4/15/2023

2020244 - Mohsin Zia  
2020474 - Syed Irtaza Haider

Instructor: Professor Zahid Halim

CS-351-Artificial Intelligence

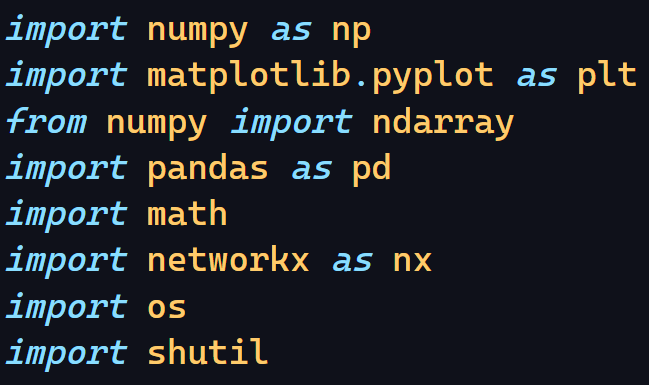
Clustering Techniques!

HOW TO RUN?

1. Open the folder in vsCode.
2. Go to main.py.
3. Click on run python file.

EXPLAINATION:

* Utility Libraries Used:

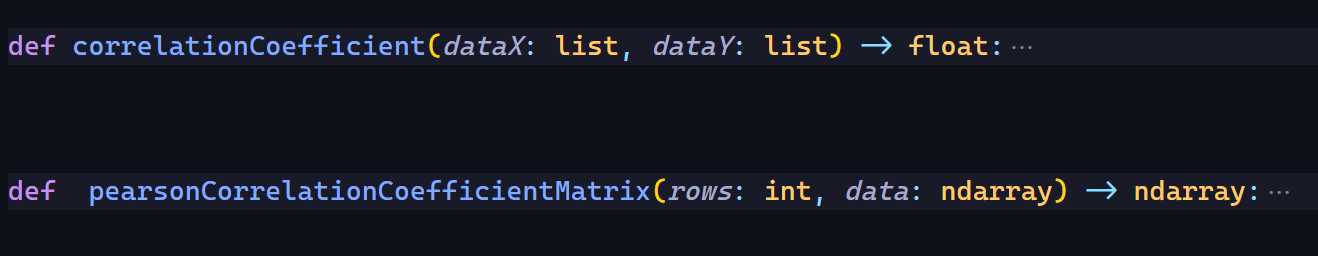


* Task 01:
* Functions For Read/Write from/to A Text File:
* Function read\_data reads data from a text file and returns a tuple containing (numberOfRows, numberOfColumns, Data).
* Function write\_data writes data from matrix to a text file including (numberOfRows, numberOfColumns, Data).

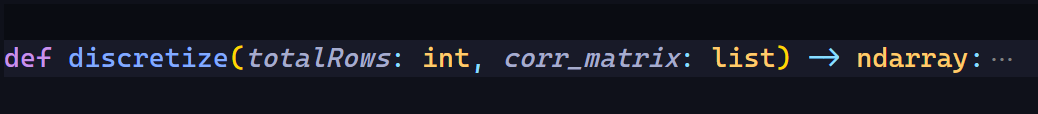
A picture containing text, screenshot, font

Description automatically generated

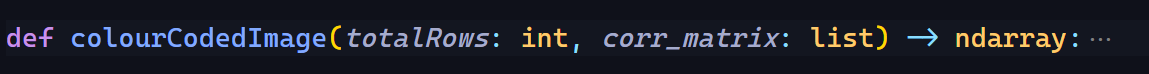
* Functions For Computing Pearson’s Correlation Coefficient:
* Function correlationCoefficient computes and returns the correlation value between two rows using Pearson’s correlation coefficient formula.
* Function pearsonCorrelationCoefficientMatrix computes and returns the correlation matrix of NxN dimension.



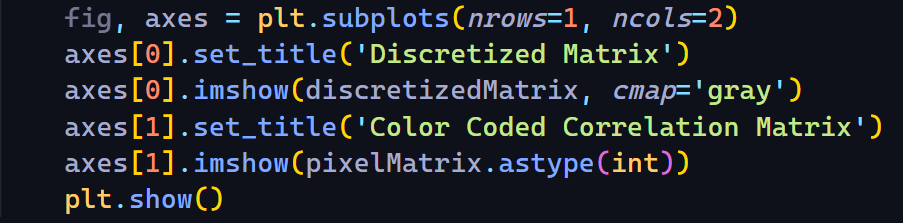
* Function For Computing Discretized Matrix:
* Function discretize computes and returns the discretize matrix from the correlation matrix data. If the value in the correlation matrix is ≥ mean of it’s respective column, then it’s replaced with 0 else it is kept 1. Here doing vice versa of what we should’ve made us closer to sample output.

