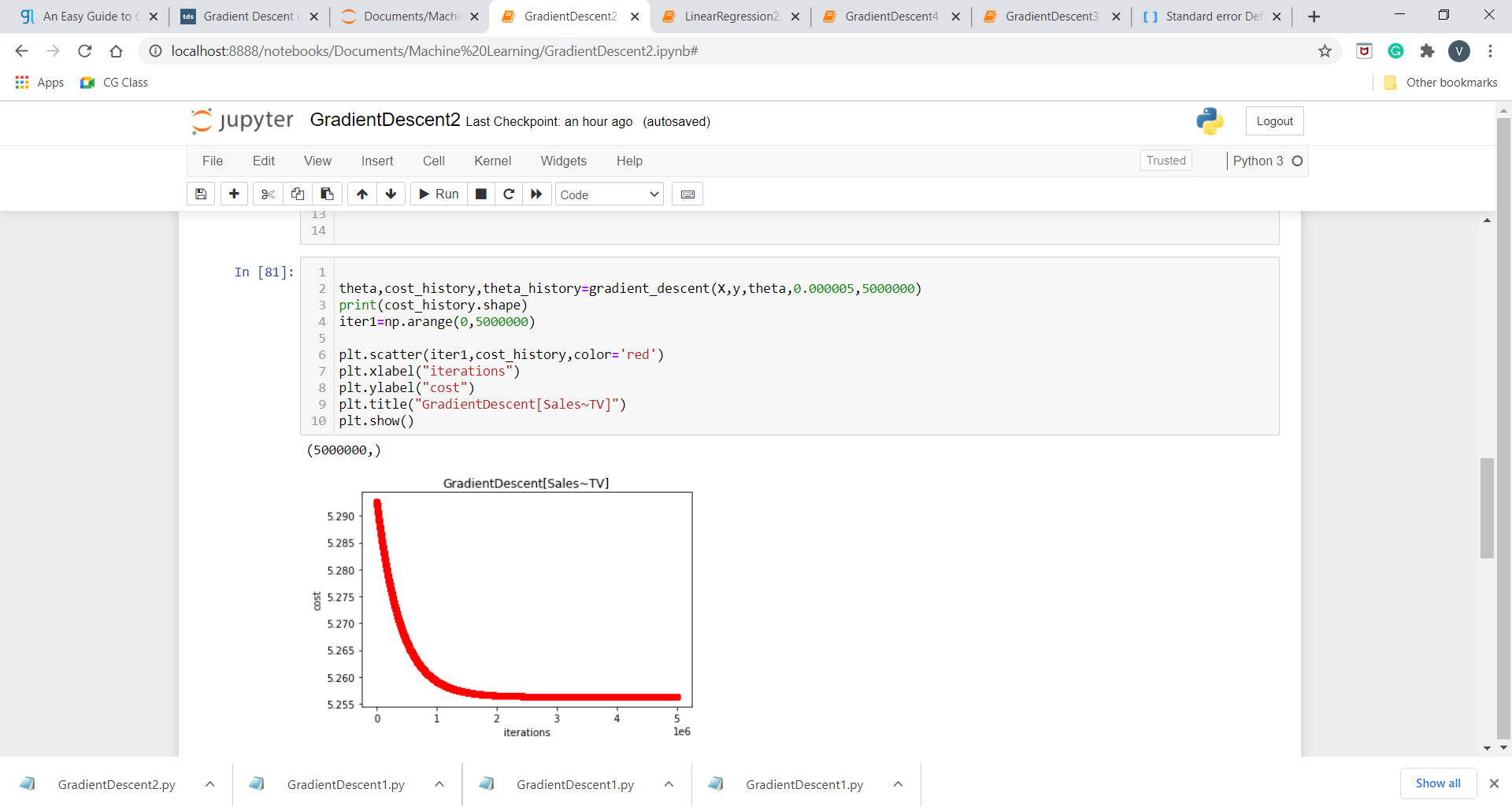
# In[86]:

R2=1-(RSS/SST)

