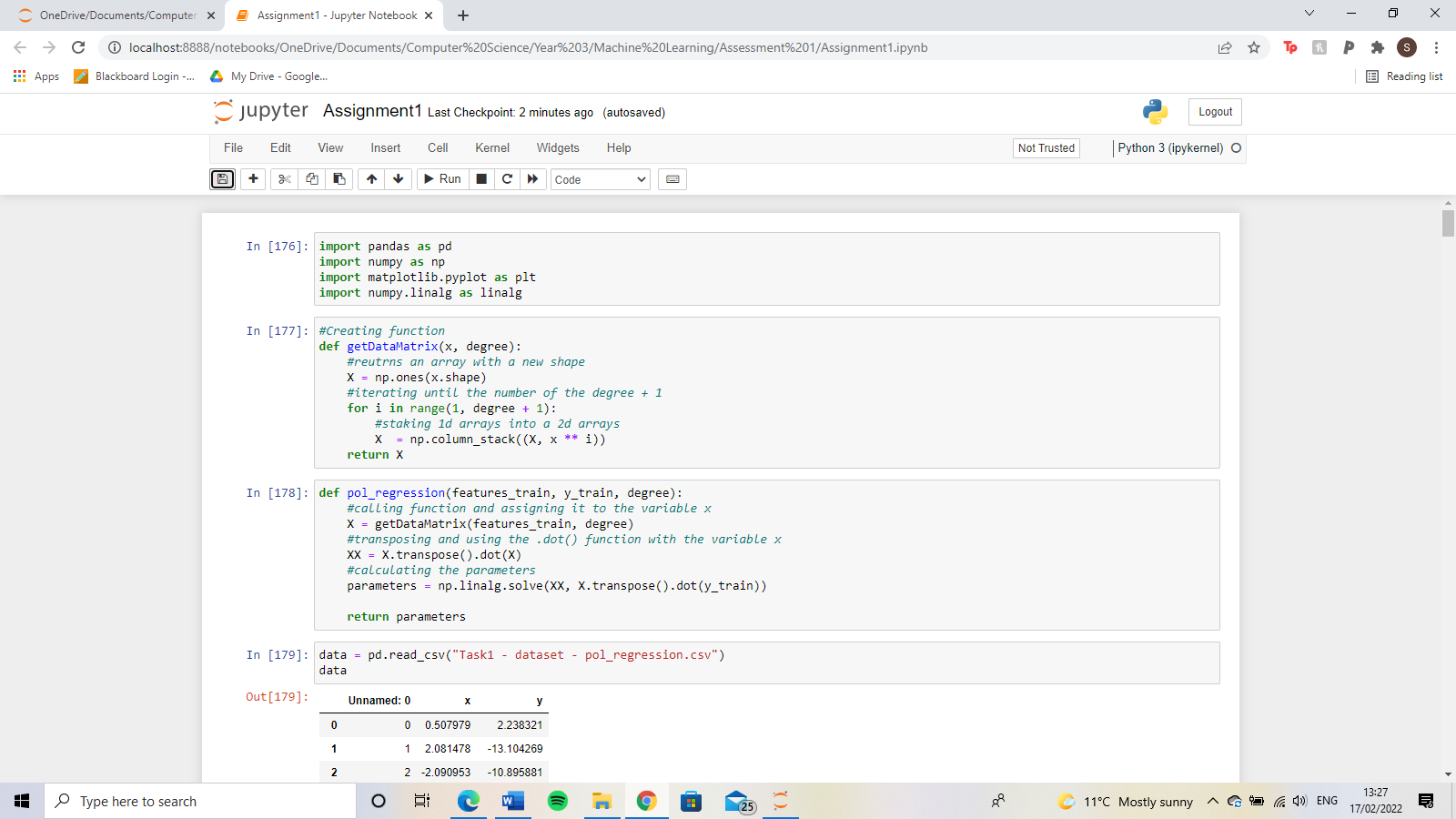
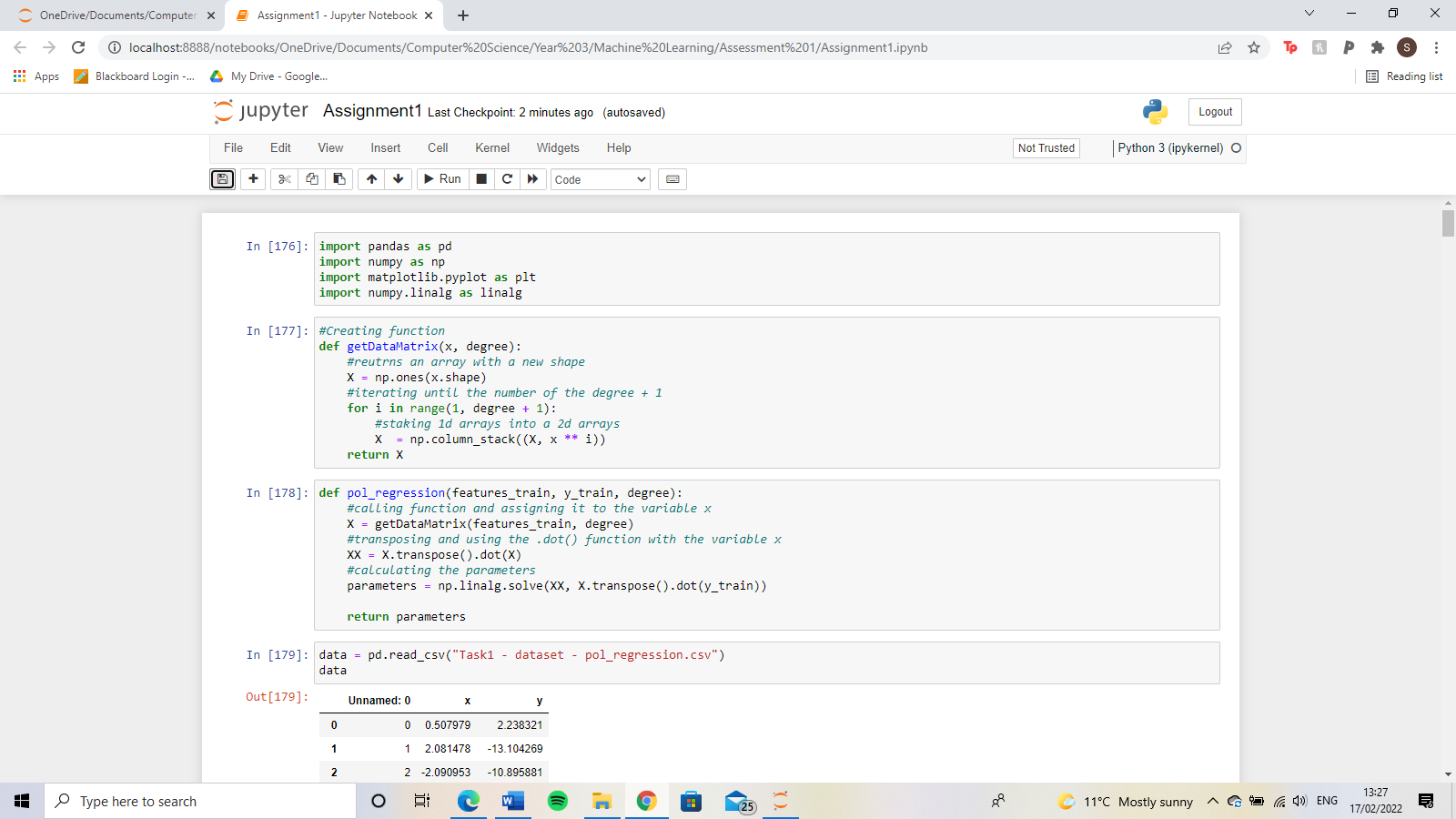
Machine Learning Assignment 1

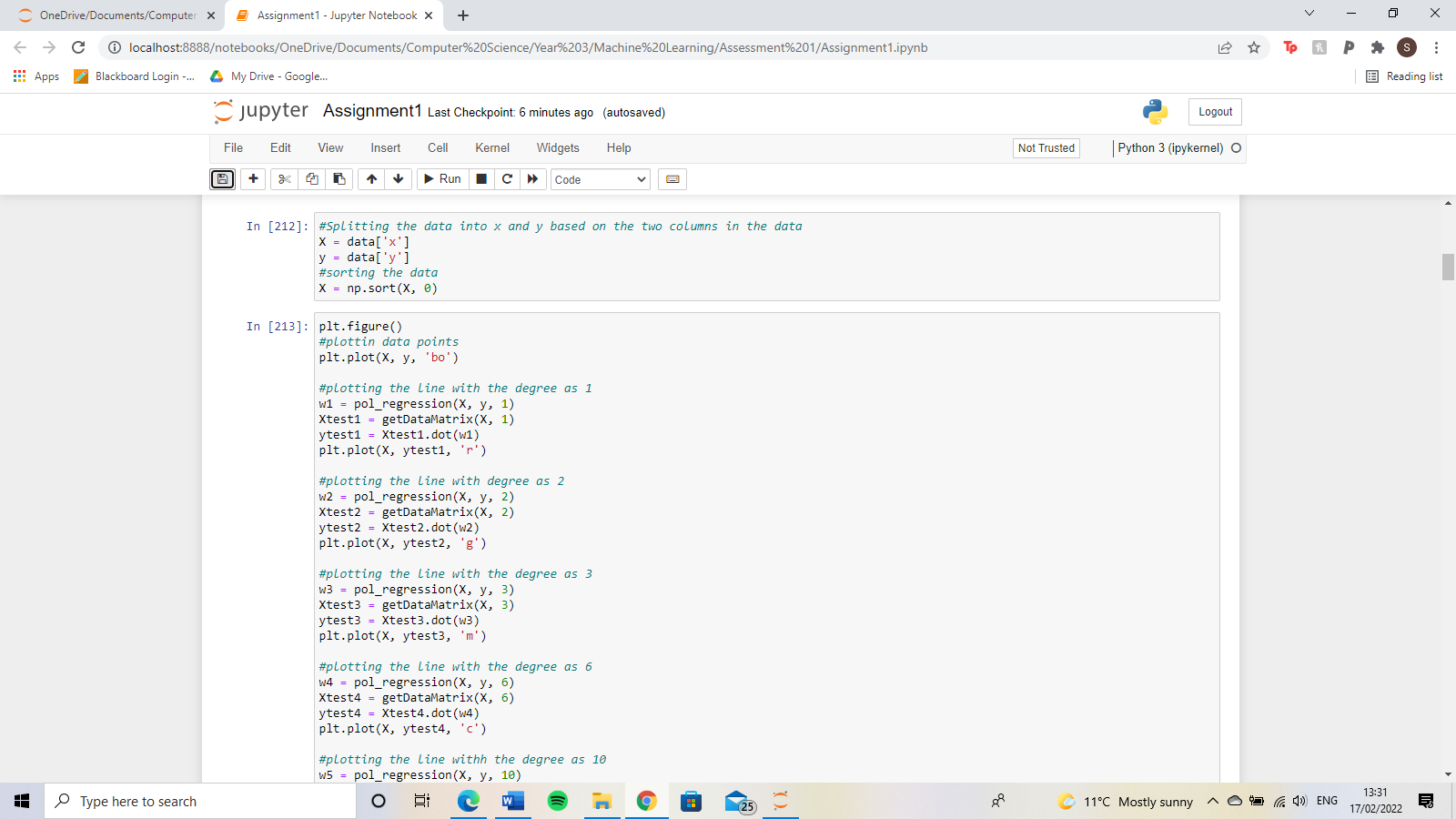
# Task 1

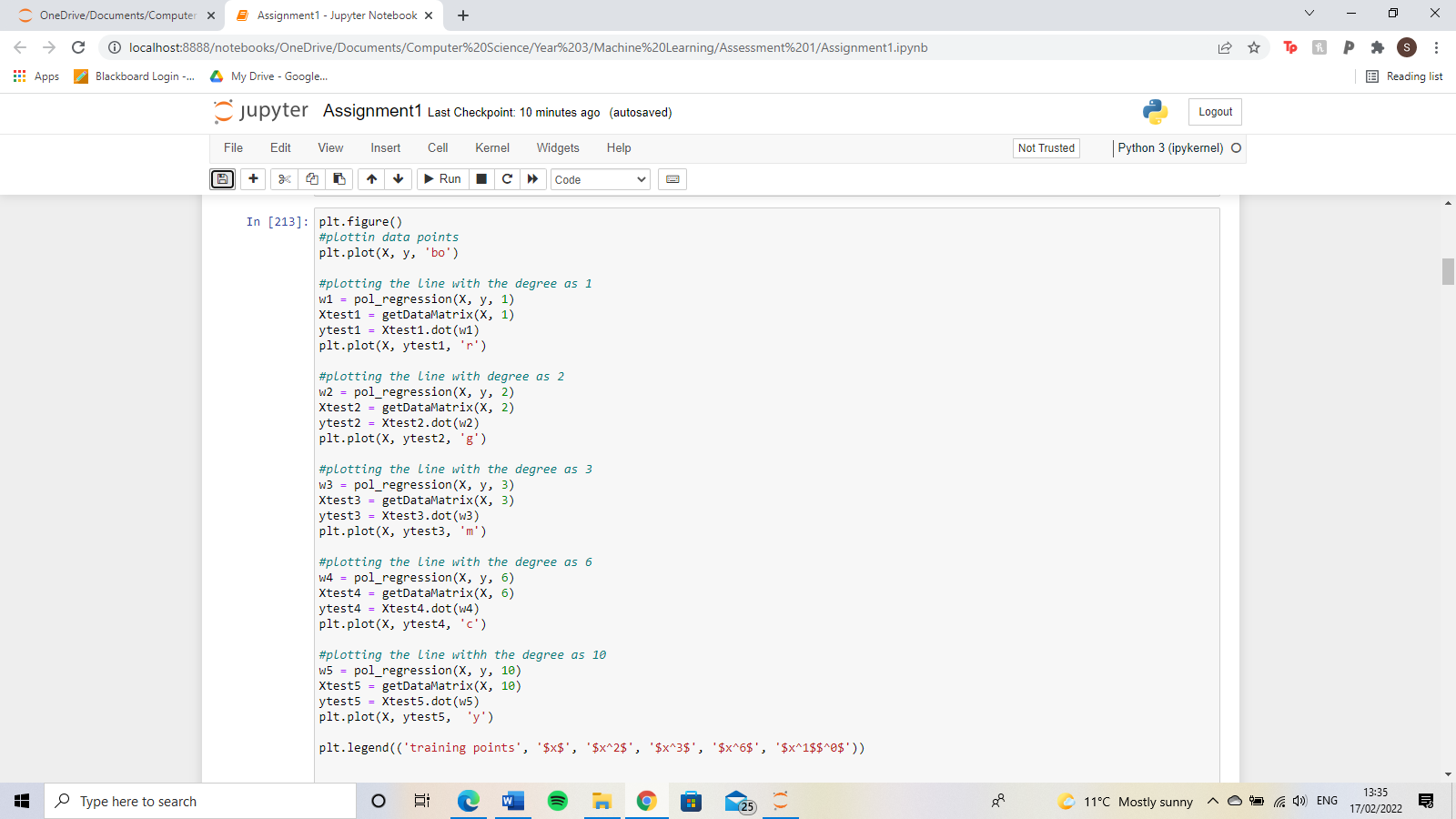
1. To make the polynomial regression, I, first of all, created a helper function called get data matrix which takes in the x train and the degree as parameters. It iterates until it gets to the same number as the degree plus 1 and takes the x train data with the index of the loop as the power as a 1D array. These 1D arrays get stacked into a 2D array which is returned from the function.

