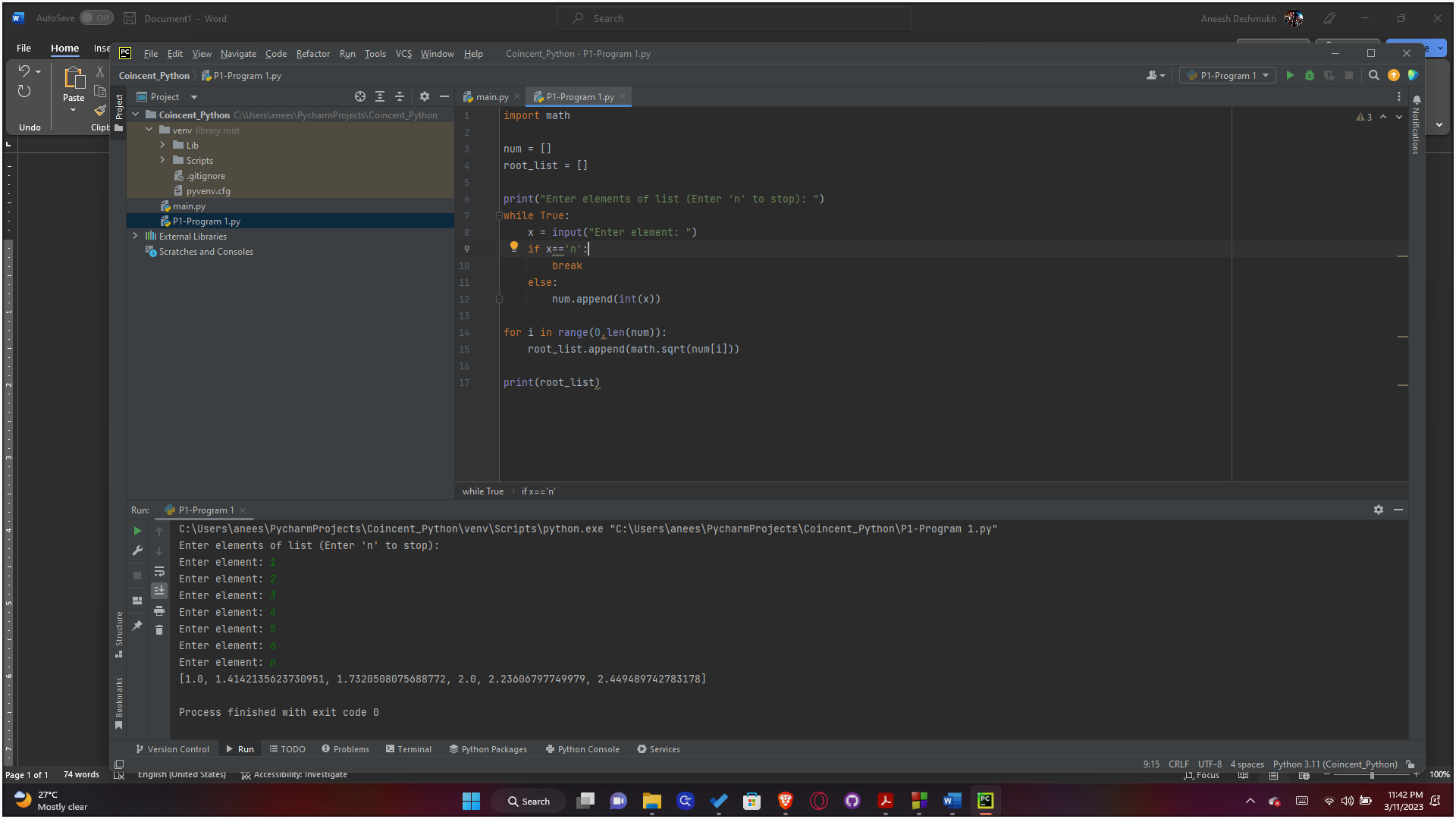
**PART 1:**

1. Take a list of elements from the user and find the square root of each member in the list and store in it another list and store it in another list and print that list.

Program:

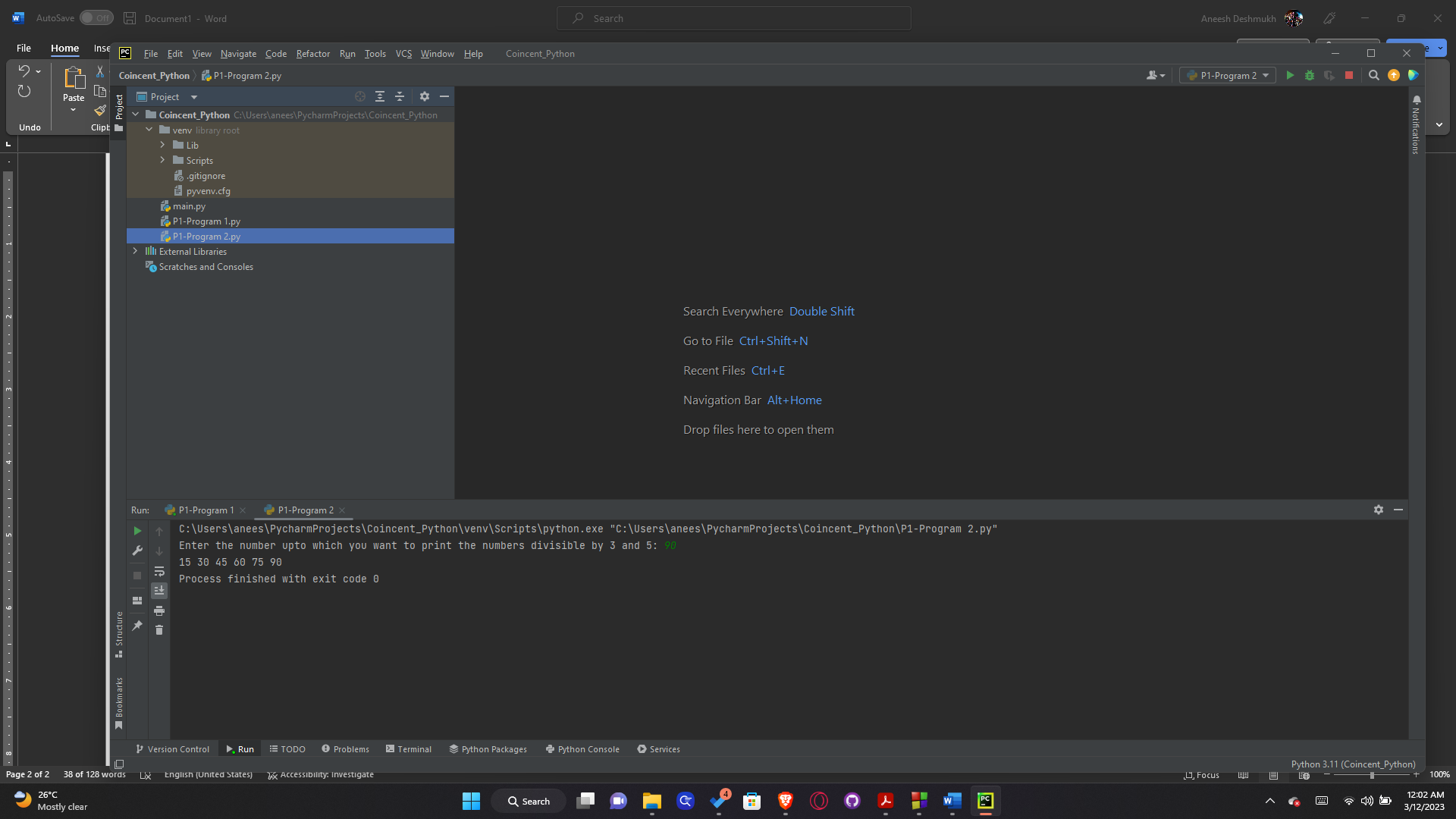
import math  
  
num = []  
root\_list = []  
  
print("Enter elements of list (Enter 'n' to stop): ")  
while True:  
 x = input("Enter element: ")  
 if x=='n':  
 break  
 else:  
 num.append(int(x))  
  
for i in range(0,len(num)):  
 root\_list.append(math.sqrt(num[i]))  
  
print(root\_list)

Output:

1. Write a function that prints all the numbers divisible by 3 and 5.

Program:

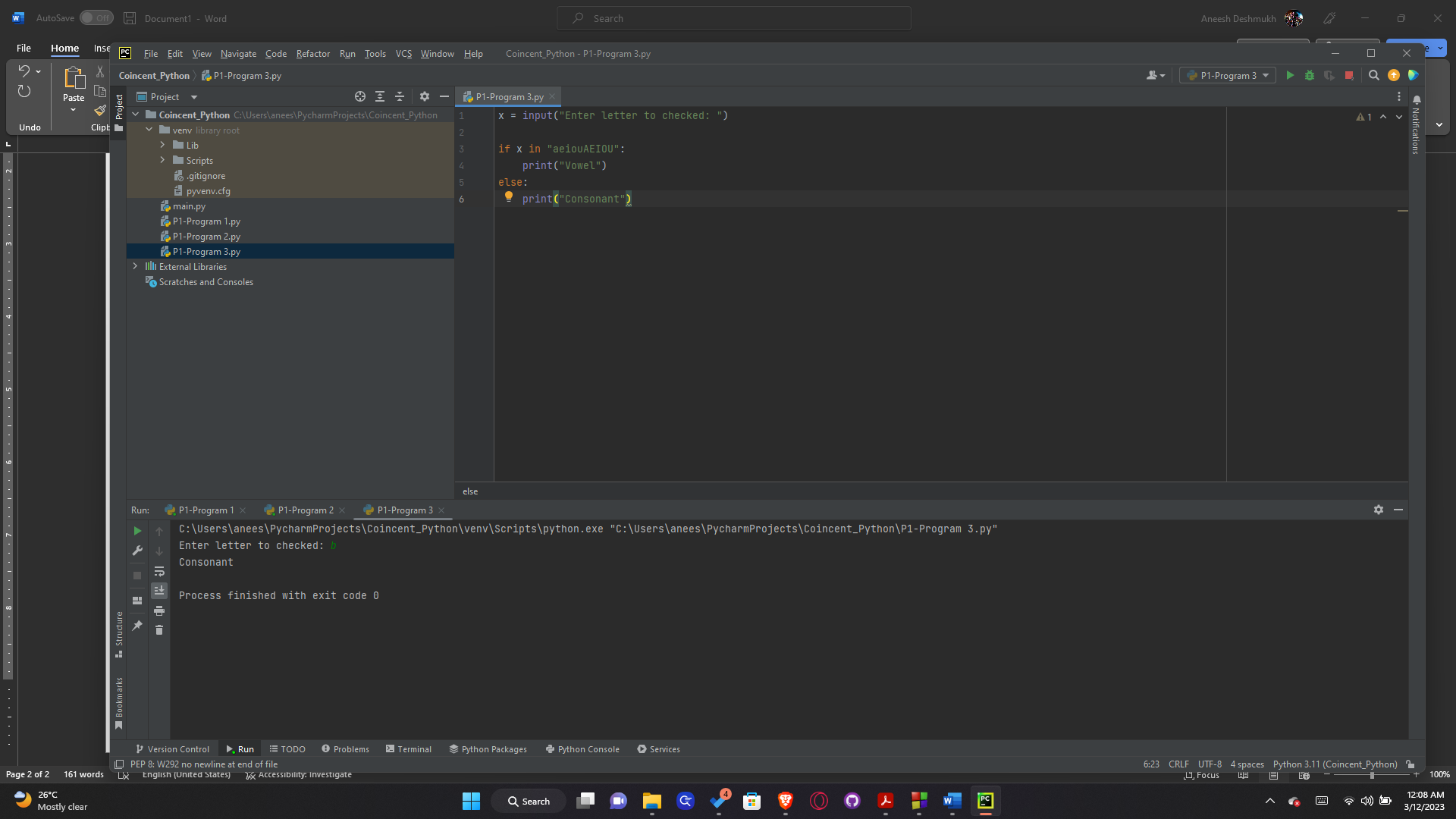
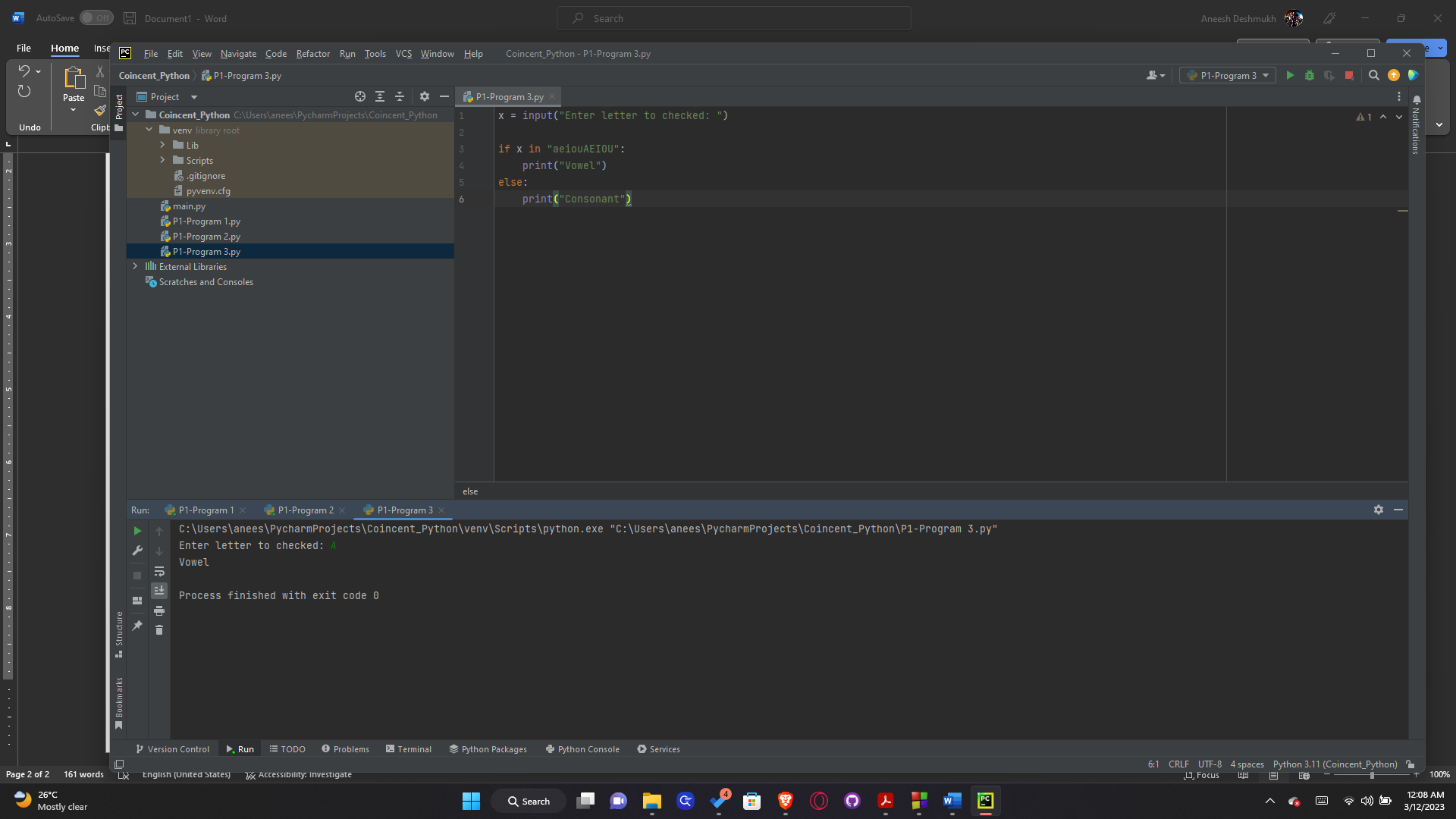
**n = int(input("Enter the number upto which you want to print the numbers divisible by 3 and 5: "))  
  
for i in range(1,n+1):  
 if i % 3 == 0 and i % 5 == 0:  
 print(i, end=" ")**

Output:

1. Write a program to check whether a given letter is a vowel or a consonant.

Program:

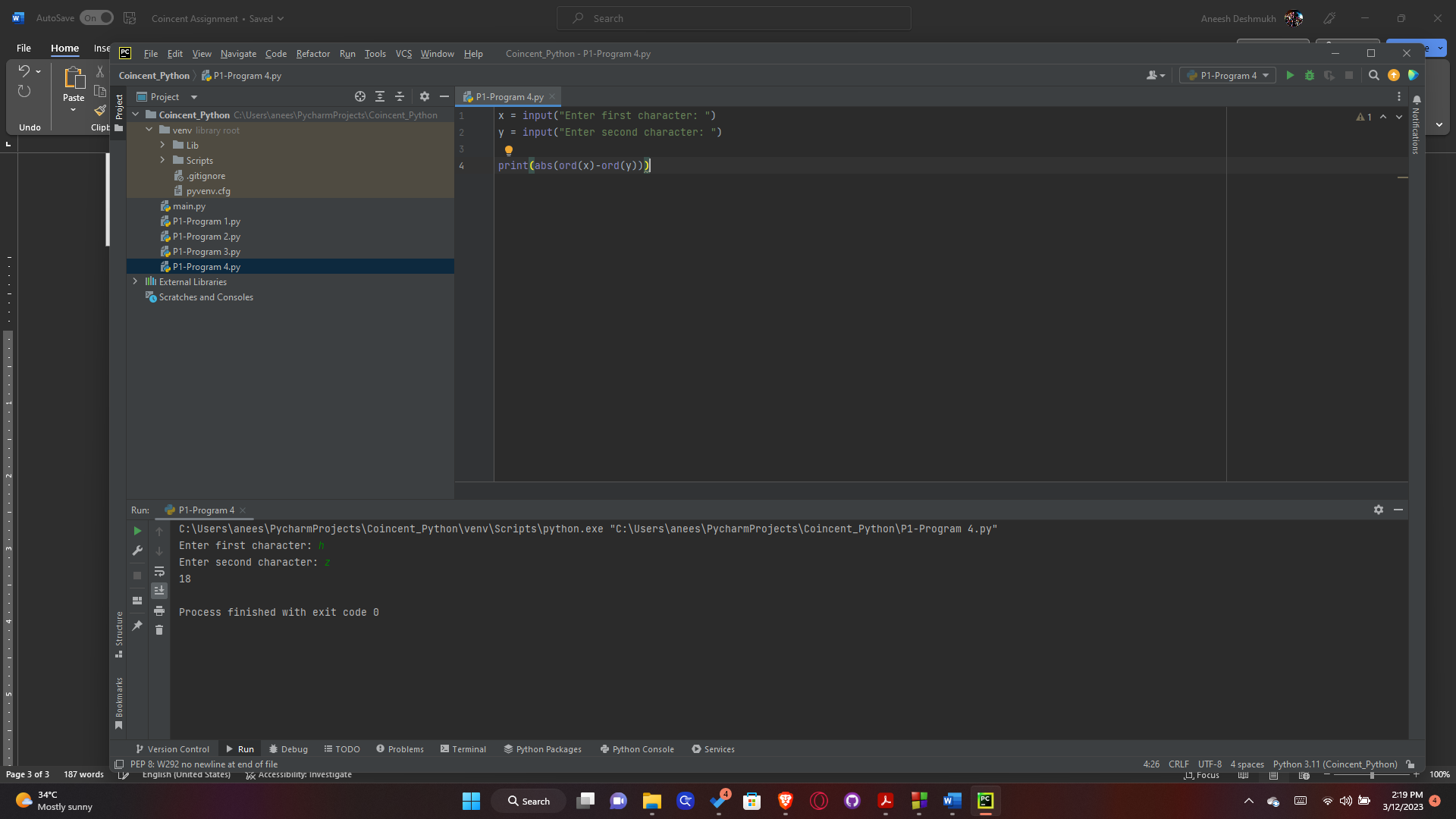
x = input("Enter letter to checked: ")  
  
if x in "aeiouAEIOU":  
 print("Vowel")  
else:  
 print("Consonant")

Output:

1. Calculate the distance between any two characters given by user.

Program:

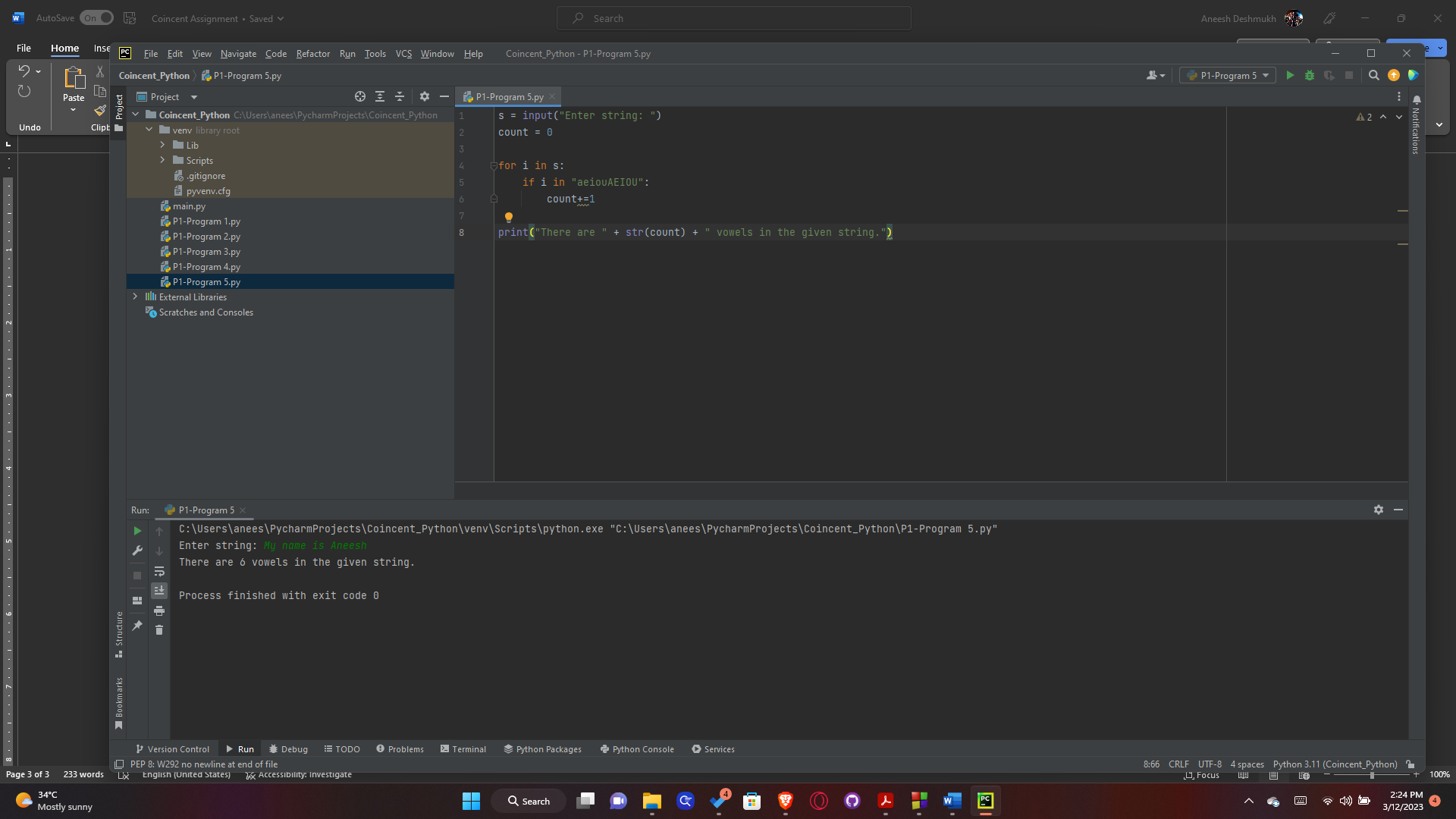
x = input("Enter first character: ")  
y = input("Enter second character: ")  
  
print(abs(ord(x)-ord(y)))

Output:

1. Write a function which returns the number of vowels present in the given string.

Program:

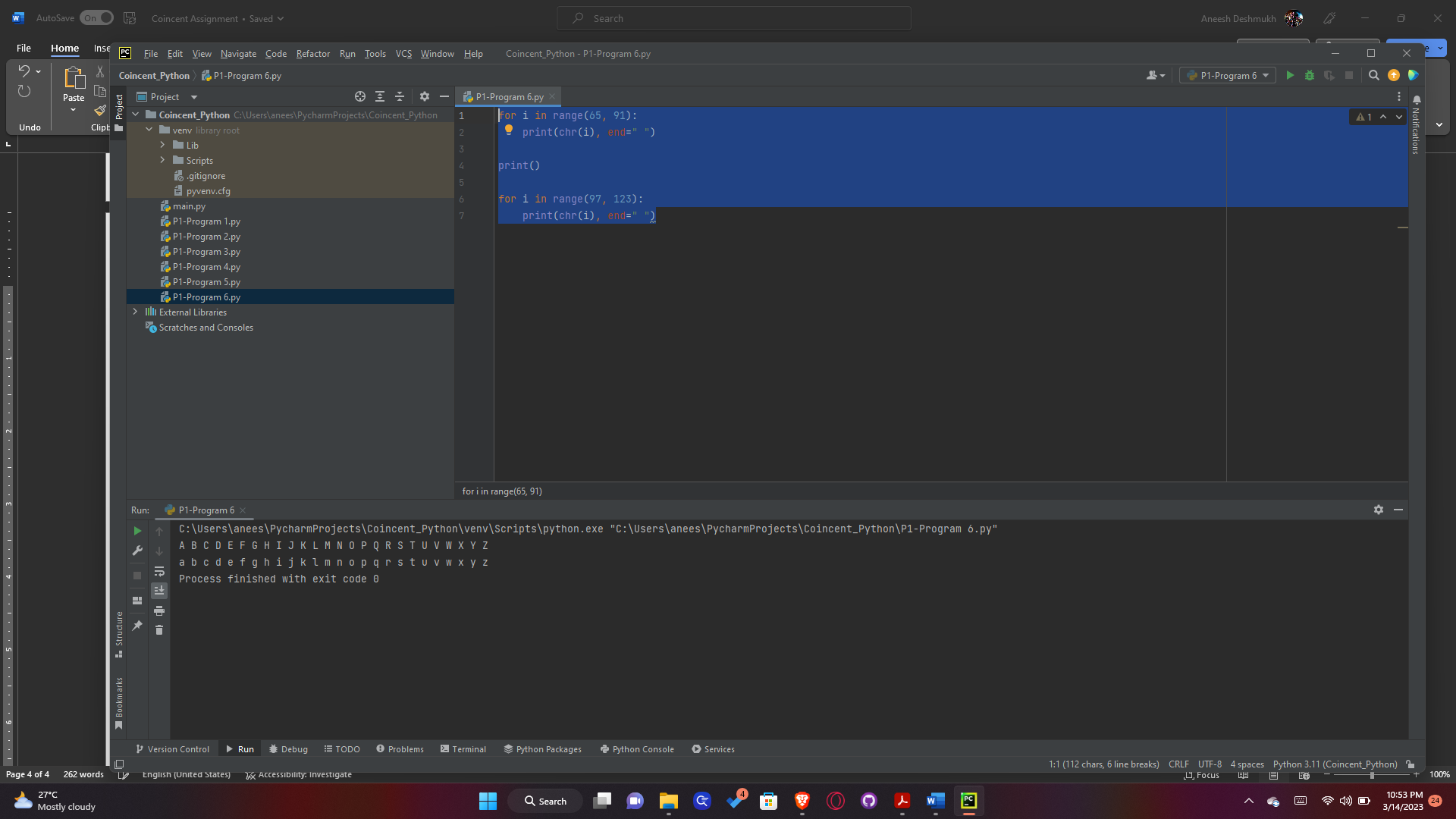
s = input("Enter string: ")  
count = 0  
  
for i in s:  
 if i in "aeiouAEIOU":  
 count+=1  
  
print("There are " + str(count) + " vowels in the given string.")

Output:

1. Print all the alphabets using loop and ascii code.

Program:

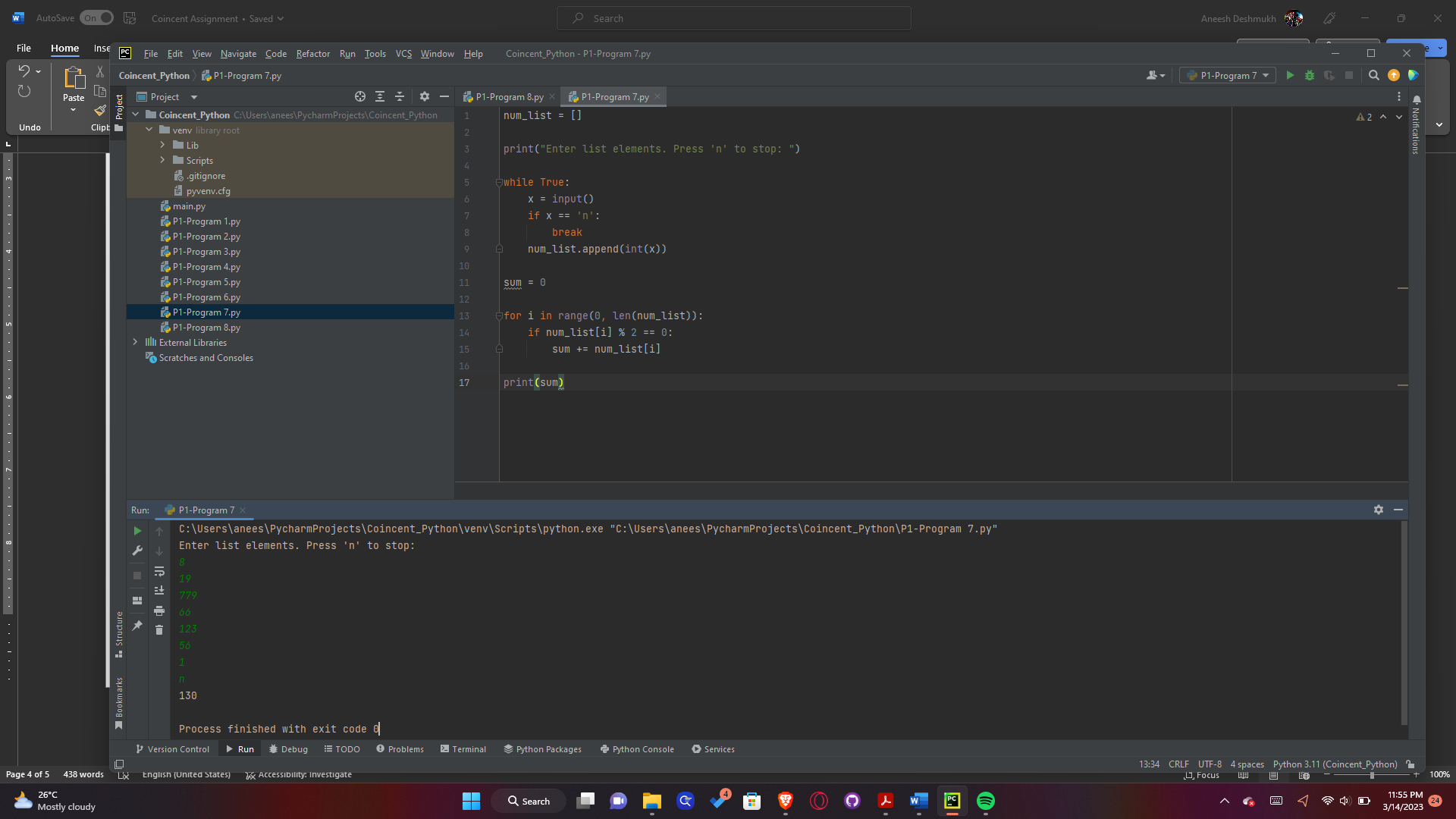
for i in range(65, 91):  
 print(chr(i), end=" ")  
  
print()  
  
for i in range(97, 123):  
 print(chr(i), end=" ")

