END SEMESTER ASSESSMENT (ESA)
B.TECH. (CSE)

III SEMESTER

UE19CS203 – Statistics for Data Science

PROJECT REPORT

ON

"Weather in Delhi"

SUBMITTED BY

NAME	SRN
1) Bhargav N	PES2UG19CS088
2) Deepthi B	PES2UG19CS107
3)Dev Darshan J	PES2UG19CS108
4) Gourav.Aravinda	PES2UG19CS130

AUGUST – DECEMBER 2020

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ELECTRONIC CITY CAMPUS,

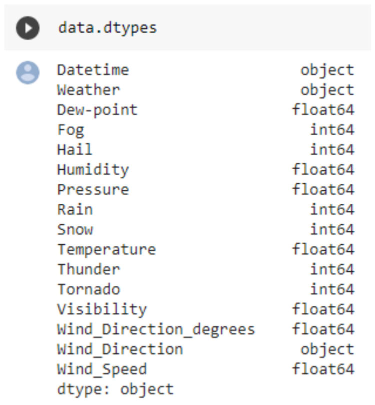
BENGALURU – 560100, KARNATAKA, INDIA

We chose a dataset on the weather in Delhi so that we could find out the factors that affect the weather in the Delhi. We chose a dataset that had the factors and the weather conditions recorded from Kaggle. We first cleaned the data using various methods that we will discuss. After cleaning the data, we used various methods such as box plots, pie charts, line graphs and others to see the relations between our data. We then proceeded to normalize our data to find out the distributions of the data. We then came up with some hypothesis and relations that we tested with the help of our visualized data. We could come to a few conclusions about what were the factors that could actually affect the weather in Delhi. Let's discuss in detail about the procedure that we followed to come up with these conclusions.

We started by deciding our topic of discussion first. We decided to do weather analysis and started searching for datasets. We found many datasets from various sources but decided to do our study on the dataset found on Kaggle about the weather in Delhi. This data was taken out of wunderground with the help pf their easy to use API. It contains various features such as temperature, pressure, rain, humidity, precipitation and many more We could see that there were many variables that we would need to do adequate study about the weather. The source also has credibility as Kaggle is known worldwide and the contributors are people who are more well versed then we are in the subject. Therefore, we selected the Delhi weather dataset that recorded the weather in Delhi from the years 1997 to 2017.

We felt the need to first standardize the names of the columns and the datatypes of the columns. We changed the data type of the date object to a timestamp as it allowed us to code functions better and the wind direction were made to represent 8 cardinal points. The names of the columns were also changed to represent the column better.

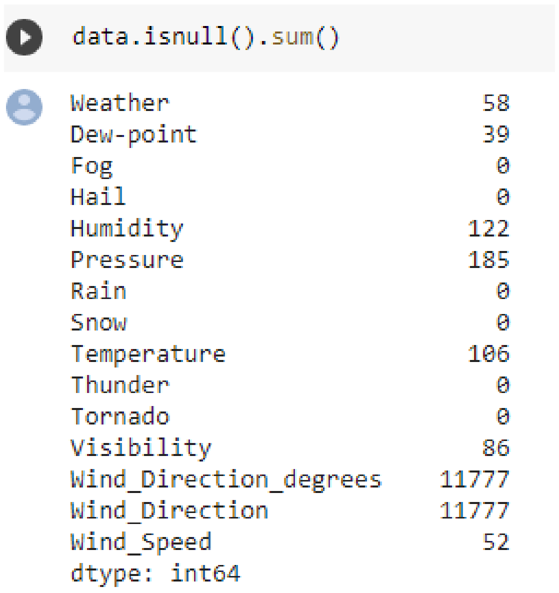
Our dataset was very detailed about the conditions of weather it provided. We have a date-time column at the start. They are in the intervals of 3 hours. We have categorical variables like the Conditions and Wind Direction. We have numerical data such as Dew point , Humidity ,Pressure, Visibility Temperature, and Wind direction. We have Boolean data such as Rain , Snow ,Thunder, Tornado, Hail and Fog. This data was sufficient for us to find a few relations between the weather conditions and the factors affecting it. We needed to clean the dataset first to ensure that the results that we come up with are credible and not faulty. To do so we first decided to check and handle the empty values of the dataset. The datatypes of our columns are shown in the following image.



```
data.dtypes
```

```
Datetime                    object
Weather                     object
Dew-point                  float64
Fog                          int64
Hail                         int64
Humidity                   float64
Pressure                   float64
Rain                         int64
Snow                         int64
Temperature                float64
Thunder                      int64
Tornado                      int64
Visibility                 float64
Wind_Direction_degrees     float64
Wind_Direction              object
Wind_Speed                 float64
dtype: object
```

We first found out the number of the null values in our dataset so that we could see what needed to be done. We found out that there were quite a few in different columns of the dataset. There were 4 columns in the dataset initially that were found to have more then 80% null values. Due to this we decided to drop these columns as they wouldn't have been of much use to us as they were mostly missing. The following image shows the number of null values in each column. We needed to handle these null values in such a manner that we don't lose any important that would have been lost had we just decided to drop them.

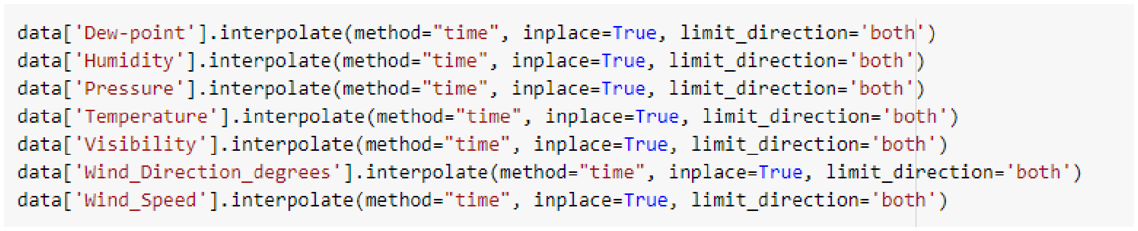


```
data.isnull().sum()
```

```
Weather                      58
Dew-point                    39
Fog                           0
Hail                          0
Humidity                    122
Pressure                    185
Rain                          0
Snow                          0
Temperature                 106
Thunder                       0
Tornado                       0
Visibility                   86
Wind_Direction_degrees    11777
Wind_Direction            11777
Wind_Speed                   52
dtype: int64
```

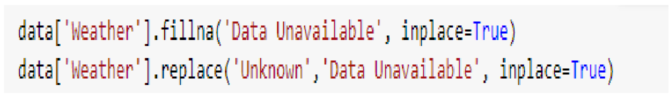
We decided to make use of the interpolate method to replace the values of the dataset that were missing in the columns Dew-Point, Humidity, Pressure, Temperature and Visibility. This method was preferred as we know that weather might not have changed too much from the time before unless. We shall see what the interpolate method does exactly so that we can justify our use of the function.

Interpolation is the process of estimating unknown values that fall between two known values. There are various methods for interpolating missing data, so here we have interpolated based on the time interval between two known values. This takes care of the condition that, if the timestamp of the missing data entry for a particular variable is closer to one of the two known values for that variable, then the interpolated value is closer to that know value. The code we used to execute the method is shown in the next image.



