**Report of my AdvisorBot**

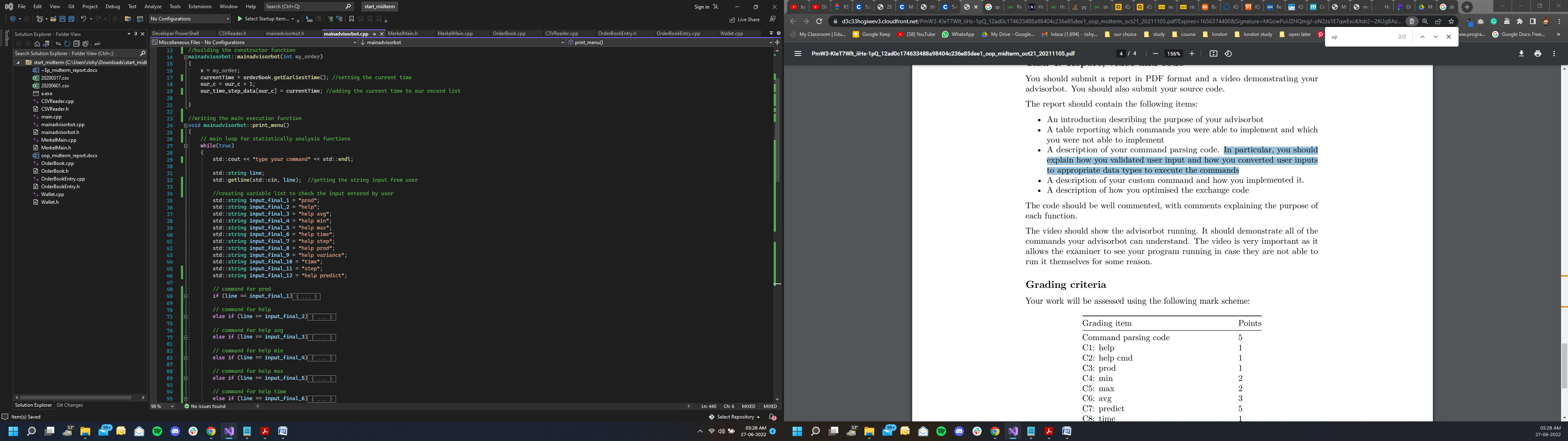
Introduction:

This AdvisorBot is programmed in C++ and it is build on top of the merklerex platform, I have added two more files in it (mainadvisorbot.cpp & mainadvisorbot.h) and integrated it in the MerkelMain.cpp file. The main purpose of this AdvisorBot is to perform statistically analysis for the user. Initially we can only trade on this platform but now a user can also calculate statistical function like average, variance, minimum value or maximum value at a particular time step. And apart from that AdvisorBot can also perform predictions for the next time step (EX: predict max ETH/BTC ask). Through our platform user can not only buy/sell products but also perform some good analysis in between.

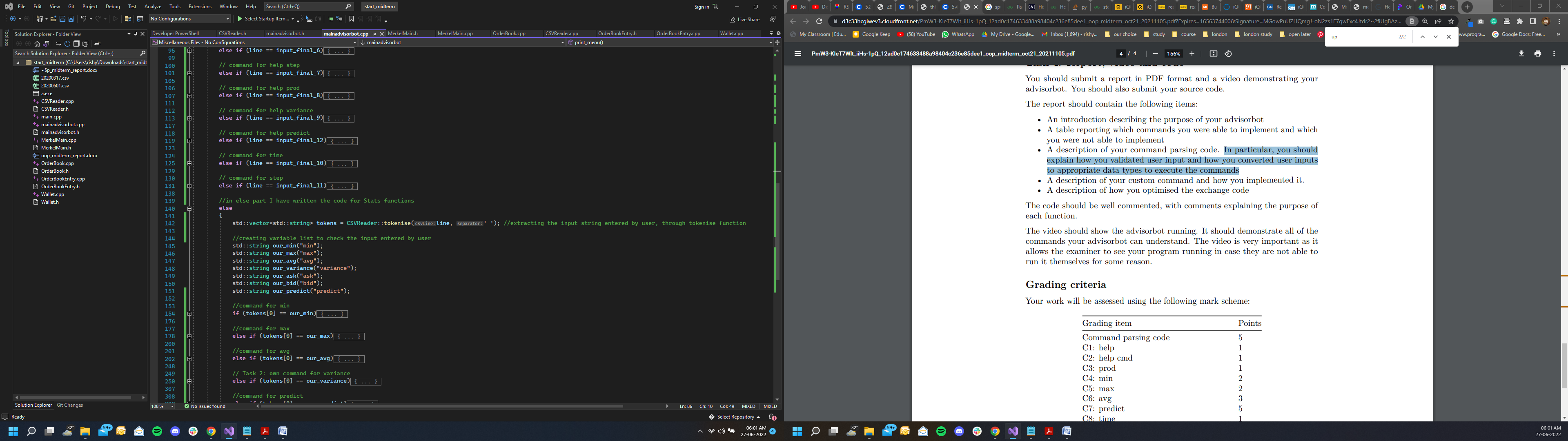
Table reporting how many commands I was successfully able to execute:



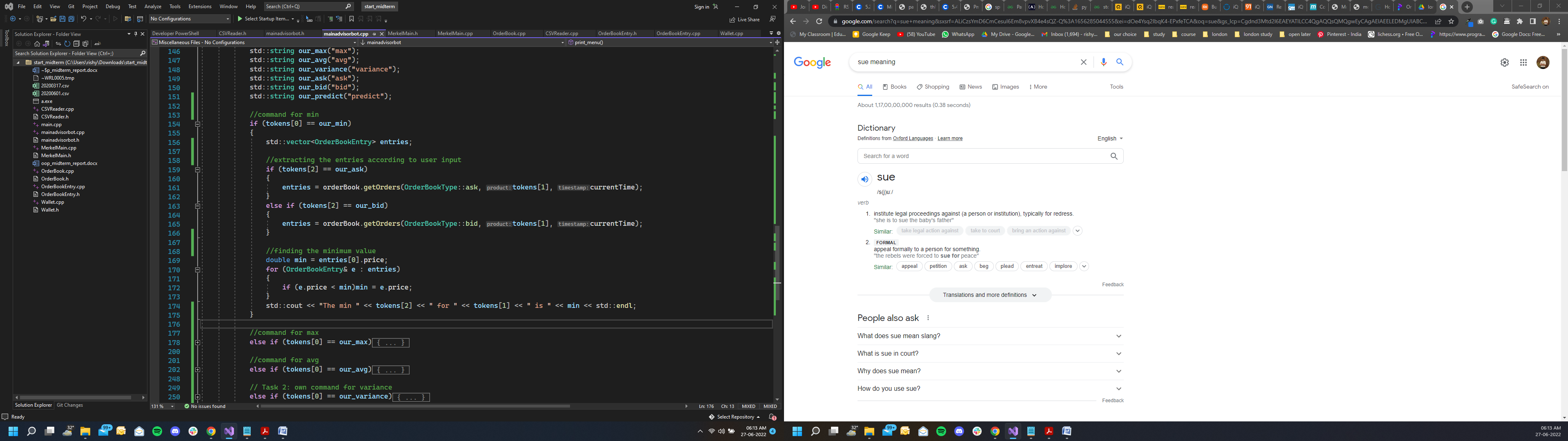
Command parsing code description:



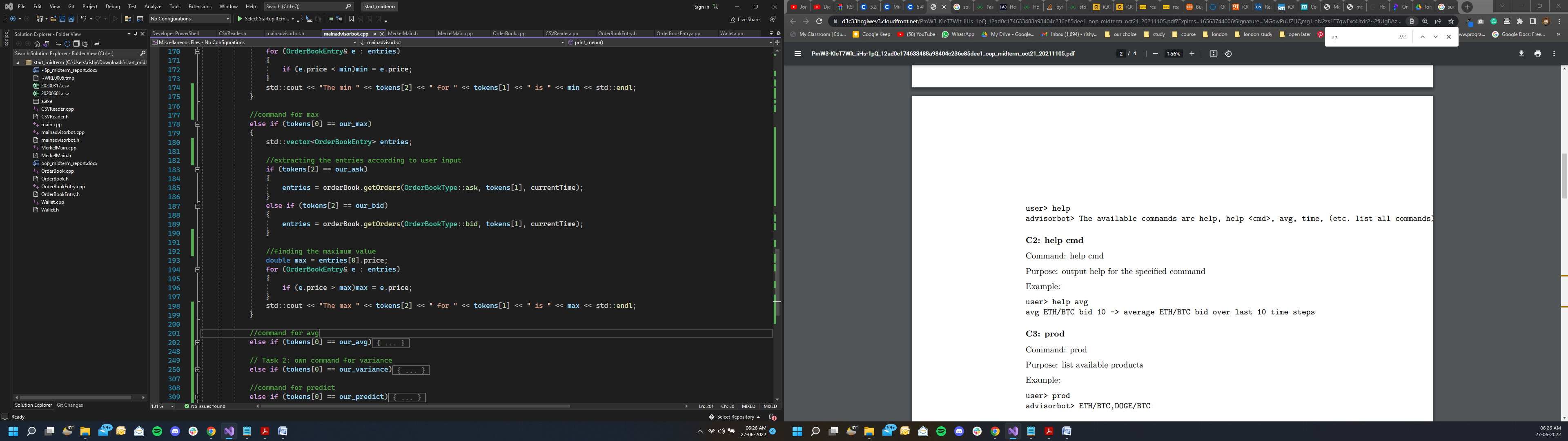
As we can see in the above image the very first thing is, I’m requesting a string of words from user through getline function in print\_menu function.

