**IST652 Scripting for Data Analysis**

**Final Project Report**

**Name: Vijet Muley**

**Project Name: Austin Weather Forecast**

**Primary purpose:**

The proposal for this project defined the goals to be achieved. It was proposed that this project will focus on initial cleaning of the collected data, then model specific munging and transformation and finally 2 machine learning models will be implemented in order to set up a method for predicting weather conditions/events based on different weather parameters.

**About the Data:**

The data was acquired from ‘Kaggle’. The data, on Kaggle, was obtained from weatherunderground.com, at Austin KATT station. The weather data, and this project, is actually a modularity of a bigger project that goes by the title Austin Bike sharing. The purpose of this module is to look at weather conditions, especially adverse events and then create a model to predict the occurrence of any such event which will help explaining the effect weather conditions have on bike sharing.

The data is time series data, that is, the data is a collection of measurement of a few variables over the period of time (in this case, every day’s record). The time span of the data is from 21st December 2013 to 31st July 2017. The data set comprises of 1319 rows, one for each day. The data set has 21 columns. The columns are as follows:

a). Date

b). Temperature, in Fahrenheit (High, Average, Low)

c). Dew Point Temperature in Fahrenheit (High, Average, Low)

d). Humidity Percentage (High, Average, Low)

e). Sea Level Pressure in Inches (High, Average, Low)

f). Visibility in Miles (High, Average, Low)

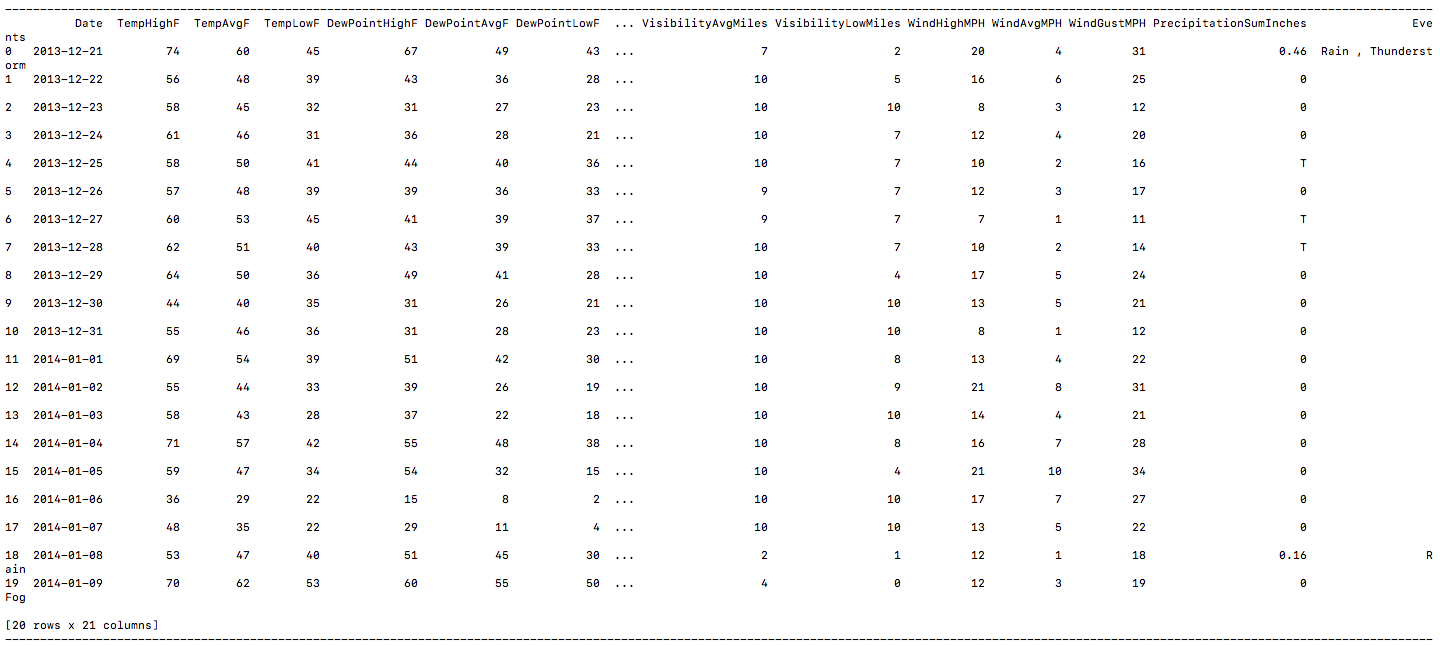
g). Wind Speed in Miles per Hour (High, Average, Low)

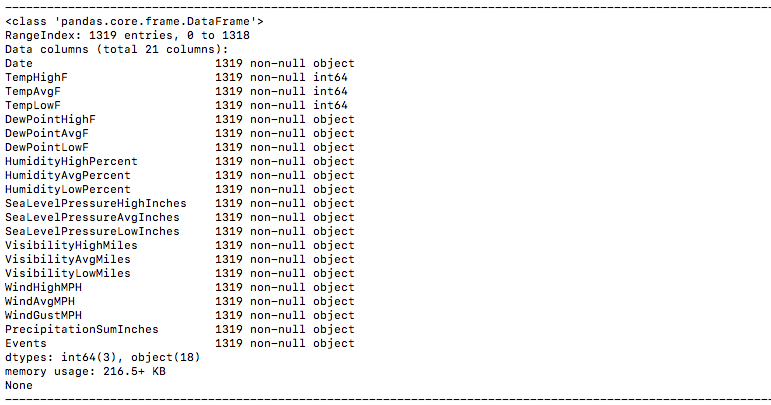
h). Precipitation in Inches (Sum for the day)

i). Events (Adverse Weather Conditions) – This is the target variable.

