# INFOSYS 722 Data Mining and Big Data

# Iteration 4 BDAS

# Analysis of Air-Quality in Shunyi

Yuchen Deng (yden881@Aucklanduni.ac.nz)
Department of Business, University of Auckland
Dr Ami Peiris

12 June 2020

GitHub:

https://github.com/yd90/Yuchen_Deng_Iteration4

## Contents

Yuchen Deng
237381181

# 1 Business Understanding

## 1.1 Identify the objectives of the business

### 1.1.1 Background information

Nunez (2019, February 4) stated "Air pollution has been linked to higher rates of cancer, heart disease, stroke, and respiratory diseases such as asthma". The air pollution has a negative impact on people' health. In other words, poor air-quality will cause high mortality rate. Li et al. (2019, April) said "We estimated a total of 169,862 additional deaths from short-term PM2.5 exposure throughout China in 2015". Air pollution becomes a crucial social issue in recent years, so reducing air pollution is very important in the future. However, there are many factors which make increase the amount of particulate matter. We need to find out main factors which have a great influence on air-quality. Therefore, my task is finding out the relationships between the air pollution and these factors.

### 1.1.2 Business Problem

Air pollution kills an estimated seven million people worldwide every year. WHO data shows that 9 out of 10 people breathe air containing high levels of pollutants (World Health Organization, n.d.). Air pollution becomes a major global issue to our health. The most serious form of air pollution is particulate matter, PM2.5 is one type of particulate matter, it means the particles with a diameter less than or equal 2.5μm. It is necessary for us to find some solutions in order to reduce the amount of PM2.5. Therefore, big data can help us analyse and build data model. According the data model, we can find out what factors may affect our air-quality. Then we will find some approaches to release the air pollutant.

### 1.1.3 Business goal

This Big Data Analytics Solution will address the Goal 12 of the UN.

Goal 12: Ensure sustainable consumption and production patterns

My approach to achieve this goal is using big data to predict the air-quality in the future. The PM2.5 is one of the major pollutants, so it is necessary to for government to reduce the amount of particle pollution (PM2.5/PM10). We can explore and build a model to analyse the relationship between atmospheric pollutants and particle pollution, because I assume that there are some factors which may affect the amount of particle pollution. Assumption and prediction both are important concept in big data analysis, they can help me have a basic understanding before using a professional software. In this project, I will use a dataset about air-quality in Shunyi to analyse whether exists a relationship between PM2.5 and gas emission. In addition, I can also build a data model to predict the air-quality in the following years.

### 1.1.4 Business objectives

Thus, a study is commissioned with the following objectives:

i. Finding out the relationships between PM 2.5 and atmospheric pollutants, like sulphur dioxide, nitrogen dioxide, etc.
ii. Finding out the relationship between PM2.5 and temperature.
iii. Predicting the air-quality in the future.

### 1.1.5 Standard of data mining success

Tentatively, the study will be judged a success if:

i. Finding out details about how these atmospheric pollutants the air-quality.
ii. There are clear and reliable evidence to prove that there is a relationship between PM 2.5 and temperature.
iii. The accuracy of prediction is high (may more than 80%)

## 1.2 Access the Situation

### 1.2.1 Resource Inventory

**Hardware:** Personal laptop

**Software:** Spyder (Python 2.7)

**Date Source**: There are several websites which I can use to collect data. Also, they are free to download.

- Kaggle: https://www.kaggle.com/
- KDnuggets: https://www.kdnuggets.com/
- Knoema: https://www.kdnuggets.com/
- NYC OpenData: https://opendata.cityofnewyork.us/
- Data.Govt.nz: https://data.govt.nz/

**Personnel:** There is no Personnel needed to complete the project, I'm the only one to do this project.

### 1.2.2 Requirements, Assumptions, and Constraints

**Requirement:** There is no security and legal restrictions on the data or project results. Also, there is no requirement on results deployment.

**Assumption:** There is no economic factor which may affect data analysis. However, there is an assumption about data quality. In my dataset, there are some small problems which has only a little influence to my data analysis. This dataset may not include every feature which I want to use in my project, but I can still assume these records is reliable and effective.

**Constraints:** I have verified all the legal constraints on data usage. There is no limitation of data usage, because this dataset was download from the Kaggle which is a free website for data scientists. In addition, there is no financial constraints in the project budget, it is unnecessary to spend money on this project. However, I may purchase a dataset, if I cannot find an appropriate dataset for my data analysis.

### 1.2.3 Risks and Contingencies

**Risk 1:** The quality of dataset could be poor. For example, I may find an incomplete dataset which misses too many values to analyse. In addition, the coverage of data could also be poor. For example, I can only find a dataset for one mouth or one year. These conditions will influence the accuracy of my project.

**Risk 2:** We may download a fake or incorrect dataset from unauthoritative website, like personal blog.

**Risk 3:** The dataset may include several irrelevant features/attributes, which will increase the difficulty of data analysis.

**Contingency plan 1:** I can purchase a dataset of high quality and coverage, which is an effective approach to quickly find reliable dataset.

**Contingency plan 2:** I need make sure the source of the dataset is reliable and authoritative. Therefore, I should avoid to download dataset from personal blog.

**Contingency plan 3:** Cleaning the data is an effective approach to decrease the difficulty of data analysis. For example, I can remove irrelevant features/attributes before using Spyder.

## 1.3 Data Mining Objectives

### 1.3.1 Data Mining Goals

**Prediction**: This project will use the dataset of air-quality in Shunyi to build a data model in order to predict the air-quality in the future.

**Relationship**: This project will use the dataset of air-quality in Shunyi to find relationships between PM 2.5 and other attributes.

### 1.3.2 Data Mining Success Criteria

**Prediction**: If the accuracy of the prediction is more than 80%, then this study would be judged as success.

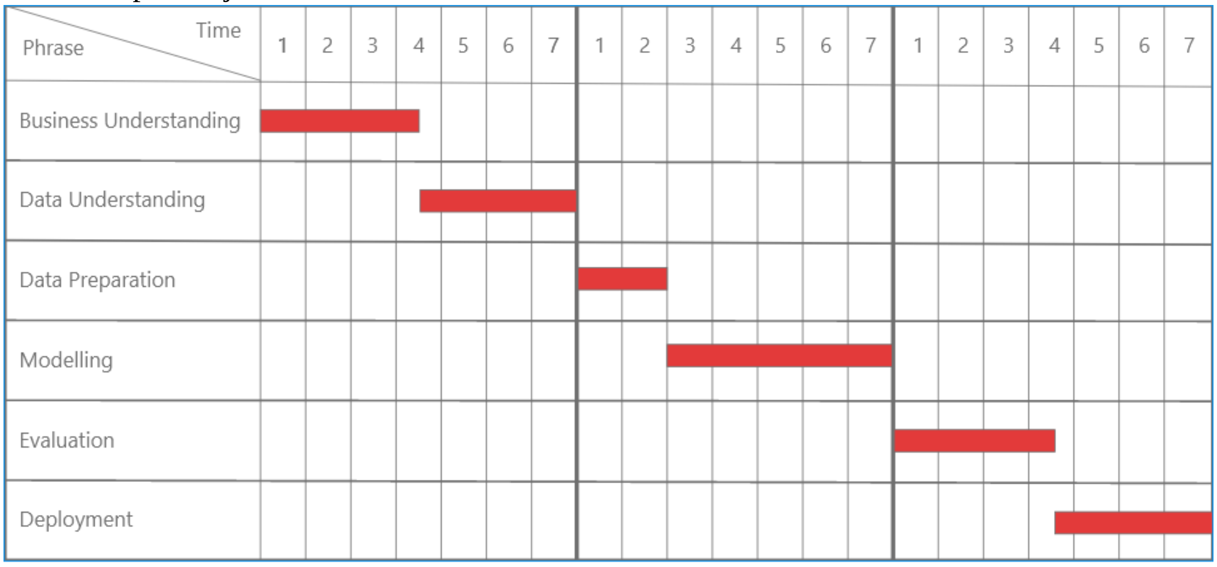
**Relationship**: If I successfully find the relationship between PM 2.5 and other attributes, I would say the study is successful.

## 1.4 Produce a Project Plan

### 1.4.1 Writing the Project Plan

I have to finish this project in three weeks, and there are six phrases. Therefore, I need properly manage my time. The project plan will be displayed in next step.

### 1.4.2 Sample Project Plan



| Phrases | Resources | Risks |
|---|---|---|
| Business Understanding | All analysts | Lack data |
| Data Understanding | All analysts | Data problems, technology problems |
| Data Preparation | Spyder (Python2.7), Data mining consultant, some database analyst time | Data problems, technology problems |
| Modelling | All analysts | Technology problems, inability to find adequate |
| Evaluation | Spyder (Python2.7), Data mining consultant, some database analyst time | Lack data, Inability to implement results |
| Deployment | Data mining consultant, some database analyst time | Lack data, Inability to implement results |

# 2 Data Understanding

## 2.1 Collect Initial Data

The dataset was downloaded from Kaggle which is a subsidiary of Google LLC. The Kaggle authoritative and free website for data scientist. This website allows people to publish and download datasets in order to build and explore models for data mining. Therefore, I believe that all the datasets which are downloaded from the Kaggle is true and reliable. Additionally, the existing data are enough to meet my needs, so it is not necessary for me to purchase or search additional data.

The dataset includes 35,064 records and 18 attributes. In my opinion, I think it is enough for me to make an accurate and reliable prediction. And, I do not need to merge different data source, because all the records are recorded in one Excel file. Before using Spyder, I find some attributes can be removed, which include record number, year, day and station. Without them, the result of prediction may not be influenced.

**Download Link:** https://www.kaggle.com/sid321axn/beijing-multisite-airquality-data-set

**Publisher:** Manu Siddhartha

## 2.2 Describe Data

This dataset has 35,064 rows and 18 attributes.

| Name | Value Type | Description | Unit |
|---|---|---|---|
| No | Continuous | | |
| Year | Continuous | | |
| Month | Continuous | | |
| Day | Continuous | | |
| Hour | Continuous | | |
| PM 2.5 | Continuous | Target | ug/m^3 |
| PM 10 | Continuous | | ug/m^3 |
| SO2 | Continuous | | ug/m^3 |
| NO2 | Continuous | | ug/m^3 |
| CO | Continuous | | ug/m^3 |
| O3 | Continuous | | ug/m^3 |
| TEMP | Continuous | Temperature | degree Celsius |
| PRESS | Continuous | Pressure | hPa |
| DEWP | Continuous | Dew Point Temperature | degree Celsius |
| RAIN | Flag | | mm |
| WD | Categorical | Wind Direction | |
| WSPM | Continuous | Wind Speed | m/s |
| Station | Categorical | Name of the air-quality monitoring site | |

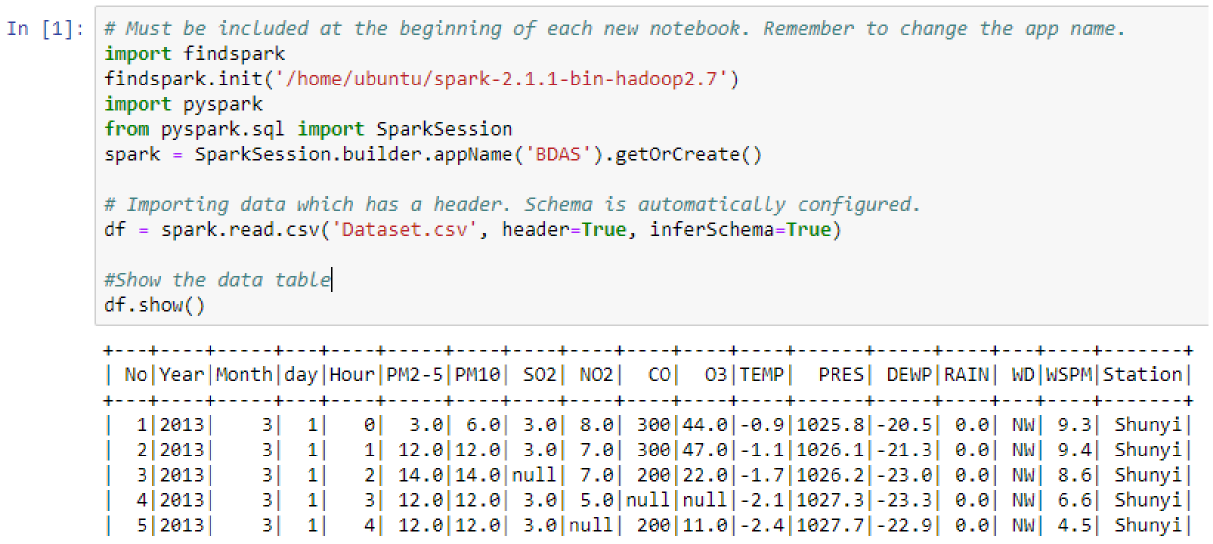
**Table 2: The description of dataset**

There is no specific coding scheme.

