MSBD5002 Individual Project Report

Haojun Chen

The Hong Kong University of Science and Technology

Abstract

This paper is about forecasting several air pollutants over the 48 hours for 35 stations in Beijing, China. The prediction framework is consisting of four parts: data preprocessing, feature engineering, modeling and evaluation output and conclusion.

Introduction

The requirement provided air quality data and meteorological(weather) data from January 2017 to April 30 (including) 2018 and we need to predict the pollution level of PM2.5, PM10, O3 between May 1 to May 2, 2018 (once an hour, 48 times for one station in total). Besides, the processed data folder has the following csv files:

- **air quality data includes:** station_id,utc_time, PM2.5,PM10,NO2,CO,O3,SO2
- **grid weather includes:** id,station_id,utc_time,weather,temperature,pressure,humidity,wind_direction, wind_speed
- **observed weather includes:** id,station_id,utc_time,temperature,pressure,humidity,wind_direction,wind_speed,weather

In those files, **station_id** is unique and utc_time is limited to seconds (YY-MM-DD HH:MM: SS). **Weather** is categorical, and other features such as pm2.5, pm10, O3, pressure, and humidity, etc. are numerical data. We are going to predict the PM2.5, PM10 and O3 through history data from Beijing air history in upcoming 48 hours (May 1 to May 2).

Data Preprocessing

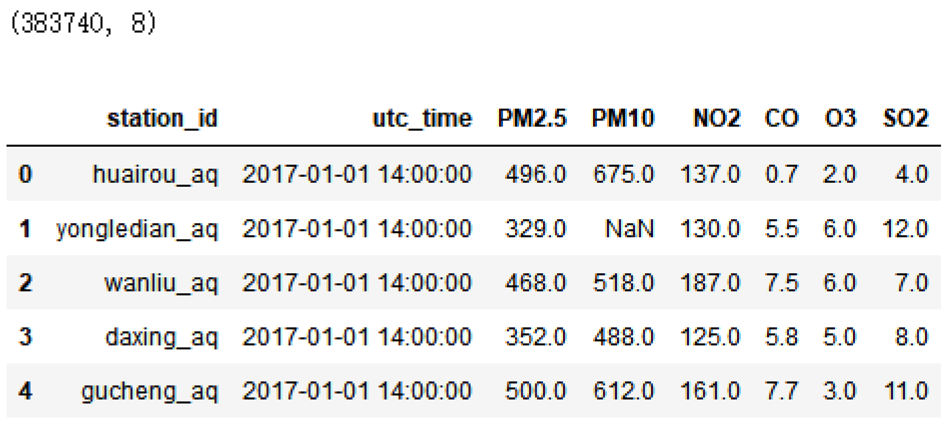
1. Data Integration

From the files I've got, all the data was confusing so I merged all of the **air quality** data into a file named **air_quality.csv**, however, the columns of data are mixed and different and some are even incorrect. What I think is that this is a human wrong or equipment faults when collecting and inputting data. Hence, I merged all the **grid weather** into **grid_weather.csv** and observed the weather by doing the same thing.

At this step, I have implemented some methods to fix the confusing data that include the same columns but named differently, same columns but different data. For example, on column **‘utc_time’,** some of the data format is like ‘2018-01-01 23:00:00’, while some is like ‘2018/5/1/ 13:00:00’. Thus, I have to transform them to a uniformed format: ‘YY-mm-dd HH:MM: SS’. Then I will be able to do the remaining steps.

2.Missing Value Imputation

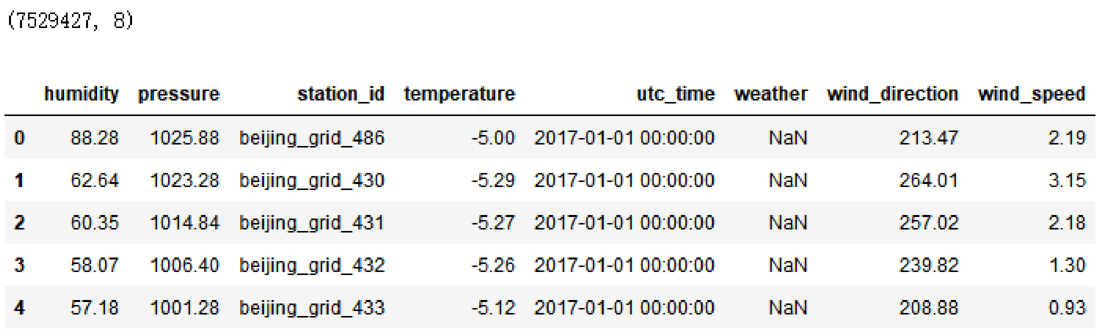
To get the overview of the data in each csv file (air_quality, grid_weather and observed_weather). I used the head. () function to get the top five rows for each table (1 to 3) and check the format of all the column values. In this case, they are all strings.



```
(383740, 8)
```

	station_id	utc_time	PM2.5	PM10	NO2	CO	O3	SO2
0	huairou_aq	2017-01-01 14:00:00	496.0	675.0	137.0	0.7	2.0	4.0
1	yongledian_aq	2017-01-01 14:00:00	329.0	NaN	130.0	5.5	6.0	12.0
2	wanliu_aq	2017-01-01 14:00:00	468.0	518.0	187.0	7.5	6.0	7.0
3	daxing_aq	2017-01-01 14:00:00	352.0	488.0	125.0	5.8	5.0	8.0
4	gucheng_aq	2017-01-01 14:00:00	500.0	612.0	161.0	7.7	3.0	11.0

Table 1. Air_Quality

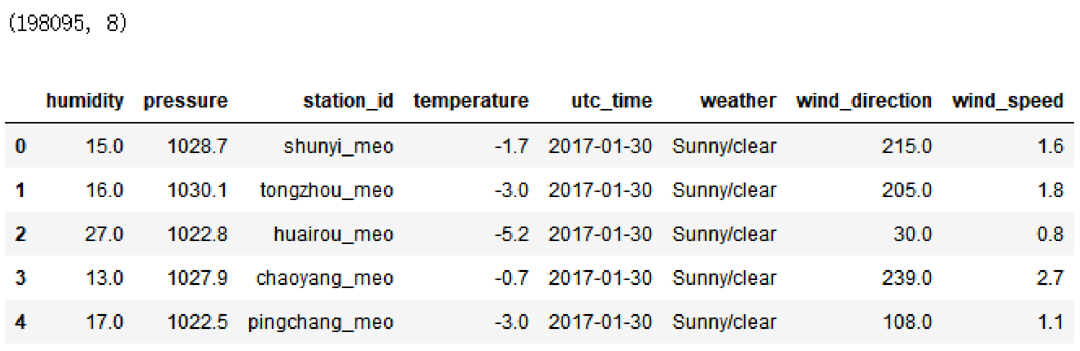


```
(7529427, 8)
```

	humidity	pressure	station_id	temperature	utc_time	weather	wind_direction	wind_speed
0	88.28	1025.88	beijing_grid_486	-5.00	2017-01-01 00:00:00	NaN	213.47	2.19
1	62.64	1023.28	beijing_grid_430	-5.29	2017-01-01 00:00:00	NaN	264.01	3.15
2	60.35	1014.84	beijing_grid_431	-5.27	2017-01-01 00:00:00	NaN	257.02	2.18
3	58.07	1006.40	beijing_grid_432	-5.26	2017-01-01 00:00:00	NaN	239.82	1.30
4	57.18	1001.28	beijing_grid_433	-5.12	2017-01-01 00:00:00	NaN	208.88	0.93

Table 2. Grid_Weather

Notice that there are missing values NaN, I went ahead and applied the padding method from Pandas.