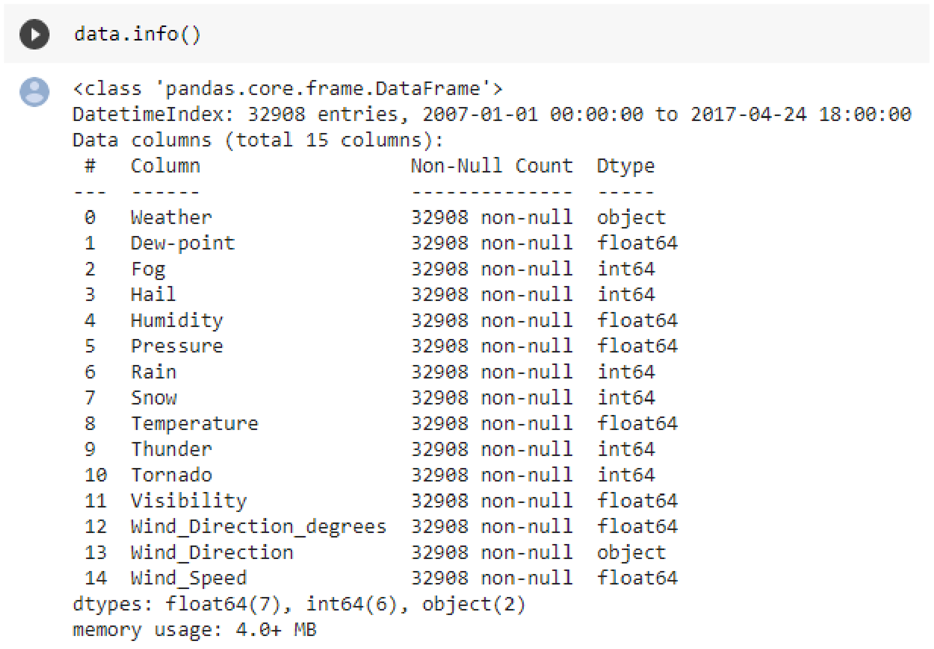
```
data['Dew-point'].interpolate(method="time", inplace=True, limit_direction='both')
data['Humidity'].interpolate(method="time", inplace=True, limit_direction='both')
data['Pressure'].interpolate(method="time", inplace=True, limit_direction='both')
data['Temperature'].interpolate(method="time", inplace=True, limit_direction='both')
data['Visibility'].interpolate(method="time", inplace=True, limit_direction='both')
data['Wind_Direction_degrees'].interpolate(method="time", inplace=True, limit_direction='both')
data['Wind_Speed'].interpolate(method="time", inplace=True, limit_direction='both')
```

This handles the numerical data. We had to also replace the null values present in the categorical variables list as well. Therefore we decided that to not lose important data we will replace the values of the missing in the categorical column the weather conditions with a NULL value. The code used to implement the above is shown in the next image.



```
data['Weather'].fillna('Data Unavailable', inplace=True)
data['Weather'].replace('Unknown','Data Unavailable', inplace=True)
```

This ensured that there were no more null values in the dataset. This was confirmed by the following output.

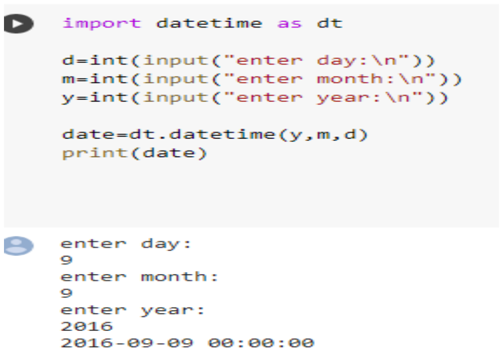


```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 32908 entries, 2007-01-01 00:00:00 to 2017-04-24 18:00:00
Data columns (total 15 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Weather                 32908 non-null  object 
 1   Dew-point               32908 non-null  float64
 2   Fog                     32908 non-null  int64  
 3   Hail                    32908 non-null  int64  
 4   Humidity                32908 non-null  float64
 5   Pressure                32908 non-null  float64
 6   Rain                    32908 non-null  int64  
 7   Snow                    32908 non-null  int64  
 8   Temperature             32908 non-null  float64
 9   Thunder                 32908 non-null  int64  
 10  Tornado                 32908 non-null  int64  
 11  Visibility              32908 non-null  float64
 12  Wind_Direction_degrees  32908 non-null  float64
 13  Wind_Direction          32908 non-null  object 
 14  Wind_Speed              32908 non-null  float64
dtypes: float64(7), int64(6), object(2)
memory usage: 4.0+ MB
```

After handling the null values we had to also handle the outliers. But to do so we had to first visualize the data to find these outliers in the data as they are not visible on seeing the dataset at first. We will handle them as they appear was our strategy.

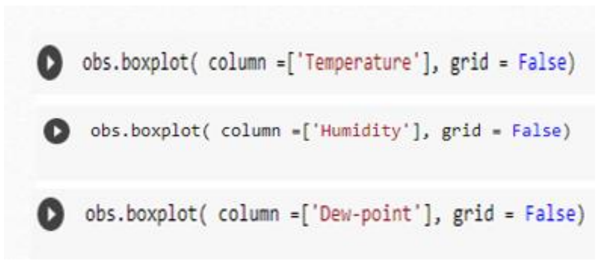
We had to select an appropriate sample to do our study as we had a large numbers of values. We decided to select a month at random so that we could study the factors of that month. All of our code ensures that this can be done for every month. The visualizations and the hypothesis testing ensures that we can input a month of our choice to study from. We selected the September of 2016 as the month to study from. We then had to visualize the data to find some correlations and the relations between the factors of the weather that were present in the dataset.



```
import datetime as dt

d=int(input("enter day:\n"))
m=int(input("enter month:\n"))
y=int(input("enter year:\n"))

date=dt.datetime(y,m,d)
print(date)
```

```
enter day:
9
enter month:
9
enter year:
2016
2016-09-09 00:00:00
```

We decided to start by doing box plots as they would give us an idea of how the data is spread out and give us an idea about the distribution of the data for each of the factors. We used the following code and got the following box plots for each of the conditions.

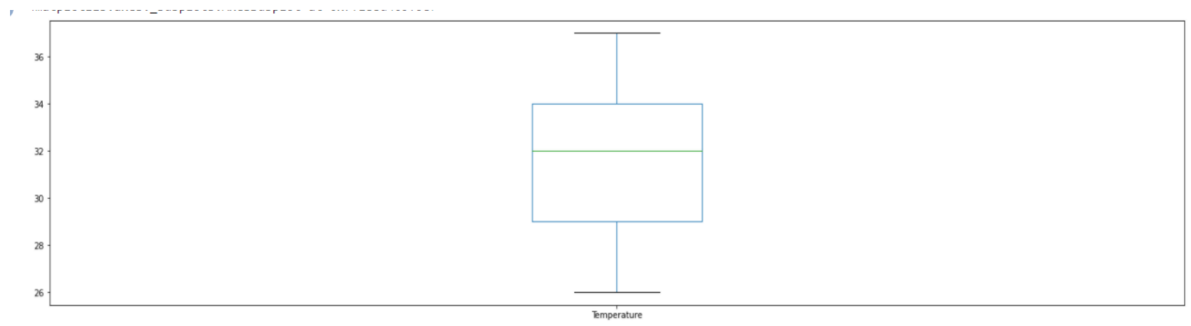


```
obs.boxplot( column =['Temperature'], grid = False)
```