## 2.3 Explore the Data

At this stage, I will use Jupyter Notebook (Python 3) to analyse this dataset. First of all, I imported the file and display the table of my dataset using spark.read.csv() function. In this step, I had a general understanding of this dataset. Figure 1 shows the first five rows and the last five rows. According to Figure 1, I observe that there are 18 attributes and 35,064 rows. In addition, I also found there are some nan value, it may influence the accuracy of prediction. I will discuss this problem in section 3 (Data Preparation). The figure 2 shows the data types of these attributes. The majority data type of these attributes is numeric, but the WD and Station are object. I do not need to change or modify any data type in the following steps, because all the attributes have a proper data type.

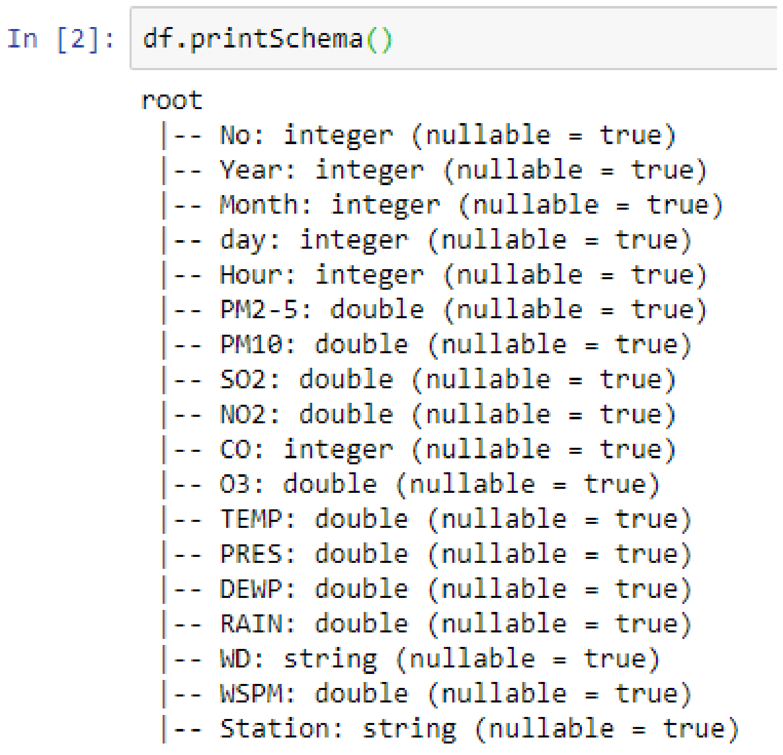
**Important: In this table, the attribute 'PM2-5' means 'PM2.5'. When I try use PM2.5 as my attribute name, I will receive a reception that about name error on this attribute. Then, I google this error, and I find '.' (Point) will cause some error. Therefore, I use 'PM2-5' to represent 'PM2.5'.**



```
In [1]: # Must be included at the beginning of each new notebook. Remember to change the app name.
        import findspark
        findspark.init('/home/ubuntu/spark-2.1.1-bin-hadoop2.7')
        import pyspark
        from pyspark.sql import SparkSession
        spark = SparkSession.builder.appName('BDAS').getOrCreate()

        # Importing data which has a header. Schema is automatically configured.
        df = spark.read.csv('Dataset.csv', header=True, inferSchema=True)

        #Show the data table
        df.show()
```

```
+---+----+-----+---+----+-----+----+----+----+----+----+----+------+-----+----+---+----+-------+
| No|Year|Month|day|Hour|PM2-5|PM10| SO2| NO2|  CO|  O3|TEMP|  PRES| DEWP|RAIN| WD|WSPM|Station|
+---+----+-----+---+----+-----+----+----+----+----+----+----+------+-----+----+---+----+-------+
|  1|2013|    3|  1|   0|  3.0| 6.0| 3.0| 8.0| 300|44.0|-0.9|1025.8|-20.5| 0.0| NW| 9.3| Shunyi|
|  2|2013|    3|  1|   1| 12.0|12.0| 3.0| 7.0| 300|47.0|-1.1|1026.1|-21.3| 0.0| NW| 9.4| Shunyi|
|  3|2013|    3|  1|   2| 14.0|14.0|null| 7.0| 200|22.0|-1.7|1026.2|-23.0| 0.0| NW| 8.6| Shunyi|
|  4|2013|    3|  1|   3| 12.0|12.0| 3.0| 5.0|null|null|-2.1|1027.3|-23.3| 0.0| NW| 6.6| Shunyi|
|  5|2013|    3|  1|   4| 12.0|12.0| 3.0|null| 200|11.0|-2.4|1027.7|-22.9| 0.0| NW| 4.5| Shunyi|
```

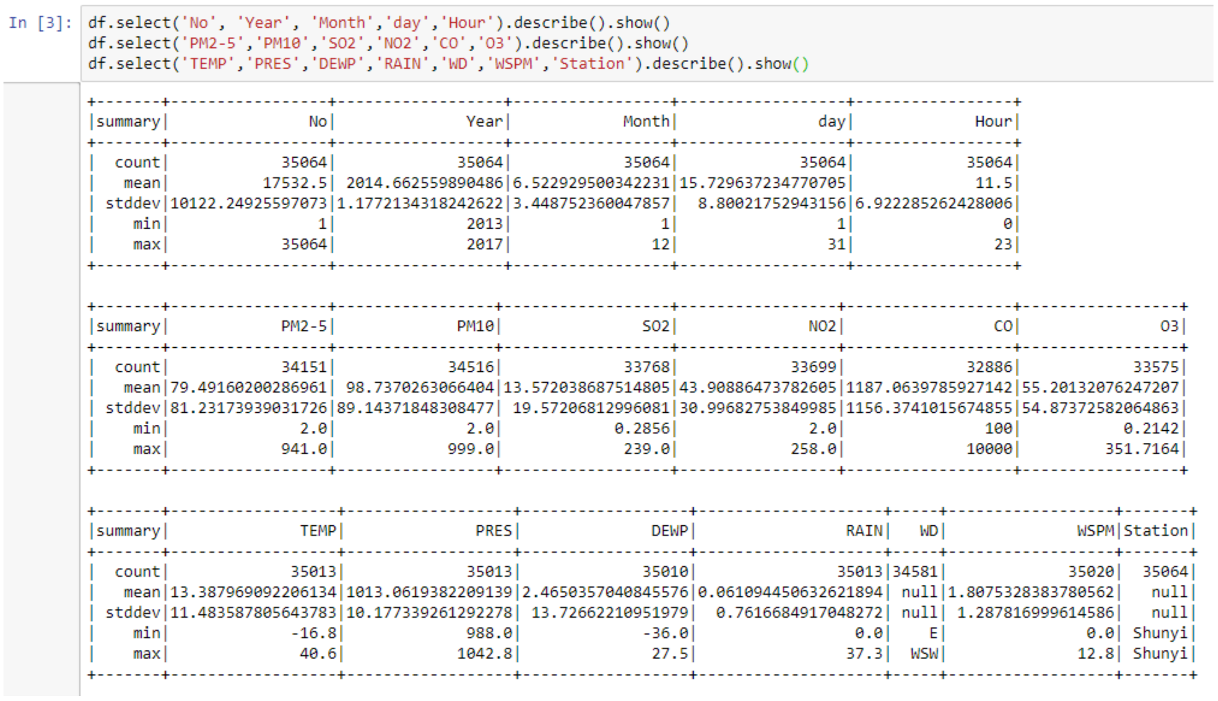
Figure 1: Table of dataset



```
In [2]: df.printSchema()

root
 |-- No: integer (nullable = true)
 |-- Year: integer (nullable = true)
 |-- Month: integer (nullable = true)
 |-- day: integer (nullable = true)
 |-- Hour: integer (nullable = true)
 |-- PM2-5: double (nullable = true)
 |-- PM10: double (nullable = true)
 |-- SO2: double (nullable = true)
 |-- NO2: double (nullable = true)
 |-- CO: integer (nullable = true)
 |-- O3: double (nullable = true)
 |-- TEMP: double (nullable = true)
 |-- PRES: double (nullable = true)
 |-- DEWP: double (nullable = true)
 |-- RAIN: double (nullable = true)
 |-- WD: string (nullable = true)
 |-- WSPM: double (nullable = true)
 |-- Station: string (nullable = true)
```

Figure 2: Data Type for Attributes

Secondly, I checked the statistics of numeric attributes, it will make sure the value of numeric attributes is in a reasonable and proper range. For example, the value of PM2.5 should greater than or equal zero. If a value of PM2.5 is negative, then this value is incorrect and it should be removed from my dataset. I use describe() function to show the statistics of numeric attributes, this function is used to view some basic statistical details like min, max, mean, etc. Figure 3 show eight statistical details. According the Figure 3, I observed that every attribute has a proper range. Therefore, I do not need to change or modify any row.



```
df.select('No', 'Year', 'Month','day','Hour').describe().show()
df.select('PM2-5','PM10','SO2','NO2','CO','O3').describe().show()
df.select('TEMP','PRES','DEWP','RAIN','WD','WSPM','Station').describe().show()
```

```
+-------+-----------------+------------------+-----------------+------------------+-----------------+
|summary|               No|              Year|            Month|               day|             Hour|
+-------+-----------------+------------------+-----------------+------------------+-----------------+
|  count|            35064|             35064|            35064|             35064|            35064|
|   mean|          17532.5| 2014.662559890486|6.522929500342231|15.729637234770705|             11.5|
| stddev|10122.24925597073|1.1772134318242622|3.448752360047857|  8.80021752943156|6.922285262428006|
|    min|                1|              2013|                1|                 1|                0|
|    max|            35064|              2017|               12|                31|               23|
+-------+-----------------+------------------+-----------------+------------------+-----------------+

+-------+-----------------+-----------------+------------------+-----------------+------------------+-----------------+
|summary|            PM2-5|             PM10|               SO2|              NO2|                CO|               O3|
+-------+-----------------+-----------------+------------------+-----------------+------------------+-----------------+
|  count|            34151|            34516|             33768|            33699|             32886|            33575|
|   mean|79.49160200286961| 98.7370263066404|13.572038687514805|43.90886473782605|1187.0639785927142|55.20132076247207|
| stddev|81.23173939031726|89.14371848308477| 19.57206812996081|30.99682753849985|1156.3741015674855|54.87372582064863|
|    min|              2.0|              2.0|            0.2856|              2.0|               100|           0.2142|
|    max|            941.0|            999.0|             239.0|            258.0|             10000|         351.7164|
+-------+-----------------+-----------------+------------------+-----------------+------------------+-----------------+

+-------+------------------+------------------+------------------+--------------------+-----+------------------+-------+
|summary|              TEMP|              PRES|              DEWP|                RAIN|   WD|              WSPM|Station|
+-------+------------------+------------------+------------------+--------------------+-----+------------------+-------+
|  count|             35013|             35013|             35010|               35013|34581|             35020|  35064|
|   mean|13.387969092206134|1013.0619382209139|2.4650357040845576|0.061094450632621894| null|1.8075328383780562|   null|
| stddev|11.483587805643783|10.177339261292278| 13.72662210951979|  0.7616684917048272| null| 1.287816999614586|   null|
|    min|             -16.8|             988.0|             -36.0|                 0.0|    E|               0.0| Shunyi|
|    max|              40.6|            1042.8|              27.5|                37.3|  WSW|              12.8| Shunyi|
+-------+------------------+------------------+------------------+--------------------+-----+------------------+-------+
```