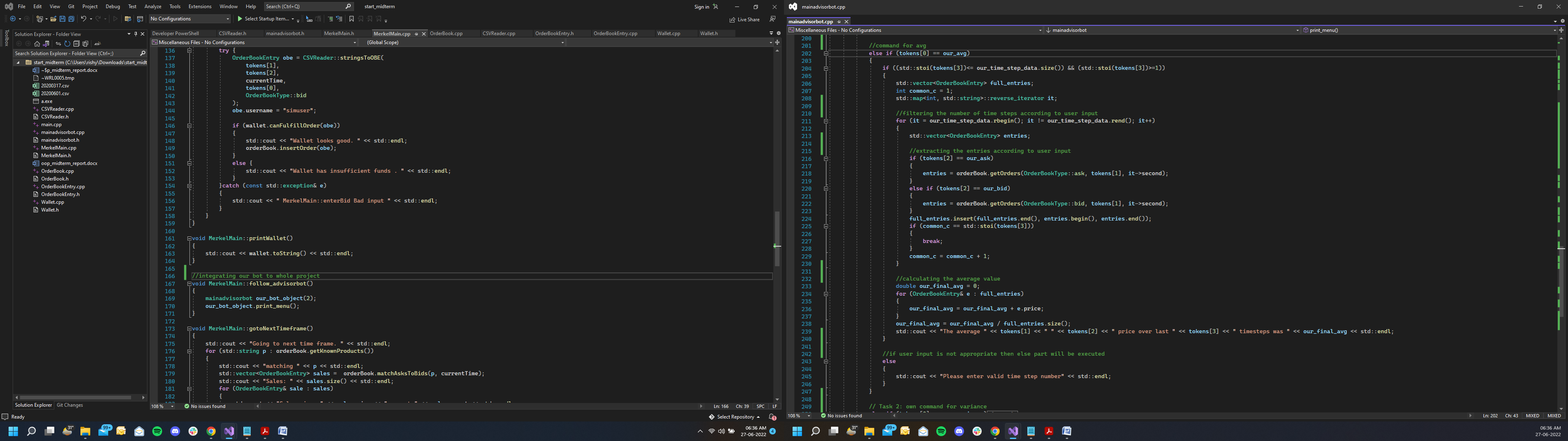


After that I’m using tokenise function from CSV reader file to add the individual word into our vector. Then we are comparing our vector elements to the appropriate keyword to execute a particular statistical function.