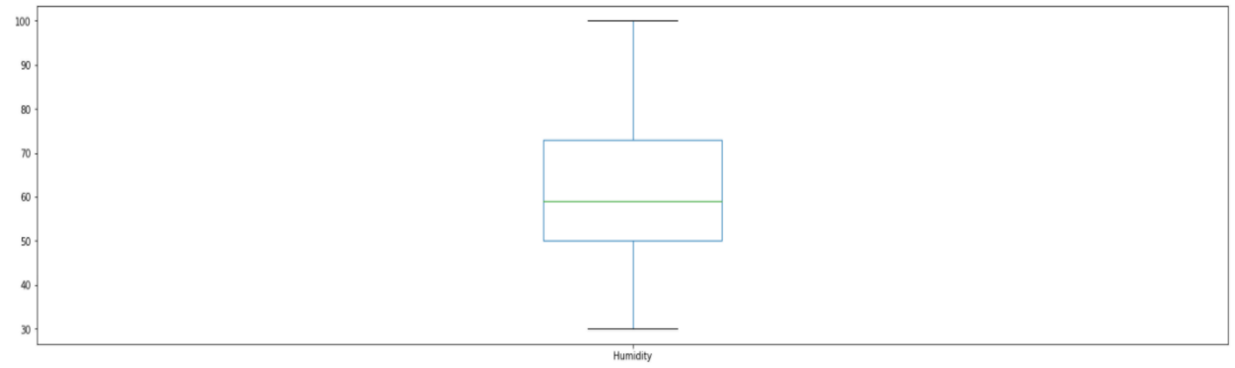
```
obs.boxplot( column =['Humidity'], grid = False)
```

```
obs.boxplot( column =['Dew-point'], grid = False)
```

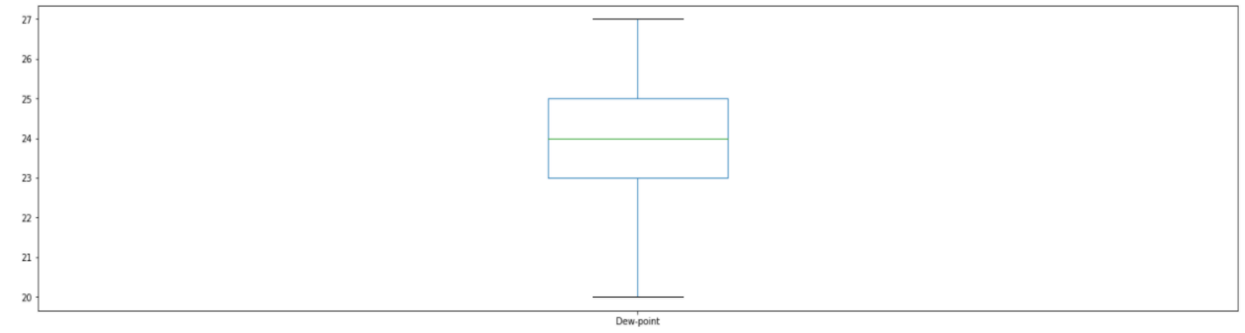
Temperature



Humidity

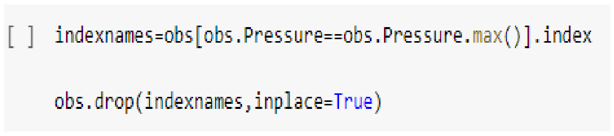


Dew Point



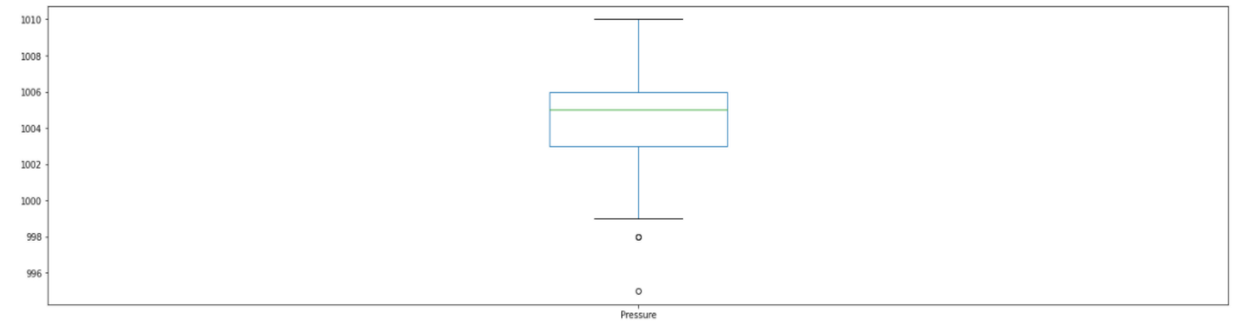
It was in the box plot for pressure that we noticed an outlier in the data. The following image is what the box plot looked like with the outlier.

We used the following code to remove the outlier and the plot the box plot for it without the outlier.

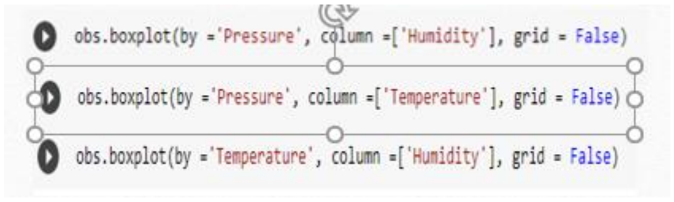


```
[ ]  indexnames=obs[obs.Pressure==obs.Pressure.max()].index

     obs.drop(indexnames,inplace=True)
```

Pressure

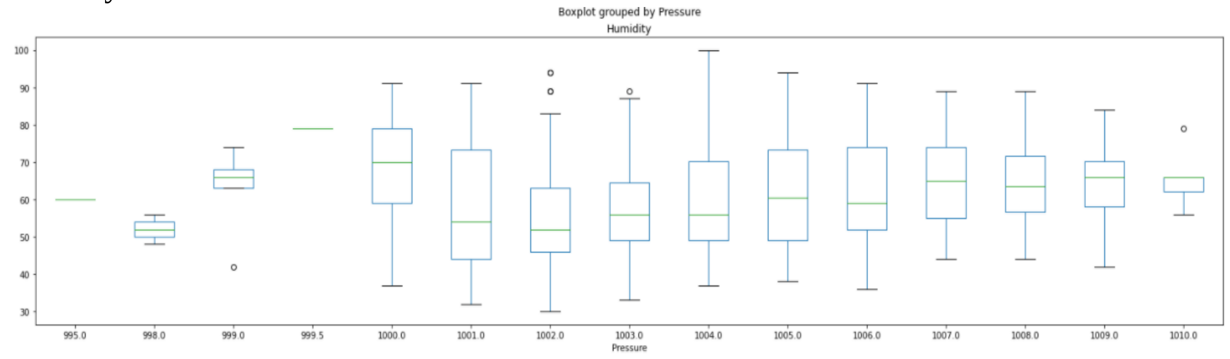


We then decided to do some relational box plots plotting one factor against the other to see if we could come up with any relations between the factors. The code and the box plots plotted are shown in the following images.