Figure 3: Statistics of Numeric Attributes

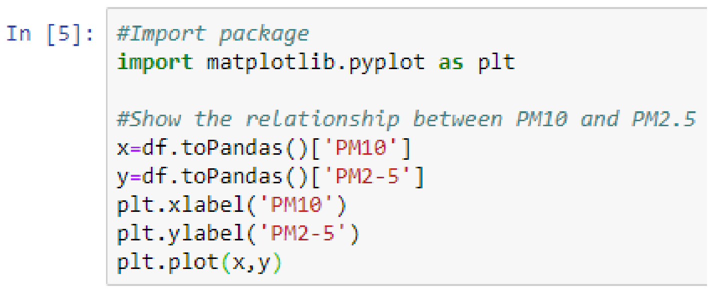
In the previous section, we assume that there are some relationships between PM2.5 and other attributes. The dataset has over 30k rows, so it is hard to find any trend or pattern without graph. Therefore, visualising the raw data is very important for data analysis. I use Spyder to generate some graphs, they can help me having a basic understanding of my dataset. According to these graphs, I can understand which attribute will influence the value of PM2.5. Then, I can use data analysis model to confirm my assumption. I divided these attributes in three groups, I will discuss each group in the following step.

Group One:

Target: PM2.5

Factor: PM10

Figure 4 show the relationship between PM2.5 and PM10, it may indicate that the value of PM2.5 will increase when the value of PM10 increase. However, the relationship is not very clear, which means I need do more analyse in order to receive a clear result.



```
In [5]: #Import package
        import matplotlib.pyplot as plt

        #Show the relationship between PM10 and PM2.5
        x=df.toPandas()['PM10']
        y=df.toPandas()['PM2-5']
        plt.xlabel('PM10')
        plt.ylabel('PM2-5')
        plt.plot(x,y)
```

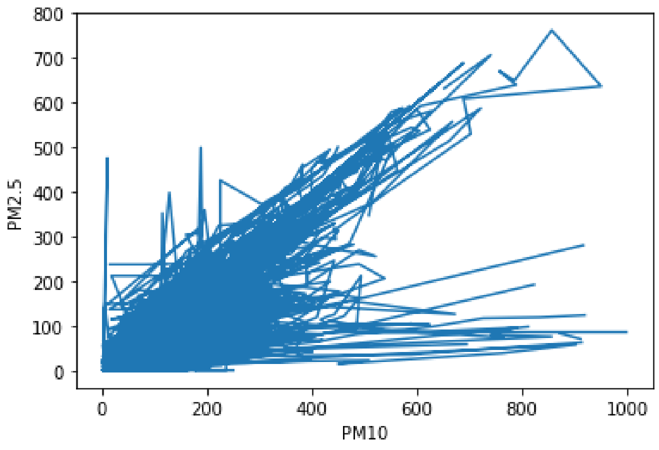


Figure 4: Relationship between PM2.5 and PM10

Group Two:

Target: PM2.5

Factors: SO2, NO2, CO, O3

Figure 5, Figure 6, Figure 7 and Figure 8 show the relationships between PM2.5 and gas emission, but the trends are unclear. Therefore, I cannot make a conclusion. I will use python to clarify the relationships in the following section.

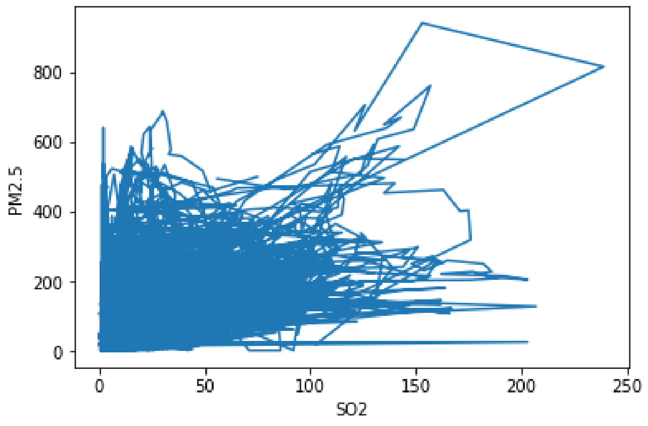


Figure 5: Relationship between PM2.5 and SO2

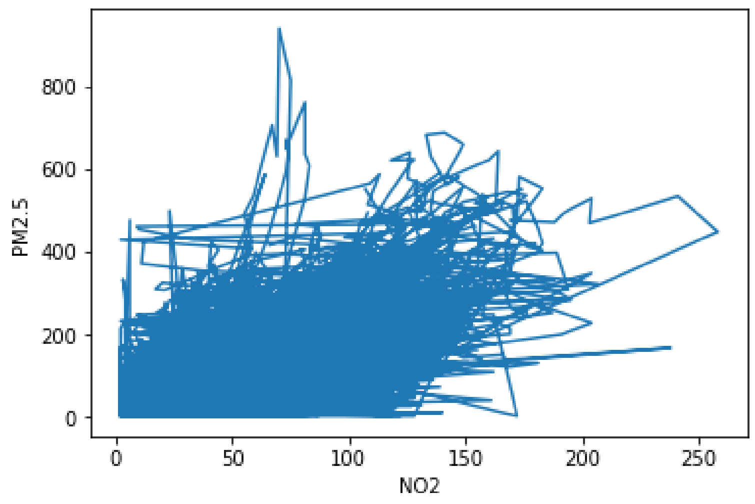


Figure 6: Relationship between PM2.5 and NO2

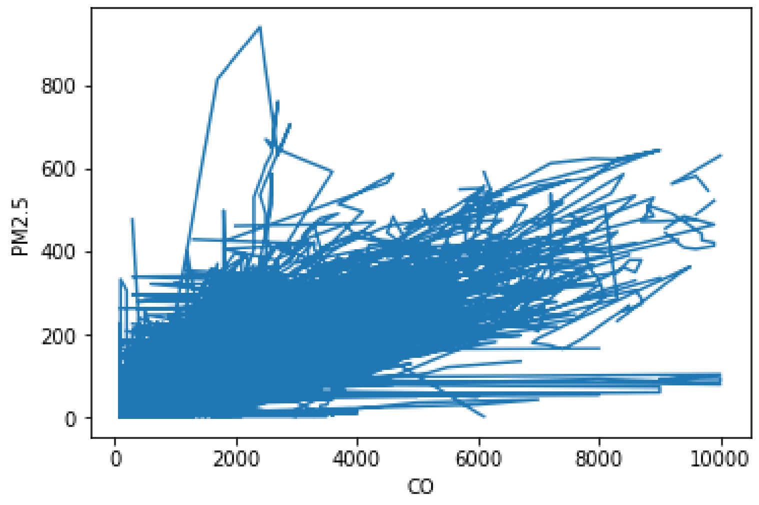


Figure 7: Relationship between PM2.5 and CO

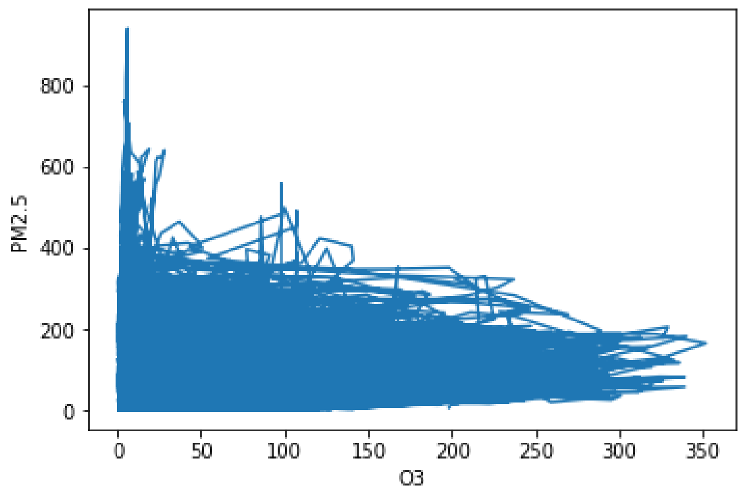


Figure 8: Relationship between PM2.5 and O3

Group Three:

Target: PM2.5

Factors: TEMP, WSPM

Figure 9 and Figure 10 show that the relationship between PM2.5 and TEMP, WSPM. According to figure 9, I cannot confirm there is a clear relationship between PM2.5 and TEMP, because the graph is messy and unclear. According to Figure 10, I make an assumption which is that the value of PM2.5 is low when the value of WSPM has a high value. However, I need to do more processes to receive reliable and accurate results.

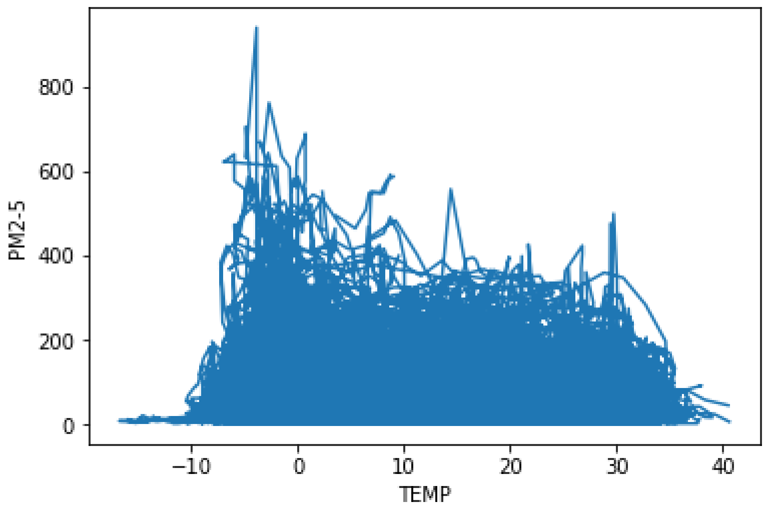


Figure 9: Relationship between PM2.5 and TEMP

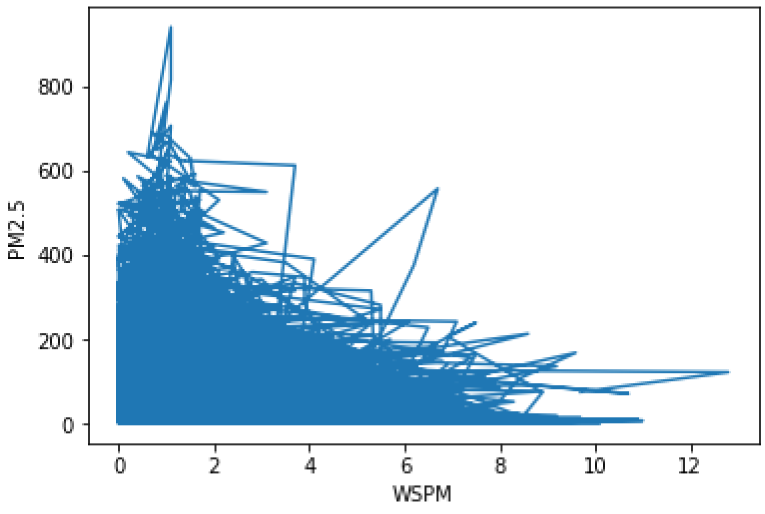


Figure 10: Relationship between PM2.5 and WSPM

## 2.4 Verify the Data Quality

In this step, I will verify the data quality. This dataset has more than 30k rows and 18 attributes, which means it contain a large number of records. I need to confirm that every attribute has reliable and appropriate value. In addition, I also need to deal with missing value before starting data mining, the missing value may have a negative impact to our prediction. Therefore, I will explore the table and verify the data quality.