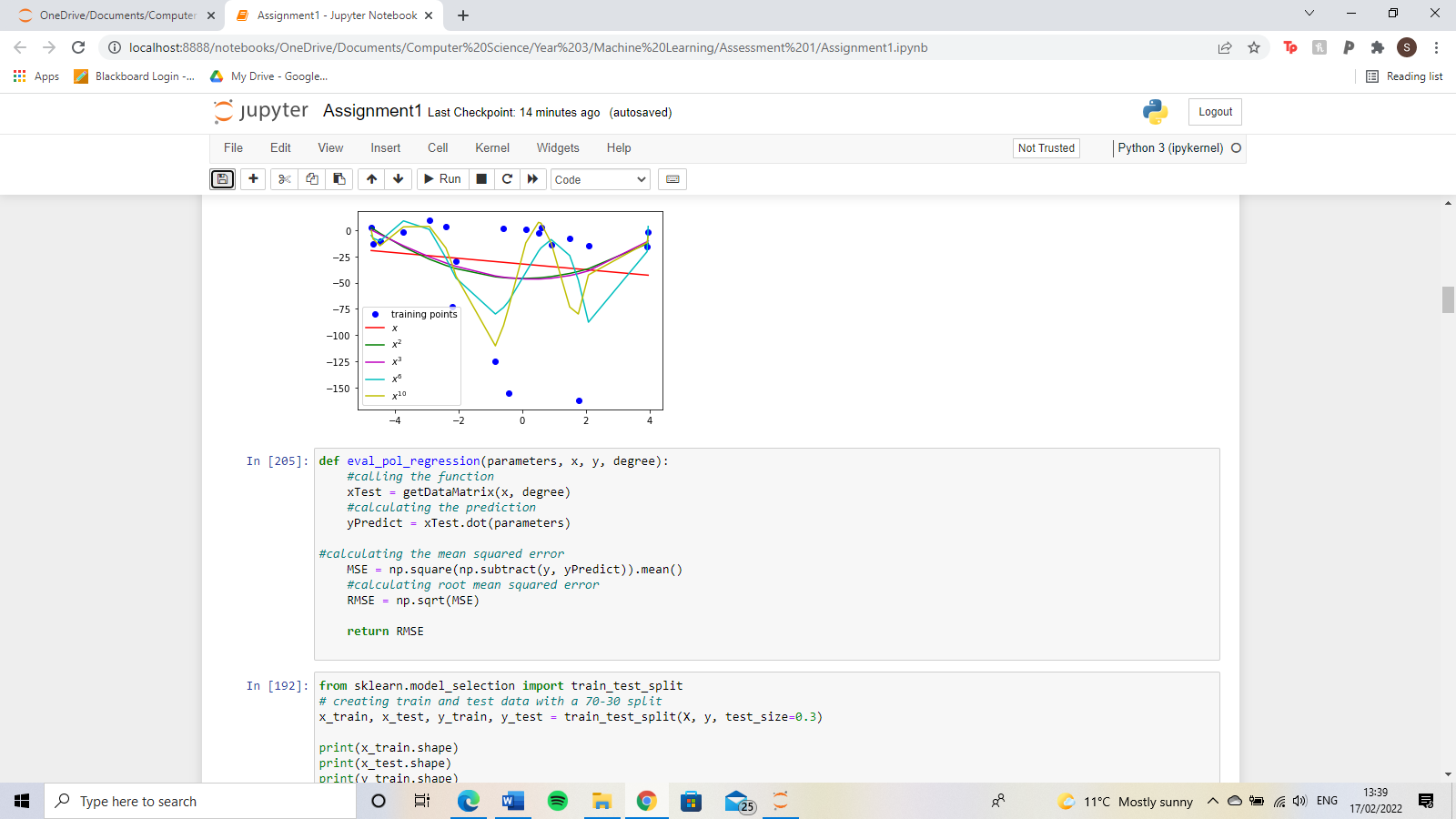
Then, in the polynomial regression function, the 2D array is then transposed and multiplied by itself. This is assigned to the variable XX. To get the value of the parameters, NumPy linear algebra solve is used with the XX variable and the transpose of 2D array multiplied by the y train data. This is returned from the function.



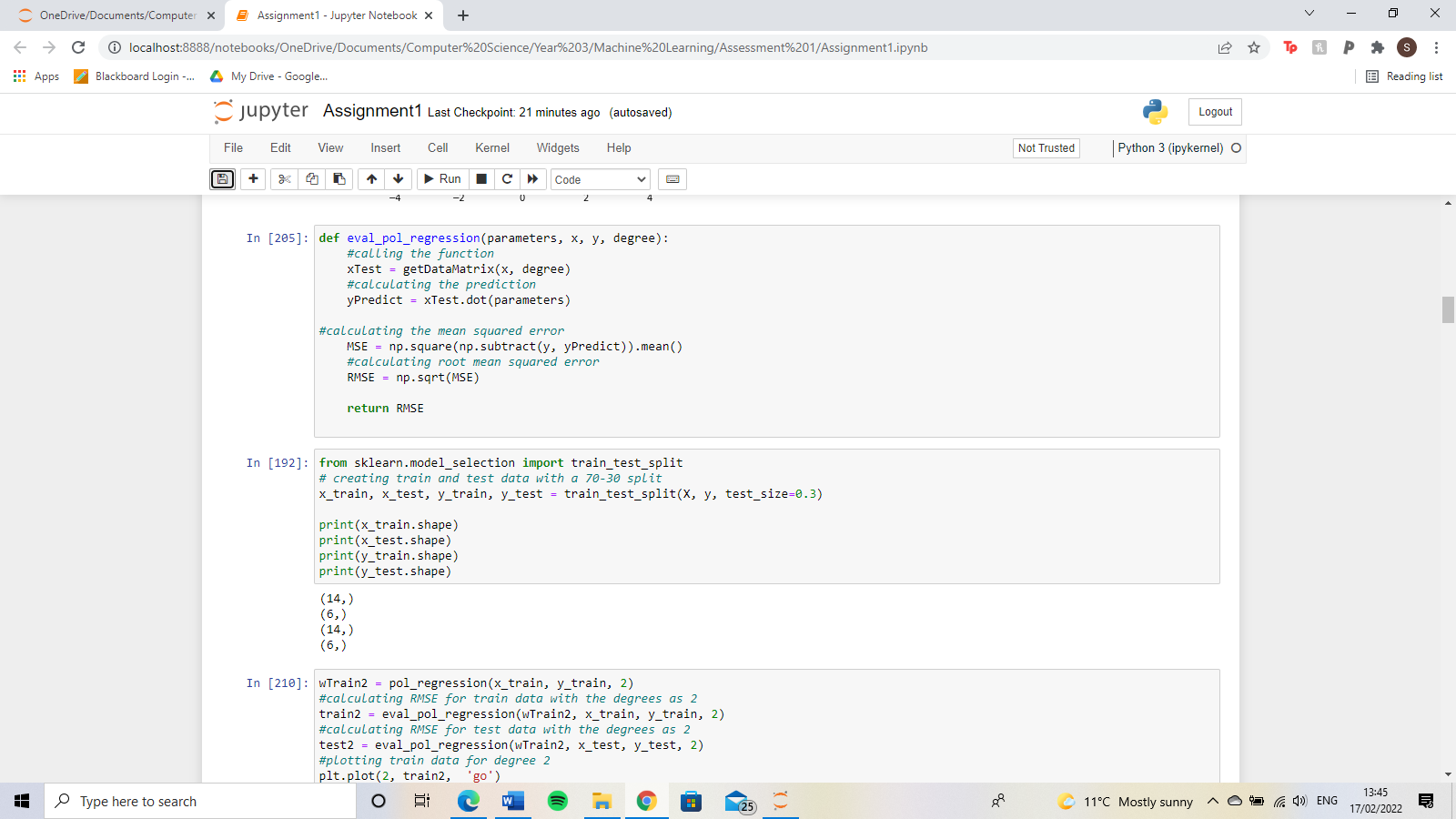
1. To plot the graph, the polynomial regression function is called and assigned to the variable w. The result of the data matrix function is assigned to the variable xTest. Using these variables, xtest and w are multiplied together to make ytest. Then the X column of the data is plotted on the x-axis and ytest on the y axis. Based on the lines plotted on the graph, the polynomial regression that I would choose is the one with a degree of 10. This is because the line is the closest to the training points from the original data.



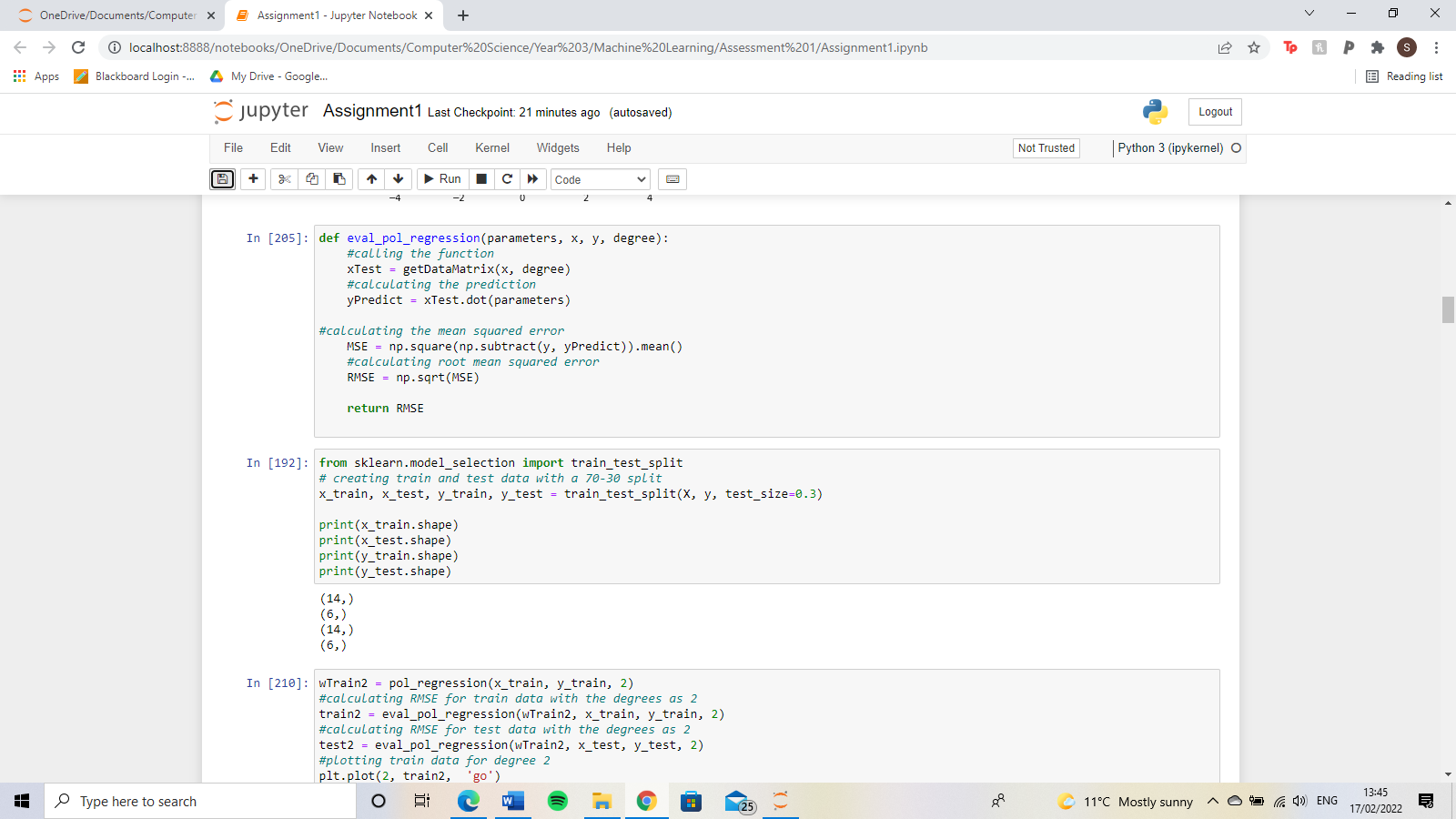


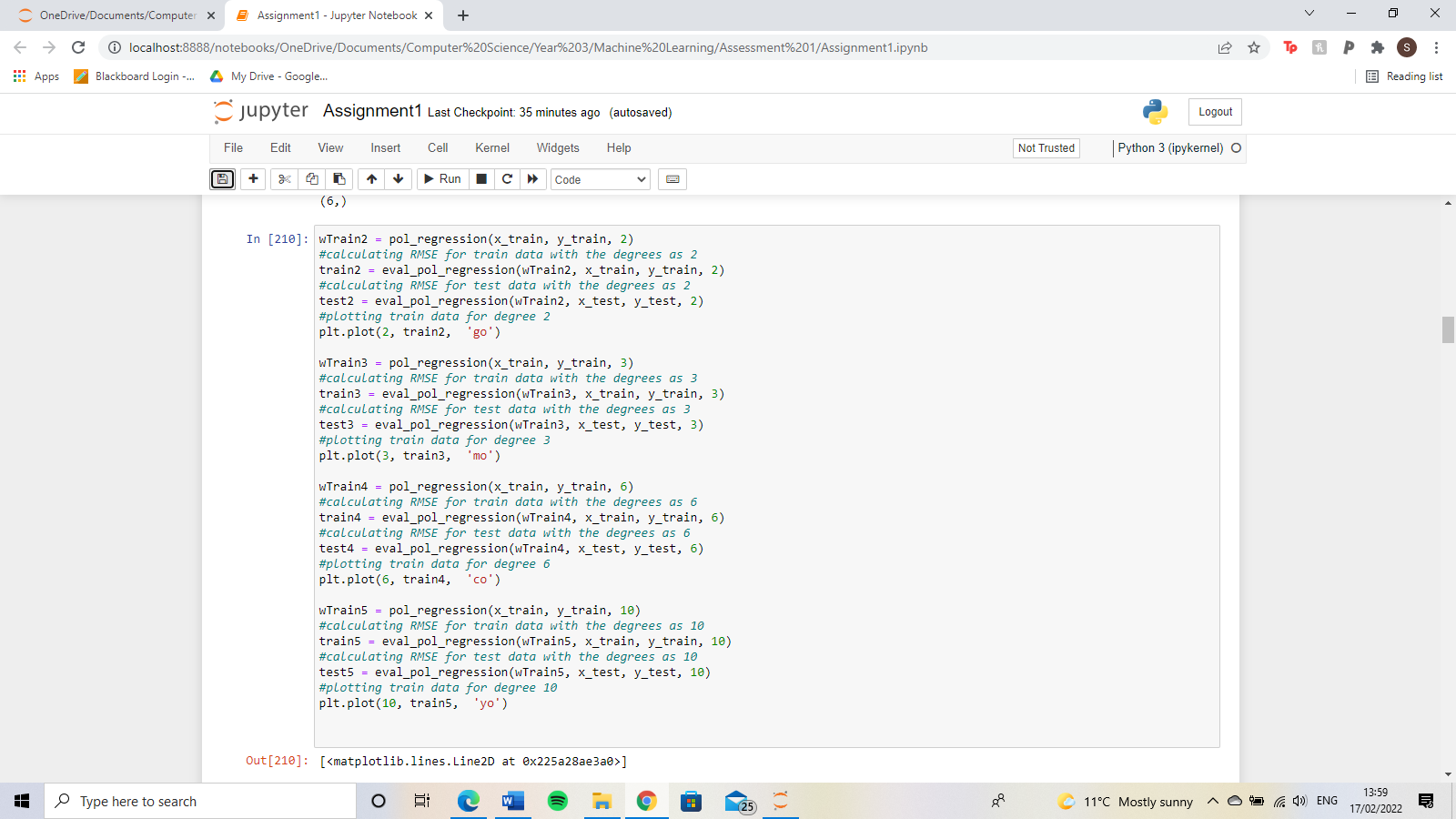


1. To calculate the RMSE, first of all, I created a function that takes parameters, x, y and the degree as parameters. The get data matrix function is called and assigned to the variable xTest. This is then used to calculate the prediction by using the dot function with xTest and parameters. Then I calculated the mean square deviation. This is done by subtracting y and the prediction, then squaring it, then getting the mean of these values. Finally, to get the RMSE I got the square root of the MSE. This is returned in the function.