The factors from ‘b).’ to ‘g).’ all have 3 individual columns for 3 different measure, and in addition to these 18 columns, the data set has 3 columns for ‘Date’, ‘Precipitation’ and ‘Events’, making the total number of columns to be 21.

Following is a glimpse at what the raw data looks like. The first image is a tabular view of the first 20 rows. I use text editor (Atom) and terminal to run my code and Terminal’s way of displaying something that exceeds its boundaries is to show the first few and last few columns with ‘…’ in between, as a tilde. The next image is a descriptive view of data.





The data types of the column show that only 3 out of the 21 columns have a numeric data type. In reality, I was expecting all but 2 columns to have a non-numeric data type, i.e. ‘Date’ and ‘Events’. Rest all are quantities that are measured and ideally, they should have had numeric data type. Their classification as non-numeric ‘Object’ data is an indication that there are non-numeric characters present as entries, in lieu of absent or missing data.

After getting this idea about the data and some possible issues, I moved onto the task of general cleaning of data before model implementation.

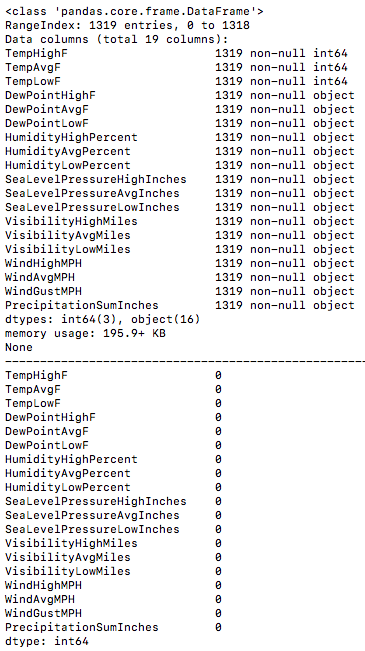
**Primary Pre-processing:**

As mentioned before, the data set has 21 columns. However, just by sheer observation, it is evident there are a few columns that have repetitive type of measurements. For eg., there are 3 columns for measurement of Temperature. Same is the case for 5 other columns, namely, Dew Point Temperature, Humidity Percentage, Sea level Pressure, Visibility ad Wind Speed. Although the measurements are particular to different points in time, they still are same type of measurements.

**Principal Component Analysis:**

As an independent effort, I implemented Principal Component Analysis (PCA). PCA makes linear combinations of all the dimensions in a data set and tries to maximize the variance explained by the first principal component and then build upon it.

PCA works on combining the columns that show covariance, that is, they are related or dependent on each other. It works towards simplification and data dimensionality reduction while minimizing the loss of accuracy from analysis techniques. As PCA works on reducing covariance, it can only work on data that is of numeric type. Thus, to run PCA on this data, I straight up removed the ‘Date’ and ‘Events’ columns. Furthermore, the data had columns that were of the Pandas data type ‘Object’, which meant they had a few non-numeric data entries within them. I thought it might be missing data, so I tried counting the number of ‘NA’ or ‘NaN’ values in the data set.

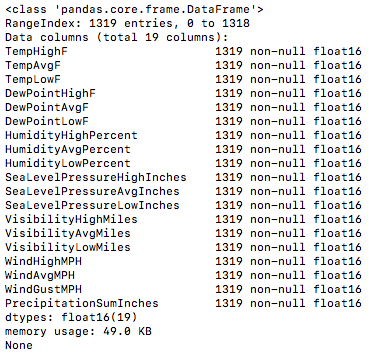


There were 0 occurrences of NA or NaN values in any column of the data set. However, eyeballing a few rows of the data showed that the non-numeric data entries were:

a). ‘-‘ in DewPointHighF, DewPointAvgF, DewPointLowF, HumidityHighPercent, HumidityAvgPercent, HumidityLowPercent, SeaLevelPressureHighInches, SeaLevelPressureAvgInches, SeaLevelPressureLowInches, VisibilityHighMiles, VisibilityAvgMiles, VisibilityLowMiles, WindHighMPH, WindAvgMPH, WindLowMPH.

b). ‘T’ in PrecipitationSumInches

For PCA, I tried a much simple approach of replacing all the non-numeric data occurrences with 0. After that, I type casted every entry to float16 data type. That resulted in:



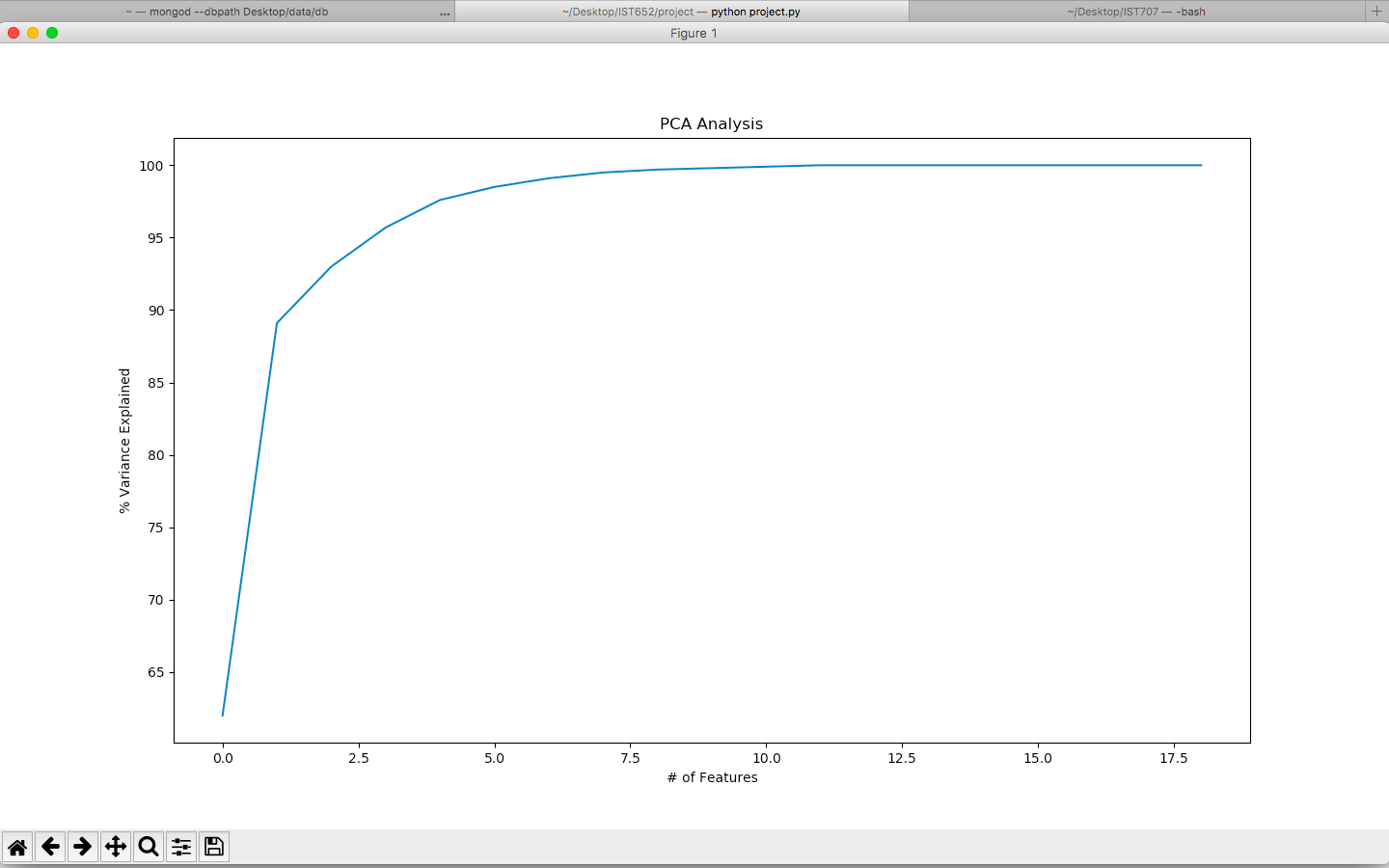
Since all the columns were type casted successfully into float16 type, it can be concluded that there are no more non-numeric entries.

The decomposition class from sklearn package has the method called PCA which implements Principal Component Analysis. I have used that in this project. While calling the function, I have passed only one variable, n\_components, which is the number of components or dimensions the data originally has. In this case, it is 19 (after removal of 2 columns earlier). Then I made a numpy array that showed the percentage of variance explained by every principal component formed as a result of PCA. The array looks as follows:



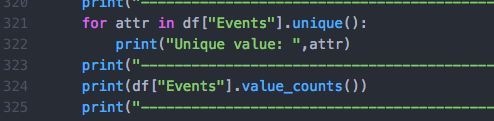
What this means is that PCA tries linearly combining dependent components into a single principal component with an aim of maximizing the percentage of variance explained by the first component. The first component formed from this data set explains 62% of variance. Further, the first and second component explain 89.1% of variance and so on. Around the 12th component, the percentage of variance explained becomes 100.

I also made a visual representation of this same concept. It is as follows:

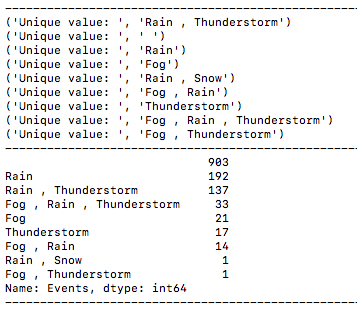


I used ‘matplotlib’ package’s ‘pyplot’ class for plotting this visualization.