Figure 11 shows the number of null values for all attributes. There are 12 attributes have null value, I will remove rows with null value in next section. Removing null value can improve the accuracy of prediction.

```
#Count the null value for each column
from pyspark.sql.functions import isnan, when, count, col
df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns]).show()
```

```
+---+----+-----+---+----+-----+----+----+----+----+----+----+----+----+----+---+----+-------+
| No|Year|Month|day|Hour|PM2-5|PM10| SO2| NO2|  CO|  O3|TEMP|PRES|DEWP|RAIN| WD|WSPM|Station|
+---+----+-----+---+----+-----+----+----+----+----+----+----+----+----+----+---+----+-------+
|  0|   0|    0|  0|   0|  913| 548|1296|1365|2178|1489|  51|  51|  54|  51|483|  44|      0|
+---+----+-----+---+----+-----+----+----+----+----+----+----+----+----+----+---+----+-------+
```

Figure 11: Missing value

Figure 12 shows the boxplot about four attributes, they are year, month, day and hour. According to these graphs, I can confirm that these attributes have no outlier and extremes. Therefore, I do not need to make any change for these attributes in next section.

```
In [15]: df['Year'].plot.box()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e9c480278>
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e9a26b048>
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e9a21f3c8>
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e9a2b5550>
```

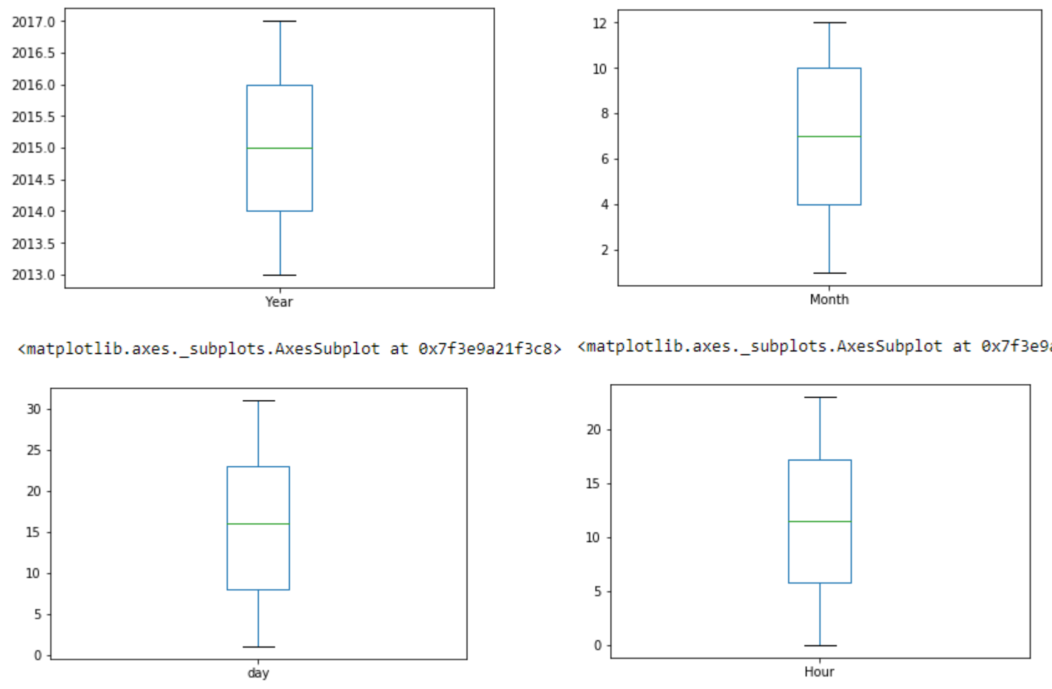
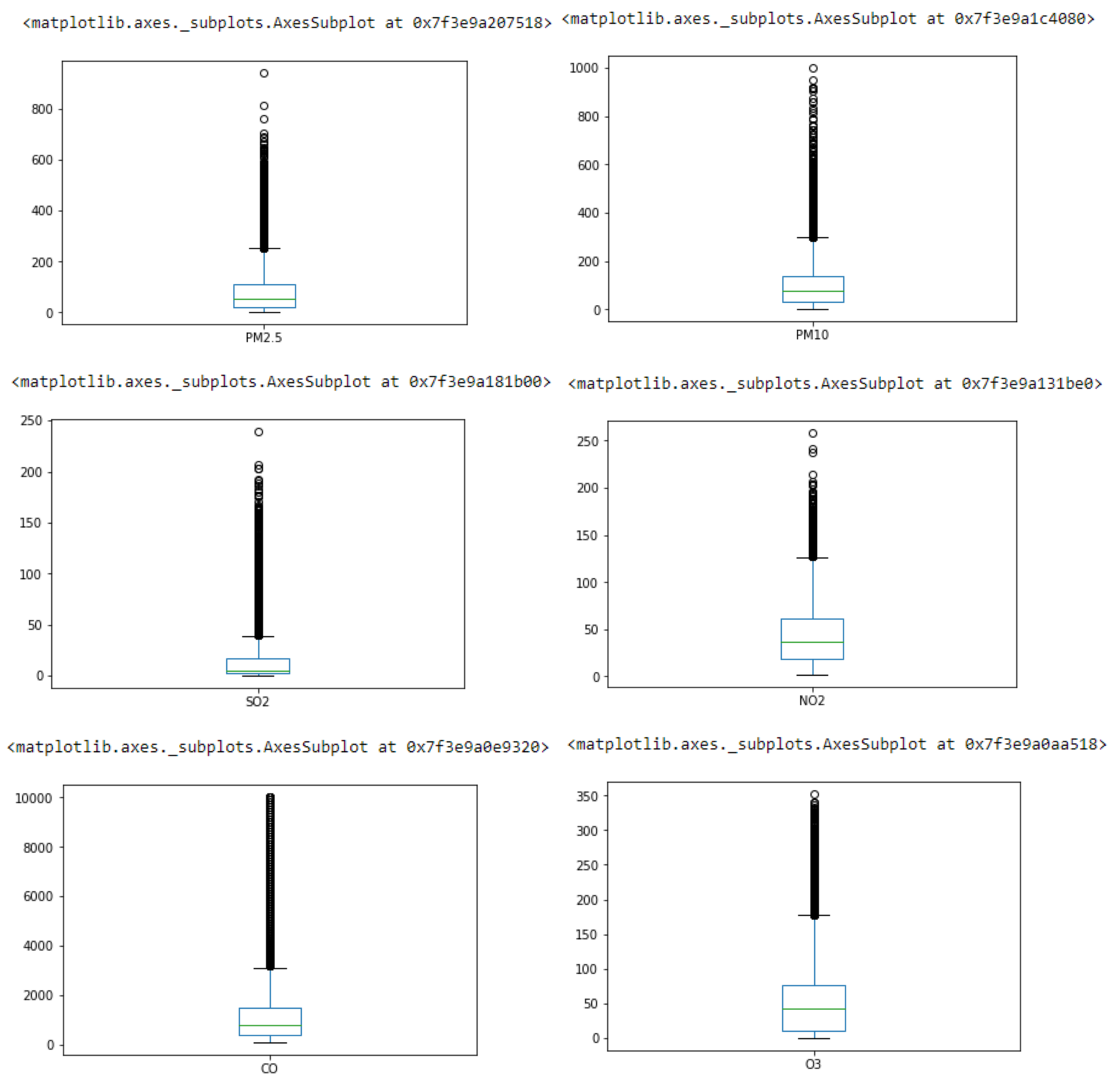


Figure 12: Boxplot (“Year”, “Month”, “Day”, “Hour”)

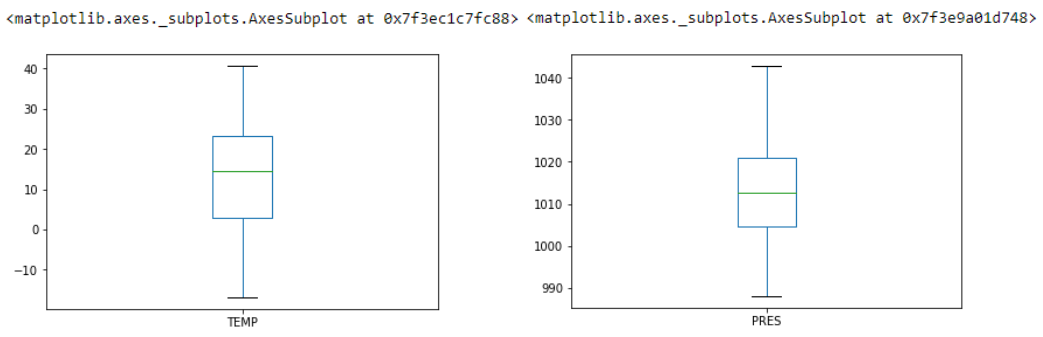
Figure 13 shows the boxplot about six attributes, they are PM2.5, PM10, SO2, NO2, CO and O3. I observe that there are some outliers and extremes in all graphs. I will discuss whether remove these outliers and extremes in next section.



```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e9a207518> <matplotlib.axes._subplots.AxesSubplot at 0x7f3e9a1c4080>
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e9a181b00> <matplotlib.axes._subplots.AxesSubplot at 0x7f3e9a131be0>
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e9a0e9320> <matplotlib.axes._subplots.AxesSubplot at 0x7f3e9a0aa518>
```

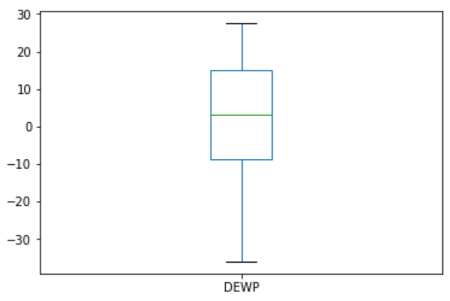
Figure 13: Boxplot (“PM2.5”, “PM10”, “SO2”, “NO2”, “CO”, “O3”)

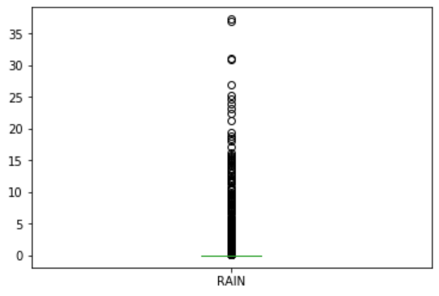
Figure 14 shows the boxplot about six attributes, they are TEMP, PRES, DEWP, RAIN and WSPM. For the graphs of TEMP, PRES and DEWP, they do not have any outliers and extremes, so we do not need to modify any row for these three attributes. For the graphs of RAIN and WSPM, there are some outliers and extremes. I will discuss whether remove these outliers and extremes in next section.



```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3ec1c7fc88> <matplotlib.axes._subplots.AxesSubplot at 0x7f3e9a01d748>
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e9a04ff60> <matplotlib.axes._subplots.AxesSubplot at 0x7f3e99c56da0>
```





```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e99c13f60>
```