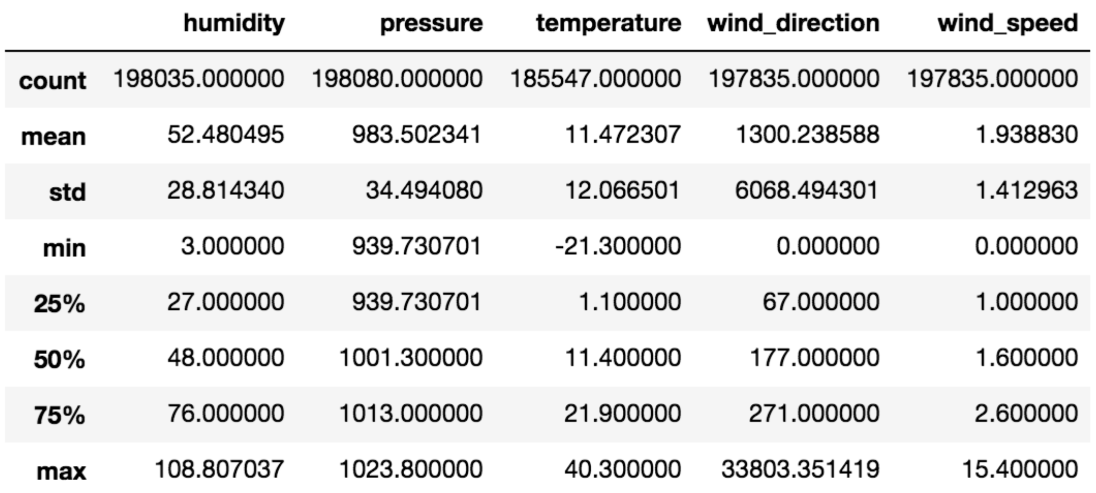
```
(198095, 8)
```

	humidity	pressure	station_id	temperature	utc_time	weather	wind_direction	wind_speed
0	15.0	1028.7	shunyi_meo	-1.7	2017-01-30	Sunny/clear	215.0	1.6
1	16.0	1030.1	tongzhou_meo	-3.0	2017-01-30	Sunny/clear	205.0	1.8
2	27.0	1022.8	huairou_meo	-5.2	2017-01-30	Sunny/clear	30.0	0.8
3	13.0	1027.9	chaoyang_meo	-0.7	2017-01-30	Sunny/clear	239.0	2.7
4	17.0	1022.5	pingchang_meo	-3.0	2017-01-30	Sunny/clear	108.0	1.1

Table 3. Observed_Weather

3. Data Cleansing

I used the describe. () function to obtain the table for each file (air_quality, grid_weather and observed_weather) after defining the upper limit and lower limit of the data. And then, based on the definition of outliers in boxplot analysis, certain outliers like 99,999 are padded. The result is as follows in table 4:



	humidity	pressure	temperature	wind_direction	wind_speed
count	198035.000000	198080.000000	185547.000000	197835.000000	197835.000000
mean	52.480495	983.502341	11.472307	1300.238588	1.938830
std	28.814340	34.494080	12.066501	6068.494301	1.412963
min	3.000000	939.730701	-21.300000	0.000000	0.000000
25%	27.000000	939.730701	1.100000	67.000000	1.000000
50%	48.000000	1001.300000	11.400000	177.000000	1.600000
75%	76.000000	1013.000000	21.900000	271.000000	2.600000
max	108.807037	1023.800000	40.300000	33803.351419	15.400000

Table 4 New_Observed_Weather

Feature Engineering

1. Original Features

The features like air_quality, observed_weather and grid_weather come from the original file.

For air_quality I used: **utc_time, PM2.5, PM10, O3,** additionally, **PM2.5, PM10, O3 as my label.** For **observed_weather** I used: **utc_time, temperature, pressure,**

humidity, wind_direction, wind_speed, weather

2. Other Features

Weather Region

For weather like Sunny/Clear, Hail, Cloudy and Rain**.** I hashed them into different numbers. For example, Hail is denoted as 1, Sunny/Clear is denoted as 2, Cloudy **is** denoted as 3, Rain is denoted as 4 and so on.

Utc_time Region

For all the utc_times, I observed the statistics of each day's characters, for example, this day is weekend, holiday or the first day of work day, or the end day of work day, work day, the end of a specific holiday in China, the first day of a specific holiday in China and so on. Besides, if the day is the weekend, I mark it as 1, otherwise, mark it as 0. Furthermore, the weekend itself can be a column in features. Others like work day, holiday, I applied the same thing.

Exploratory Data Analysis

In order to better understand the correlation and the trend of data, I randomly picked one station (huairou) to see the trend of O3, PM2.5 and PM 10 over time (figure1 to 3), ranging from 0 to 10,000 (time series data from 1/1/2017 to 30/4/2018). Although all the 35 stations can be plotted separately, for simplicity, I only chose three stations (huairou, xizhimen and donggaocun) to see the trend of PM2.5 level for each station. PM2.5 concentration (y axis) is corresponding to the time series (x-axis) including holiday, work day and so on. Intuitively, the PM2.5 values are distinctly different over time, indicating a periodicity property of the time series data. Geographically, among the three stations I picked, I see that for huairou station, there is only one day when the PM2.5 level exceeds 1,400 ug/m3 and that the rest of the time the PM2.5 is at a relatively low-level ranging from 50 ug/m3 to 200 ug/m3, with only three days ranging from 400 ug/m3 to 600 ug/m3. This makes sense in real life because Huairou is a suburban district that is far from urban Beijing. The pollution level is not as bad as that of the urban area. In contrast, Xizhimenbei(figure 4) station, as it is located in the urban area, the overall PM2.5 level is significantly higher than that of Huairou with around 10 days exceeding 300 ug/m3. Conversely, the PM2.5 concentration for Donngaocun(figure 5), as a rural town that is about 70 kilometers away from urban Beijing, is fairly low with only 2 days exceeding 400 ug/m3 and 1 day approximately close to 1,000 ug/m3.