After this attempt, coming back to the generalized pre-processing and cleaning of the data. I decided to start with the target column, that is ‘Events’. Visually, it was evident that the ‘Events’ column had a lot many blanks. This simple snippet of code helps knowing the composition of the ‘Events’ column:



The output is as follows:



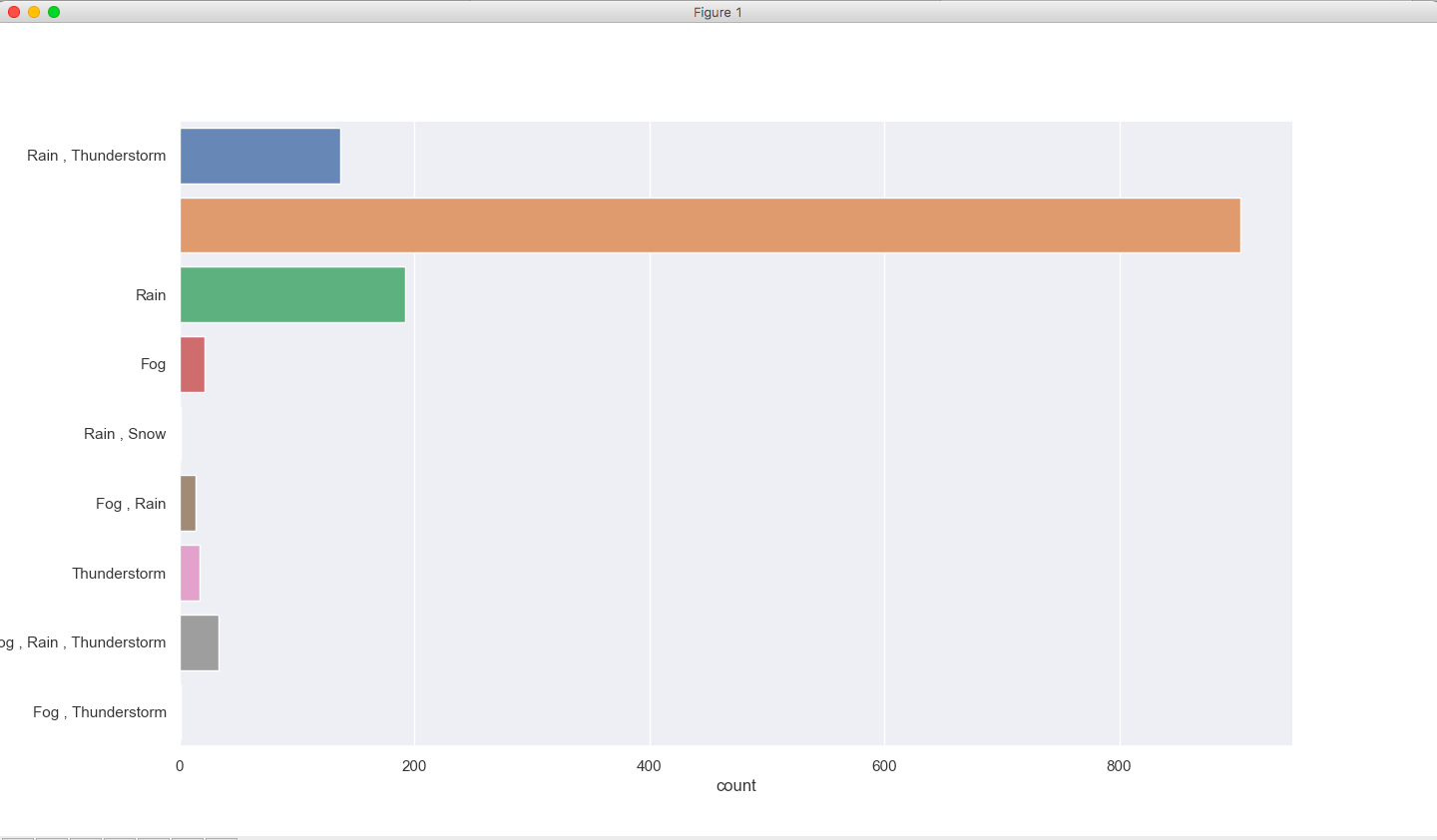
This helps realize why the data had no entries instead of NaN values. Ideally, python tries replacing missing data values with NaN while importing. However, in this case, there was no missing data in the ‘Events’ column. Instead, whenever there was no special adverse weather condition, the data set comprised of a ‘ ‘ (a space). Hence, there were no NaN values.

Further, the exploration helps us realize that there are 903 occurrences of these non-special days. This number is significantly large when compared to the size of the data, that is 1319. Thus, it cannot be ignored directly. Also, clear weather is a weather condition too.

Using the following code to print out a visual distribution:



Using the ‘countplot’ function from ‘seaborn’ package helps plotting a frequency bar graph for all categories in a column. The output is as follows:



I decided to reduce down on the conditions and fixing the “ “ problem as well. The method I followed is:

a). Replacing “ “ with “Clear Weather”. Since all these days had no adverse or special weather condition, it will make more sense to classify them as days with clear weather.

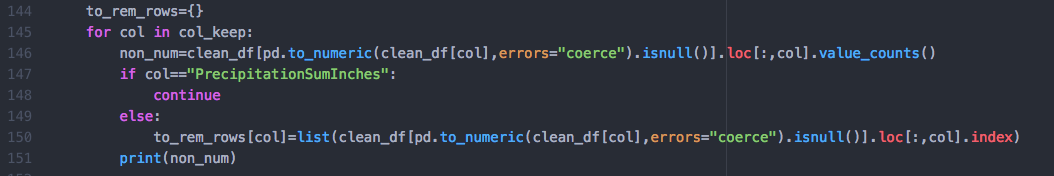
b). “Rain” and “Rain, Snow” get classified as “Rain”. The reason is that “Rain, Snow” has just 1 occurrence over all of the data set for all the days it was recorded. Besides, Austin is in Texas, which hardly gets any snow.

c). “Rain, Thunderstorm” get classified as “Rain+Thunderstorm”. This is just for simplification.