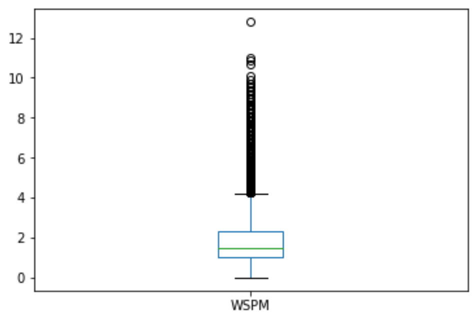
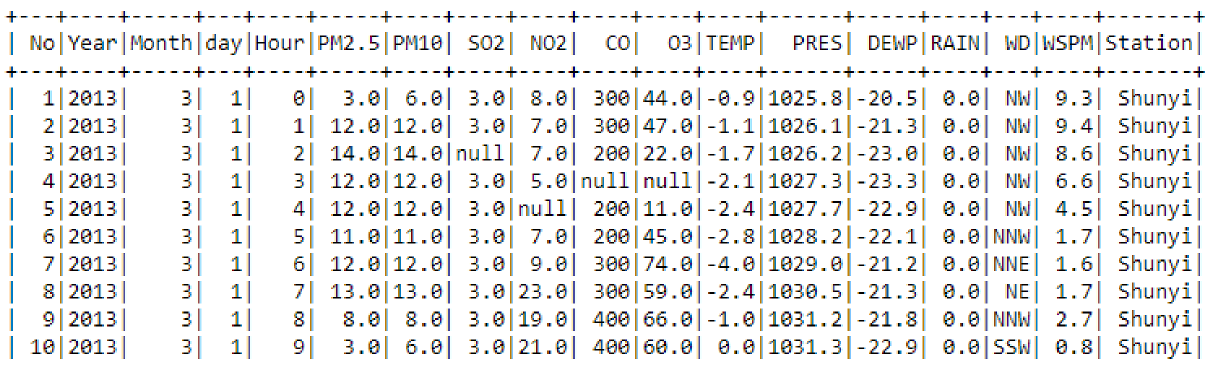


Figure 14: Boxplot (“TEMP”, “PRES”, “DEWP”, “RAIN”, “WSPM”)

# 3 Data Preparation

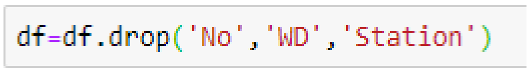
## 3.1 Select the Data

The initial dataset contains 35,064 rows and 18 attributes. This dataset contains a large number of rows, which will slow the process of data mining. We need select proper data size in order to increase the speed of data analysis. Therefore, selecting suitable data is very important, which can reduce time on data analysis. First of all, I use the spark.read.csv() function to import my dataset in the Spyder, and I use the print function to display the whole table. The Figure 15 displays the initial table which include all fields. Then, I will select proper attributes according to the initial table. There are 18 attributes in initial table, I will remove some attributes which are irrelative to my prediction. First of all, I will remove the No (row number). The row number is unique for each row, it is label to distinct different rows. Therefore, the row number can be removed from my dataset. Removing No will not influence the accuracy of my prediction. In addition, the station also can be removed, because all the values were measured in same station which is Shunyi. The WD also can be removed, because it describes a series of textual data. These three attributes are irrelative to my prediction, so I can remove them at this step. In the Spyder, I used the drop function to remove irrelative attributes. Figure 17 show the new table which includes 15 attributes.



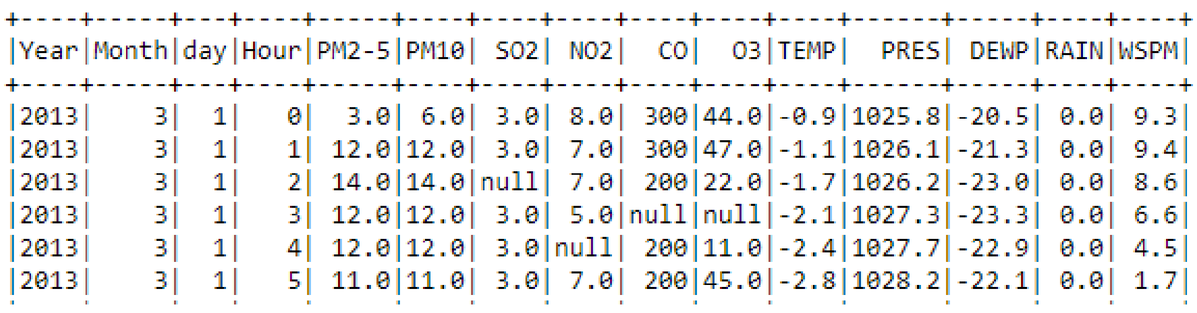
```
+---+----+-----+---+----+-----+----+----+----+----+----+----+------+-----+----+---+----+-------+
| No|Year|Month|day|Hour|PM2.5|PM10| SO2| NO2|  CO|  O3|TEMP|  PRES| DEWP|RAIN| WD|WSPM|Station|
+---+----+-----+---+----+-----+----+----+----+----+----+----+------+-----+----+---+----+-------+
|  1|2013|    3|  1|   0|  3.0| 6.0| 3.0| 8.0| 300|44.0|-0.9|1025.8|-20.5| 0.0| NW| 9.3| Shunyi|
|  2|2013|    3|  1|   1| 12.0|12.0| 3.0| 7.0| 300|47.0|-1.1|1026.1|-21.3| 0.0| NW| 9.4| Shunyi|
|  3|2013|    3|  1|   2| 14.0|14.0|null| 7.0| 200|22.0|-1.7|1026.2|-23.0| 0.0| NW| 8.6| Shunyi|
|  4|2013|    3|  1|   3| 12.0|12.0| 3.0| 5.0|null|null|-2.1|1027.3|-23.3| 0.0| NW| 6.6| Shunyi|
|  5|2013|    3|  1|   4| 12.0|12.0| 3.0|null| 200|11.0|-2.4|1027.7|-22.9| 0.0| NW| 4.5| Shunyi|
|  6|2013|    3|  1|   5| 11.0|11.0| 3.0| 7.0| 200|45.0|-2.8|1028.2|-22.1| 0.0|NNW| 1.7| Shunyi|
|  7|2013|    3|  1|   6| 12.0|12.0| 3.0| 9.0| 300|74.0|-4.0|1029.0|-21.2| 0.0|NNE| 1.6| Shunyi|
|  8|2013|    3|  1|   7| 13.0|13.0| 3.0|23.0| 300|59.0|-2.4|1030.5|-21.3| 0.0| NE| 1.7| Shunyi|
|  9|2013|    3|  1|   8|  8.0| 8.0| 3.0|19.0| 400|66.0|-1.0|1031.2|-21.8| 0.0|NNW| 2.7| Shunyi|
| 10|2013|    3|  1|   9|  3.0| 6.0| 3.0|21.0| 400|60.0| 0.0|1031.3|-22.9| 0.0|SSW| 0.8| Shunyi|
```

Figure 15: Initial Table



```
df=df.drop('No','WD','Station')
```

Figure 16: The Process of Drop Function



```
+----+-----+---+----+-----+----+----+----+----+----+----+------+-----+----+----+
|Year|Month|day|Hour|PM2-5|PM10| SO2| NO2|  CO|  O3|TEMP|  PRES| DEWP|RAIN|WSPM|
+----+-----+---+----+-----+----+----+----+----+----+----+------+-----+----+----+
|2013|    3|  1|   0|  3.0| 6.0| 3.0| 8.0| 300|44.0|-0.9|1025.8|-20.5| 0.0| 9.3|
|2013|    3|  1|   1| 12.0|12.0| 3.0| 7.0| 300|47.0|-1.1|1026.1|-21.3| 0.0| 9.4|
|2013|    3|  1|   2| 14.0|14.0|null| 7.0| 200|22.0|-1.7|1026.2|-23.0| 0.0| 8.6|
|2013|    3|  1|   3| 12.0|12.0| 3.0| 5.0|null|null|-2.1|1027.3|-23.3| 0.0| 6.6|
|2013|    3|  1|   4| 12.0|12.0| 3.0|null| 200|11.0|-2.4|1027.7|-22.9| 0.0| 4.5|
|2013|    3|  1|   5| 11.0|11.0| 3.0| 7.0| 200|45.0|-2.8|1028.2|-22.1| 0.0| 1.7|
```

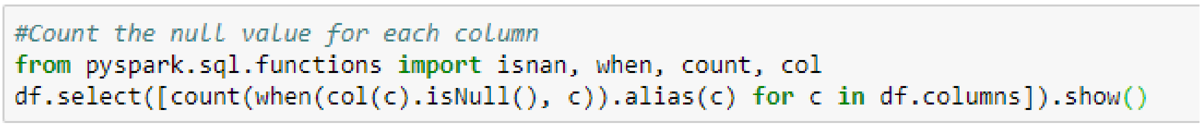
Figure 17: New Table

## 3.2 Clean the data

This step will focus on cleaning the data in order to increase the accuracy of data analysis. In my opinion, I think that there are no perfect datasets, which means the majority datasets have to do the process of cleaning the data. It is non-professional for a data scientist using a dataset without data preparation. Therefore, data preparation is significant for data analysis. Cleaning data is one of the most important concepts in data preparation. In addition, the majority of datasets have missing values, outliers and extremes, we have to decide whether these outliers and extremes should be removed. Therefore, we need to deal with those situations before starting data analysis.

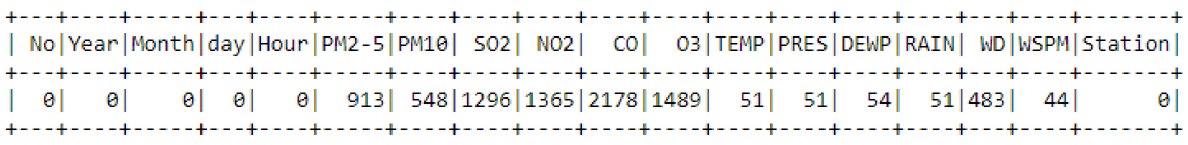
### 3.2.1 Missing Value

The first step of cleaning data is getting rid of missing value. In the Jupyter Notebook, I will use the isNull() function display the number of null value for each attribute. Figure 19 show the summary of the total number of null values for each attributes. I can observe there are 11 attributes that have null value, and there are only four attributes that have no null values. Then, I can will remove all null values in order to increase accuracy of prediction.



```
#Count the null value for each column
from pyspark.sql.functions import isnan, when, count, col
df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns]).show()
```

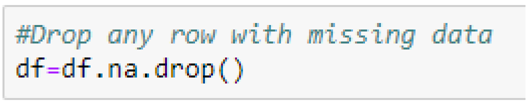
Figure 18: The Process of Checking Null Value



```
+---+----+-----+---+----+-----+----+----+----+----+----+----+----+----+----+---+----+-------+
| No|Year|Month|day|Hour|PM2-5|PM10| SO2| NO2|  CO|  O3|TEMP|PRES|DEWP|RAIN| WD|WSPM|Station|
+---+----+-----+---+----+-----+----+----+----+----+----+----+----+----+----+---+----+-------+
|  0|   0|    0|  0|   0|  913| 548|1296|1365|2178|1489|  51|  51|  54|  51|483|  44|      0|
+---+----+-----+---+----+-----+----+----+----+----+----+----+----+----+----+---+----+-------+
```