Output:

1. Write a program to find the sum of all even numbers in a list.

Program:

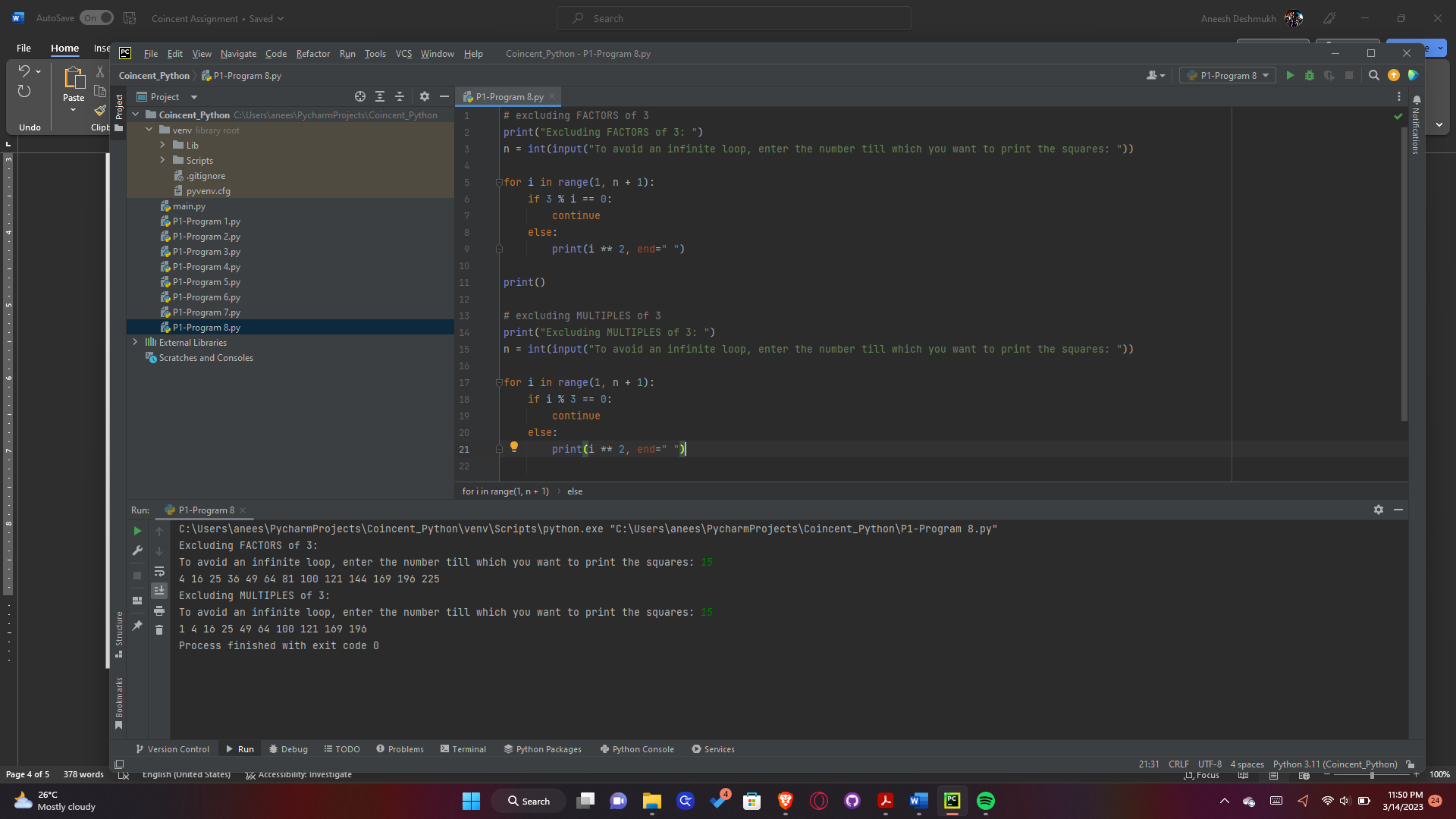
num\_list = []  
  
print("Enter list elements. Press 'n' to stop: ")  
  
while True:  
 x = input()  
 if x == 'n':  
 break  
 num\_list.append(int(x))  
  
sum = 0  
  
for i in range(0, len(num\_list)):  
 if num\_list[i] % 2 == 0:  
 sum += num\_list[i]  
  
print(sum)

Output:

1. Write a program to print the squares of all the numbers, except for factors of 3.

Program:

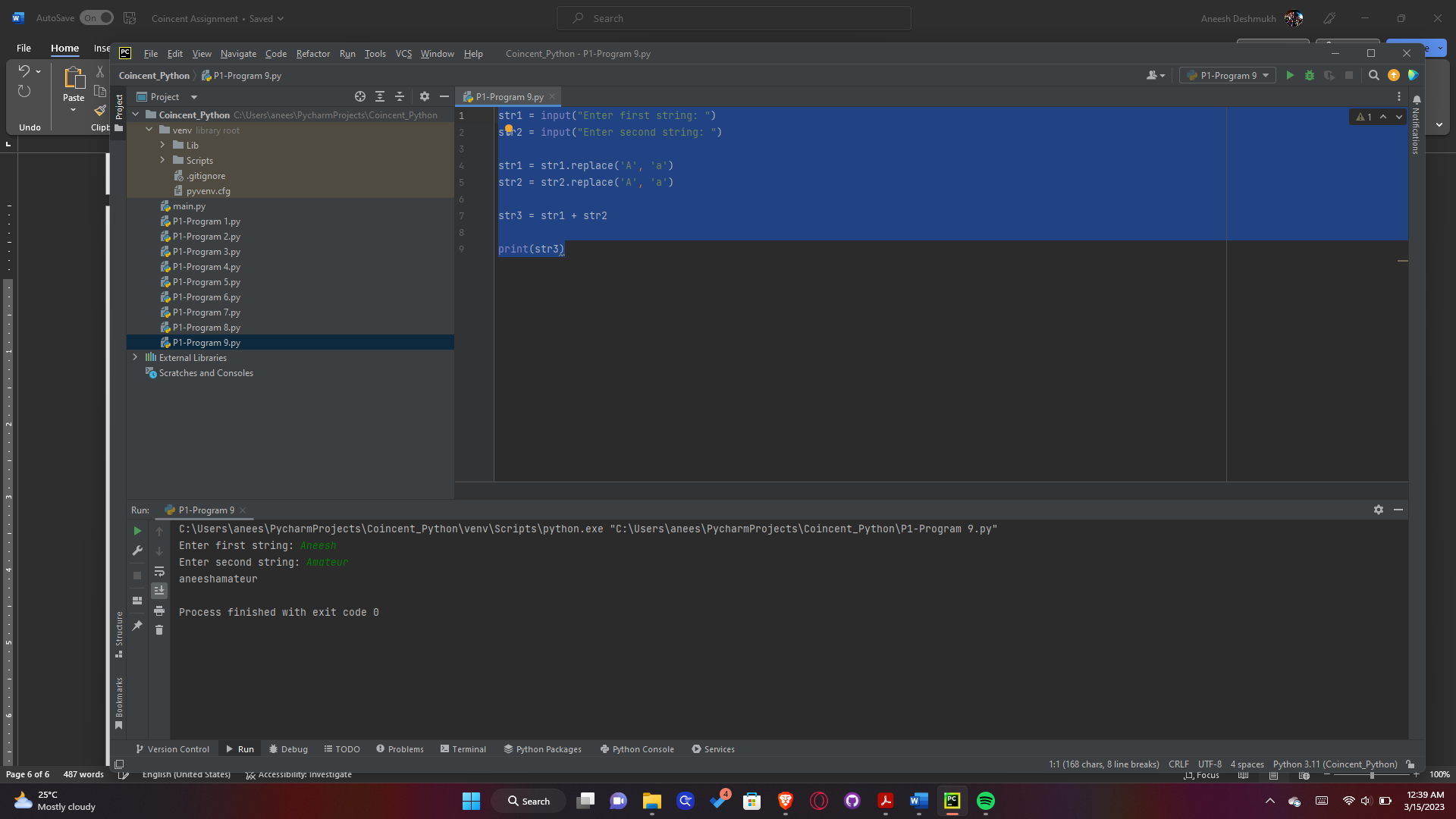
# excluding FACTORS of 3  
print("Excluding FACTORS of 3: ")  
n = int(input("To avoid an infinite loop, enter the number till which you want to print the squares: "))  
  
for i in range(1, n + 1):  
 if 3 % i == 0:  
 continue  
 else:  
 print(i \*\* 2, end=" ")  
  
print()  
  
# excluding MULTIPLES of 3  
print("Excluding MULTIPLES of 3: ")  
n = int(input("To avoid an infinite loop, enter the number till which you want to print the squares: "))  
  
for i in range(1, n + 1):  
 if i % 3 == 0:  
 continue  
 else:  
 print(i \*\* 2, end=" ")

Output:

1. Take 2 strings from user and then replace all the A’s with a’s, and then concatenate the 2 strings and print.

Program:

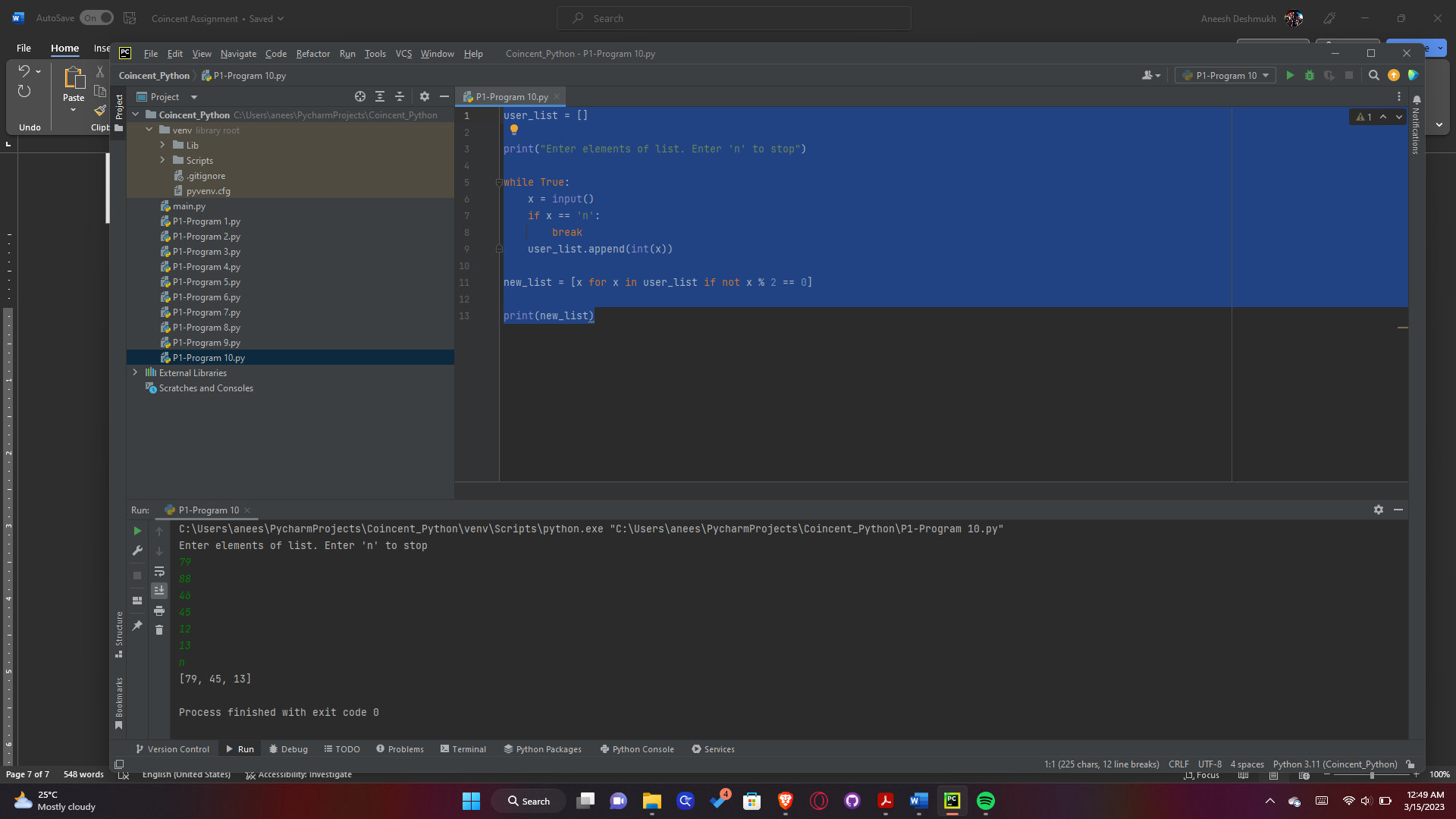
str1 = input("Enter first string: ")  
str2 = input("Enter second string: ")  
  
str1 = str1.replace('A', 'a')  
str2 = str2.replace('A', 'a')  
  
str3 = str1 + str2  
  
print(str3)

Output:

1. Write a program to get a list of odd numbers from the list of numbers given by user (use list comprehension).

Program:

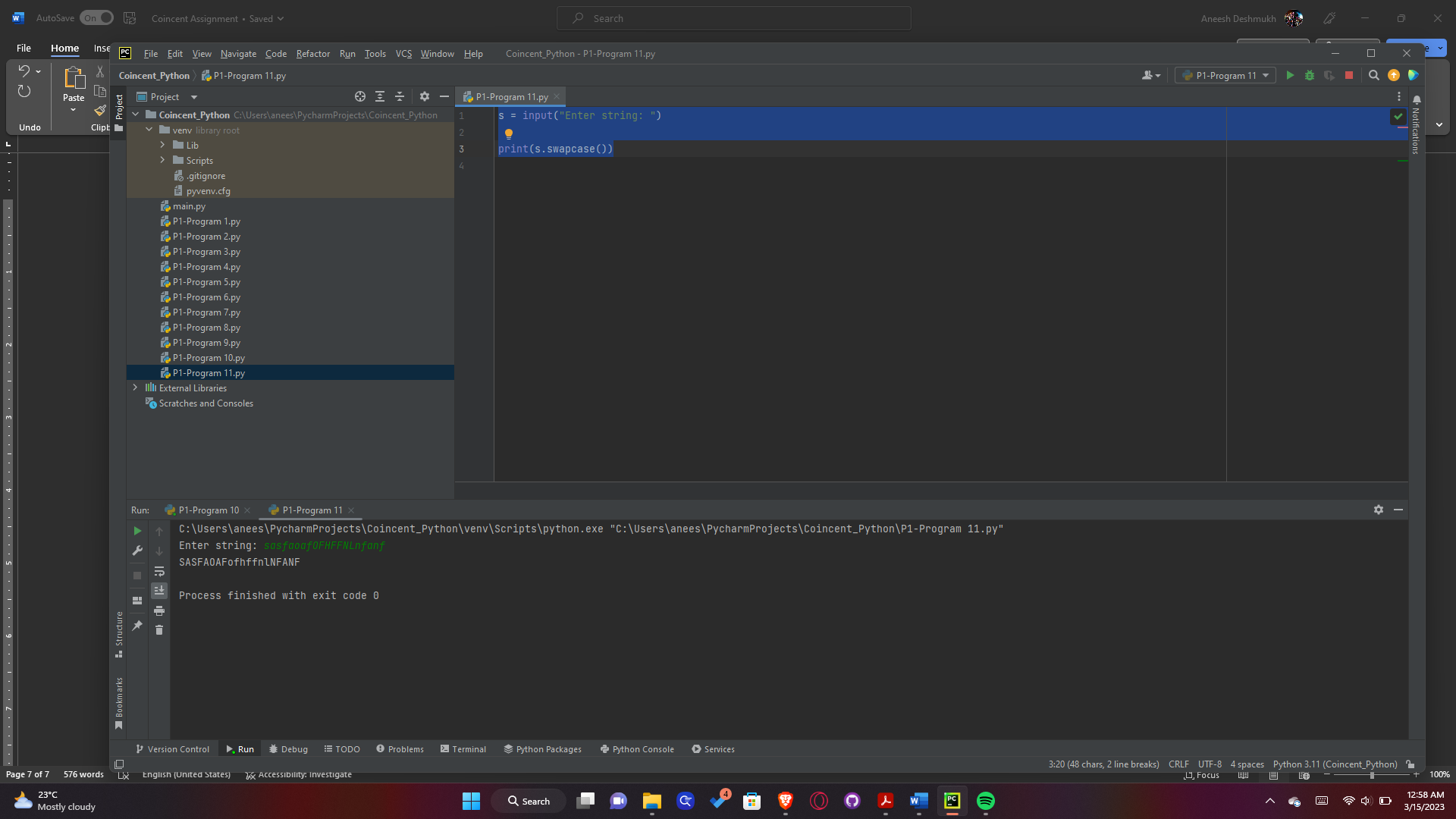
user\_list = []  
  
print("Enter elements of list. Enter 'n' to stop")  
  
while True:  
 x = input()  
 if x == 'n':  
 break  
 user\_list.append(int(x))  
  
new\_list = [x for x in user\_list if not x % 2 == 0]  
  
print(new\_list)

