**Group Members:**  Jaswanth Das Gunturu, Prakhar Gupta, Sudhanshu Shekhar.

**Objective:**

Maximize the total fare revenue collected by our car.

**Datasets:**

A number of network datasets were provided. Each dataset is meant to represent a distinct city or metro area. For each network dataset, there several random instances were generated.

**Decision variables for us to code:**

* Each time a ride becomes available and the car is close enough to the origin vertex to be offered the ride, a decision must be made to accept or reject the ride.
* Each time a car becomes idle at the end of a ride, a decision about relocating towards a hire demand area must be made.

**Factors we are taking into consideration:**

1. Driver Density
2. Probability of getting a ride( Probability capture)
3. Average Fare ($)
4. Distance (in time)

Driver density is the density of drivers in a particular area. The higher the density, the lesser chance it is for our driver to get a ride in that area.

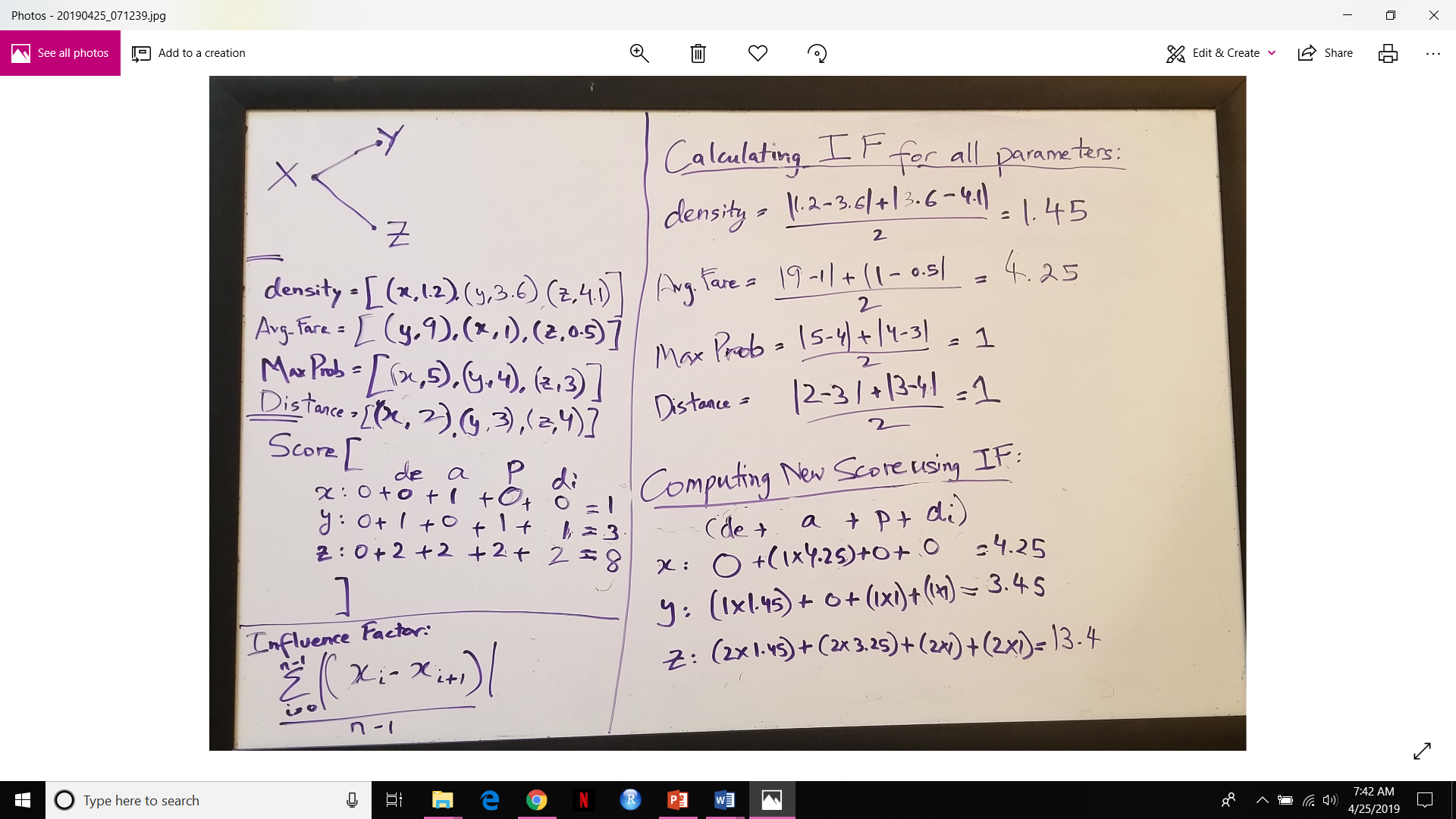
Probability of getting a ride is self-explanatory, and the higher it is, the better for our driver. It only increases the chances of getting a ride from a particular node.

