Prac\_1- lIST

**INPUT:**

list = [1,2,3,4,5,6,7]

print(list)

list.append(10)

print(list)

list.insert(2,21)

print(list)

number = [30,89,70]

number.extend(list)

print(number)

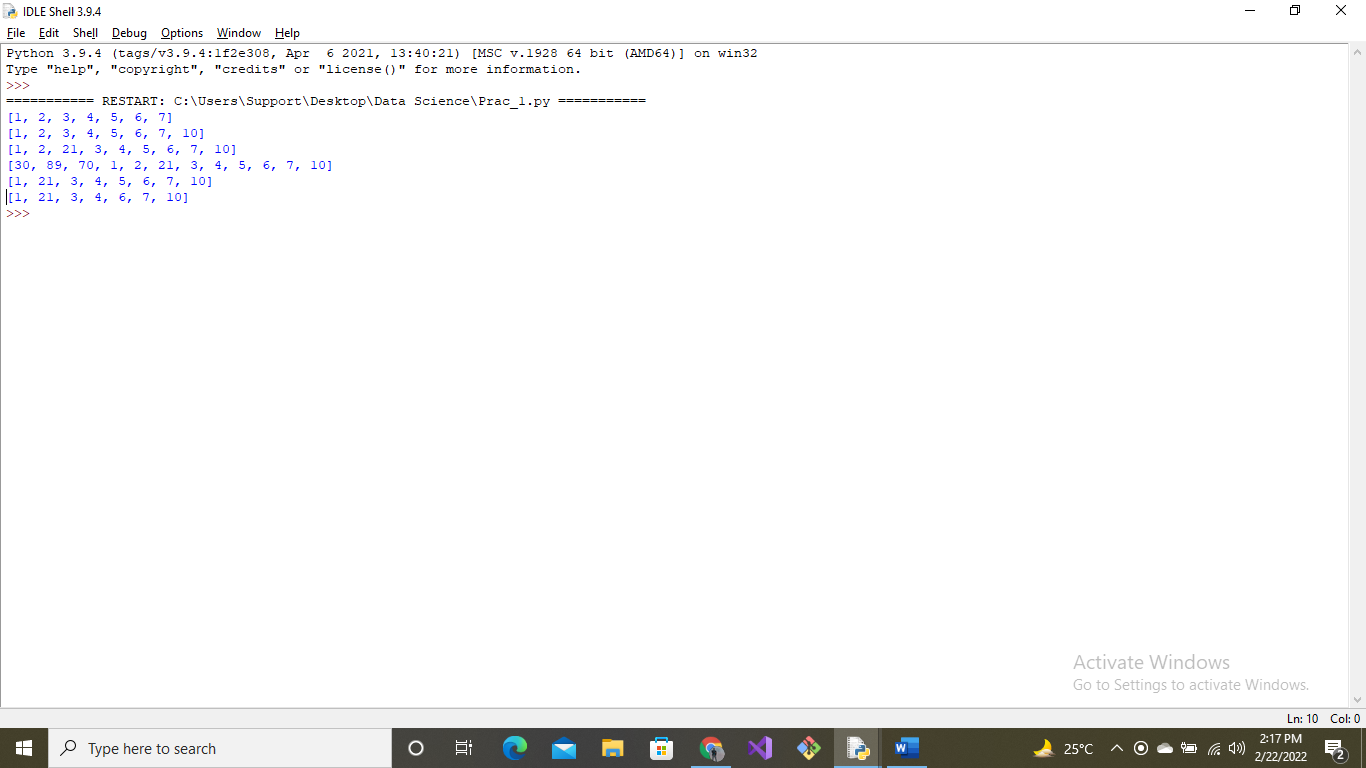
list.remove(2)

print(list)

list.pop(4)

print(list)

**OUTPUT**:



Prac\_2 ATTENDANCE

**INPUT**:

att = ()

start = True

while(start):

name = input("Enter the student's name:")

lecture\_attended = input("Enter the lecture\_attended:")

roll\_no = input("Enter your roll\_no:")

stud = ("name:"+name, "roll\_no:"+roll\_no, "lecture\_attended:"+lecture\_attended)

att += stud

alternative = input("Enter 1 to add more data and 2 to retrieve data:")

if (alternative == 1):

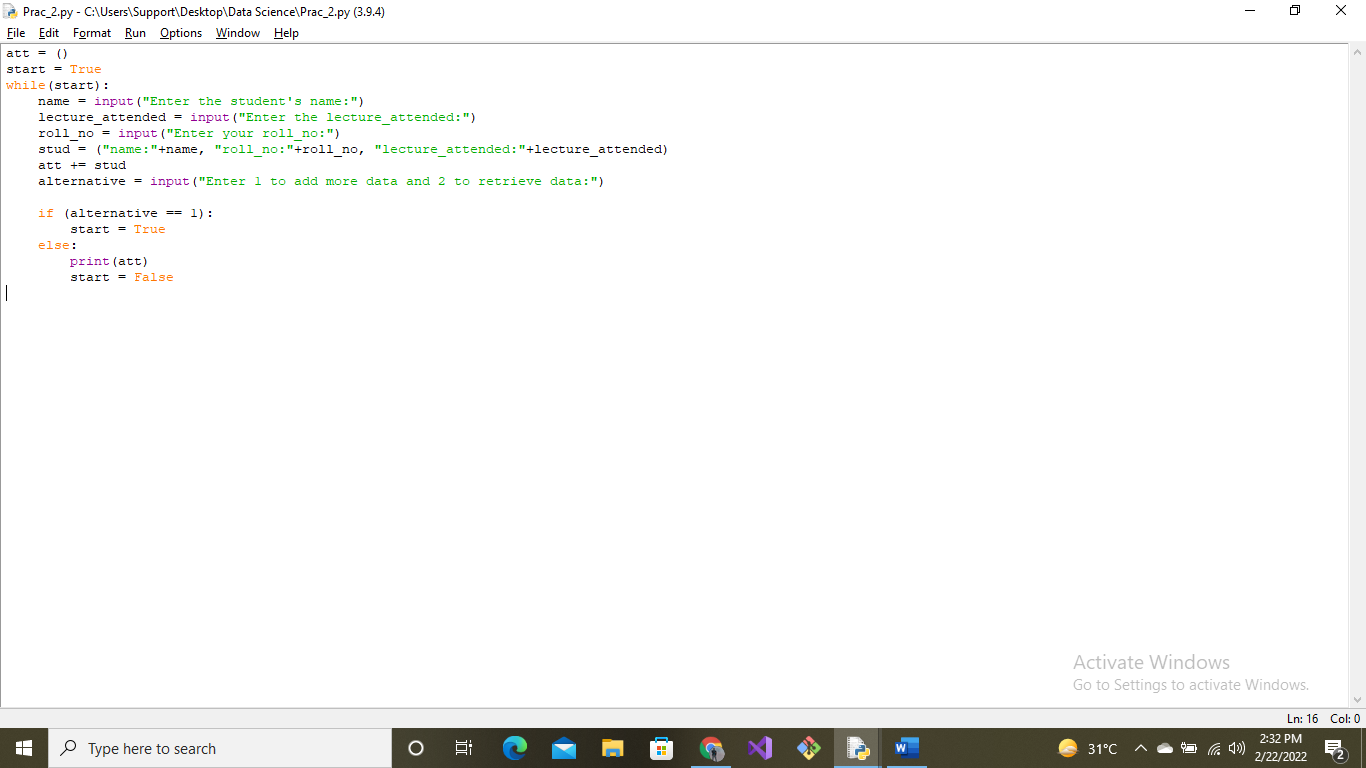
start = True

else:

print(att)

start = False

**OUTPUT**:



Prac\_3-SET

**INPUT**:

employee1 = set(["Akash", "Sana", "Ram", "Ayush"])

employee2 = set(["Vineet", "Krishna", "Anjali"])

print("Employee records:")

print(employee1)

print(employee2)

employee1.discard("Ayush")

print("\nIntersection of two said sets:")

Intersection = employee1 & employee2

print(Intersection)

print("\nUnion of two said sets:")

Union = employee1 | employee2

print(Union)

print("Difference between two said sets:")

difference = employee1 - employee2

print(difference)

print("Compare two said sets:")

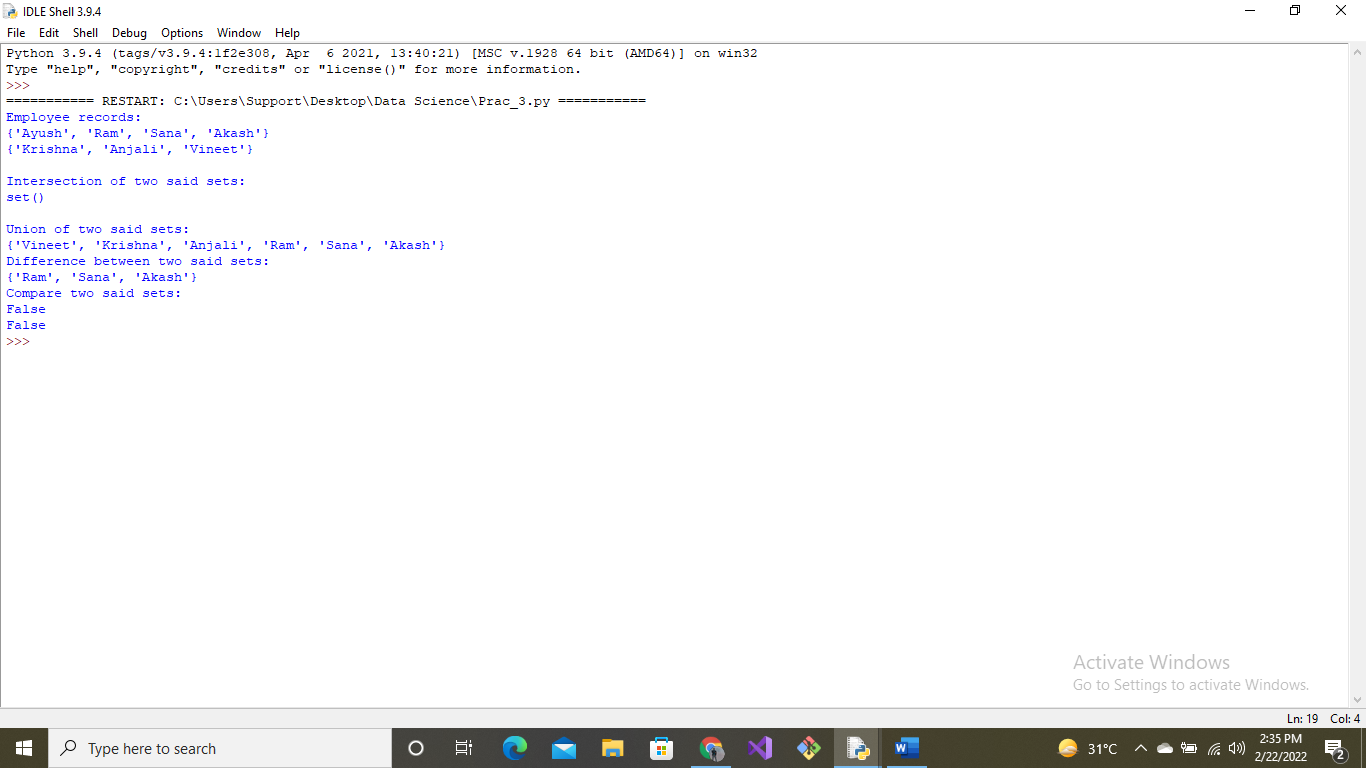
compare = employee1 <= employee2

compare2 = employee1 >= employee2

print(compare)

print(compare2)

