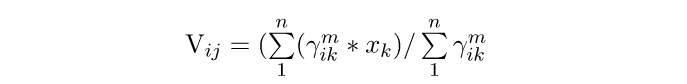
**EXPERIMENT: 8**

**Aim:** Write a program to perform Fuzzy c-means Clustering.

**Description:**

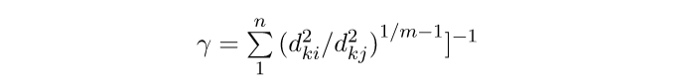
Fuzzy C-Means clustering is a soft clustering approach, where each data point is assigned a likelihood or probability score to belong to that cluster. The step-wise approach of the Fuzzy c-means clustering algorithm is:

* Fix the value of c (number of clusters), and select a value of m (generally 1.25<m<2), and initialize partition matrix U.
* Calculate cluster centers (centroid).

Here,

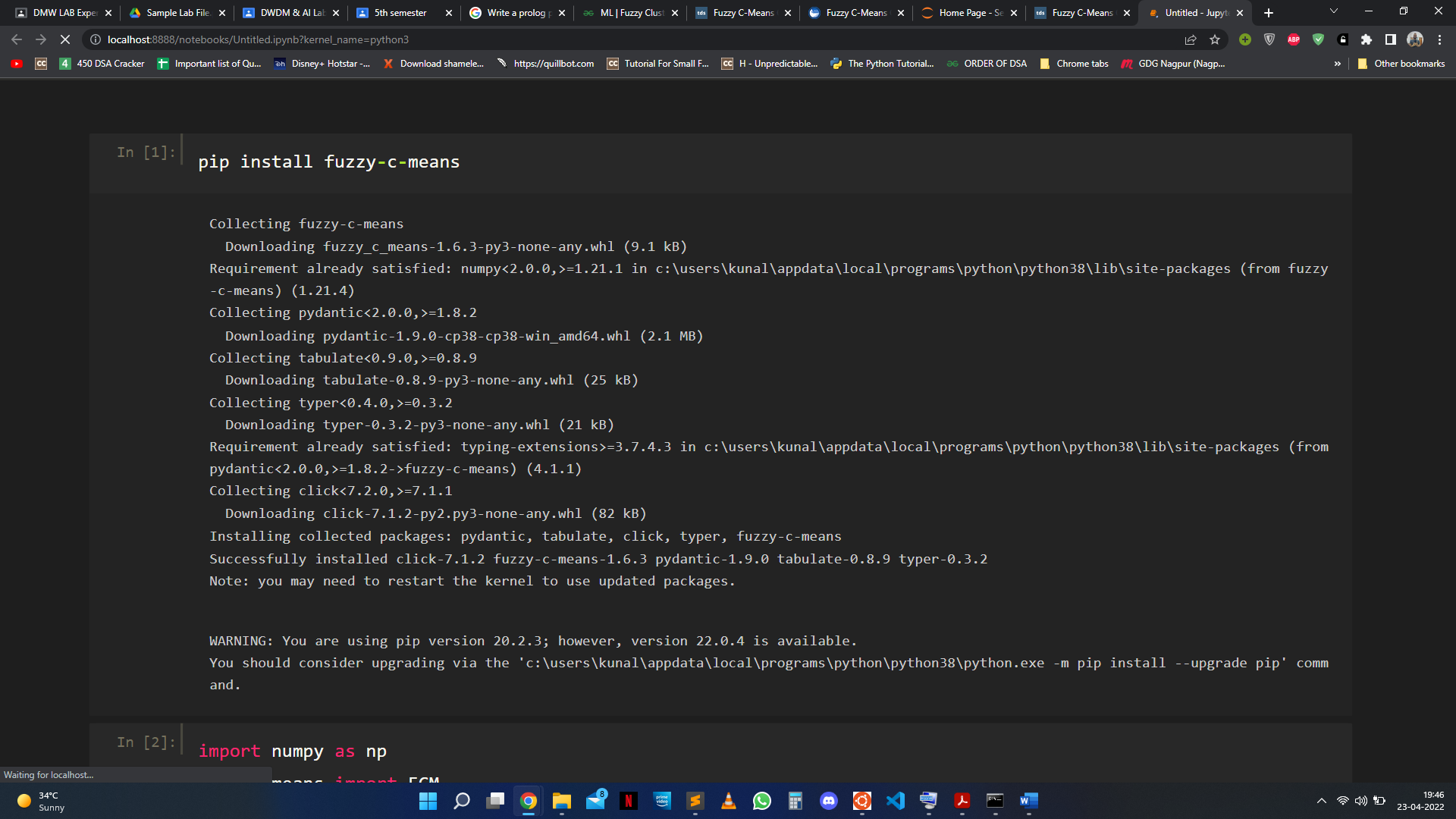
µ: Fuzzy membership value

m: fuzziness parameter

* Update Partition Matrix
* Repeat the above steps until convergence.