Output:

1. Write a program to print lower case letter when you have upper case letter in string and vice-versa.

Program:

s = input("Enter string: ")  
  
print(s.swapcase())

Output:

**PART 2:**

**Iris Flower Classification using different algorithms**

Code:

import pandas as pd

import numpy as np

import plotly

import plotly.express as px

import plotly.offline as pyo

import cufflinks as cf

from plotly.offline import init\_notebook\_mode,plot,iplot

import matplotlib.pyplot as plt

%matplotlib inline

from sklearn.metrics import accuracy\_score

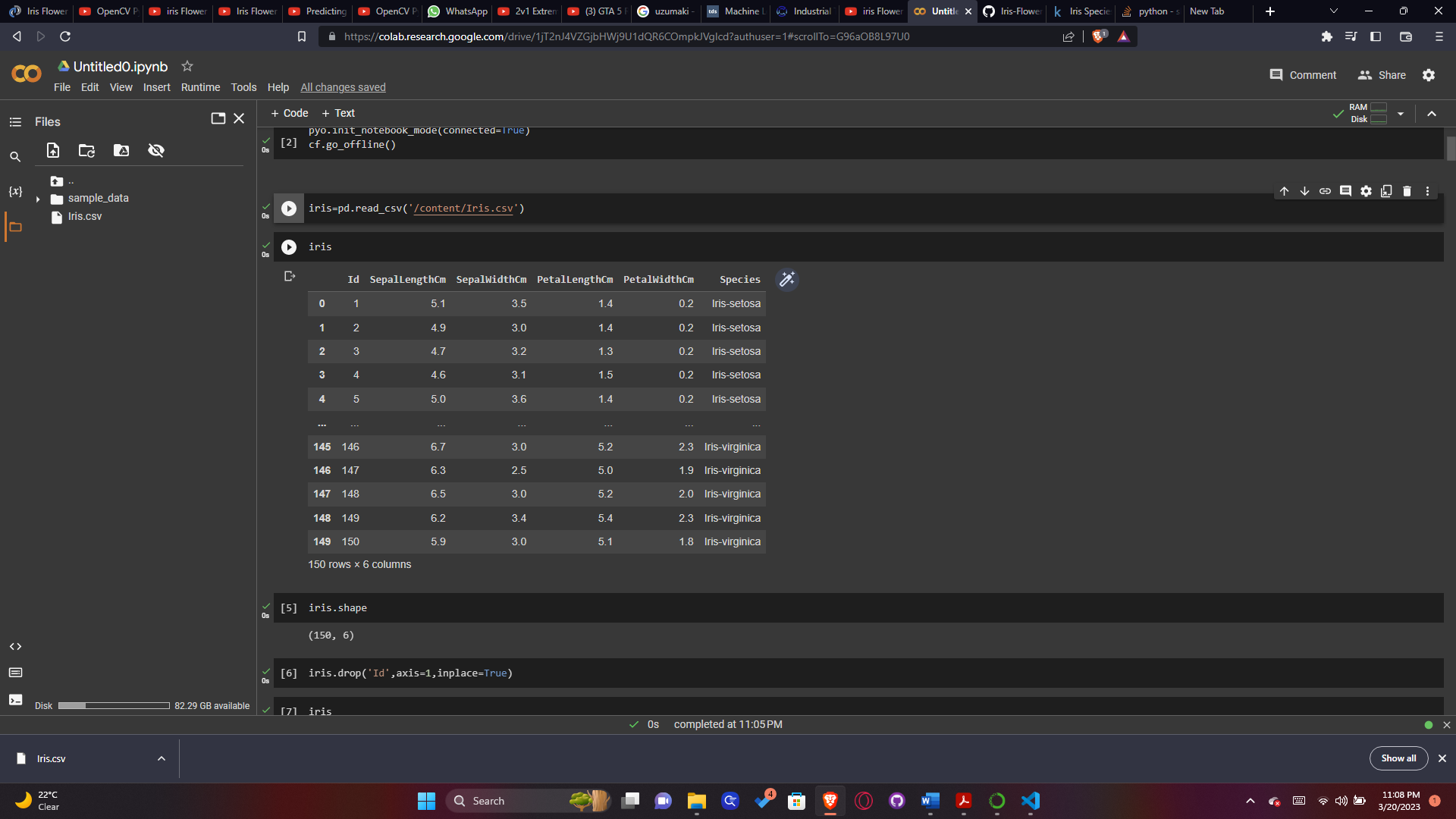
import os

pyo.init\_notebook\_mode(connected=True)

cf.go\_offline()

iris=pd.read\_csv('/content/Iris.csv')

iris

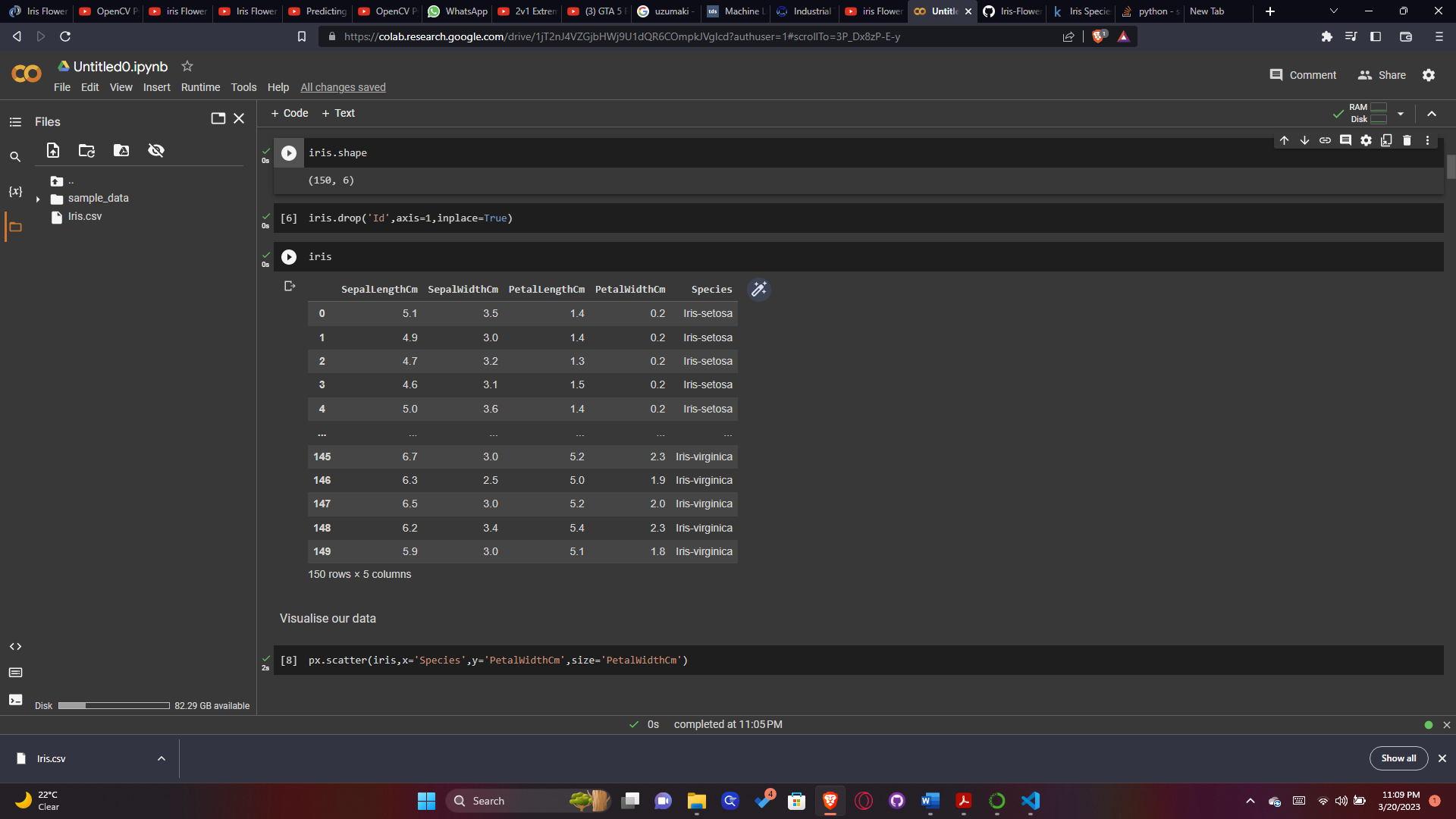


iris.shape

(150, 6)

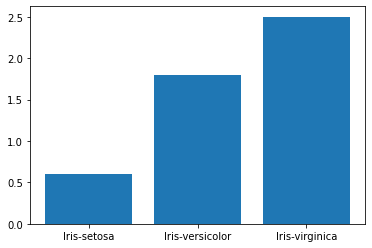
iris.drop('Id',axis=1,inplace=True)

iris



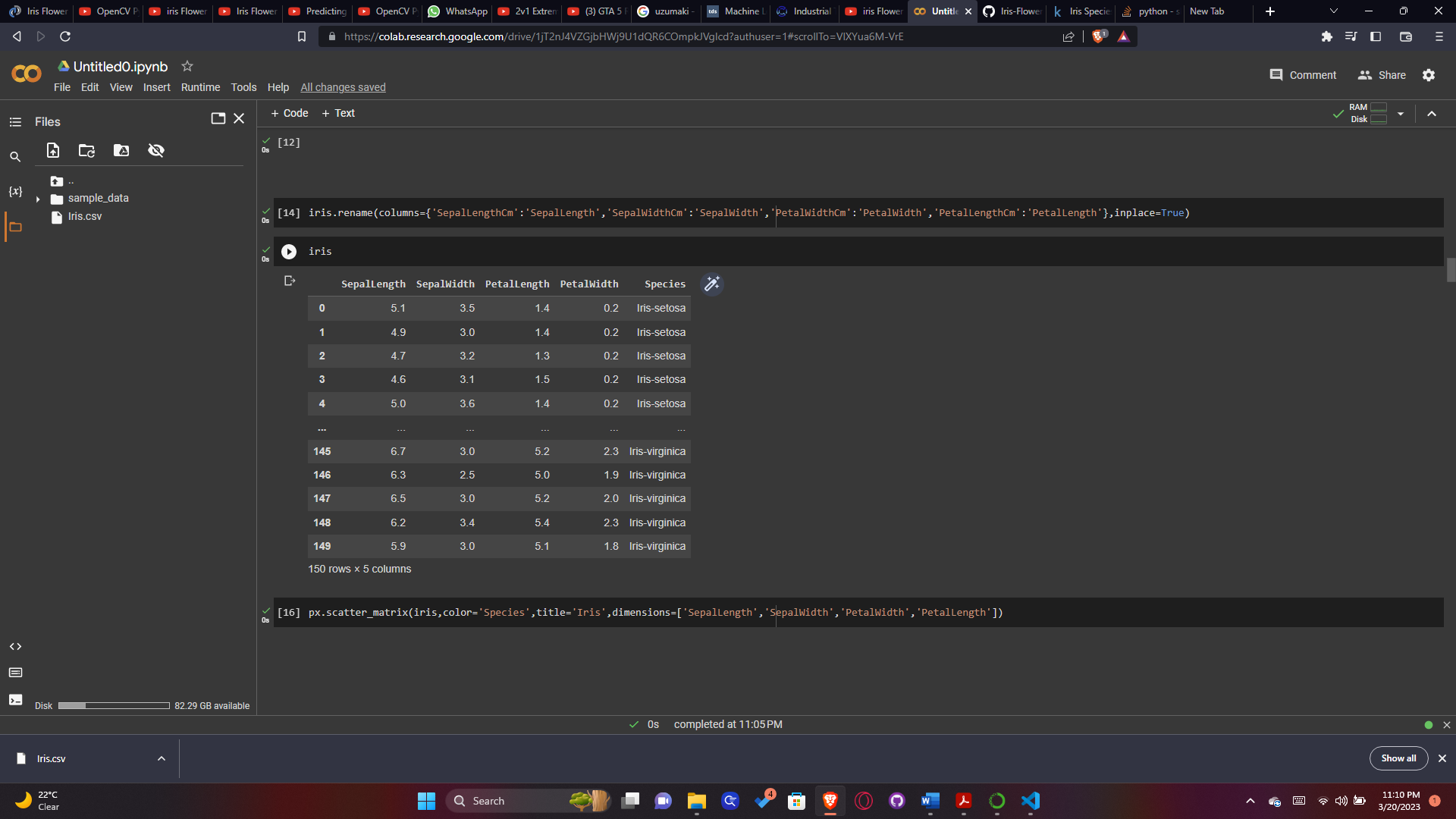
# Visualise our data

plt.bar(iris['Species'],iris['PetalWidthCm'])