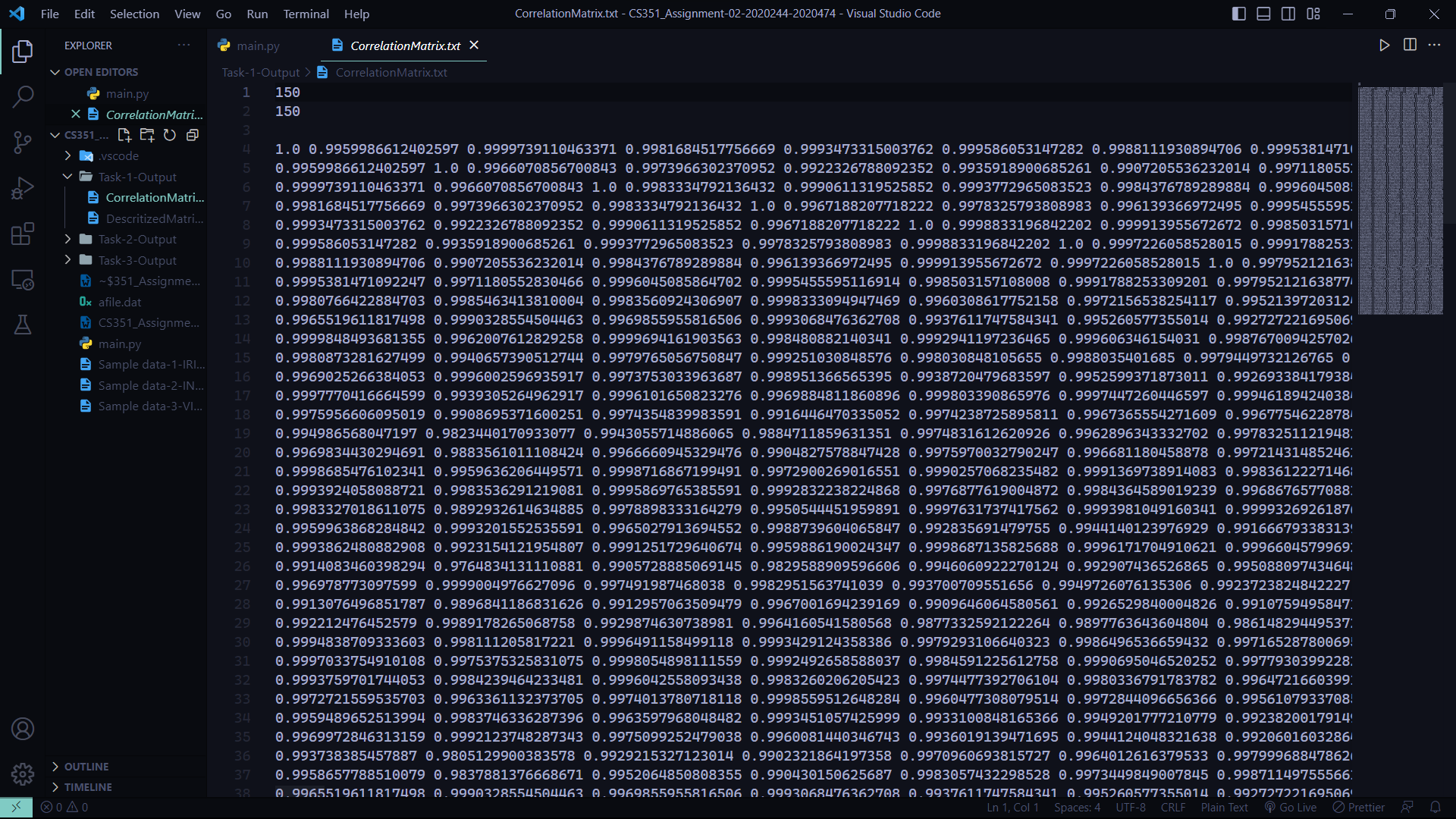


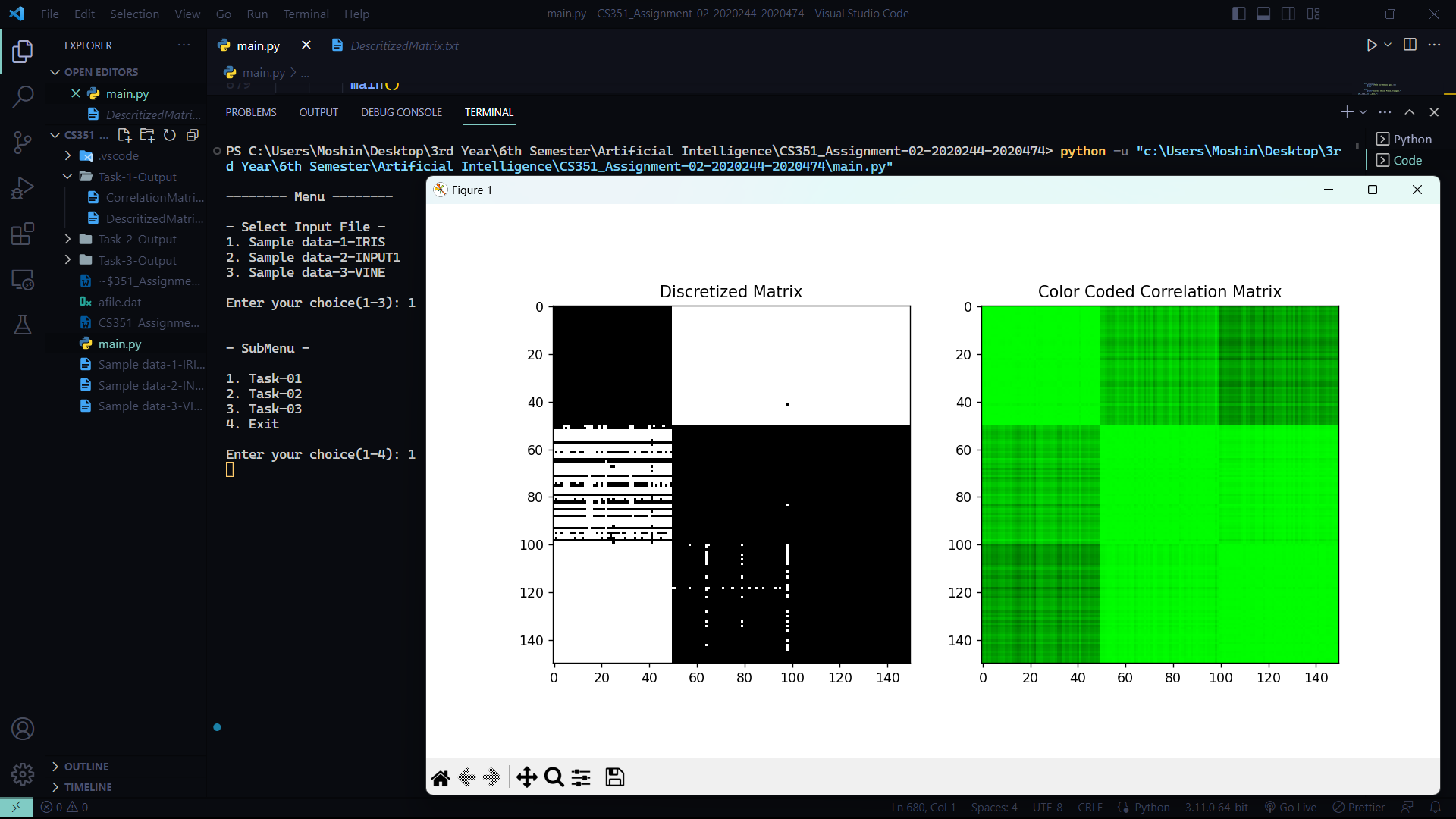
* Function For Computing Pixel Matrix for Color Coded Image:
* Function colourCodedImage computes and returns the Pixel Matrix of the correlation matrix data. It is calculated by dividing each element by maximum value of their respective column and then multiplying each value with 255. Then it’s placed in 3-D Matrix [R, G, B]. We compute the value of G here as we are required to print sample image in green shade and then place it like [0 G 0].