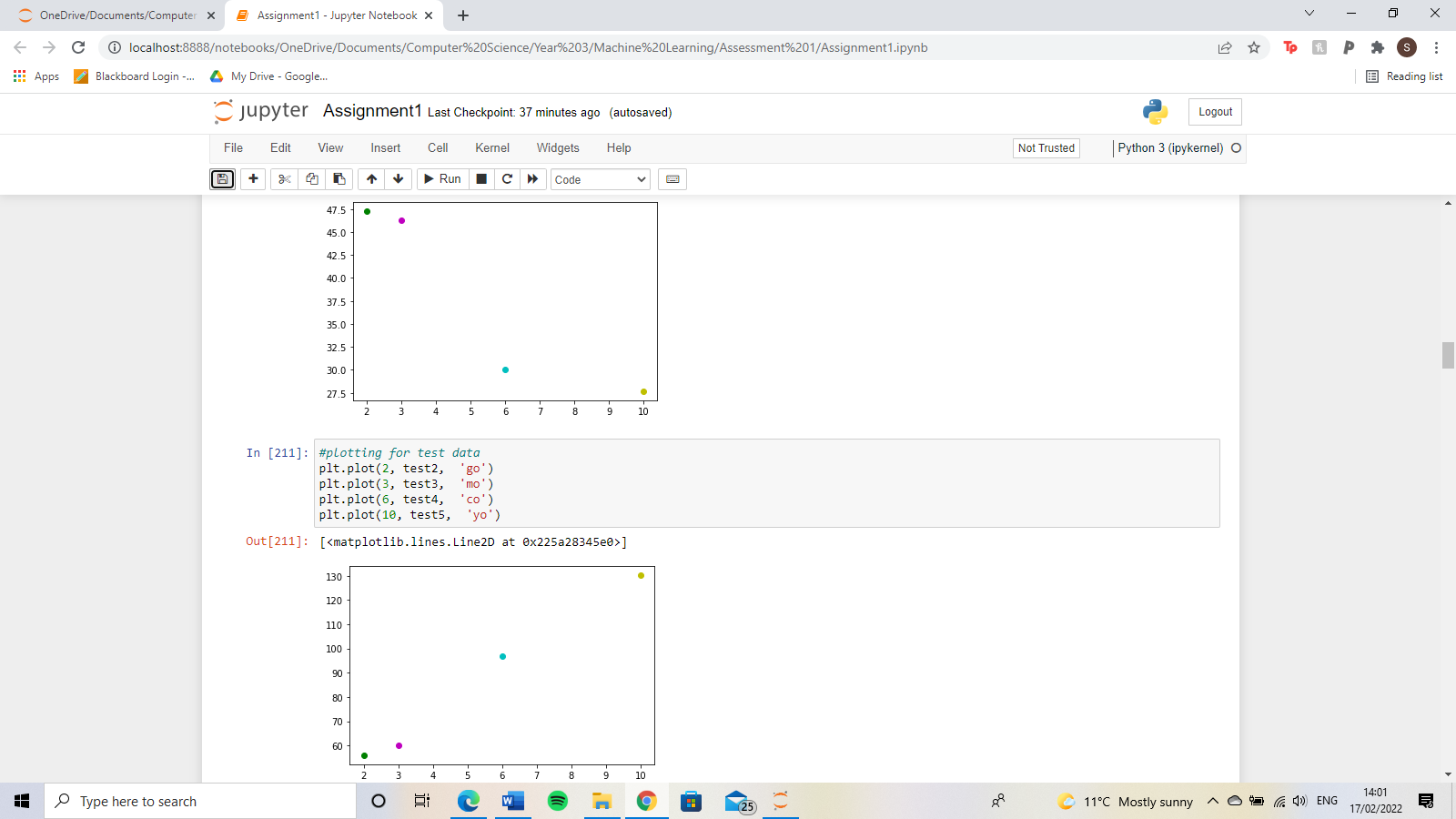


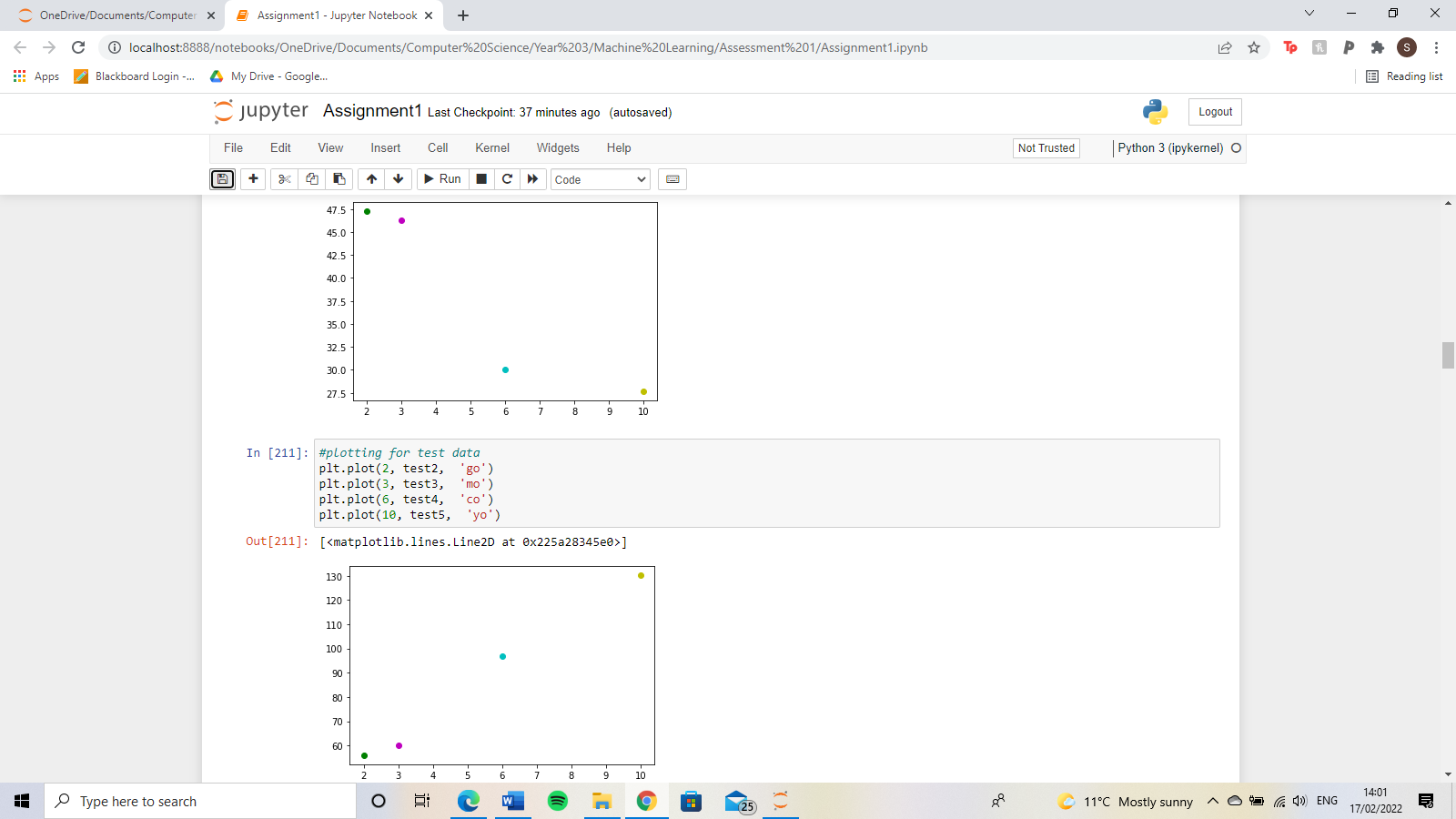
To plot the RMSE against the degree, I, first of all, split the data up into x train and test as well as y test and train with 30% test data and 70% train data. With this, I then call the polynomial regression using the x and y train data. This is then used for evaluation polynomial regression function using either the train data or test data. This is then plotted through matplotlib.





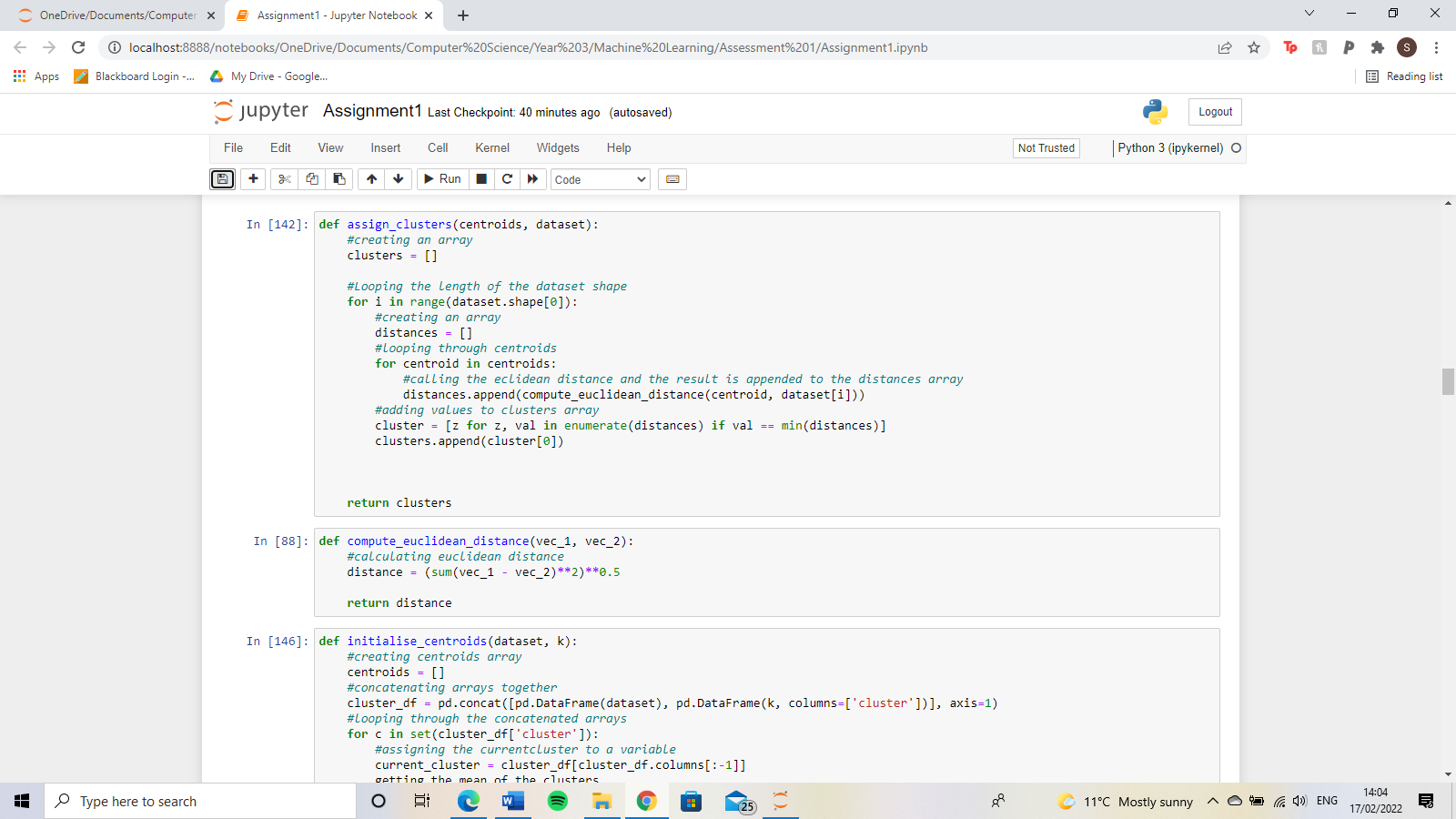
The graph for the training data RMSE shows that the greater the degree the lower the RMSE. This is good because it shows that the greater the degree, the better the fit to the model is. However, for the test data, it was the opposite of train data. As the degree increased, the RMSE increased. This is called overfitting. Overfitting is where the model performs well on the training data, however, it does not perform well on the evaluation. This is because the model learns the detail in the training data that negatively impacts the performance on the data. Underfitting is where a model performs poorly on the training data. This is due to the model not capturing the relationship between the inputs and the target values. Underfitting is shown in these graphs in the lower degrees where the line in the polynomial regression does not fit the data points very well.



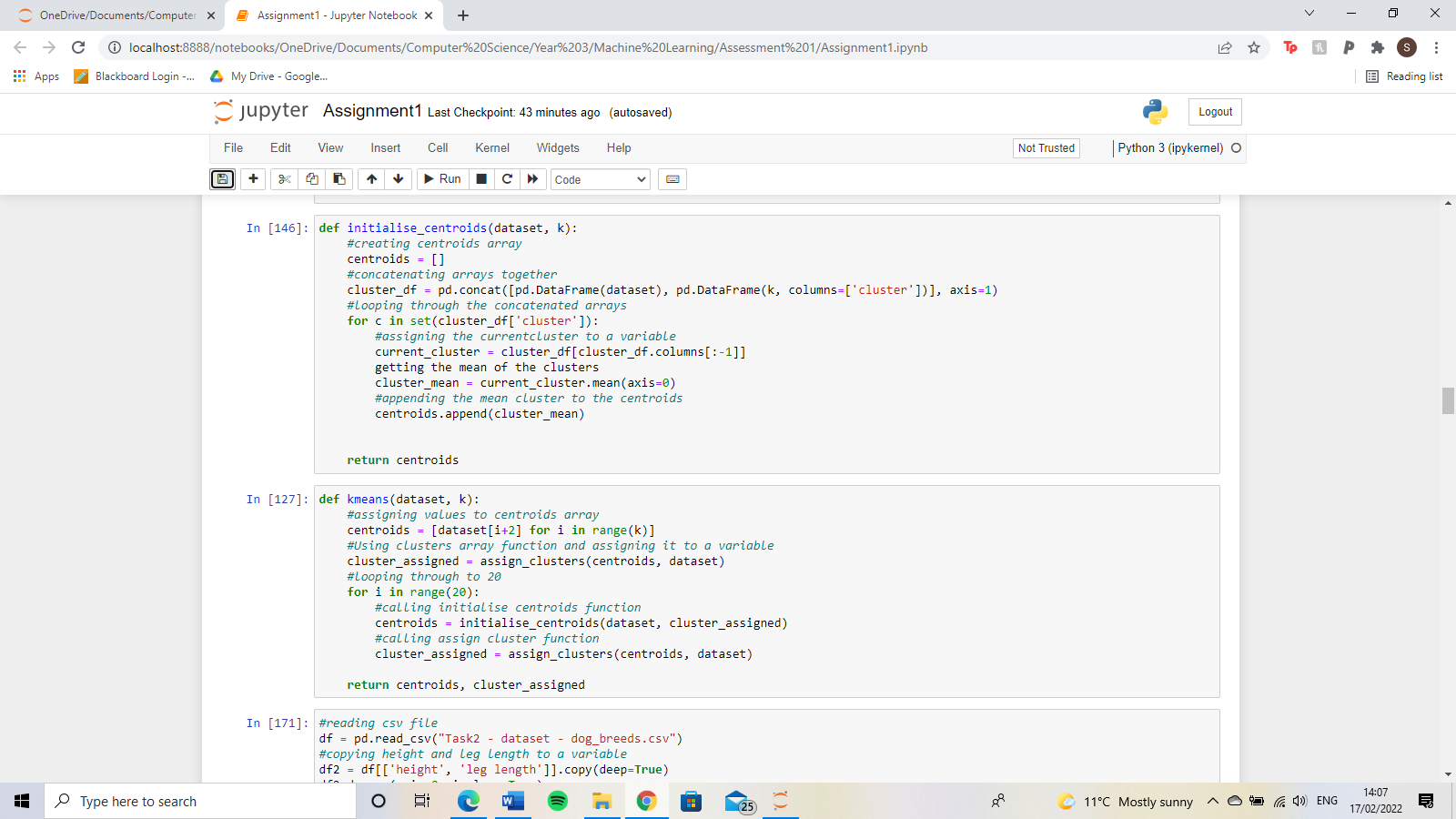


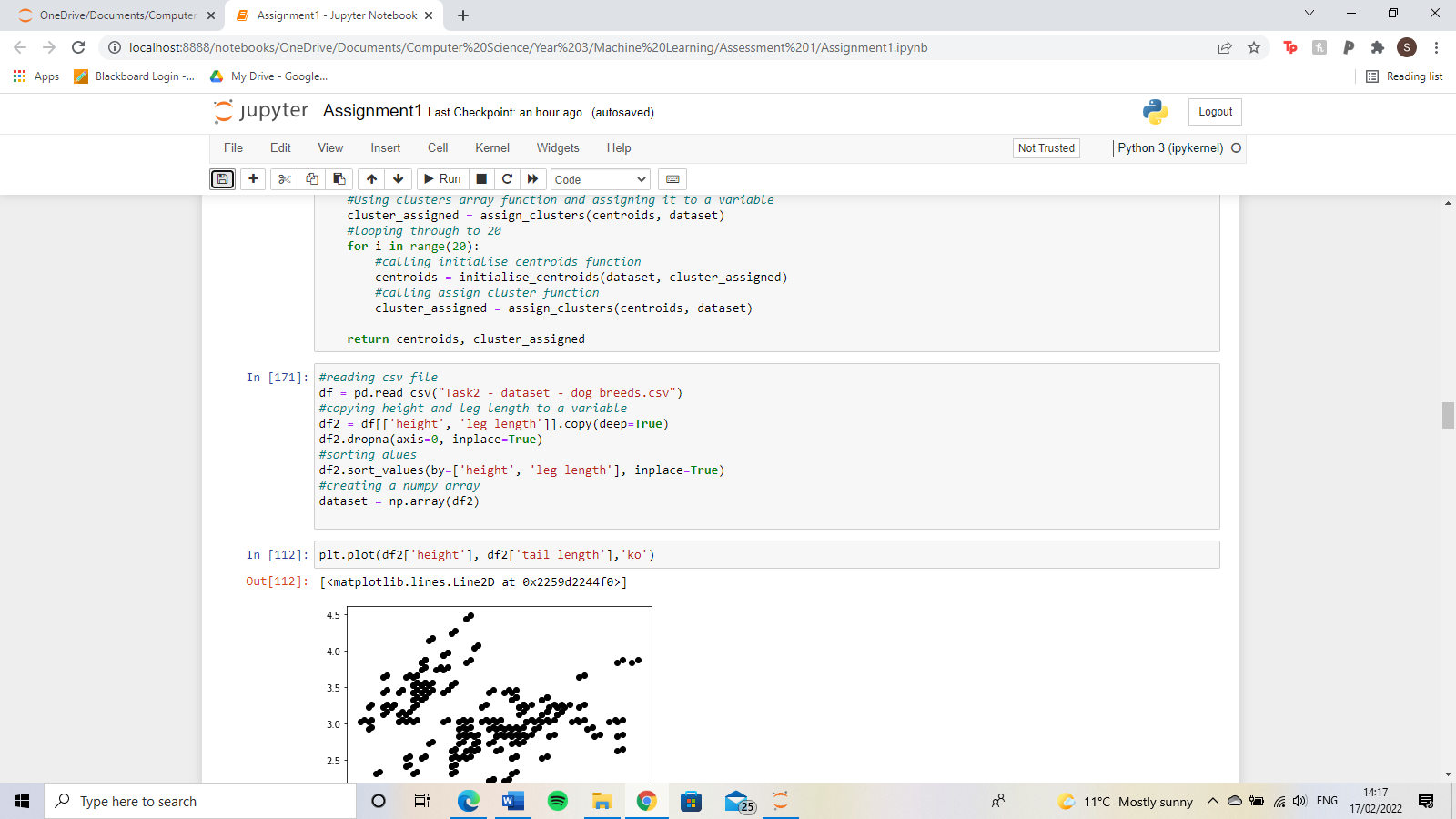
# Task 2

1. For the k-means algorithm, I implemented four functions. One to calculate the Euclidean distance, one to initialise the centroids, one to assign the clusters and finally one called k-means which combines all these functions. To calculate Euclidean distance, the sum of vector 1 subtract vector 2 is squared. This result is then raised by the power of 0.5. This value is returned from the function. The assign clusters function uses a nested loop to iterate over the Euclidean distance function, this is then added to a distances array. This array is then used to enumerate to get the closest centroid to a specific cluster.