**OUTPUT**:



Prac\_4-Employee

**INPUT**:

salary = {"Akash":9000000, "Sana":2000000, "Ram":200000, "Max":50000}

print("\nEMPLOYEE DETAILS:", salary)

#for getting highest salary

print("Employee having highest salary is:", max(salary, key=salary.get))

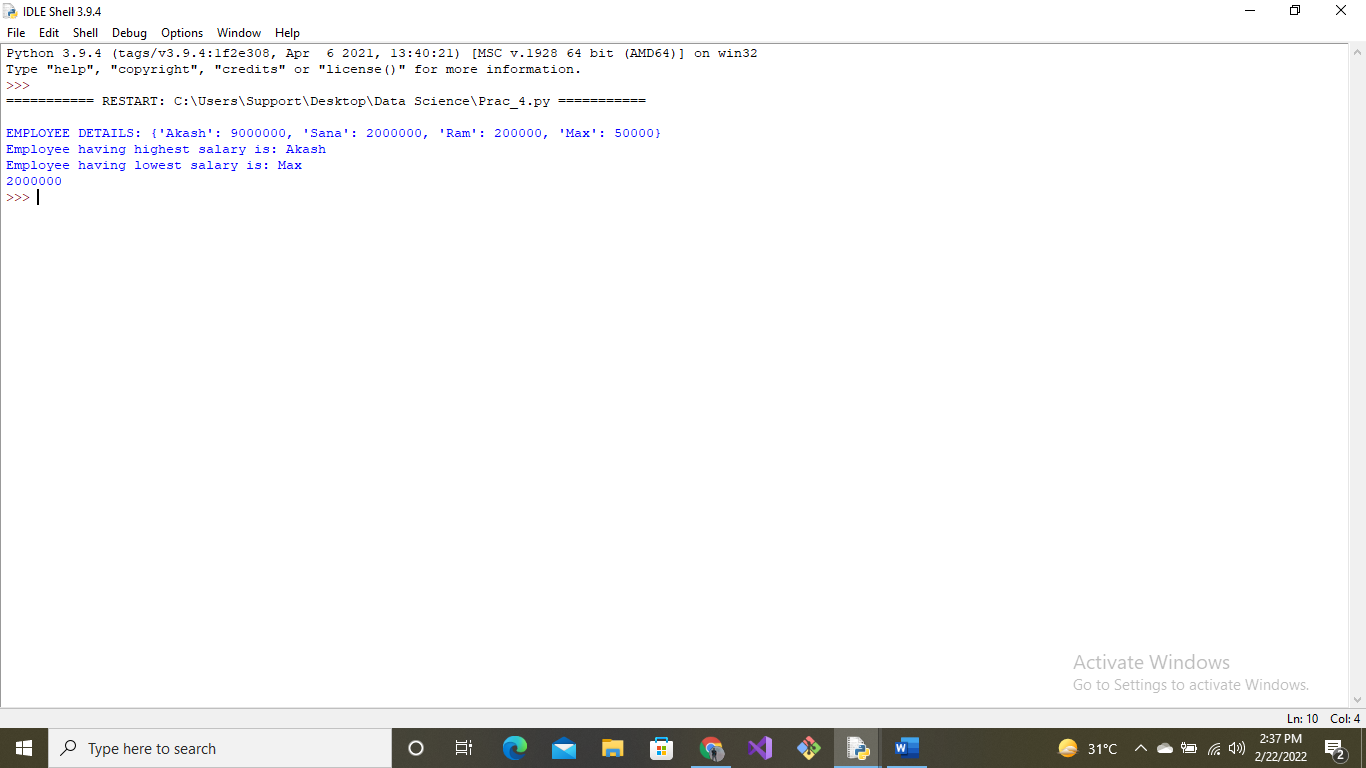
#for getting lowest salary

print("Employee having lowest salary is:", min(salary, key=salary.get))

#for getting particular employee's salary

print(salary.get("Sana"))

**OUTPUT**:



Prac\_5-String

**INPUT**:

s0 = "orange"

s1 = "Hello this is python language"

s2 = "lets learn about strings in python"

#CONCATENATION

print("Concatenation:")

print(s1+','+s2)

#ITEARTION

print("Iteration:")

for i in s0:

print(i)

#UPPER()

print("UPPERCASE:")

print(s0.upper())

#JOIN()

print("JOIN:")

str1 = '->'

str2 = '1234'

print(str1.join(str2))

#SPLIT()

print("SPLIT:")

print(s1.split())

#FIND()

print("FIND:")

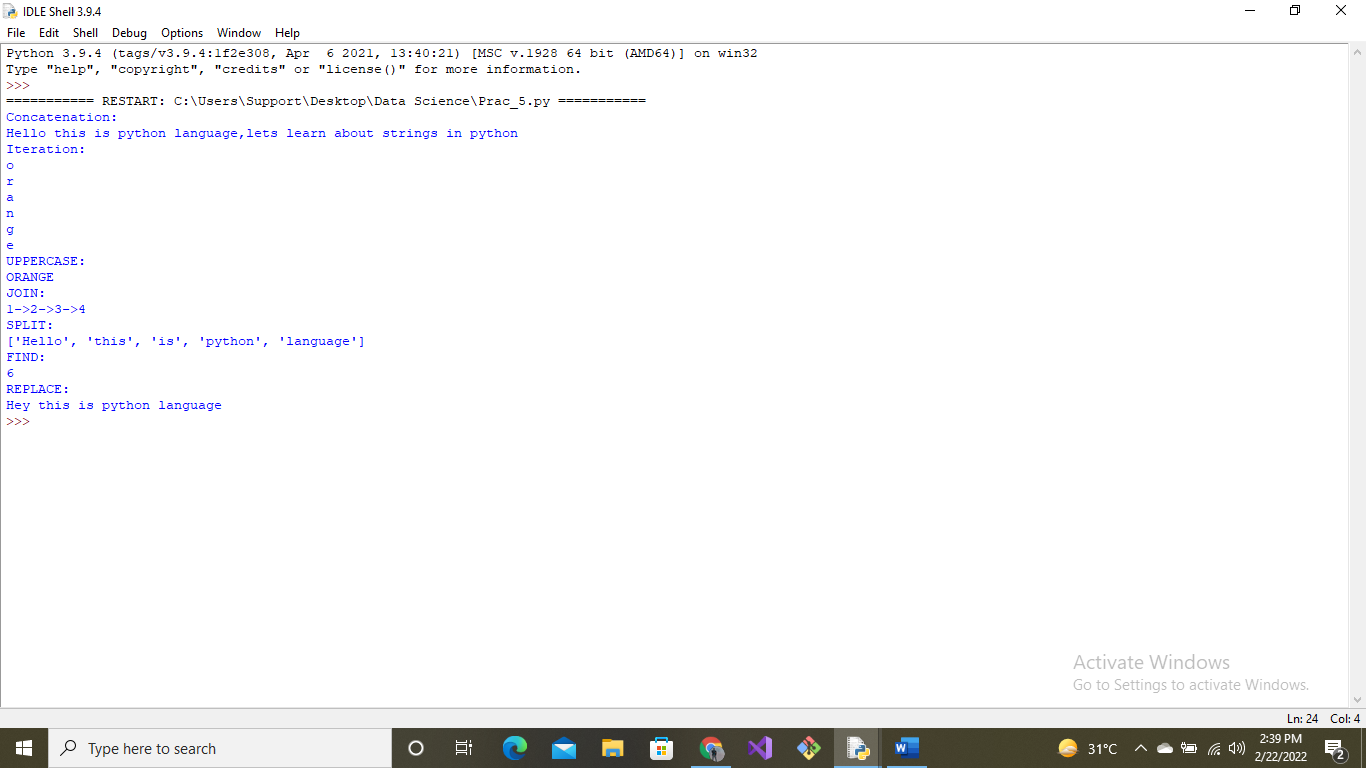
print(s1.find("this"))

#REPLACE

print("REPLACE:")

print(s1.replace("Hello", "Hey"))

**OUTPUT**:



Prac\_6-Matplotlib

**INPUT**:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

df = pd.read\_csv('pokemon.csv') #usage of pandas

df.head()

df.drop('#', inplace=True, axis=1)

df.head()

df.shape #usage of numpy

# frame[‘DataFrame Column’]= frame[‘DataFrame Column’].apply(str)

df['Type 2'] = df['Type 2'].apply(str)

df.info()

newdf = pd.DataFrame(df, columns=['Name','HP'])

newdf.head()