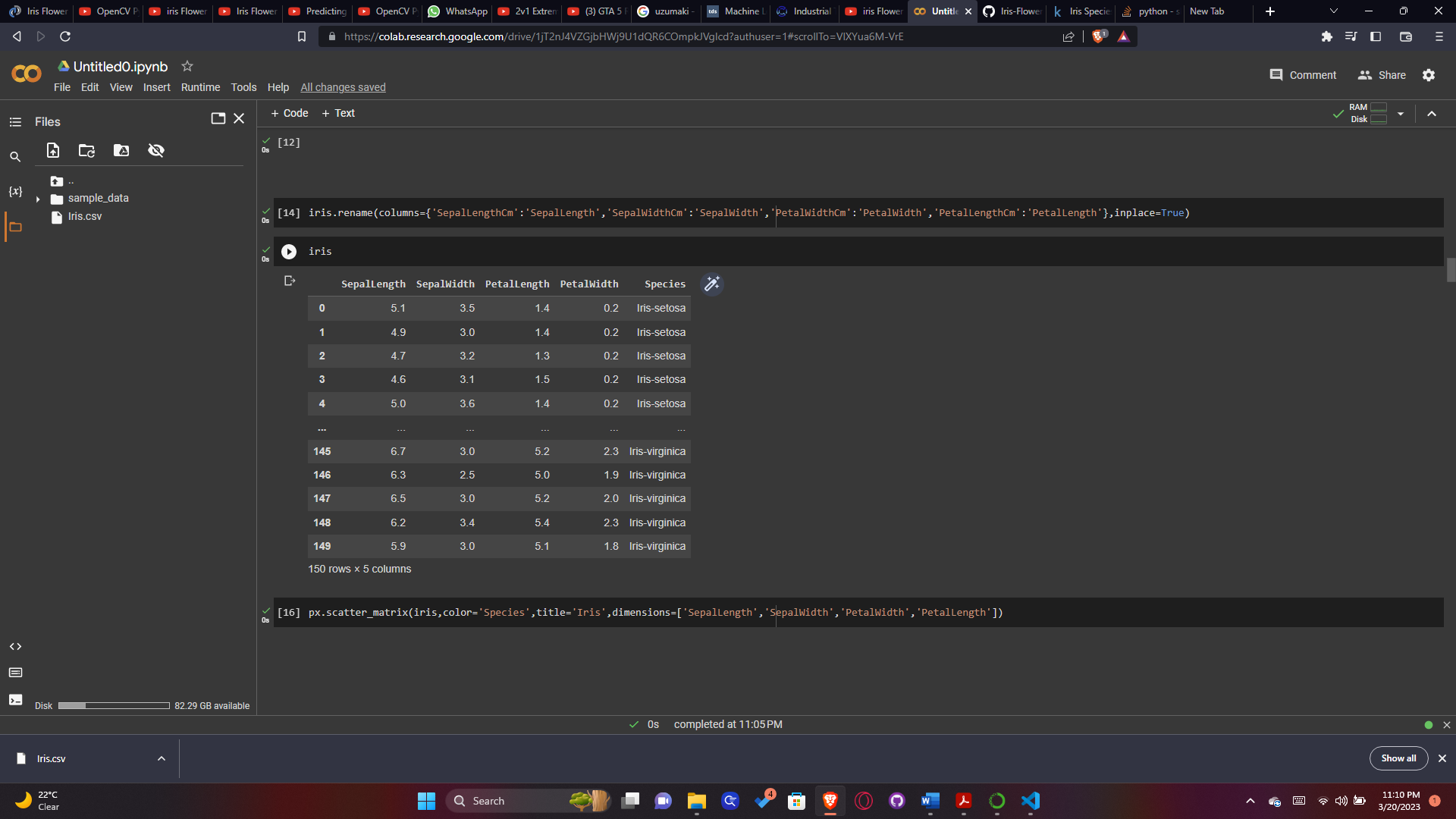
iris.rename(columns={'SepalLengthCm':'SepalLength','SepalWidthCm':'SepalWidth','PetalWidthCm':'PetalWidth','PetalLengthCm':'PetalLength'},inplace=True)

iris



# Data Preprocessing

iris



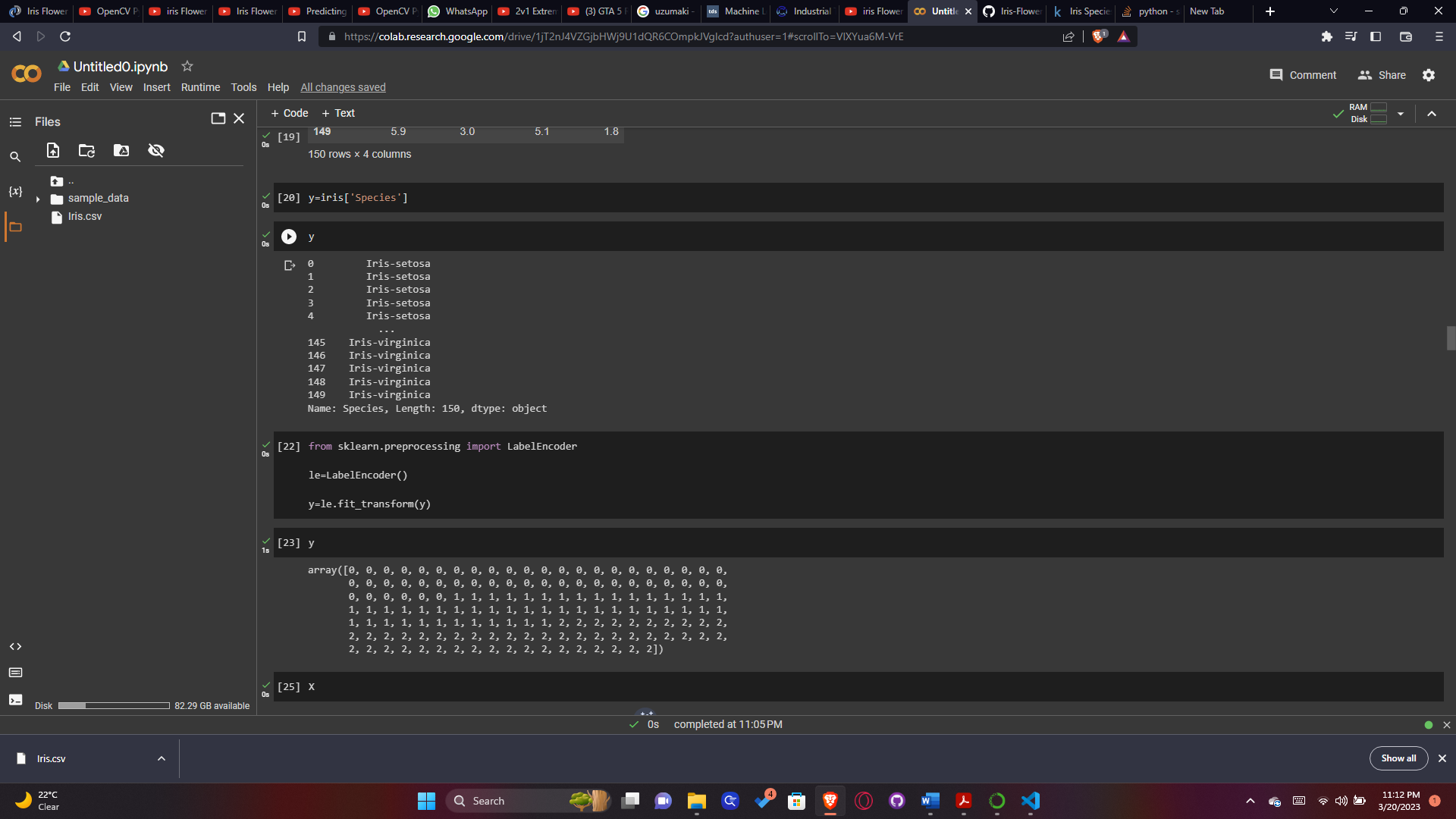
X=iris.drop(['Species'],axis=1)

X



y=iris['Species']

y



from sklearn.preprocessing import LabelEncoder

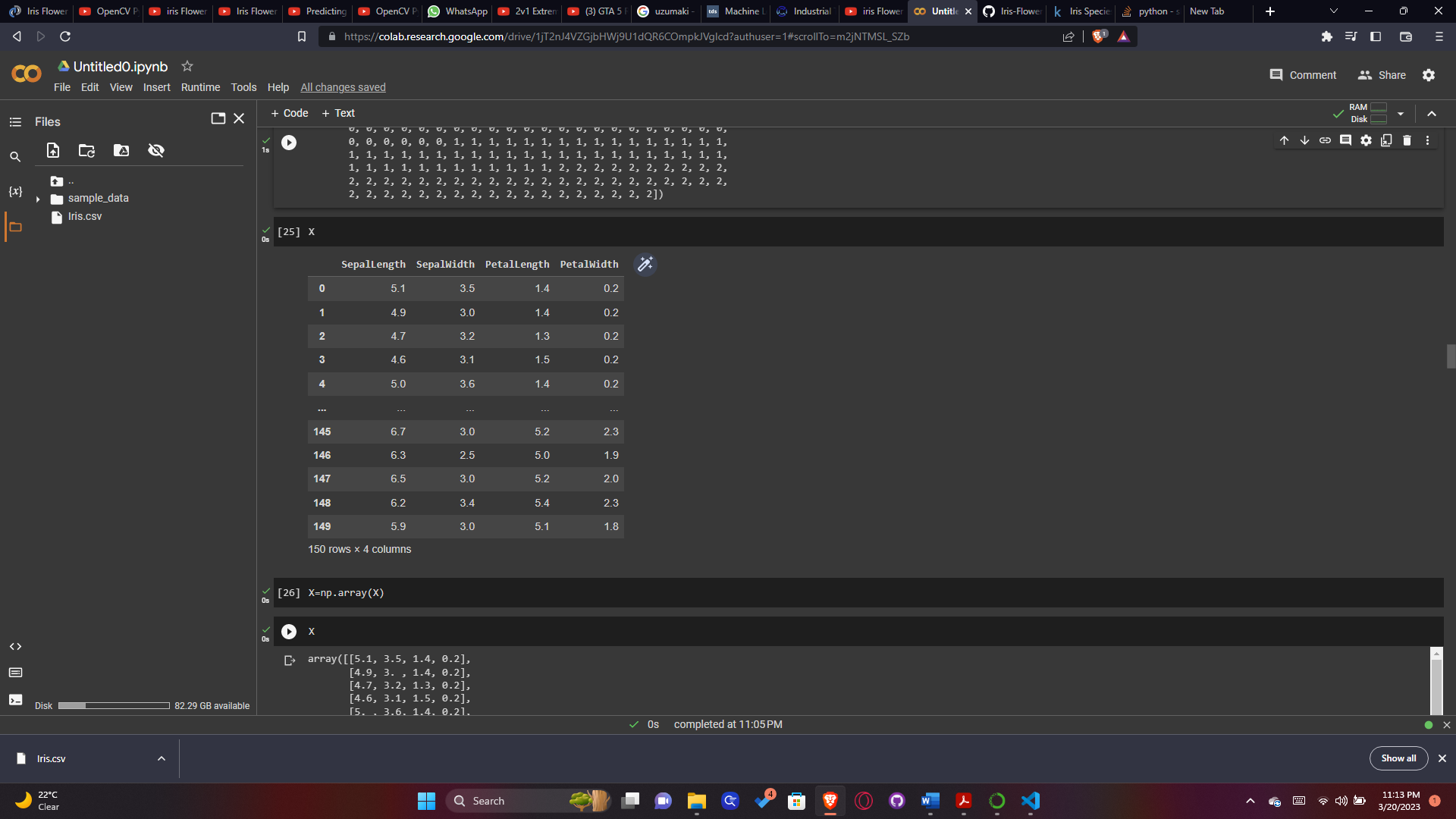
le=LabelEncoder()

y=le.fit\_transform(y)

y

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])

X



X=np.array(X)

X

array([[5.1, 3.5, 1.4, 0.2], [4.9, 3. , 1.4, 0.2], [4.7, 3.2, 1.3, 0.2], [4.6, 3.1, 1.5, 0.2], [5. , 3.6, 1.4, 0.2], [5.4, 3.9, 1.7, 0.4], [4.6, 3.4, 1.4, 0.3], [5. , 3.4, 1.5, 0.2], [4.4, 2.9, 1.4, 0.2], [4.9, 3.1, 1.5, 0.1], [5.4, 3.7, 1.5, 0.2], [4.8, 3.4, 1.6, 0.2], [4.8, 3. , 1.4, 0.1], [4.3, 3. , 1.1, 0.1], [5.8, 4. , 1.2, 0.2], [5.7, 4.4, 1.5, 0.4], [5.4, 3.9, 1.3, 0.4], [5.1, 3.5, 1.4, 0.3], [5.7, 3.8, 1.7, 0.3], [5.1, 3.8, 1.5, 0.3], [5.4, 3.4, 1.7, 0.2], [5.1, 3.7, 1.5, 0.4], [4.6, 3.6, 1. , 0.2], [5.1, 3.3, 1.7, 0.5], [4.8, 3.4, 1.9, 0.2], [5. , 3. , 1.6, 0.2], [5. , 3.4, 1.6, 0.4], [5.2, 3.5, 1.5, 0.2], [5.2, 3.4, 1.4, 0.2], [4.7, 3.2, 1.6, 0.2], [4.8, 3.1, 1.6, 0.2], [5.4, 3.4, 1.5, 0.4], [5.2, 4.1, 1.5, 0.1], [5.5, 4.2, 1.4, 0.2], [4.9, 3.1, 1.5, 0.1], [5. , 3.2, 1.2, 0.2], [5.5, 3.5, 1.3, 0.2], [4.9, 3.1, 1.5, 0.1], [4.4, 3. , 1.3, 0.2], [5.1, 3.4, 1.5, 0.2], [5. , 3.5, 1.3, 0.3], [4.5, 2.3, 1.3, 0.3], [4.4, 3.2, 1.3, 0.2], [5. , 3.5, 1.6, 0.6], [5.1, 3.8, 1.9, 0.4], [4.8, 3. , 1.4, 0.3], [5.1, 3.8, 1.6, 0.2], [4.6, 3.2, 1.4, 0.2], [5.3, 3.7, 1.5, 0.2], [5. , 3.3, 1.4, 0.2], [7. , 3.2, 4.7, 1.4], [6.4, 3.2, 4.5, 1.5], [6.9, 3.1, 4.9, 1.5], [5.5, 2.3, 4. , 1.3], [6.5, 2.8, 4.6, 1.5], [5.7, 2.8, 4.5, 1.3], [6.3, 3.3, 4.7, 1.6], [4.9, 2.4, 3.3, 1. ], [6.6, 2.9, 4.6, 1.3], [5.2, 2.7, 3.9, 1.4], [5. , 2. , 3.5, 1. ], [5.9, 3. , 4.2, 1.5], [6. , 2.2, 4. , 1. ], [6.1, 2.9, 4.7, 1.4], [5.6, 2.9, 3.6, 1.3], [6.7, 3.1, 4.4, 1.4], [5.6, 3. , 4.5, 1.5], [5.8, 2.7, 4.1, 1. ], [6.2, 2.2, 4.5, 1.5], [5.6, 2.5, 3.9, 1.1], [5.9, 3.2, 4.8, 1.8], [6.1, 2.8, 4. , 1.3], [6.3, 2.5, 4.9, 1.5], [6.1, 2.8, 4.7, 1.2], [6.4, 2.9, 4.3, 1.3], [6.6, 3. , 4.4, 1.4], [6.8, 2.8, 4.8, 1.4], [6.7, 3. , 5. , 1.7], [6. , 2.9, 4.5, 1.5], [5.7, 2.6, 3.5, 1. ], [5.5, 2.4, 3.8, 1.1], [5.5, 2.4, 3.7, 1. ], [5.8, 2.7, 3.9, 1.2], [6. , 2.7, 5.1, 1.6], [5.4, 3. , 4.5, 1.5], [6. , 3.4, 4.5, 1.6], [6.7, 3.1, 4.7, 1.5], [6.3, 2.3, 4.4, 1.3], [5.6, 3. , 4.1, 1.3], [5.5, 2.5, 4. , 1.3], [5.5, 2.6, 4.4, 1.2], [6.1, 3. , 4.6, 1.4], [5.8, 2.6, 4. , 1.2], [5. , 2.3, 3.3, 1. ], [5.6, 2.7, 4.2, 1.3], [5.7, 3. , 4.2, 1.2], [5.7, 2.9, 4.2, 1.3], [6.2, 2.9, 4.3, 1.3], [5.1, 2.5, 3. , 1.1], [5.7, 2.8, 4.1, 1.3], [6.3, 3.3, 6. , 2.5], [5.8, 2.7, 5.1, 1.9], [7.1, 3. , 5.9, 2.1], [6.3, 2.9, 5.6, 1.8], [6.5, 3. , 5.8, 2.2], [7.6, 3. , 6.6, 2.1], [4.9, 2.5, 4.5, 1.7], [7.3, 2.9, 6.3, 1.8], [6.7, 2.5, 5.8, 1.8], [7.2, 3.6, 6.1, 2.5], [6.5, 3.2, 5.1, 2. ], [6.4, 2.7, 5.3, 1.9], [6.8, 3. , 5.5, 2.1], [5.7, 2.5, 5. , 2. ], [5.8, 2.8, 5.1, 2.4], [6.4, 3.2, 5.3, 2.3], [6.5, 3. , 5.5, 1.8], [7.7, 3.8, 6.7, 2.2], [7.7, 2.6, 6.9, 2.3], [6. , 2.2, 5. , 1.5], [6.9, 3.2, 5.7, 2.3], [5.6, 2.8, 4.9, 2. ], [7.7, 2.8, 6.7, 2. ], [6.3, 2.7, 4.9, 1.8], [6.7, 3.3, 5.7, 2.1], [7.2, 3.2, 6. , 1.8], [6.2, 2.8, 4.8, 1.8], [6.1, 3. , 4.9, 1.8], [6.4, 2.8, 5.6, 2.1], [7.2, 3. , 5.8, 1.6], [7.4, 2.8, 6.1, 1.9], [7.9, 3.8, 6.4, 2. ], [6.4, 2.8, 5.6, 2.2], [6.3, 2.8, 5.1, 1.5], [6.1, 2.6, 5.6, 1.4], [7.7, 3. , 6.1, 2.3], [6.3, 3.4, 5.6, 2.4], [6.4, 3.1, 5.5, 1.8], [6. , 3. , 4.8, 1.8], [6.9, 3.1, 5.4, 2.1], [6.7, 3.1, 5.6, 2.4], [6.9, 3.1, 5.1, 2.3], [5.8, 2.7, 5.1, 1.9], [6.8, 3.2, 5.9, 2.3], [6.7, 3.3, 5.7, 2.5], [6.7, 3. , 5.2, 2.3], [6.3, 2.5, 5. , 1.9], [6.5, 3. , 5.2, 2. ], [6.2, 3.4, 5.4, 2.3], [5.9, 3. , 5.1, 1.8]])