Average Fare gives us the fare the driver will earn if he decides to accept a ride. The higher it is, the better for us.

Distance tells us the how far (in time) our driver is from the next pick-up location. Hence, the lower it is, the better.

We did try using Radial and Uniform Random distribution. By using the Radial distribution, we got a very low average fare revenue. The results have been shown below at the end of the report for the radial distribution. The uniform random distribution was faring much better, the average fare revenue was much higher, and hence we decided to go with the uniform random distribution.

**Logic behind Relocation function:**

  
\*There is a mistake in the above image, the new score calculated using Influence Factor for node Z=15.4, not 13.4.

From the image above, we can explain the logic step by step.

Suppose that the driver is at node X and his neighboring nodes are Y and Z.

In this function, we are looking to relocate to the best possible neighboring node or choose to stay at the same node the driver ends the previous trip on. We choose the best relocation node based on the factor’s: Driver density, Average fare, Probability capture and Distance.

We create a dictionary for all the four variables we are going to base the relocation on, which are mentioned above. After this, we create a variable called option for relocation in which we call the keys of the graph which has been generated.

After this, we start adding the key items from the keys of the graph G to specific dictionaries that we made, like the following for all the four variables:

* for item in optionsForRelocation:

dirverDensityAtLocations[item] = driverDensity[item]

* for item in optionsForRelocation:

maxPickUpProbabilty[item] = probabilityCapture(currLoc, item, G)

* for item in optionsForRelocation:

my\_AvgFareMultiplier[item] = avgFareMultiplier[item]

* for item in optionsForRelocation:

my\_distance[item] = G[currLoc[0]][item]

Now, we have 4 dictionaries with its respective nodes and values.

Next, we realize that the lesser the driver density, it is better for our driver as his chances of getting a ride increases. Similarly, the higher the pick up probability, the better for our driver. The higher the Average fare multiplier, the better for the driver and the lesser the distance, the better for the driver.

Hence, we sort the dictionaries in this way:

Driver density: Ascending

Pick up Probability: Descending

Avg. Fare Multiplier: Descending

Distance: Ascending

Now, we have sorted the four dictionaries. What this does is that it puts all the best nodes and values on the left side of the dictionary. Meaning, ideally, we should select the nodes that are on the left-most of the dictionary.

Now we have to choose the best possible node to relocate to.

Initially, we tried to calculate the score of each node, based on their indexes in the sorted array. Now referring back to the image shown above,

Score: Here, the first 0 is an initialization, it is common for all the nodes.

X: 0 + 0 + 1 + 0 + 0 = 1   
Y: 0 + 1 + 0 + 1 + 1 = 3  
Z: 0 + 2 + 2 + 2 + 2 = 8

Since we have sorted the arrays such that the least indices are the best option to relocate to, we calculated the score above by adding the index of each node, X, Y and Z in the sorted dictionaries of Driver Density, Average Fare, Probability Capture (pick-up) and Distance(in time).

From that we can see that the node X has the least score, so the driver should stay at node X according to this logic. The least score is the best node to relocate to because we have sorted the dictionaries in a way that the least indices are the best options, so the sum of the indices, the lesser will be the better.

The problem with this solution was that the high difference in fare and driver density was not being accounted for. So, we tried to accommodate the differences in the values as well to determine the score. This was done by introducing a factor called the influence factor. The influence factor is computed as shown in the image above. What the influence factor does is, it takes into consideration the difference between the values of each dictionary and gives a weight to the four factors.

Now, we will calculate the new score of the nodes using the influence factor as well.

So, by doing that, we get the new scores as:

X: 0 + 0 + (1\*4.25) + (0\*1) +( 0\*1) = 4.25   
Y: 0 + (1\*1.45) + (0\*4.25) + (1\*1) + (1\*1) = 3.45  
Z: 0 + (2\*1.45) + (2\*4.25) + (2\*1) + (2\*1) = 15.4

From the new scores, we can see that Y has a lesser score than X. This shows that the difference in fare of Y and X, Y being $9 and X being $1 is being accounted for and the logic suggests that despite X having a lower driver density and lower distance and higher probability, because the difference in fare is so high, we should relocate to Y, instead of X.