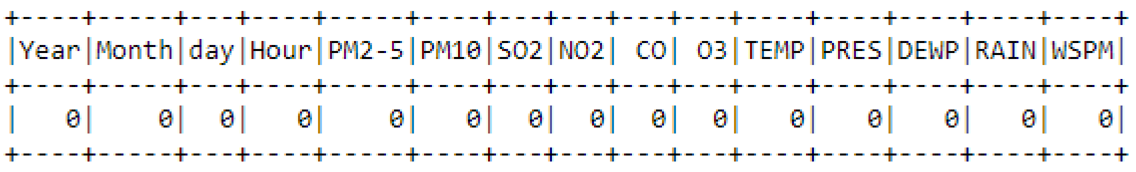
Figure 19: Summary of Null Value (Before Removing Null Value)

I will use na.drop() function to achieve the goal of removing null values. Figure 20 shows the process of removing null values in Spyder. Meanwhile, Figure 21 shows new summary after removing rows with null value.



```
#Drop any row with missing data
df=df.na.drop()
```

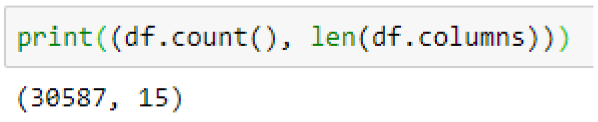
Figure 20: The Process of Removing Null Value



```
+----+-----+---+----+-----+----+---+---+---+---+----+----+----+----+----+
|Year|Month|day|Hour|PM2-5|PM10|SO2|NO2| CO| O3|TEMP|PRES|DEWP|RAIN|WSPM|
+----+-----+---+----+-----+----+---+---+---+---+----+----+----+----+----+
|   0|    0|  0|   0|    0|   0|  0|  0|  0|  0|   0|   0|   0|   0|   0|
+----+-----+---+----+-----+----+---+---+---+---+----+----+----+----+----+
```

Figure 21: Summary of Null Value (After Removing Null Value)

In conclusion, I also used isNull() function check the number of null values after the process of removing. According to the Figure 21, I can clearly observe all the number goes to zero, which means there is no null value for each attribute. In other words, the process of removing null values is successful. Figure 22 shows the current number of rows and attributes. In next step, I will discuss outliers and extremes.



```
print((df.count(), len(df.columns)))
```

```
(30587, 15)
```

Figure 22: Row Number and Attribute Number

### 3.2.2 Extreme Values and Outliers

An outlier or extreme value means a data point which distributes far from other data points. We need to remove outliers and extremes in some cases. There are two conditions that we need to get rid of outliers and extremes. The first condition is that the data was recorded incorrectly. For example, I want to investigate the average age in a company. If I find there are some records have negative value or exceed normal range, I would remove these outliers and extremes in order to increase the accuracy of results. The second condition is irrelative records. For example, I collect a dataset about height from 100 people (include 3 NBA players). If I want to predict average height in the US, I would remove these 3 NBA players. These data were recorded correctly, but preserving them many influence the accuracy of prediction. Therefore, I will discuss each attribute that have outliers and extremes. Then, I will decide whether should I preserve them.

In section two, I find eight attributes that have outliers and extremes, they are PM2.5, PM10, SO2, NO2, CO, O3, RAIN and WSPM.

PM2.5:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e9a207518>
```

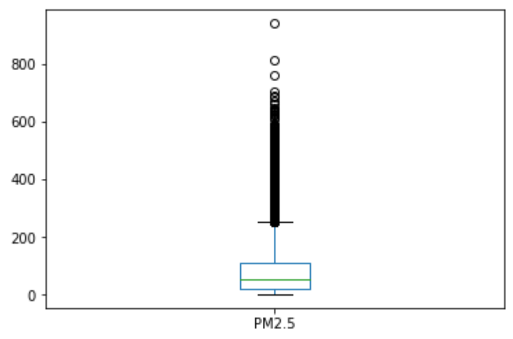
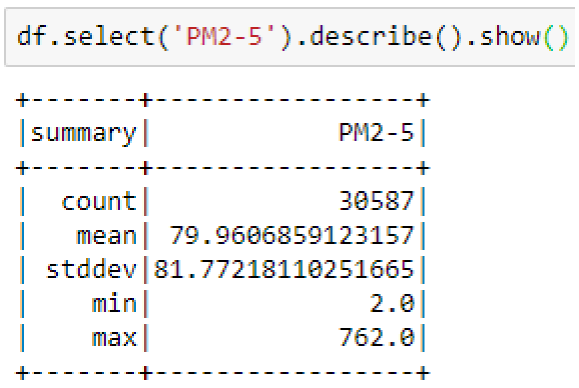


Figure 23: PM2.5



```
df.select('PM2-5').describe().show()
```

```
+-------+-----------------+
|summary|            PM2-5|
+-------+-----------------+
|  count|            30587|
|   mean| 79.9606859123157|
| stddev|81.7721811025166 5|
|    min|              2.0|
|    max|            762.0|
+-------+-----------------+
```

Figure 24: Describe of PM2.5

The value range is from 2 to 762 in PM2.5, and this range is reasonable and normal. In other word, all the values from PM2.5 are reliable and reasonable. I want to predict the air-quality in the future, every record should be considered. Therefore, I will keep outliers and extremes in PM2.5.

PM10:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e9a1c4080>
```

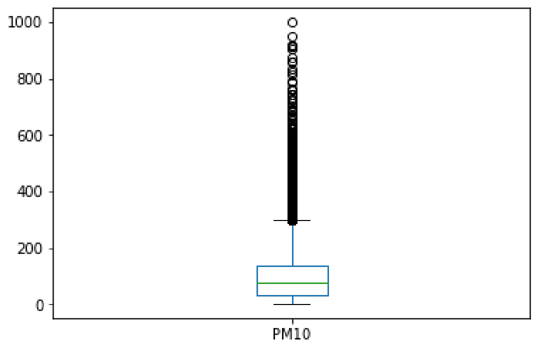
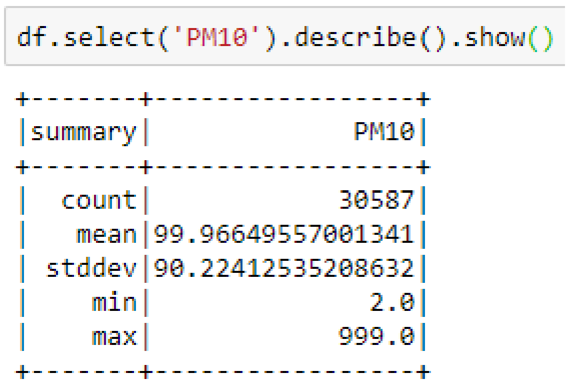


Figure 25: PM10



```
df.select('PM10').describe().show()
```

```
+-------+-----------------+
|summary|             PM10|
+-------+-----------------+
|  count|            30587|
|   mean|99.96649557001341|
| stddev|90.22412535208632|
|    min|              2.0|
|    max|            999.0|
+-------+-----------------+
```

Figure 26: Describe of PM10

The value range is from 2 to 999 in PM10, and this range is reasonable and normal. In other word, all the values from PM10 are reliable and reasonable. I want to predict the air-quality in the future, every record should be considered. Therefore, I will keep outliers and extremes in PM10.

SO2:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e9a181b00>
```

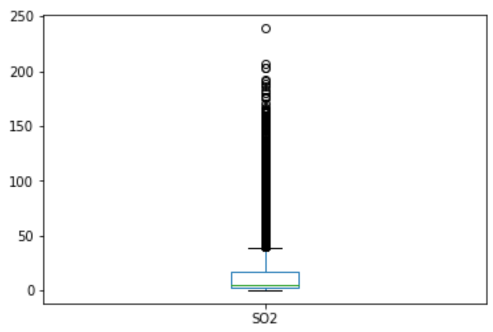
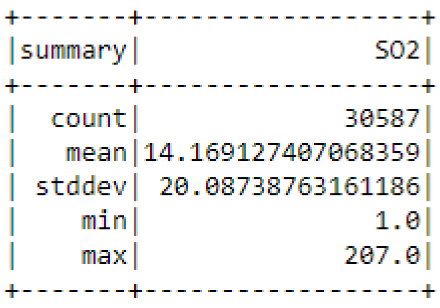


Figure 27: SO2



```
+-------+------------------+
|summary|               SO2|
+-------+------------------+
|  count|             30587|
|   mean|14.169127407068359|
| stddev| 20.08738763161186|
|    min|               1.0|
|    max|             207.0|
+-------+------------------+
```

Figure 28: Describe of SO2

The value range is from 1 to 207 in SO2, and this range is reasonable and normal. In other word, all the values from SO2 are reliable and reasonable. I want to predict the air-quality in the future, every record should be considered. Therefore, I will keep outliers and extremes in SO2.

NO2:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e9a131be0>
```

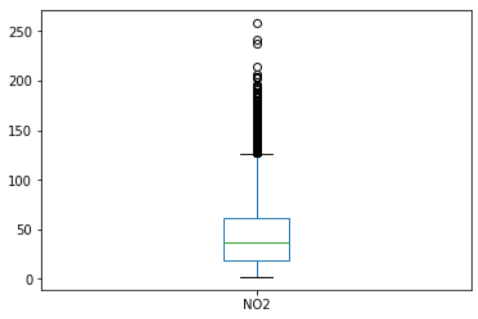


Figure 29: NO2

```
df.select('NO2').describe().show()
```

```
+-------+-----------------+
|summary|              NO2|
+-------+-----------------+
|  count|            30587|
|   mean| 45.2643005394486|
| stddev|31.28814926810135 7|
|    min|              2.0|
|    max|            258.0|
+-------+-----------------+
```

Figure 30: Describe of NO2

The value range is from 2 to 258 in NO2, and this range is reasonable and normal. In other word, all the values from NO2 are reliable and reasonable. I want to predict the air-quality in the future, every record should be considered. Therefore, I will keep outliers and extremes in NO2.

CO:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e9a0e9320>
```

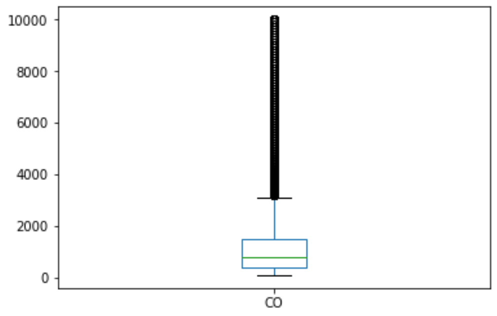


Figure 31: CO

```
df.select('CO').describe().show()
```

```
+-------+-----------------+
|summary|               CO|
+-------+-----------------+
|  count|            30587|
|   mean|1212.7764409716547|
| stddev|1171.4237334013976|
|    min|              100|
|    max|            10000|
+-------+-----------------+
```

Figure 32: Describe of CO

The value range is from 100 to 10000 in CO, and this range is reasonable and normal. In other word, all the values from CO are reliable and reasonable. I want to predict the air-quality in the future, every record should be considered. Therefore, I will keep outliers and extremes in CO.

O3:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e9a0aa518>
```

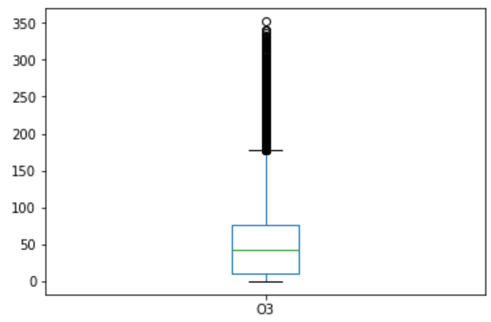
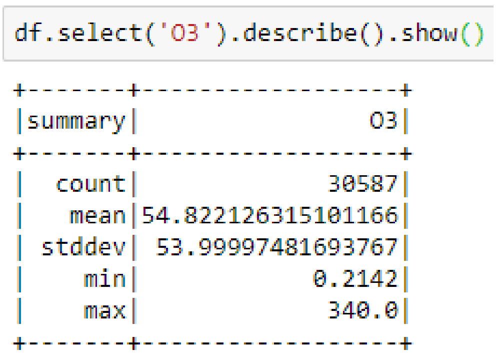


Figure 31: O3



```
df.select('O3').describe().show()
```

```
+-------+------------------+
|summary|                O3|
+-------+------------------+
|  count|             30587|
|   mean|54.822126315101166|
| stddev| 53.99997481693767|
|    min|            0.2142|
|    max|             340.0|
+-------+------------------+
```

Figure 32: Describe of O3

The value range is from 100 to 10000 in O3, and this range is reasonable and normal. In other word, all the values from O3 are reliable and reasonable. I want to predict the air-quality in the future, every record should be considered. Therefore, I will keep outliers and extremes in O3.