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)

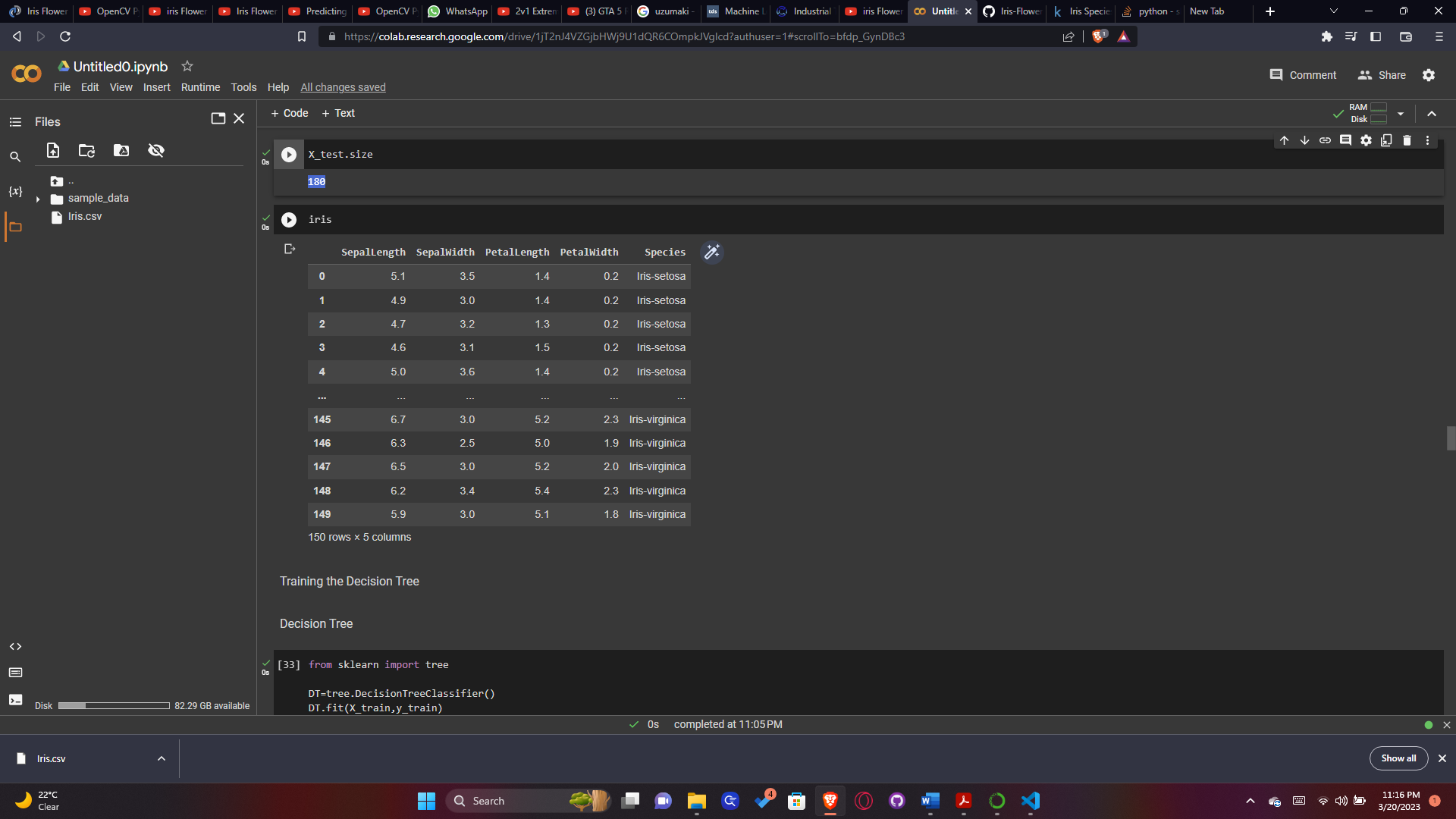
X\_test

array([[5.8, 2.8, 5.1, 2.4], [6. , 2.2, 4. , 1. ], [5.5, 4.2, 1.4, 0.2], [7.3, 2.9, 6.3, 1.8], [5. , 3.4, 1.5, 0.2], [6.3, 3.3, 6. , 2.5], [5. , 3.5, 1.3, 0.3], [6.7, 3.1, 4.7, 1.5], [6.8, 2.8, 4.8, 1.4], [6.1, 2.8, 4. , 1.3], [6.1, 2.6, 5.6, 1.4], [6.4, 3.2, 4.5, 1.5], [6.1, 2.8, 4.7, 1.2], [6.5, 2.8, 4.6, 1.5], [6.1, 2.9, 4.7, 1.4], [4.9, 3.1, 1.5, 0.1], [6. , 2.9, 4.5, 1.5], [5.5, 2.6, 4.4, 1.2], [4.8, 3. , 1.4, 0.3], [5.4, 3.9, 1.3, 0.4], [5.6, 2.8, 4.9, 2. ], [5.6, 3. , 4.5, 1.5], [4.8, 3.4, 1.9, 0.2], [4.4, 2.9, 1.4, 0.2], [6.2, 2.8, 4.8, 1.8], [4.6, 3.6, 1. , 0.2], [5.1, 3.8, 1.9, 0.4], [6.2, 2.9, 4.3, 1.3], [5. , 2.3, 3.3, 1. ], [5. , 3.4, 1.6, 0.4], [6.4, 3.1, 5.5, 1.8], [5.4, 3. , 4.5, 1.5], [5.2, 3.5, 1.5, 0.2], [6.1, 3. , 4.9, 1.8], [6.4, 2.8, 5.6, 2.2], [5.2, 2.7, 3.9, 1.4], [5.7, 3.8, 1.7, 0.3], [6. , 2.7, 5.1, 1.6], [5.9, 3. , 4.2, 1.5], [5.8, 2.6, 4. , 1.2], [6.8, 3. , 5.5, 2.1], [4.7, 3.2, 1.3, 0.2], [6.9, 3.1, 5.1, 2.3], [5. , 3.5, 1.6, 0.6], [5.4, 3.7, 1.5, 0.2]])

X\_test.size

180

iris



# Training the Decision Tree

# Decision Tree

from sklearn import tree

DT=tree.DecisionTreeClassifier()

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

y\_train.size

105

prediction\_DT=DT.predict(X\_test)

accuracy\_DT=accuracy\_score(y\_test,prediction\_DT)\*100

accuracy\_DT

97.77777777777777

y\_test

array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1, 0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 1, 1, 1, 2, 0, 2, 0, 0])

prediction\_DT

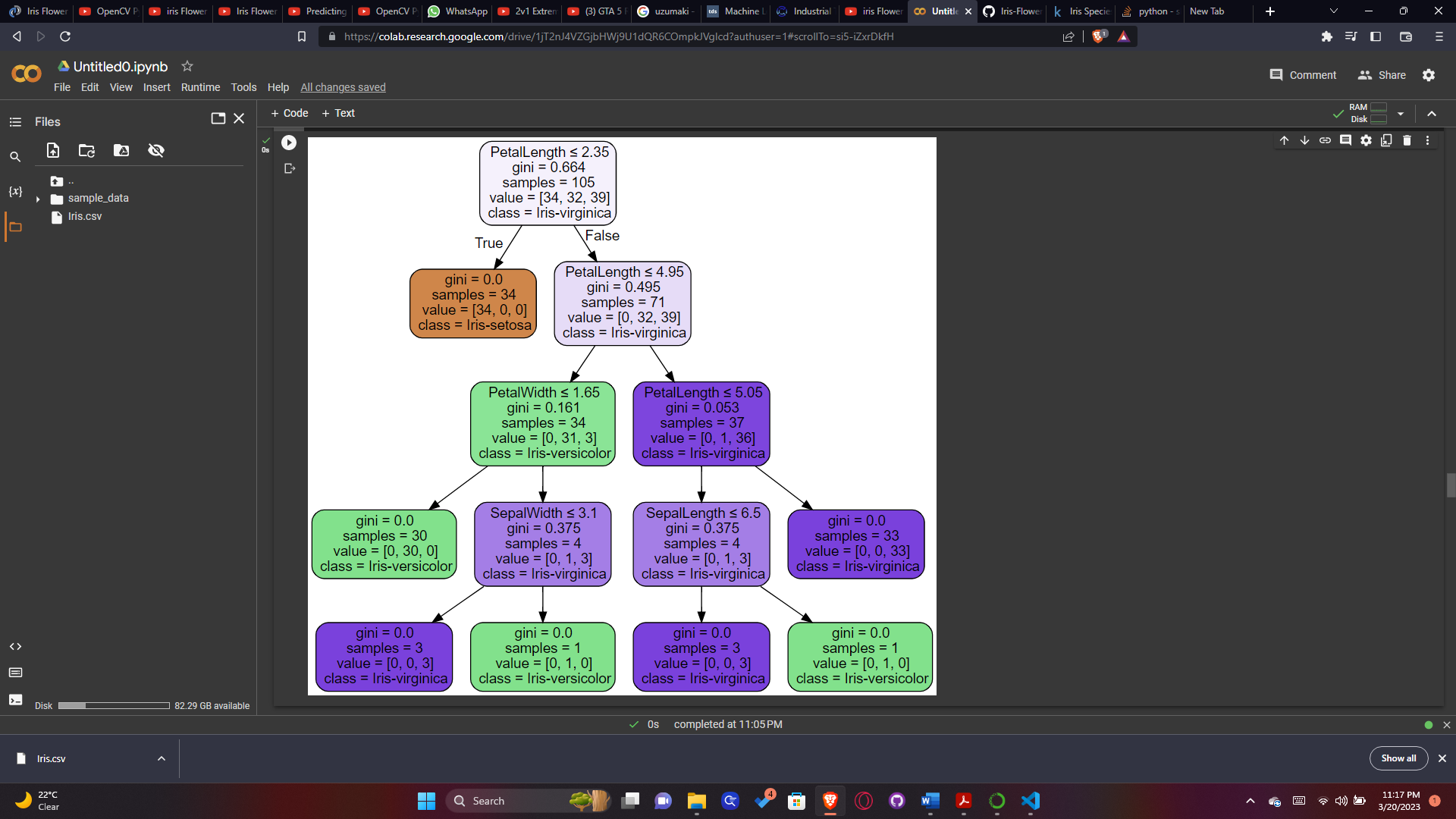
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1, 0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 2, 1, 1, 2, 0, 2, 0, 0])

os.environ["PATH"]+= os.pathsep+(r'C:\Python37\Graph\_Viz\bin')

import graphviz

vis\_data=tree.export\_graphviz(DT,out\_file=None, feature\_names=iris.drop(['Species'],axis=1).keys(),class\_names=iris['Species'].unique(),filled=True,rounded=True,special\_characters=True)

graphviz.Source(vis\_data)



Catagory=['Iris-Setosa','Iris-Versicolor','Iris-Virginica']

#Lets predict on custom input value

X\_DT=np.array([[1 ,1, 1, 1]])

X\_DT\_prediction=DT.predict(X\_DT)

X\_DT\_prediction[0]

print(Catagory[int(X\_DT\_prediction[0])])

Iris-Setosa

# KNN Algorithm

# Preprocessing for Knn

from sklearn.preprocessing import StandardScaler

sc = StandardScaler().fit(X\_train) # Load the standard scaler

X\_train\_std=sc.transform(X\_train)

X\_test\_std=sc.transform(X\_test)

from sklearn.neighbors import KNeighborsClassifier

knn=KNeighborsClassifier(n\_neighbors=5)

knn.fit(X\_train\_std,y\_train)

predict\_knn=knn.predict(X\_test\_std)

accuracy\_knn=accuracy\_score(y\_test,predict\_knn)\*100

accuracy\_knn

97.77777777777777

#Lets predict on custom input value

X\_knn=np.array([[7.7 ,3.5, 4.6, 4]])

X\_knn\_std=sc.transform(X\_knn)

X\_knn\_std

array([[2.07024529, 1.03637663, 0.42953569, 3.56608443]])

X\_knn\_prediction=knn.predict(X\_knn\_std)

X\_knn\_prediction[0]

print(Catagory[int(X\_knn\_prediction[0])])

Iris-Virginica

#Finding Best K Value

k\_range=range(1,26)

scores={}

scores\_list=[]

for k in k\_range:

knn=KNeighborsClassifier(n\_neighbors=k)

knn.fit(X\_train\_std,y\_train)

prediction\_knn=knn.predict(X\_test\_std)

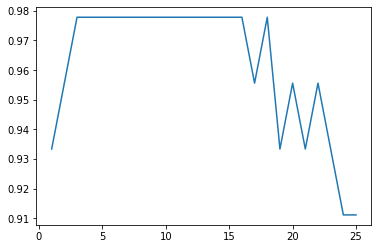
scores[k]=accuracy\_score(y\_test,prediction\_knn)

scores\_list.append(accuracy\_score(y\_test,prediction\_knn))

scores\_list

[0.9333333333333333, 0.9555555555555556, 0.9777777777777777, 0.9777777777777777, 0.9777777777777777, 0.9777777777777777, 0.9777777777777777, 0.9777777777777777, 0.9777777777777777, 0.9777777777777777, 0.9777777777777777, 0.9777777777777777, 0.9777777777777777, 0.9777777777777777, 0.9777777777777777, 0.9777777777777777, 0.9555555555555556, 0.9777777777777777, 0.9333333333333333, 0.9555555555555556, 0.9333333333333333, 0.9555555555555556, 0.9333333333333333, 0.9111111111111111, 0.9111111111111111]

plt.plot(k\_range,scores\_list)



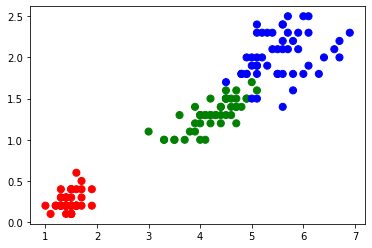
#K MEANS Clustering

y

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])

colormap=np.array(['Red','green','blue'])

fig=plt.scatter(iris['PetalLength'],iris['PetalWidth'],c=colormap[y],s=50)



from sklearn.cluster import KMeans

km=KMeans(n\_clusters=3,n\_init=4,random\_state=2)

km.fit(X)

centers=km.cluster\_centers\_

print(centers)

[[5.006 3.418 1.464 0.244 ]

[5.9016129 2.7483871 4.39354839 1.43387097]

[6.85 3.07368421 5.74210526 2.07105263]]

km.labels\_

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1], dtype=int32)

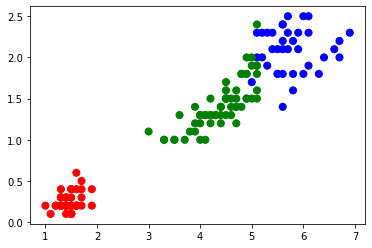
Catagory\_kmeans=['Iris-Versicolor', 'Iris-Setosa', 'Iris-Virginica']

