Name: Christopher Leung

UCID: cl48

Preferred email address: [cl48@njit.edu](mailto:cl48@njit.edu)

Option Number: Option 1

**CS 634 Final Project**

**General Information**

* Permission: Professor Wang has granted me permission to predict regression target variable instead of classification. He also checked the data set I provided him and allowed me to use neural networks.
* Algorithms used to predict regression: Neural Networks and Linear Regression
* Dataset link: <http://archive.ics.uci.edu/ml/datasets/Air+Quality>
* Software used for algorithm implementation: Python 3.6 using Jupyter Notebooks IDE from Anaconda

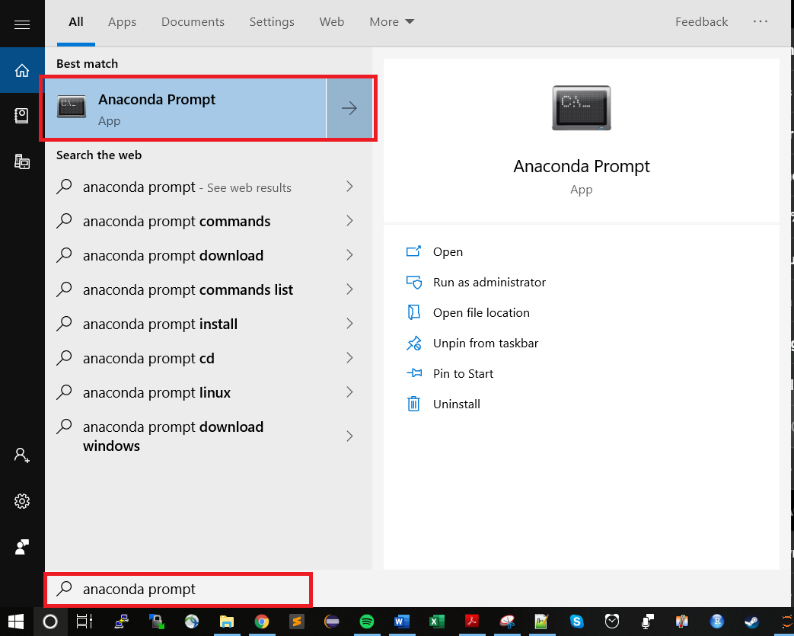
**Objective**

* The goal of this analysis is to measure the accuracy of two algorithms (Neural Networks and Linear Regression) on the Air Quality dataset I have chosen from the UCI website using the 10-fold cross validation. Additional details such as data preprocessing (cleaning data), snapshot of entire dataset, histogram of data variables, 10-fold cross validation and predicted values from both algorithms will be included in the next few sections.

**Software prerequisites**

* Anaconda/Jupyter Notebooks with Python: The installation of these programs can be found on this website (<https://jupyter.readthedocs.io/en/latest/install.html>) and (<https://www.anaconda.com/distribution/>). First Anaconda needs to be installed and within the Anaconda prompt, similar to windows prompt, you have to install jupyter notebooks using Anaconda scripts and also Python is installed with Anaconda.
* TensorFlow: This is a required application to install in order for neural networks to work. Steps are shown here in snapshots:

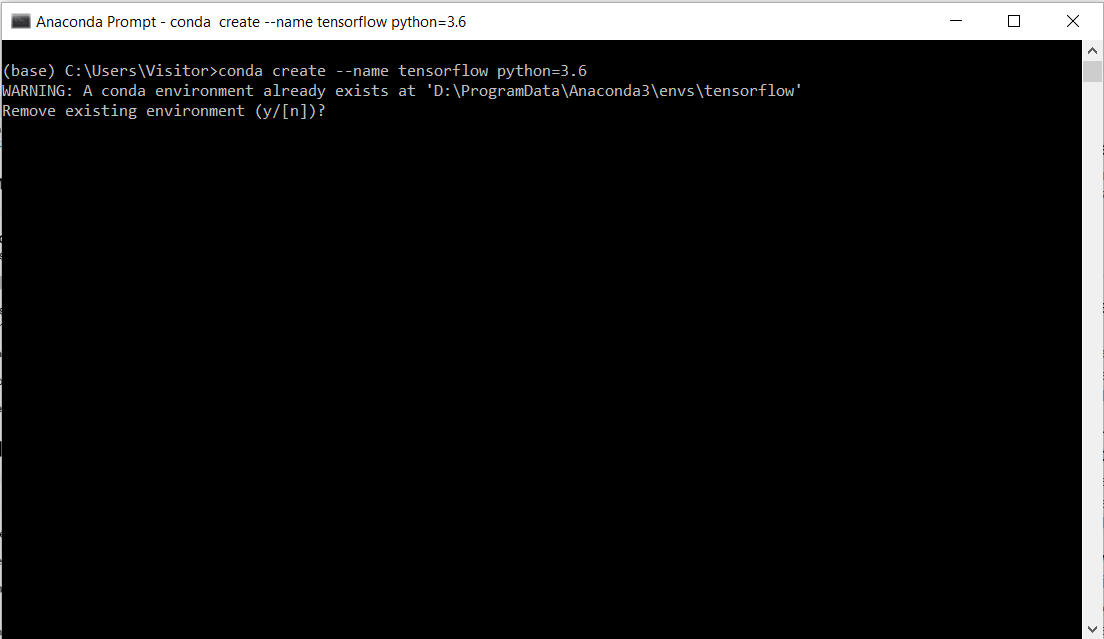
1. Once Python is installed with Jupyter Notebook in Anaconda environment, open Anaconda Prompt.



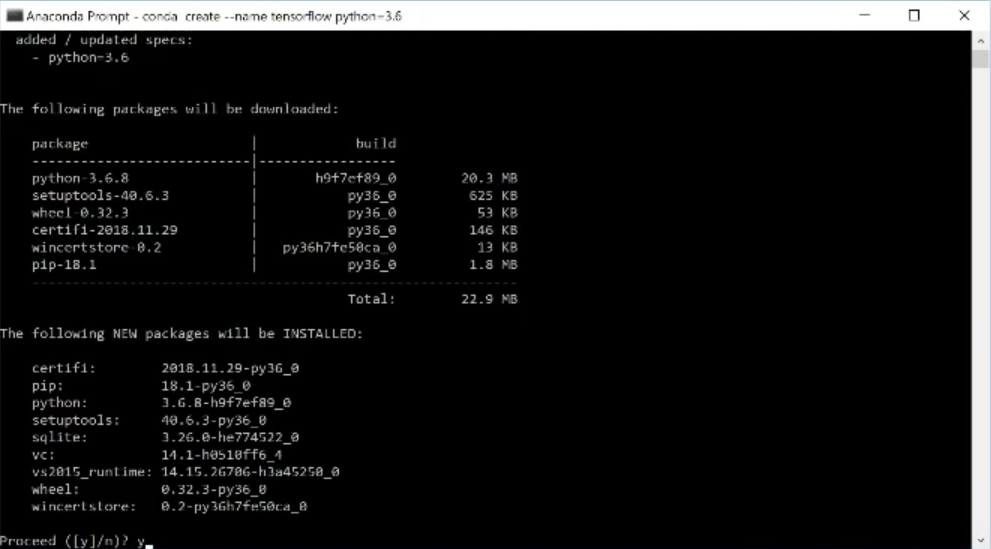
1. Once Anaconda Prompt has been opened, type in this command:

**conda create –name tensorflow python=3.6**

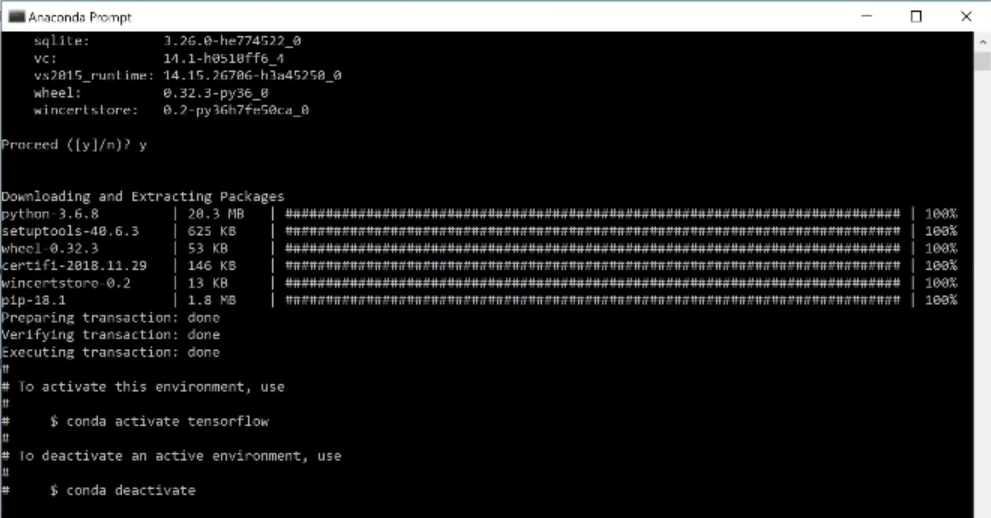
* This command installs a directory tensorflow within your Anaconda environment.