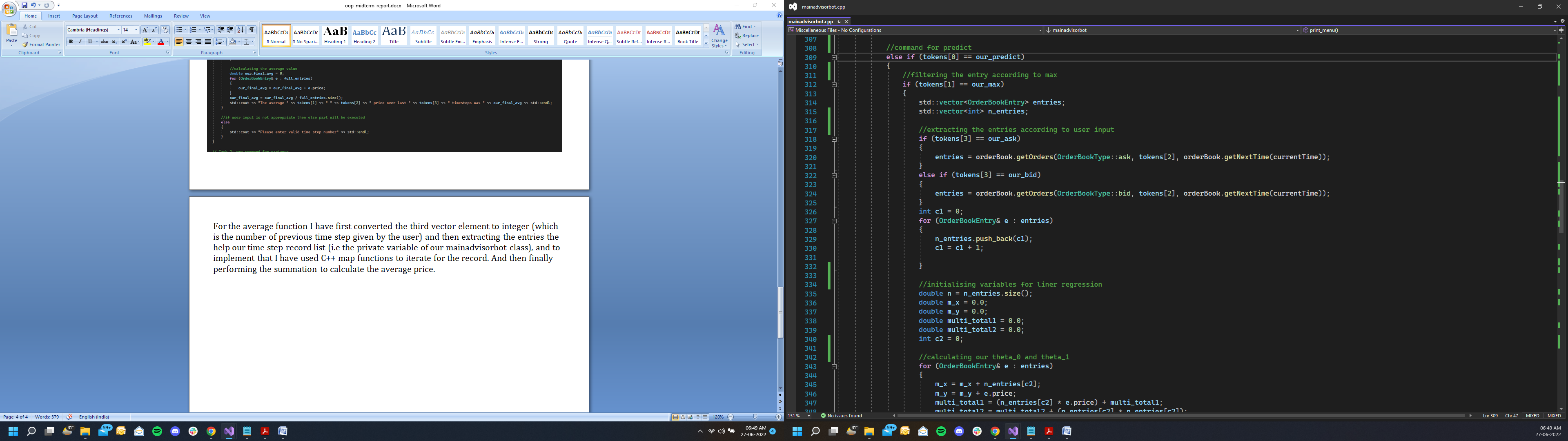


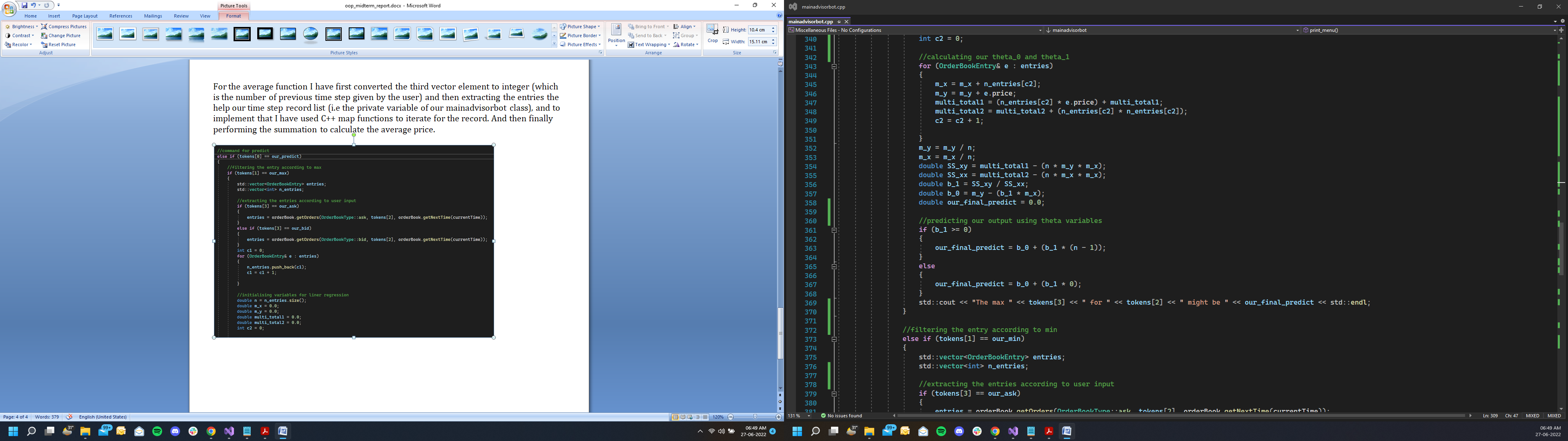
For example if the first element of our vector is min, then I have written the program to calculate the min, which starts by extracting entries through get order function from order book file (as we can notice in the above image I have filtered our get order query through vector elements). Once we have extracted the entries, we can iterate over it to get the minimum price, and finally writing the console output statement.