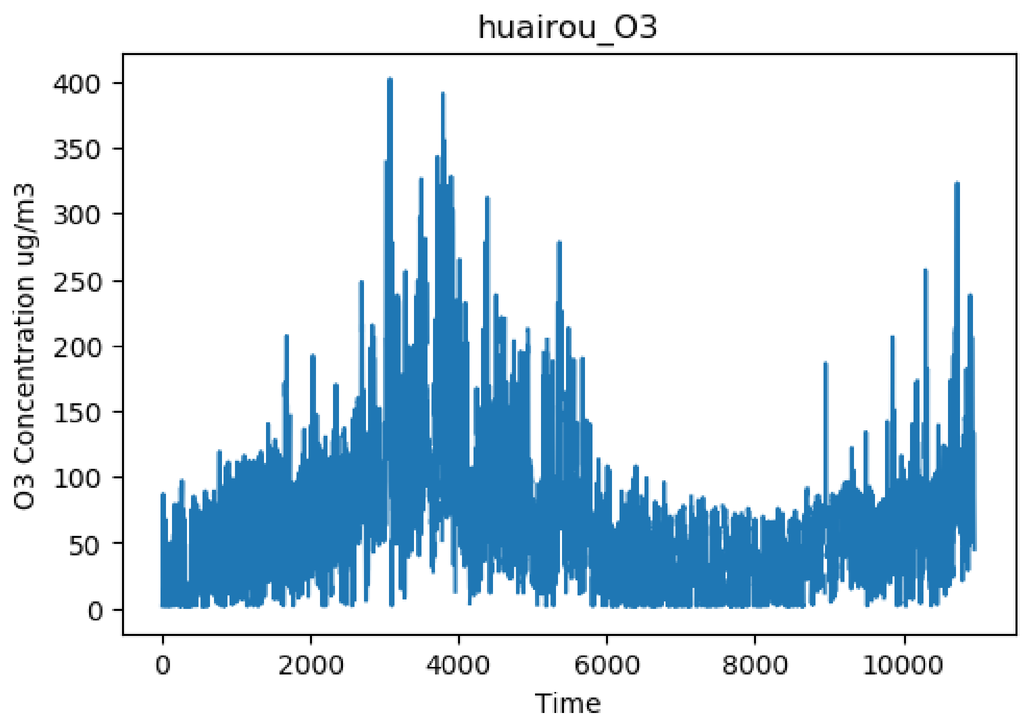


Figure.1 The Concentration Level of O3 for Huairou Station

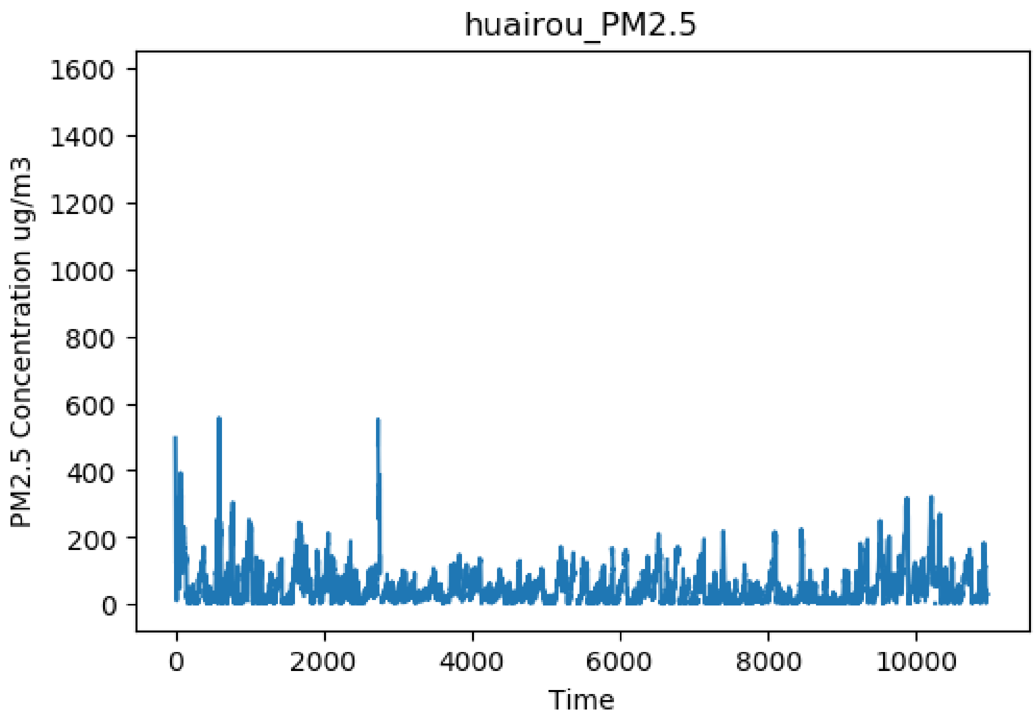


Figure.2 The Concentration Level of PM2.5 for Huairou Station

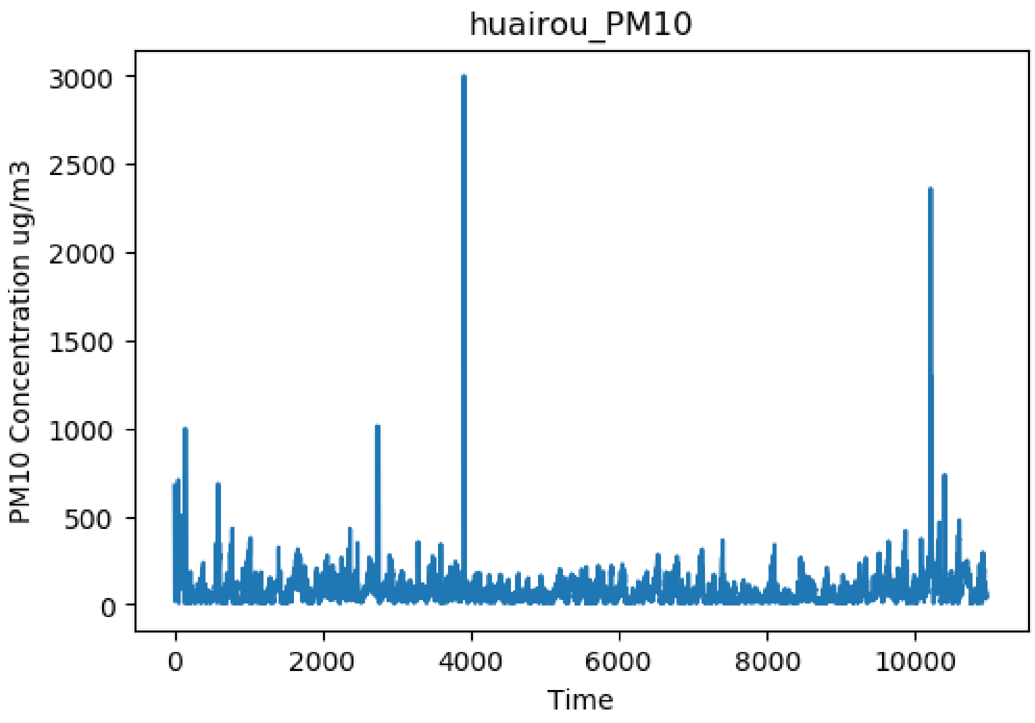


Figure.3 The Concentration Level of PM10 for Huairou Station

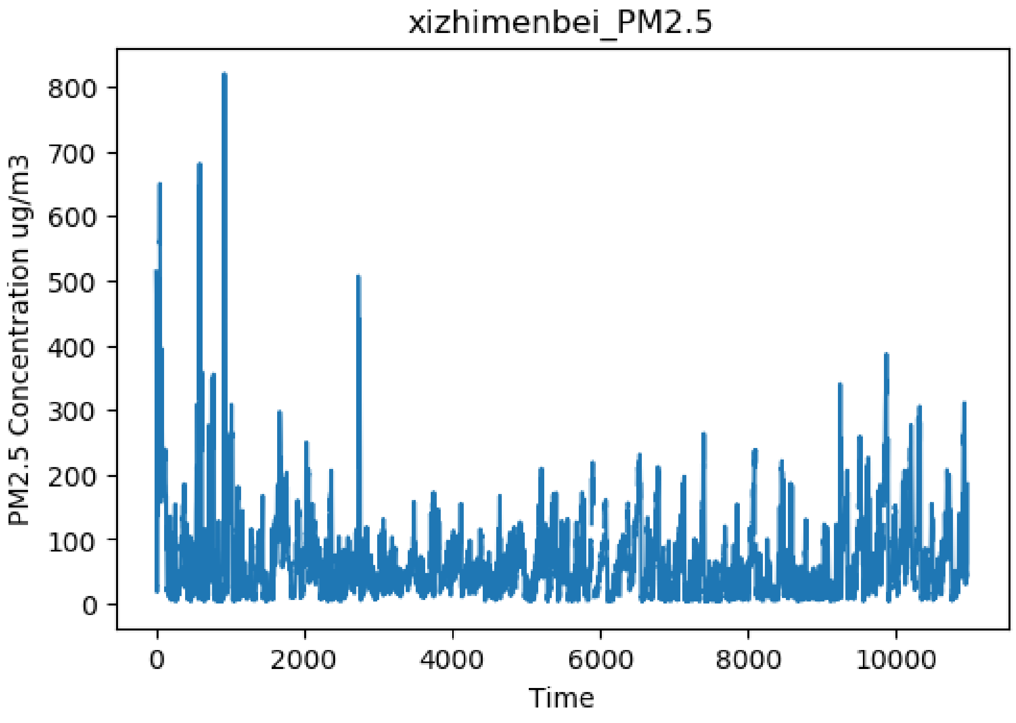


Figure.4 The Concentration Level of PM2.5 for Xizhimenbei Station

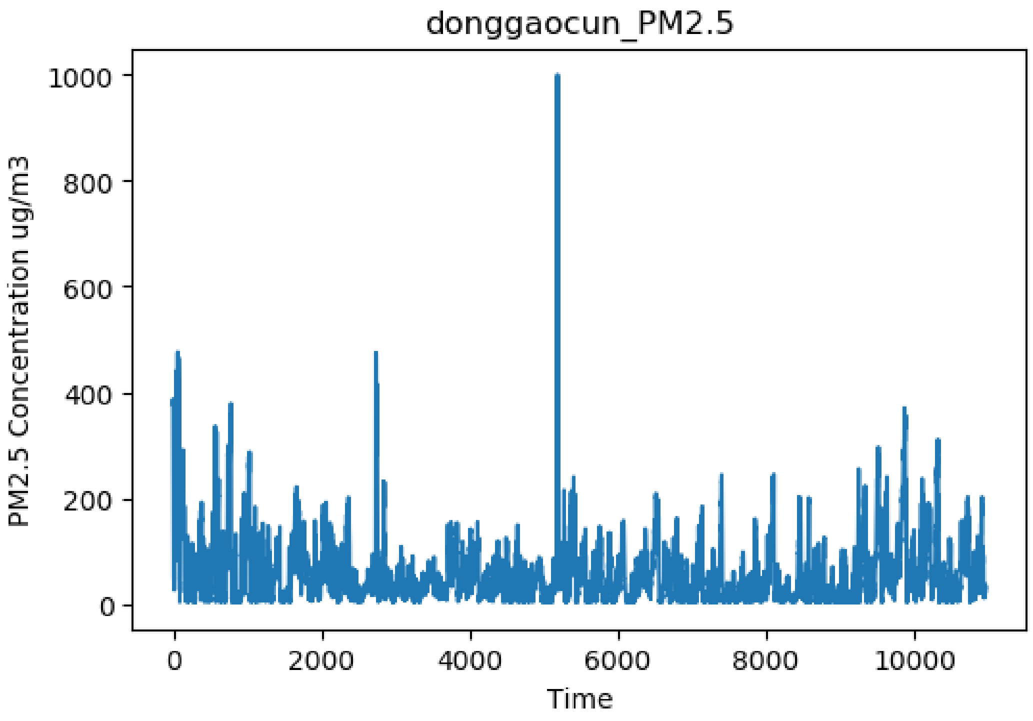


Figure.5 The Concentration Level of PM2.5 for Donggaocun Station

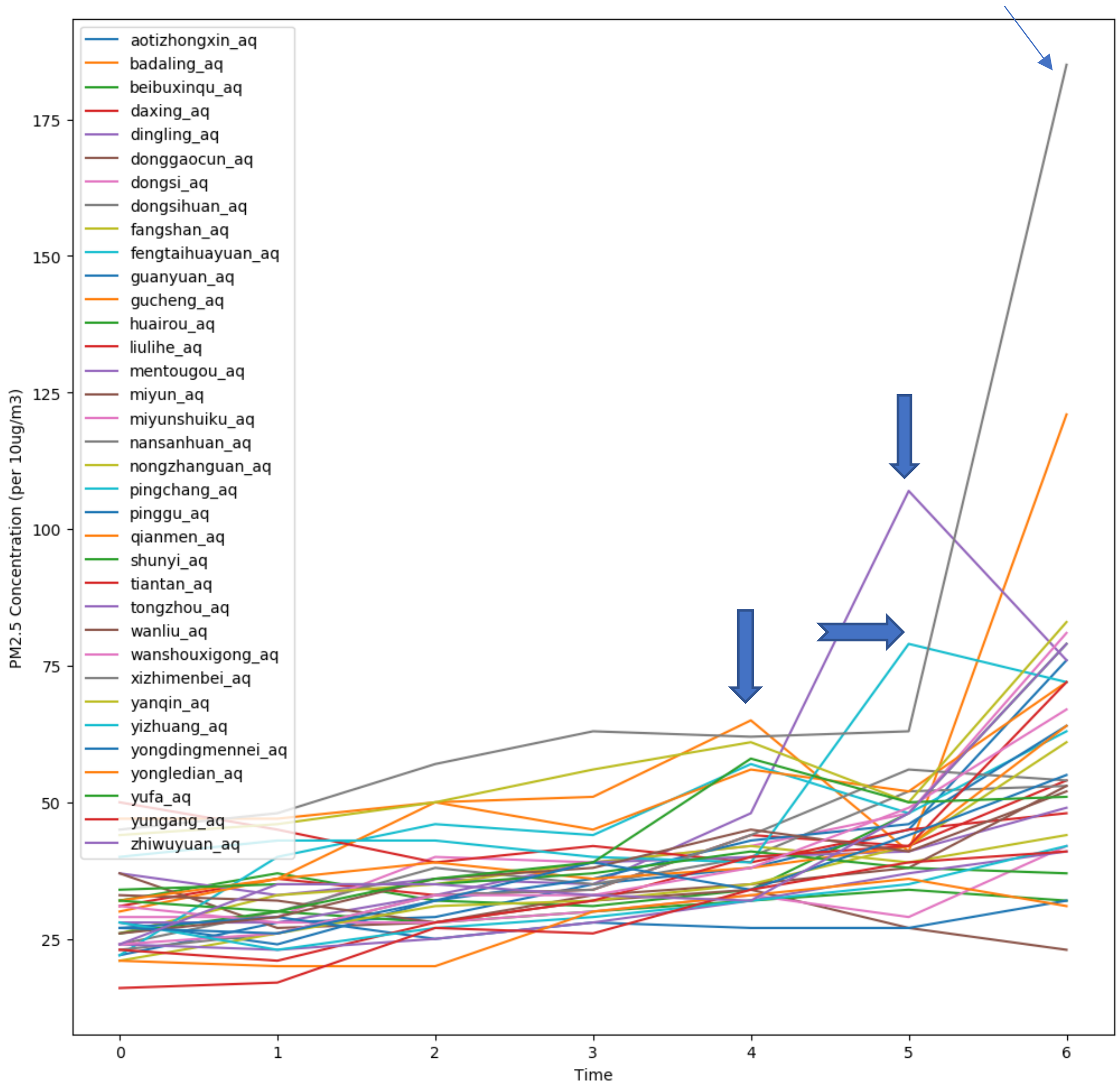


Figure.6 The Fluctuation of PM2.5 Concentration from Jan 2017 to Apr 2018

The figure 6 above shows a fluctuation of PM2.5 concentration in the course of time (January 1st 2017 to April 30th 