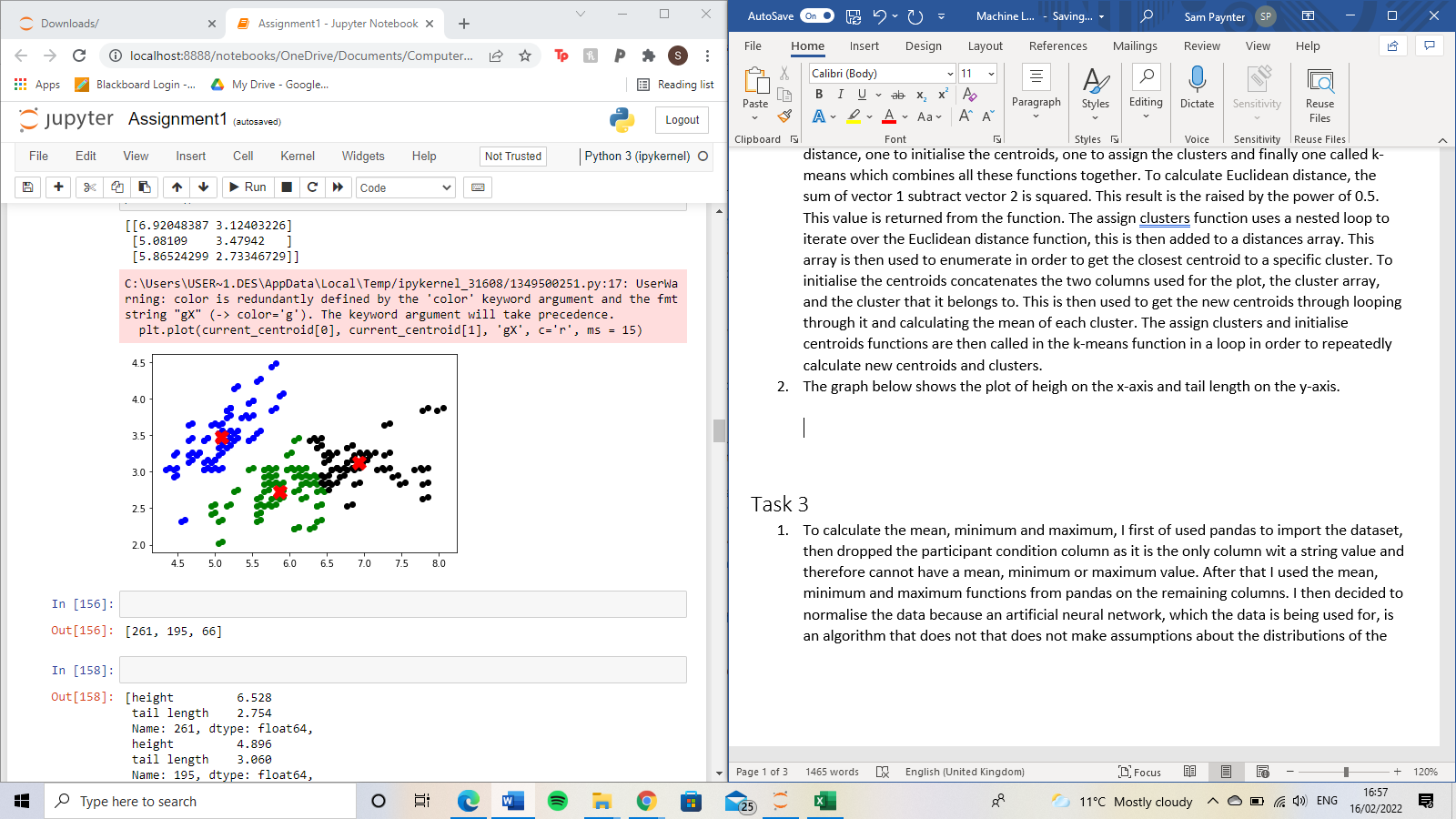


To initialise the centroids concatenates the two columns used for the plot, the cluster array, and the cluster that it belongs to. This is then used to get the new centroids by looping through them and calculating the mean of each cluster. The assign clusters and initialise centroids functions are then called in the k-means function in a loop to repeatedly calculate new centroids and clusters.

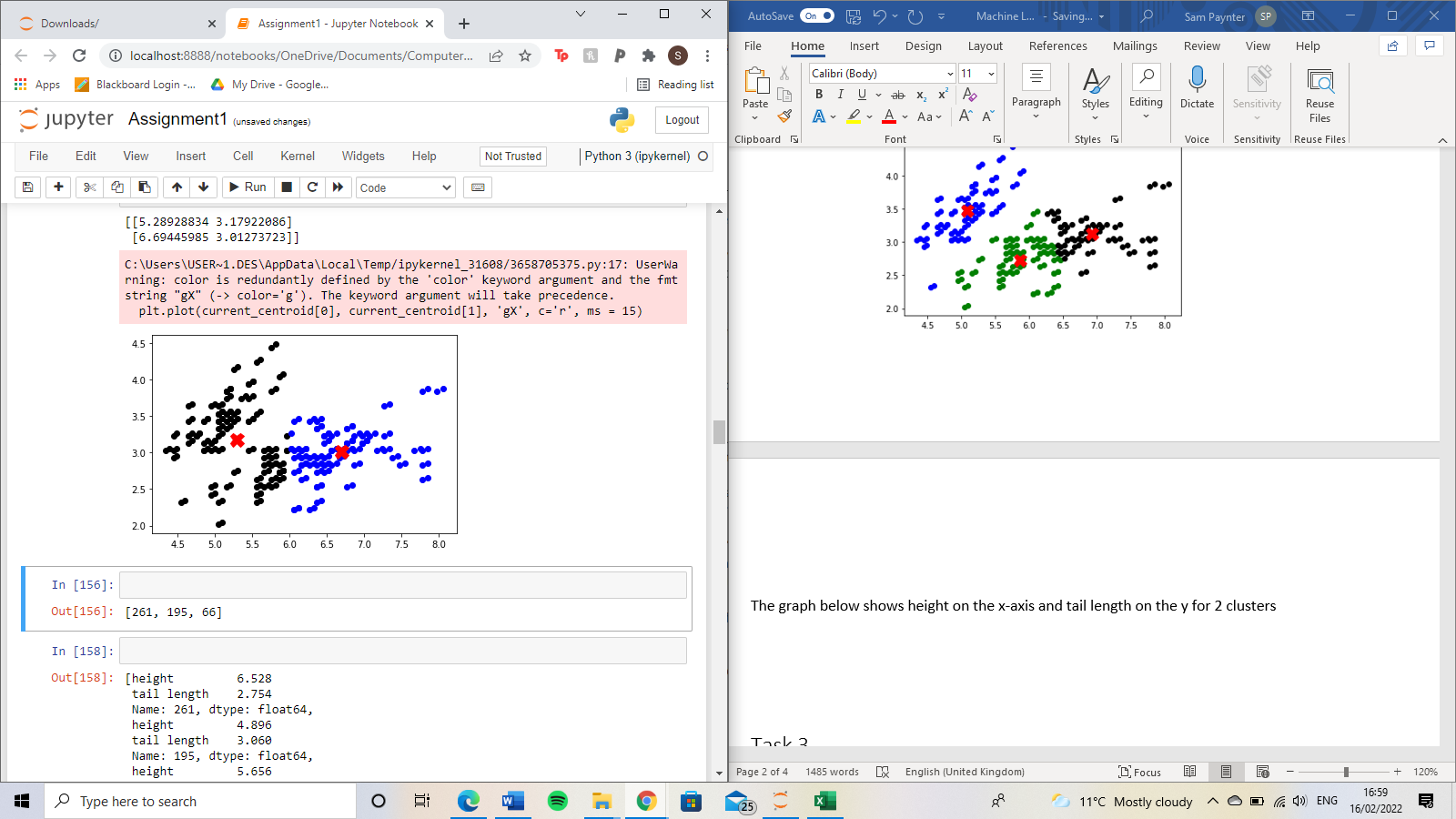




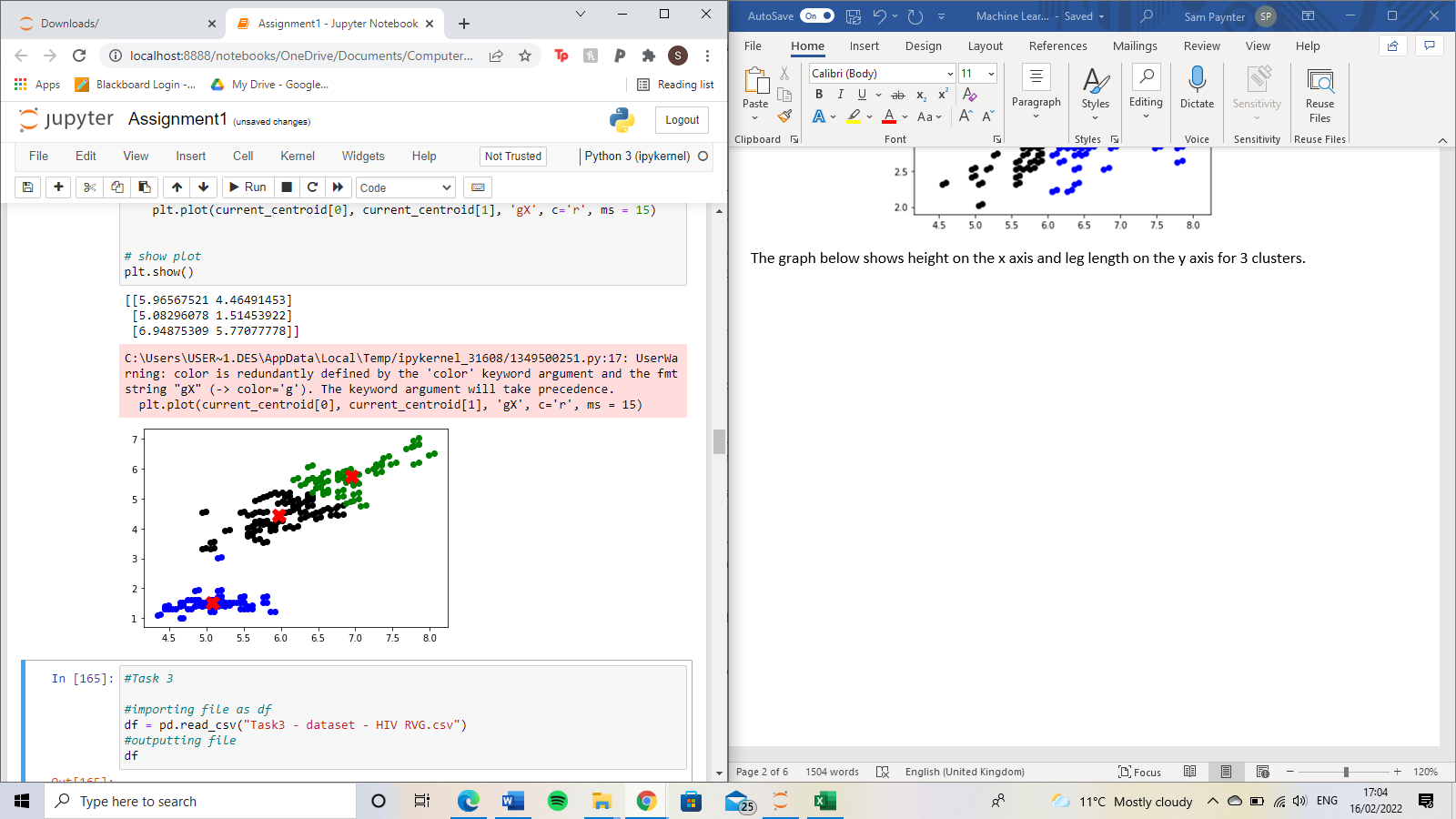
1. The graph below shows the plot of height on the x-axis and tail length on the y-axis for 3 clusters.



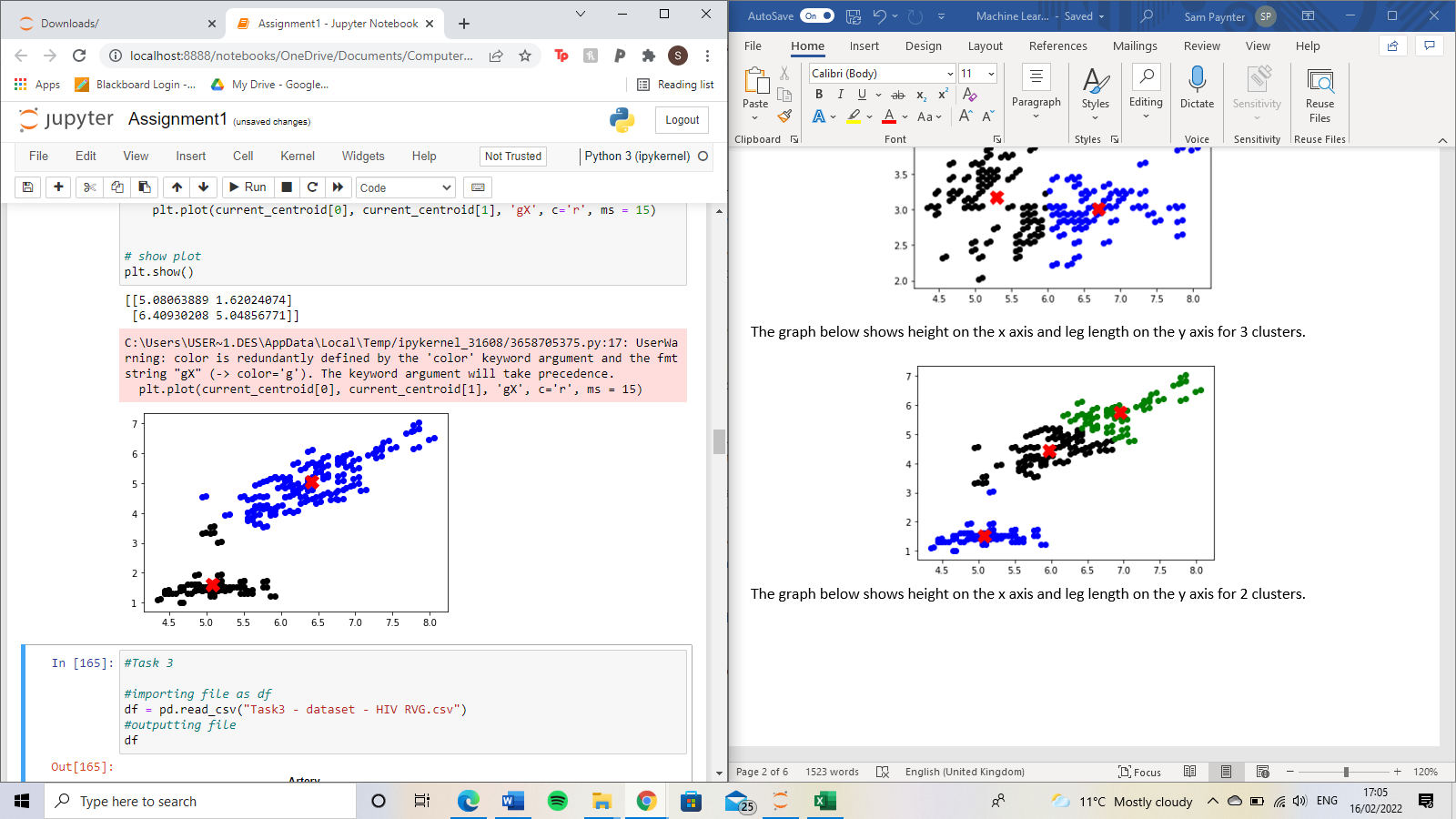
The graph below shows height on the x-axis and tail length on the y for 2 clusters.