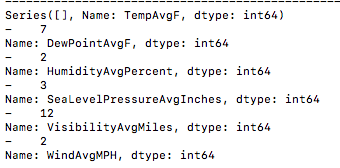
d). Everything else gets classified as “Fog+Rain+Thunderstorm”. The reason behind a different category is mainly the occurrence of Fog. Fog has small number of occurrences on its own (21). However, it occurs a lot with other conditions and that number is significantly big enough to get its own category. (The + in this classification can be interpreted as an ‘OR’, that is at least one of the 3 conditions happened for every row under this classification. However, there can be more than one condition occurring too.)

After this reasonable classification, I turned my focus back towards the original issue of non-numeric entries in many of the columns. I, first off, decided to drop ‘Date’ column as it won’t contribute to analysis, unless I am conducting time-series analysis. I chose not to conduct time-series analysis due to the limitation of scope and time. I chose to include the columns that recorded the average of the weather measurements. The reason is that there is an interdependency between the columns measuring the High and Low with the columns measuring Average values. Highs and Lows of a day will drive the Average. Removing interdependency will make the models and apparent predictions more accurate. Also, High and Low are very specific, that is, they are very point specific recordings and Average give a much better idea when it comes to a whole day. None of the parameters measured as High, Low and Average will showcase extreme recordings for High or Low for extremities in weather are very rare. Even if there are any extremities, Average reading will cover that extremity very well. These are the reasons I chose to include only TempAvgF, DewPointAvgF, HumidityAvgPercent, SeaLevelPressureAvgInches, VisibilityAvgMiles, WindAvgMPH for further cleaning and analysis.

The data set is big, and it is not practically to look at the data set manually and look for different non-numeric characters occurring over every column. So, I used a masking technique. The following snippet of code runs on all the columns except of PrecipitationSumInches. The reason to leave this column out is that the data source had already mentioned that this column has ‘T’ as non-numeric entries and no other non-numeric entries in this particular column. However, I was still oblivious about the composition of other columns.



This code works with a dictionary, ‘to\_rem\_rows’. The first line inside the for loop is a data-frame operation within another data-frame operation. Python will execute the inner operation first. The inner operation is trying to convert each and every data cell to a numeric type; that maybe integer or float depending on what the data looks like. If the function comes across something that cannot be converted into a numeric type, that is a non-numeric data entry, it will use the coerce strategy, that is it will replace that entry with a NA. The ‘isnull()’ will give a numpy array with row indexed that have null values (NaN or NA) which are the rows which were coerced into null values due to having non-numeric data entries. Then I used value\_counts() which will count number of occurrences of every non-numeric entry.



Now we know that the columns with non-numeric entries all have just ‘-‘ as the non-numeric occurrence and the number of occurrences over all the individual columns. In order to take suitable action, I needed the row numbers of the rows with these entries. That was obtained from the if-else block code in the above snippet. The if-else is just to make sure that the code skips precipitation column. The code inside the else block is sort of similar to the earlier bit of code involving masking. The only difference is that, instead of counting the number of occurrences of non-numeric characters, this bit will use the ‘index’ function to get the row indexes as a numpy array. Also, I am making an entry in the dictionary for every column with column name as the key and the numpy array of row indexes as the value to that key.



Further, I wrote code to take all these row indexes for different columns and combine them into one single list, taking multiple occurrences of a single row index only once. This resulted in creation of following list:



Thus, these 12 rows have one or more than one non-numeric entries for all the included columns except PrecipitationSumInches. I am choosing to simply drop these 12 rows from the data set. There are 2 main reasons for that:

a). 12 is a small number when compared to the size of the data set, 1319. Removal of these 12 rows will still leave us with 1307 rows for analysis.

b). In this case, absence of data does not mean absence of those factors at all. Replacing them with mean may introduce outliers for that particular period. I say this since the data is time series data and is seasonal. I had the choice to fill these fields up with predictions and it’d be required to conduct some sort of time-series prediction technique (like moving average or so) to make these predictions. I decided to not choose that option due to the limitation of scope and time.

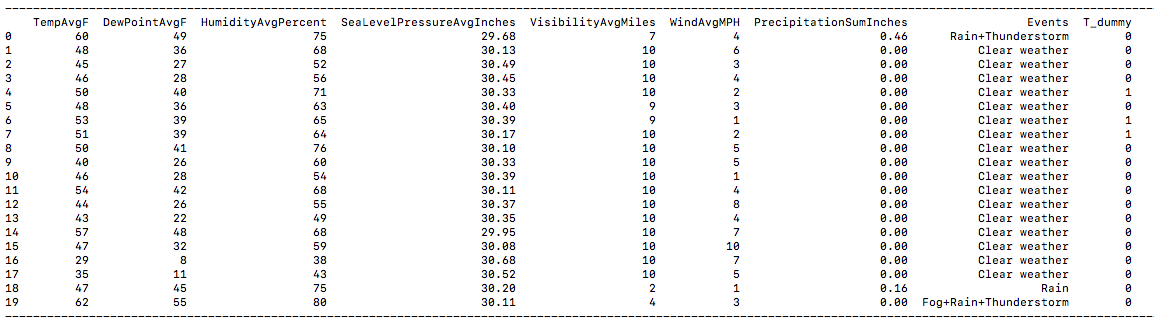
Moving on to the PrecipitationSumInches column, it has non-numeric entries as well. The non-numeric entries are in form of ‘T’. As per the data source, presence of ‘T’ instead of a number is an indication that there was very small amount of precipitation, very close to 0 but not 0. I chose to tackle this issue as follows:

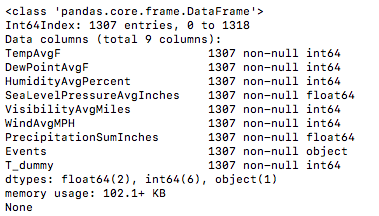
a). Replacing all ‘T’ entries with 0.

b). However, the 0 for ‘T’ is not exactly same as normal 0 entries. Normal 0 entries indicate that there was no precipitation at all. Hence to differentiate between these 2 different versions of 0, I chose to introduce a new column, T\_dummy. This field is a Boolean field and will be 1 if there was ‘T’ or trace amount of precipitation and 0 for all other cases, including no precipitation at all.

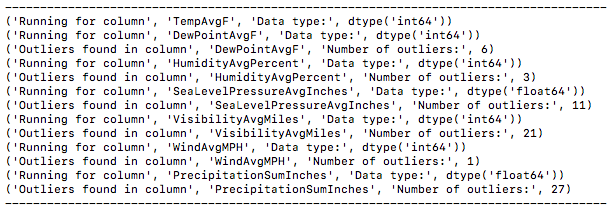
Finally, after all the replacing, dropping of non-numeric rows and removal of unwanted columns, I converted every column to numeric type using ‘to\_numeric’ function on the data frame columns in addition to ‘apply’ function.

The cleaned and transformed data frame is as follows:

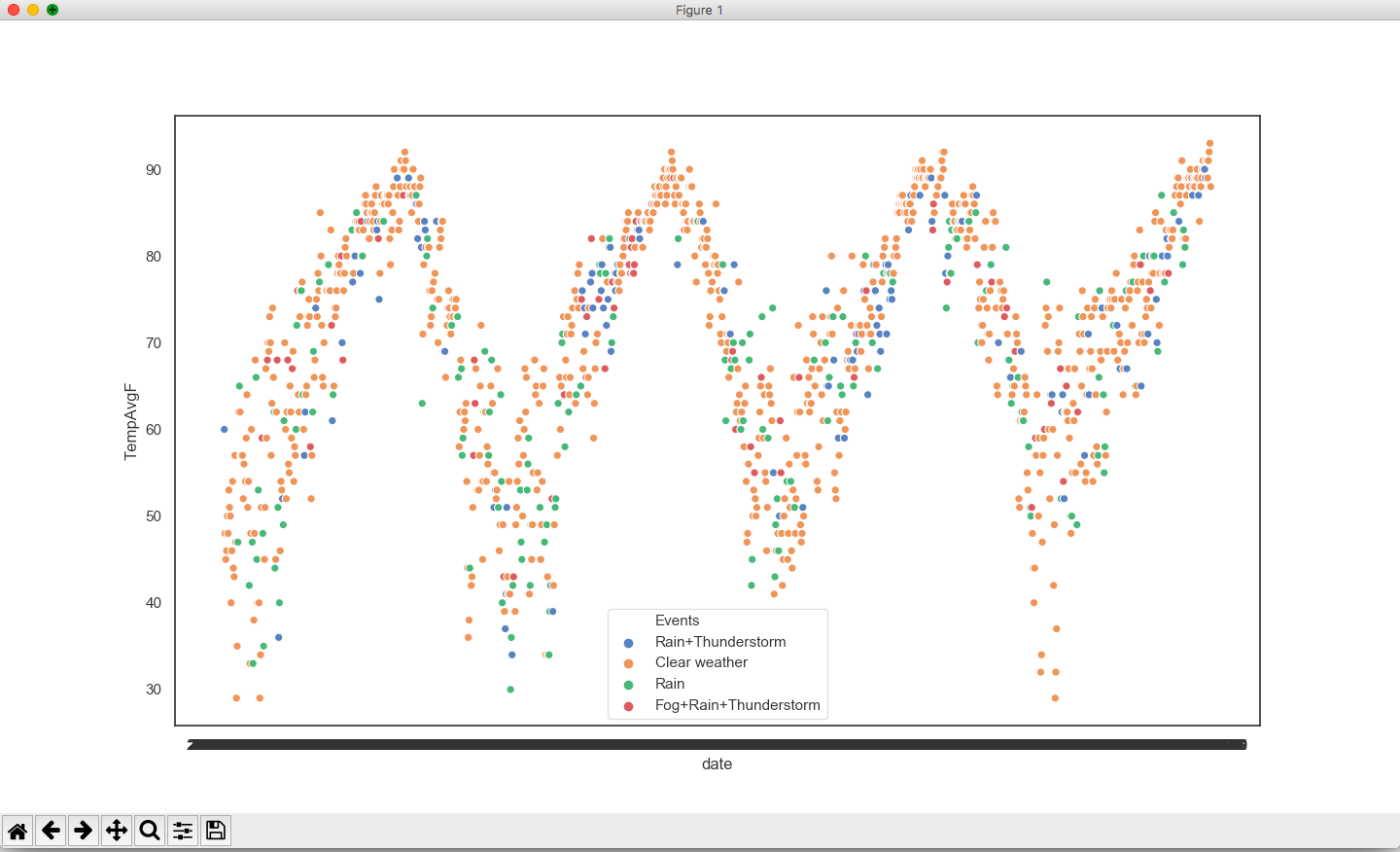




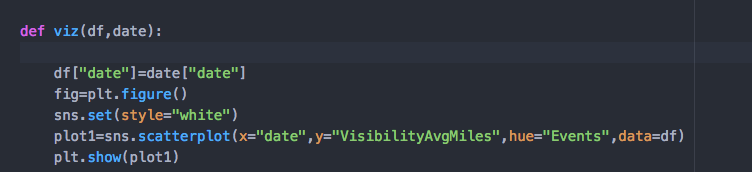
All the columns are now in numeric form, except for ‘Events’ (target column) as it is a text column. I wrote my own function that took a column as an argument/input and calculated mean and standard deviation for the column and based on these 2, calculated z-scores for every data point. I set an absolute z-score value being greater than 3 as threshold for a data point being an outlier and the function return an array with row indexes of all the data points that are determined to be outliers.



The visualization ahead shows the distribution of TempAvgF over time (using ‘Date’ column). My intention behind looking at this visualization was to look at the seasonal pattern and how outliers may occur.



Following code was used for creating this visualization:



I used ‘seaborn’ package’s scatterplot function to make this visualization. I limited visualizations to just one column as it took large amounts of time to get them created.

**Predictive Modeling:**

After understanding the data more and getting it cleaned and transformed into correct format, finally moving to implementing predictive models. I chose to implement 2 predictive modeling techniques, namely:

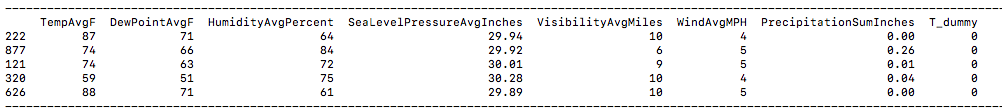
a). Gaussian Naïve Bayes

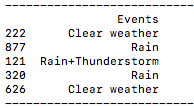
b) Multinomial Logistic Regression

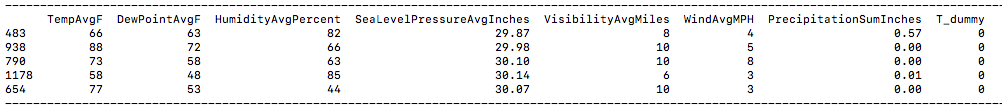
**a). Gaussian Naïve Bayes:**

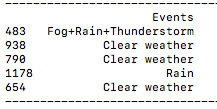
The first model I chose to implement is Gaussian Naïve Bayes. Bayes theorem assumes that all the explanatory variables are independent of each other, that is why the term ‘Naïve’. There are, broadly, 3 types of Naïve Bayes models, namely: Multinomial, Gaussian and Bernoulli. I chose Gaussian for the explanatory variables are continuous numeric.

Implementing Naïve Bayes model in general using python is fairly easy. I used ‘sklearn’ package’s ‘naive\_bayes’ class. The class has a function; ‘GaussianNB’. First off, I separated the data-frame into 2 subsets (column-wise). ‘Events’ (Target column) formed a data-frame ‘y’ and the rest of the columns (explanatory variables) formed the data-frame ‘x’. Furthermore, both the data-frames were divided into training and testing sets using ‘train\_test\_split’ function from ‘sklearn’ package’s ‘model\_selection’ class. The data is split into 2:1 ration for training and testing respectively. Also, I gave a random seed amount, which resulted in randomized sampling while splitting. If not given, the data will be divided serially and I wanted to avoid that as the data is seasonal.





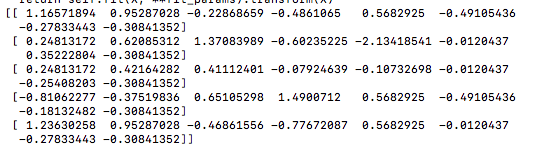




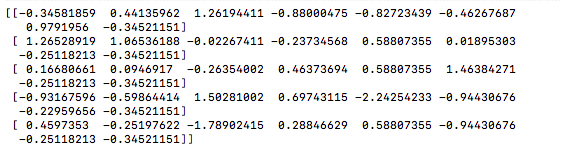
Then I chose to subject the explanatory variables to scaling procedure. The reason behind this is the difference in ranges and variances of the different variables. For eg., Temperature can lie between 20 to 120 (roughly), while precipitation is ranging from 0 to, maybe, 5. In case that the data is not scaled, the variables with bigger range and variance would dominate the analysis. I chose to use 2 different types of scaling methods, namely, MinMaxScaler and StandardScaler from ‘sklearn’ package’s ‘preprocessing’ class.

The difference between these 2 scalers is the methodology behind them. MinMaxScaler will squeeze the data in the range 0 to 1. It will scale down the minimum value in the column to 0 and the maximum to 1 and all the other values would be scaled down to this range with those 2 values as constraints. MinMaxScaler works well when there is no massive variance in the data, since it has no regards for adjusting standard deviation, it will diminish or minimize the effects of outliers and variance in general.

StandardScaler acknowledges the standard deviation and preserves the effects of outliers and variance. It shifts the mean to 0 and squeezes the data such that the standard deviation becomes 1 9Basically, it is calculating z-scores).