2018). The time (x axis) is divided into 6 periods thereby each one dot of time stands for about two and a half months in one year while each line encodes 35 stations respectively. We can clearly conclude that, the concentration of PM2.5 was fairly stable during the year except in the holiday like the Chinese National Day, the Chinese Spring Festival when the air pollutant level reaches its peak. It makes sense because people tend to drive out of town in holiday and the emissions from the vehicles contribute to the potential increase of PM2.5 and other pollutant.

Next, I did the feature correlation heatmap by using the seaborn, a data visualization library that is based on matplotlib.

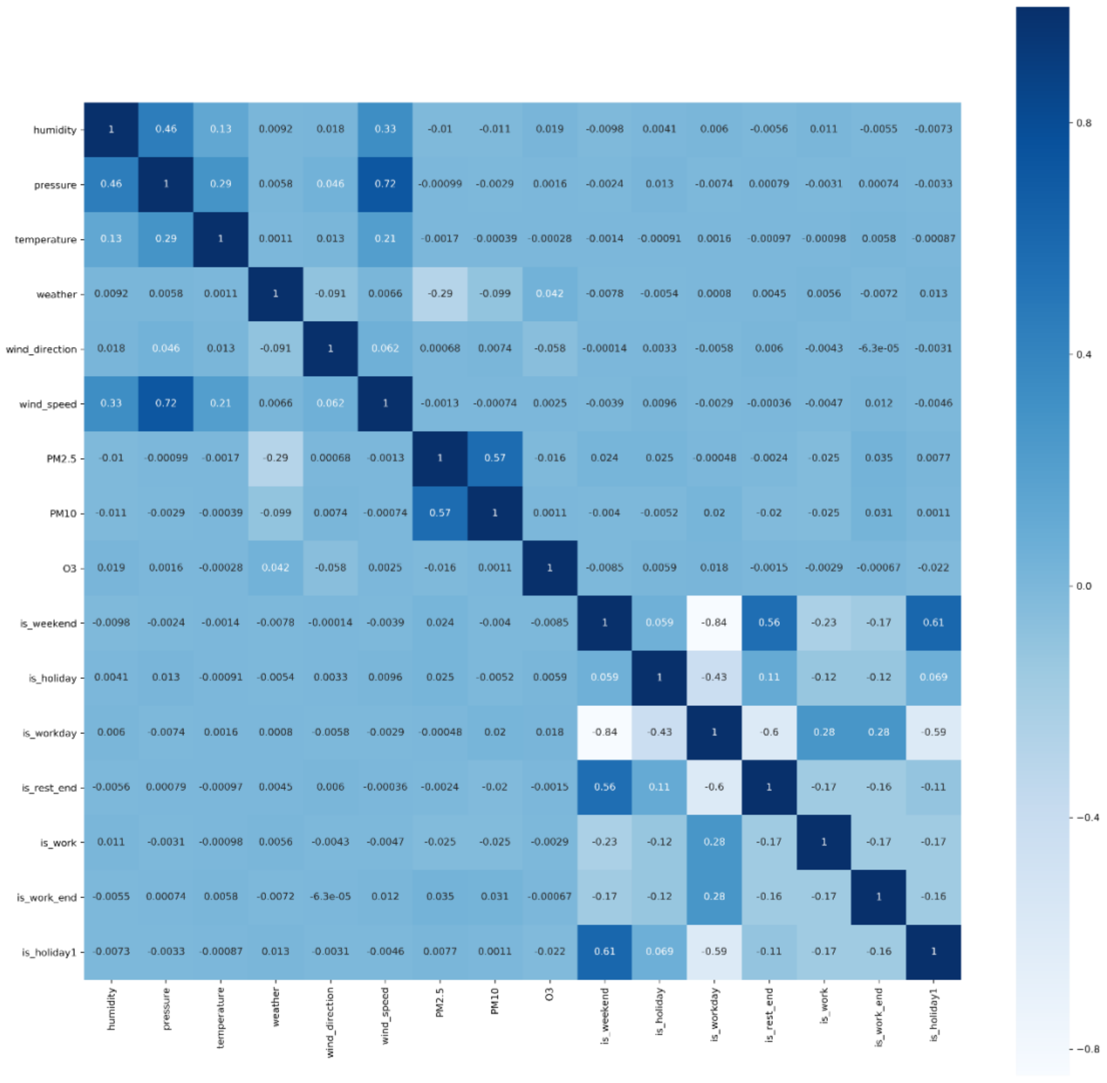


Figure.7 Correlation Plot Matrix for All Features

From the correlation plot matrix (figure 7) above, we can see that the closer the absolute value is to 1, the more correlated the corresponding two features are, vice versa. Wind speed seems to be highly correlated with pressure; PM2.5 seems to be strongly correlated with PM10; is_week also seems to be highly correlated with is_holiday1. While is_weekend seems the least correlated with is_workday and is_workday is also the least correlated with is_rest_end.

The importance of the features provides a score that indicates how useful each feature was in the construction of the boosted decision trees within the model. The values are ranging from 0 to 1; a higher the value means that the feature will have a higher effect on the outputs, vice versa.