The graph below shows height on the x axis and leg length on the y axis for 3 clusters.

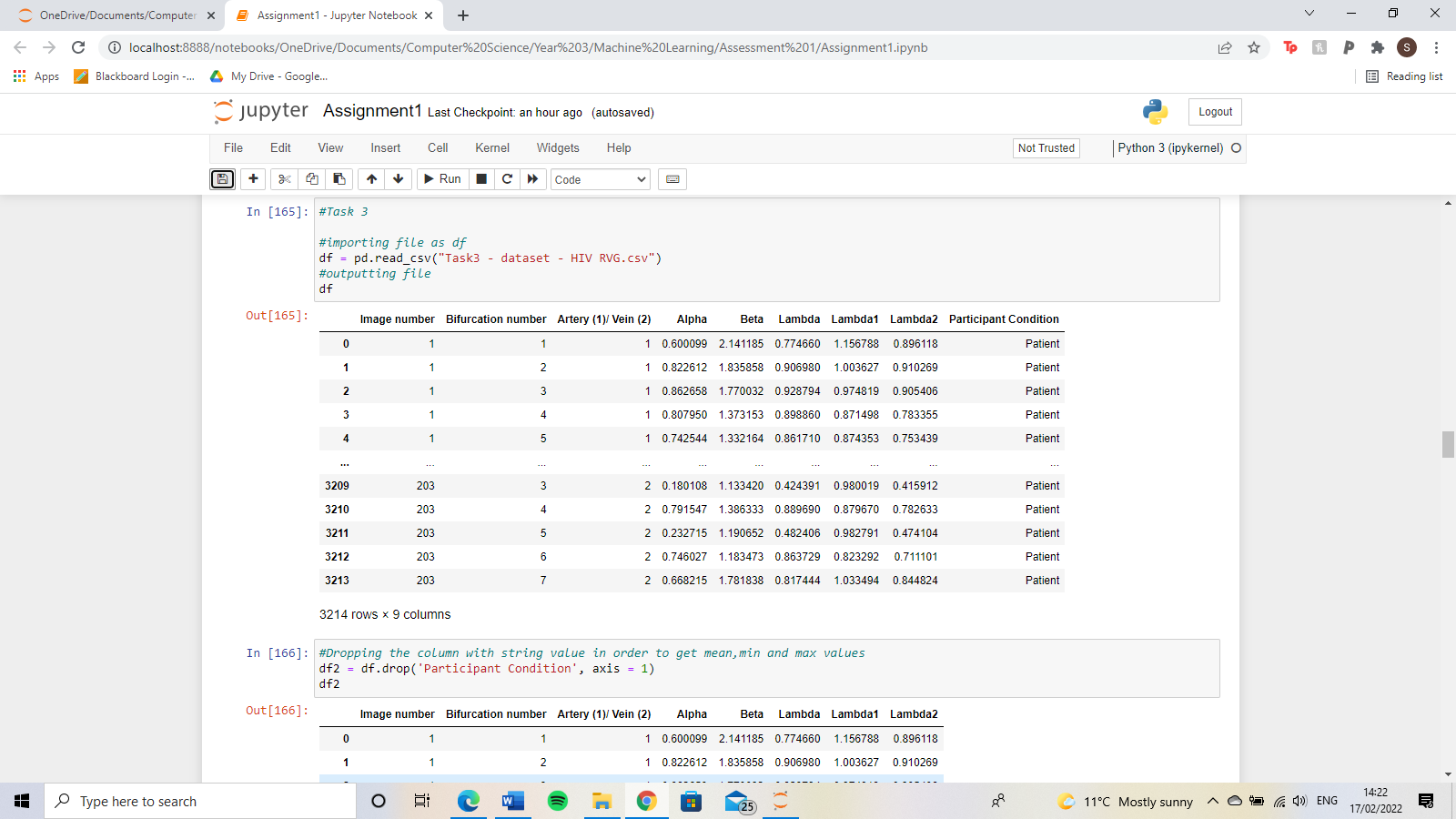


The graph below shows height on the x axis and leg length on the y axis for 2 clusters.



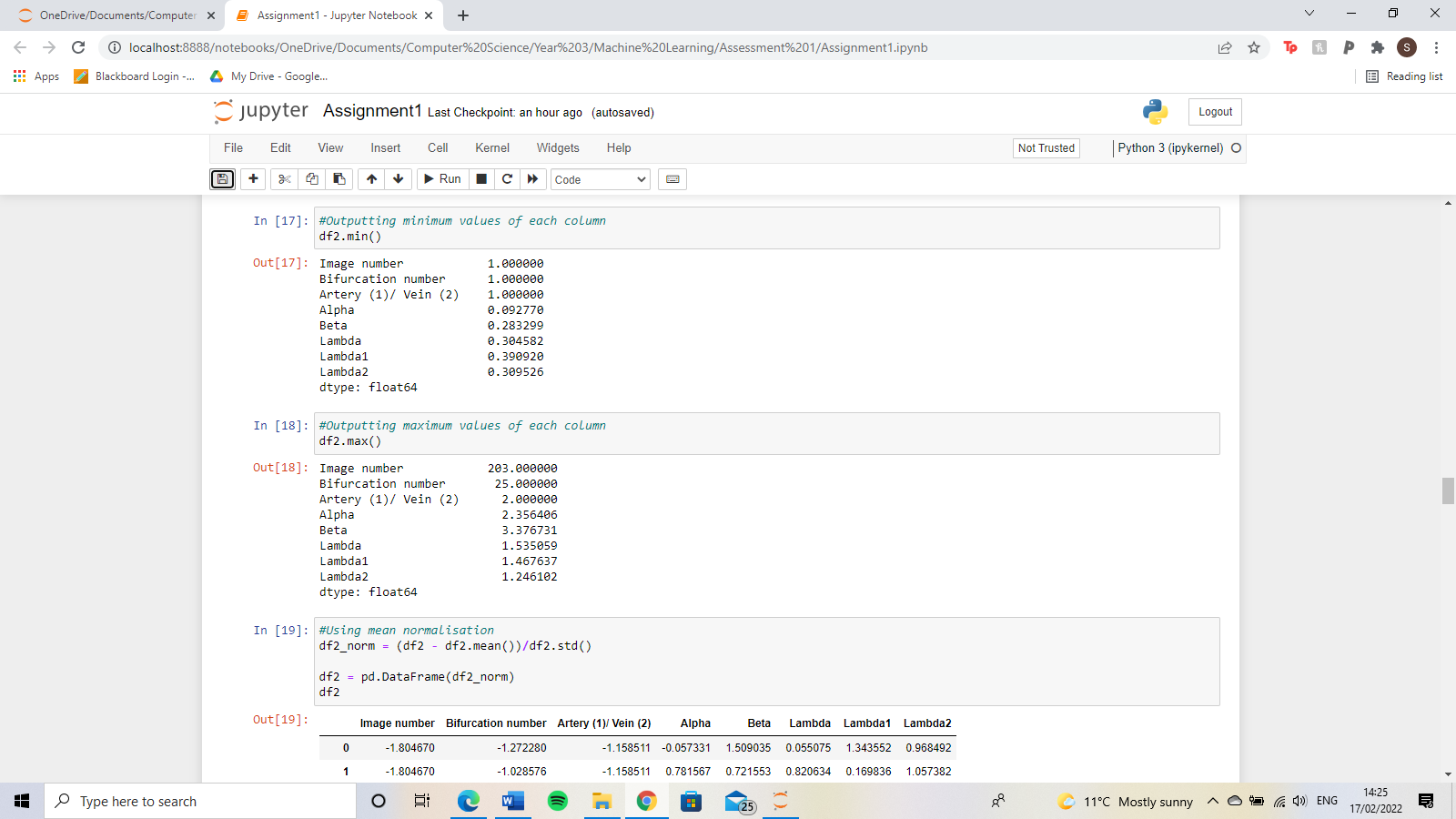
# Task 3

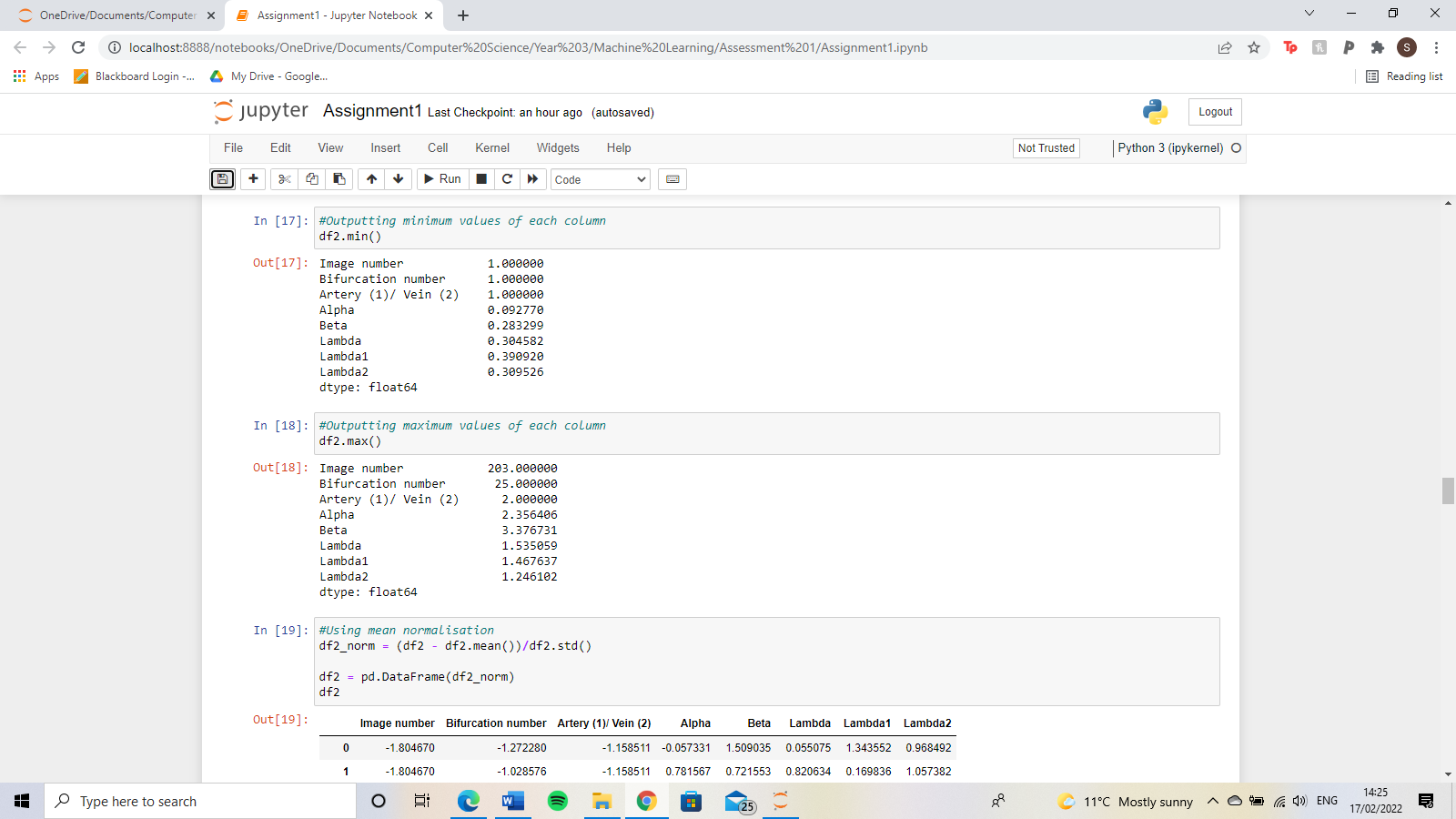
1. To calculate the mean, minimum and maximum, I first used pandas to import the dataset, then dropped the participant condition column as it is the only column with a string value and therefore cannot have a mean, minimum or maximum value. After that, I used the mean, minimum and maximum functions from pandas on the remaining columns. I then decided to normalise the data because an artificial neural network, which the data is being used for, is an algorithm that does not make assumptions about the distributions of the data. As well as that, it is also good for when you don’t know the distribution of the data. To generate the two plots, I used the seaborn boxplot function and for the density plot, I used the ‘kdeplot’ function.