**Logic behind Ride Decision function:**

For this function, we had to decide when to accept and when to reject a ride. Since the instances are generated randomly and are not realistic values, there might be many cases where a particular ride would be very long but would not offer a justified fare for that when compared to a ride of a similar duration but with a much higher fare.

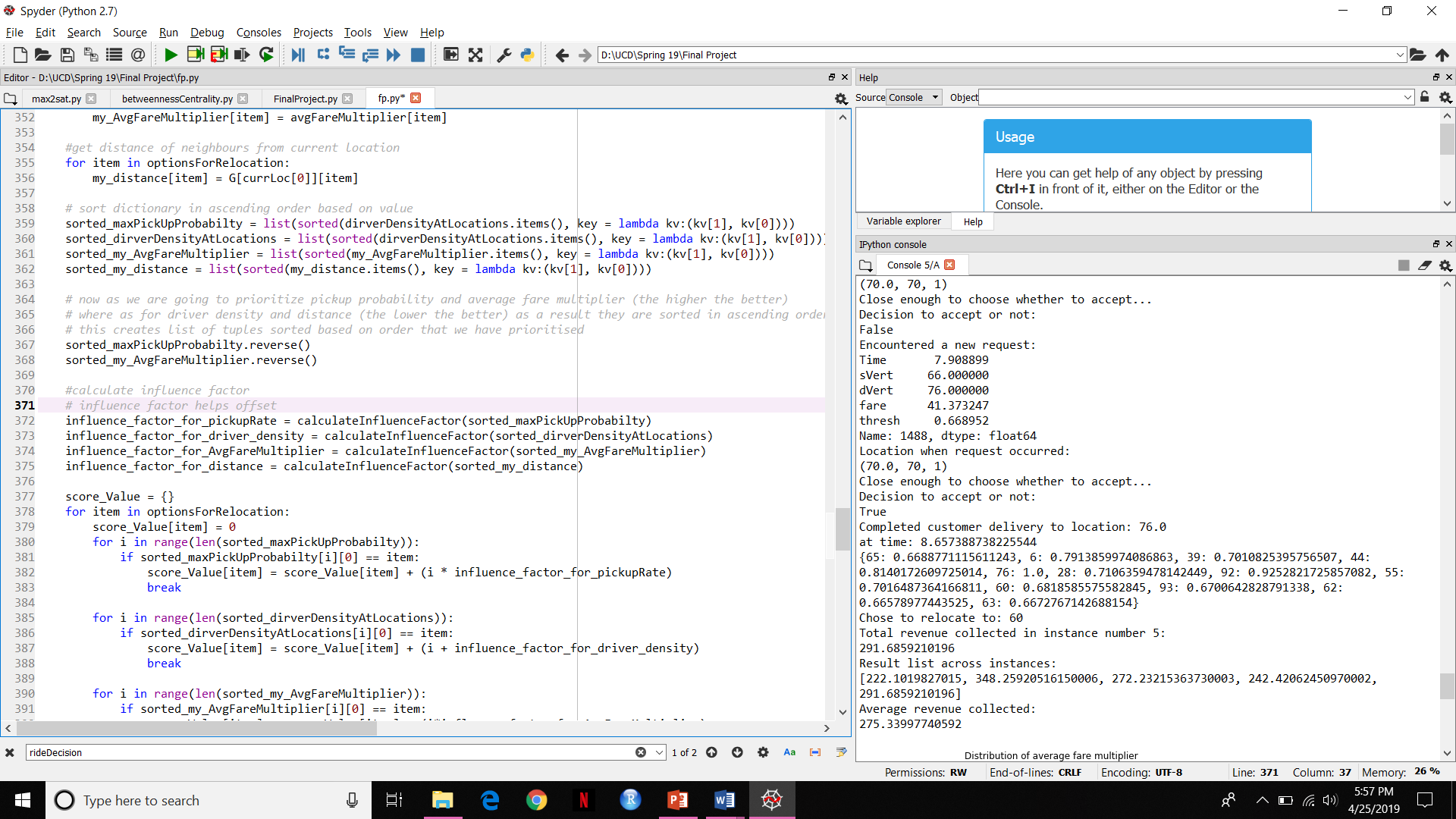
Hence, we should not always accept a ride because there are enough rides being offered by the instances and we can afford to reject a few rides.