**Code and Output:**

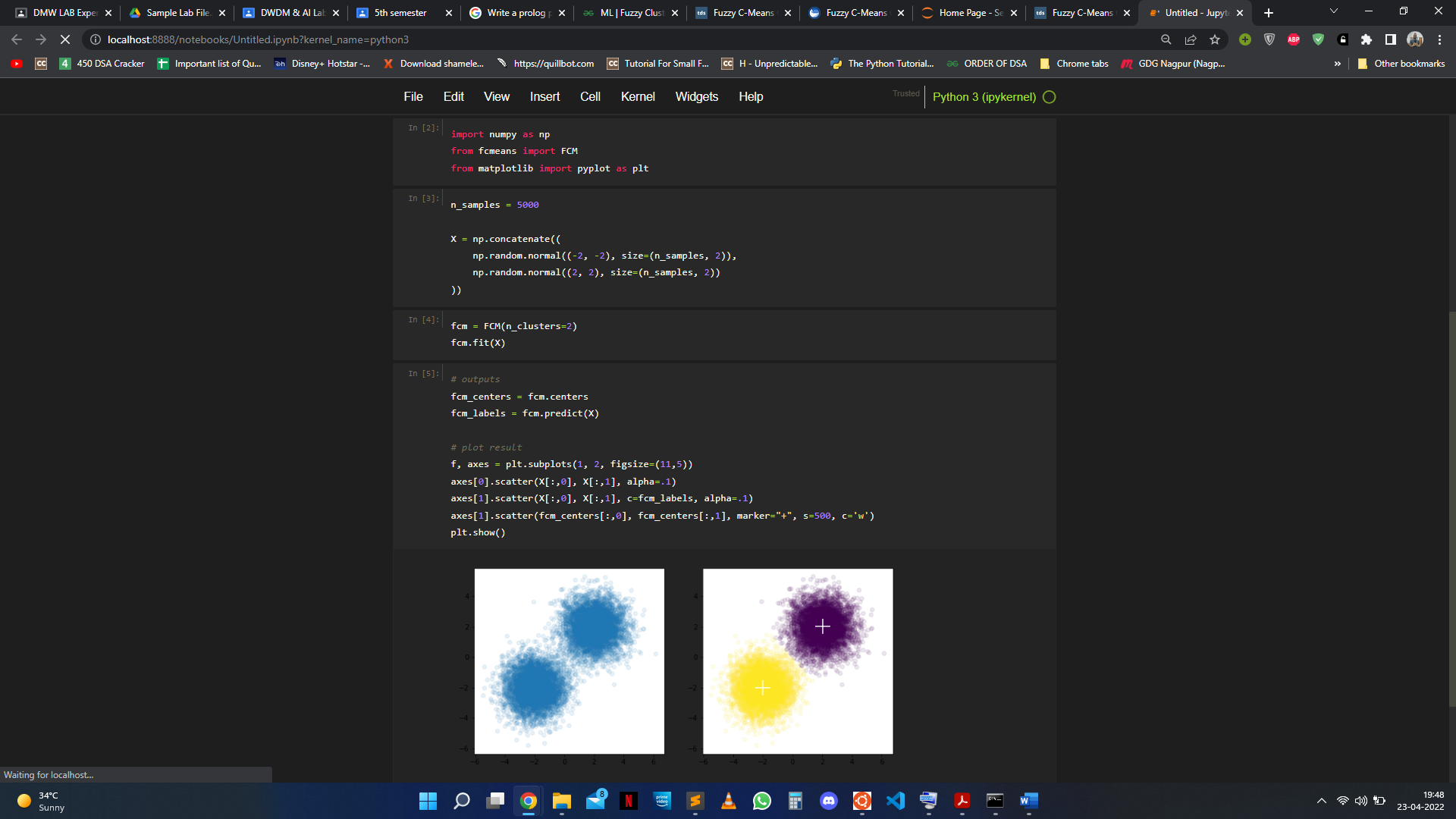
To implement the fuzzy c-means algorithm, we have an open-sourced Python package, that can be installed using PyPl:

pip install fuzzy-c-means

Fuzzy c-means is a Python module that can implement the fuzzy c-means algorithm. This module has an API similar to that of Scikit-learn.