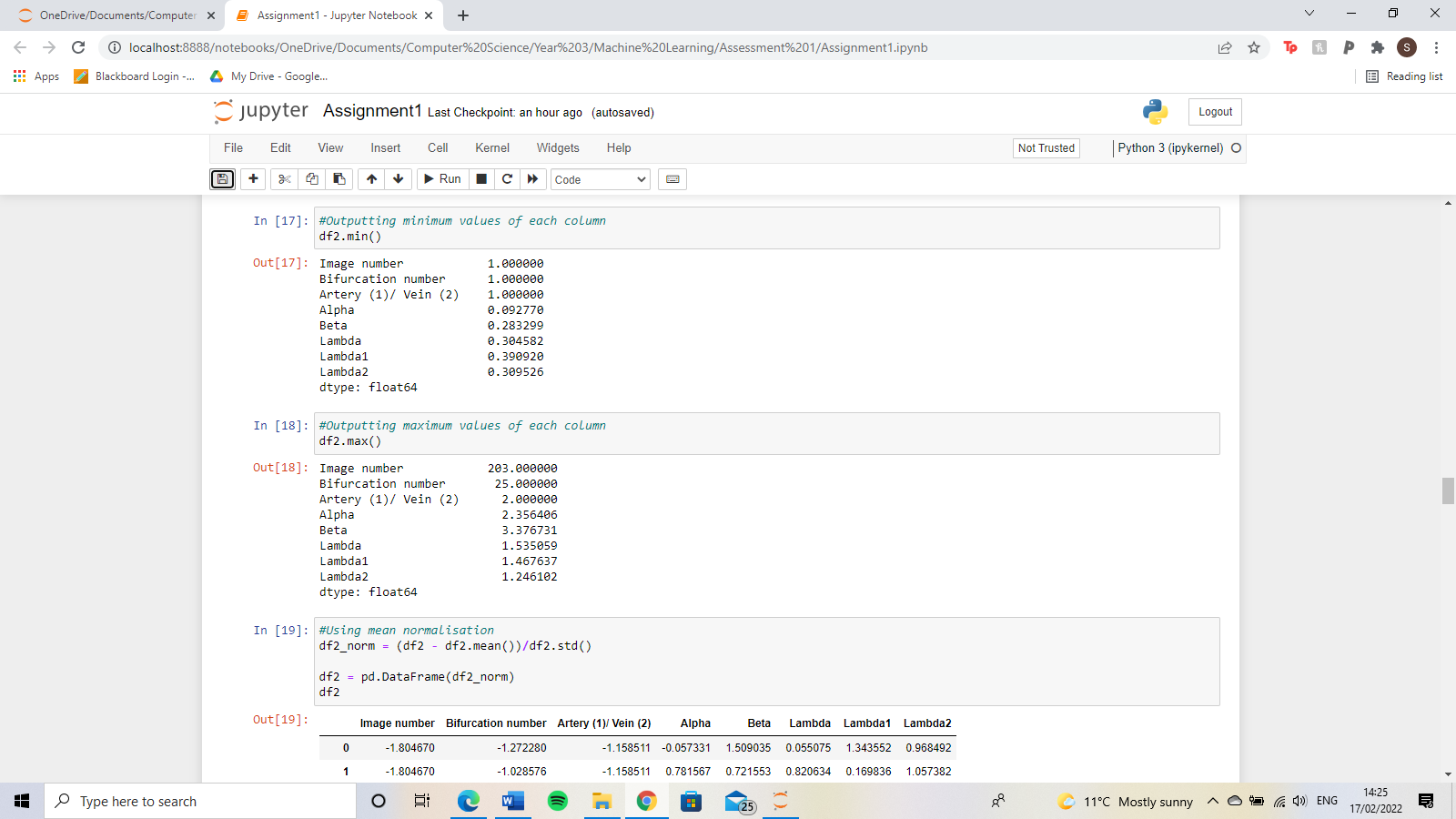


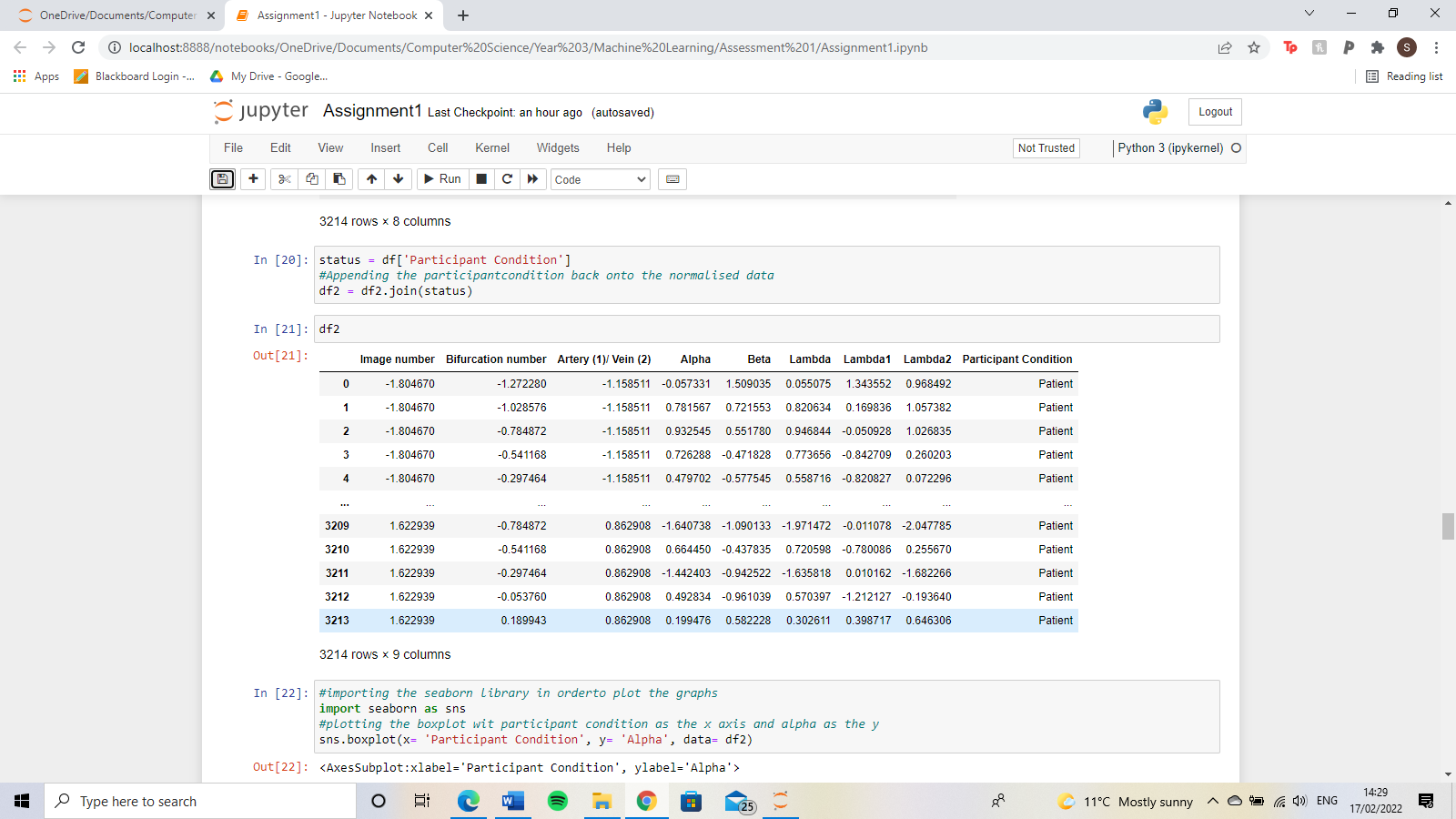


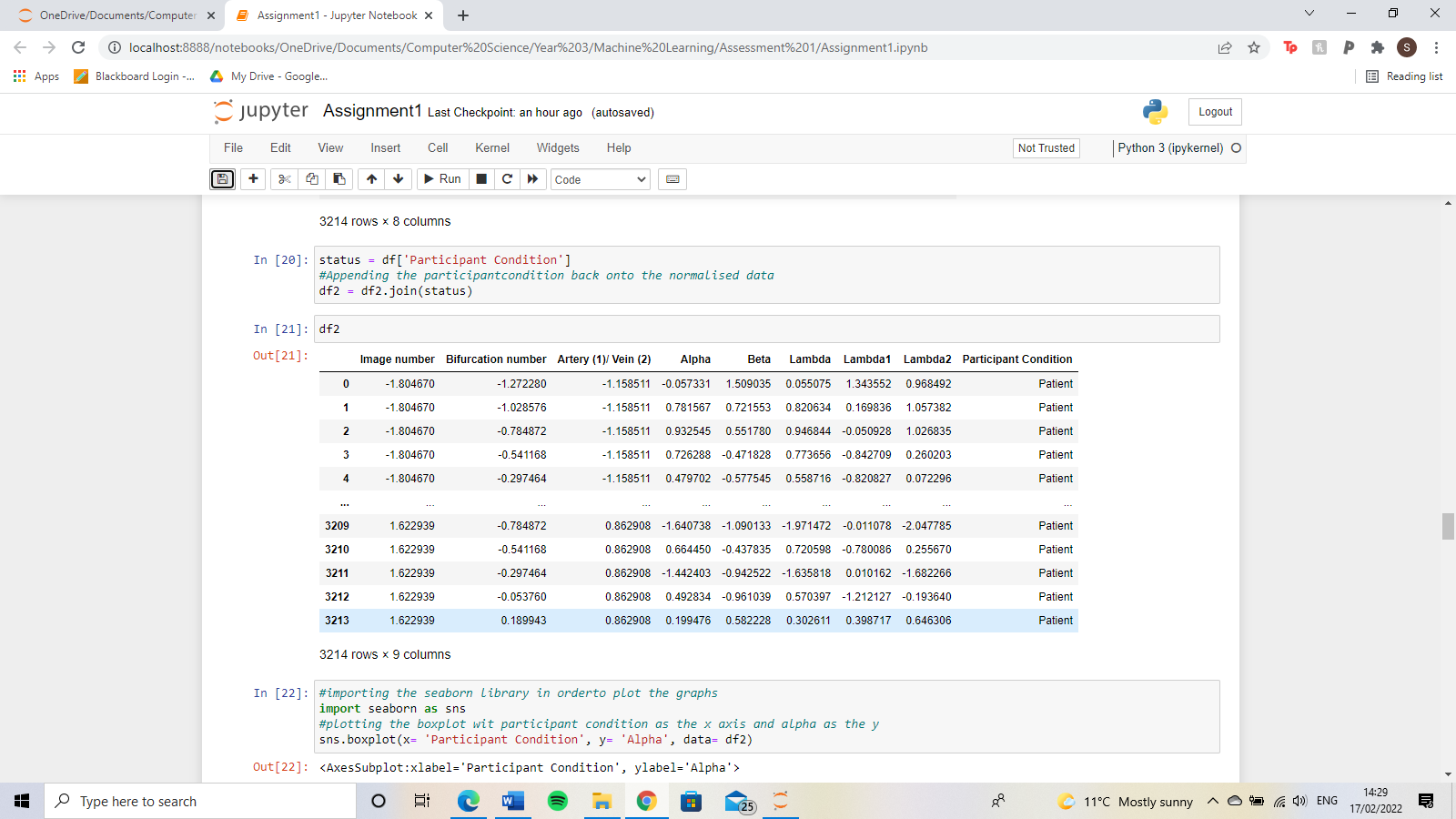


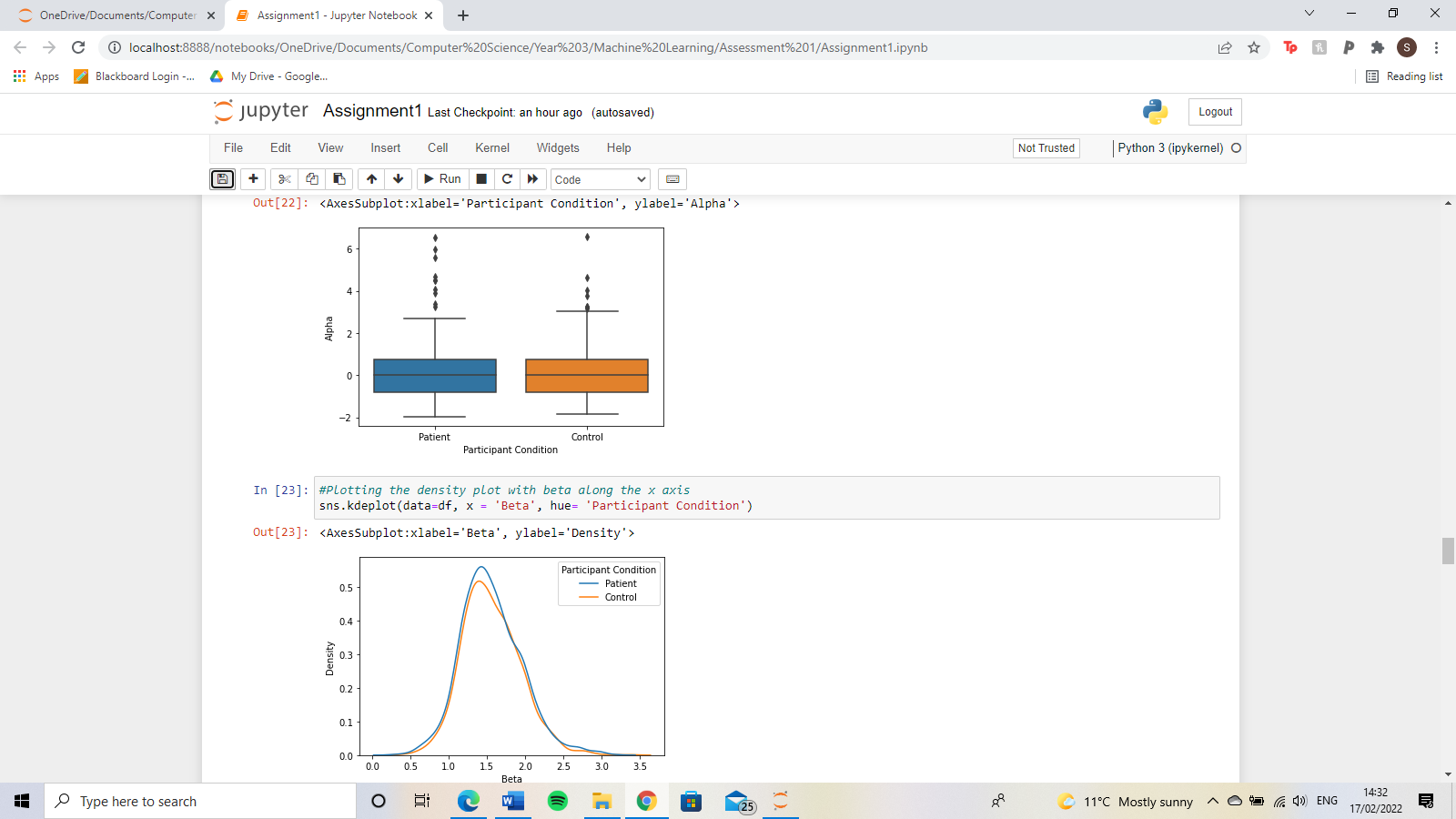




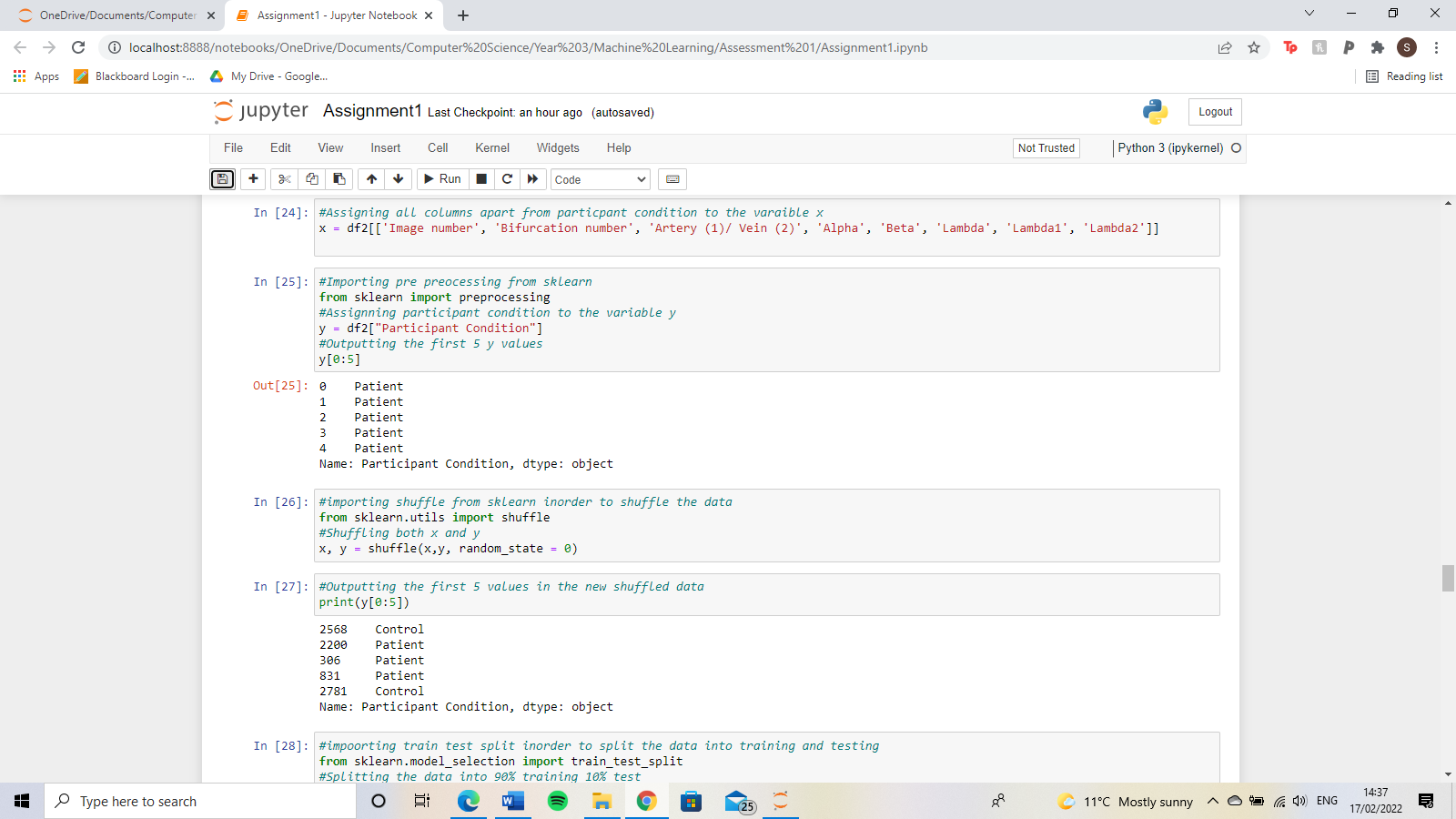
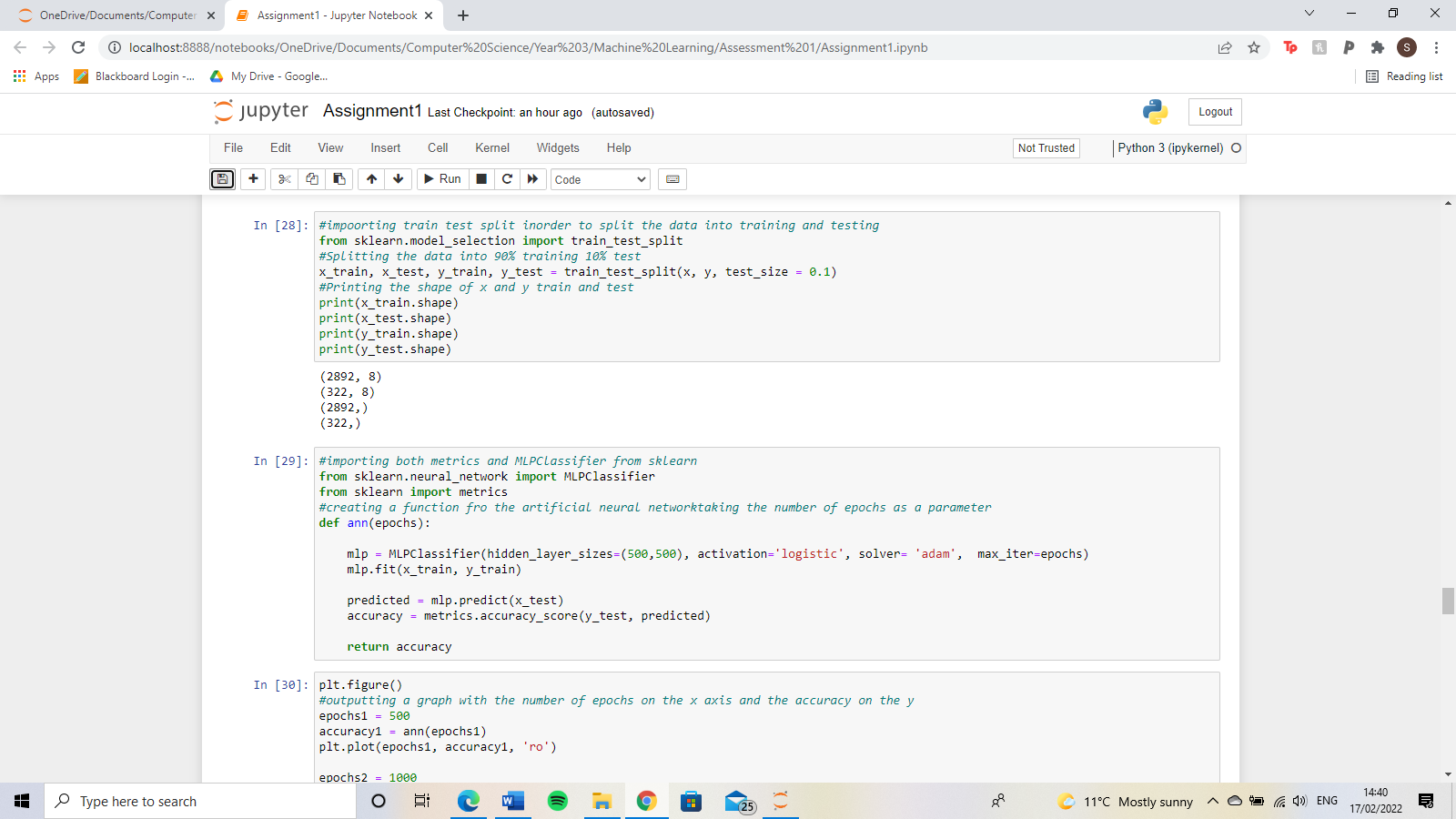