* I already have this installed but the screenshot below shows what you would usually see if you are first time installing tensorflow environment

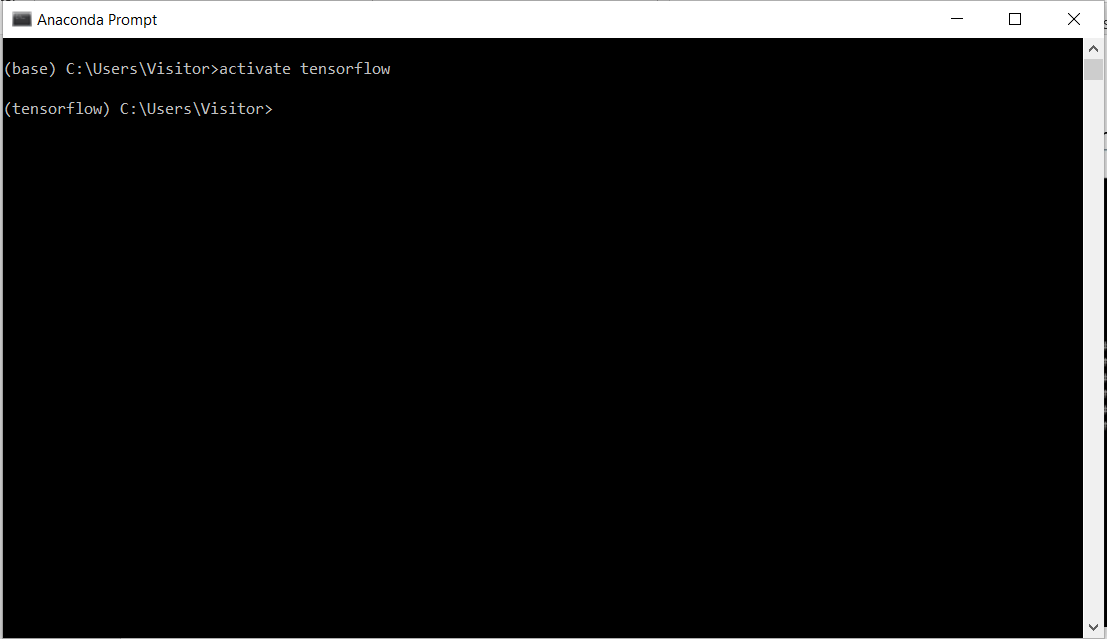


1. Type in y. Then it will look like it downloaded a few packages similar to screen below:



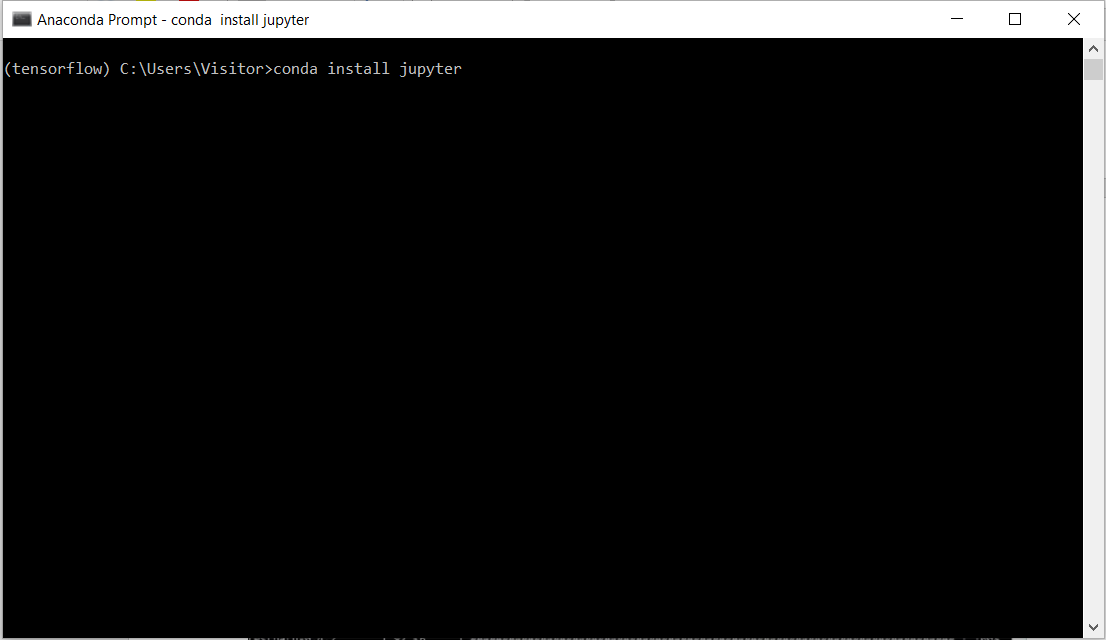
Command: **activate tensorflow**

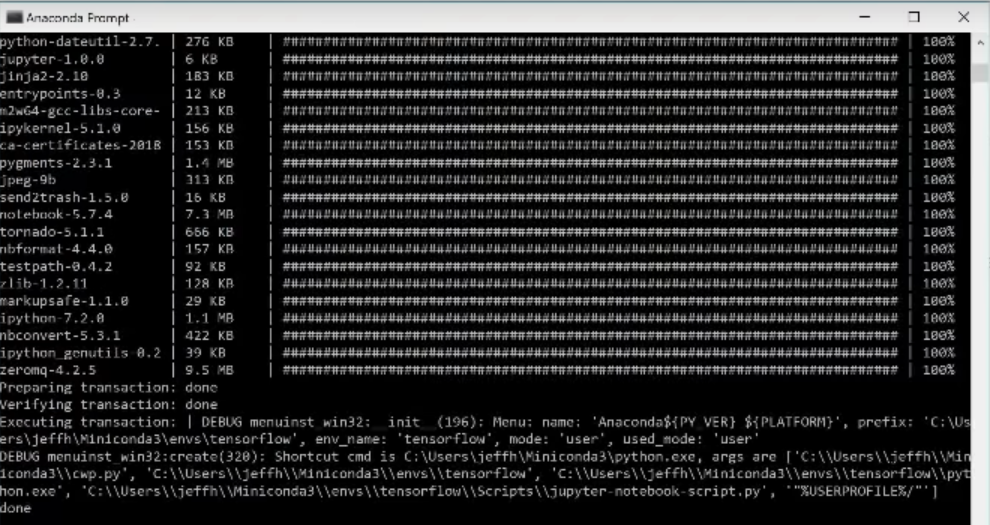
1. Type in the command and the result will look like this:



Command: **conda install jupyter**

1. Type in the command to install jupyter within the tensorflow environment you created.





Command:

**conda install scipy**

**pip install --upgrade sklearn**

**pip install --upgrade pandas**

**pip install --upgrade pandas-datareader**

**pip install --upgrade matplotlib**

**pip install --upgrade pillow**

**pip install --upgrade requests**

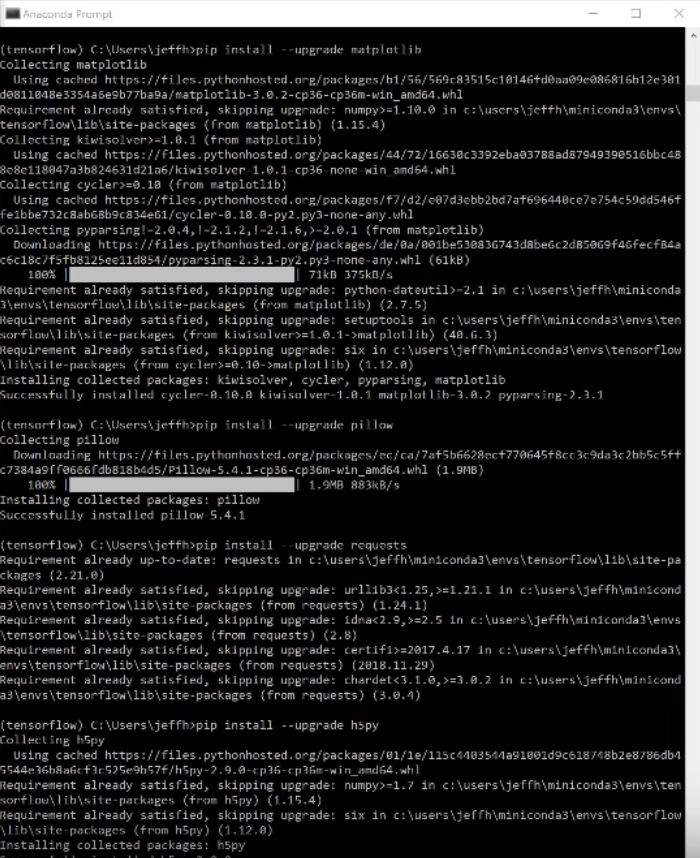
**pip install --upgrade h5py**

**pip install --upgrade psutil**

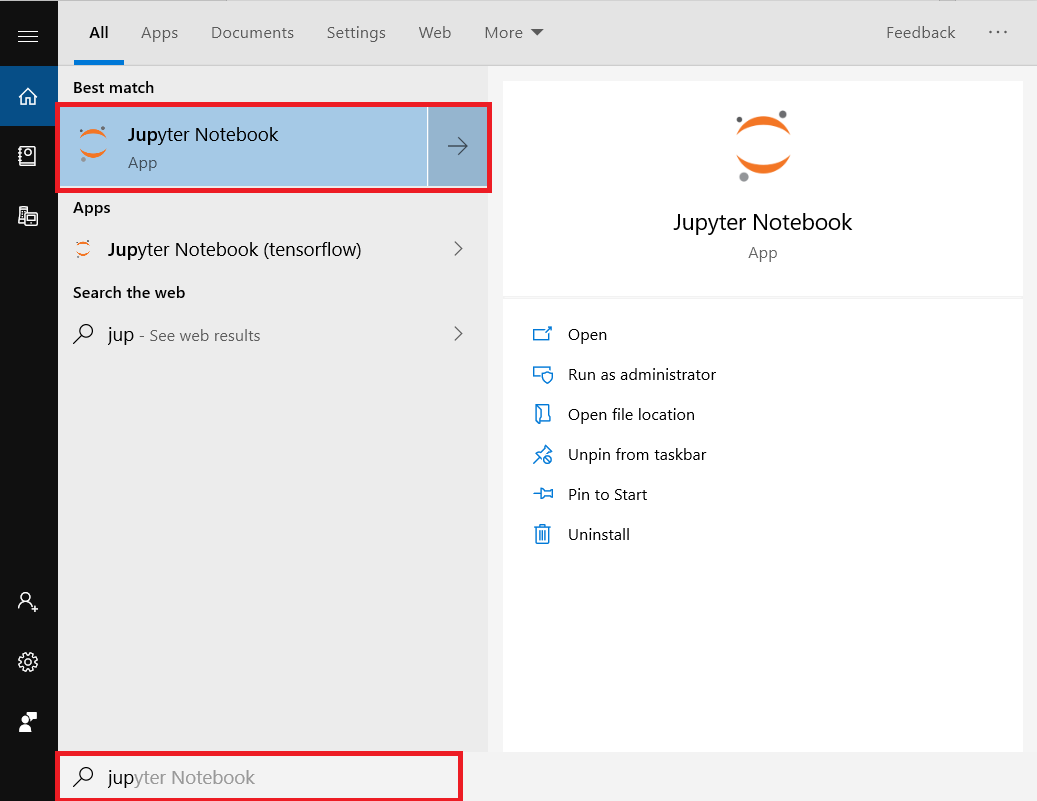
**pip install --upgrade tensorflow==1.14.0**