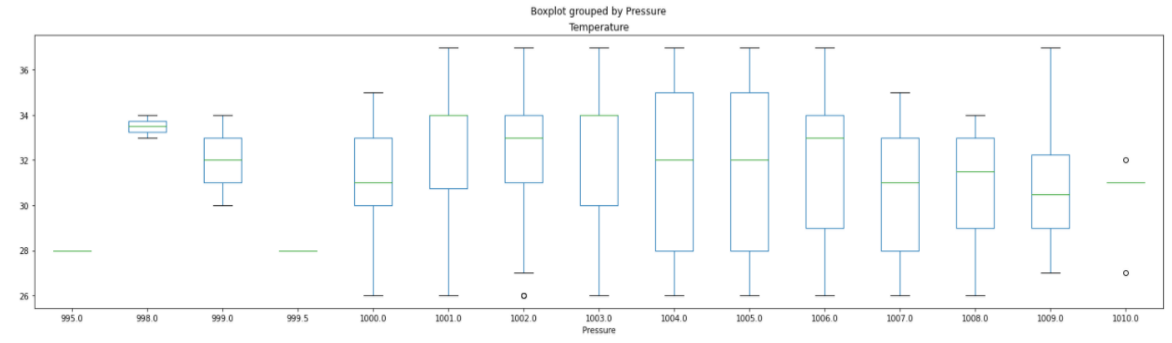


```
obs.boxplot(by ='Pressure', column =['Humidity'], grid = False)
obs.boxplot(by ='Pressure', column =['Temperature'], grid = False)
obs.boxplot(by ='Temperature', column =['Humidity'], grid = False)
```

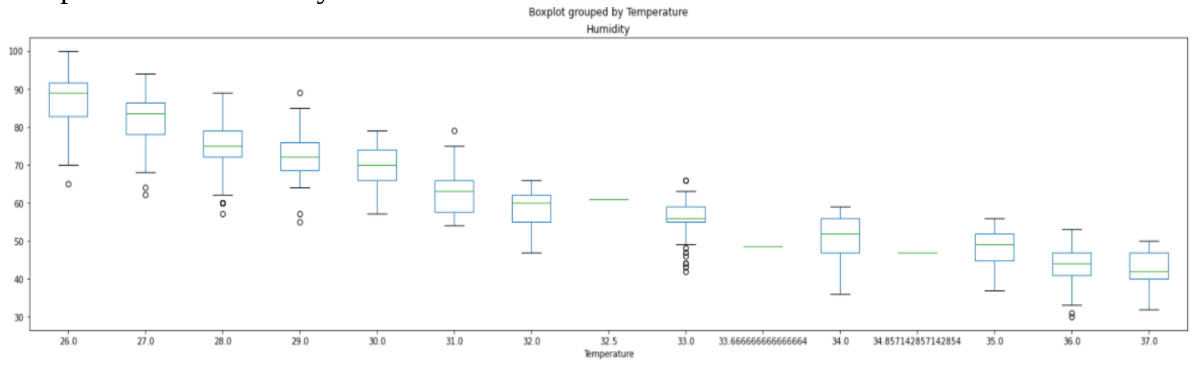
Humidity vs Pressure



We could learn from this that there is no clear relation seen yet.
Pressure vs Temperature



We could learn from this that there is no clear relation seen yet
Temperature vs Humidity

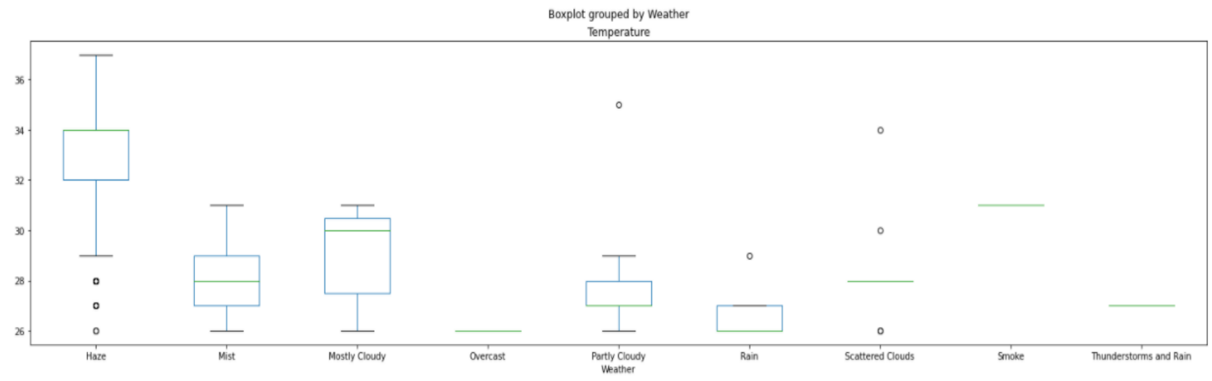


We could learn from this that as temperature increases there is a decrease in the humidity.

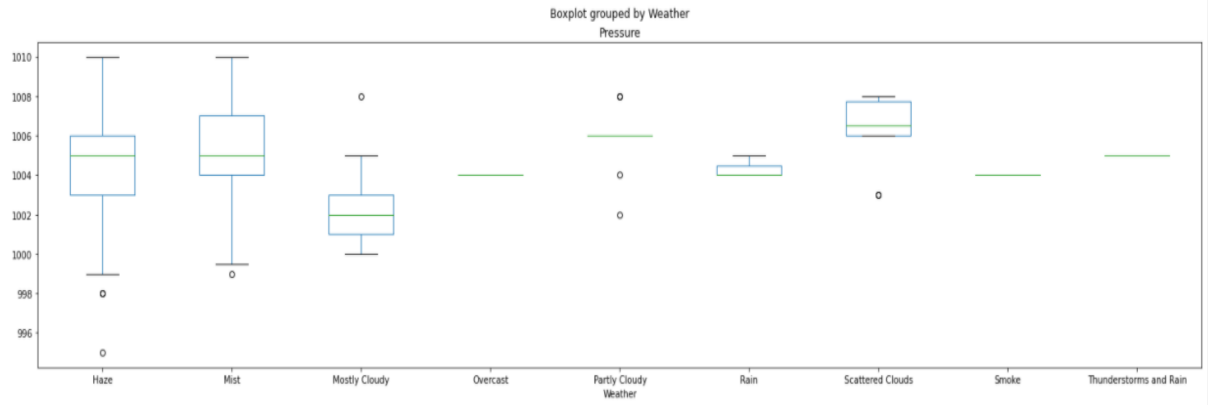
We then decided to plot the weather against all of the factors to get some relations.

```
obs.boxplot(by ='Weather', column =['Temperature'], grid = False)
obs.boxplot(by ='Weather', column =['Pressure'], grid = False)
obs.boxplot(by ='Weather', column =['Humidity'], grid = False)
obs.boxplot(by ='Weather', column =['Wind_Direction_degrees'], grid = False)
```