**import** numpy **as** np

**from** fcmeans **import** FCM

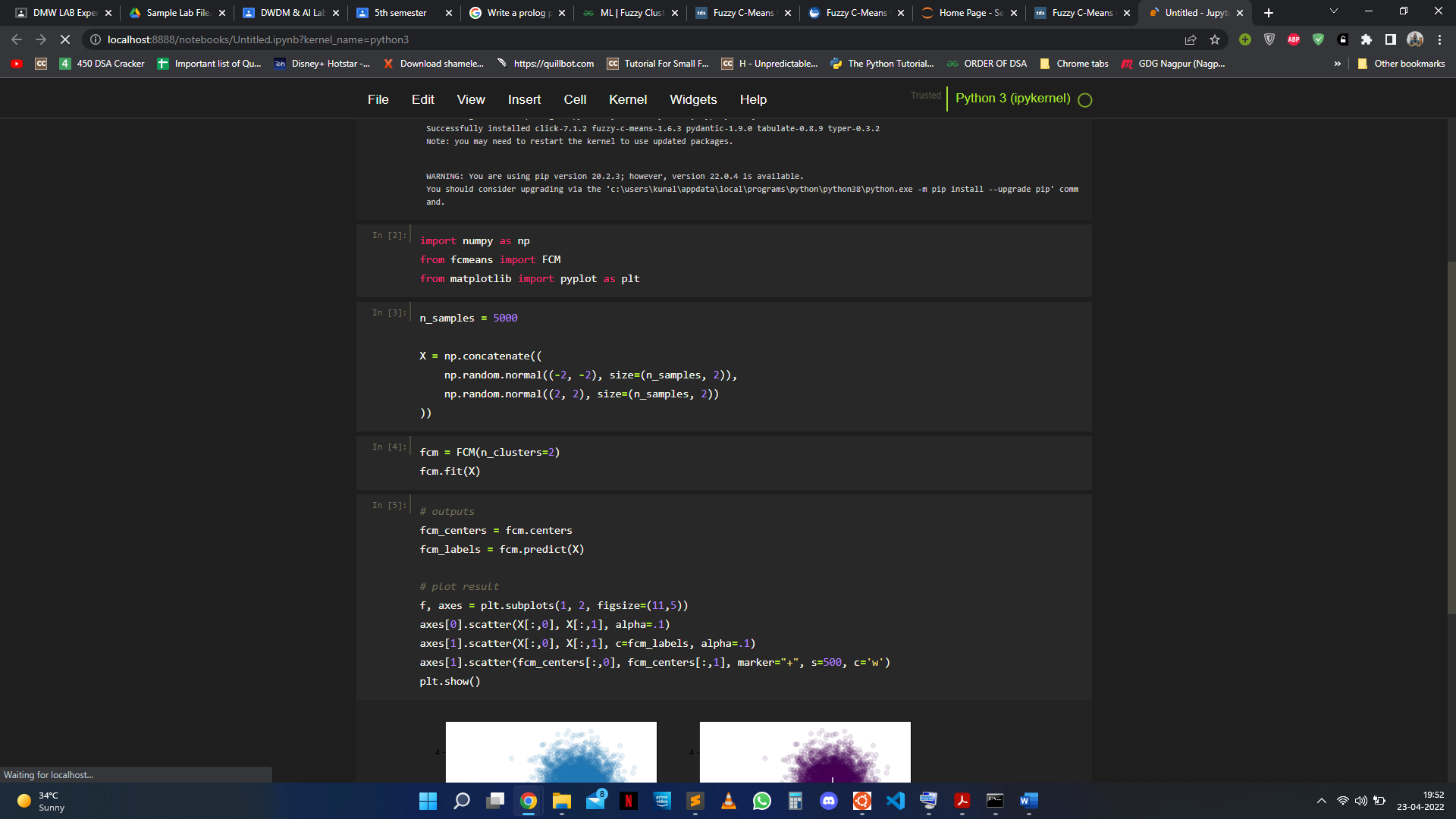
**from** matplotlib **import** pyplot **as** plt