**pip install --upgrade keras==2.2.4**

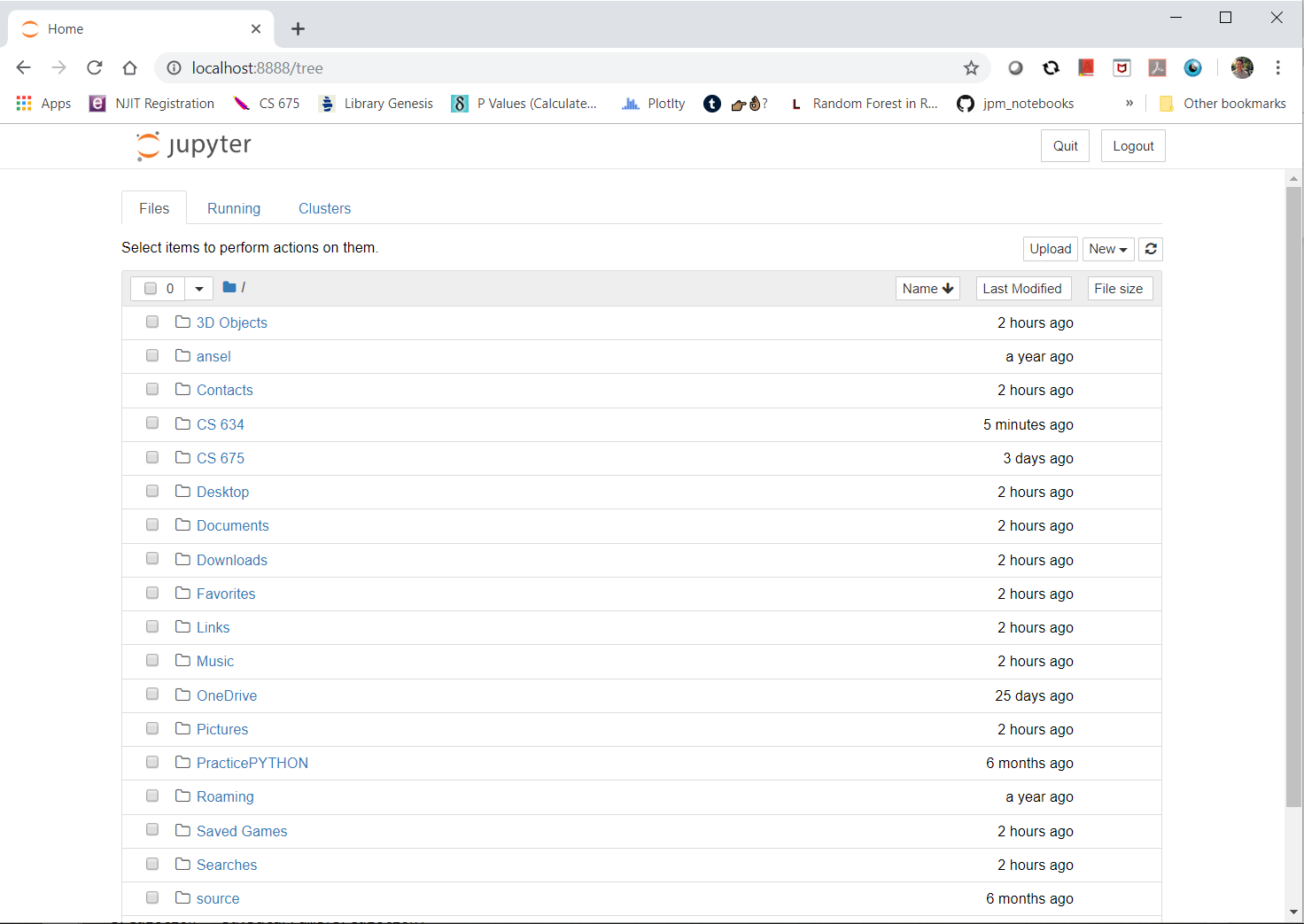
1. After jupyter notebooks has been installed in tensorflow copy and paste the following commands and the results should look like this:



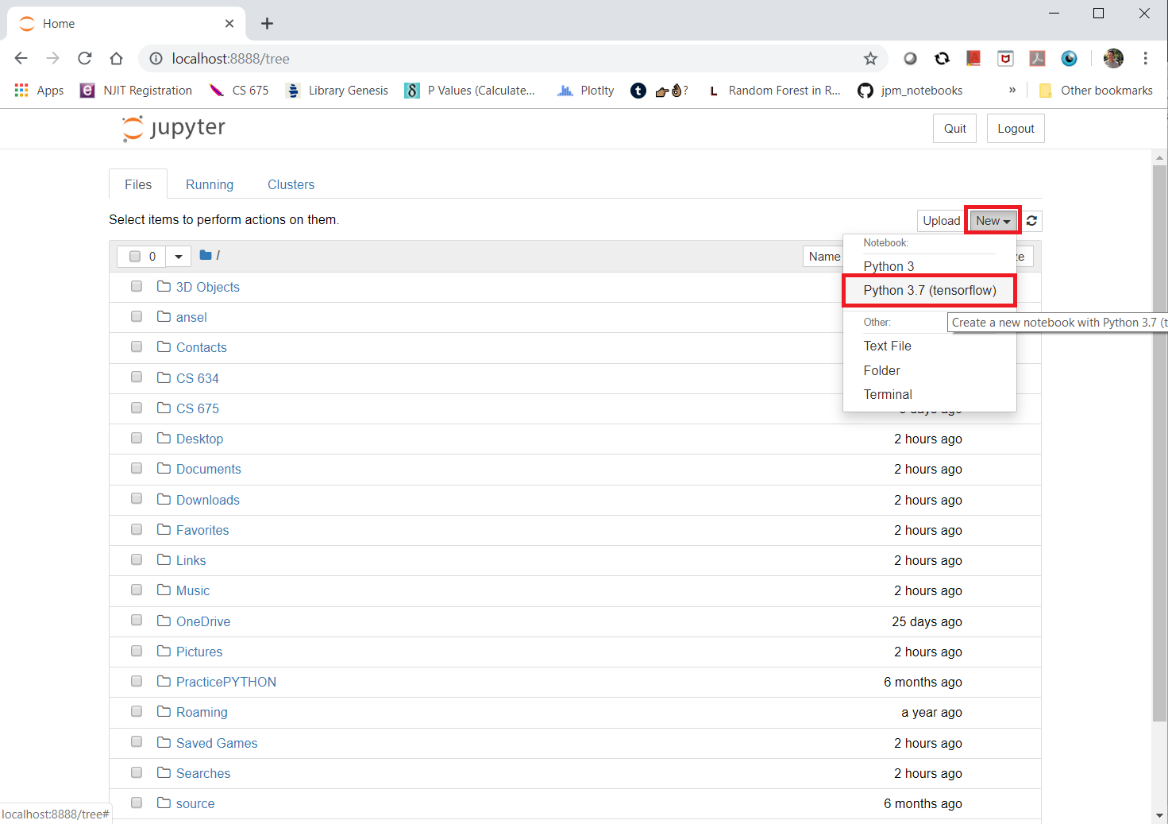
1. Once the packages have been finally installed, you should have tensorflow working properly. You can then close the Anaconda Prompt and open Jupyter Notebooks.



* A screen similar to below should pop up:

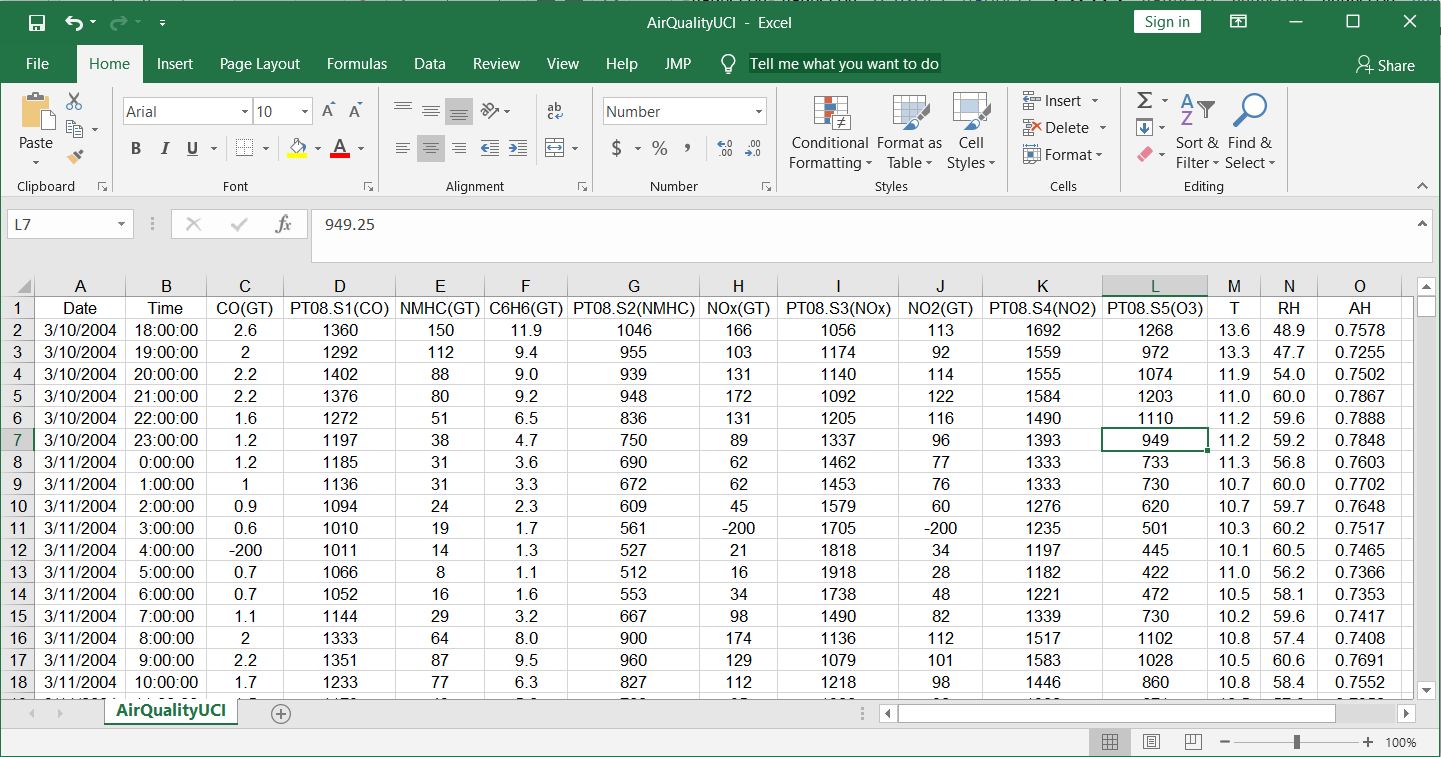


1. Click New and you should see a Python 3.7 (tensorflow) clickable icon.

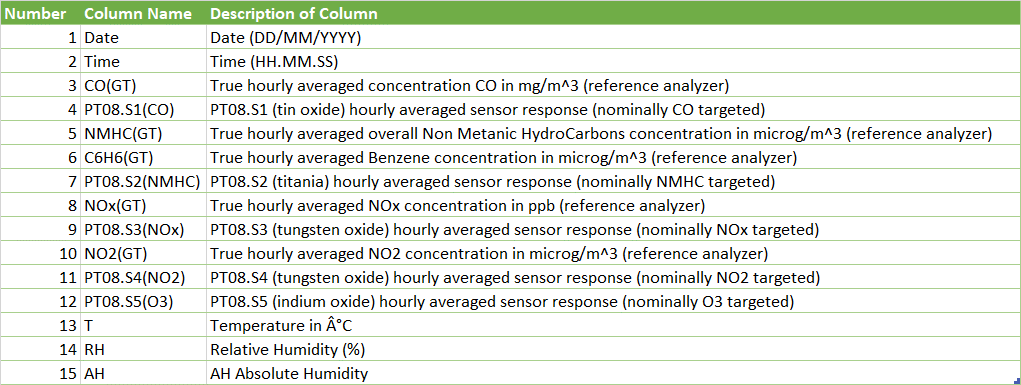


**Dataset Description**

* After downloading the Air Quality data set (<http://archive.ics.uci.edu/ml/datasets/Air+Quality>), it comes in a csv file as shown in screenshot below.



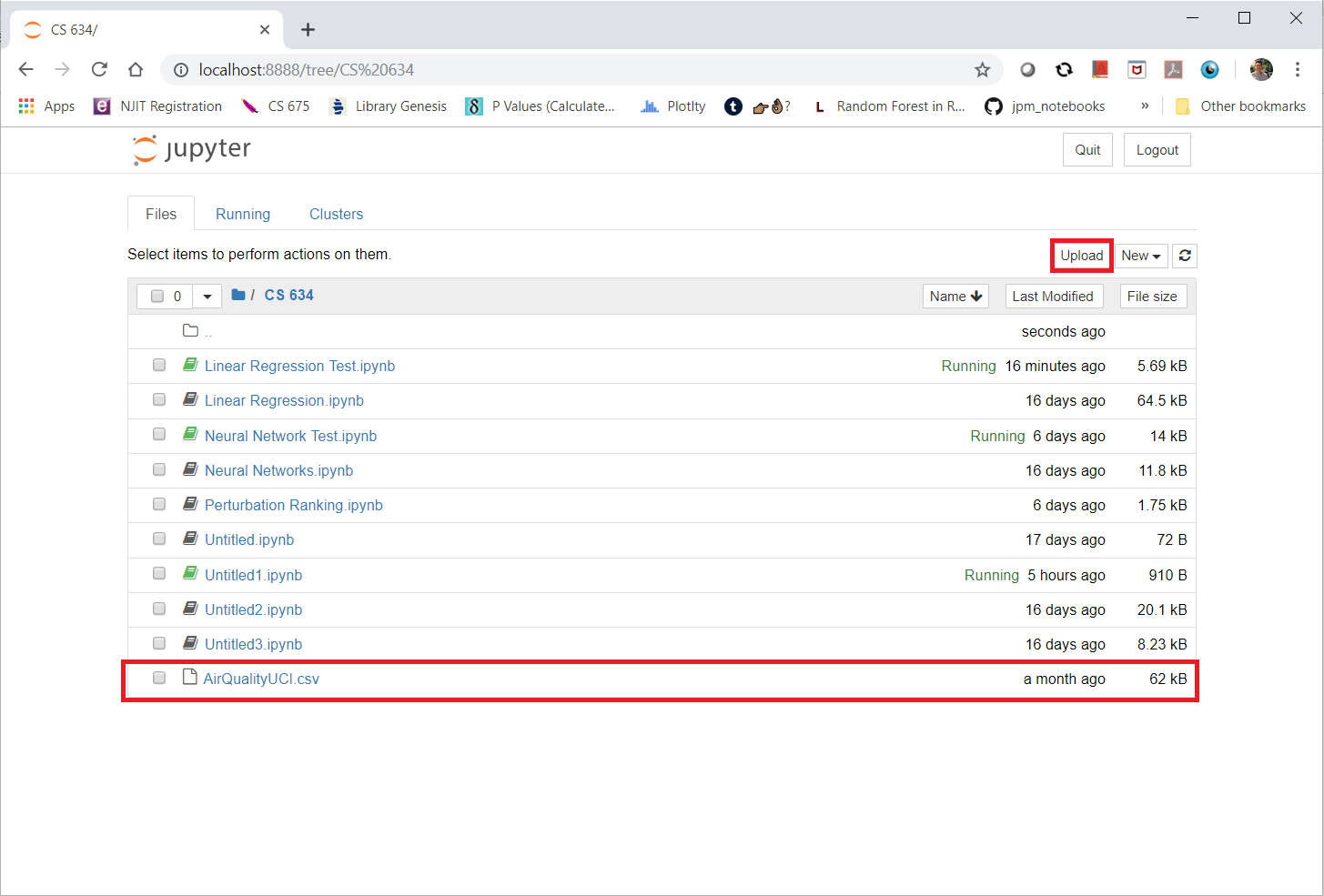
* The data set contains 15 columns and has 9,358 rows of data
* The descriptions of each column are below:



* The first two columns are simply the date and times for when each measurement has been taken.
* Dependent variables – columns numbered 3 to 12 and 14 to 15 (total 12 variables)
* Target variables – column 13 (one variable) and is measured in Celsius
* For this dataset it makes sense to predict regression for target variable temperature because we want to find out how gases and humidity levels affect the temperature.

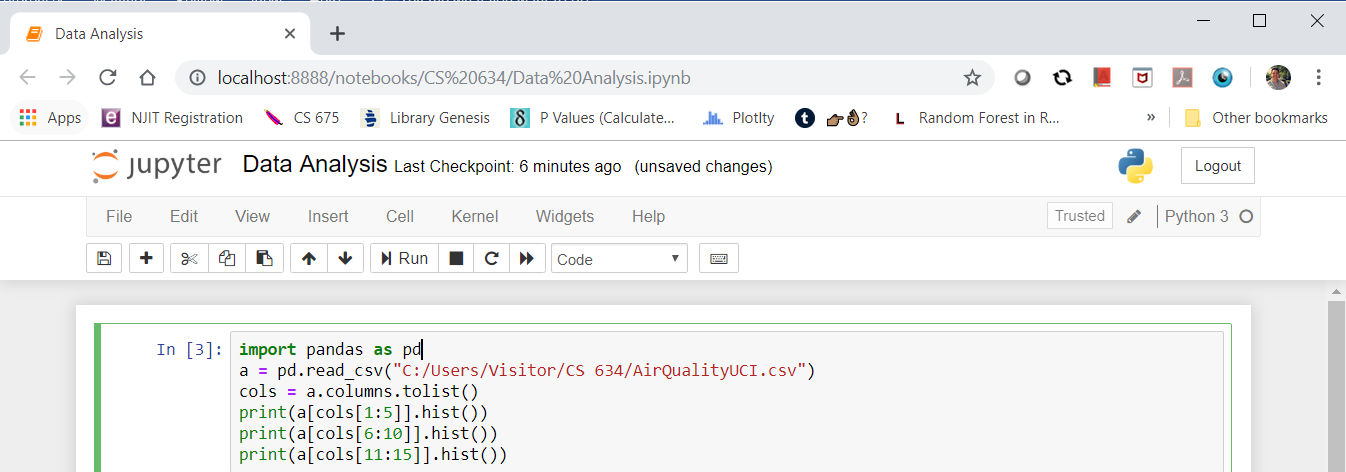
**Data Preprocessing**

* Before diving into the data and analyzing it or applying any algorithms to it, we want to note for any anomalies. According to the UCI website for the Air Quality data, missing values were tagged with a -200 value. Instead of including blank values or -200 values in my data, which would skew the analysis, I excluded all rows with -200 values from the entire data set.
* After removal of these rows, we are left with 827 rows of data from the original 9,358 rows of data. It is a bit disappointing to see we are left with less than 10% of rows from the original data set but keeping -200 or 0 values would not make any sense and would add a lot of sparsity in our analysis which becomes a problem with the algorithms.
* The next step is to upload the data set into Jupyter Notebooks. I uploaded the data already into my CS 634 folder and is highlighted with red boxes below:

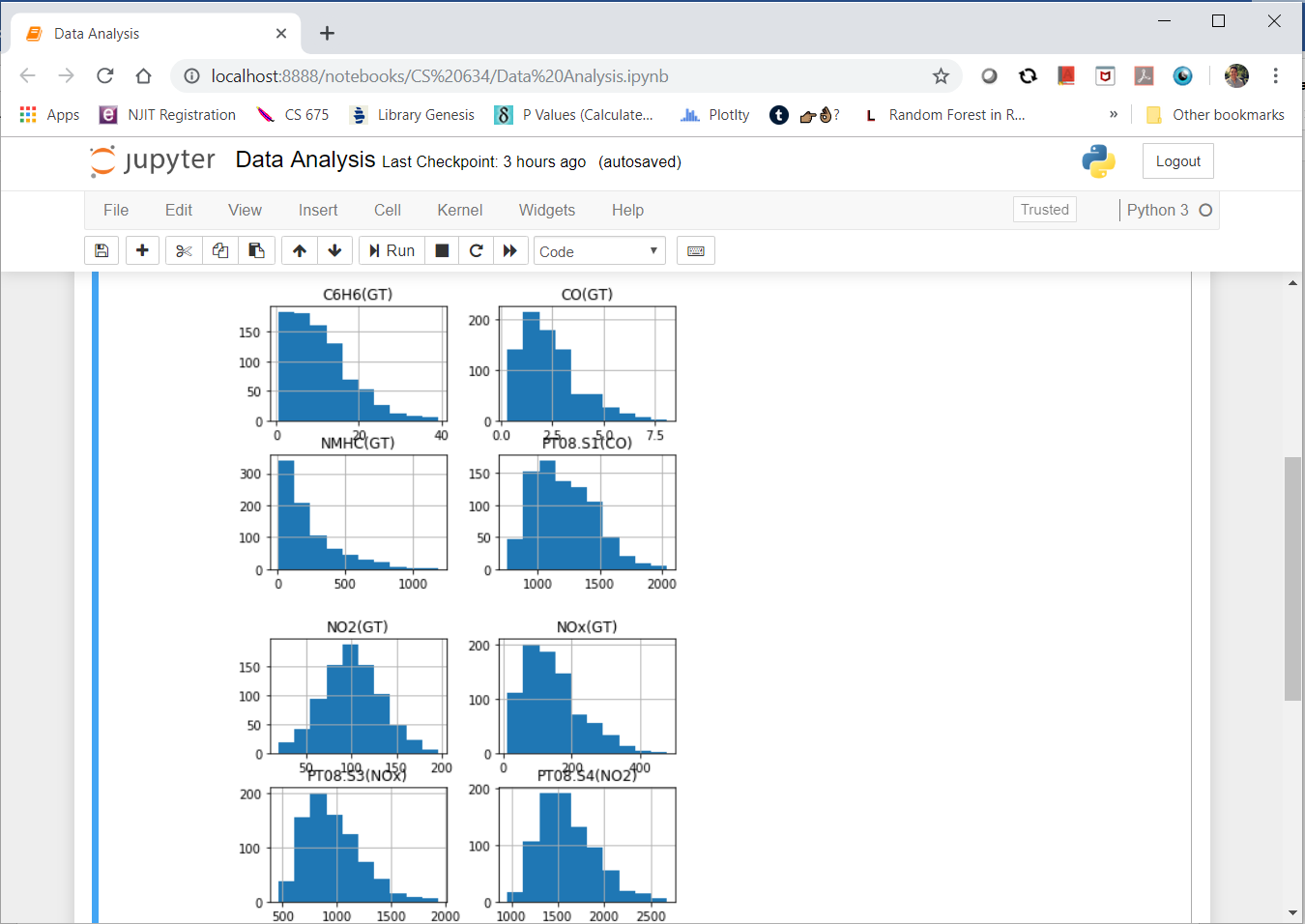


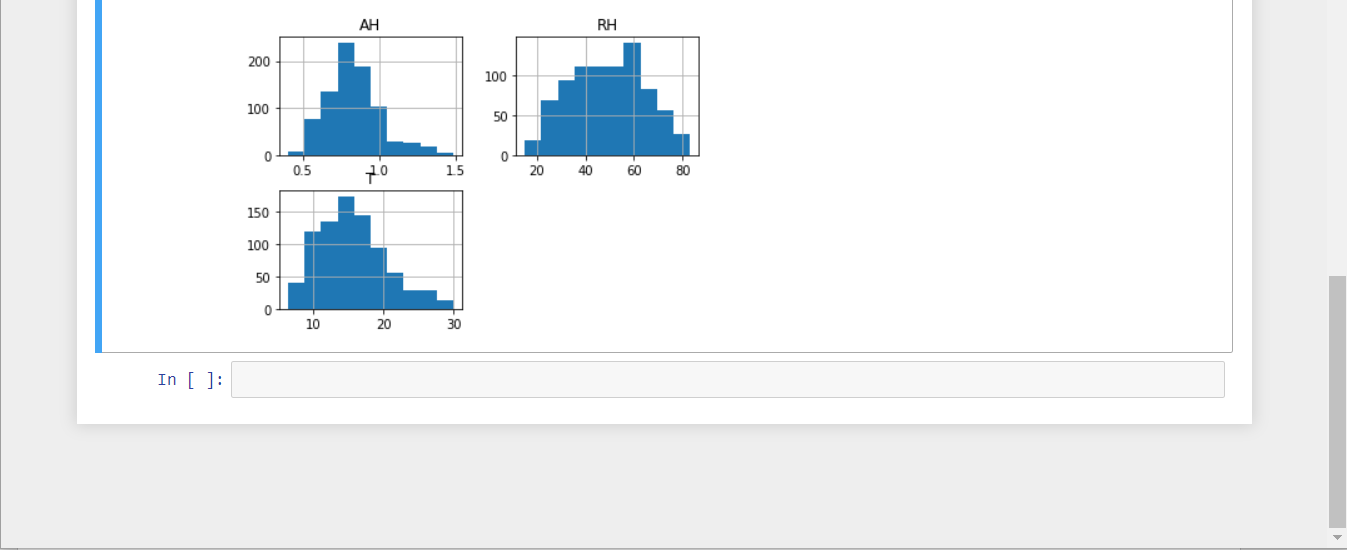
**Data Analysis**

* To better understand the data, I decided plotting histograms for all variables would help see where the min, max, average and any anomalies are.



* First, I imported pandas library from Python to read the csv file, this location is in “C:/Users/Visitor/CS 634/AirQualityUCI.csv”. Next, I used alias name “pd” for pandas because it is easier to type to call the library.
* The third line of code I used is **cols = a.columns.tolist()**. I created cols variable to get the names of the columns so when I pass the print statement, I can separately print out the histograms for 4 columns (dependent variables) at a time to make it neater because when I print out for all columns, it looks squished and overlap each other with their axis. Screenshot below.

