**Lab Assignment -5**

**Machine Learning**

**Name** : **Bandla Lavanya**

**Regd Num** : **19MCB0008**

**Topic: Clustering Algorithms**

**Hierarchical clustering:**

Hierarchical clustering is a type of unsupervised machine learning algorithm used to cluster unlabeled data points. Like [K-means clustering](https://stackabuse.com/k-means-clustering-with-scikit-learn/), hierarchical clustering also groups together the data points with similar characteristics. In some cases the result of hierarchical and K-Means clustering can be similar.

Source code:

**Importing the libraries that are required for the doing the hierarchical clustering**

**import matplotlib.pyplot as plt**

**import pandas as pd**

**%matplotlib inline**

**import numpy as np**

**loading the data.**

**The next step is to import or create the dataset**.

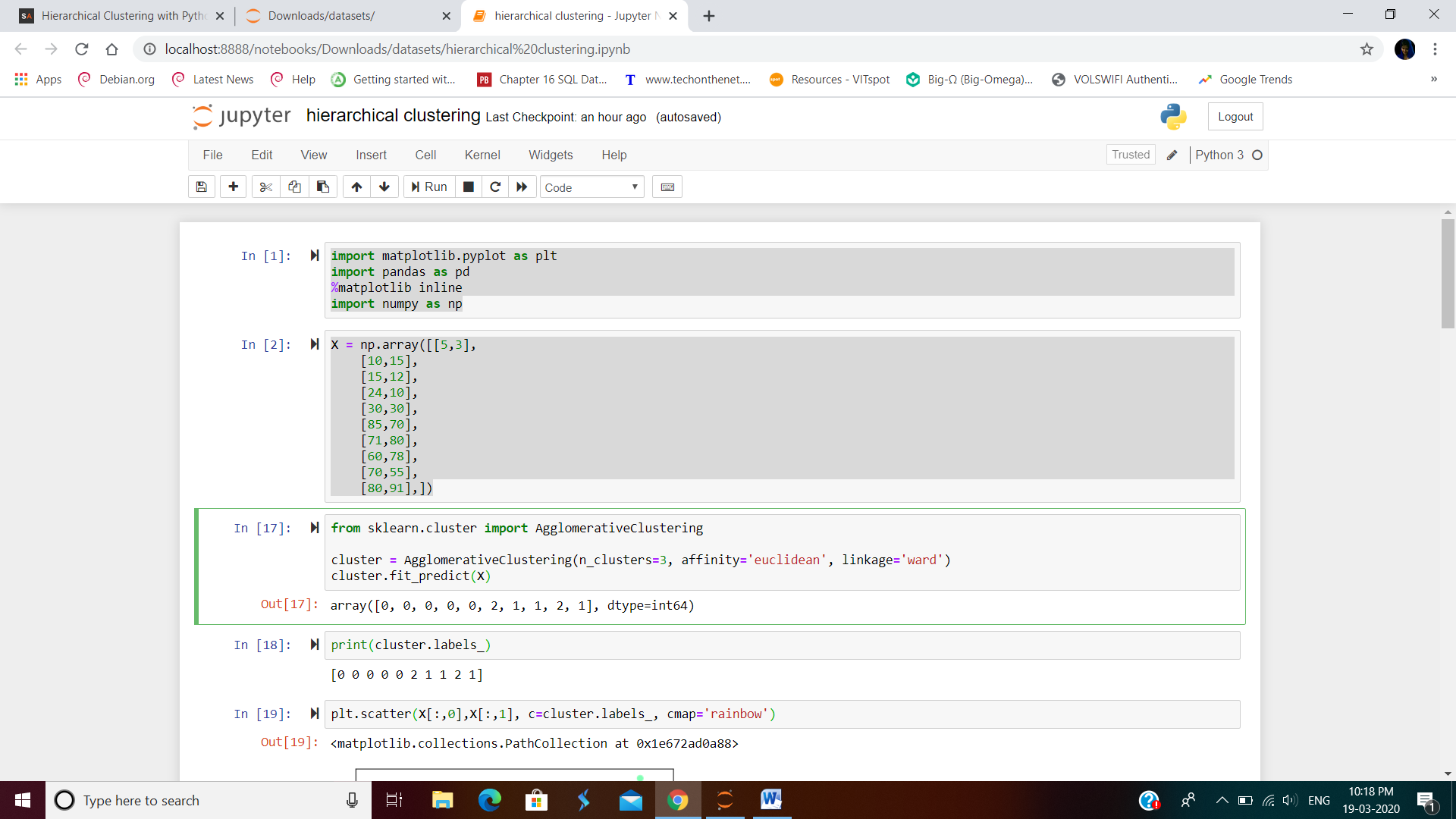
X = np.array([[5,3], [10,15], [15,12], [24,10], [30,30],[85,70], [71,80], [60,78], [70,55],[80,91],])