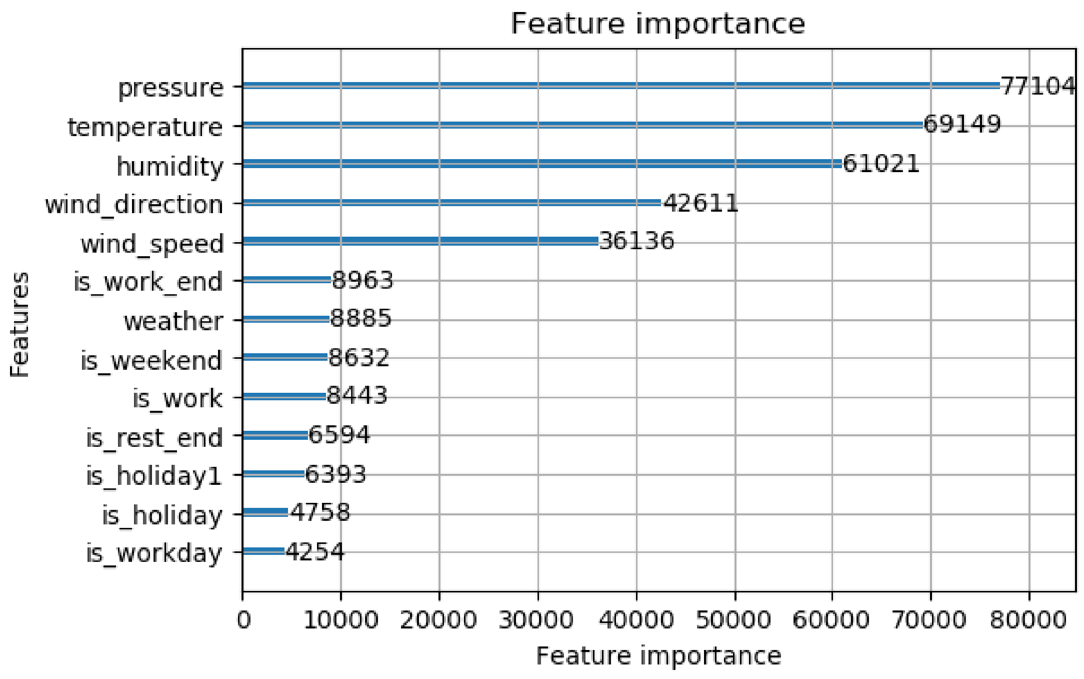


Figure.8 Feature Importance Plot for All Features

From the Feature Importance bar plot (figure.8) above, we can see that pressure, temperature, humidity, wind_direction and wind_speed are the most important five features among all.

Modeling

Since LightGBM is a gradient boosting tree-based framework, it has a faster training speed, higher efficiency, relatively low memory usage, better accuracy and most importantly, capable of dealing with large-scale data, I choose LightGBM for this problem. In this problem, a large number of categorical variables are present and categorical variables are easy to be handled with the tree-based learning algorithm. Hence, LightGBM is a fit for this task.

Here are the parameters:

Parameter	Value
tree	1000
num_leaves	400
objective	regression
learning_rate	0.005

metric	loss function
max_depth	10
min_data_in_leaf	20
bagging_fraction	0.9
feature_fraction	1
bagging_freq	1
bagging_seed	3

Parameter Description:

tree: The tree numbers in the model should not be too small or too big. If the value is too small, the model will not perform as well as we want. And if the value is too large, the model will be overfitting so it performs well on the training set but poorly on the test set.

num_leaves: an important parameter because it controls the complexity of the tree model. Technically, we usually set its value to 2^max_depth in order to avoid overfitting.

objective: control the assignment's type: regression or classification

learning rate: The model's learning speed, if the value is too small then the model will spend too much time in learning, if it is too large, the model will dive into a local minimum value.

metric: evaluation function, here I use l1 and l2

max_depth: the maximum depth of the tree.

min_data_in_leaf: the minimum samples on the trees' leaf

bagging_fraction: the data samples fraction for iteration

feature_fraction: the feature numbers fraction for iteration

bagging_freq: k=1 means perform bagging at every k=1 iteration

bagging_seed: random seed for bagging, the default value = 3

Parameter Tuning for Gradient Boosting

To obtain the optimal number of iterations, I implemented the graph (figure 9)to show the relationship between l2 regression and the number of iterations. The lower l2 is, the better the model is.

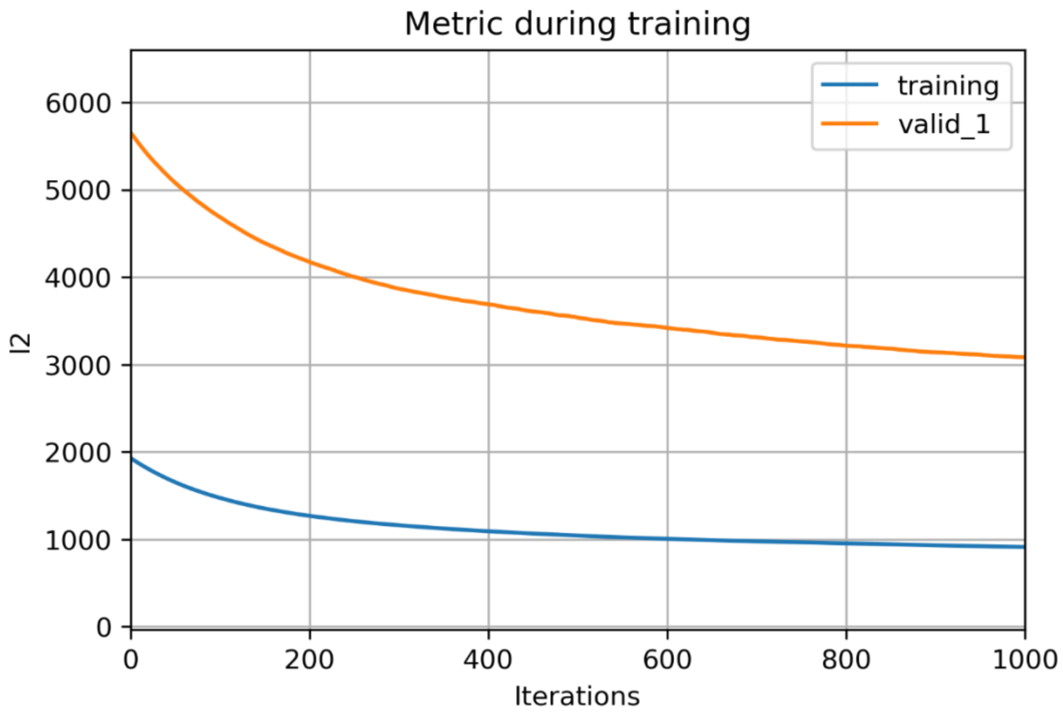


Figure.9 The Comparison Metric of Number of Iterations and l2 for Training Set and Validation Set

For iterations, we can see that the model starts to converge when the Iterations = 600 and the optimal iteration number can be achieved when it equals to 800.

Last but not least, I conducted a graph (figure 10 to 12) showing the relationship between the predicted value and the true value (labeled value) for O3, PM2.5 and PM10 respectively