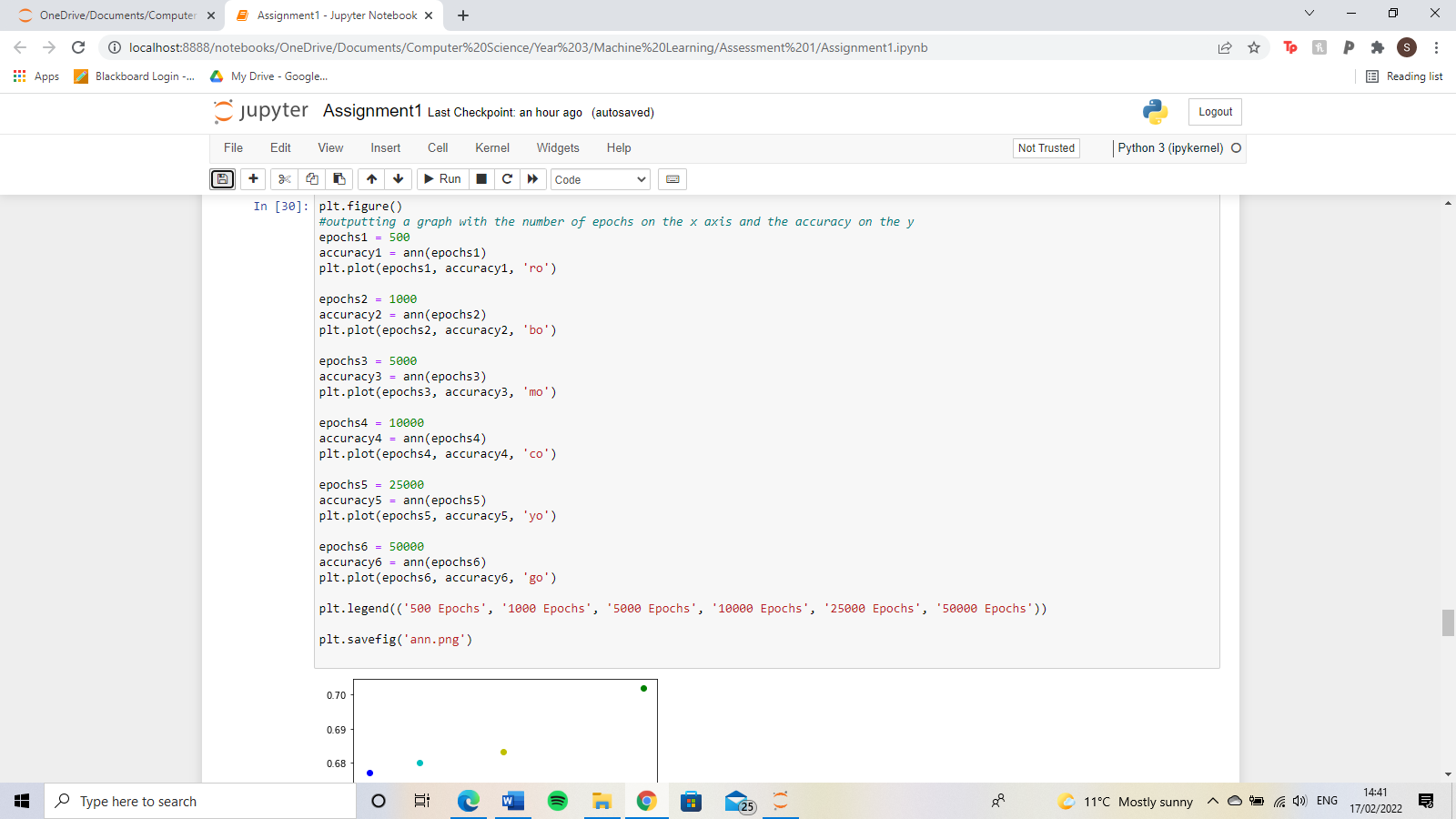


1. For both the neural network and random forest classifier, I used the sklearn machine learning library. I choose to use this due to its robustness, meaning that you can do a lot with it from pre-processing and splitting data to perceptron and regression. Therefore, the use of its built-in functions allows for less and more efficient code compared to the use of other machine learning libraries or coding these two classifiers from scratch. To prepare the data for the classifiers, I, first of all, appended the participant condition column back onto the data. I then assigned the participant condition column to the variable ‘y’ and the rest to the variable ‘x’. Then using shuffle from sklearn to shuffle both x and y. Then using sklearn again to split the data 90% train and 10% test data into the variables x train, x test, y train and y test. I then created a function for the artificial neural network which takes an integer value as an input. This will be used to determine the number of epochs. Inside the function, mlp classifier from sklearn is used to create an artificial neural network that includes two hidden layers, with 500 neurons in each layer, and a logistic activation is used. This is then fitted with the x and y train data. A variable called predicted uses the predict function from sklearn with the x test data to make a prediction. Finally, the accuracy is calculated using metrics accuracy score from sklearn with the y test data and the predicted variable as the parameters. The accuracy is then returned from the function.



To plot the graph for the artificial neural network, I used the matplotlib library. I called the function with the number of epochs in the parameters and assigned it to a variable, the used the number of epochs as the x-axis and the accuracy as the y axis. I chose to plot 6 points, with the epochs being 500, 1000, 5000, 10000, 25000 and 50000. The results of the graph are presented below. The average for these is 65.83%.

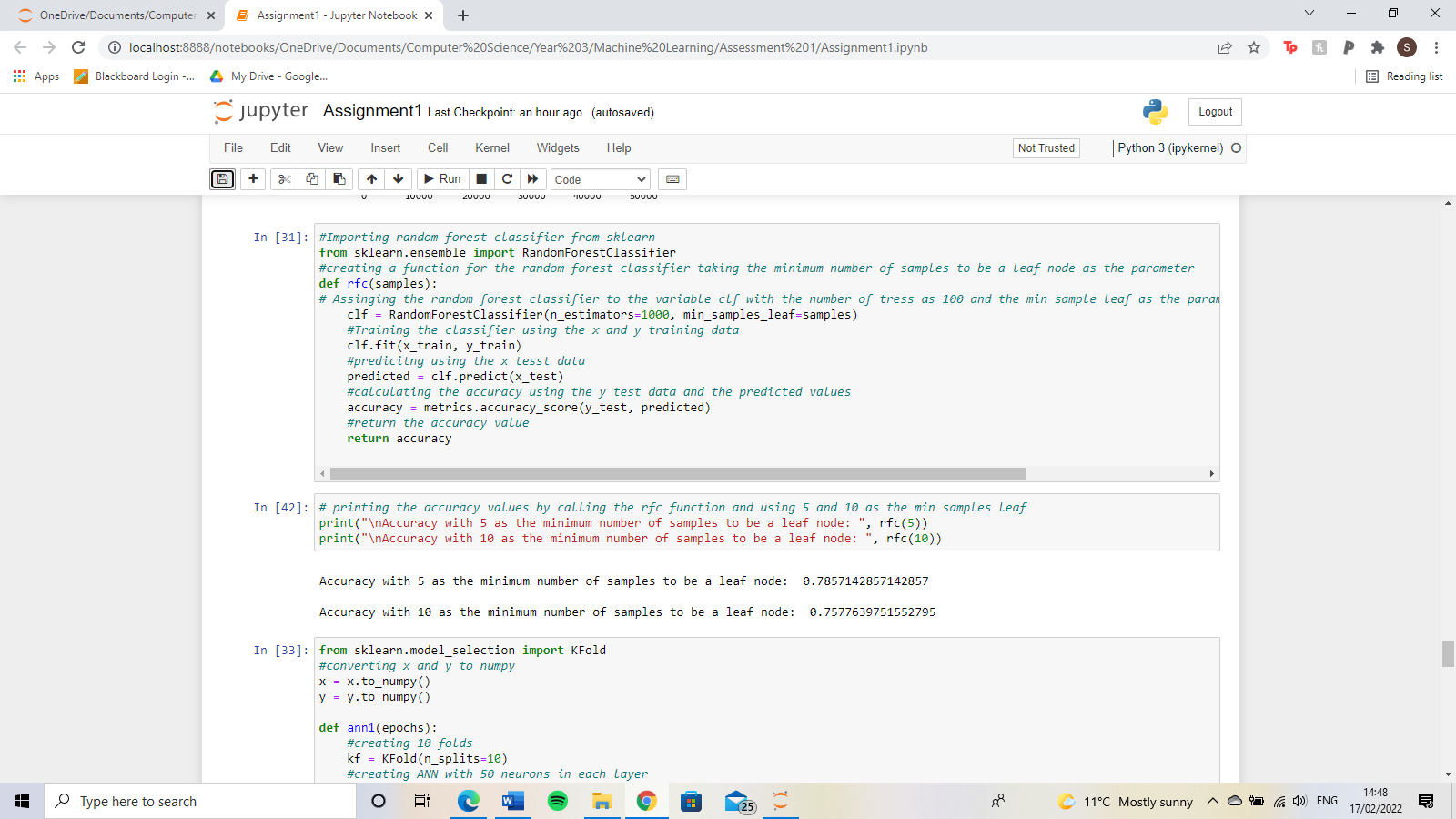
For the random forest classifier, I, first of all, created a function that takes the minimum samples required to be at a leaf node as a parameter. I then used random forest classifier



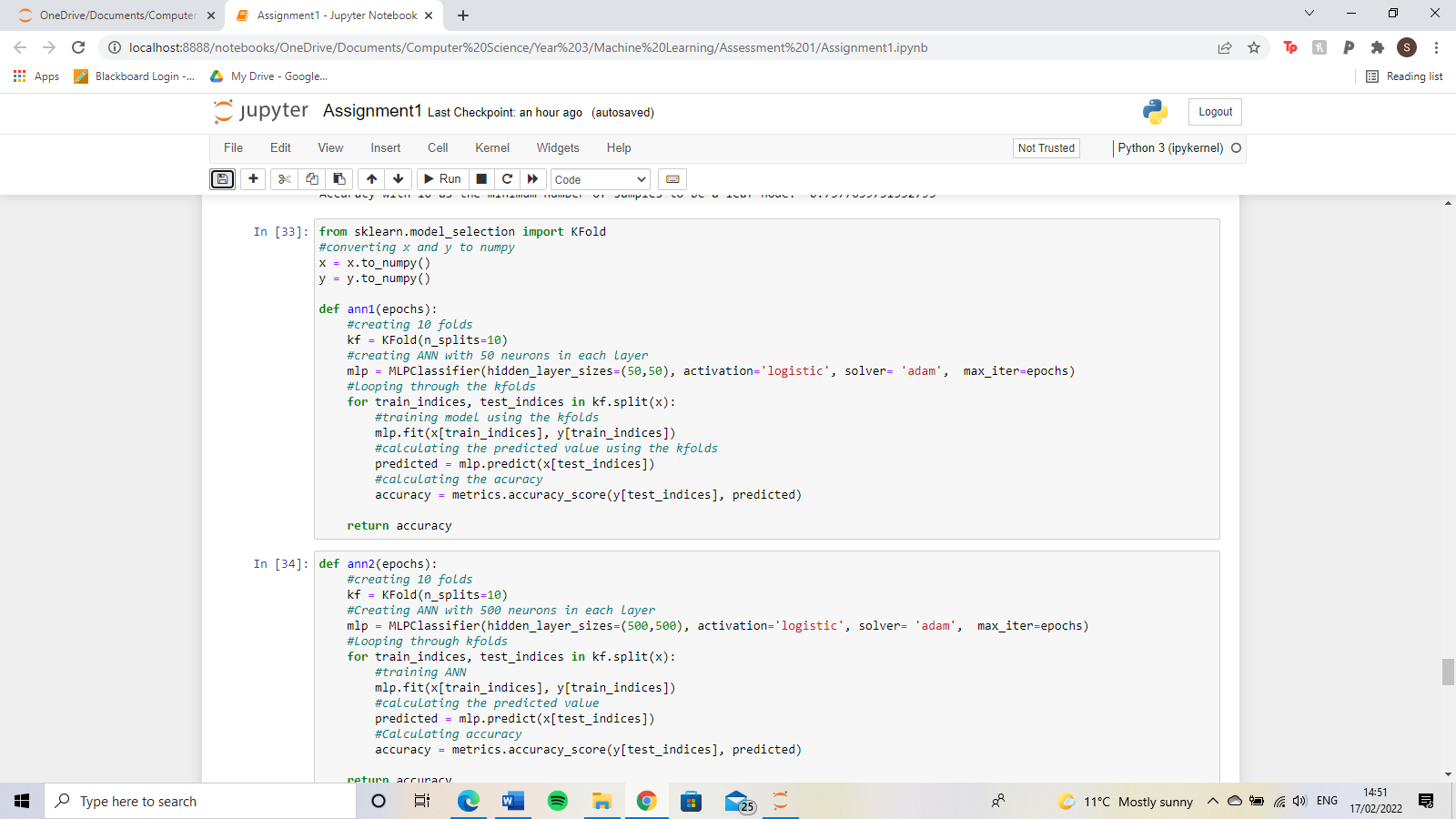
Graphical user interface, text, application