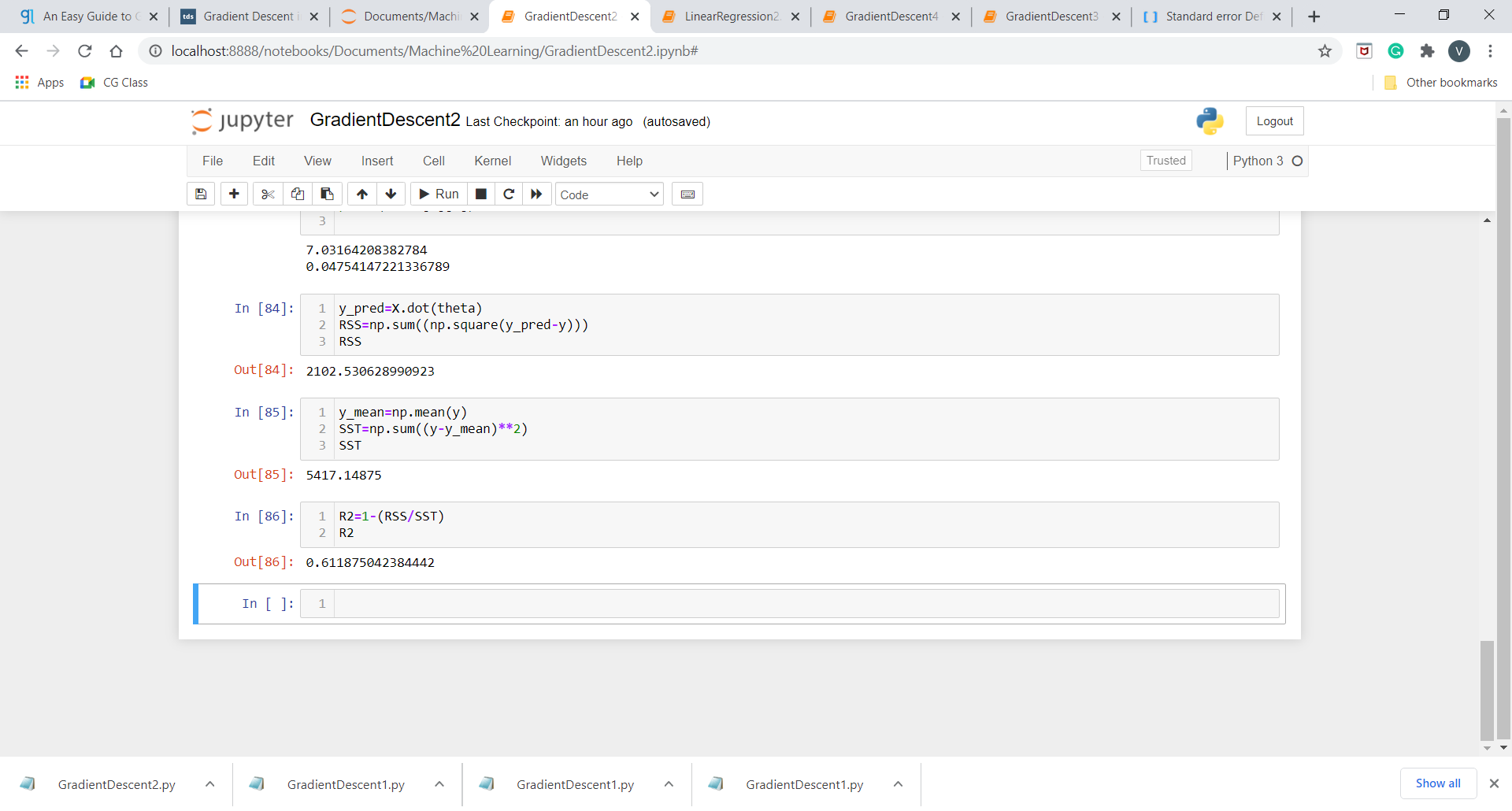
R2

# In[ ]:

**Output:**







1. **Sales~Newspaper**

**Program:**

#!/usr/bin/env python

# coding: utf-8

# In[45]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# In[46]:

col\_list = ["TV", "radio","newspaper","sales"]

data = pd.read\_csv("Advertising.csv", usecols=col\_list)