**The next step is to import the class for clustering and call its fit\_predict method to predict the clusters that each data point belongs to.In the code above we import the AgglomerativeClustering class from the "sklearn.cluster" library. The number of parameters is set to 3 using the n\_clusters parameter while the affinity is set to "euclidean" (distance between the datapoints). Finally linkage parameter is set to "ward", which minimizes the variant between the clusters.**

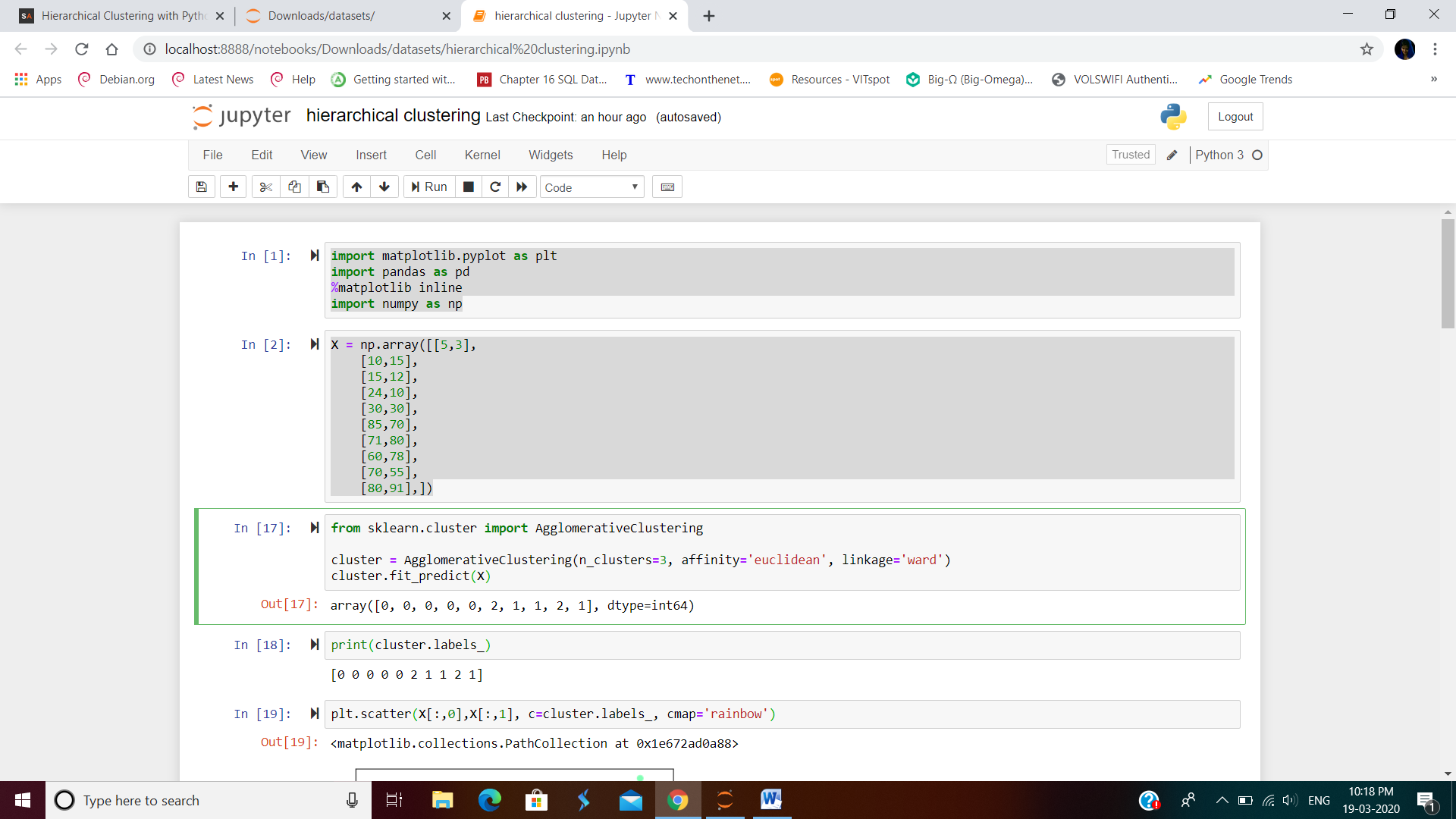
**from sklearn.cluster import AgglomerativeClustering**