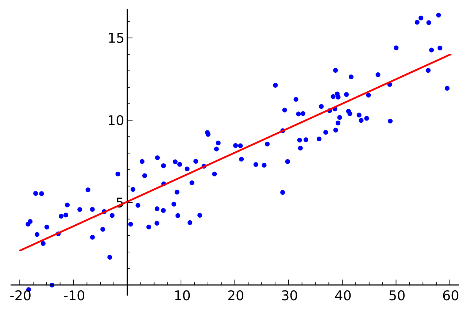




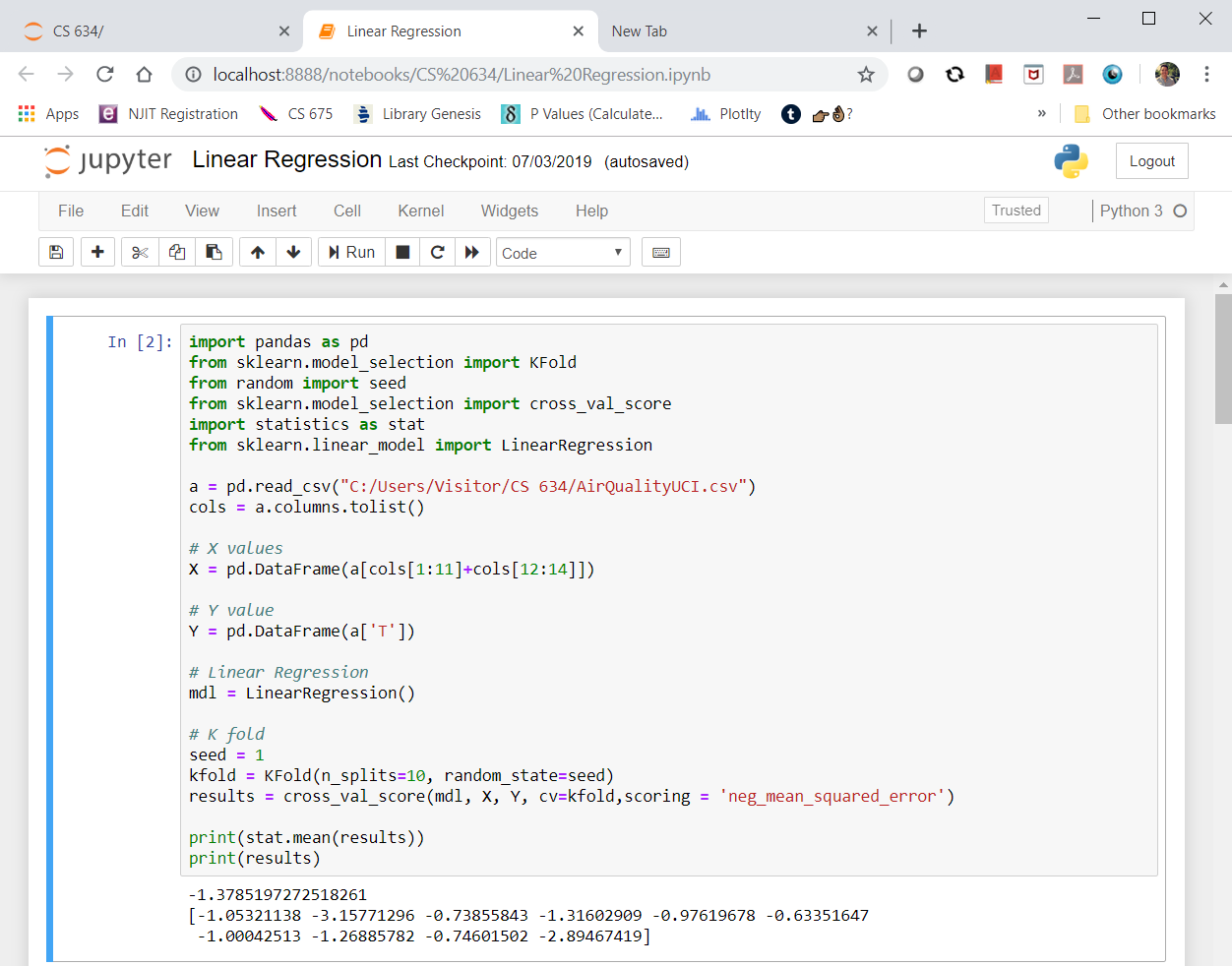
* Reviewing these histograms, there isn’t too many outliers or skewness except for columns C6H6(GT), CO(GT), NMHC(GT), and NOx(GT). The reason we review the histograms is to ensure data is good and for algorithms, garbage in garbage out. It is important to ensure the data is legitimate, does not have anomalies and makes relative sense. In our scenario, the data does make sense and doesn’t have many outliers or abnormalities.

**Linear Regression**

* Before diving into the linear regression algorithm, we must figure out what the linear regression is. Linear Regression is simply a line that fits between dependent variables and the target variable. A diagram is below:



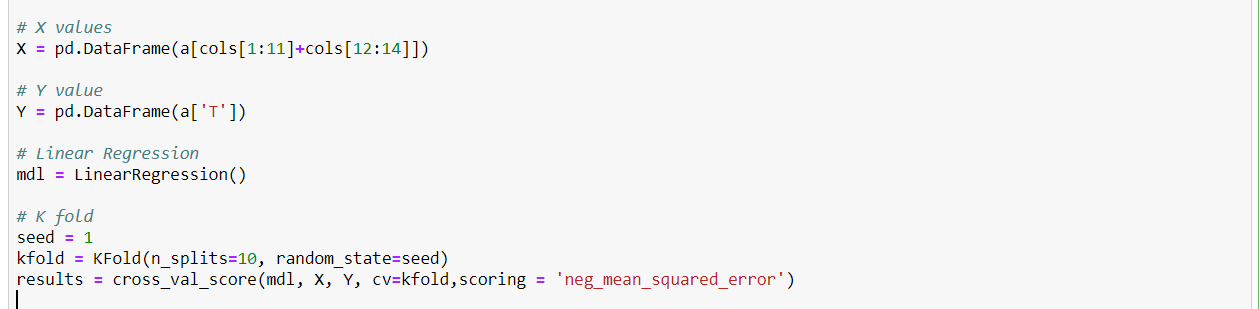
* Now that we have cleaned the data, it is time to put in our first algorithm – the linear regression.



* The first 6 line of code is to call out the libraries that I plan to use to allow linear regression to work.
  + **Import pandas as pd** = allows me to use DataFrame which is a table that makes the data neater than putting it in array
  + **from sklearn.model\_selection import KFold** = This is the 10-fold cross validation method to measure accuracy with data by splitting it into 10 parts with 1 testing data set and 9 training data set.
  + **from random import seed** = Randomizes a number and stores it to remember it. This is used for the KFold in order to split the data properly with random ordering.
  + **from sklearn.model\_selection import cross\_val\_score** = This allows for the KFold cross validation to provide a score, whether it is r^2 or MSE for each fold.
  + **import statistics as stat** = This will help me average out the cross\_val\_score to provide how accurate the model is.
  + **from sklearn.linear\_model import LinearRegression** = This library performs the entire linear regression operation.



* The next two lines of code was discussed before, it imports the csv file and cols stores the column names of the dataset (shown above).

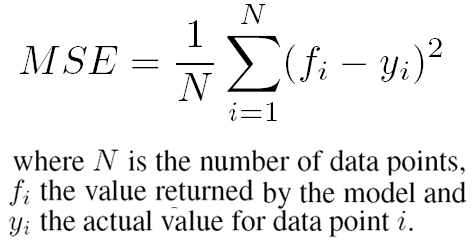


* **X = pd.DataFrame(a[cols[1:11]+cols[12:14]]).** This code stores the Dependent variables into a DataFrame from pandas library. Most of these variables will be used as X values for our linear regression model.
* **Y = pd.DataFrame(a['T']).** Our target variable is temperature so I declared ‘T’ from table a and converted it to a DataFrame.
* **mdl = LinearRegression().** This is the code used to start the Linear Regression model from the library sklearn.linear\_model.
* **seed = 1.** Initializes the number for randomization in kfold
* **kfold = KFold(n\_splits=10, random\_state=seed).** The KFold splits the dataset randomly into 10 equal sets.
* **results = cross\_val\_score(mdl, X, Y, cv=kfold,scoring = 'neg\_mean\_squared\_error').** This code allows for the Linear Regression model to run with dependent variables and target variables using the kfold split as 10 equal parts and I decided to use negative mean squared error. Unfortunately, there isn’t an option to use mean squared error for **cross\_val\_score** function but negative mean squared error is the same except the actual source code doesn’t flip the sign. Therefore, mean squared error = (-1)\*(negative mean squared error). This will be discussed in detail later.
* **print(stat.mean(results)).** This is the average result of the negative mean squared error for the 10-fold cross validation.
* **print(results).**  This is all the results of all the negative mean squared error for the 10-fold cross validation.

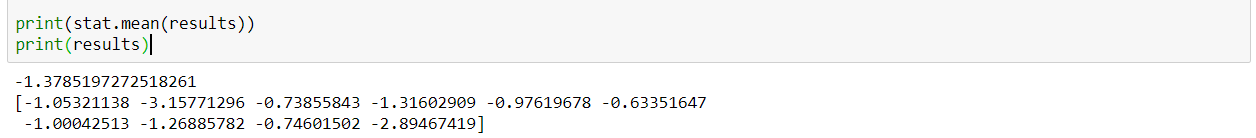
**Results of Linear Regression**

**10-Fold Cross Validation Using Mean Squared Error (MSE)**

* Before diving into the result and the meaning of the value, a general introduction to Mean Squared Error (MSE) would be necessary as this is not R^2 where values vary from (0,1). Mean Squared Error formula is below:



* Basically, MSE measures how different the predicted values are from the actual values. It measures the vertical spread of data around the regression line (in squared vertical units).



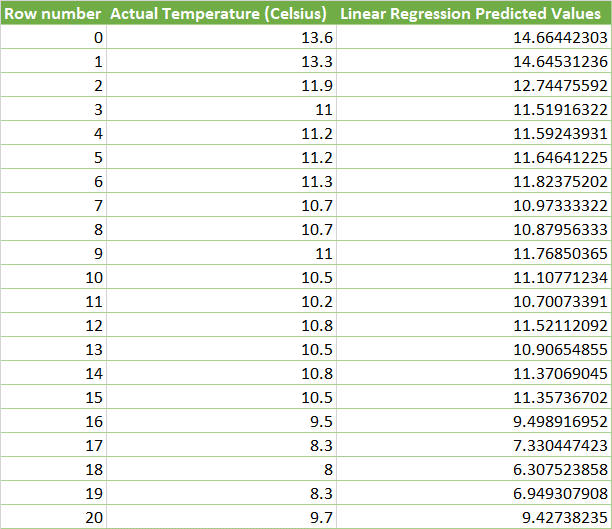
* From the Linear Regression average MSE result [(-1)\*(-1.3785) = 1.3785], we can tell this is a fairly low error rate and shows the model is relatively accurate when predicting the temperature.
* In the next analysis, I will talk about finding predicted values for temperature using the entire dataset instead of the 10-fold cross validation.

**Secondary Analysis**

* The reason I decided to analyze this was to get a better feel for how accurate the model is in predicting regression. Since MSE is a metric that does not provide high level meaning as it is just a value, I decided to get the model’s actual predicted values for the target variable to give us a better view as to how accurate the model is. Since obtaining the predicted values for the linear regression model is a bit complicated for 10-fold cross validation, I ran the entire dataset as the training data set and test data set just to see if the experimental result would provide value. The code is below:

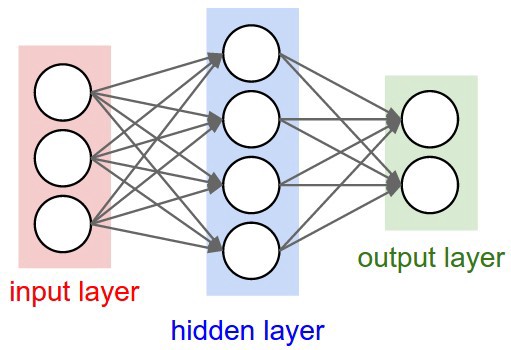


* **mdl.fit(X.values,Y.values).** The mdl.fit allows for the fitting of X dependent variables with Y target variable. X.values and Y.values simply transform the data from DataFrame to an array, to ensure there wouldn’t be an issue with the fit function.
* **prediction = mdl.predict(X.values).** This is the prediction for target variable Y using all the X dependent variables from data set.
* **prediction = pd.DataFrame(prediction).** I reconverted the prediction from an array into a table because it would be cleaner and easier to look at.
* I exported the results in excel and below is a sample of the differences.