* Using Matplotlib For Visualization:

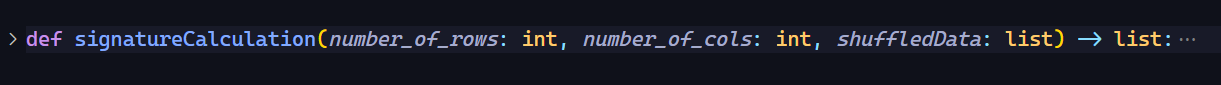


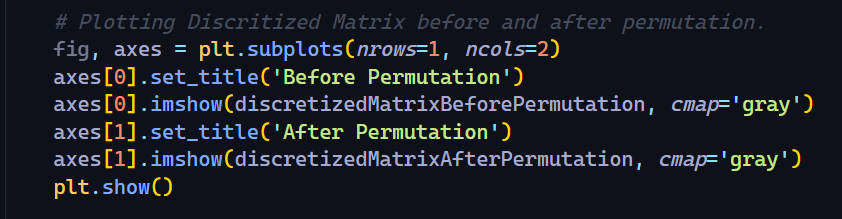
* Output Screenshots:
* Correlation Matrix:
* A screenshot of a computer screen

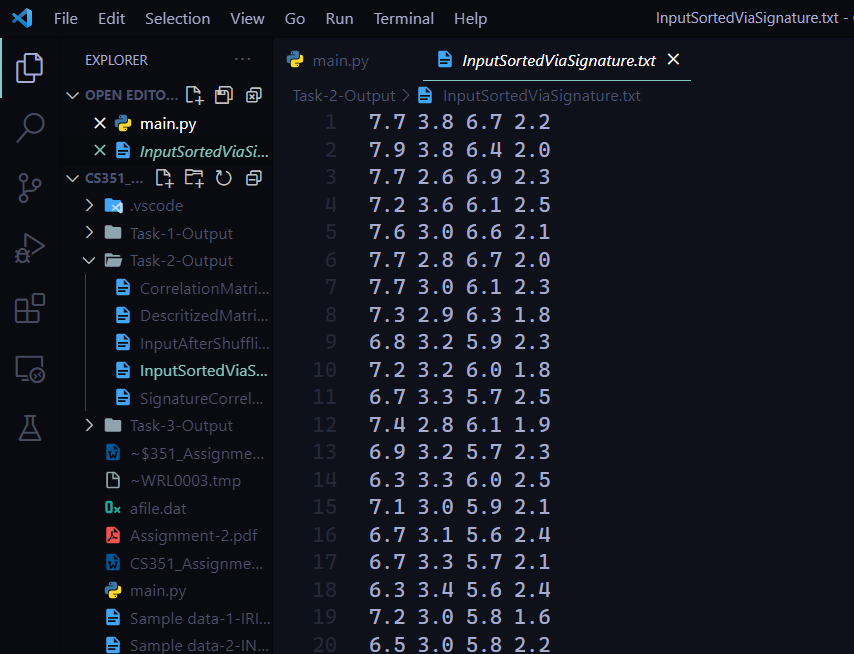
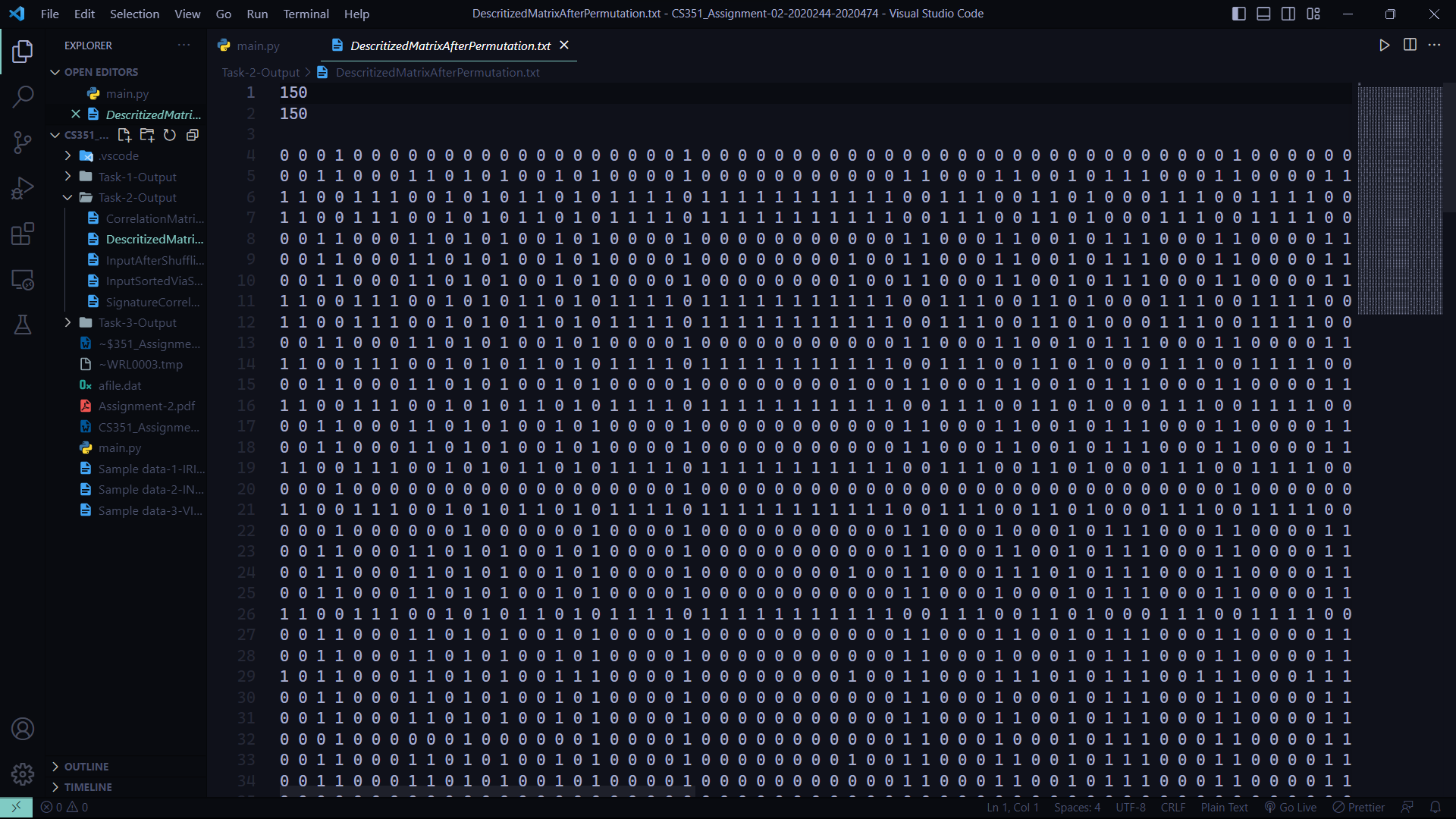
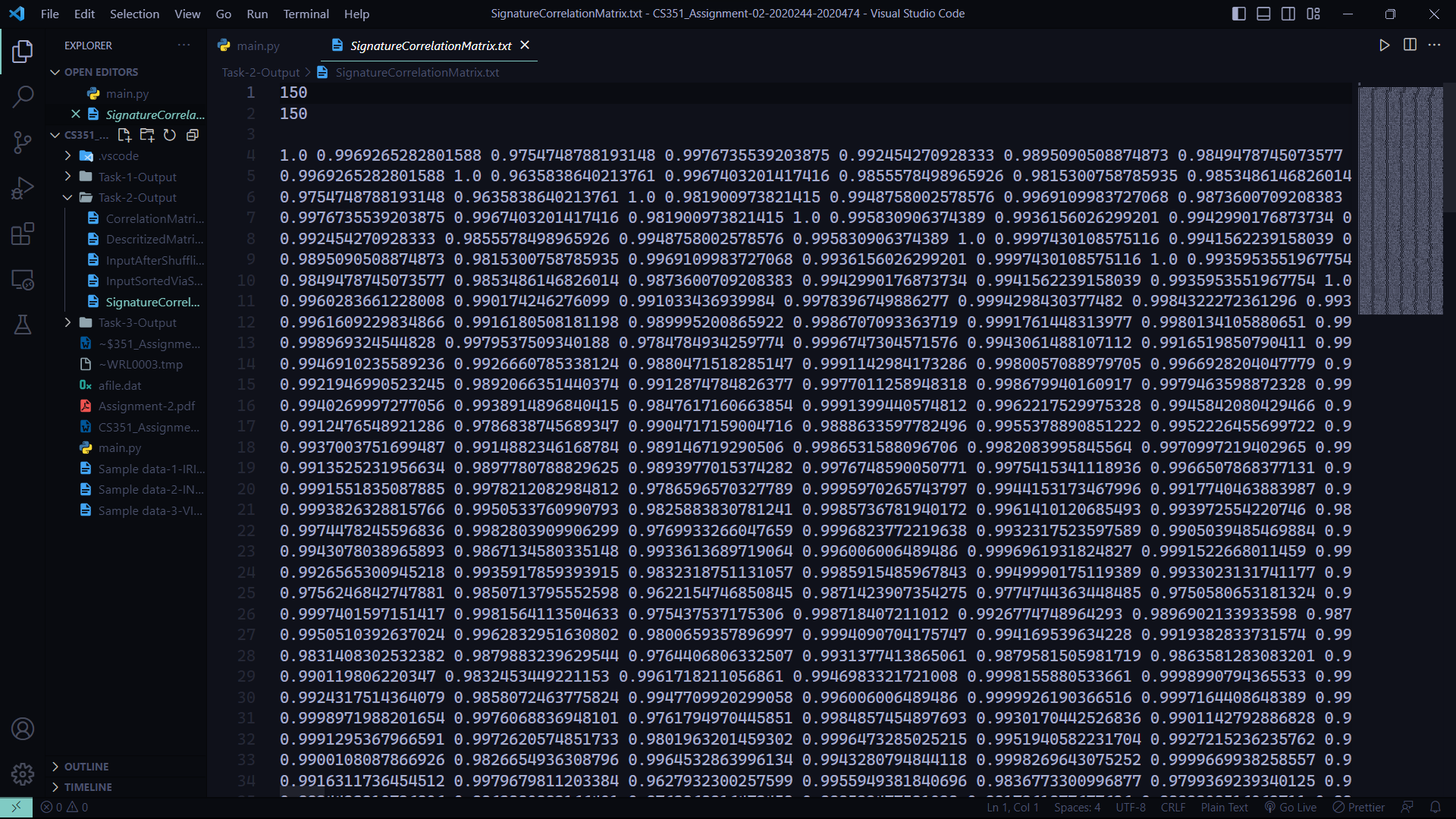