**cluster = AgglomerativeClustering(n\_clusters=3, affinity='euclidean', linkage='ward')**

**cluster.fit\_predict(X)**

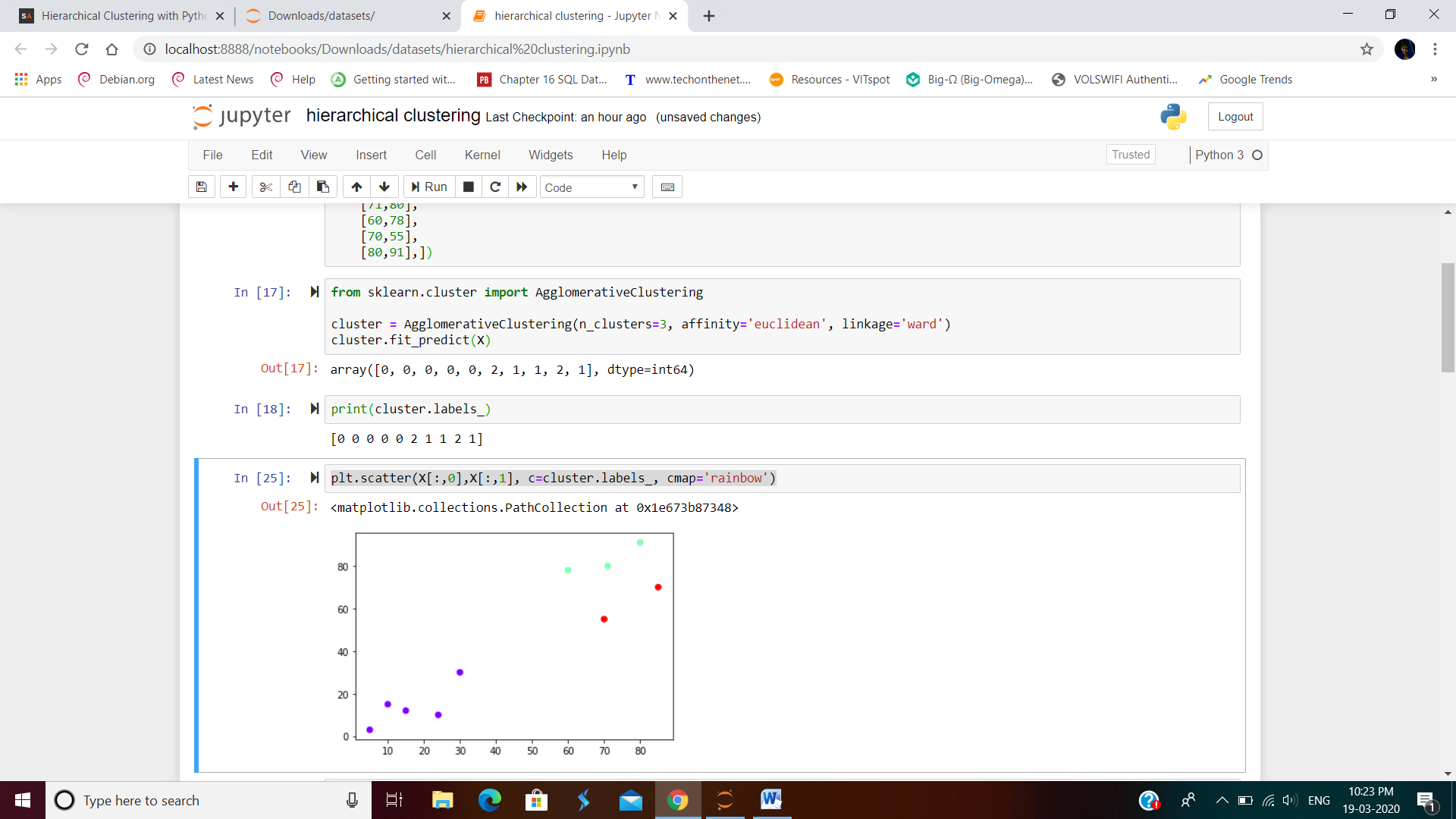


**Next we call the fit\_predict method from the AgglomerativeClustering class variable cluster. This method returns the names of the clusters that each data point belongs to.**