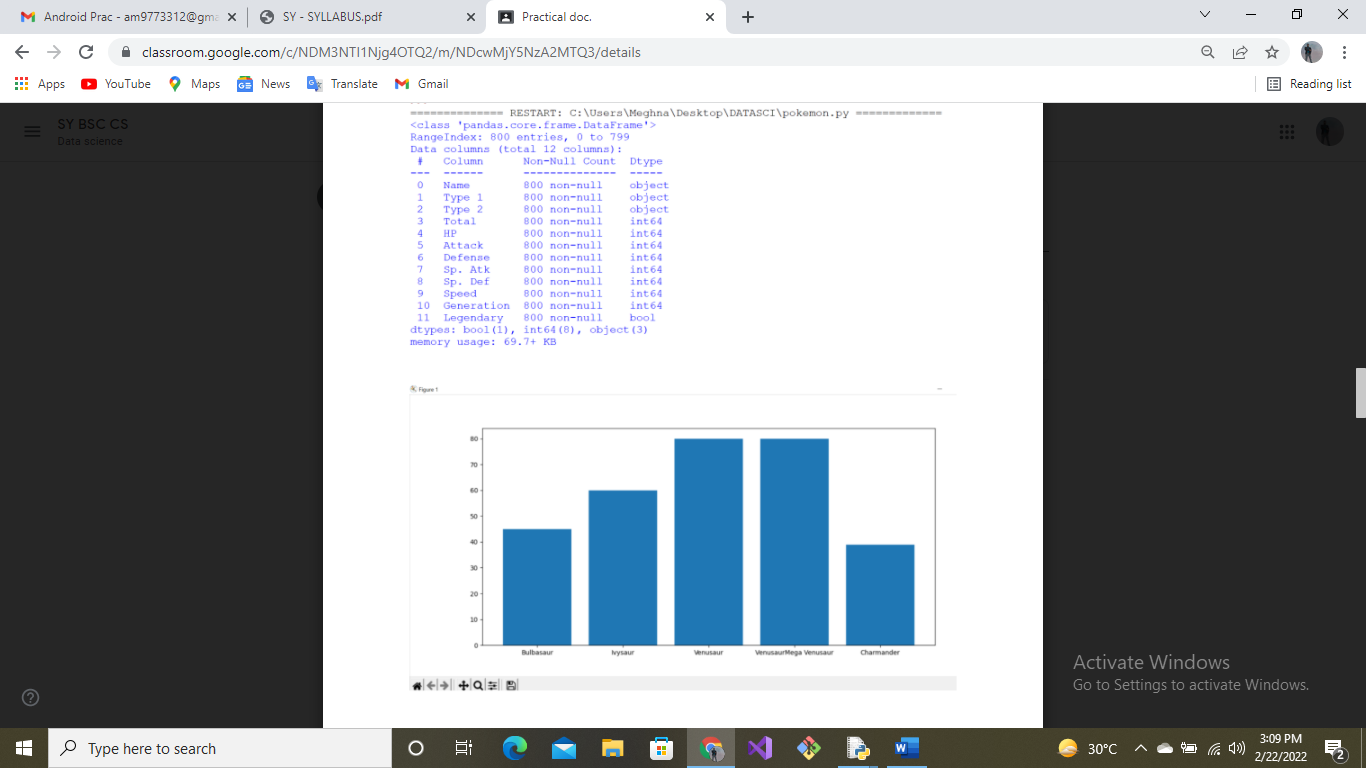
newdf.shape

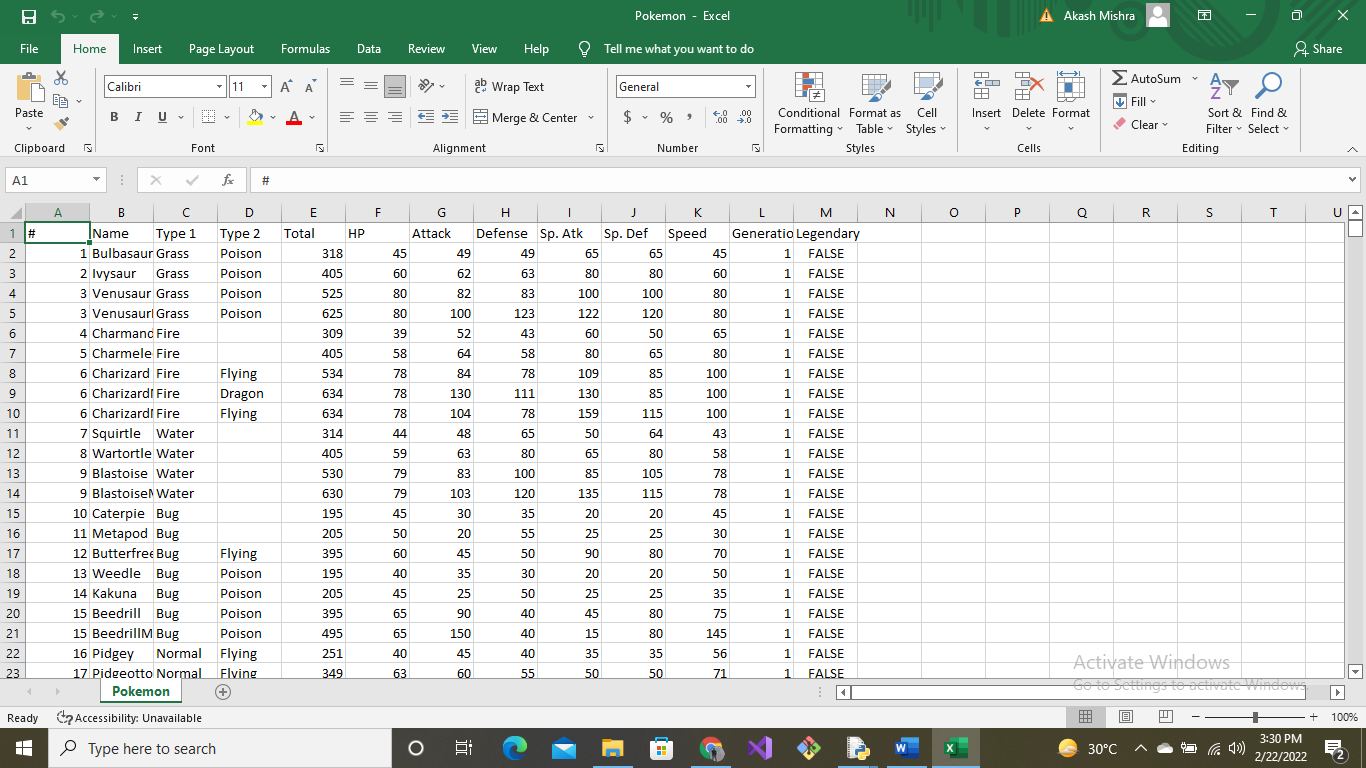
plt.rcParams["figure.figsize"] = (20, 9)

plt.bar(newdf['Name'].head(), newdf['HP'].head())

plt.show() **:**

**OUTPUT**:





Prac\_7-Numpy

**INPUT**:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

df = pd.read\_csv('pokemon.csv') #usage of pandas

df.head()

df.drop('#', inplace=True, axis=1)

df.head()

df.shape #usage of numpy

# frame[‘DataFrame Column’]= frame[‘DataFrame Column’].apply(str)

df['Type 2'] = df['Type 2'].apply(str)

df.info()

newdf = pd.DataFrame(df, columns=['Name','HP'])

newdf.head()