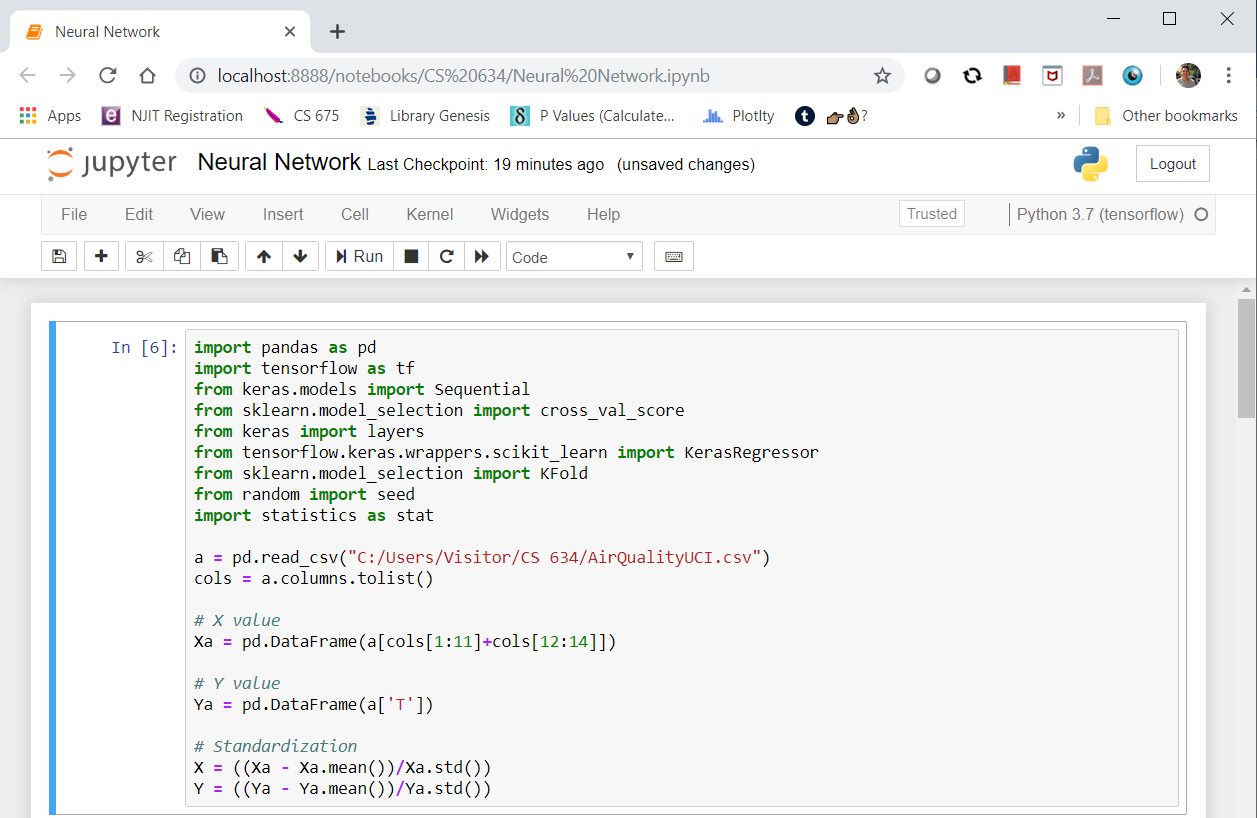


**Neural Network**

* Before proceeding to the algorithm code and the results, I will provide general information on what Neural Networks are.
* A diagram of the Neural Network is below:



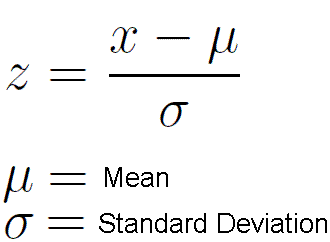
* Neural Networks contain 3 different types of layers (input layer, hidden layer, and output layer).
* The input layer brings the initial data into the system for further processing by subsequent layers of artificial neurons, the hidden layer is where the artificial neurons take in a set of weighted input and produce an output through an activation function. The output layer is the last layer of neurons that produces given outputs for the program.
* The activation function defines the output given the set of inputs from the output layer. With our data set we want to use linear regression as the activation function.
* Neural networks also use forward propagation and back propagation. Forward propagation is the process of feeding a Neural Network with a set of inputs to get their dot product with their weights then feeding the latter to an activation function and comparing its numerical value to the actual output.
* The Back propagation is a method used in artificial neural networks to calculate a gradient that is needed in the calculation of the weights to be used in the network. Which is used in our algorithm.
* The Neural Network algorithm is below:



* **import tensorflow as tf.** This command is used to initiate the tensorflow library that was installed earlier from the Anaconda Jupyter Notebook environment.
* **from keras.models import Sequential.** Sequential is to initialize the neural network, keras was created by Google engineers which is a high-level neural network API that is written in Python.
* **from keras import layers.** Creates the layers in a Neural Network.
* **from tensorflow.keras.wrappers.scikit\_learn import KerasRegressor.** This enables the sci-kit learn workflow in Python to regression for our Neural Network.
* **from sklearn.model\_selection import KFold.** This is to create the 10-fold as mentioned before.
* **from random import seed.** This is to randomize the 10-fold mentioned before.
* **from sklearn.model\_selection import cross\_val\_score.** Calculates the MSE for the 10-fold cross validation, as mentioned before.
* **import statistics as stat.** This library is used to measure average for the 10-fold cross validation MSE results.

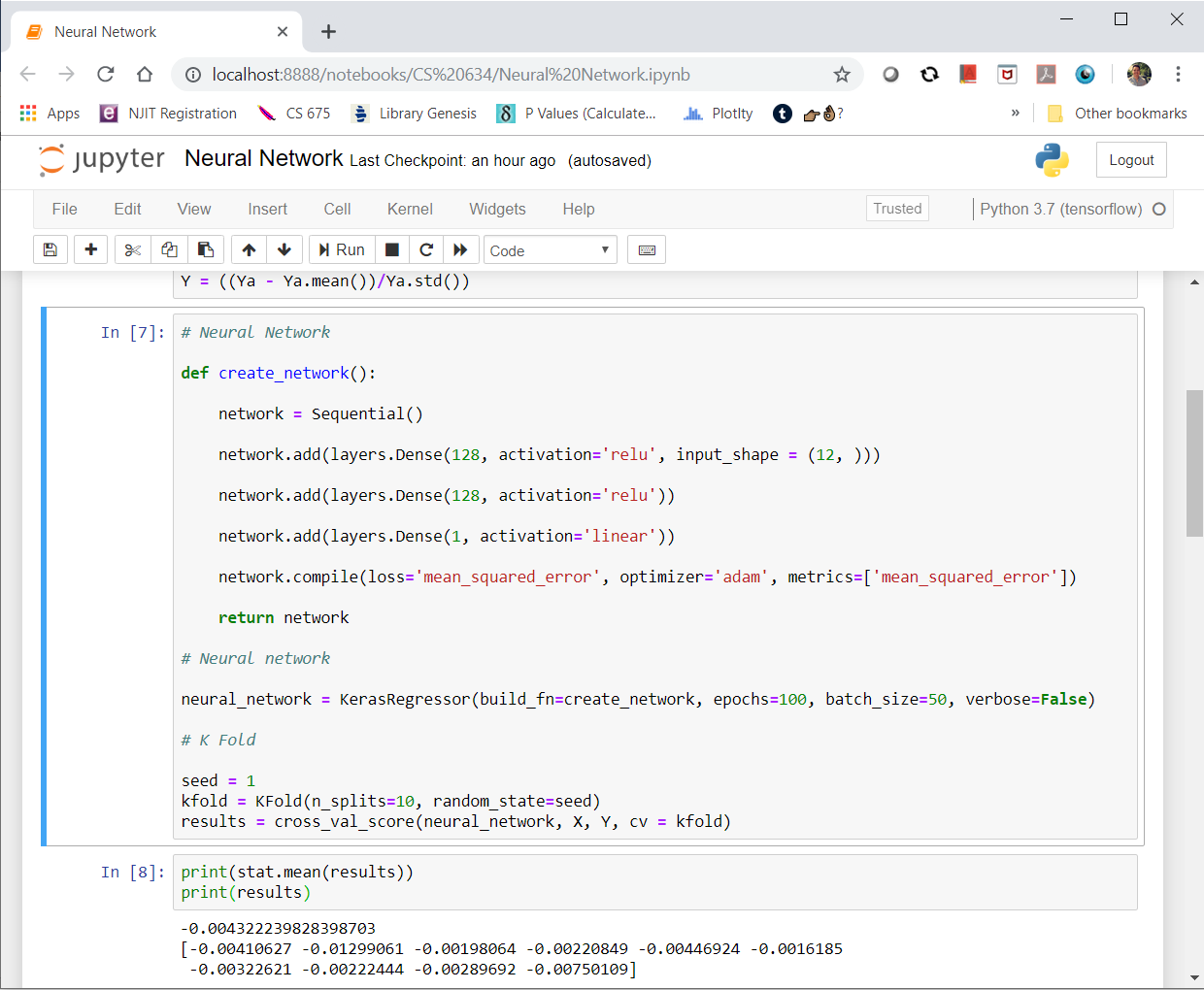
After activating the libraries, we have to import pandas to convert the csv file into a neat data table and create a cols name so I can call dependent variables or target variable.