**Plotting the cluster.**

**plt.scatter(X[:,0],X[:,1], c=cluster.labels\_, cmap='rainbow')**



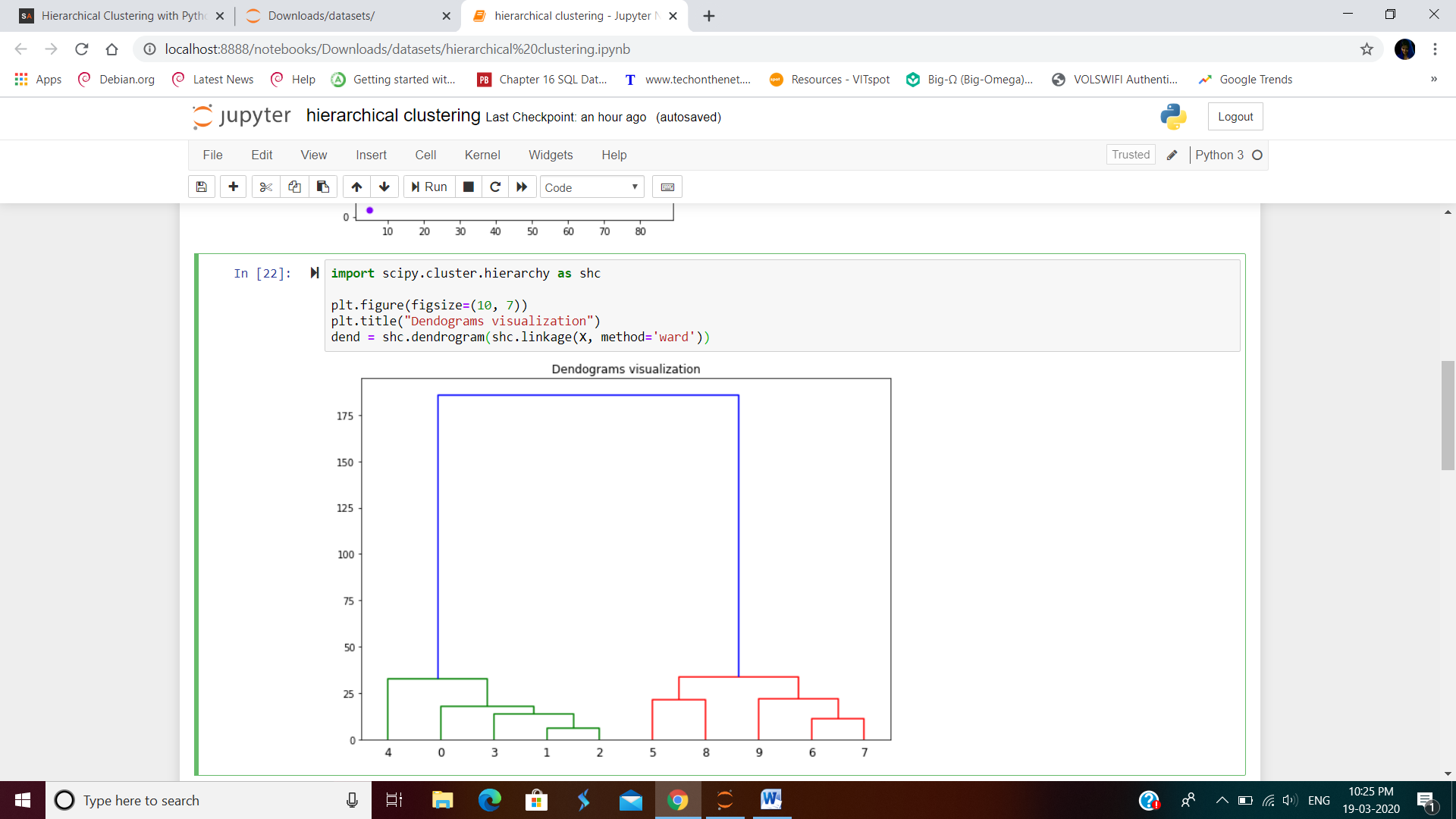
**Next, we need to know the clusters that we want our data to be split to. We will again use the scipy library to create the dendrograms for our dataset. Execute the following script to do so:**

**import scipy.cluster.hierarchy as shc**

**plt.figure(figsize=(10, 7))**

**plt.title("Dendograms visualization")**

**dend = shc.dendrogram(shc.linkage(X, method='ward'))**



**K means clustering:**

***k*-means clustering** is a method of [vector quantization](https://en.wikipedia.org/wiki/Vector_quantization), originally from [signal processing](https://en.wikipedia.org/wiki/Signal_processing), that is popular for [cluster analysis](https://en.wikipedia.org/wiki/Cluster_analysis) in [data mining](https://en.wikipedia.org/wiki/Data_mining). *k*-means clustering aims to [partition](https://en.wikipedia.org/wiki/Partition_of_a_set) *n* observations into *k* clusters in which each observation belongs to the [cluster](https://en.wikipedia.org/wiki/Cluster_(statistics)) with the nearest [mean](https://en.wikipedia.org/wiki/Mean), serving as a prototype of the cluster. This results in a partitioning of the data space into [Voronoi cells](https://en.wikipedia.org/wiki/Voronoi_cell" \o "Voronoi cell). *k*-Means minimizes within-cluster variances (squared Euclidean distances), but not regular Euclidean distances, which would be the more difficult [Weber problem](https://en.wikipedia.org/wiki/Weber_problem): the mean optimizes squared errors, whereas only the geometric median minimizes Euclidean distances

source code:

importing the libarary kmeans for performing the clustering on the data that has taken manually.