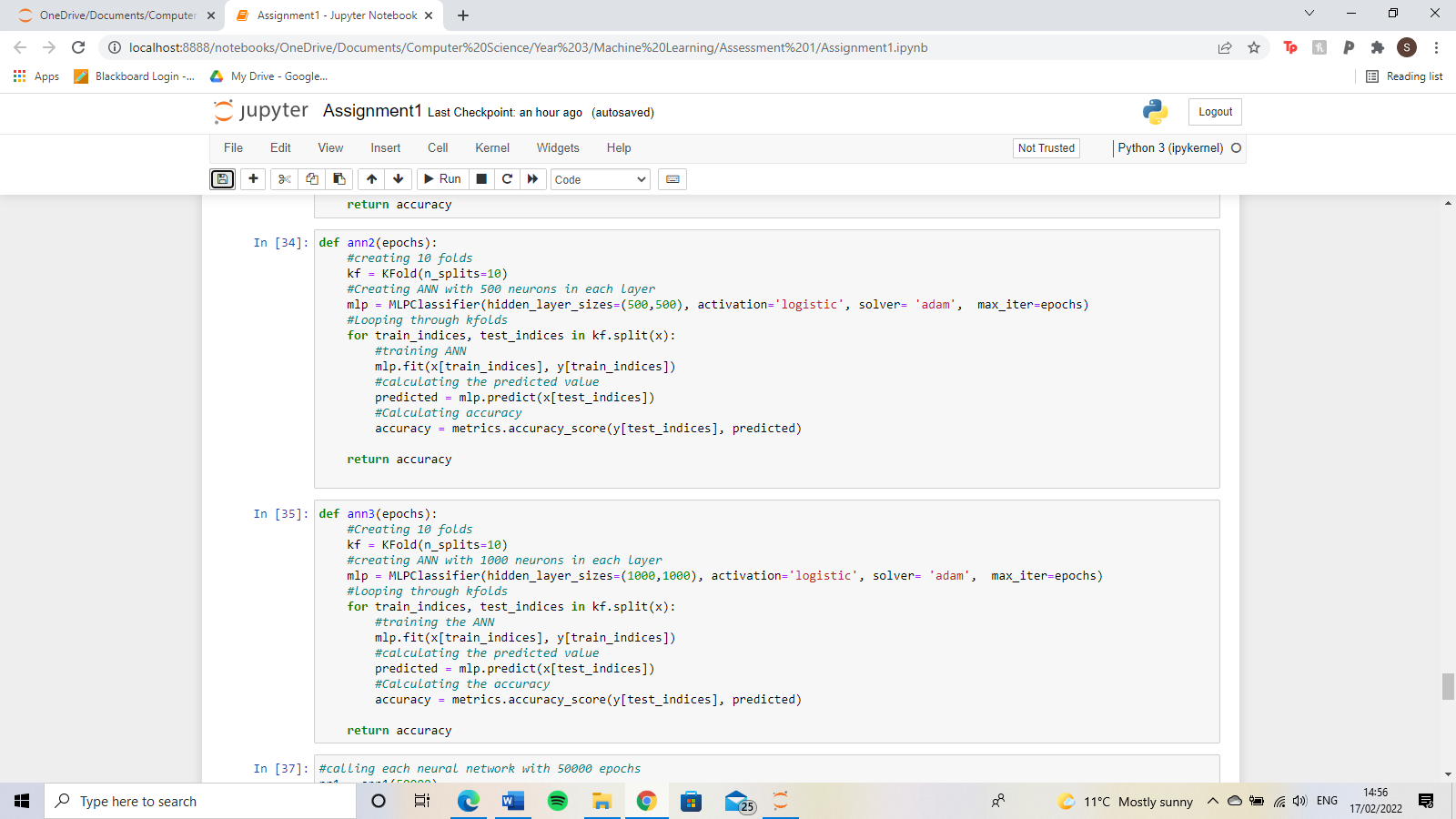
Description automatically generated

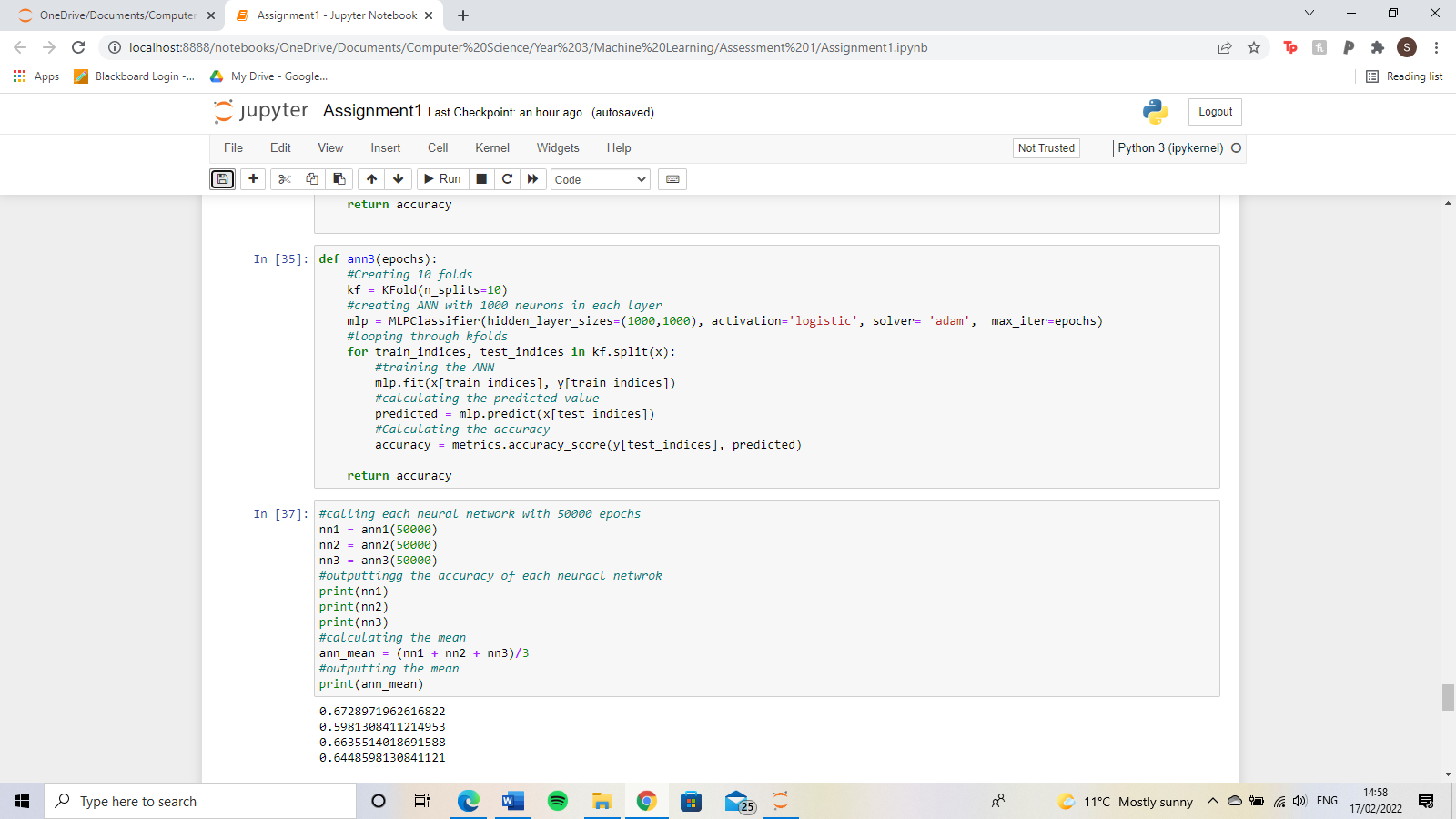
from sklearn with 1000 trees. Then using the fit function from sklearn to train the classifier with the x and y train data. Then predicting using the x test data. Finally to calculate the accuracy by using metrics accuracy score from sklearn with the y test and predicted as parameters. The accuracy is then returned from the parameters. The accuracy of the minimum number of samples to be a leaf node as 5 was 78.57142857142857%. The accuracy of the minimum number of samples to be a leaf node as 10 was 75.77639751552795%.



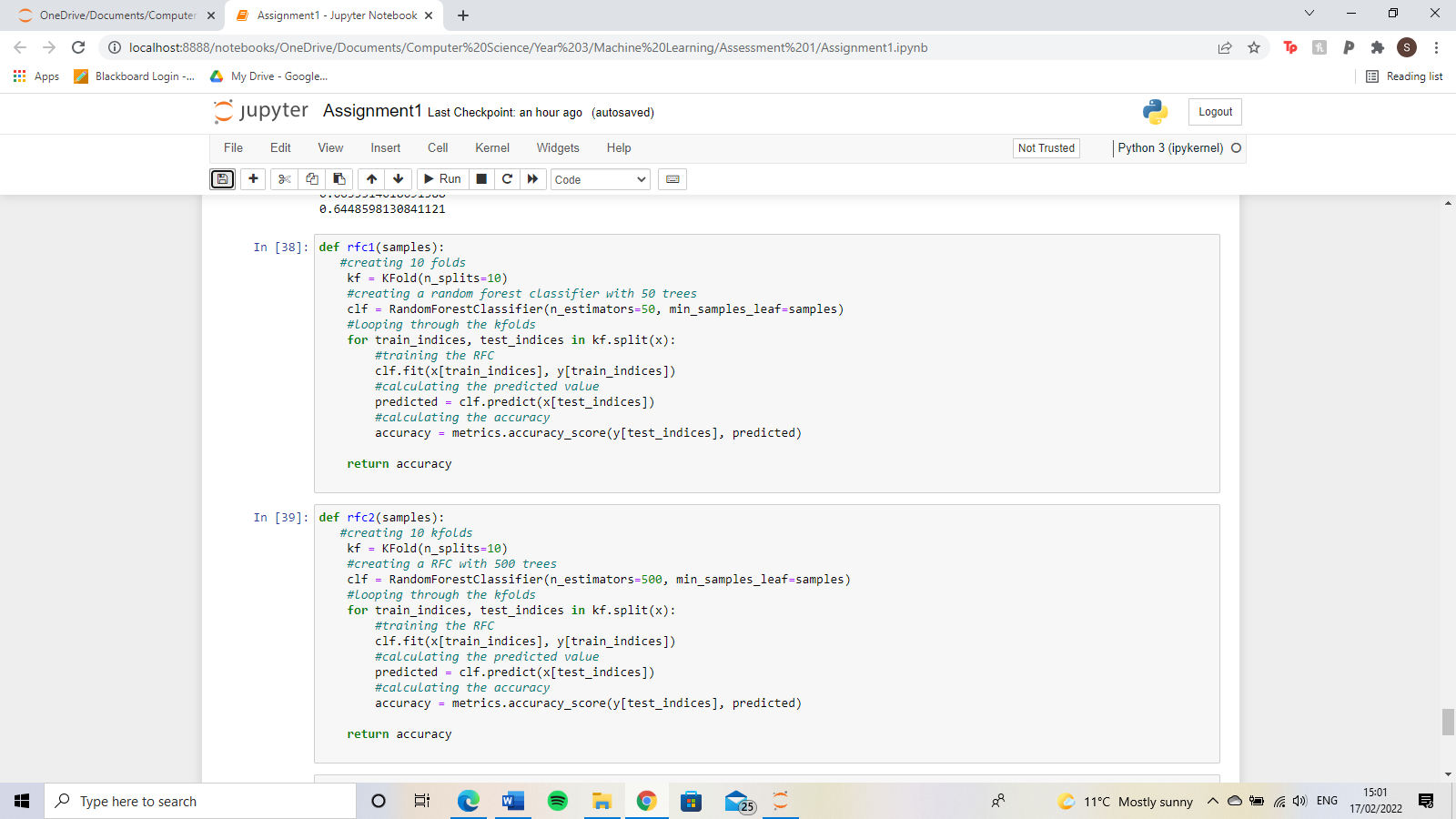
1. For the cross-validation using 10 folds, I used the KFold function from sklearn. The artificial neural network is set up like before however this time there are 2 hidden layers with either 50, 500 or1000 neurons in each layer. I chose to have 50000 epochs for each artificial neural network. Then looping through the split KFold for the x data, the artificial neural network is trained using the KFold x and y train, then predicted using the KFold x test data. With this, the accuracy is calculated with the y test and the predicted. The random forest classifier is done like the artificial neural network apart from using the random forest classifier function from sklearn with the number of trees varying from 50,500 and 10000 and the minimum number of samples to be a leaf node being 10. The training, predicting and accuracy is done the same as in the artificial neural network.

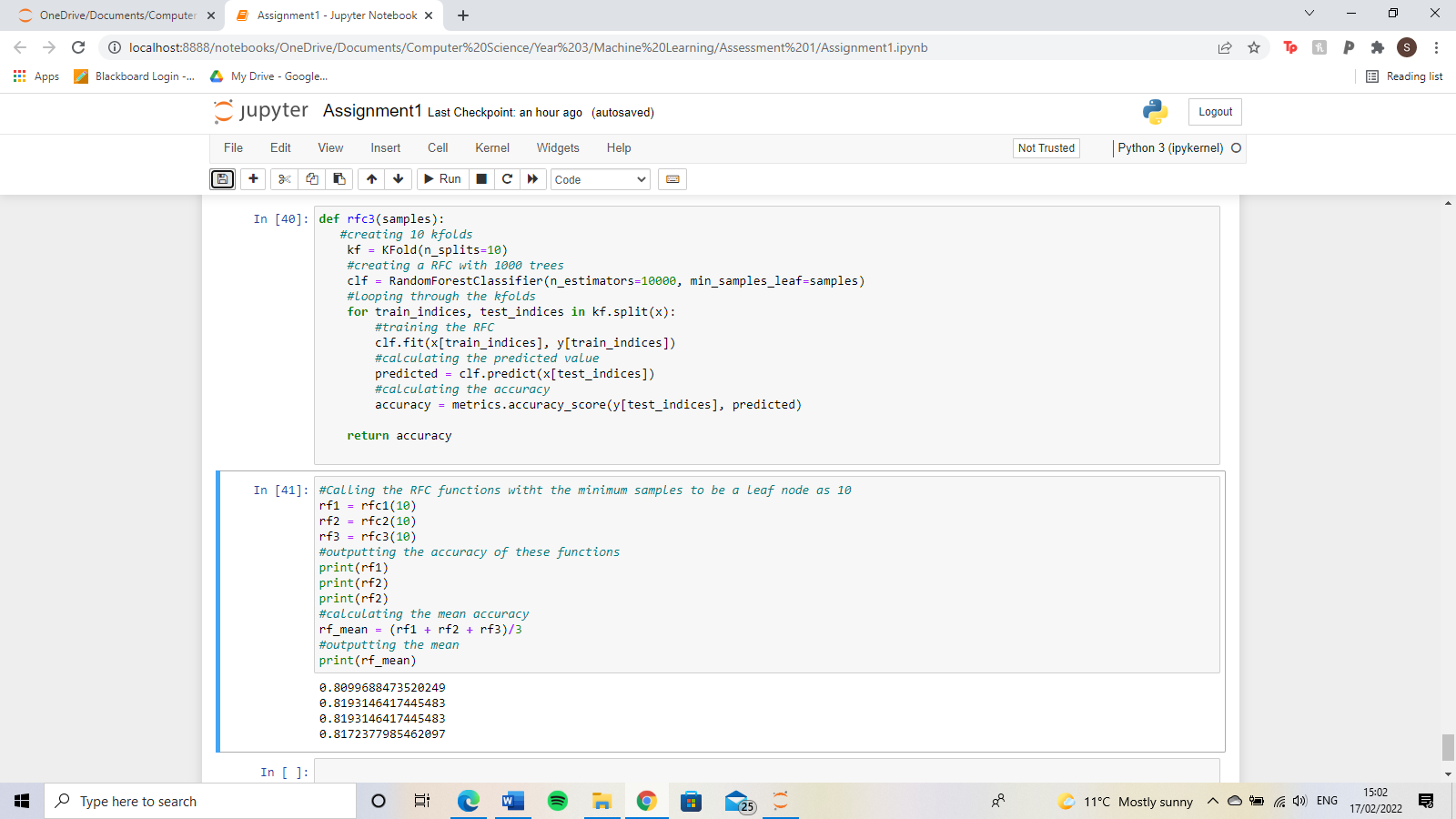






For the artificial neural network with 50 neurons in each layer, the accuracy was 67.28971962616822%. For 500 neurons, the accuracy was 59.81308411214953%. For 1000 neurons, the accuracy was 66.35514018691588%. The mean of the artificial neural network’s accuracy was 64.48598130841121%. I believe the best parameter for the artificial neural network is the number of epochs. The average for the number of epochs was just higher at 65.83% compared to the number of neurons per layer which was 64.49%. However, based on the graph, for the number of epochs over 10000, the accuracy increased as the number of epochs did. Compared to the number of neurons in each layer where there seems to be no pattern.





For the random forest classifier with 50 trees, the accuracy was 80.99688473520249%.

For 500 trees, the accuracy was 81.93146417445483%. Finally, for 10000 trees, the accuracy was 81.93146417445483%. This makes the mean of the random forest classifier to be 81.72377985462097%. The best parameter for the random forest classifier seems to be the number of trees. This was because the average accuracy for the number of trees was 81.7% compared to 77.1739130435% for the minimum number of samples to be a leaf node.

Based on the information I have collected; the random forest classifier was better than the artificial neural network. This is because, for both tests for each parameter, the random forest classifier had an accuracy of over 75% compared to the artificial neural network with around 65%. This would not be good enough when monitoring a patient with HIV as there are still 35% of patients who could be identified as a false positive or false negative which may put the patient in danger.