* **Xa = pd.DataFrame(a[cols[1:11]+cols[12:14]]).** This is to create a neat table in dataframe for dependent variables.
* **Ya = pd.DataFrame(a['T']).** This is to create a neat table in dataframe for target variable.
* **X = ((Xa - Xa.mean())/Xa.std()) and Y = ((Ya - Ya.mean())/Ya.std()).** These codes are transforming the data into standardize form, without this step, our algorithm may fail to be efficient and accurate. The .mean() and .std() operations apply the mean and standard deviation for all the columns, not just one or two values.

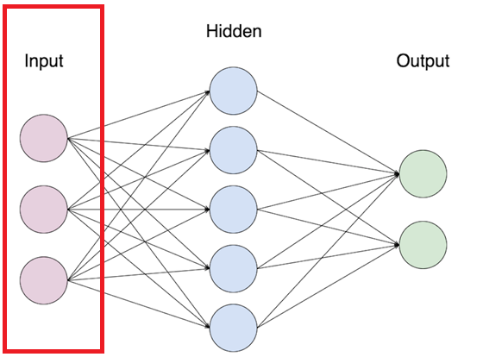


**Further notes**

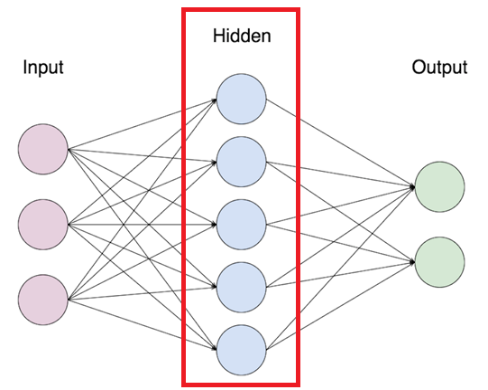
* The reason for standardizing the data before applying a Neural Network is because a Neural Network uses gradient descent in its calculations, if the gradient reaches local minimum instead of global minimum then the predicted values would be not be optimal, efficient or precise. More information on this is provided in the website (<http://www.faqs.org/faqs/ai-faq/neural-nets/part2/>).
* The next step is to create the Neural Network model in screenshot below:



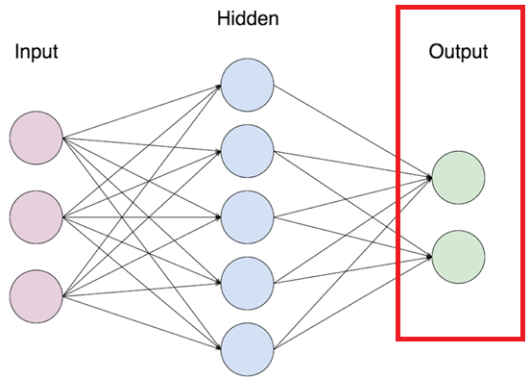
* **def create\_network():** I created a function called create\_network which will contain the Neural Network layers and activation function.
* **network = Sequential().**  This is to initialize the Neural Network.
* **network.add(layers.Dense(128, activation='relu', input\_shape = (12, ))).** The network.add() is a function that allows the addition of layers, layers.Dense() allows you to create any number of nodes for the input layer. In this scenario, I felt it was appropriate to contain 128 nodes for the input layer as highlighted in below diagram. The activation function for this would be ‘relu’ which is Rectified Linear Unit for the input and hidden layers to allow the Neural Network complex weights as opposed to not applying it which will make it a linear regression, which isn’t what we want. The input\_shape = (12, ) is the number of columns for the dependent variable.

****

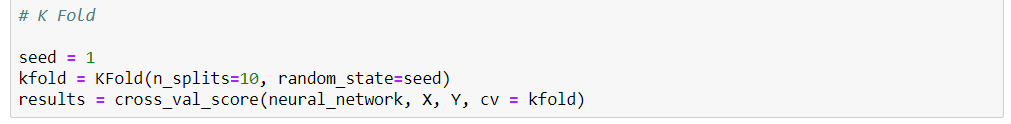
* **network.add(layers.Dense(128, activation='relu')).** This code is using the same layer and number of nodes (128) as the code before except this will be considered the hidden layer.

****

* **network.add(layers.Dense(1, activation='linear')).** This is the output layer and we use activation function as linear regression to predict the target variable (temperature).

****

* **network.compile(loss='mean\_squared\_error', optimizer='adam', metrics=['mean\_squared\_error']).** Once the layers have been created, the network.compile() statement will compile the Neural Network and the parameters are the loss function, the optimizer and MSE as the metric to measure for accuracy. I used optimizer = ‘adam’ because this is gradient descent and is the key unit to minimize our MSE loss function. Also, the metric = [‘mean\_squared\_error’] which is the negative mean square error which is the same metric we used for linear regression.
* **neural\_network = KerasRegressor(build\_fn=create\_network, epochs=100, batch\_size=50, verbose=False).** The KerasRegressor inputs require the Neural Network mode, the number of epochs, number of batch size and whether we want to print out the progress of each epoch being scanned. **Epochs** is 1 round trip when an entire dataset has been past forward and backward in a neural network. For my model, I wanted the dataset to be passed 100 times to fit the data better. Because the Neural Network cannot pass an entire dataset at once, I decided to determine the **batch\_size** to be 50 rows for the Neural Network to fit with. **Verbose=False** is the display of the algorithm going through each epoch and it displays a loading screen for each epoch but we don’t need to display it and also it speeds up the algorithm a bit more as it takes around 10 minutes to run.



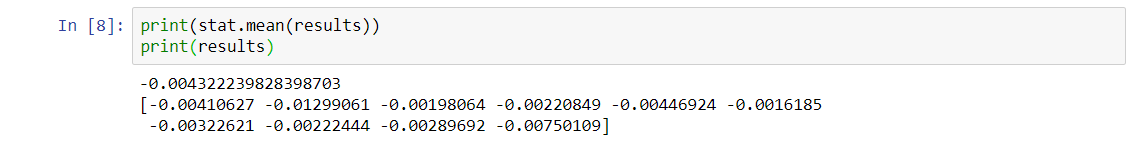
**seed = 1**

**kfold = KFold(n\_splits=10, random\_state=seed)**

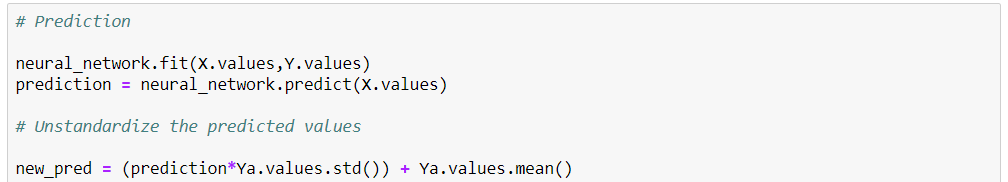
**results = cross\_val\_score(neural\_network, X, Y, cv = kfold)**

* For the three codes above, they are used to create the 10-fold cross validation. This is the same code/process from the linear regression model.

**Results of Neural Network**



* As shown above, the average MSE (-1)\*(-0.004322) = 0.004322, this accuracy is overfit because the MSE values are incredibly much lower than linear regression and this is testament to why neural networks is so powerful. The accuracy and predictive power of Neural Networks is incredible.
* Similar to linear regression, I predicted the entire dataset using Neural Networks model without using 10-fold cross validation.



**neural\_network.fit(X.values,Y.values)**

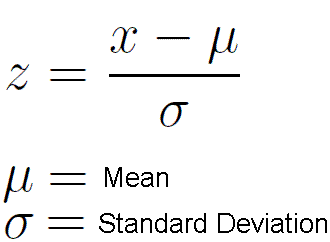
**prediction = neural\_network.predict(X.values)**

**new\_pred = prediction\*Ya.values.std()) + Ya.values.mean()**

**pd.DataFrame(new\_pred)**

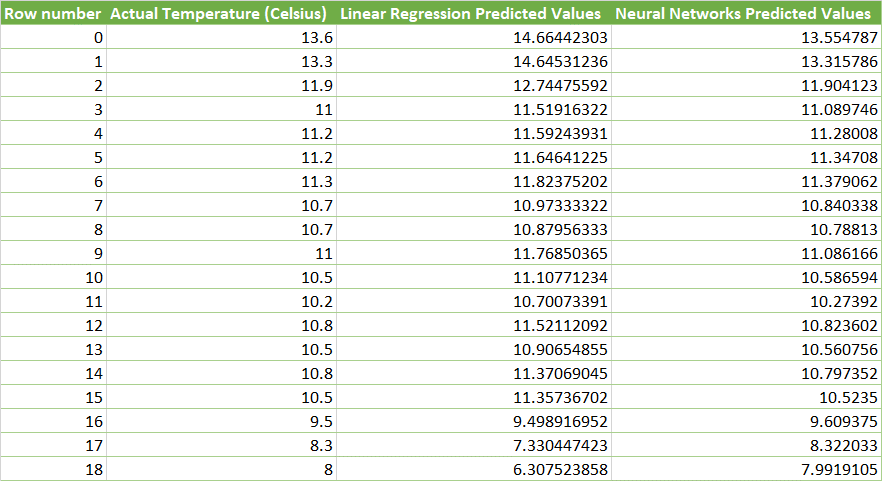
* The first two line of codes above simply predict the neural network with the dependent variables to the target variable.
* **new\_pred = prediction\*Ya.values.std()) + Ya.values.mean().** This line of code is to unstandardized the results since we standardized it earlier, rewriting the formula we get:

From standardization:



To unstandardized form:

* I exported the results in excel and below is a sample of the differences including the linear regression.



**Conclusion**

* As we reviewed previously the average 10-fold cross validation for linear regression is 1.3785 and for Neural Networks it is 0.004322. The difference is 1.374178 which is huge when compared with one another. Also, when I predicted the values for the entire dataset for each algorithm (as shown above), you can see how much more accurate the Neural Network is in comparison to Linear Regression, in fact it looks like it might be overfitting. The only way we can test it is if we have more data but based on these testing results, Neural Networks is the more effective and accurate model for this data set.