For this function, we chose to make a threshold on the fare. So, we chose $35 as the threshold of fare. What this means is that whenever the ride offered is less than $35, we will always reject the ride. But when the ride offered is more than $35, we will accept it.

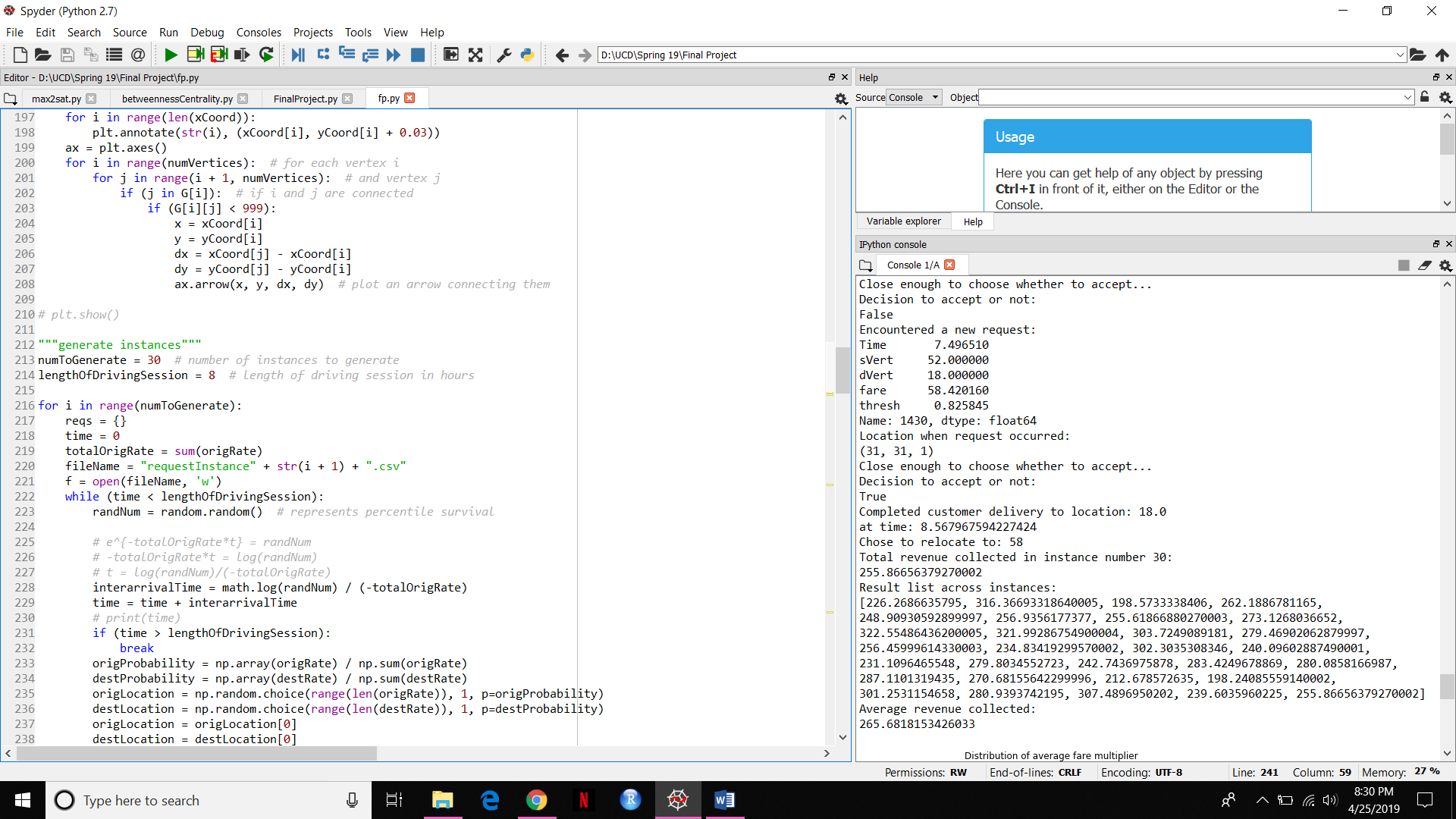
This improved our model and revenue considerably. We had a revenue of around $100 when we were accepting all the rides by default, but after putting this threshold, we had a jump in revenue from $100 to around $260-270 for 8 hours of driving over 5 instances.