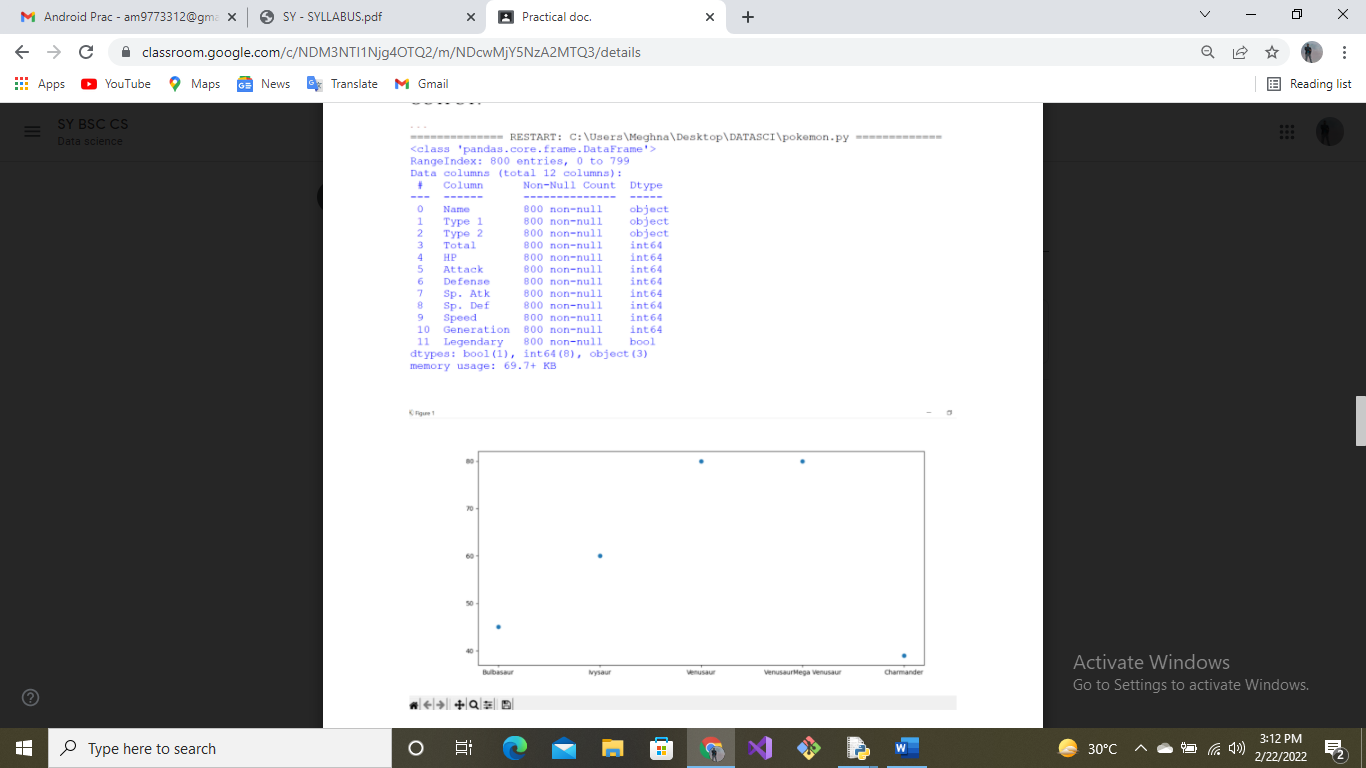
newdf.shape

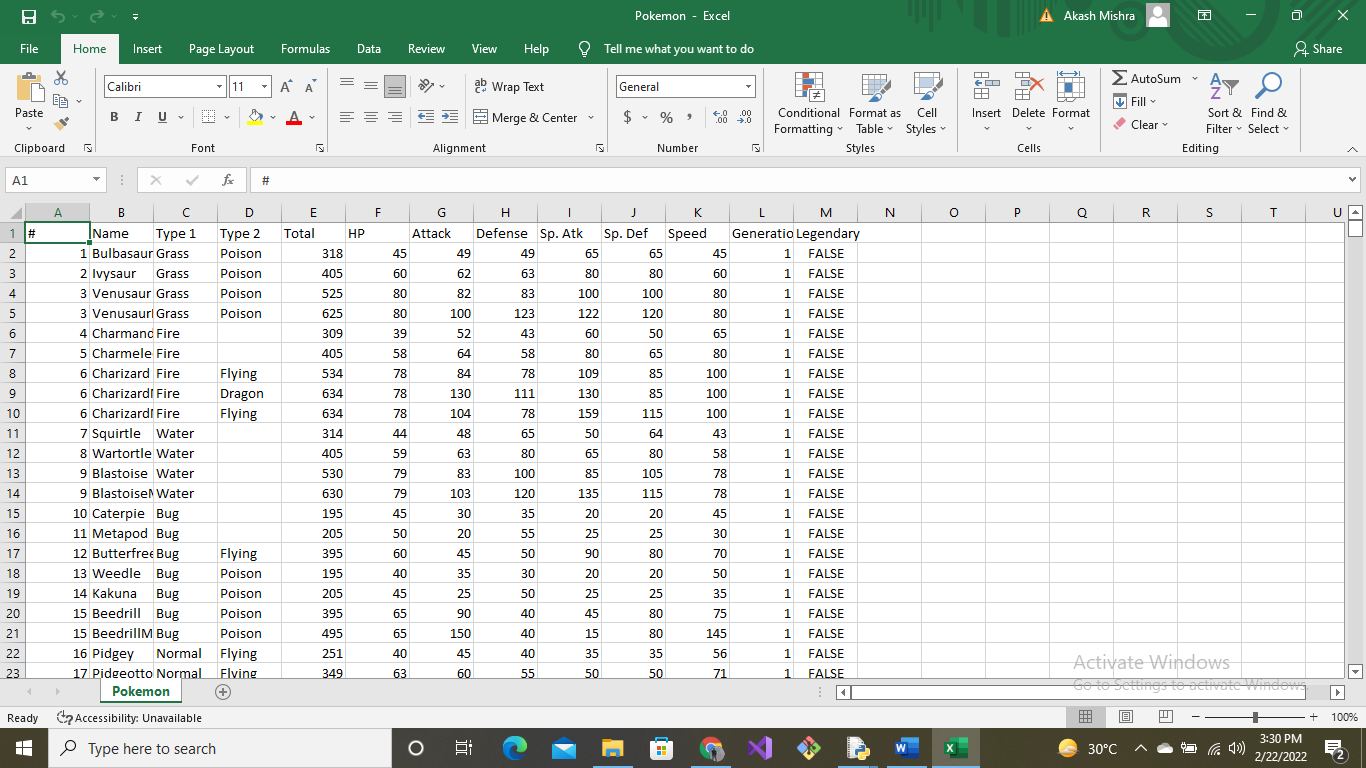
plt.rcParams["figure.figsize"] = (20, 9)

plt.scatter(newdf['Name'].head(), newdf['HP'].head())

plt.show()

**OUTPUT**:





Prac\_8-Panda

**INPUT**:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

df = pd.read\_csv('pokemon.csv') #usage of pandas

df.head()

df.drop('#', inplace=True, axis=1)

df.head()

df.shape #usage of numpy

# frame[‘DataFrame Column’]= frame[‘DataFrame Column’].apply(str)

df['Type 2'] = df['Type 2'].apply(str)

df.info()

newdf = pd.DataFrame(df, columns=['Name','HP'])

newdf.head()

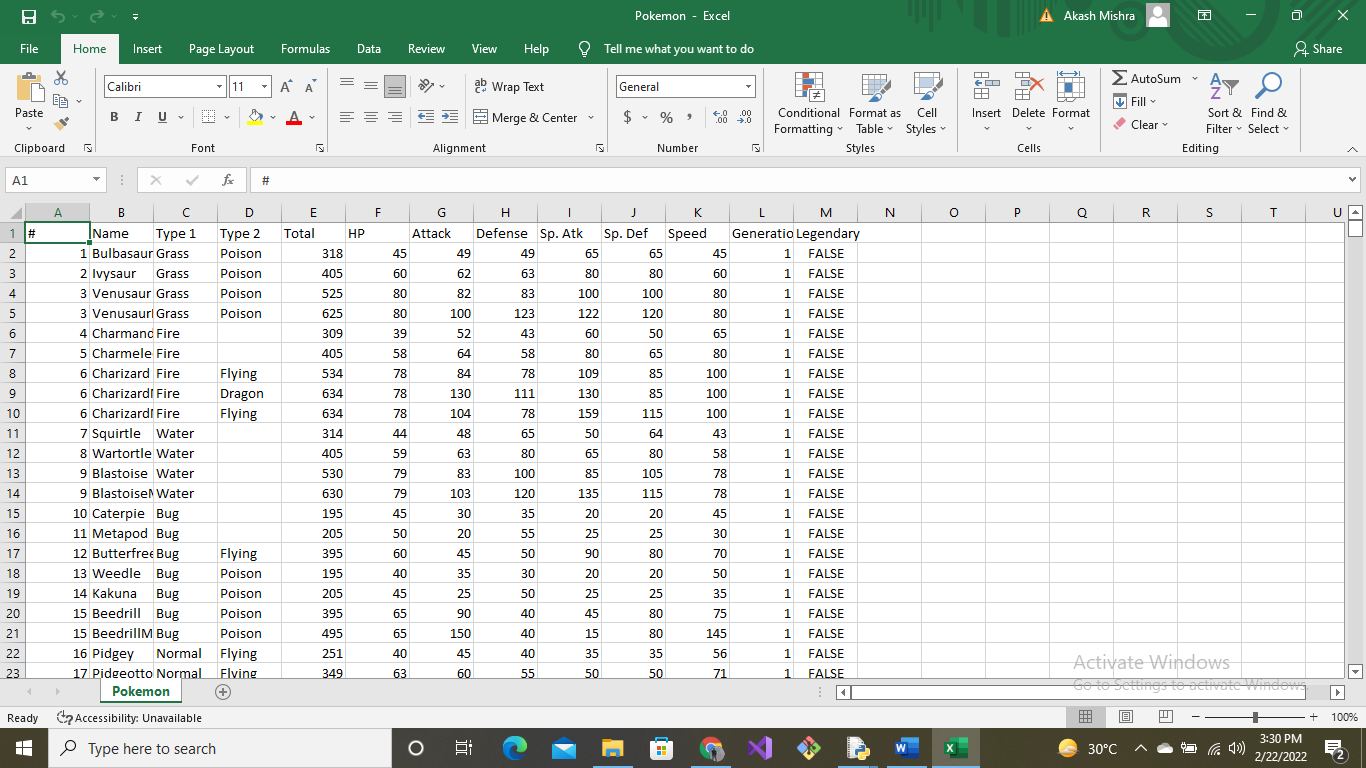
newdf.shape

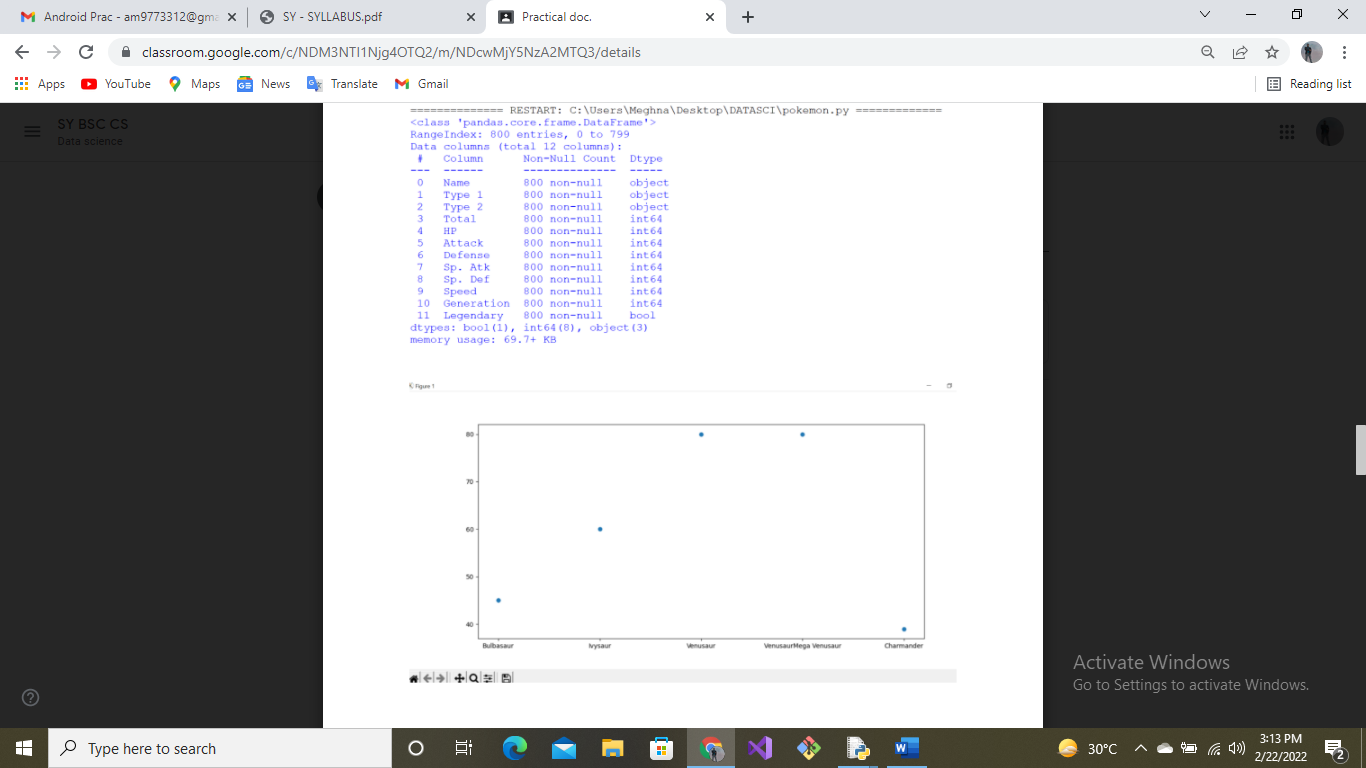
plt.rcParams["figure.figsize"] = (20, 9)

plt.scatter(newdf['Name'].head(), newdf['HP'].head())

plt.show()

**OUTPUT**:





Prac\_9-PCA

**INPUT**:

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

url = "https://archive.ics.uci.edu/ml/datasets/iris"

df = pd.read\_csv(url, names=["sepal length", "sepal width", "petal length", "petal width", "target"])

std\_dev\_sepal\_length = df["sepal length"].std()

std\_dev\_sepal\_length

(5.1-mean\_sepal\_length)/std\_dev\_sepal\_length

feature = ["sepal length", "sepal width", "petal length", "petal width"]

x = df.loc[:,feature]

x = df.loc[:,"target"]

x = StandardScaler().fit\_transform(x)

standardised\_values = pd.DataFrame(x, columns = feature)

standardised\_values

from sklearn.decomposition import PCA

pca = PCA(n\_components = 2)

pct = pca.fit\_transform(x)

principal\_df = pd.DataFrame(pct, columns=["pc1", "pc2"])

finaldf = pd.concat([principal\_df, df[["target"]]],axis = 1)

finaldf.head()