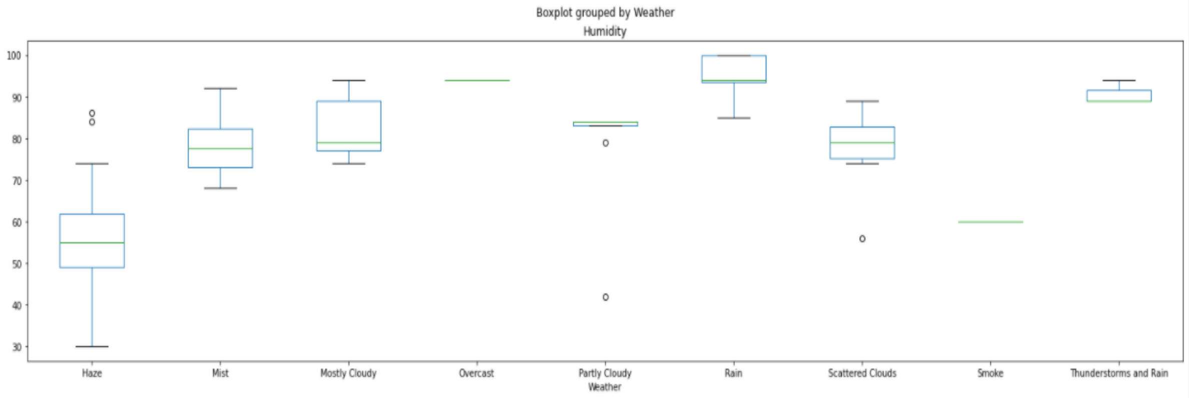
Temperature



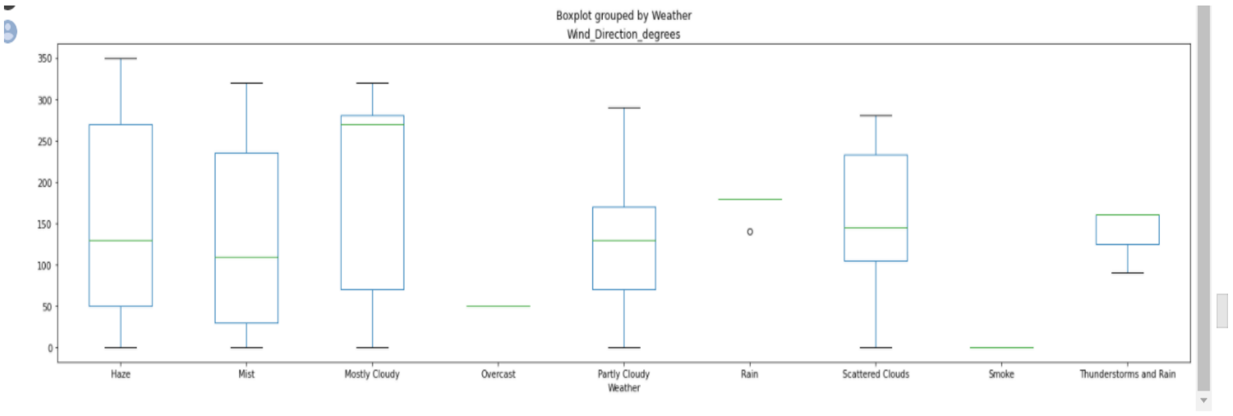
Pressure



Humidity

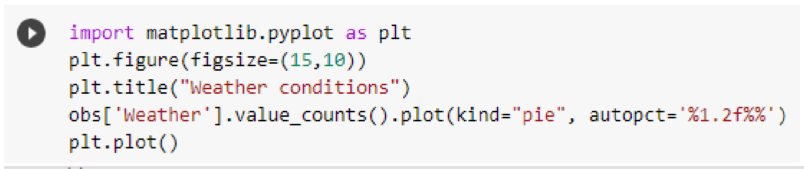


Wind direction

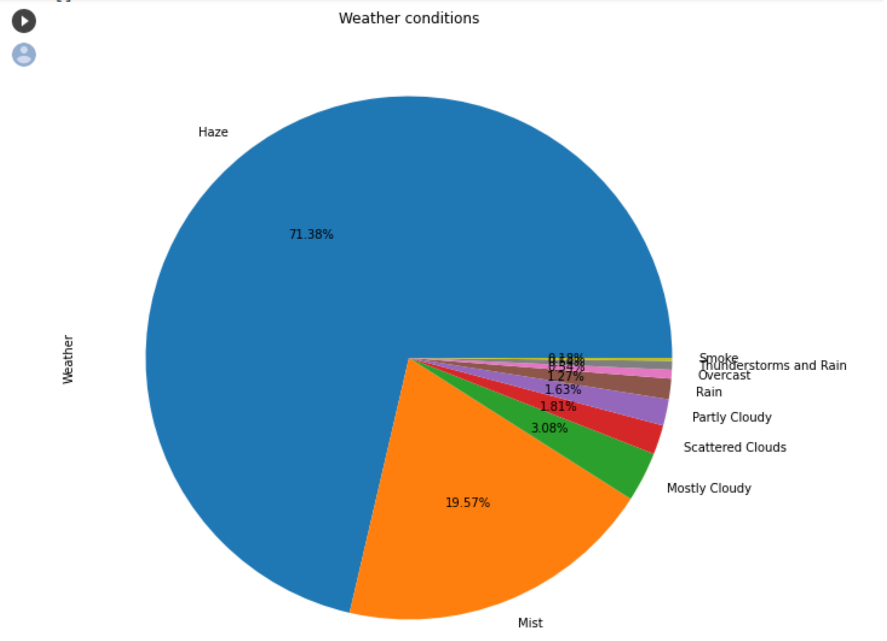


We had seen enough box plots to identify some relations. We wanted to find if there are any values that appear very consistently from the weather conditions and the wind direction. The best way to visualize that was to use pie chart. Therefore we made a 2 pie charts for the weather conditions and the wind direction.

Weather Conditions

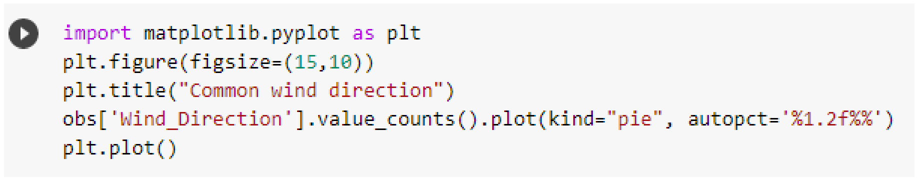


```
import matplotlib.pyplot as plt
plt.figure(figsize=(15,10))
plt.title("Weather conditions")
obs['Weather'].value_counts().plot(kind="pie", autopct='%1.2f%%')
plt.plot()
```

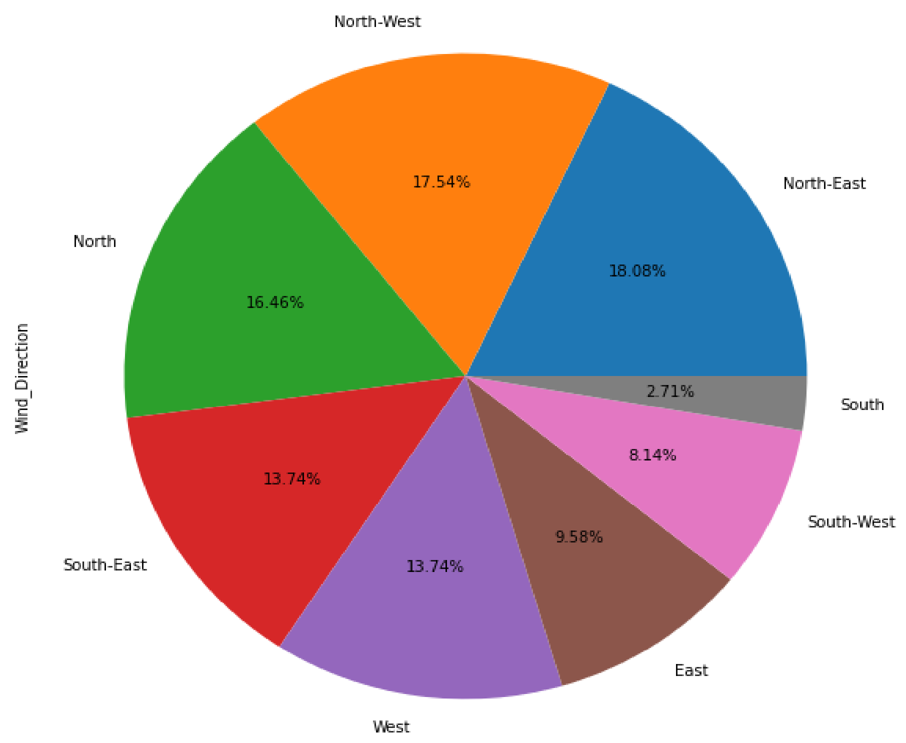


We could clearly see that Hazy conditions were the most common throughout. It makes a large portion of the pie chart more then 70% . The next common condition was mist with a much lower 19.57 percenatage.

Weather Directions



```
import matplotlib.pyplot as plt
plt.figure(figsize=(15,10))
plt.title("Common wind direction")
obs['Wind_Direction'].value_counts().plot(kind="pie", autopct='%1.2f%%')
plt.plot()
```



There is no major wind direction that was predominant.

We then moved on to normalize the data as we didn't know if they were normal distributions or not. We used the lambda method to do this. We will explain the lambda method with the formula that is used under it.

We use a formula to normalize our data which is given below. This will ensure that all the values will be between the interval 0 to 1 for our numerical data.

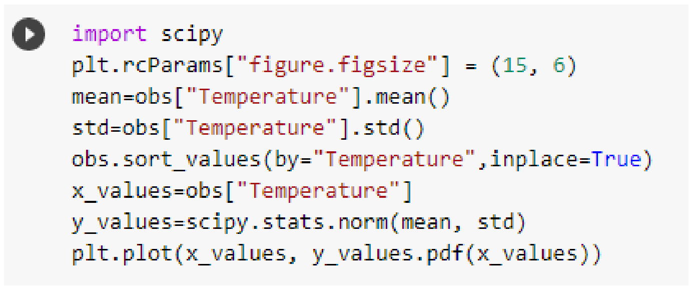
Normalization Formula

$$X_{normalized} = \frac{(X - X_{minimum})}{(X_{maximum} - X_{minimum})}$$

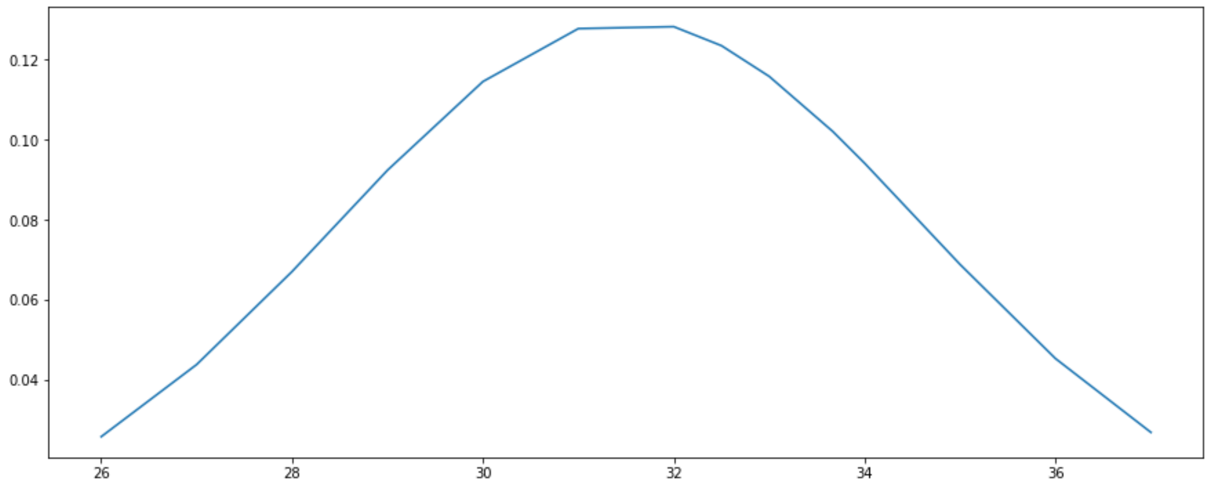
The formula makes use of the minimum and maximum value of the distribution to come up with a normalized value for the data that will be in between the interval 0 to 1. We then wanted to check for the normality of the data.

After the normalisation was done, we wanted to plot the normal probability curves to see if it was a Guassian distribution and had a bell shaped. We plotted these graphs for our 4 different numerical values. They are added below with the code as well

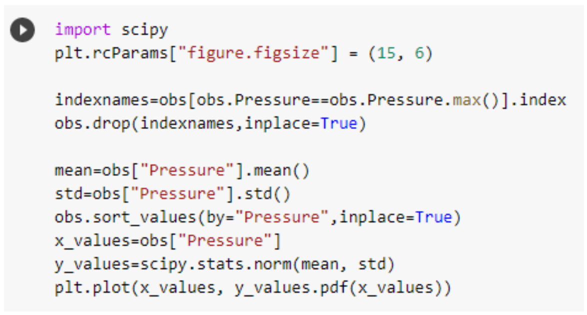
Temperature



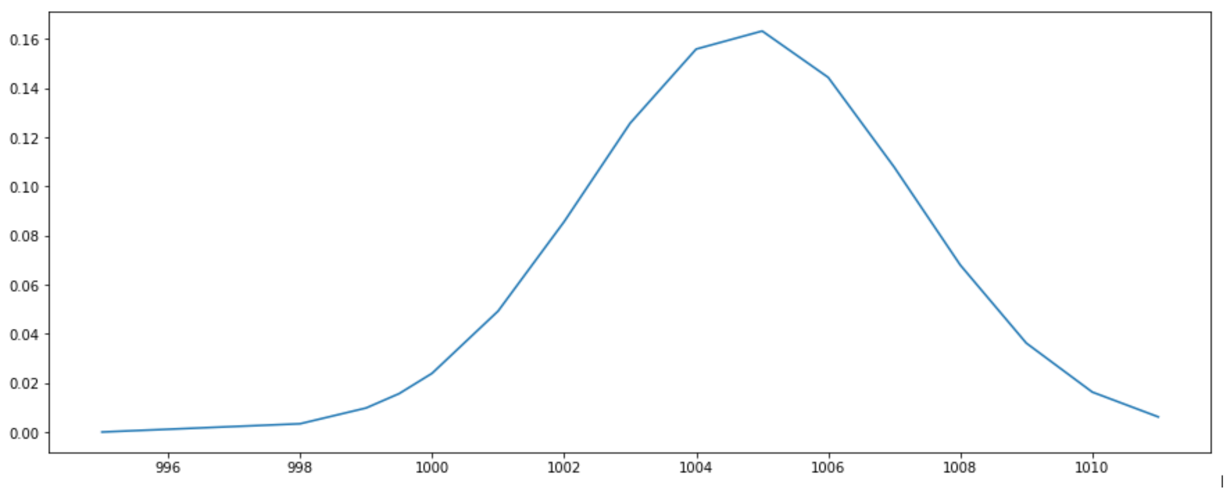
```
import scipy
plt.rcParams["figure.figsize"] = (15, 6)
mean=obs["Temperature"].mean()
std=obs["Temperature"].std()
obs.sort_values(by="Temperature",inplace=True)
x_values=obs["Temperature"]
y_values=scipy.stats.norm(mean, std)
plt.plot(x_values, y_values.pdf(x_values))
```



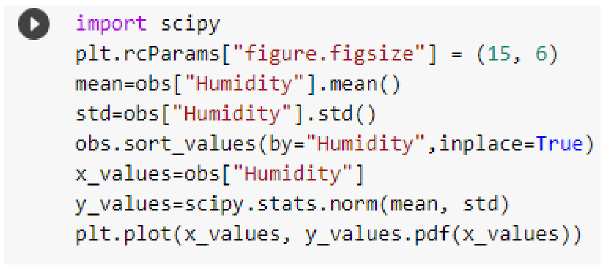
Pressure



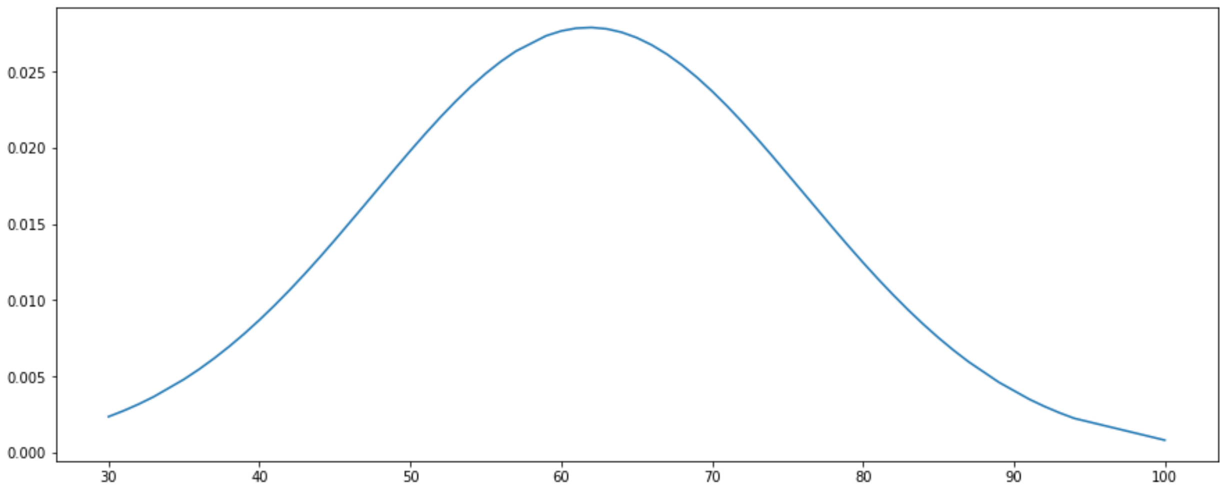
```
import scipy
plt.rcParams["figure.figsize"] = (15, 6)

indexnames=obs[obs.Pressure==obs.Pressure.max()].index
obs.drop(indexnames,inplace=True)

mean=obs["Pressure"].mean()
std=obs["Pressure"].std()
obs.sort_values(by="Pressure",inplace=True)
x_values=obs["Pressure"]
y_values=scipy.stats.norm(mean, std)
plt.plot(x_values, y_values.pdf(x_values))
```



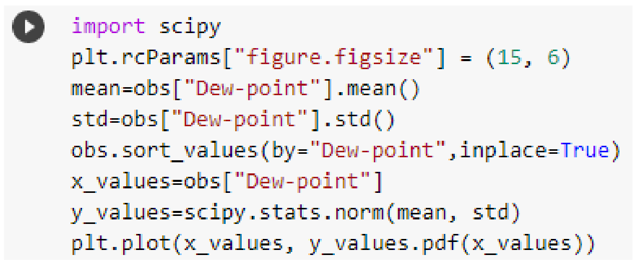
Humidity



```
import scipy
plt.rcParams["figure.figsize"] = (15, 6)
mean=obs["Humidity"].mean()
std=obs["Humidity"].std()
obs.sort_values(by="Humidity",inplace=True)
x_values=obs["Humidity"]
y_values=scipy.stats.norm(mean, std)
plt.plot(x_values, y_values.pdf(x_values))
```

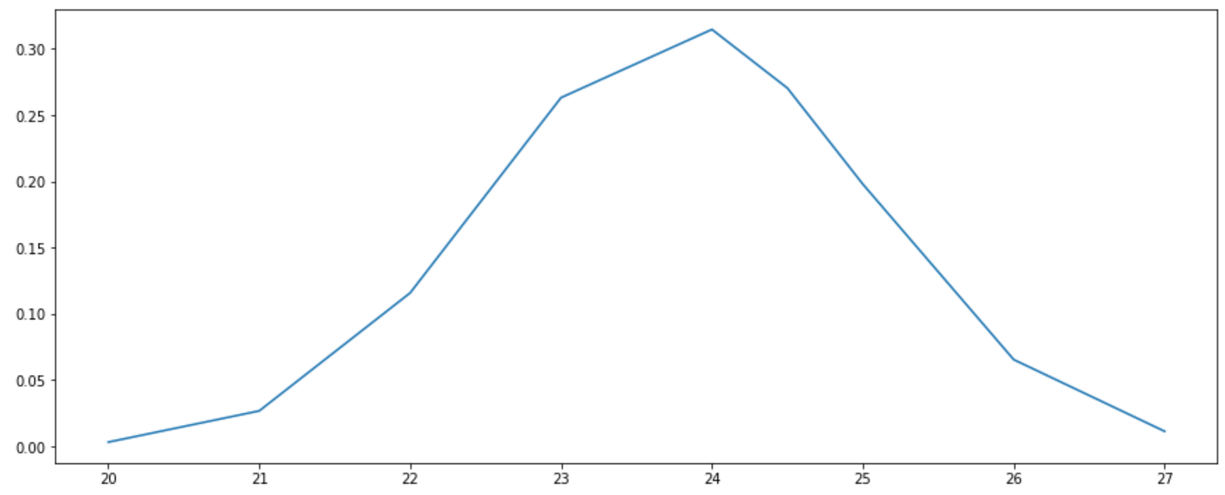


Dew Point



```
import scipy
plt.rcParams["figure.figsize"] = (15, 6)
mean=obs["Dew-point"].mean()
std=obs["Dew-point"].std()
obs.sort_values(by="Dew-point",inplace=True)
x_values=obs["Dew-point"]
y_values=scipy.stats.norm(mean, std)
plt.plot(x_values, y_values.pdf(x_values))
```

```
[<matplotlib.lines.Line2D at 0x7f063a0bb780>]
```



Seeing these plots we decided that the data is all normal as proved by the bell shaped curves that we get. We then move on do hypothesis testing to get to some conclusions about our visualized data.

We decided to test two hypothesis for their validity. They are explained in detail with the code and the way they are proved.

1)We have considered the temperature as our factor here. We would like to say with a guarantee of 0.05 that the temperature will not cross 38 Degree celsusis? Can we trust this guarantee?

Null Hypothesis: H_0: $\mu \geq 38$

Alternate Hypothesis: H_1: μ< 38

Therefore we will be doing a left tailed test to find the p-value to compare with the significance level we have taken to be 0.05.

```
mean=obs["Temperature"].mean()
std=obs["Temperature"].std()
zv1=(mean-38)/std
zv1
```

```
-2.1016749810666253
```

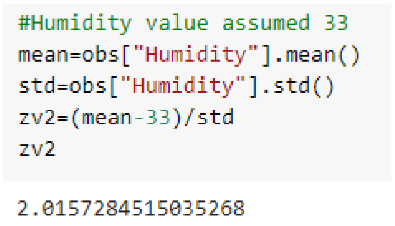
Since we saw the z-value that we got is -2.10 and this is a left tailed test we have to consider thet p-value to be that the p(z<-2.10)=0.0179. Since the significance level>P-value, we can reject our null hypothesis and say that the temperature cannot cross above 38 degree celsius in the month of September.

2)We have considered the humidity as our factor here. We would like to say with a guarantee of 0.05 that the humidity cannot be below 33%? Can we trust this guarantee?

Null Hypothesis: H_0: $\mu \leq 33$

Alternate Hypothesis: H_1: $\mu > 33$

Therefore we will be doing a right tailed test to find the p-value to compare with the significance level we have taken to be 0.05.



```
#Humidity value assumed 33
mean=obs["Humidity"].mean()
std=obs["Humidity"].std()
zv2=(mean-33)/std
zv2
```

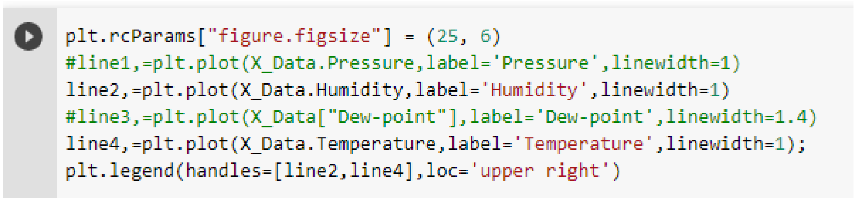
```
2.0157284515035268
```

Since we saw the z-value that we got is 2.01 and this is a right tailed test we have to consider that p-value to be that the p(z>2.01)=0.0222. Since the significance level>P-value, we can reject our null hypothesis and say that the humidity cannot go below 33% in the month of September.

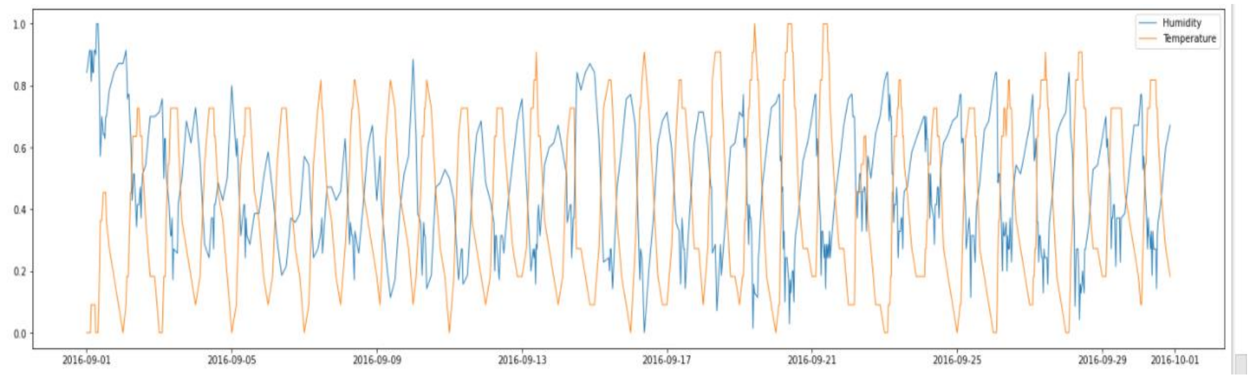
These were the two hypothesis that we had decided to test out. Then we wanted to find some correlations. We could see a clear correlations from a line graph that we had plotted to find them

1)Correlation between temperature and humidity

We plot a line graph to show the relation between these two.



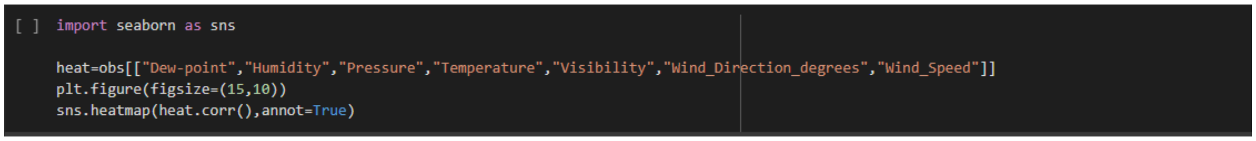
```
plt.rcParams["figure.figsize"] = (25, 6)
#line1,=plt.plot(X_Data.Pressure,label='Pressure',linewidth=1)
line2,=plt.plot(X_Data.Humidity,label='Humidity',linewidth=1)
#line3,=plt.plot(X_Data["Dew-point"],label='Dew-point',linewidth=1.4)
line4,=plt.plot(X_Data.Temperature,label='Temperature',linewidth=1);
plt.legend(handles=[line2,line4],loc='upper right')
```



We can clearly see that as temperature increases the humidity decreases every time. We can say confidently that there is inversely proportional relation between humidity and temperature.

2) Correlation between all the numeric variables using heatmap.

Here, we plot the heat map for all the variables to display the correlation coefficients against each other.



```
[ ] import seaborn as sns

    heat=obs[["Dew-point","Humidity","Pressure","Temperature","Visibility","Wind_Direction_degrees","Wind_Speed"]]
    plt.figure(figsize=(15,10))
    sns.heatmap(heat.corr(),annot=True)
```

The default method used by the function to find the correlation is the Pearson correlation coefficient method also known as the population correlation coefficient.

Given a pair of random variables X and Y, we have the population correlation coefficient ρ *given as*

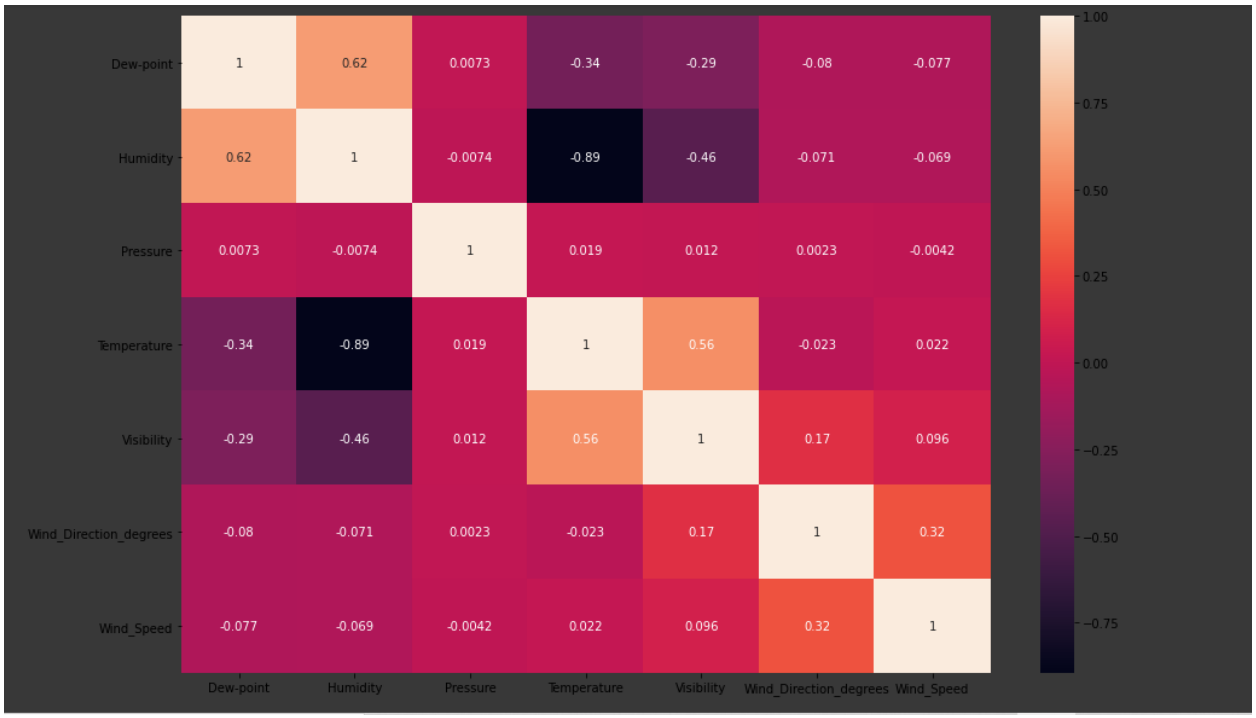
$$\boxed{\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} \quad \textbf{(Eq.1)}}$$

where:

cov is the covariance

σ_X is the standard deviation of X

σ_Y is the standard deviation of Y



The above plot proves that our line graph of humidity and dew-point being directly proportional to each other with a positive correlation coefficient of 0.62 and humidity being inversely proportional to temperature with a strong negative correlation of 0.89 is valid .

We have used 3 collab notebooks with the links to each being provided here. We will attach them as pythonnotebooks to the drive as well.

1)Cleaning Colab

https://colab.research.google.com/drive/1Wcn0o8MwoLlL5oyOj4JhJE18m-y119uh?usp=sharing

2)Initial Visualisation with hypothesis testing

https://colab.research.google.com/drive/1aCTzWPZrFv0STlvxefJ67WXGaBnbIzUt?usp=sharing

3)Final Visualisation with correlation

https://colab.research.google.com/drive/1m3KnW2eiYTI2iXGCmeBs8v7sAlCarPqw?usp=shari
ng