**Sample Output:**

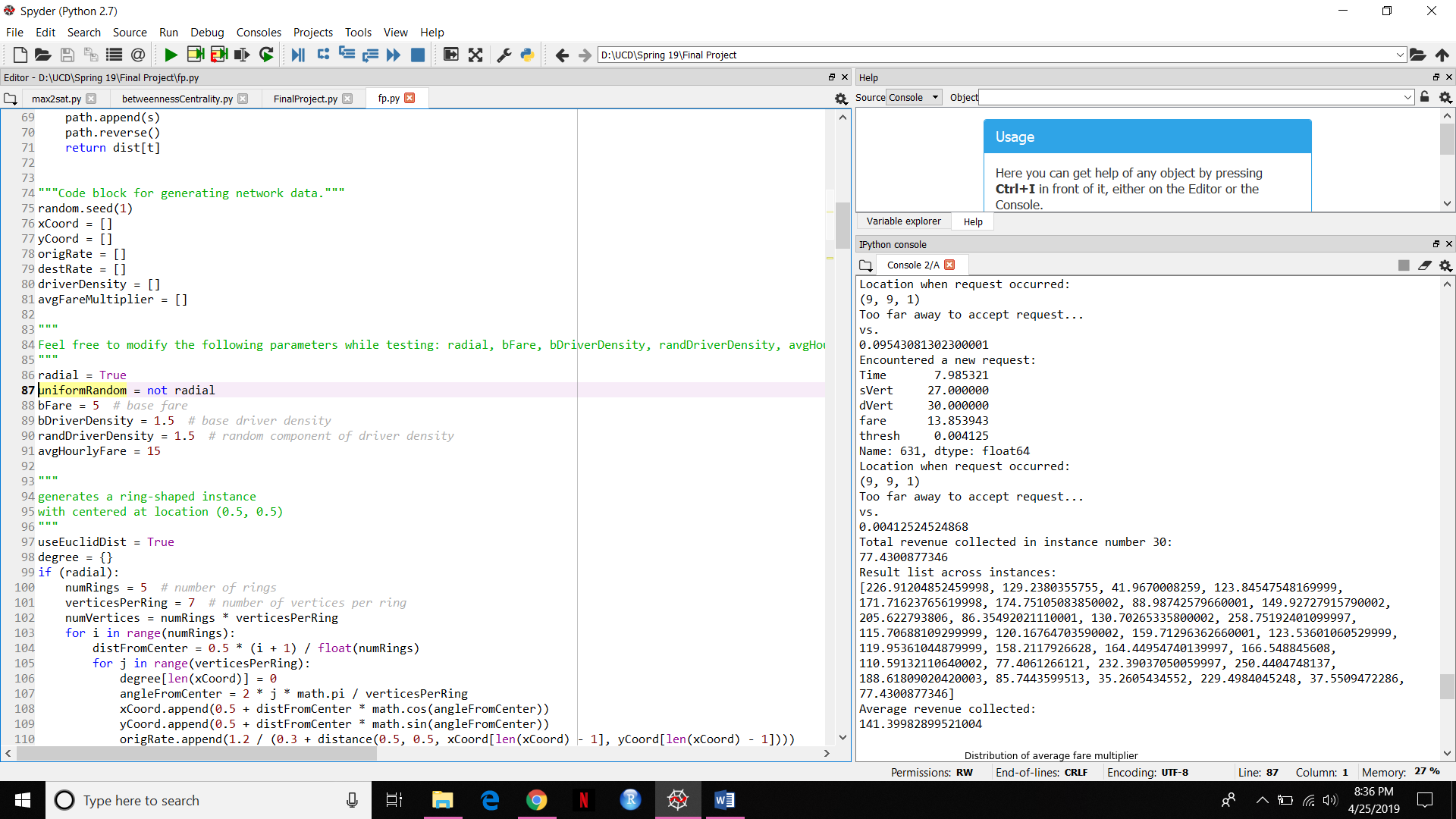
**(Not Radial)For 5 instances and 8 hours:** Average Revenue collected from 5 instances was $275.34



**(Not radial) For 30 instances and 8 hours:** Average Revenue collected from 30 instances was $265.68



**(Radial) For 30 instances and 8 hours:** Average Revenue collected from 30 instances was $141.39



**(Radial) For 5 instances and 8 hours:** Average Revenue collected from 30 instances was $195.49