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

ax = fig.add\_subplot(1,1,1)

ax.set\_xlabel("Principal Component 1", fontsize = 15)

ax.set\_ylable("Principal Component 2", fontsize = 15)

ax.set\_title("2 Component PCA", fontsize = 20)

targets =["Iris-setsoa", "Iris-versicolor", "Iris-virginica"]

colors = ["r","g","b"]

for target, color in zip(targets, colors):

indicesToKeep = finaldf["target"] == target

ax.scatter(finaldf.loc[indicesToKeep,"pc1"],

finaldf.loc[indicesToKeep,"pc2"],

c = colors,

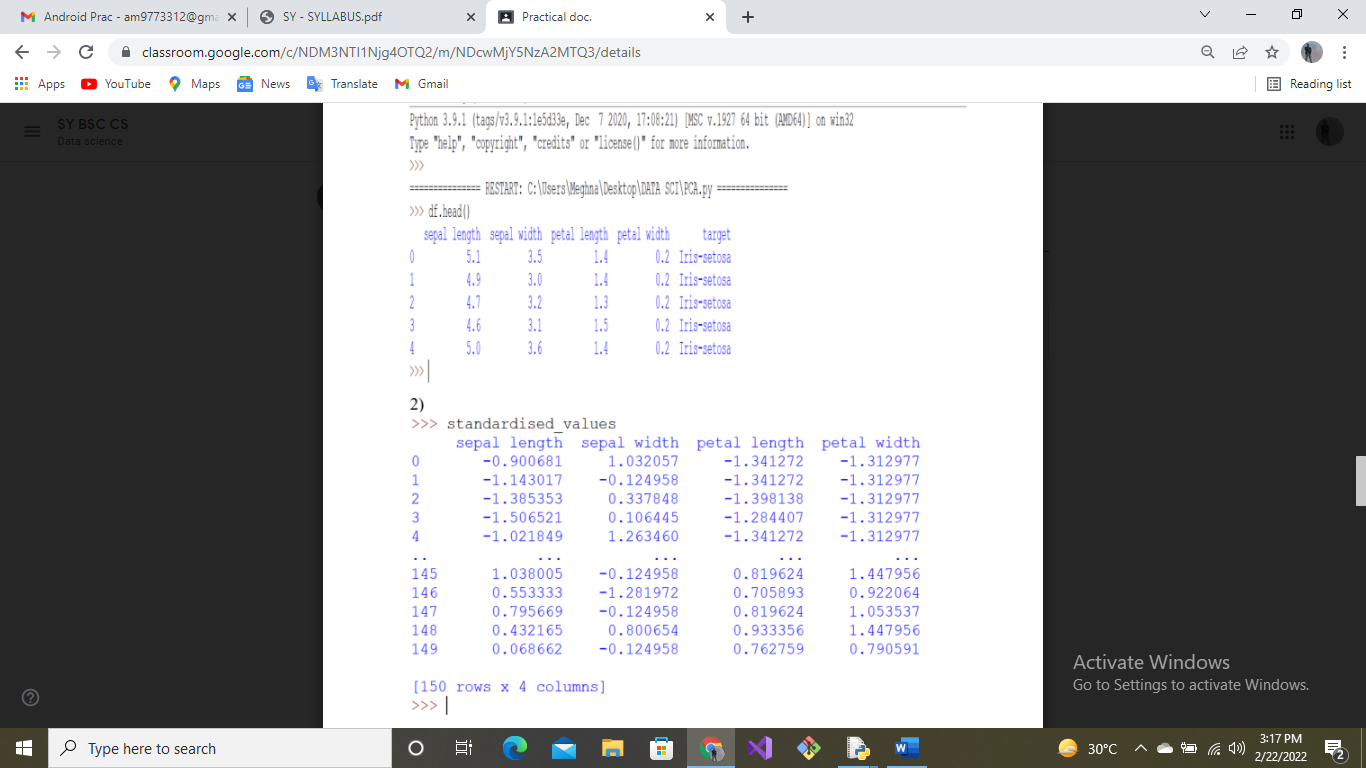
s = 50)

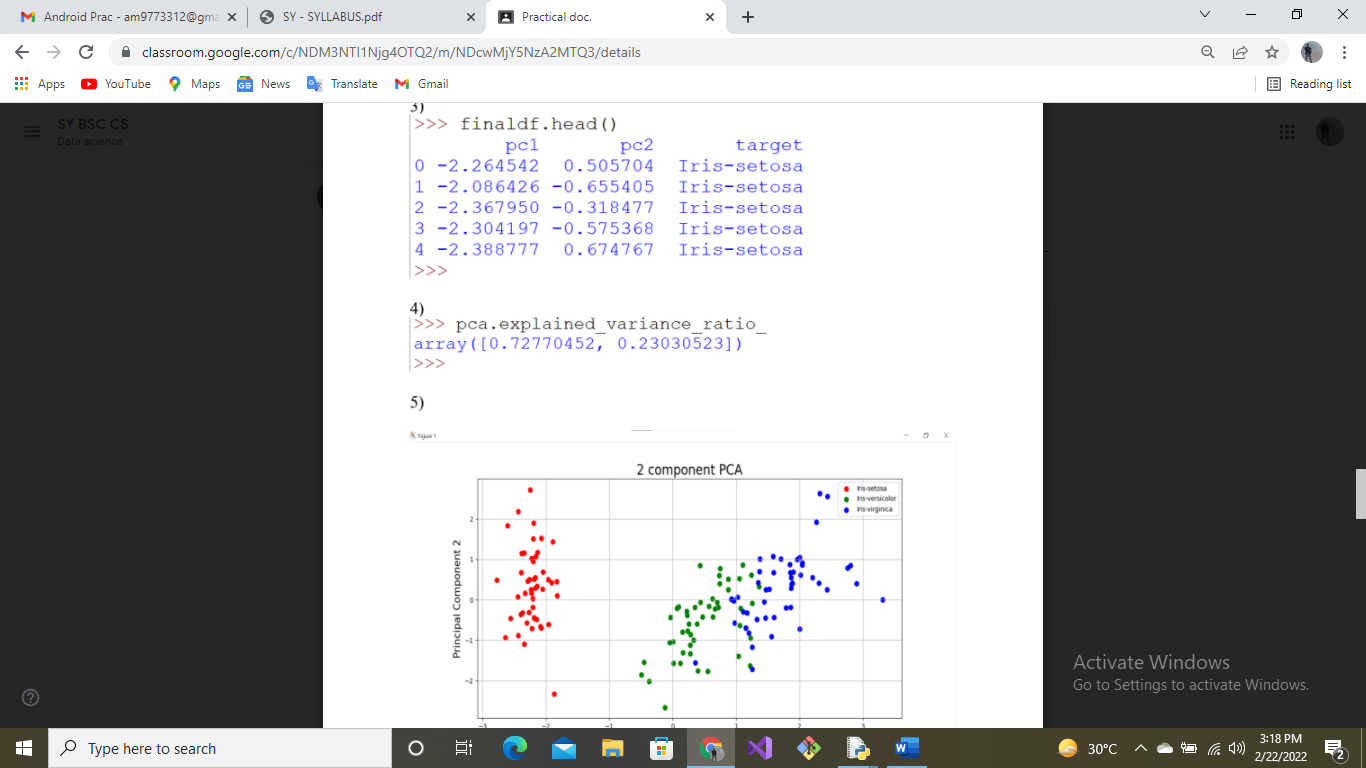
ax.legend(targets)

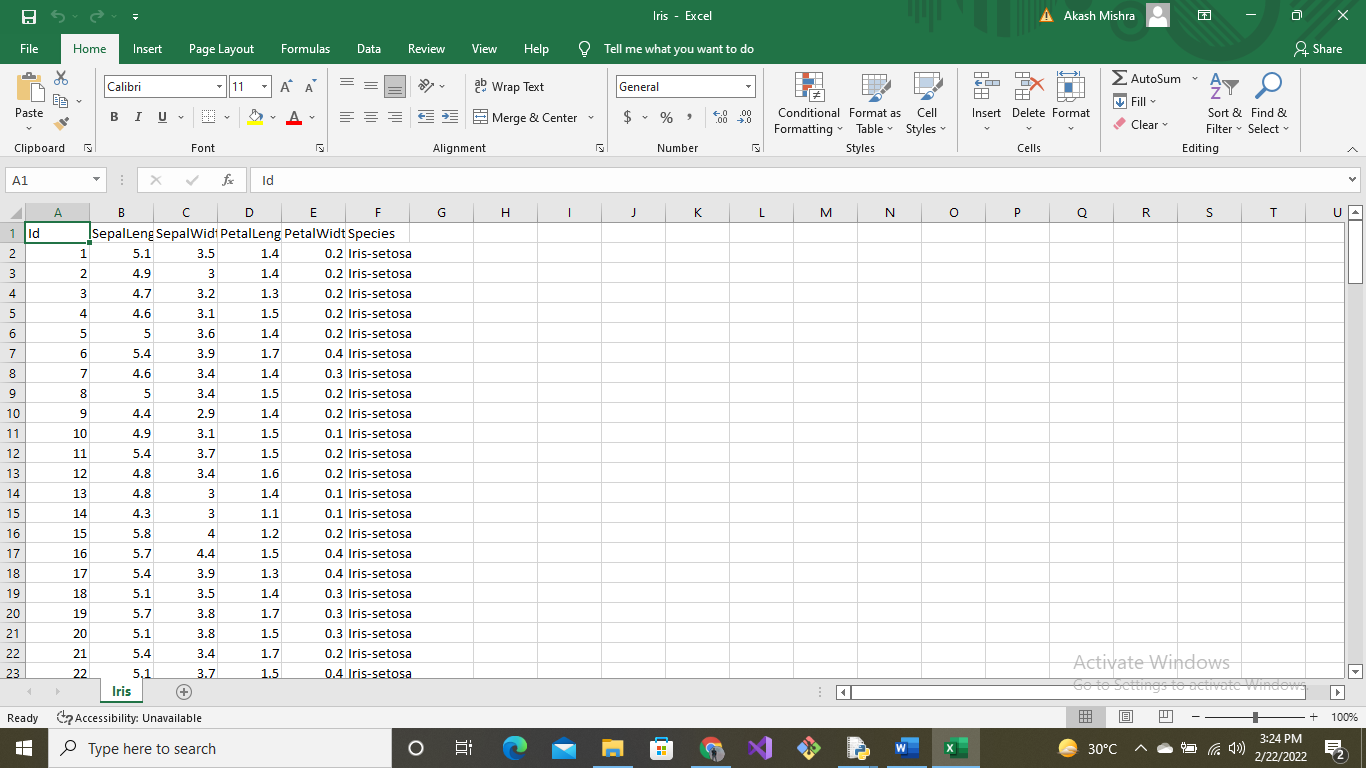
ax.grid()

pca.explained\_variance\_ratio

**OUTPUT**:







Prac\_10-80% 2-%

**INPUT**:

import numpy as np

import matplotlib.pyplot as plt

def estimate\_coef(x, y):