# In[47]:

data["sales"] #y-axis

# In[48]:

data["newspaper"] #x-axis

# In[49]:

x1=data.newspaper.values

x1=x1.reshape(200,1)

x0=np.ones([200,1])

X=np.column\_stack((x0,x1))

print(X.shape)

y=data.sales.values

y=y.reshape(200,1)

theta=np.random.randn(2,1)

theta

# In[50]:

def cal\_cost(theta,X,y):

m=len(y)

y\_pred=X.dot(theta)

cost=np.sum((np.square(y\_pred-y)))/(2\*m)

return cost

# In[51]:

def gradient\_descent(X,y,theta,alpha=0.01,iterations=100):

m=len(y)

cost\_history=np.zeros(iterations)

theta\_history=np.zeros((iterations,2))

for it in range(iterations):

prediction=np.dot(X,theta)

theta=theta-(1/m)\*alpha\*(X.T.dot((prediction-y)))

theta\_history[it,:]=theta.T

cost\_history[it]=cal\_cost(theta,X,y)

return theta,cost\_history,theta\_history

# In[65]:

theta,cost\_history,theta\_history=gradient\_descent(X,y,theta,0.000005,5000000)

print(cost\_history.shape)

iter1=np.arange(0,5000000)

print(theta[0][0])

print(theta[1][0])

plt.plot(iter1,cost\_history,color='red')

plt.xlabel("iterations")

plt.ylabel("cost")

plt.title("GradientDescent[Sales~newspaper]")

plt.show()

# In[66]:

print(theta[0][0])

print(theta[1][0])

# In[68]:

y\_pred=X.dot(theta)

RSS=np.sum((np.square(y\_pred-y)))

RSS

# In[69]:

y\_mean=np.mean(y)

SST=np.sum((y-y\_mean)\*\*2)

SST

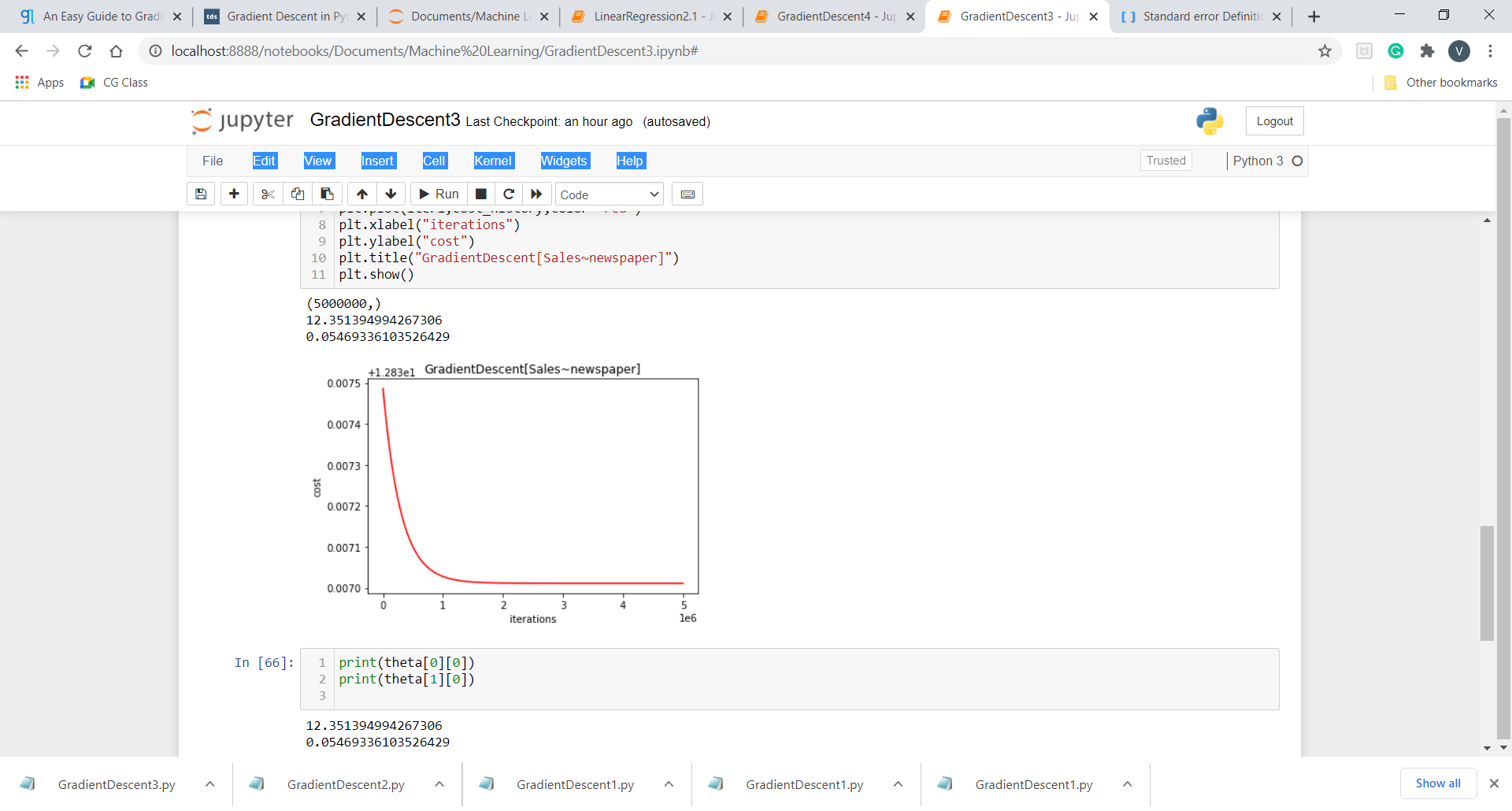
# In[71]:

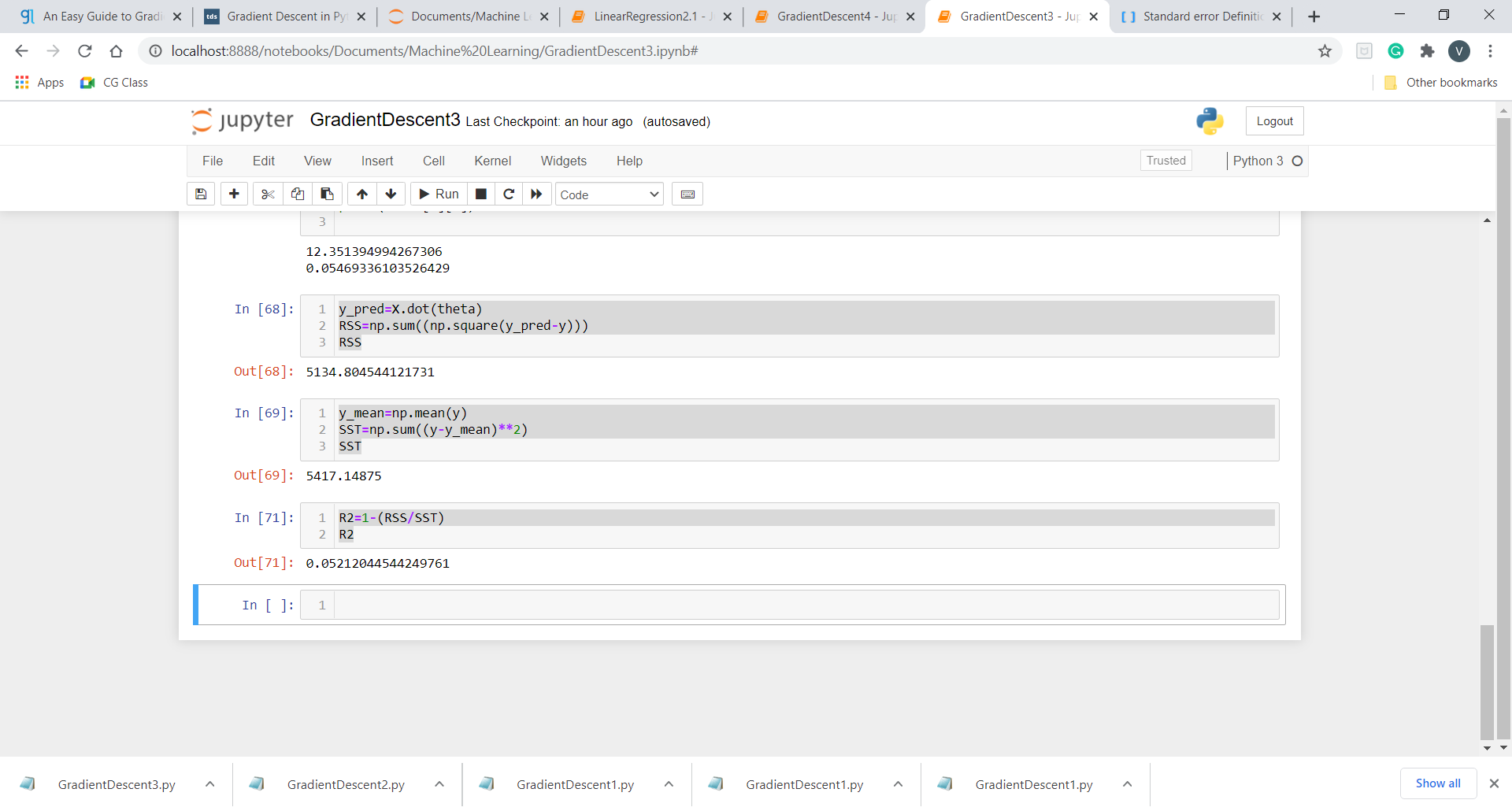
R2=1-(RSS/SST)