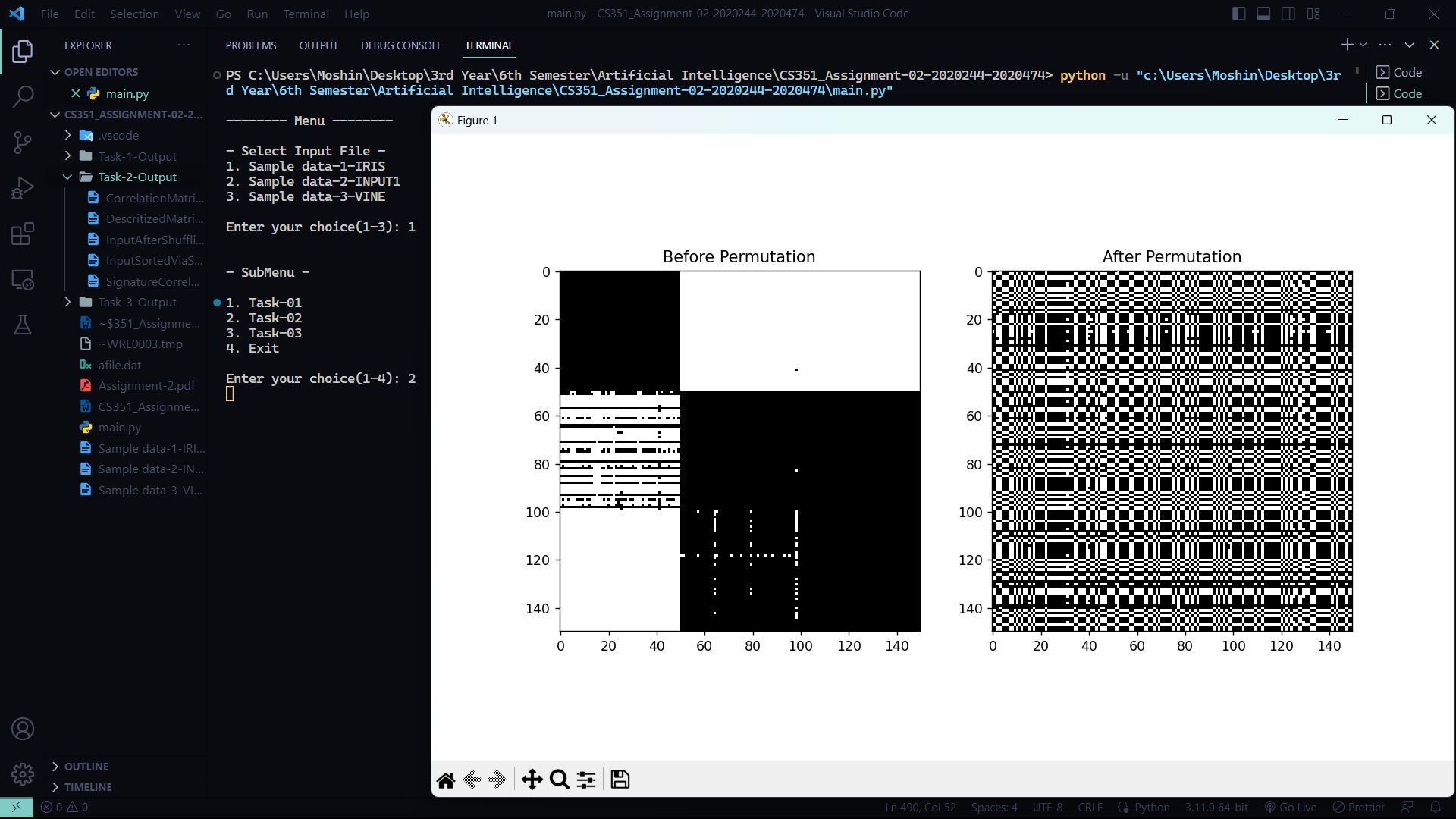
  Description automatically generated with medium confidenceDiscretized Matrix:
* Bitmap Visualization:
* Task 02:
* Function For Permutation/Shuffling:
* Function permutateMatrix provides constant shuffle over a period. It shuffles the rows of inputData. A random seed (50) is used to ensure that results are reproducible.