**from pandas import DataFrame**

**import matplotlib.pyplot as plt**

**from sklearn.cluster import KMeans**

By creating the data frame we can see data that is imported which consists of X and Y.

**Data = {'x': [25,34,22,27,33,33,31,22,35,34,67,54,57,43,50,57,59,52,65,47,49,48,35,33,44,45,38,43,51,46],**

**'y': [79,51,53,78,59,74,73,57,69,75,51,32,40,47,53,36,35,58,59,50,25,20,14,12,20,5,29,27,8,7] }**

created the DataFrame based on the above data, you’ll need to import 2 additional Python modules:

* matplotlib – for [creating charts in Python](https://datatofish.com/scatter-line-bar-charts-using-matplotlib/)
* sklearn – for applying the K-Means Clustering in Python

In the code below, I specified the number of clusters. For this example, assign 3 clusters as follows:

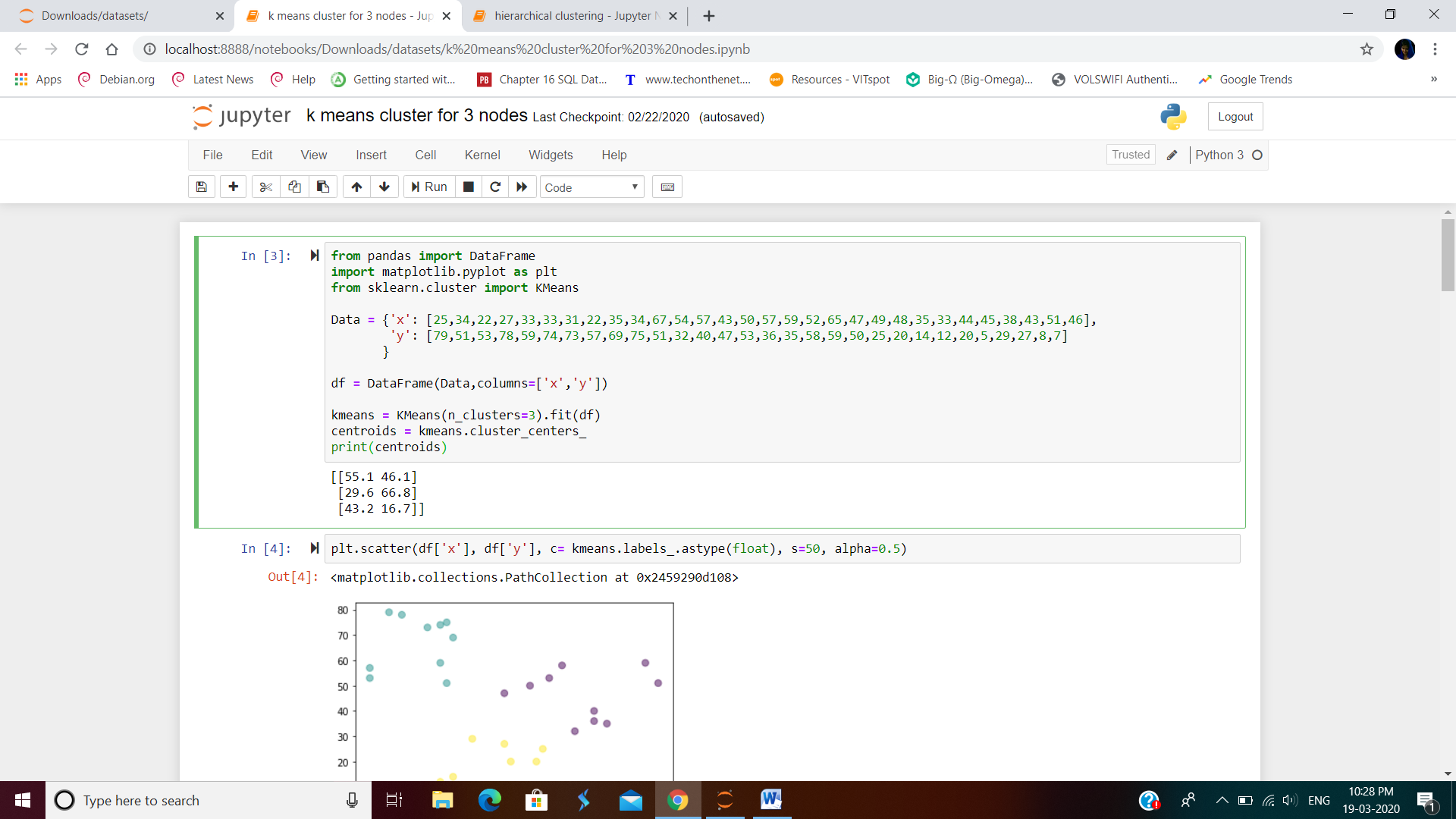
**df = DataFrame(Data,columns=['x','y'])**

**kmeans = KMeans(n\_clusters=3).fit(df)**

**generating the cluster centroids and printing them assigning them to the clusters**

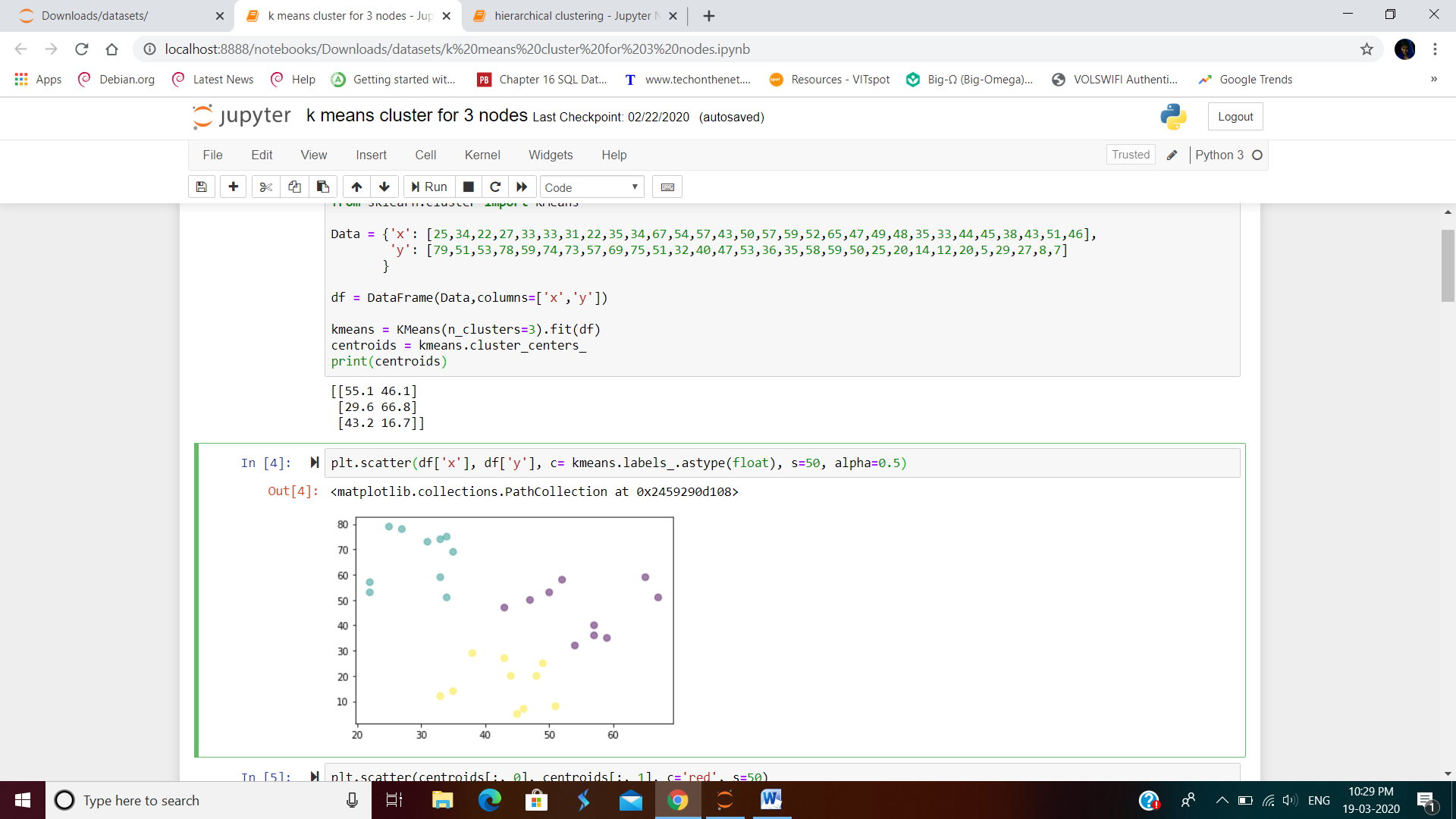
**centroids = kmeans.cluster\_centers\_**

**print(centroids)**



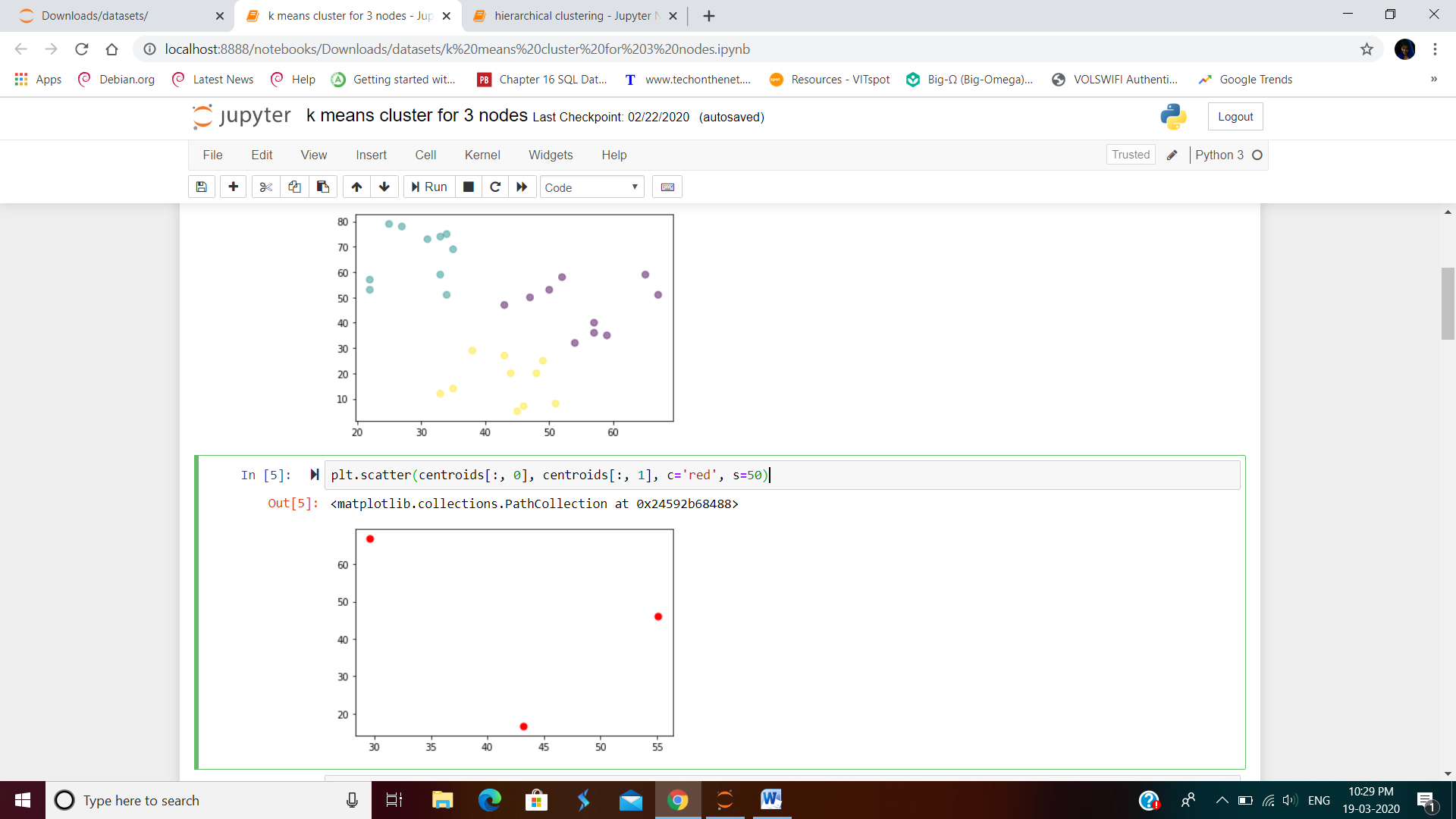
**By using the scatter plot the data is represented with the value of alpha as 0.5**

**plt.scatter(df['x'], df['y'], c= kmeans.labels\_.astype(float), s=50, alpha=0.5)**



**Plotting only the centroids**

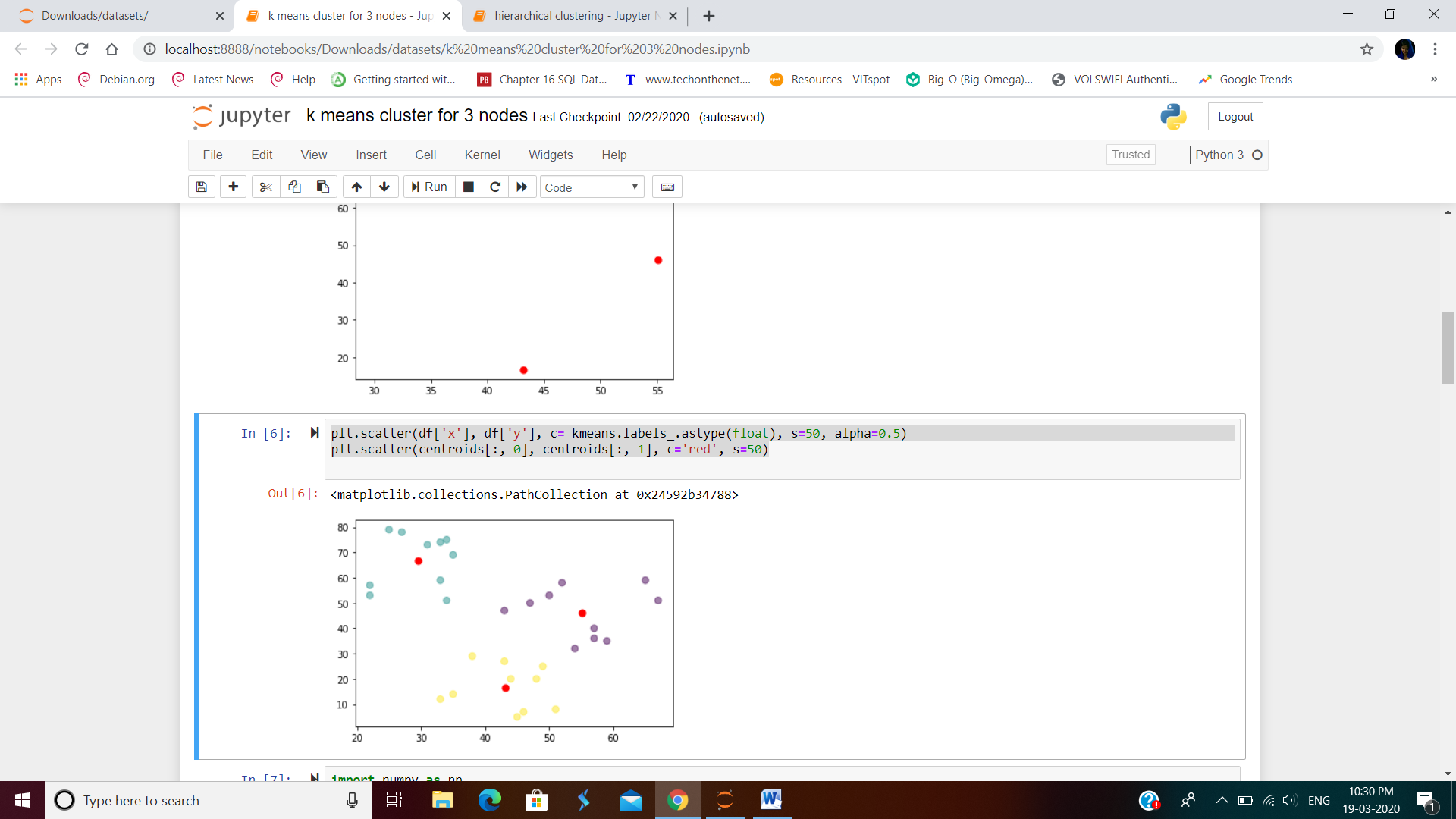
**plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=50)**



**the observations that belong to a given cluster are closer to the center of that cluster, in comparison to the centers of other clusters.**

**plt.scatter(df['x'], df['y'], c= kmeans.labels\_.astype(float), s=50, alpha=0.5)**

**plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=50)**



**K mode clustering:**

k-modes is used for clustering categorical variables. It defines clusters based on the number of matching categories between data points. (This is in contrast to the more well-known k-means algorithm, which clusters numerical data based on Euclidean distance.) The k-prototypes algorithm combines k-modes and k-means and is able to cluster mixed numerical / categorical data.

**First for running the code we need to install the kmode package**

The code is modeled after the clustering algorithms in scikit-learn and has the same familiar interface.

Pip install kmodes

To upgrade the package latest version.

pip install --upgrade kmodes

source code:

import numpy as np

from kmodes.kmodes import KModes

# random categorical data

data = np.random.choice(20, (100, 10))

km = KModes(n\_clusters=4, init='Huang', n\_init=5, verbose=1)

clusters = km.fit\_predict(data)

# Print the cluster centroids

print(km.cluster\_centroids\_)