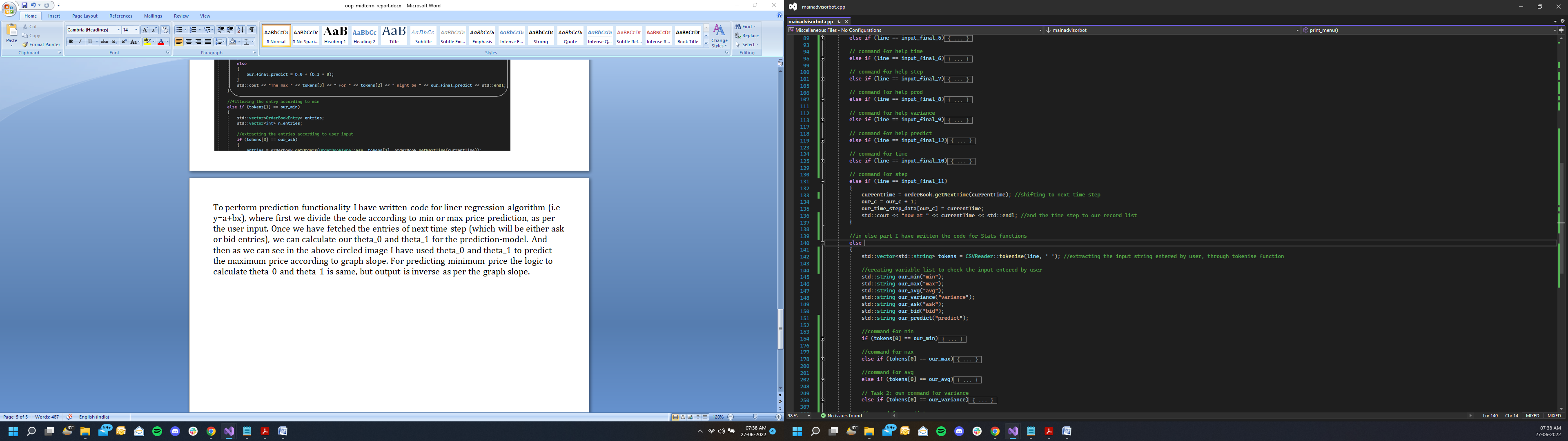
Just below the min function I have written max function, which is more or less similar to min function logic. However, in the end our for loop logic is just inverse, to find the max price.

For the average function I have first converted the third vector element to integer (which is the number of previous time step given by the user) and then extracting the entries the help our time step record list (i.e the private variable of our mainadvisorbot class), and to implement that I have used C++ map functions to iterate for the record. And then finally performing the summation to calculate the average price.

