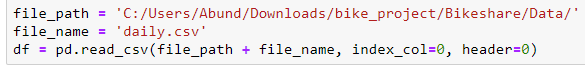
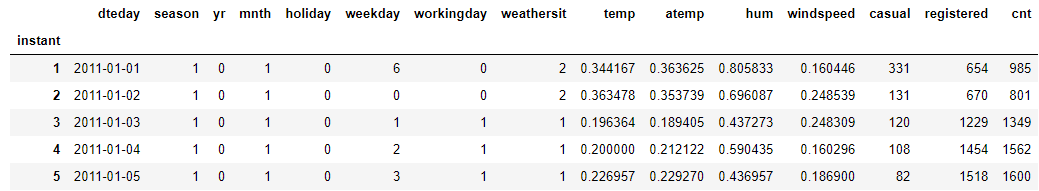
**Multiple Linear Regression Model (MLR)**

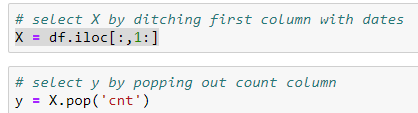
Decided to build my Linear Regression model based on the daily dataset instead of the hourly data. After reviewing the data it appears that the first column is an index column, first row are the headers. Imported my data as follows using the Pandas read\_csv() function defining the previous assertions.



From there I reviewed to see what the data looked like using the Pandas head() function in order to view the first 5 rows.



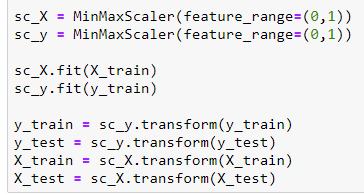
From this I determined that there were 15 columns in the data, including a date column at index 0. Since dates can’t be used in Multiple Linear Regression models I decided to drop this data from the selection, leaving 13 columns of predictors and the last column as my value to be predicted. But because the columns such as humidity or windspeed are very small values by comparison to the registered column my next step was to normalize the data.



Before normalizing, I split the data into 80% training data and 20% test data. I did this using the train\_test\_split() function from sklearn to randomly split into X and y, test and train variables. Because a linear regression model does not take into account a time series’ positioning I randomly did so using 123 as my seed to be able to recreate my results.



In order to normalize the data you take each column, find the min value, max value, and for each point subtract min and divide by max-min. Or, in order to make my life easier, I could use the MaxMinScaler function from the sklearn library. Used the training X’s and training y’s to scale the test data to a number between 0 and 1



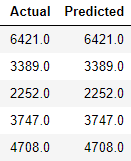
Because all columns are numerical I decided to test them all out in an exhaustive feature search. In order to select the best features to use, instead of running individual analysis on 13 columns vs the y variable, I used the ExhaustiveFeatureSelector function from the mlxtend.feature\_selection library. This does an exhaustive search between all variables to find the variables with the highest correlation. The function returned the following output, which aligns with columns for workingday, casual, and registered as the highest predictors of count.



Based on these columns I built my multiple linear regression model using my train X and y variables. Coefficients and Intercept of the final model shown below:



From my test dataset I then predicted values based on this model. As you can see below, the Actual and Predicted values are spot-on:



Testing using the sklearn Mean\_Squared\_Error function you can see that the model is extremely accurate. The difference between predicted and actual values is extremely low and a Multiple Linear Regression model proved to be extremely accuratre:



**Recurrent Neural Network Model (RNN) using Long Short Term Memory (LSTM)**

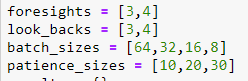
To build my RNN I decided to use the same parameters for the MLR. However, because I am treating the data as a Time Series, it was important to split the data without shuffling it.



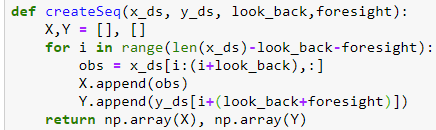
I decided to use the same 3 variables as above to predict count as they showed the highest correlation to count: workingday, casual, registered.

Because I have 3 variables I defined my RNN as a deep neural network. Went with 4 layers: input, LSTM, LSTM, and an output Dense layer with a Linear activation since we are dealing with linear data. Went with a dropout of 10%. I did this in order to teach the network not to rely too much on a handful of neurons. Went with 32 neurons per hidden layer.

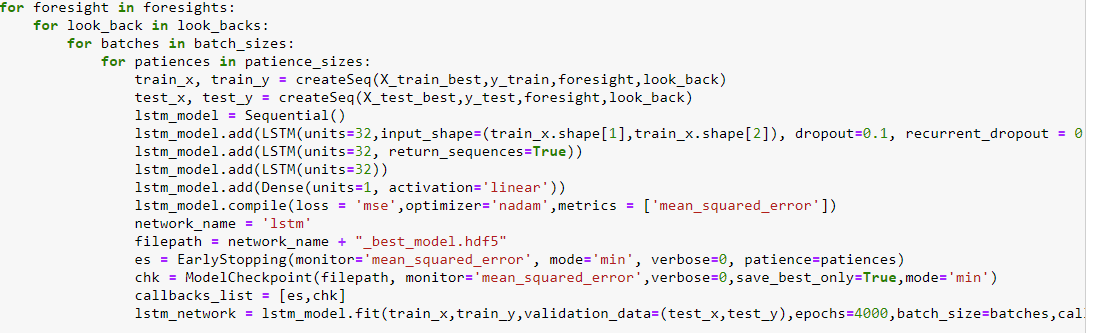
In order to find the optimum lookback window and foresight parameters, batch size, and patience level, I decide to loop through these parameters and see which one returned the lowest MSE.



By feeding those I selected my time windows and subset my data using a createSequence() function that was customly designed:



I then fed them through a loop to find the best model based on these parameters:



The lowest MSE was with a look\_back window of \_\_\_, with a foresight window of \_\_\_, updating in batches of \_\_\_\_\_, using a patience of \_\_\_ to designate an early dropout, for a calculated MSE of \_\_\_.

**Evaluation**

Even though the RNN is fancy, here the MLR easily outperformed it with less work. Shows the importance of determining the best method for the assigned problem. Why create a Neural Network for a problem that can be easily solved with a Linear Regression model?