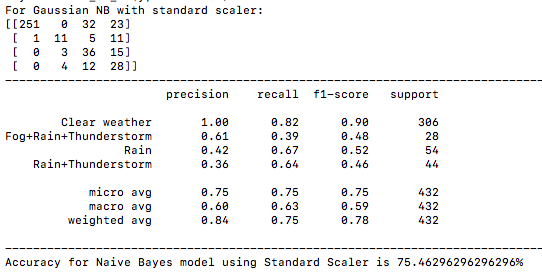


x\_train scaled using StandardScaler look as shown above. The numpy arrays inside the outer numpy array are scaled values for each row.

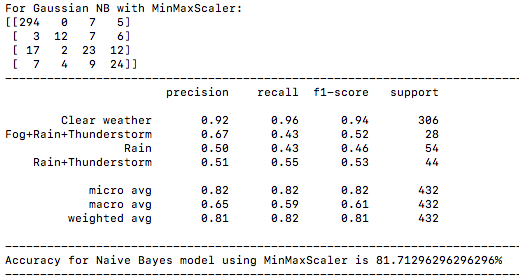


x\_test scaled using StandardScaler is as shown in the image above.

In order to evaluate the performance for each version of model, I decided to use confusion matrix, classification report and accuracy score for both. I used ‘metrics’ class from the ‘sklearn’ package. The functions are ‘confusion\_matrix’, ‘classification\_report’ and ‘accuracy\_score’. When I used the x\_train ad y\_train data scaled using StandardScaler to fit the Gaussian Naïve Bayes model. Then I used the ‘predict’ function to use the model on x\_test data to get predictions and finally compared them with the y\_test, I got the following output:



When I used the same training data (x\_train and y\_train), scaled using MinMaxScaler to fit another Gaussian Naïve Bayes model and checked the accuracy using the training data (y\_test) and predicted output generated using ‘predict’ function on the x\_test data, it was as follows:



The model using data scaled using MinMaxScaler gives a better result, with an accuracy of roughly 81.71% as compared to the model using the data scaled using StandardScaler which gives an accuracy of around 75.46%.

The sample of the data used (that is the division of data obtained using train\_test\_split) was exactly the same for both the models. This means that scaling proved to be the differentiating step. It is an interesting find as scaling comes under pre-processing of data.

My understanding as to why MinMaxScaler maybe giving a model is that, although this data set has outliers (findings are above), the outliers are not that extreme, since it is weather data. The variance won’t be too massive and also the number of outliers is low, thus dampening their effect. That maybe why MinMaxScaler resulted in a model that performs better.

**b). Multinomial Logistic Regression:**

Multinomial Logistic Regression is used when the output or target variable is a categorical variable with more than 2 categories. I this case, generalized cleaning and transformation lead to 4 categories in the target variable ‘Events’.

I have used ‘LogisticRegression’ function from ‘sklearn’ package’s ‘linear\_model’ class. The ‘multi\_class’ parameter specifies that the model to be implemented is of type multinomial. The solver used is ‘newton-cg’ which enables the model to be multinomial by adding an L2 type of penalty. I have used the same function (as in Gaussian Naïve Bayes) to split the data into, first x and y, and then training and testing sets. However, while testing, I just used the accuracy score:





The Multinomial Logistic Regression model gives an accuracy of 76.16%.

**Conclusion:**

Using the data that had the same generalized cleaning and transformation, the performance of the models suggests that Gaussian Naïve Bayes model, that used MinMaxScaler for scaling, performed better than the Multinomial Logistic Regression model. I base this conclusion on the following findings:

a). Accuracy for Multinomial Logistic Regression is 76.16%

b). Accuracy for Gaussian Naïve Bayes model using StandardScaler is 75.46%

c). Accuracy of Gaussian Naïve Bayes model using MinMaxScaler is 81.71%

As discussed earlier, I’d say that it was scaling that was the deciding factor in the increase of the accuracy, as rest all the procedures were same. Especially when it comes to Gaussian Naïve Bayes, where the raw data and the models used were exactly same, MinMax scaling proved to improve the performance.

I feel the reason is that, the number of outliers is relatively less (relative to the size of data set) and even after being outliers, I feel they won’t be too extreme of outliers. I credit this statement to the fact that the data is weather data, and there can only be so much variance in the data. For eg., if we take the PrecipitationSumInches column, many of the entries are 0 or very close to 0, with a few being higher when it rained exceptionally heavy. The mean would lie closer to 0 and the exceptionally heavy rainy days may get classified as outliers. However, it can be said that even though the rain may be heavy enough for the value to get classified as an outlier, the value may not be that high that it will be too extreme of an outlier. Also, logically, there are very few days where heavy rains occur. So, the overall effect of these outliers seems relatively low. MinMaxScaler is ignorant to outliers as it diminishes their overall effect.

**How would this project fit into the bigger picture?**

As I mentioned at the start, this project is a functionality for a bigger project, Austin Bike Sharing. This functionality can help predict weather conditions which may help the developers of the bigger project predict trends in bike sharing.

To get this project to that sort of working, it’d be necessary to implement time-series prediction techniques, such as weighted average or moving window average (or maybe some other technique). That will help predict the numeric components of the data, such as temperature, pressure, precipitation etc. for the upcoming days. Then these values can be used to predict the weather conditions in the ‘Events’ column which can finally be a feed to the bigger model used for predicting bike sharing trends.