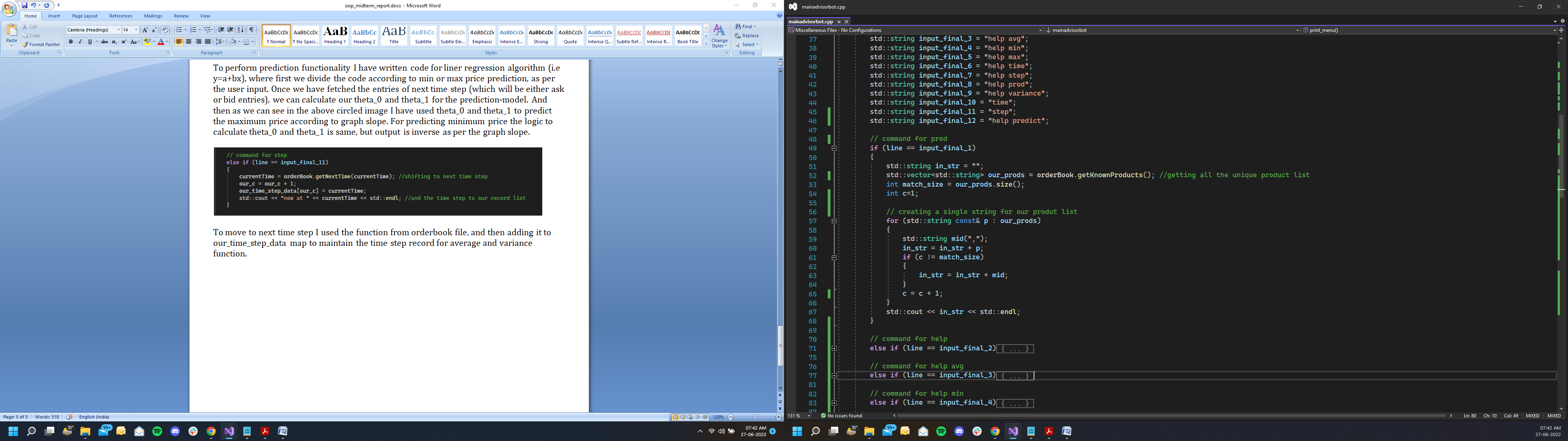




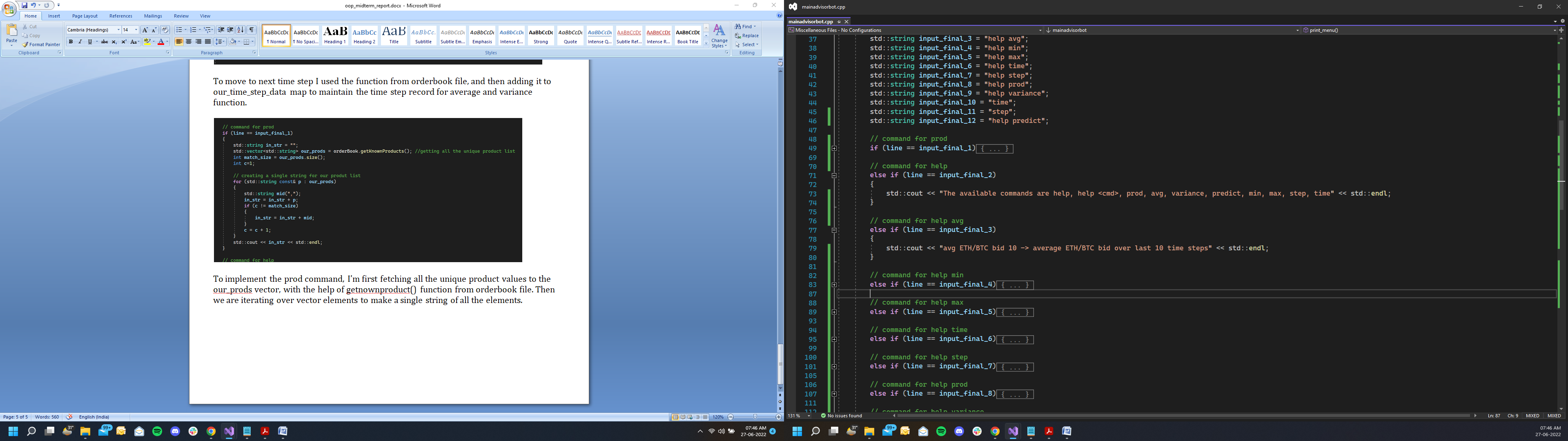
To perform prediction functionality I have written code for liner regression algorithm (i.e y=a+bx), where first we divide the code according to min or max price prediction, as per the user input. Once we have fetched the entries of next time step (which will be either ask or bid entries), we can calculate our theta\_0 and theta\_1 for the prediction-model. And then as we can see in the above circled image I have used theta\_0 and theta\_1 to predict the maximum price according to graph slope. For predicting minimum price the logic to calculate theta\_0 and theta\_1 is same, but output is inverse as per the graph slope.



To move to next time step I used the function from orderbook file, and then adding it to our\_time\_step\_data map to maintain the time step record for average and variance function.