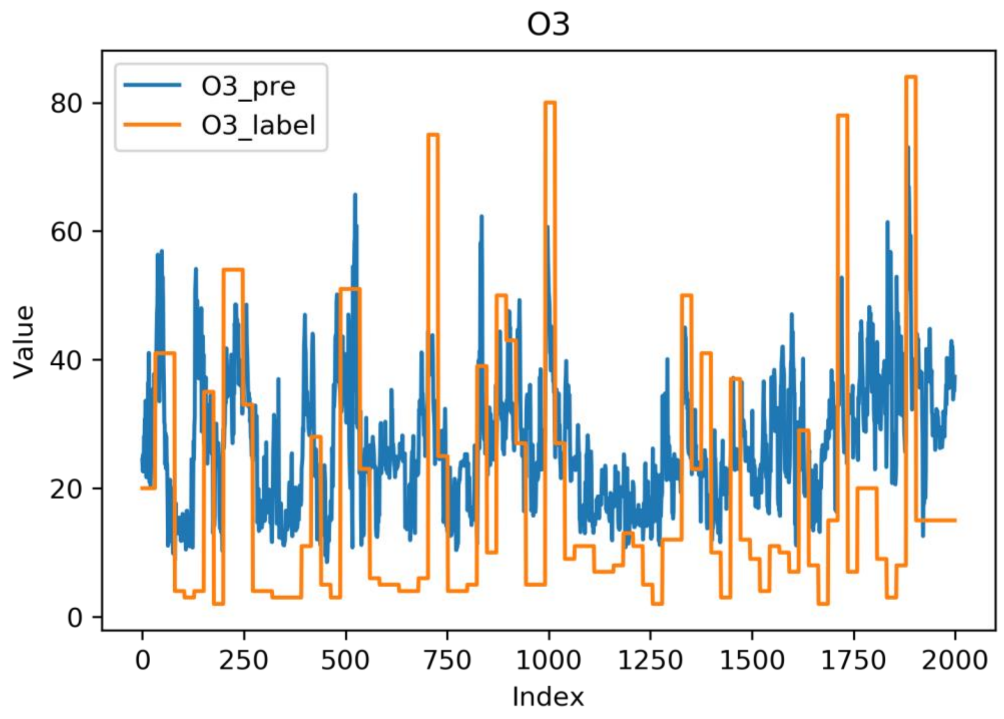


Figure.10 The Difference Between Predicted Value and the Actual Value for O3

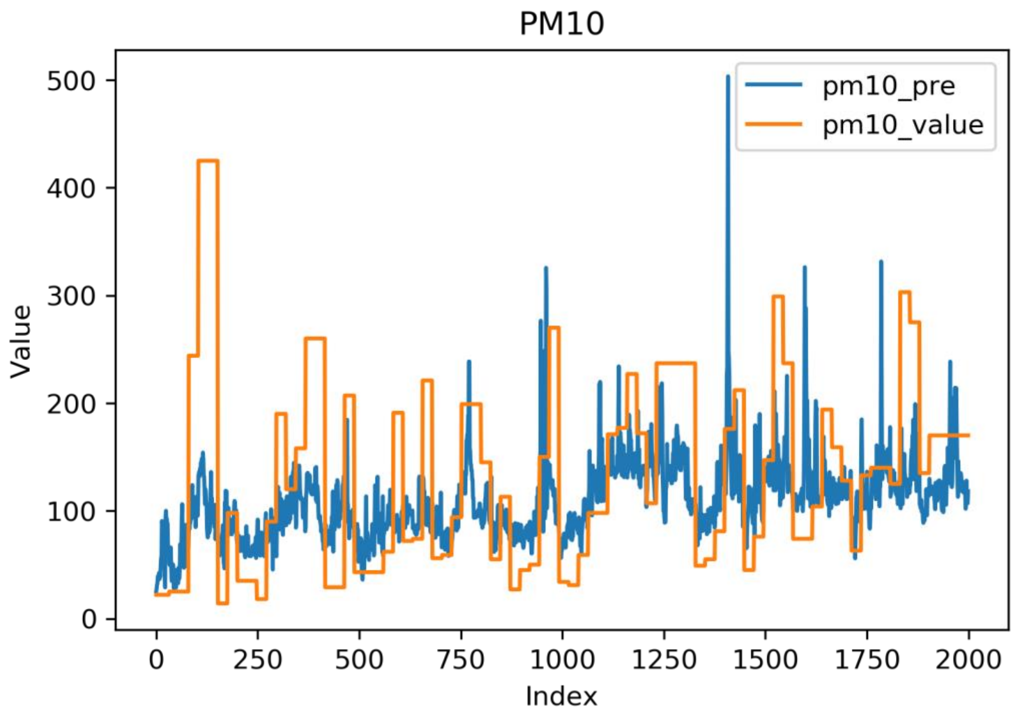


Figure.11 The Difference Between Predicted Value and the Actual Value for PM10

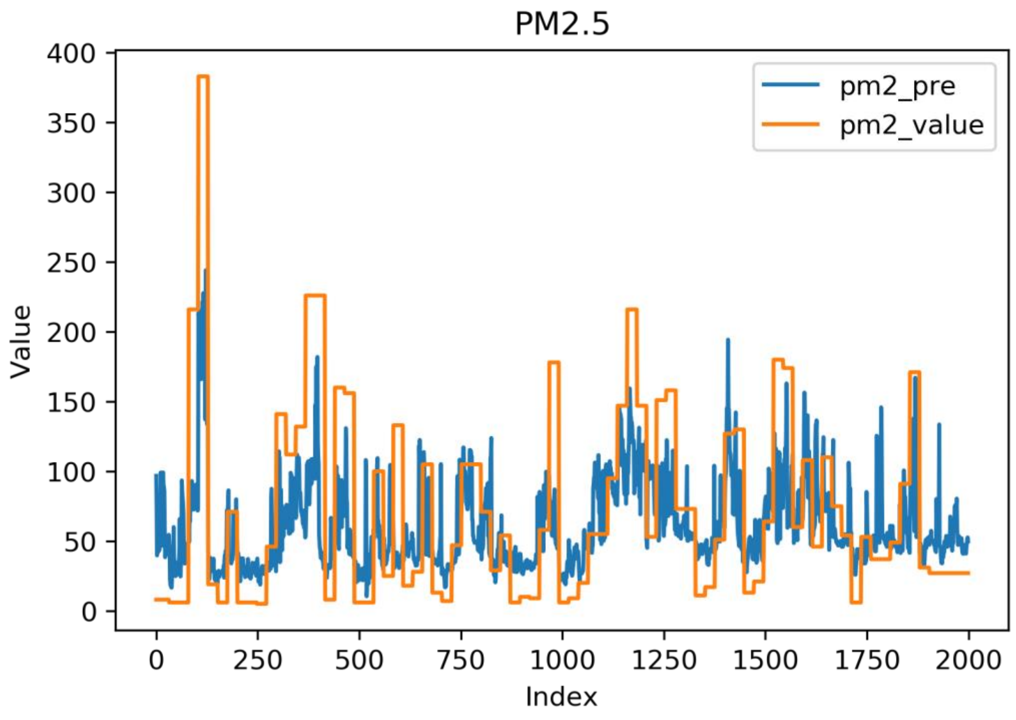
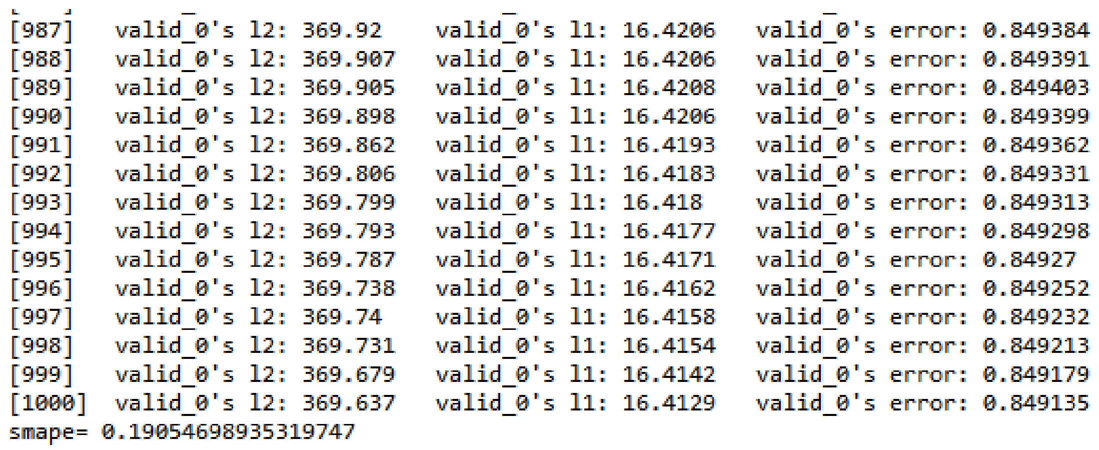