RAIN:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e99c56da0>
```

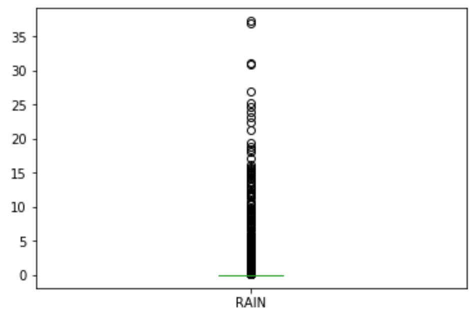
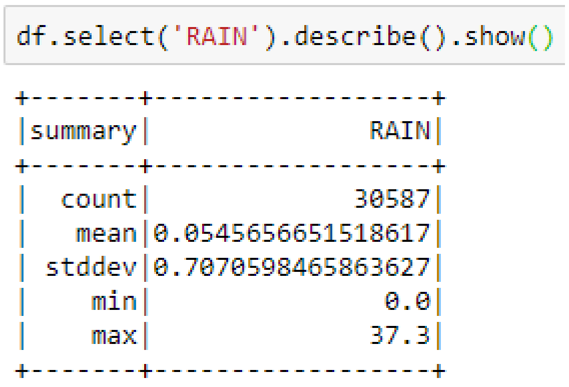


Figure 33: RAIN



```
df.select('RAIN').describe().show()
```

```
+-------+------------------+
|summary|              RAIN|
+-------+------------------+
|  count|             30587|
|   mean|0.0545656651518617|
| stddev|0.7070598465863627|
|    min|               0.0|
|    max|              37.3|
+-------+------------------+
```

Figure 34: Describe of RAIN

The value range is from 0 to 37.3 in RAIN, and this range is reasonable and normal. In other word, all the values from RAIN are reliable and reasonable. I want to predict the air-quality in the future, every record should be considered. Therefore, I will keep outliers and extremes in RAIN.

WSPM:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e99c13f60>
```

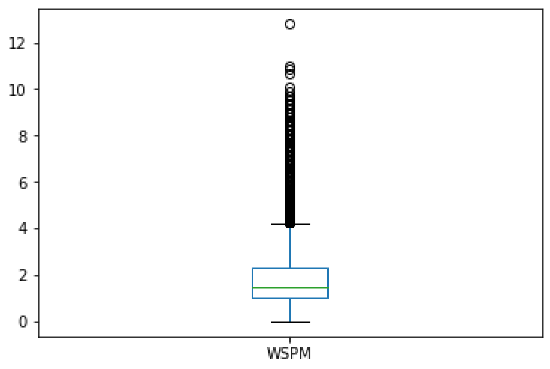
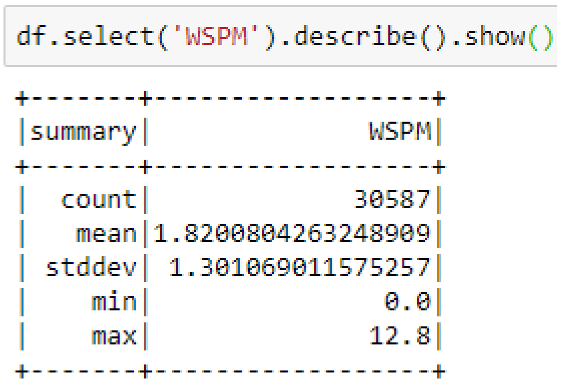


Figure 35: WSPM



```
df.select('WSPM').describe().show()
```

```
+-------+------------------+
|summary|              WSPM|
+-------+------------------+
|  count|             30587|
|   mean|1.8200804263248909|
| stddev| 1.301069011575257|
|    min|               0.0|
|    max|              12.8|
+-------+------------------+
```

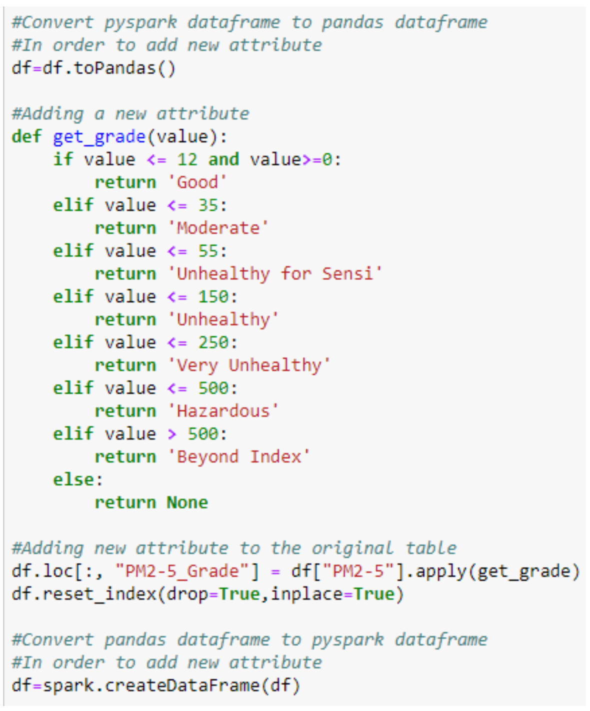
Figure 36: Describe of WSPM

The value range is from 0 to 12.8 in WSPM, and this range is reasonable and normal. In other word, all the values from WSPM are reliable and reasonable. I want to predict the air-quality in the future, every record should be considered. Therefore, I will keep outliers and extremes in WSPM.

In conclusion, I will preserve all outliers and extremes in order to increase accuracy of prediction. In my project, I need to consider all values that are in reasonable and normal range. There are no negative values, which means all records were records correctly. The values of outliers and extremes are true, so I will keep them in this step.

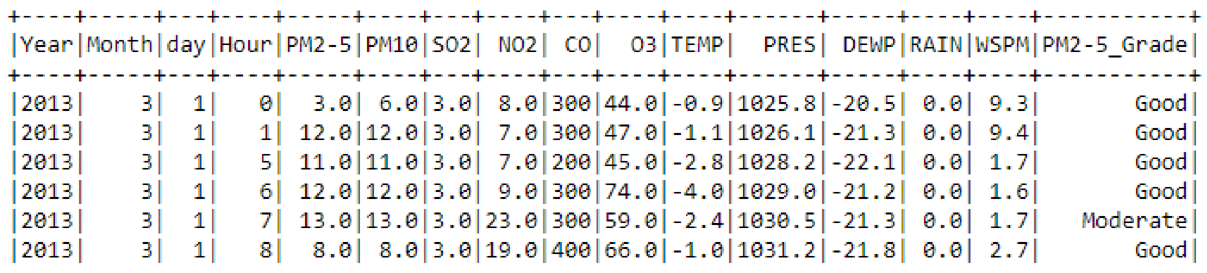
## 3.3 Construct the data

It is frequently the case that I need to construct new data. There are two approaches to construct new data, they are deriving new attributes (columns) and generating new records(rows). In my report, I will choose the first approach to construct new data. Before constructing new data, I have 15 attributes in my dataset. Then, I did some research and collected some information, I decided to add a new attribute named PM2.5_GRADE. In my dataset, PM2.5 is described by a series of numeric value, which is unclear for me. I cannot evaluate by numbers, so I try to find an approach which can convert these numeric values to textual value. I find that PM2.5 has seven levels, they are “Good”, “Moderate”, “Unhealthy for Sensi”, “Unhealthy”, “Very Unhealthy”, “Hazardous” and “Beyond Index”. Each level has different range of values, so I can clearly understand what the values of PM2.5 stand for. Figure 37 shows the process of adding a new attribute than named PM2.5_GRADE. Then, Figure 38 shows the table with new attribute, and the total number of attributes goes to 16.



```python
#Convert pyspark dataframe to pandas dataframe
#In order to add new attribute
df=df.toPandas()

#Adding a new attribute
def get_grade(value):
    if value <= 12 and value>=0:
        return 'Good'
    elif value <= 35:
        return 'Moderate'
    elif value <= 55:
        return 'Unhealthy for Sensi'
    elif value <= 150:
        return 'Unhealthy'
    elif value <= 250:
        return 'Very Unhealthy'
    elif value <= 500:
        return 'Hazardous'
    elif value > 500:
        return 'Beyond Index'
    else:
        return None

#Adding new attribute to the original table
df.loc[:, "PM2-5_Grade"] = df["PM2-5"].apply(get_grade)
df.reset_index(drop=True,inplace=True)

#Convert pandas dataframe to pyspark dataframe
#In order to add new attribute
df=spark.createDataFrame(df)
```

Figure 37: Process of Adding Attribute



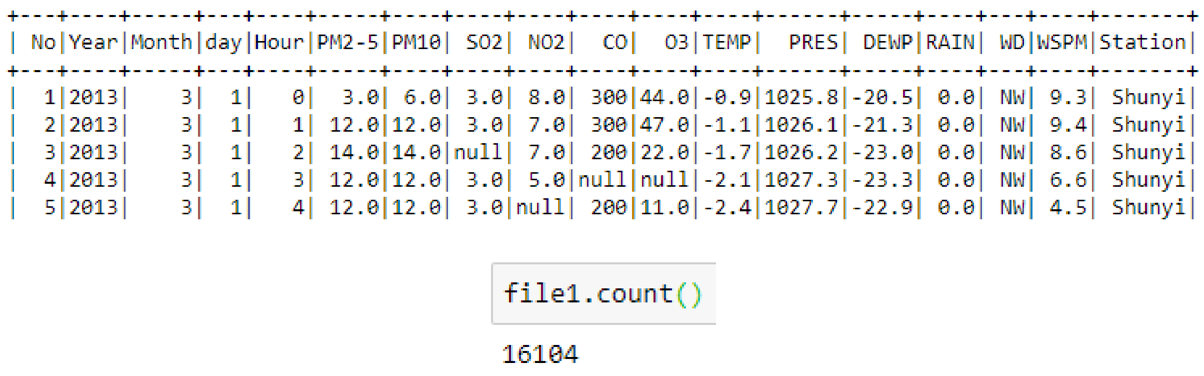
```
+----+-----+---+----+-----+----+---+----+---+----+----+------+-----+----+----+-----------+
|Year|Month|day|Hour|PM2-5|PM10|SO2| NO2| CO|  O3|TEMP|  PRES| DEWP|RAIN|WSPM|PM2-5_Grade|
+----+-----+---+----+-----+----+---+----+---+----+----+------+-----+----+----+-----------+
|2013|    3|  1|   0|  3.0| 6.0|3.0| 8.0|300|44.0|-0.9|1025.8|-20.5| 0.0| 9.3|       Good|
|2013|    3|  1|   1| 12.0|12.0|3.0| 7.0|300|47.0|-1.1|1026.1|-21.3| 0.0| 9.4|       Good|
|2013|    3|  1|   5| 11.0|11.0|3.0| 7.0|200|45.0|-2.8|1028.2|-22.1| 0.0| 1.7|       Good|
|2013|    3|  1|   6| 12.0|12.0|3.0| 9.0|300|74.0|-4.0|1029.0|-21.2| 0.0| 1.6|       Good|
|2013|    3|  1|   7| 13.0|13.0|3.0|23.0|300|59.0|-2.4|1030.5|-21.3| 0.0| 1.7|   Moderate|
|2013|    3|  1|   8|  8.0| 8.0|3.0|19.0|400|66.0|-1.0|1031.2|-21.8| 0.0| 2.7|       Good|
```

Figure 38: Table with New Attribute

In conclusion, the process of adding a new attribute is successful. In addition, every PM2.5 value has a relative level. The new attribute helps me have a clear understanding about the PM2.5 values. Therefore, constructing the data has a positive effect on understanding the content of dataset. In next step, I will try to integrate various data sources.