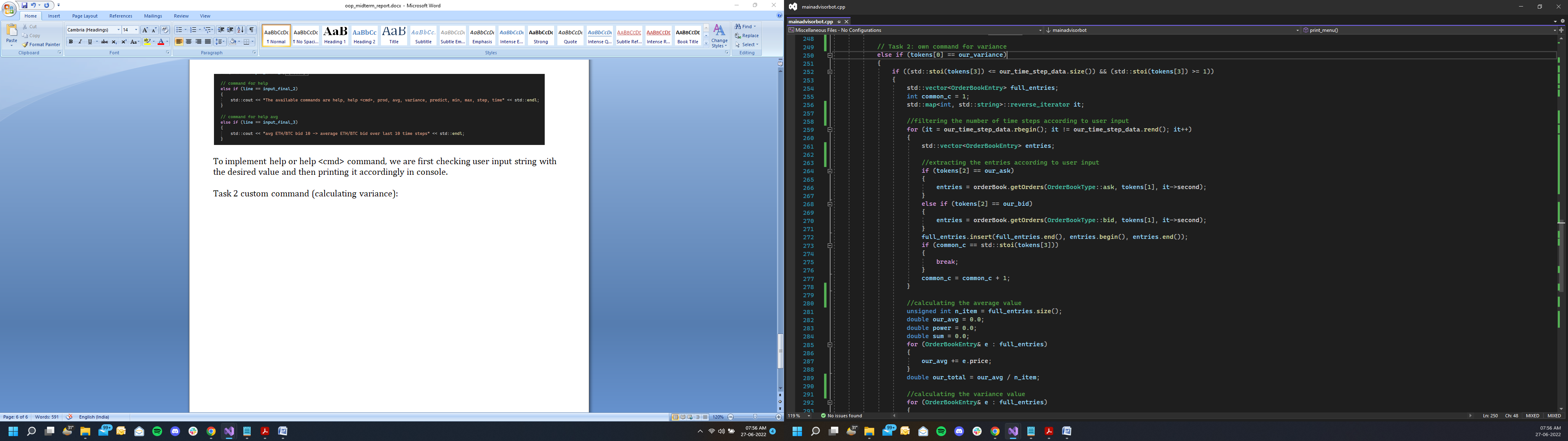
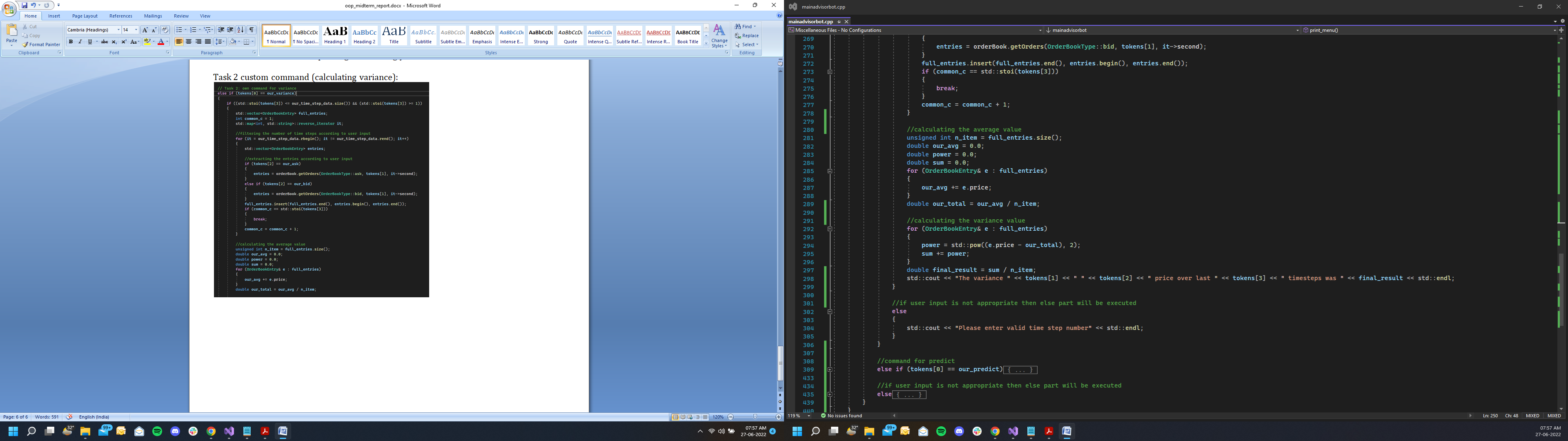


To implement the prod command, I’m first fetching all the unique product values to the our\_prods vector, with the help of getnownproduct() function from orderbook file. Then we are iterating over vector elements to make a single string of all the elements.



To implement help or help <cmd> command, we are first checking user input string with the desired value and then printing it accordingly in console.

Task 2 custom command (calculating variance):



My custom command calculates the variance for a set number of time steps, the way it works is quite similar to average command, example:

user> variance ETH/BTC ask 10

advisorbot> The variance ETH/BTC ask price over last 10 timesteps was 1.354

In this command we have to specify the product time, ask/bid and number of timesteps.

Logic behind the program is quite similar to calculating average till we reach the order-entry extraction stage, which starts with the conversion of third element of our token vector to integer, then iterating over time-step record list (i.e the private variable of our mainadvisorbot class) to add each entry in our full\_entries vector, by creating a reverse iterator of map class.

Now once we get all the entries according to user input filter, we can start doing calculation for variance.

Formula for variance

Here, I have first calculated the mean after that we iterate over the full\_entries vector again to compute with the help of power function of maths library. Then divide the final output with vector size, And in the end we are printing the output through C++ console.