# number of observations/points

n = np.size(x)

# mean of x and y vector

m\_x = np.mean(x)

m\_y = np.mean(y)

# calculating cross-deviation and deviation about x

SS\_xy = np.sum(y\*x) - n\*m\_y\*m\_x

SS\_xx = np.sum(x\*x) - n\*m\_x\*m\_x

# calculating regression coefficients

b\_1 = SS\_xy / SS\_xx

b\_0 = m\_y - b\_1\*m\_x

return (b\_0, b\_1)

def plot\_regression\_line(x, y, b):

# plotting the actual points as scatter plot

plt.scatter(x, y, color = "b",

marker = "o", s = 30)

# predicted response vector

y\_pred = b[0] + b[1]\*x

# plotting the regression line

plt.plot(x, y\_pred, color = "g")

# putting labels

plt.xlabel('x')

plt.ylabel('y')

# function to show plot

plt.show()

def main():

# observations / data

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

# estimating coefficients

b = estimate\_coef(x, y)

print("Estimated coefficients:\nb\_0 = {} \

\nb\_1 = {}".format(b[0], b[1]))

# plotting regression line

plot\_regression\_line(x, y, b)

if \_\_name\_\_ == "\_\_main\_\_":

main()

import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets, linear\_model, metrics

# load the boston dataset

boston = datasets.load\_boston(return\_X\_y=False)

# defining feature matrix(X) and response vector(y)

X = boston.data

y = boston.target

# splitting X and y into training and testing sets

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4,

random\_state=1)

# create linear regression object

reg = linear\_model.LinearRegression()

# train the model using the training sets

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

# regression coefficients

print('Coefficients: ', reg.coef\_)

# variance score: 1 means perfect prediction

print('Variance score: {}'.format(reg.score(X\_test, y\_test)))

# plot for residual error

## setting plot style

plt.style.use('fivethirtyeight')

## plotting residual errors in training data

plt.scatter(reg.predict(X\_train), reg.predict(X\_train) - y\_train,

color = "green", s = 10, label = 'Train data')

## plotting residual errors in test data

plt.scatter(reg.predict(X\_test), reg.predict(X\_test) - y\_test,

color = "blue", s = 10, label = 'Test data')

## plotting line for zero residual error

plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)

## plotting legend

plt.legend(loc = 'upper right')

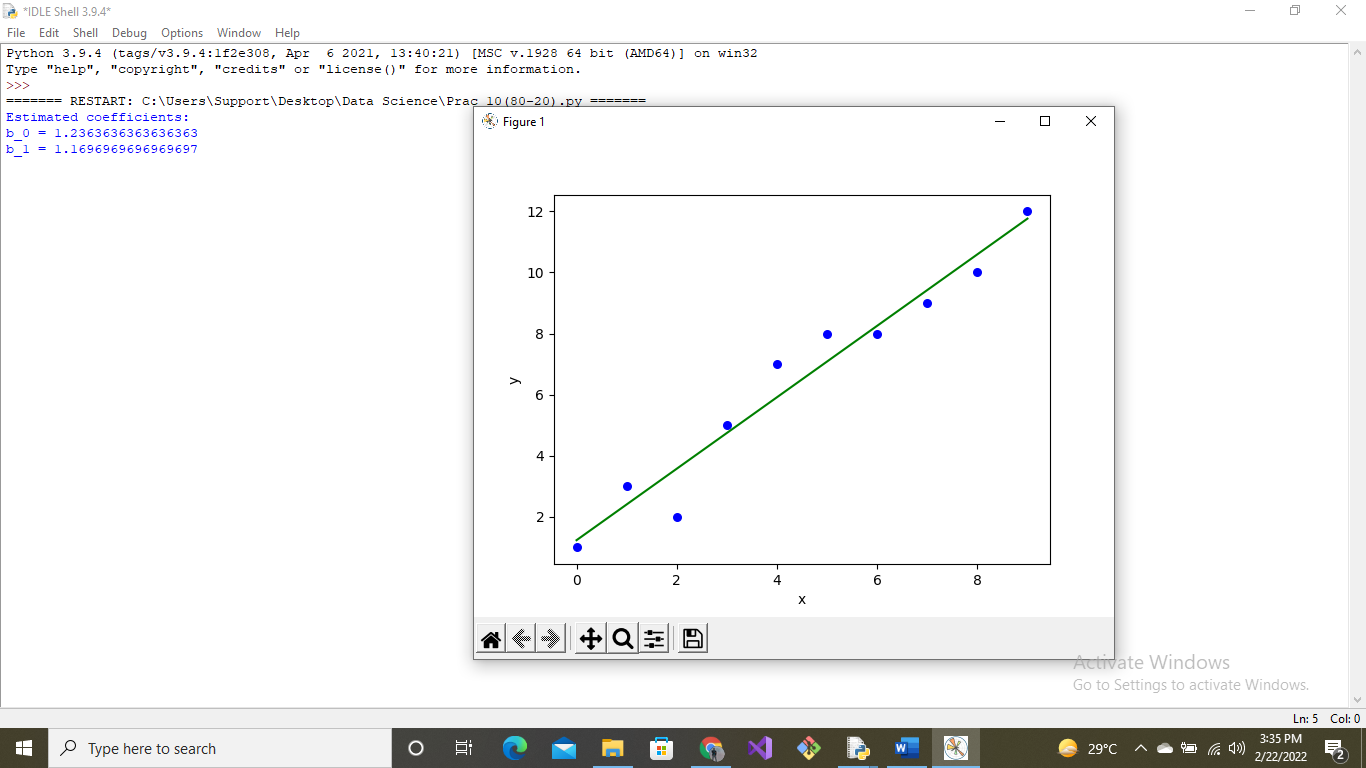
## plot title

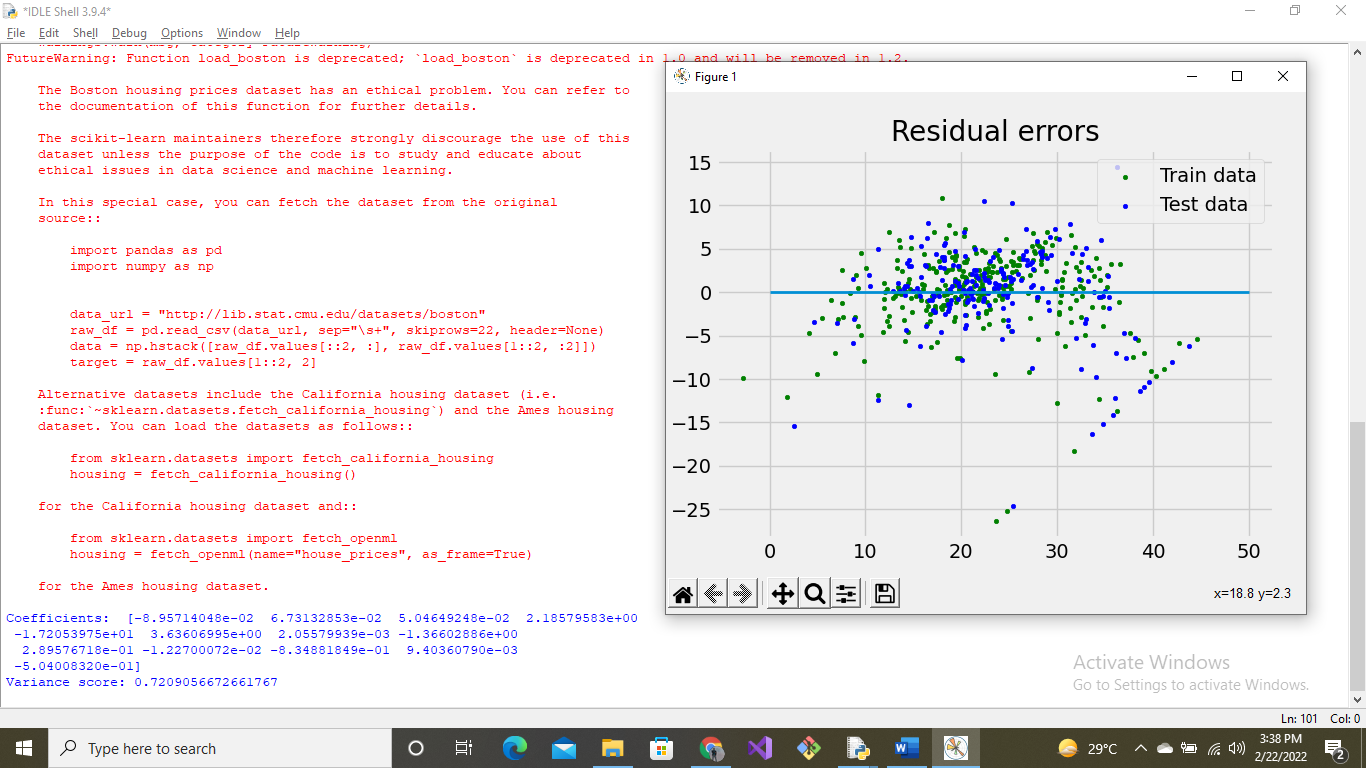
plt.title("Residual errors")

## method call for showing the plot

plt.show()

**OUTPUT**:





Prac\_11-SVM

**INPUT**:

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import pandas as pd

apples\_oranges = pd.read\_csv("apples\_and\_oranges.csv")

apples\_oranges.head()

# create a dictionary to colour classes

color\_dict = dict({'orange':'orange',

'apple':'green'})

# scatterplot

plt.xlabel('Weight')

plt.ylabel('Size')

plt.title('Sizes and Weights of apples and oranges')

sns.scatterplot(data=apples\_oranges, x="Weight", y="Size", hue="Class", palette = color\_dict)

plt.show()

# define input data

X = apples\_oranges[["Weight", "Size"]]

# define target

y = apples\_oranges.Class

# fitting the support vector machine using a linear kernel

from sklearn import svm

clf = svm.SVC(kernel = 'linear', C=10)

clf.fit(X, y)

b = clf.intercept\_

w\_1 = clf.coef\_[0][0]

w\_2 = clf.coef\_[0][1]

b, w\_1, w\_2

# plotting the hyperplane and support vector lines

ax = plt.gca()

sns.scatterplot(data=apples\_oranges, x="Weight", y="Size", hue="Class", palette = color\_dict)

xlim = ax.get\_xlim()

ylim = ax.get\_ylim()

xx = np.linspace(xlim[0], xlim[1], 30)

yy = np.linspace(ylim[0], ylim[1], 30)