n\_samples **=** 5000

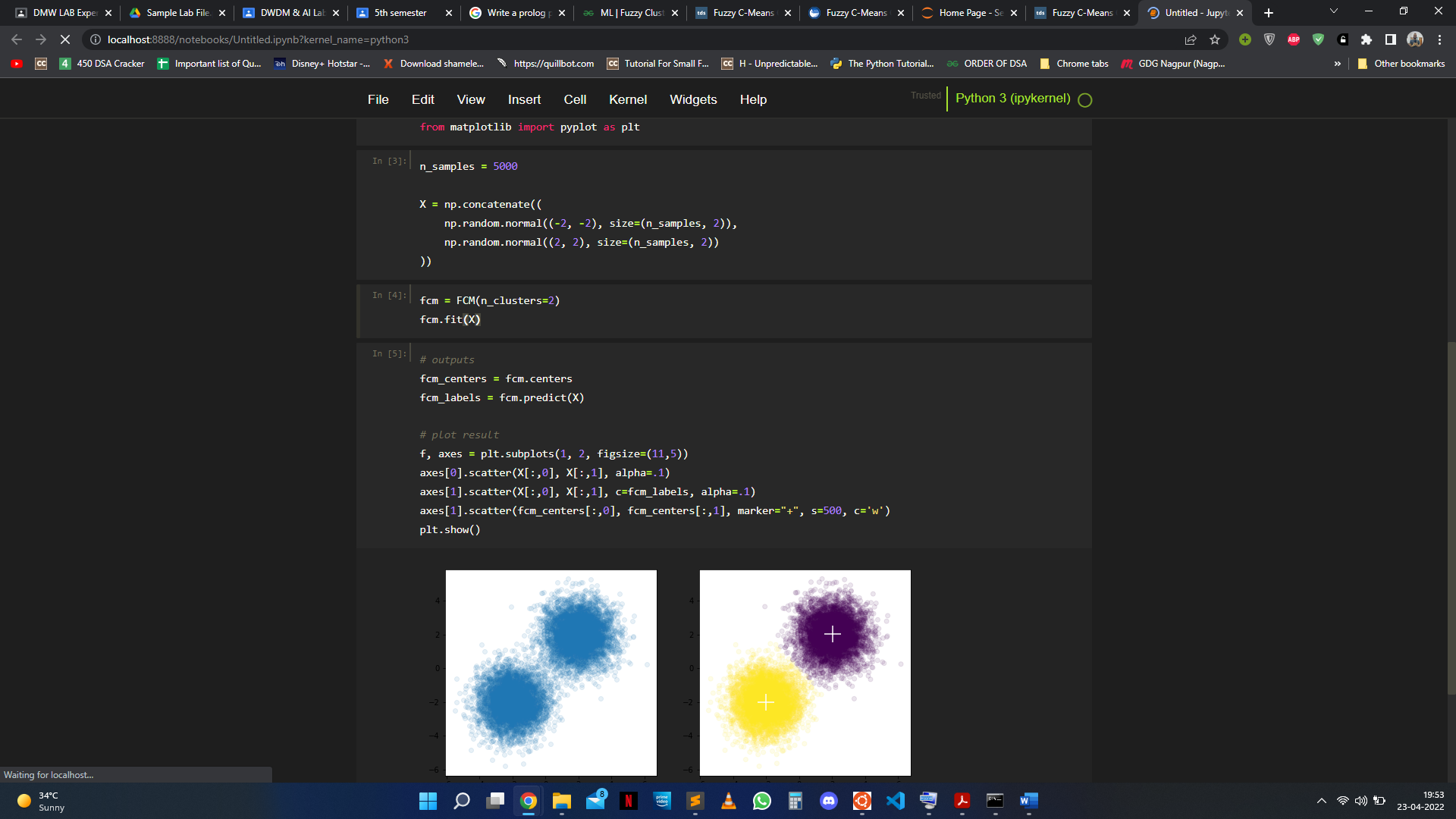
X **=** np**.**concatenate((

np**.**random**.**normal((**-**2, **-**2), size**=**(n\_samples, 2)),

np**.**random**.**normal((2, 2), size**=**(n\_samples, 2))

))

fcm **=** FCM(n\_clusters**=**2)

fcm**.**fit(X)

*# outputs*

fcm\_centers **=** fcm**.**centers

fcm\_labels **=** fcm**.**predict(X)