Figure.12 The Difference Between Predicted Value and the Actual Value for PM2.5

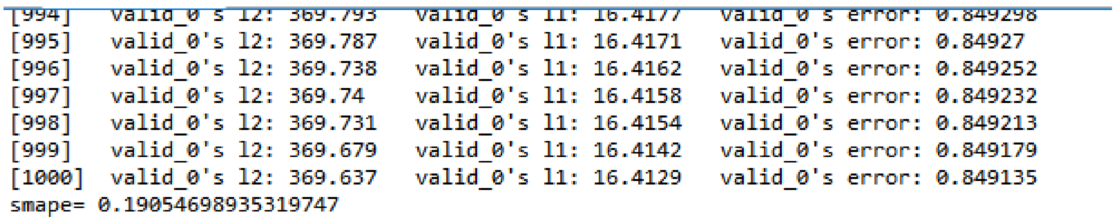
As can be seen from the graphs above, our prediction values are fairly close to the true values. Therefore, I believe that the predicted result is fairly reliable.

Evaluation Output

The best SMAPE (from evaluation.py) on test data was about 0.191.



```
[987]   valid_0's l2: 369.92    valid_0's l1: 16.4206   valid_0's error: 0.849384
[988]   valid_0's l2: 369.907   valid_0's l1: 16.4206   valid_0's error: 0.849391
[989]   valid_0's l2: 369.905   valid_0's l1: 16.4208   valid_0's error: 0.849403
[990]   valid_0's l2: 369.898   valid_0's l1: 16.4206   valid_0's error: 0.849399
[991]   valid_0's l2: 369.862   valid_0's l1: 16.4193   valid_0's error: 0.849362
[992]   valid_0's l2: 369.806   valid_0's l1: 16.4183   valid_0's error: 0.849331
[993]   valid_0's l2: 369.799   valid_0's l1: 16.418    valid_0's error: 0.849313
[994]   valid_0's l2: 369.793   valid_0's l1: 16.4177   valid_0's error: 0.849298
[995]   valid_0's l2: 369.787   valid_0's l1: 16.4171   valid_0's error: 0.84927
[996]   valid_0's l2: 369.738   valid_0's l1: 16.4162   valid_0's error: 0.849252
[997]   valid_0's l2: 369.74    valid_0's l1: 16.4158   valid_0's error: 0.849232
[998]   valid_0's l2: 369.731   valid_0's l1: 16.4154   valid_0's error: 0.849213
[999]   valid_0's l2: 369.679   valid_0's l1: 16.4142   valid_0's error: 0.849179
[1000]  valid_0's l2: 369.637   valid_0's l1: 16.4129   valid_0's error: 0.849135
smape= 0.19054698935319747
```

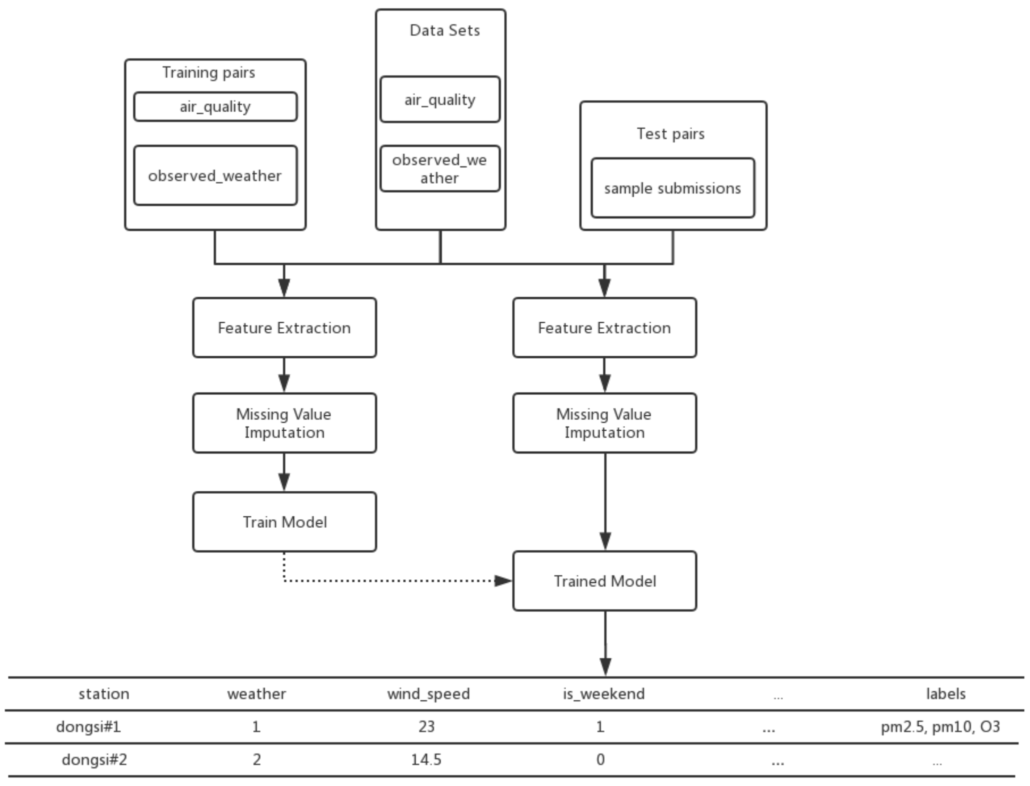


```
[994]   valid_0's l2: 369.793   valid_0's l1: 16.4177   valid_0's error: 0.849298
[995]   valid_0's l2: 369.787   valid_0's l1: 16.4171   valid_0's error: 0.84927
[996]   valid_0's l2: 369.738   valid_0's l1: 16.4162   valid_0's error: 0.849252
[997]   valid_0's l2: 369.74    valid_0's l1: 16.4158   valid_0's error: 0.849232
[998]   valid_0's l2: 369.731   valid_0's l1: 16.4154   valid_0's error: 0.849213
[999]   valid_0's l2: 369.679   valid_0's l1: 16.4142   valid_0's error: 0.849179
[1000]  valid_0's l2: 369.637   valid_0's l1: 16.4129   valid_0's error: 0.849135
smape= 0.19054698935319747
```

Conclusion

Overall, an appropriate handling of data-preprocessing and feature engineering process can contribute to a well-performed model with decent accuracy rate. In terms of the model, I did the parameter tuning by finding the optimal iteration and the results above are the best I have got.

Project's Framework



station	weather	wind_speed	is_weekend	...	labels
dongsi#1	1	23	1	...	pm2.5, pm10, O3
dongsi#2	2	14.5	0	...	...