A picture containing font, text, screenshot, graphics

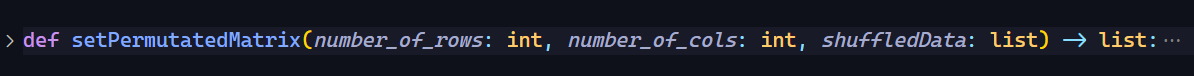
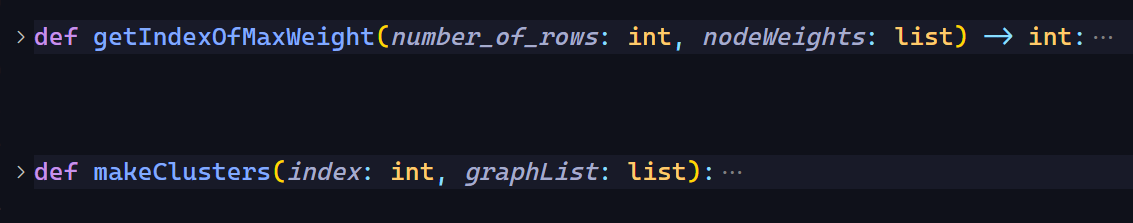
Description automatically generated

* Function For Calculating Signature Value and Sorting via Signature:
* Function signatureCalculation computes and returns sorted input data via their signature values. We passed the permutated data as input to this function, but I guess it really doesn’t matter here if we pass the non-permutated data, in the end it’s getting sorted via it’s signature values.
* A screen shot of a computer