R2

# In[ ]:

**Output:**





1. **Bonus Question**

**Program:**

#!/usr/bin/env python

# coding: utf-8

# In[56]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# In[57]:

col\_list = ["TV", "radio","newspaper","sales"]

data = pd.read\_csv("Advertising.csv", usecols=col\_list)

# In[58]:

x1=data.radio.values

x1=x1.reshape(200,1)

x2=data.TV.values

x2=x2.reshape(200,1)

x3=data.newspaper.values

x3=x3.reshape(200,1)

x0=np.ones([200,1])

X=np.column\_stack((x0,x1,x2,x3))

print(X.shape)

y=data.sales.values

y=y.reshape(200,1)

theta=np.random.randn(4,1)

theta

# In[59]:

def cal\_cost(theta,X,y):

m=len(y)

y\_pred=X.dot(theta)

cost=np.sum((np.square(y\_pred-y)))/(2\*m)

return cost

# In[60]:

def gradient\_descent(X,y,theta,alpha=0.01,iterations=100):

m=len(y)

cost\_history=np.zeros(iterations)

theta\_history=np.zeros((iterations,4))

for it in range(iterations):

prediction=np.dot(X,theta)

theta=theta-(1/m)\*alpha\*(X.T.dot((prediction-y)))

theta\_history[it,:]=theta.T

cost\_history[it]=cal\_cost(theta,X,y)

return theta,cost\_history,theta\_history

# In[72]:

theta,cost\_history,theta\_history=gradient\_descent(X,y,theta,0.00006,1000000)

print(cost\_history.shape)

iter1=np.arange(0,1000000)

plt.scatter(iter1,cost\_history,color='red')

plt.xlabel("iterations")

plt.ylabel("cost")

plt.title("GradientDescent[Sales~Radio,TV,Newspaper]")

plt.show()

# In[73]:

plt.plot(iter1,cost\_history,color='red')

plt.xlabel("iterations")

plt.ylabel("cost")

plt.title("GradientDescent[Sales~Radio,TV,Newspaper]")

plt.show()

# In[74]:

print(theta)

# In[ ]:

y\_pred=X.dot(theta)

RSS=np.sum((np.square(y\_pred-y)))

RSS

# In[ ]:

y\_mean=np.mean(y)

SST=np.sum((y-y\_mean)\*\*2)

SST

# In[ ]:

R2=1-(RSS/SST)

R2

**Output:**