Catagory\_kmeans

['Iris-Versicolor', 'Iris-Setosa', 'Iris-Virginica']

colormap=np.array(['Red','green','blue'])

fig=plt.scatter(iris['PetalLength'],iris['PetalWidth'],c=colormap[km.labels\_],s=50)



new\_labels=km.labels\_

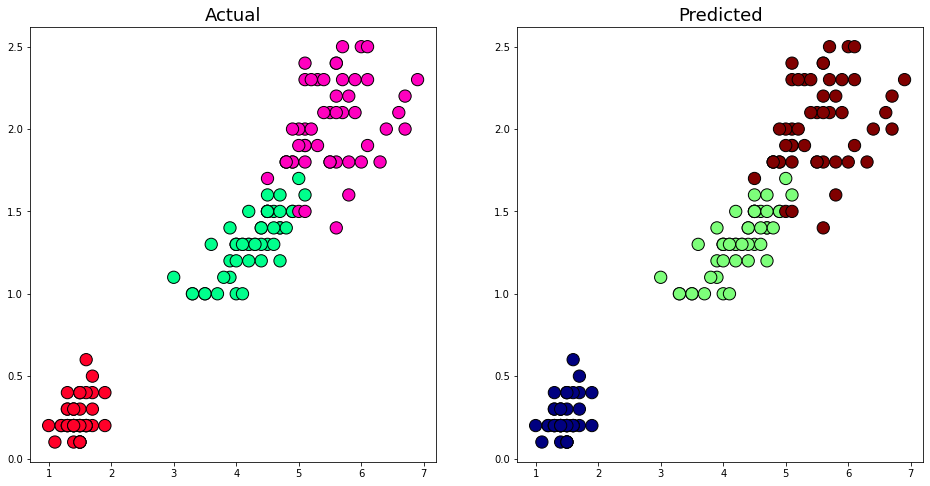
fig,axes=plt.subplots(1,2,figsize=(16,8))

axes[0].scatter(X[:,2],X[:,3],c=y,cmap='gist\_rainbow',edgecolor='k',s=150)

axes[1].scatter(X[:,2],X[:,3],c=y,cmap='jet',edgecolor='k',s=150)

axes[0].set\_title('Actual',fontsize=18)

axes[1].set\_title('Predicted',fontsize=18)



#Lets predict on custom input value

X\_km=np.array([[1 ,1, 1, 1]])

X\_km\_prediction=km.predict(X\_km)

X\_km\_prediction[0]

print(Catagory\_kmeans[int(X\_km\_prediction[0])])

Iris-Versicolor

**PART 3:**

**Haar cascade algorithm for face detection**

haarcascade\_frontalface\_default.xml file taken from https://github.com/opencv/opencv/tree/master/data/haarcascades

Code:

import cv2

face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

img = cv2.imread('test 2.jpg')

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

faces = face\_cascade.detectMultiScale(gray, 1.1, 4)

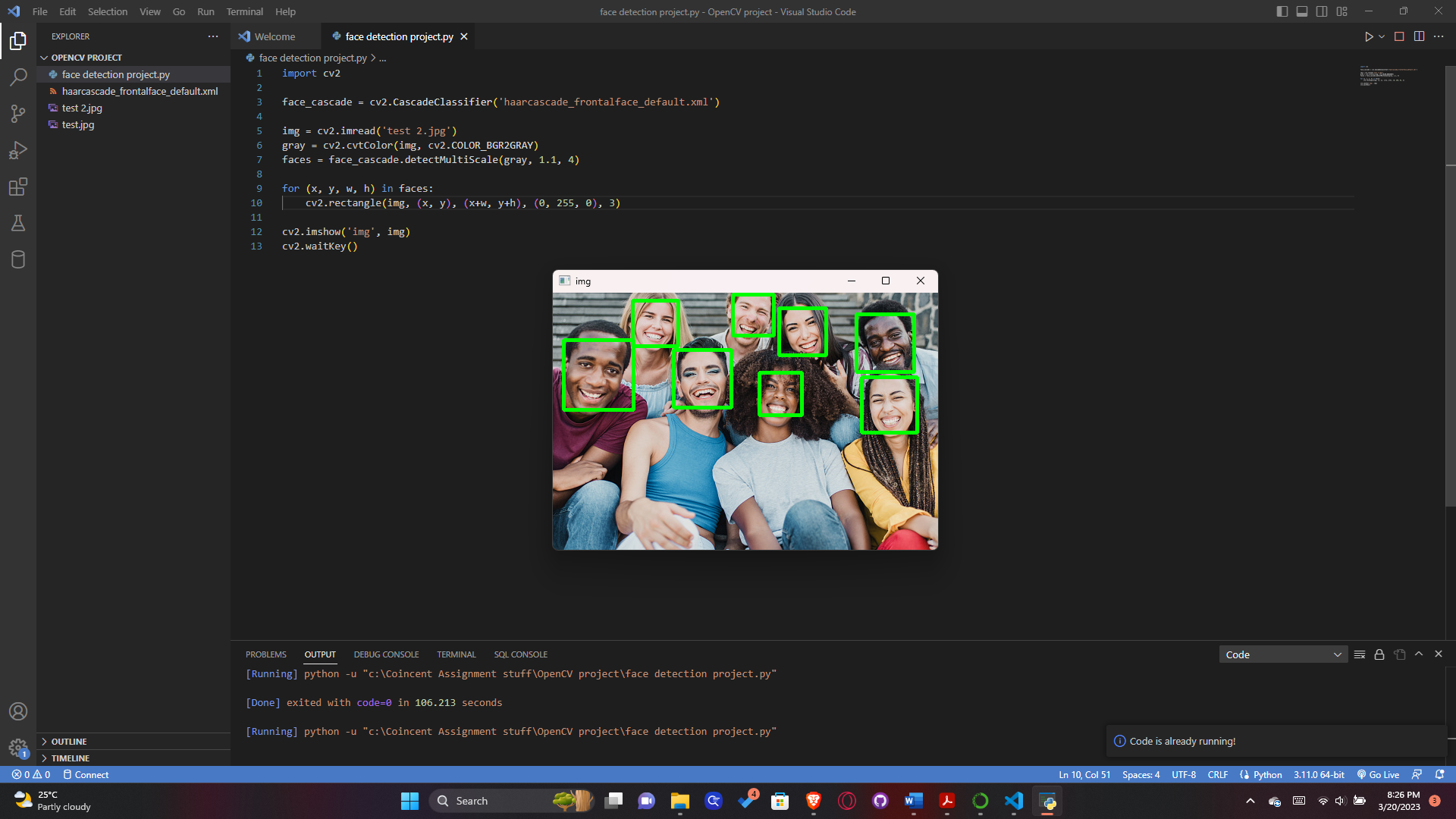
for (x, y, w, h) in faces:

    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 3)

cv2.imshow('img', img)

cv2.waitKey()

Output:



**PART 4:**

**Boston Housing Classifier (using Google Collab)**

Code:

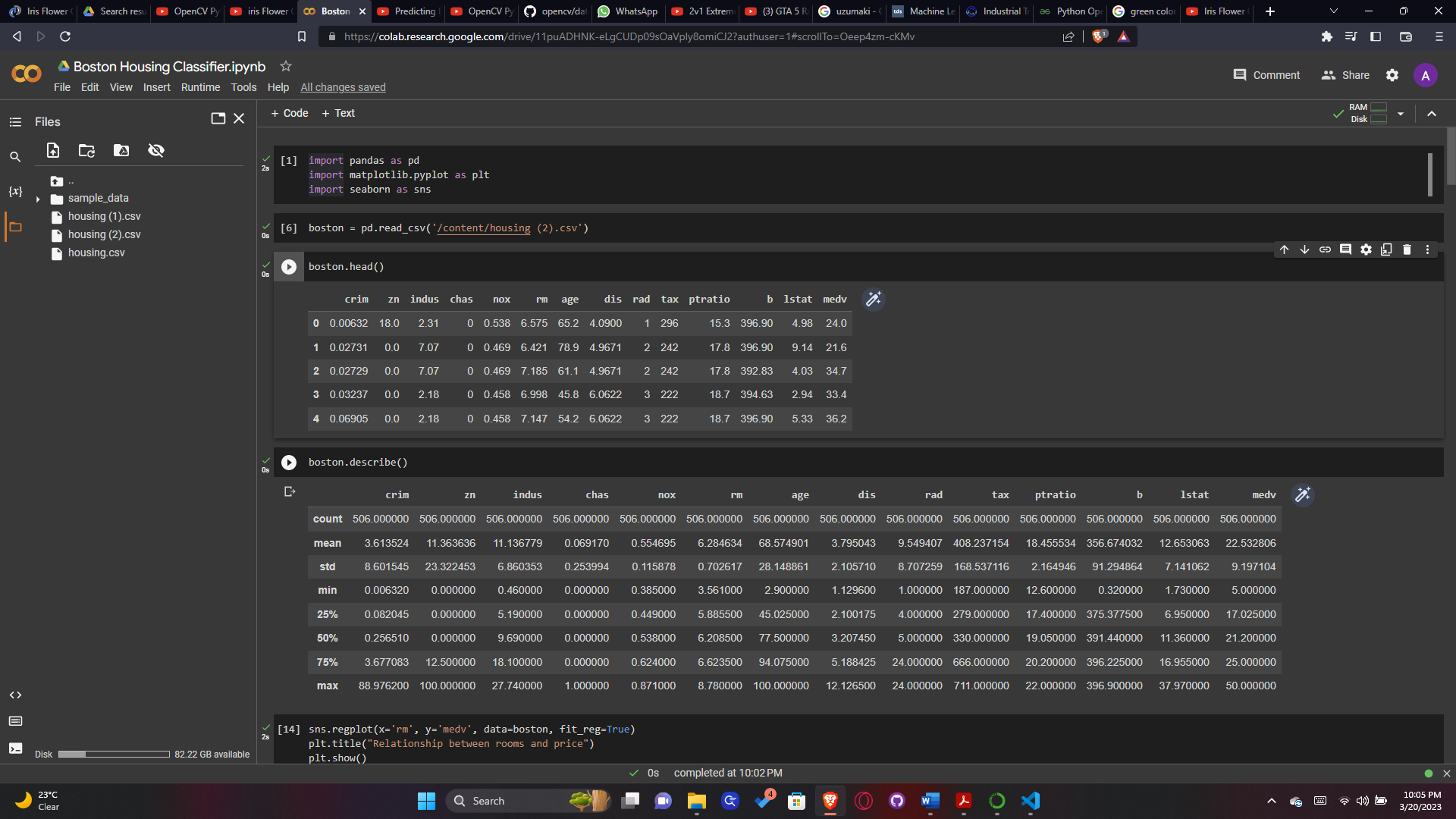
import pandas as pd

import matplotlib.pyplot as plt

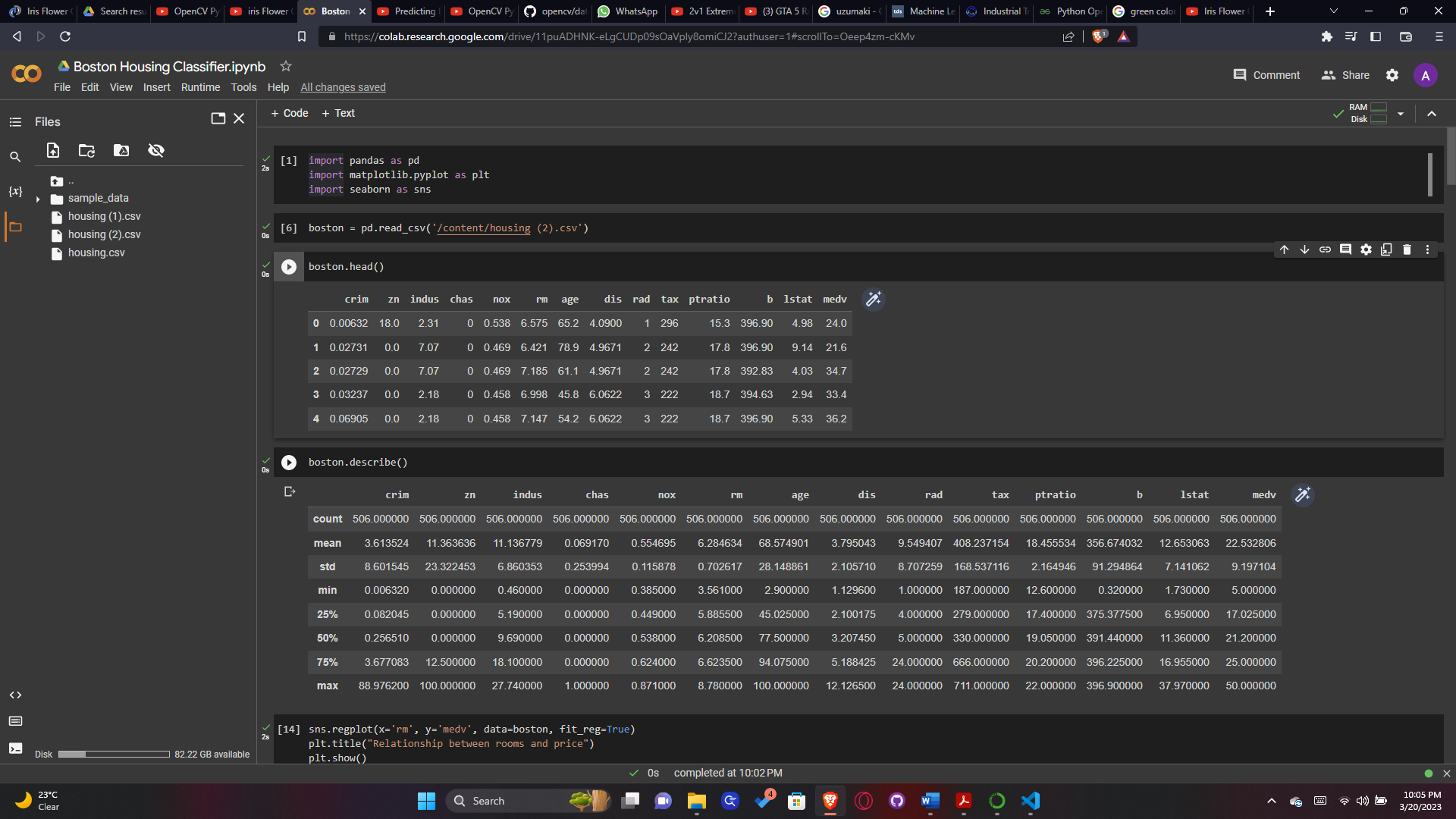
import seaborn as sns

boston = pd.read\_csv('/content/housing (2).csv')

boston.head()



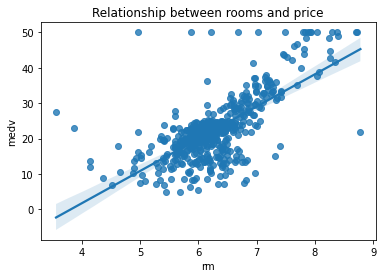
boston.describe()



sns.regplot(x='rm', y='medv', data=boston, fit\_reg=True)

plt.title("Relationship between rooms and price")

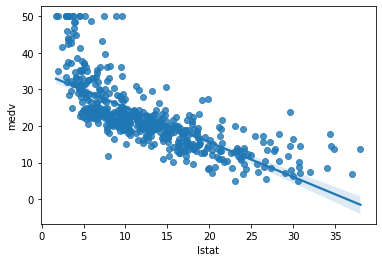
plt.show()



# showing a few more plots

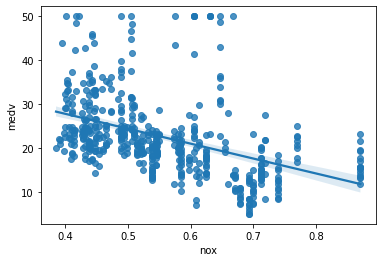
sns.regplot(y='medv', x='lstat', data=boston, fit\_reg=True)

plt.show()



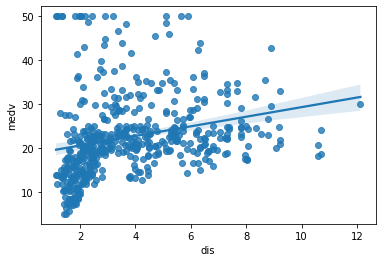
sns.regplot(x='nox', y='medv', data=boston, fit\_reg=True)

plt.show()



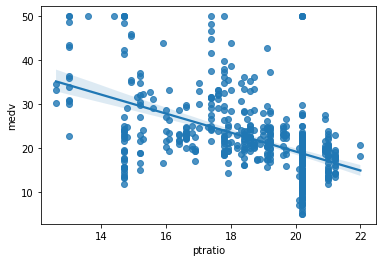
sns.regplot(y='medv', x='dis', data=boston, fit\_reg=True)

plt.show()



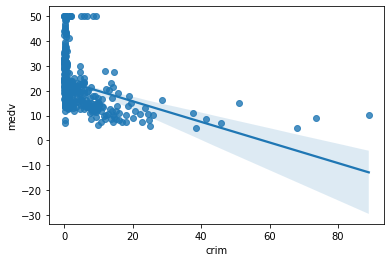
sns.regplot(x='ptratio', y='medv', data=boston, fit\_reg=True)

plt.show()

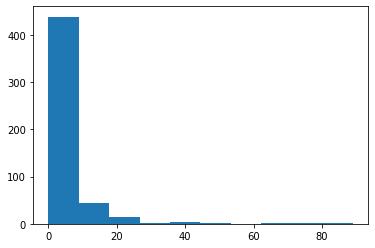


sns.regplot(x='crim', y='medv', data=boston, fit\_reg=True)

plt.show()

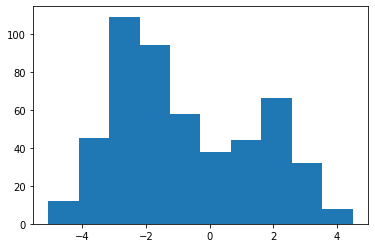


plt.hist(boston.crim)



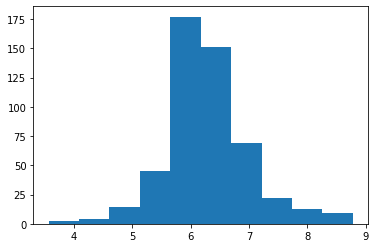
import numpy as np

plt.hist(np.log(boston.crim))



# showing a few more histograms

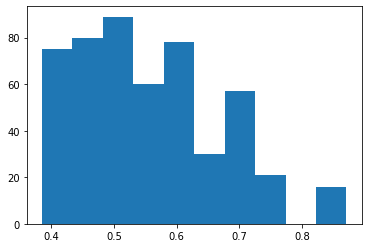
plt.hist(boston.rm)



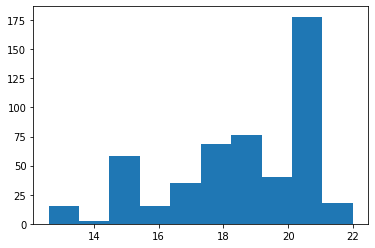
plt.hist(boston.lstat)



plt.hist(boston.nox)



plt.hist(boston.ptratio)

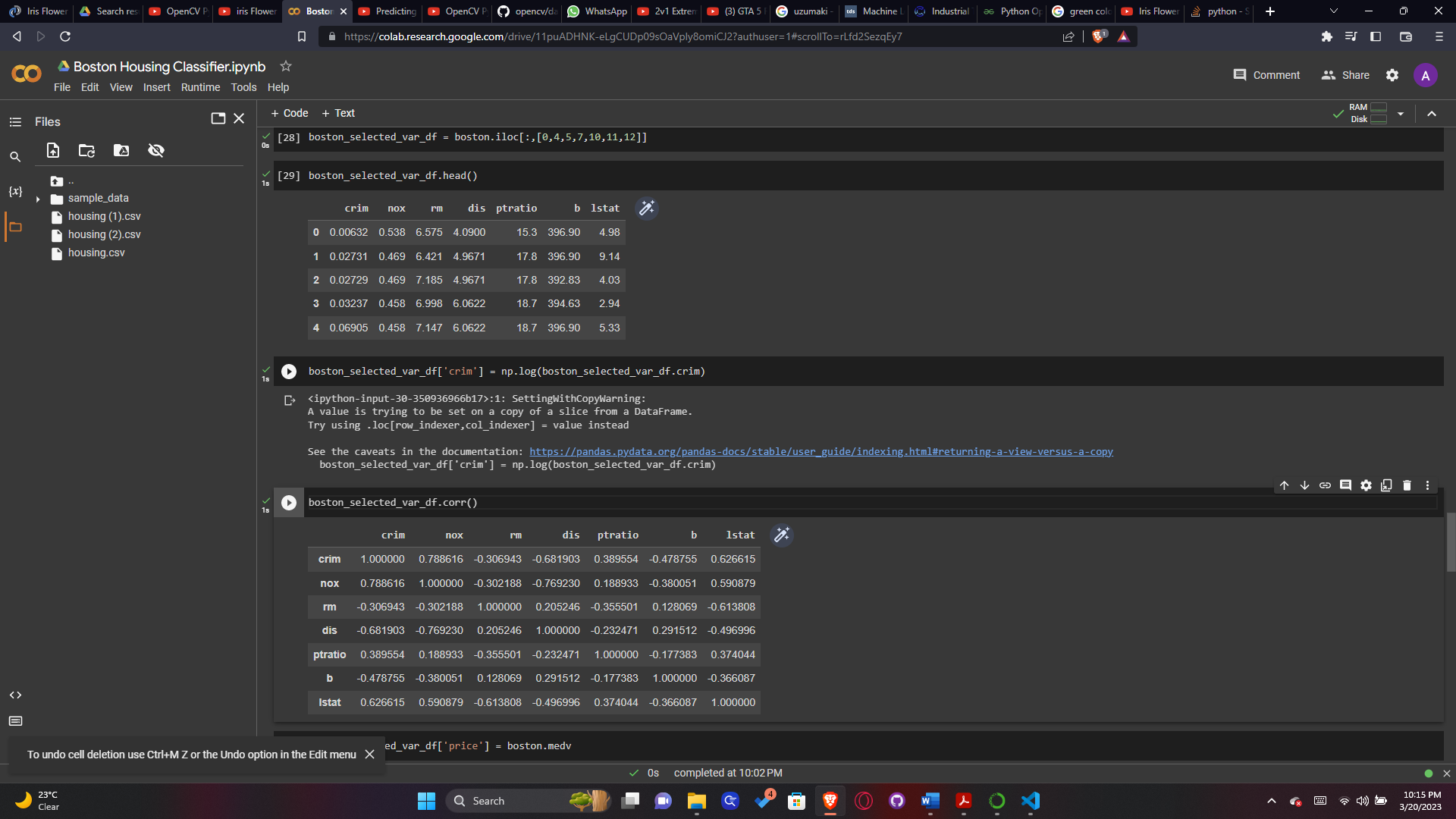


boston.columns

Index(['crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax', 'ptratio', 'b', 'lstat', 'medv'], dtype='object')

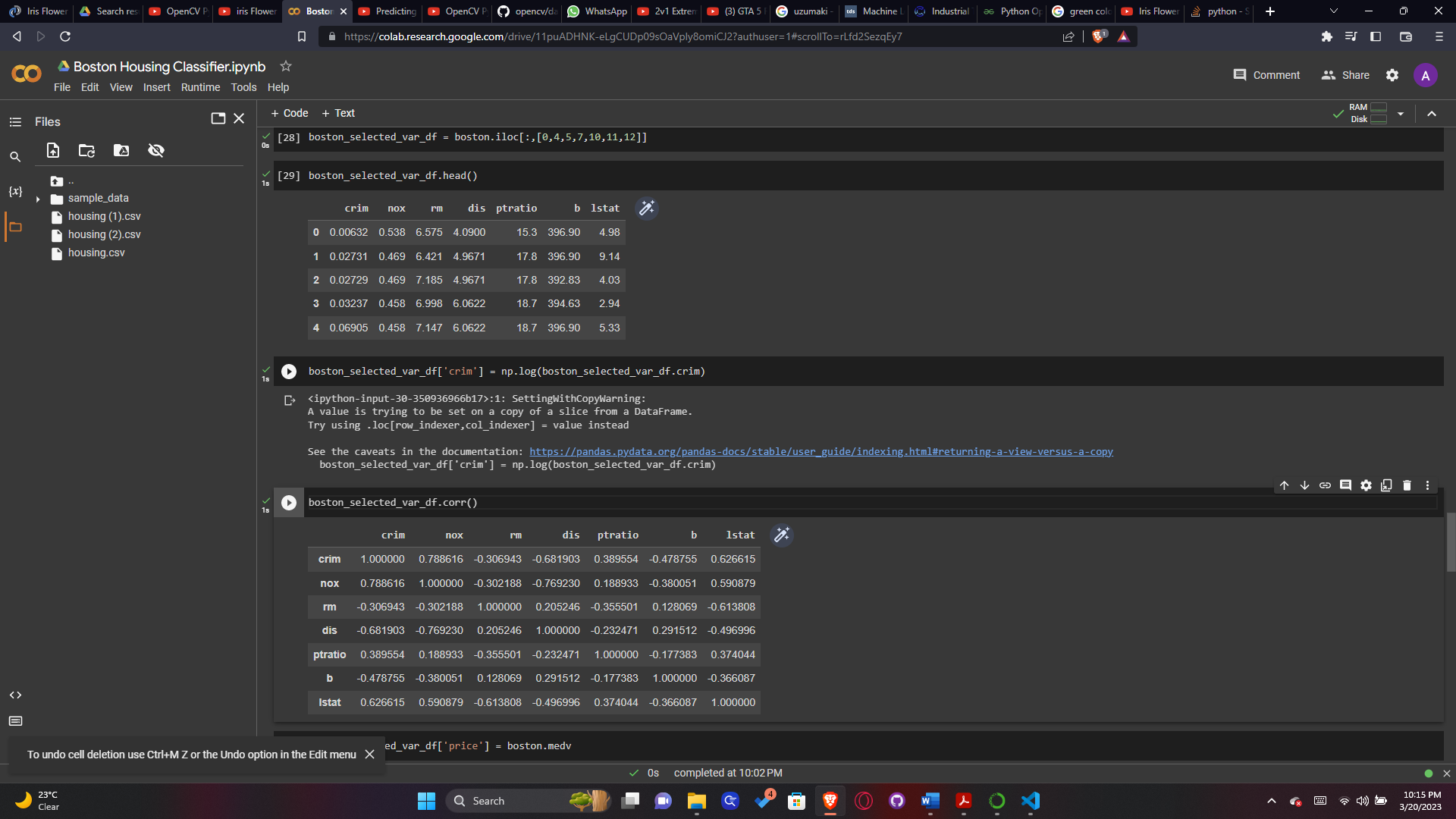
boston\_selected\_var\_df = boston.iloc[:,[0,4,5,7,10,11,12]]

boston\_selected\_var\_df.head()



boston\_selected\_var\_df['crim'] = np.log(boston\_selected\_var\_df.crim)

boston\_selected\_var\_df.corr()



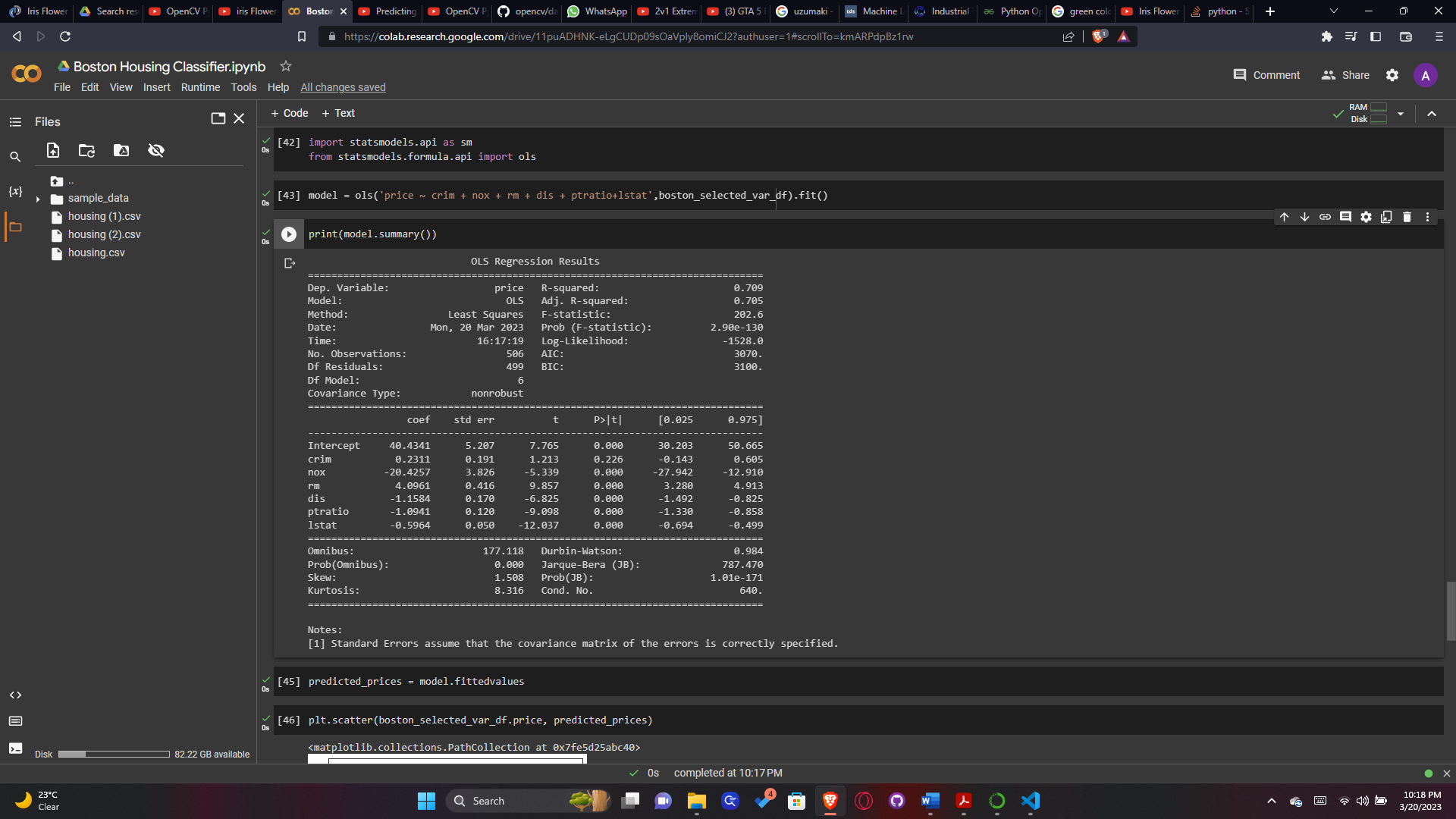
boston\_selected\_var\_df['price'] = boston.medv

import statsmodels.api as sm

from statsmodels.formula.api import ols

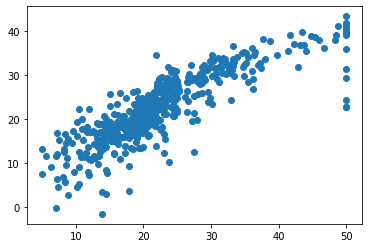
model = ols('price ~ crim + nox + rm + dis + ptratio+lstat',boston\_selected\_var\_df).fit()

print(model.summary())



predicted\_prices = model.fittedvalues

plt.scatter(boston\_selected\_var\_df.price, predicted\_prices)



from sklearn.metrics import mean\_squared\_error

error = np.sqrt(mean\_squared\_error(boston\_selected\_var\_df.price, predicted\_prices))

error

4.9568674667600074

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

x = boston\_selected\_var\_df.drop('price', axis=1)

y = boston\_selected\_var\_df['price']

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(x, y, test\_size=.3, random\_state=3)

from sklearn.linear\_model import LinearRegression

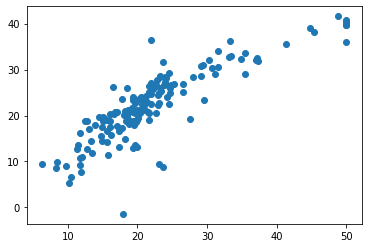
LinReg = LinearRegression()

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

Y\_pred = LinReg.predict(X\_test)

plt.scatter(Y\_test, Y\_pred)

<matplotlib.collections.PathCollection at 0x7fe5d17dcfd0>



np.sqrt(mean\_squared\_error(Y\_test, Y\_pred))

4.650074744839381