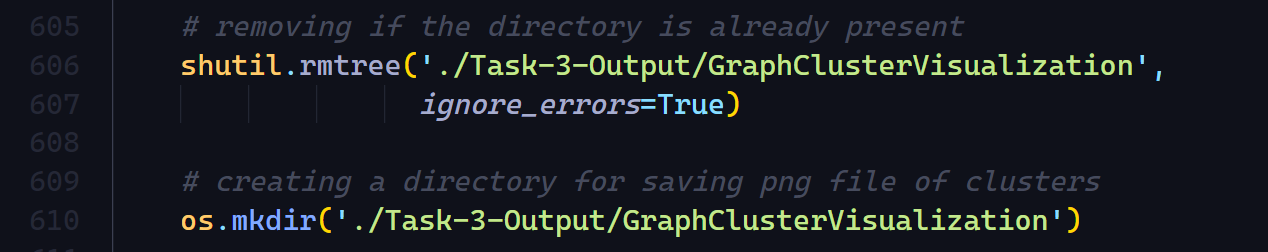
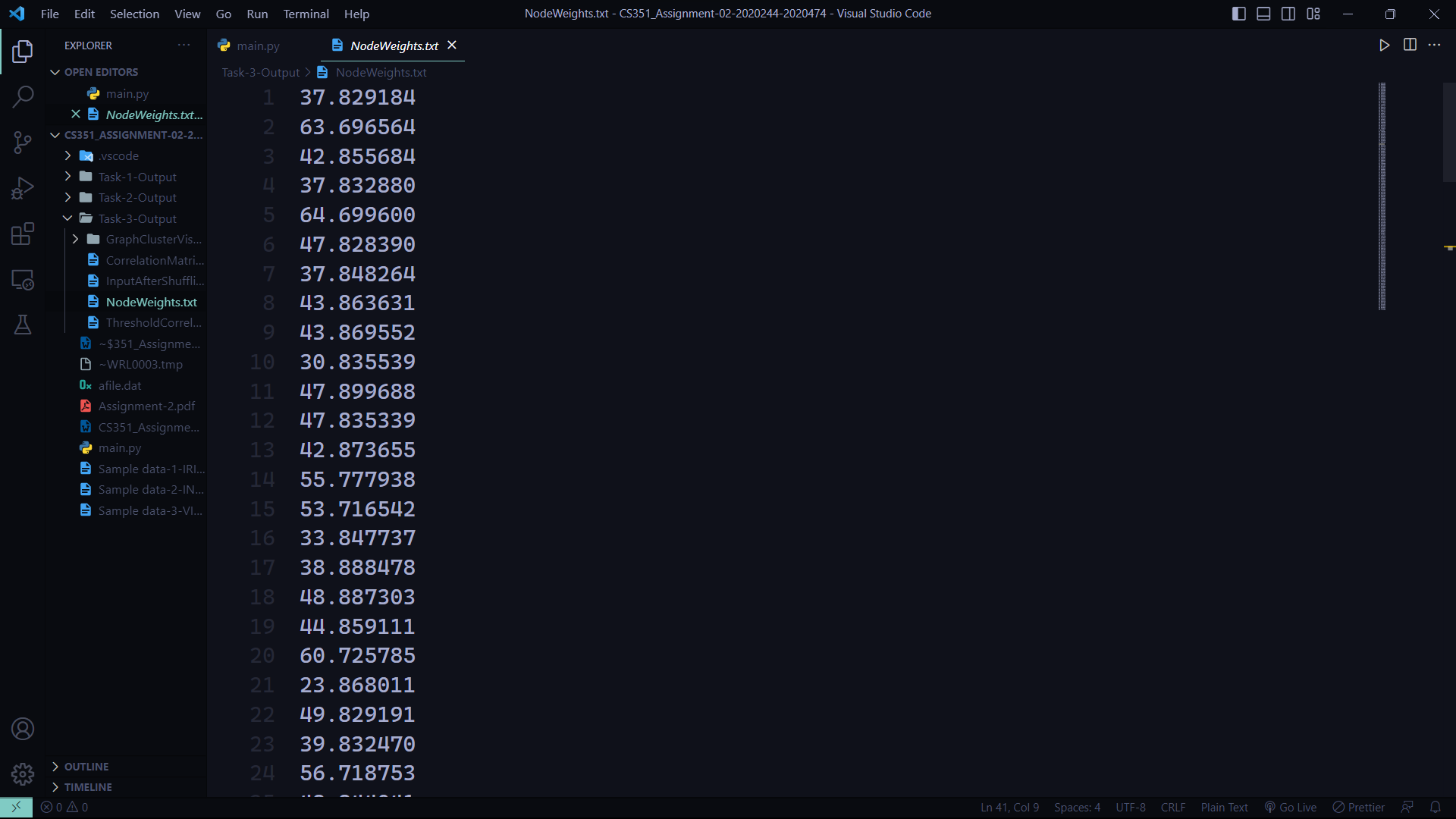
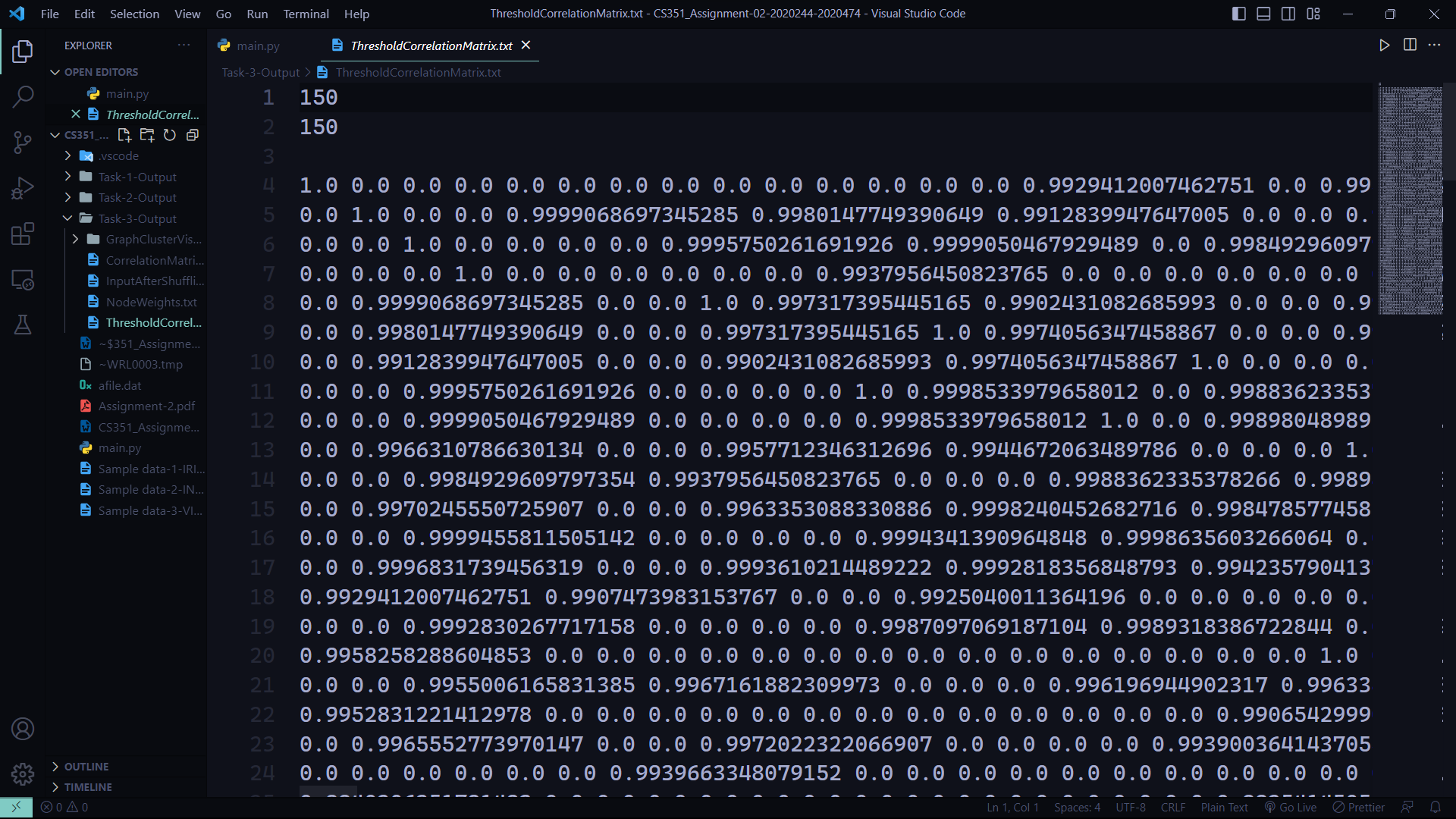
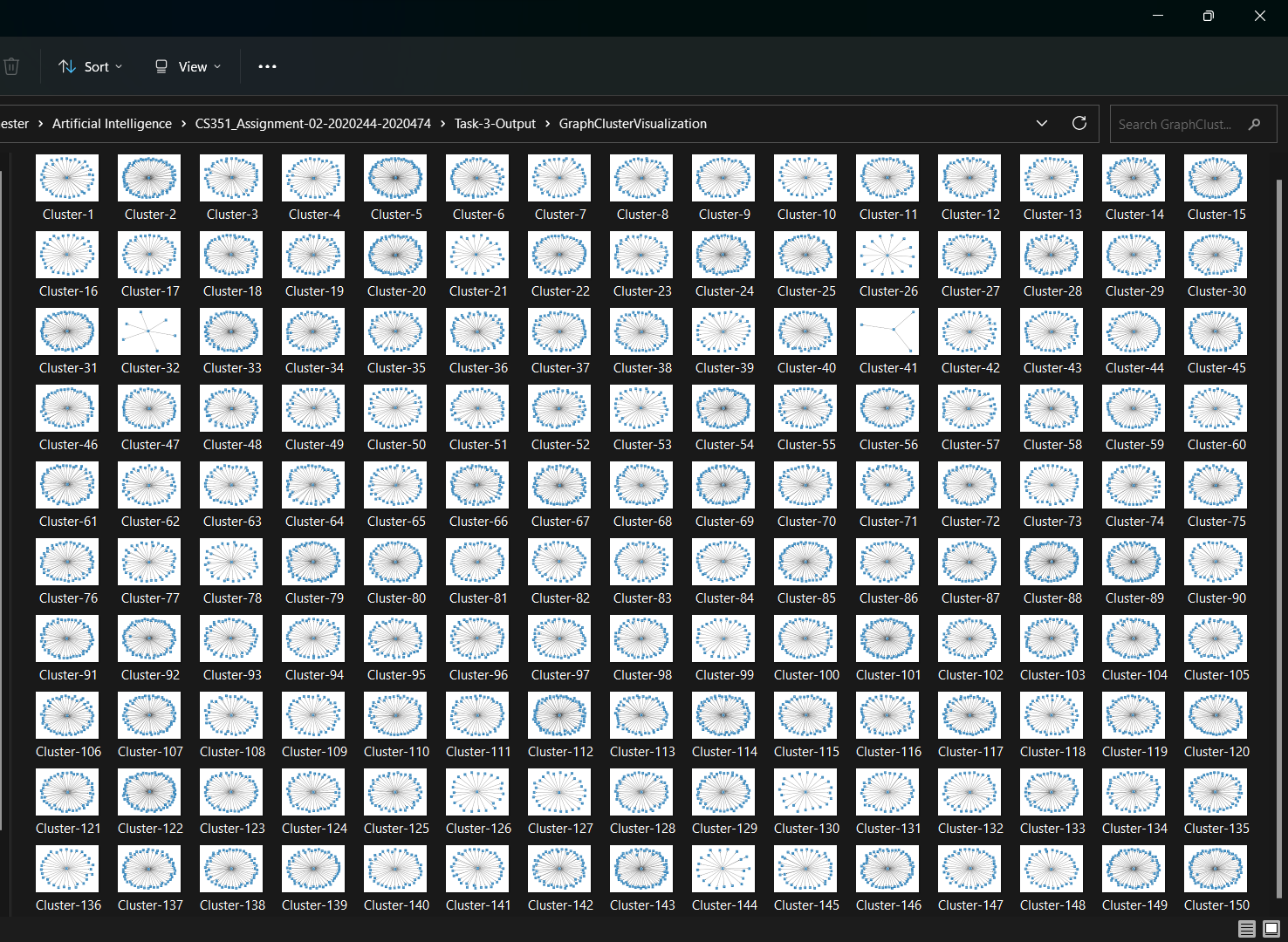
  Description automatically generated with low confidenceUsing Matplotlib For Visualization:
* Output Screenshots:
* A screenshot of a computer