YY, XX = np.meshgrid(yy, xx)

xy = np.vstack([XX.ravel(), YY.ravel()]).T

Z = clf.decision\_function(xy).reshape(XX.shape)

ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,

linestyles=['--', '-', '--'])

ax.scatter(clf.support\_vectors\_[:, 0], clf.support\_vectors\_[:, 1], s=100,

linewidth=1, facecolors='none', edgecolors='k')

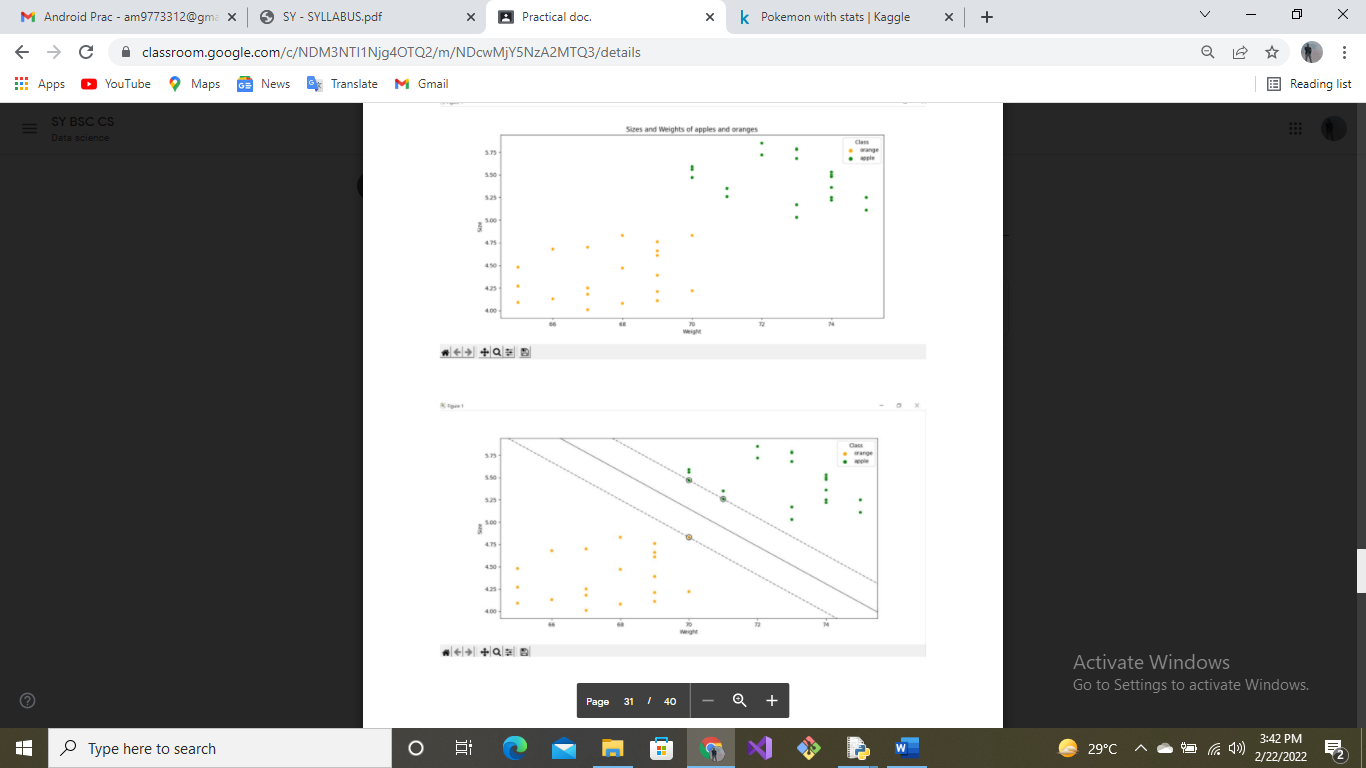
plt.show()

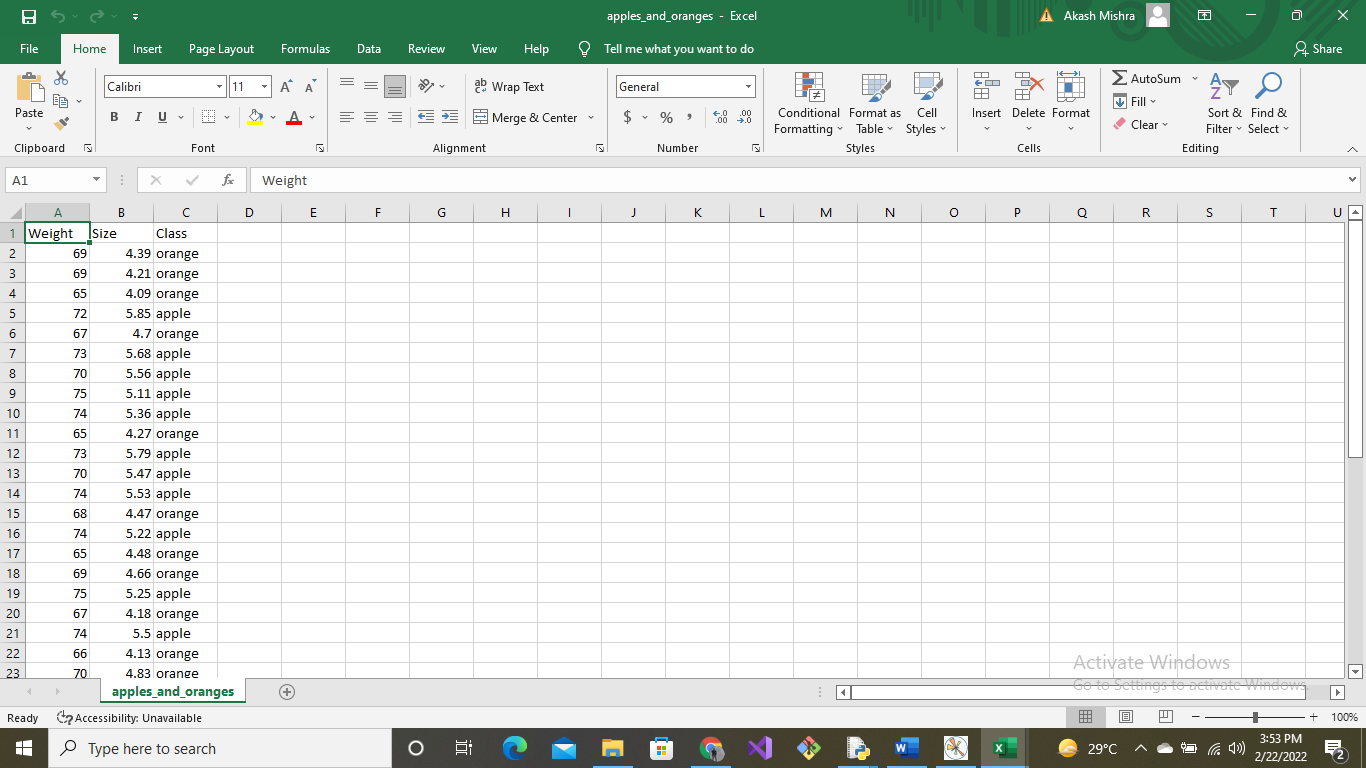
# obtain support vectors

clf.support\_vectors\_

clf.predict([[70, 4.6]])

**OUTPUT**:





Prac\_12-K means

**INPUT**:

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

import pandas as pd

df = pd.read\_csv('train\_and\_test2.csv')

df

kmeans\_model = KMeans(n\_clusters=6,max\_iter=1000)

kmeans\_model.fit(df[['Age','Fare']])

KMeans(algorithm='auto', copy\_x=True, init='k-means++', max\_iter=1000,

n\_clusters=6, n\_init=10, n\_jobs=None, precompute\_distances='auto',

random\_state=None, tol=0.0001, verbose=0)

color\_dictionary = {0: 'red', 1: 'blue', 2: 'green' , 3: 'yellow',4: 'pink', 5: 'black'}

label\_list = kmeans\_model.labels\_.tolist()

df['color'] = label\_list

for i in color\_dictionary:

df['color'] = df['color'].replace(i,color\_dictionary[i])

y = df['Fare']

x = df ['Age']

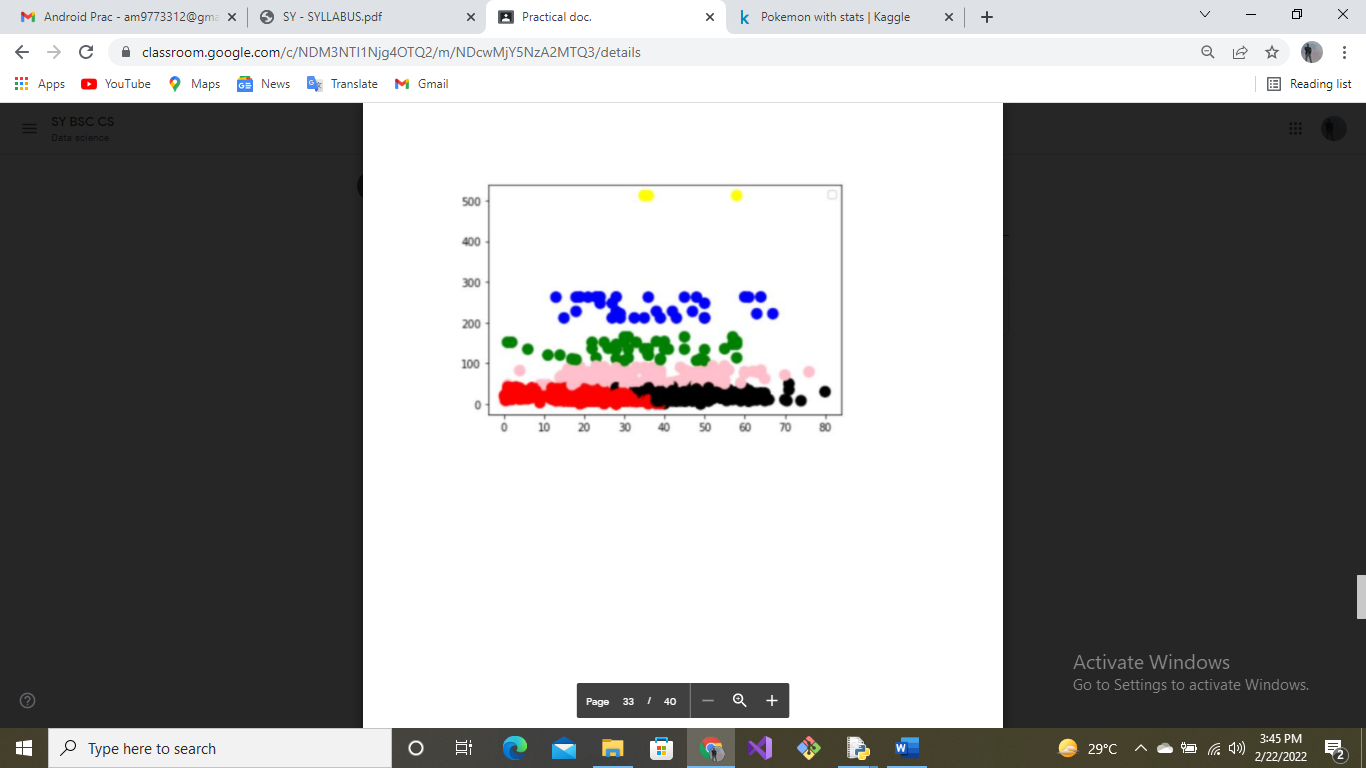
fig, ax = plt.subplots()

ax.scatter(x, y,c = df['color'], s = 100)

ax.legend()

plt.show()

**OUTPUT**:



Prac\_13-KNN

**INPUT**:

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('C:\Users\Support\Desktop\Data Science\Social\_Network\_Ads - Shortcut.lnk')

#X contains the attributes.