*# plot result*

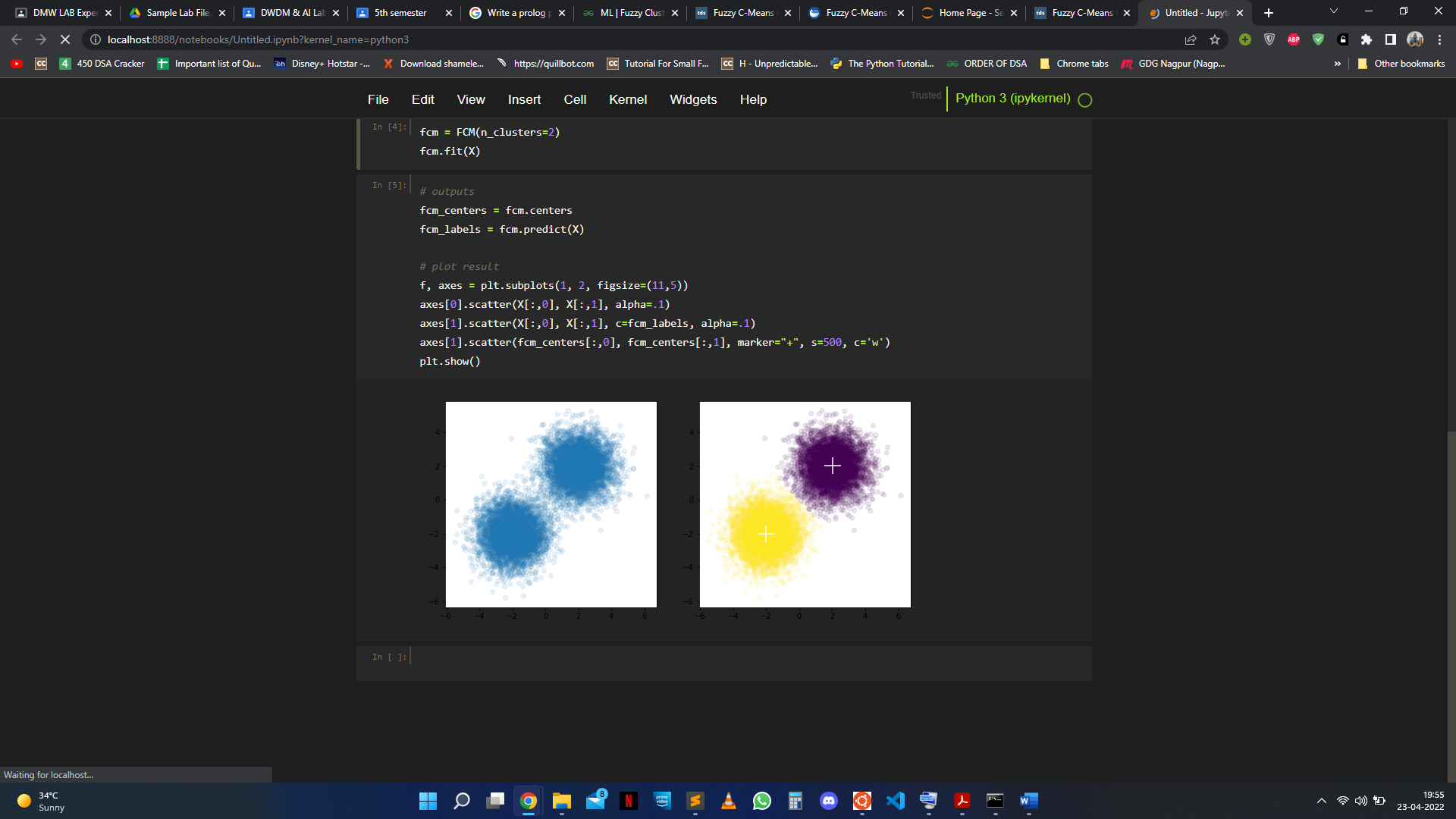
f, axes **=** plt**.**subplots(1, 2, figsize**=**(11,5))

axes[0]**.**scatter(X[:,0], X[:,1], alpha**=**.1)

axes[1]**.**scatter(X[:,0], X[:,1], c**=**fcm\_labels, alpha**=**.1)

axes[1]**.**scatter(fcm\_centers[:,0], fcm\_centers[:,1], marker**=**"+", s**=**500, c**=**'w')

plt**.**show()



**EXPERIMENT: 20**

**Aim:** Write a prolog program to reverse a given list of values.

**Description:**

The first parameter in the reverse/3 predicate is the list. The second parameter is an empty list. The third parameter is the reverse list.

The reverse/3 recursively pushes the elements from the beginning of the first list to the front of the second list. This reverses the order of the elements.

Our base condition is reverse([], Y, R). At this point, we have pushed all elements from the input list to Y. We set R to Y and backtrack.

**Code:**

reverse([], Y, R) :-

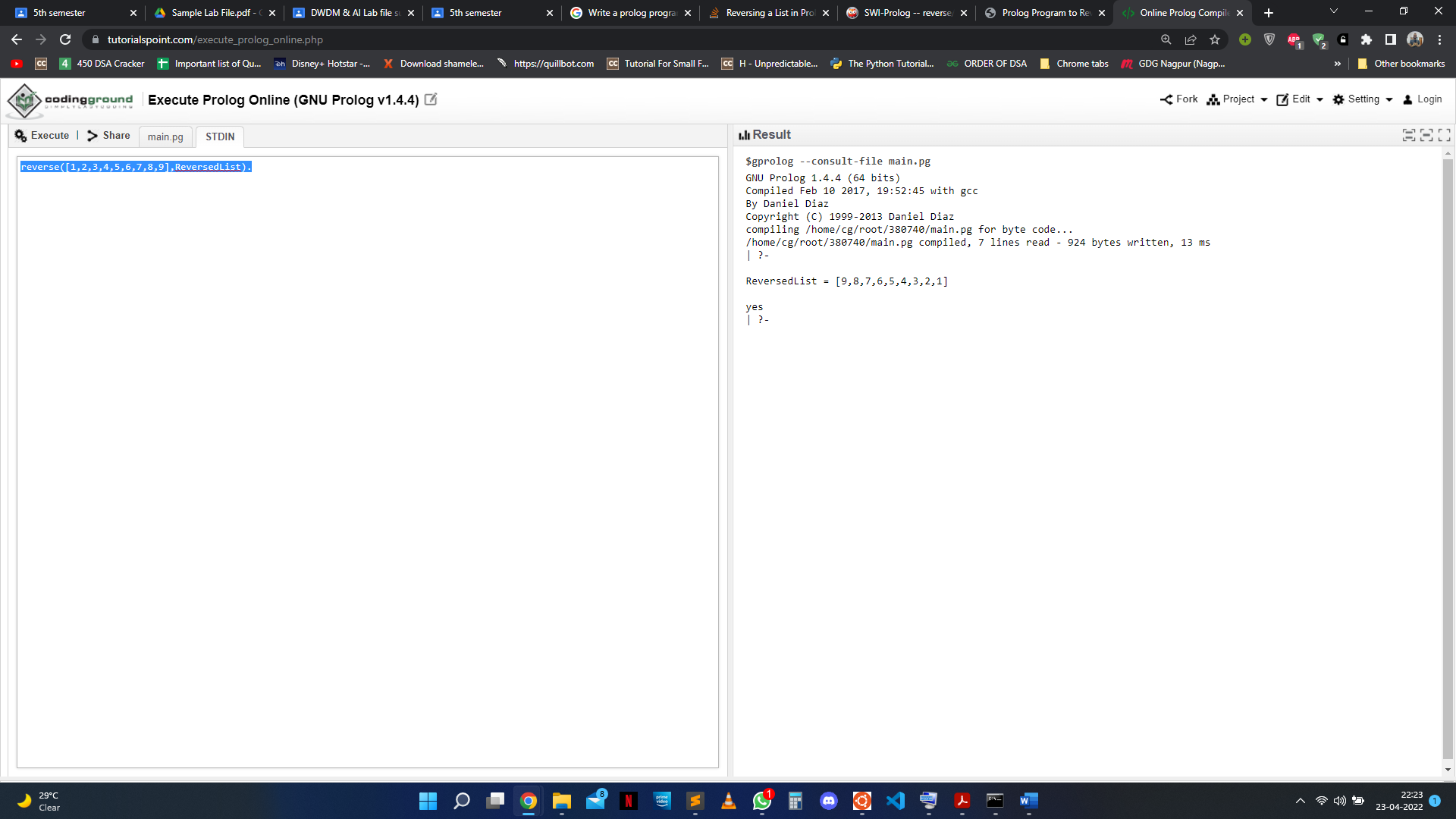
R = Y.

reverse([H|T] , Y, R) :-

reverse(T, [H|Y], R).

**STDIN:**

reverse([1,2,3,4,5,6,7,8,9],[],ReversedList).

**Output:**