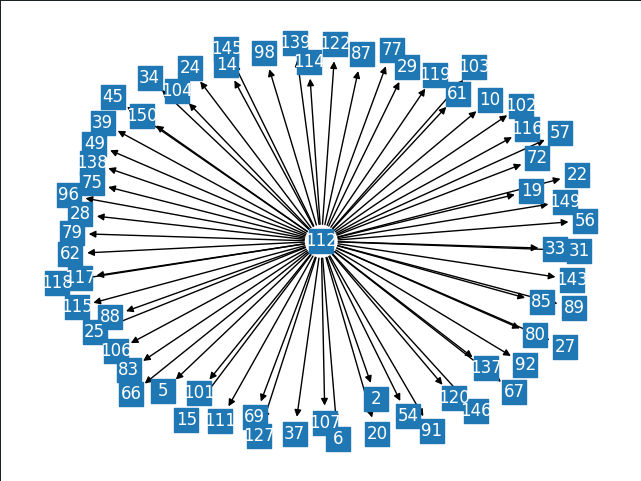
  Description automatically generatedInput after permutation:
* Input after sorting via signature:
* Discretized Matrix after permutation:
* ****Correlation Matrix after sorting via signature.
* Bitmap Visualization:

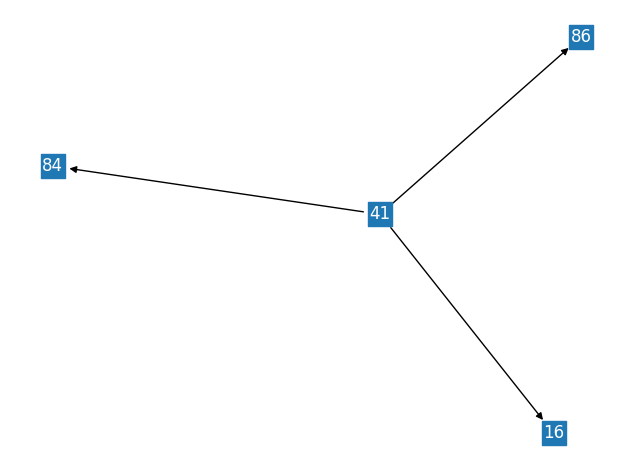


* Task 03:
* Function For Setup of Permutated Correlation Matrix:
* Function setPermutatedMatrix takes permutated data set’s correlation matrix and based on a threshold, it updates that matrix. It iterates through each value in the matrix and if the value is less than the threshold it makes it 0, else it is **not** updated.
* Functions For Computing Node Weights of the updated matrix with respect to Threshold:
* Function getNodeWeights computes and returns the nodeWeights matrix by calculating node weights of each row. It is calculated as the sum of all columns in a respective row. 
* Function For Making Cluster Of the Row in Correlation Matrix with maximum node weights:
* Function getIndexOfMaxWeight computes and returns the index of the node with maximum weightage. Once it is visited, matrix of node weights at that particular index is placed with -1 so that that node is excluded from our to-cluster list.
* Function makeCluster makes the graph of element with maximum node weight, it connects the edges to the elements in graphList that have survived the threshold test and then it connects to them and forms a cluster. Adding nodes and edges in between is done in this function.
* Function For Visualization of weighted graph in form of clusters:
* Function visualizeWeightedGraph takes input a weighted graph and index with maximum weight, it then used matplotlib to create a figure. Further it uses networkx library to draw the nodes and edges between source and destination. It can also add weights in between edges but it doesn’t look good when we are visualizing it. It saves the figure in png format.



* Creating a Directory for our Cluster images:
* Output Screenshots:
* Node Weights Matrix:
  + The threshold is set to 0.99.
* Threshold Correlation Matrix:
* Directory of png’s overview/Clusters saved:
* Most dense and least dense clusters formed:
  + Most dense cluster:



* + Least dense cluster:
* Comparison Of Task 2 and 3:
* Task-2:
* The primary goal of task 2 is to demonstrate the dissimilarity in bitmap by carrying out permutations on the data and presenting the bitmap of the discretized iris data before and after the permutation process. Subsequently, the permutated data undergoes sorting based on its signature values, and a color-coded image of the correlation data is generated and displayed.
* Task-3:
* The primary objective of task 3 is to identify clusters within the correlation matrix subsequent to the permutation of data. To accomplish this, a threshold value is established, which in our case is 0.99, and the correlation matrix is updated accordingly. Following this, we pinpoint the node with the highest weight and construct a cluster including its surviving neighbors from the threshold process. This process is repeated iteratively until clusters are generated for each row in our graph list. Ultimately, this technique enables us to effectively determine the clusters present within the correlation matrix.
* Work distribution among group members:

Throughout the completion of this assignment, it is evident that we have made substantial contributions towards its successful completion. From the initiation of the project to the final submission, we both have actively participated in the task allocation, research, and implementation phases. Additionally, the collaborative nature of the project enabled both of us to offer valuable insights and suggestions, leading to the development of a comprehensive and well-structured final submission. We both have played an integral role throughout the assignment, and combined efforts have resulted in a successful outcome.

* Insights:

During the completion of this assignment, we have gained several valuable insights. Firstly, we discovered that the Pearson correlation matrix, although correlation is always symmetric process, didn’t produce symmetric bitmap for discretized data. While initially considering that we may have made an error in our formula, using the predefined pd.DataFrame.corr() function produced results that were consistent with our output. Despite this, doubts remained about the accuracy of the sample image produced by us. It should also be noted that discretized data is calculated by taking the mean of the column and updating each value in that column. I think that it’s the reason why the bitmap is not looking symmetric. Another observation made was that sorting via signature sorts both the input data and shuffled data to the same matrix, rendering the shuffling process is unnecessary here. Finally, we recognized the immense power of the Python language. Tasks that would have taken a significant amount of time in other programming languages such as C++, such as bitmap visualization (I first tried visualization using QT after completing these tasks in C++ and it was becoming too complex☹), were easily accomplished using Matplotlib in just two lines of code in Python.

The End .