#Because we don’t want to take in consideration the first two columns, we will copy only column 2 and 3

X = dataset.iloc[:, [2, 3]].values

#The labels are in the 4th column, so we will copy this column in variable y

y = dataset.iloc[:, 4].values

#sklearn has the method called train\_test\_split, which will split our data set returning 4 values

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

#25% of the data set for test and 75% for train.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

#Distance is much heigher between salary and age column to resolve this issue we used standardscaler.

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

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

#importing the KNeighborsClassifier from sklearn. This takes multiple parameters.

#n\_neighbors,algoritm(for design structure),matric (for distance by default euclidean, Manhattan)

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors = 2)

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

# Predicting the Test set results

y\_pred = classifier.predict(X\_test)

# Making the Confusion Matrix to check the accuracy

#TP+TN/TP+FN+FP+TN (formula TP=true positive,

#FN=false negative,[Observation is positive, but is predicted negative]

#TN=true negative, [ Observation is negative, and is predicted to be negative]

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Visualising the Training set results

#meshgrid() creates a rectangular grid out of an array of x values and an array of y values here x = X1 and y = X2

#contourf method use to fill the background with the color of the class

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Classifier (Training set)')

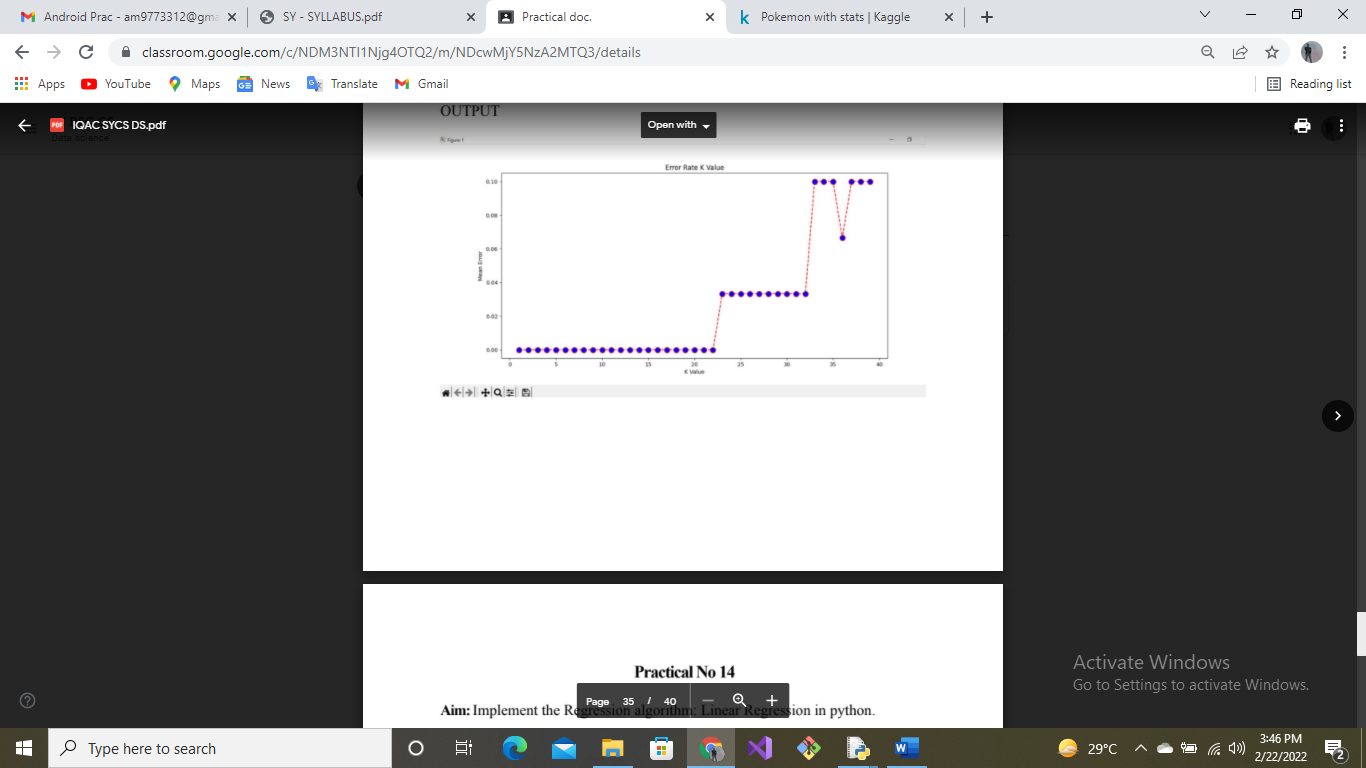
plt.xlabel('Age')

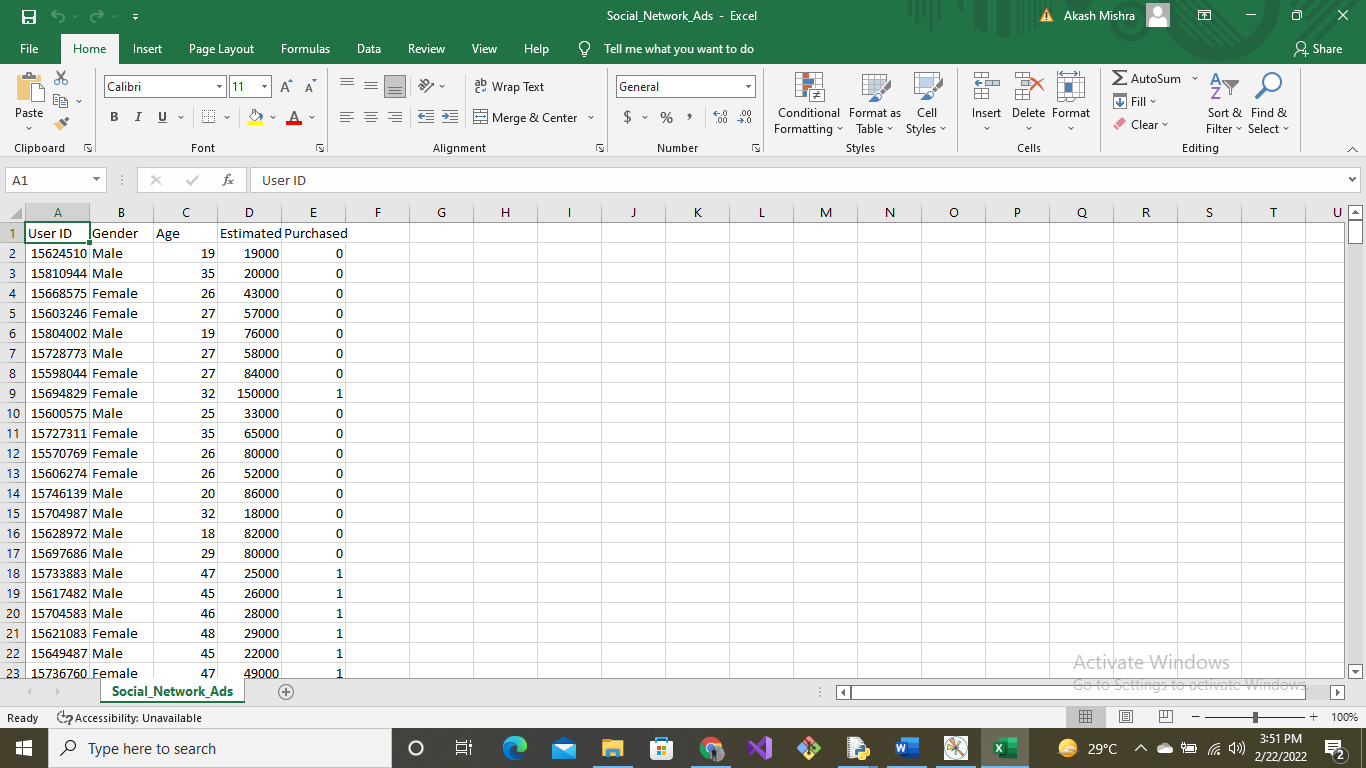
plt.ylabel('Estimated Salary')

plt.legend() #use to Place a legend on the axes.

plt.show()

**OUTPUT**:





Prac\_14

**INPUT**:-MULTIPLE

import numpy as np

import matplotlib.pyplot as plt

def estimate\_coef(x, y):

# number of observations/points

n = np.size(x)

# mean of x and y vector

m\_x = np.mean(x)

m\_y = np.mean(y)

# calculating cross-deviation and deviation about x

SS\_xy = np.sum(y\*x) - n\*m\_y\*m\_x

SS\_xx = np.sum(x\*x) - n\*m\_x\*m\_x

# calculating regression coefficients

b\_1 = SS\_xy / SS\_xx

b\_0 = m\_y - b\_1\*m\_x

return (b\_0, b\_1)

def plot\_regression\_line(x, y, b):

# plotting the actual points as scatter plot

plt.scatter(x, y, color = "b",

marker = "o", s = 30)

# predicted response vector

y\_pred = b[0] + b[1]\*x

# plotting the regression line

plt.plot(x, y\_pred, color = "g")

# putting labels

plt.xlabel('x')

plt.ylabel('y')

# function to show plot

plt.show()

def main():

# observations / data

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

# estimating coefficients

b = estimate\_coef(x, y)

print("Estimated coefficients:\nb\_0 = {} \

\nb\_1 = {}".format(b[0], b[1]))

# plotting regression line

plot\_regression\_line(x, y, b)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**OUTPUT**:

