FinalProject_Weath...

FINISHED

```
import org.apache.spark.sql.functions._
import org.joda.time.format.DateTimeFormat
import org.apache.commons.io.IOUtils
import java.net.URL
import java.nio.charset.Charset
```

```
import org.apache.spark.sql.functions._
import org.joda.time.format.DateTimeFormat
import org.apache.commons.io.IOUtils
import java.net.URL
import java.nio.charset.Charset
```

Took 3 sec. Last updated by anonymous at March 30 2017, 8:00:37 PM.

FINISHED

```
%pyspark
import timeit

start = timeit.timeit()
from pandas import Series, DataFrame
import pandas as pd
import numpy as np

end = timeit.timeit()
print("Time taken", end - start)
```

```
('Time taken', -0.0005402565002441406)
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:00:39 PM.

FINISHED

```
%pyspark
#time taken:- 1 sec
start = timeit.timeit()

inputPath =  "/Users/aparwani/Downloads/raw_weather_data_aarhus"

dewpoint = pd.read_csv(inputPath+"/dewptm.csv")

humidity = pd.read_csv(inputPath+"/hum.csv")

pressure = pd.read_csv(inputPath+"/pressurem.csv")


temp = pd.read_csv(inputPath+"/tempm.csv")

winddirection = pd.read_csv(inputPath+"/wdird.csv")

end = timeit.timeit()
print("Time taken", end - start)
```

```
('Time taken', -0.008073091506958008)
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:01:09 PM.

FINISHED

```
%pyspark
start = timeit.timeit()
#time taken:- less than second
##concatenate the data frames column wise now
weather_dataset = pd.concat([dewpoint,humidity.ix[:,1],pressure.ix[:,1],temp.ix[:,1],winddire
end = timeit.timeit()
print("Time taken", end - start)
```

```
('Time taken', 0.0011937618255615234)
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:01:58 PM.

FINISHED

```
%pyspark
start = timeit.timeit()
#time taken:- 1 second
weather_dataset.ix[:,1:].sum()
end = timeit.timeit()
print("Time taken", end - start)
```

```
('Time taken', 0.0007460117340087891)
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:02:18 PM.

FINISHED

```
%pyspark
start = timeit.timeit()
#time taken:- less than second
weather_dataset.ix[:,1:].sum(axis=1)
end = timeit.timeit()
print("Time taken", end - start)
```

```
('Time taken', -0.0069158077239990234)
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:02:38 PM.

FINISHED

```
%pyspark
start = timeit.timeit()
#time taken:- less than second
print(weather_dataset.ix[:,1:].mean(axis=1,skipna=False))
end = timeit.timeit()
print("Time taken", end - start)
```

```
0       245.0
1       246.6
2       246.4
3       244.6
4       248.4
5       248.4
6       245.8
7       248.6
8       248.4
9       246.0
10      248.4
11      251.2
12      248.2
13      251.2
14      251.2
15      250.4
16      251.2
17      251.0
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:03:49 PM.

FINISHED

```
%pyspark
start = timeit.timeit()
#time taken:- less than second
print(weather_dataset.ix[:,1:].idxmax())
end = timeit.timeit()
print("Time taken", end - start)
```

```
Dewpoint         6910
Humidity          374
Pressure         1850
Temperature      6979
Winddirection    1869
dtype: int64
('Time taken', -0.0044400691986083984)
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:03:36 PM.

FINISHED

```
%pyspark
start = timeit.timeit()
#time taken:- 3 second
print(weather_dataset.describe())
end = timeit.timeit()
print("Time taken", end - start)
```

```
          Dewpoint     Humidity     Pressure  Temperature  Winddirection
count  8207.000000  8207.000000  8203.000000  8207.000000    8146.000000
mean      4.229073    71.157914  1013.638059     9.061167     182.431868
std       3.979373    16.132295     9.515477     4.830610      89.646687
min      -9.000000    12.000000   986.000000    -3.000000       0.000000
25%       2.000000    61.000000  1007.000000     5.000000     110.000000
50%       3.000000    75.000000  1014.000000     8.000000     190.000000
75%       7.000000    84.000000  1020.000000    12.000000     260.000000
max      15.000000   100.000000  1038.000000    25.000000     360.000000
('Time taken', -0.0042760372161865234)
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:04:19 PM.

```
%pyspark
start = timeit.timeit()
#time taken:- less than second
#check the null values in dataframe if any
print(weather_dataset.isnull().any())
end = timeit.timeit()
print("Time taken", end - start)
```

FINISHED

```
DateTime         False
Dewpoint          True
Humidity          True
Pressure          True
Temperature       True
Winddirection     True
dtype: bool
('Time taken', -0.002516031265258789)
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:04:49 PM.

```
%pyspark
#time taken:- less than second
start = timeit.timeit()
#In this step, we will try to update null values.
#filling null values could be complicated.As we seen in previous data exploration steps
#that 116 was the maximum null values and total datasize is 12563. Since, maximum percent of n
#So, null values will be replaced by mean of the particular parameter.
def updatenullvalues(dataset):
    for col in dataset.ix[:,1:]:
        if dataset[col].isnull().any:
            mean = dataset[col].mean()
            dataset[col].fillna(mean,inplace=True)
    return dataset
end = timeit.timeit()
print("Time taken", end - start)
```

FINISHED

```
('Time taken', -0.006827116012573242)
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:05:18 PM.

```
%pyspark
#time taken:- less than second

#Let's update null values in our dataset.
weather_dataset = updatenullvalues(weather_dataset)
#verify is there still any null value left in the dataset
weather_dataset.isnull().sum()
```

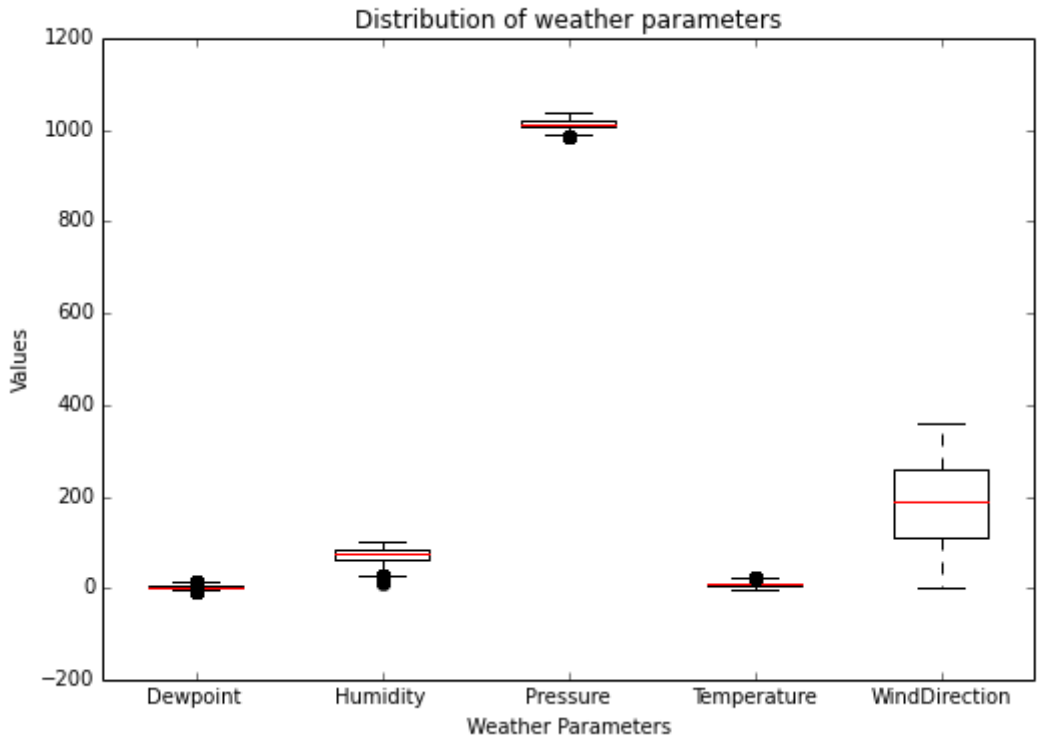
FINISHED

```
DateTime         0
Dewpoint         0
Humidity         0
Pressure         0
Temperature      0
Winddirection    0
dtype: int64
```

Took 0 sec. Last updated by anonymous at March 30 2017, 6:46:13 PM.

FINISHED

```
%pyspark
#time taken:- 2 second

import matplotlib.pyplot as plt

#now we are in good state as our null values are vanished.
#in this code step, we will check distribution of our data.

data = [weather_dataset.ix[:,1], weather_dataset.ix[:,2], weather_dataset.ix[:,3], weather_da

parameter_names = ['Dewpoint', 'Humidity', 'Pressure', 'Temperature', 'WindDirection']

fig, axis = plt.subplots()
axis.set_title("Distribution of weather parameters")
axis.set_xlabel('Weather Parameters')
axis.set_ylabel('Values')
day_plot = plt.boxplot(data, sym='o', vert=1, whis=1.5)
plt.setp(day_plot['boxes'], color = 'black')
plt.setp(day_plot['whiskers'], color = 'black')
plt.setp(day_plot['fliers'], color = 'black', marker = 'o')
axis.set_xticklabels(parameter_names)
plt.show()
```



Took 0 sec. Last updated by anonymous at March 30 2017, 6:47:10 PM.

FINISHED

```
%pyspark
start = timeit.timeit()
#time taken:- less than second
#now, let's perform data aggregation to know the hidden facts of dataset

# column-wise and Multiple Function Application
grouped_dewpoint = weather_dataset.groupby(['Dewpoint'])
```

```
end = timeit.timeit()
print("Time taken", end - start)
```

```
('Time taken', 0.0031151771545410156)
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:06:03 PM.

```
%pyspark                                                                    FINISHED
start = timeit.timeit()
#time taken:- less than second
# column-wise and Multiple Function Application
grouped_pressure = weather_dataset.groupby(['Pressure'])
end = timeit.timeit()
print("Time taken", end - start)
```

```
('Time taken', -0.0054857730865478516)
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:06:33 PM.

```
%pyspark                                                                    FINISHED
start = timeit.timeit()
print(grouped_dewpoint.apply(lambda weather_dataset: weather_dataset['Humidity'].corr(weather_
end = timeit.timeit()
print("Time taken", end - start)
```

```
Dewpoint
-9.0          NaN
-8.0     1.000000
-7.0    -0.398579
-6.0    -0.878914
-5.0    -0.802528
-4.0    -0.960769
-3.0    -0.963175
-2.0    -0.954655
-1.0    -0.946108
 0.0    -0.927748
 1.0    -0.932679
 2.0    -0.942768
 3.0    -0.954112
 4.0    -0.949298
 5.0    -0.952773
 6.0    -0.957771
 7.0    -0.956232
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:07:12 PM.

```
%pyspark                                                                    FINISHED
start = timeit.timeit()
#time taken:- less than second

# column-wise and Multiple Function Application
grouped_humidity = weather_dataset.groupby(['Humidity'])
```

```
# get an idea of average windspeed at different levels of humidity
print(grouped_humidity['Winddirection'].agg('mean'))
end = timeit.timeit()
print("Time taken", end - start)
```

```
Humidity
12.0      40.000000
13.0      60.000000
15.0     156.666667
16.0     230.000000
17.0     130.000000
18.0     265.000000
19.0     208.571429
20.0     213.750000
21.0     198.000000
22.0     160.000000
24.0      83.750000
25.0     150.000000
26.0     191.764706
27.0     200.000000
28.0     194.444444
29.0     182.307692
30.0     146.153846
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:07:41 PM.

```
%pyspark
start = timeit.timeit()
#Correlation humidity & dewpoint
print(grouped_humidity.apply(lambda weather_dataset: weather_dataset['Dewpoint'].corr(weather_
end = timeit.timeit()
print("Time taken", end - start)
```

FINISHED

```
Humidity
12.0          NaN
13.0          NaN
15.0     1.000000
16.0     1.000000
17.0     0.999884
18.0     1.000000
19.0     0.989840
20.0     0.998635
21.0     0.989782
22.0     0.825323
24.0     0.844822
25.0     0.943456
26.0     0.759595
27.0     0.998819
28.0     0.765339
29.0     0.915968
30.0     0.876387
```

Took 1 sec. Last updated by anonymous at March 30 2017, 8:08:20 PM.

```
%pyspark
#time taken:- less than second

#let's check correlation between windspeed and other variables
```

FINISHED

```
from scipy.stats.stats import pearsonr
```

```
Help on function pearsonr in module scipy.stats.stats:
pearsonr(x, y)
    Calculates a Pearson correlation coefficient and the p-value for testing
    non-correlation.

    The Pearson correlation coefficient measures the linear relationship
    between two datasets. Strictly speaking, Pearson's correlation requires
    that each dataset be normally distributed. Like other correlation
    coefficients, this one varies between -1 and +1 with 0 implying no
    correlation. Correlations of -1 or +1 imply an exact linear
    relationship. Positive correlations imply that as x increases, so does
    y. Negative correlations imply that as x increases, y decreases.

    The p-value roughly indicates the probability of an uncorrelated system
    producing datasets that have a Pearson correlation at least as extreme
    as the one computed from these datasets. The p-values are not entirely
    reliable but are probably reasonable for datasets larger than 500 or so.
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:09:04 PM.

```
%pyspark                                                                    FINISHED
#time taken:- less than second
start = timeit.timeit()
pearsonr(weather_dataset['Humidity'], weather_dataset['Temperature'])
print("Pearson's correlation coefficient, between humidity & temperature",pearsonr(weather_da
    weather_dataset['Temperature'])[0])

end = timeit.timeit()
print("Time taken", end - start)
```

```
("Pearson's correlation coefficient, between humidity & temperature", nan)
('Time taken', -0.0017211437225341797)
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:22:56 PM.

```
%pyspark                                                                    FINISHED
start = timeit.timeit()
import statsmodels.api as sm
def regression(data, yvar, xvars):
  Y = data[yvar]
  X = data[xvars]
  X['intercept'] = 1.
  result = sm.OLS(Y,X).fit()
  return result.params
end = timeit.timeit()
print("Time taken", end - start)
```

```
('Time taken', -0.004781007766723633)
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:13:02 PM.

```
%pyspark                                                                    FINISHED
start = timeit.timeit()
#Linear regression (dependent variable: Dewpoint, independent variable: Temperature & Winddir
grouped_humidity.apply(regression,'Dewpoint',['Temperature'])
```

```
end = timeit.timeit()
print("Time taken", end - start)
```

```
('Time taken', 0.00022912025451660156)
```

Took 0 sec. Last updated by anonymous at March 30 2017, 8:14:26 PM.

```
%pyspark
```

READY