## 3.4 Integrate Various Data Sources

It is common to have multiple data sources for a data mining project. For example, attributes may be stored in different files according to their characteristic. Therefore, it is very important for me to learn how to integrate various data sources. There is only one file in my project, and I decide to separate this file into two files in order to learn integrate various data sources. Figure 39 shows the table of file one and Figure 40 shows the table of file two. The file one recorded air-quality from 2013 to 2014, the file two recorded air-quality from 2015 to 2017. In the Jupyter Notebook, I will use union() function integrate two files. Figure 41 shows the code of integrating various data sources. Figure 42 shows the new table after integrating. I observe all the attributes are displayed in new file, so the process of integrating is successful.

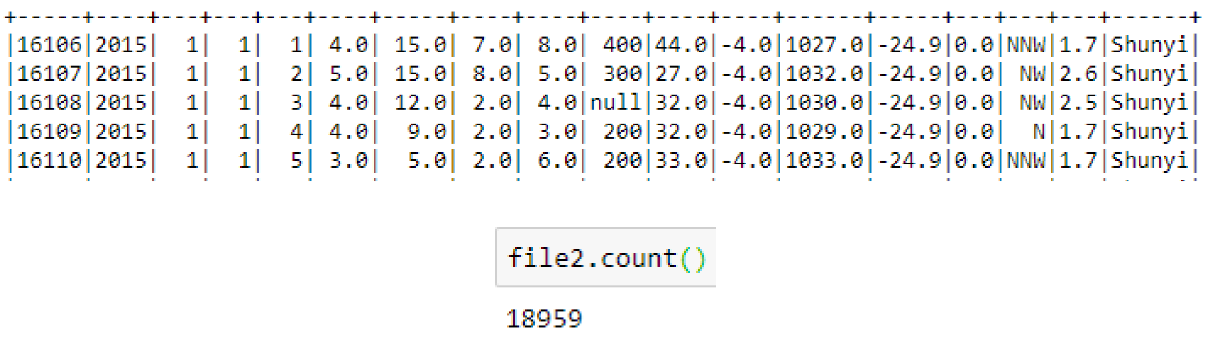


```
+---+----+-----+---+----+-----+----+----+----+----+----+----+------+-----+----+---+----+-------+
| No|Year|Month|day|Hour|PM2-5|PM10| SO2| NO2|  CO|  O3|TEMP|  PRES| DEWP|RAIN| WD|WSPM|Station|
+---+----+-----+---+----+-----+----+----+----+----+----+----+------+-----+----+---+----+-------+
|  1|2013|    3|  1|   0|  3.0| 6.0| 3.0| 8.0| 300|44.0|-0.9|1025.8|-20.5| 0.0| NW| 9.3| Shunyi|
|  2|2013|    3|  1|   1| 12.0|12.0| 3.0| 7.0| 300|47.0|-1.1|1026.1|-21.3| 0.0| NW| 9.4| Shunyi|
|  3|2013|    3|  1|   2| 14.0|14.0|null| 7.0| 200|22.0|-1.7|1026.2|-23.0| 0.0| NW| 8.6| Shunyi|
|  4|2013|    3|  1|   3| 12.0|12.0| 3.0| 5.0|null|null|-2.1|1027.3|-23.3| 0.0| NW| 6.6| Shunyi|
|  5|2013|    3|  1|   4| 12.0|12.0| 3.0|null| 200|11.0|-2.4|1027.7|-22.9| 0.0| NW| 4.5| Shunyi|
```

```
file1.count()
```

```
16104
```

Figure 39: Table of File 1

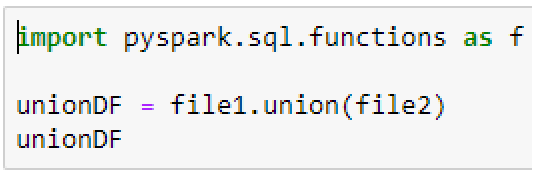


```
+-----+----+---+---+---+----+-----+----+----+----+----+----+------+-----+---+---+---+------+
|16106|2015|  1|  1|  1| 4.0| 15.0| 7.0| 8.0| 400|44.0|-4.0|1027.0|-24.9|0.0|NNW|1.7|Shunyi|
|16107|2015|  1|  1|  2| 5.0| 15.0| 8.0| 5.0| 300|27.0|-4.0|1032.0|-24.9|0.0| NW|2.6|Shunyi|
|16108|2015|  1|  1|  3| 4.0| 12.0| 2.0| 4.0|null|32.0|-4.0|1030.0|-24.9|0.0| NW|2.5|Shunyi|
|16109|2015|  1|  1|  4| 4.0|  9.0| 2.0| 3.0| 200|32.0|-4.0|1029.0|-24.9|0.0|  N|1.7|Shunyi|
|16110|2015|  1|  1|  5| 3.0|  5.0| 2.0| 6.0| 200|33.0|-4.0|1033.0|-24.9|0.0|NNW|1.7|Shunyi|
```

```
file2.count()
```

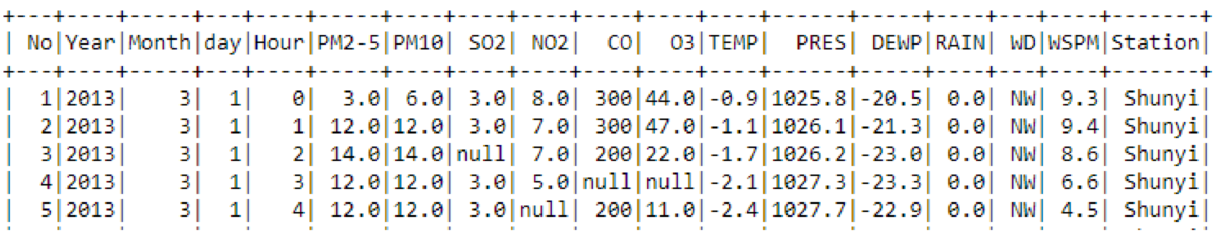
```
18959
```

Figure 40: Table of File 2

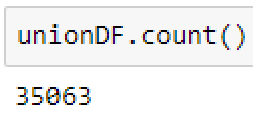


```
import pyspark.sql.functions as f

unionDF = file1.union(file2)
unionDF
```

Figure 41: Code of Integrating Two Files



```
+---+----+-----+---+----+-----+----+----+----+----+----+----+------+-----+----+---+----+-------+
| No|Year|Month|day|Hour|PM2-5|PM10| SO2| NO2|  CO|  O3|TEMP|  PRES| DEWP|RAIN| WD|WSPM|Station|
+---+----+-----+---+----+-----+----+----+----+----+----+----+------+-----+----+---+----+-------+
|  1|2013|    3|  1|   0|  3.0| 6.0| 3.0| 8.0| 300|44.0|-0.9|1025.8|-20.5| 0.0| NW| 9.3| Shunyi|
|  2|2013|    3|  1|   1| 12.0|12.0| 3.0| 7.0| 300|47.0|-1.1|1026.1|-21.3| 0.0| NW| 9.4| Shunyi|
|  3|2013|    3|  1|   2| 14.0|14.0|null| 7.0| 200|22.0|-1.7|1026.2|-23.0| 0.0| NW| 8.6| Shunyi|
|  4|2013|    3|  1|   3| 12.0|12.0| 3.0| 5.0|null|null|-2.1|1027.3|-23.3| 0.0| NW| 6.6| Shunyi|
|  5|2013|    3|  1|   4| 12.0|12.0| 3.0|null| 200|11.0|-2.4|1027.7|-22.9| 0.0| NW| 4.5| Shunyi|
```



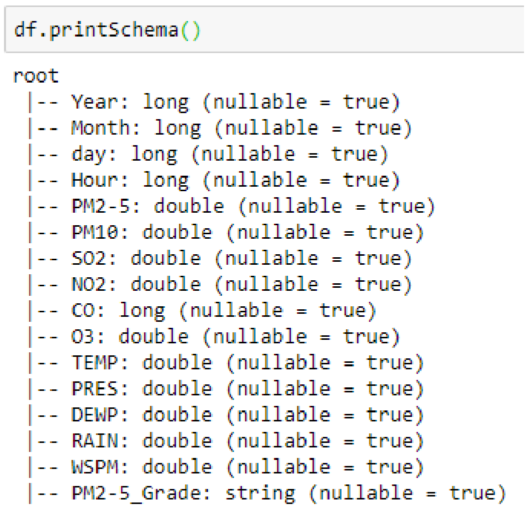
```
unionDF.count()
```

```
35063
```

Figure 42: Table (After Integrating)

## 3.5 Format the Data as Required

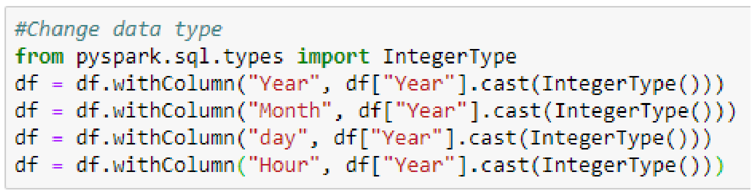
Formatting data is also an essential step in data preparation. In this step, we need to make sure every field is in correct format. In the Jupyter Notebook, I will use printSchema() function to display the types of all attributes. Figure 43 shows the results of printSchema () function. The long is not proper for year, Month, day and Hour. The range of long is too big to slow the process of execution, so I will convert long to integer.



```
df.printSchema()
```

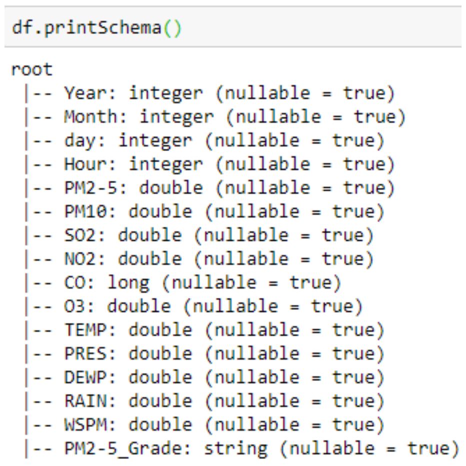
```
root
 |-- Year: long (nullable = true)
 |-- Month: long (nullable = true)
 |-- day: long (nullable = true)
 |-- Hour: long (nullable = true)
 |-- PM2-5: double (nullable = true)
 |-- PM10: double (nullable = true)
 |-- SO2: double (nullable = true)
 |-- NO2: double (nullable = true)
 |-- CO: long (nullable = true)
 |-- O3: double (nullable = true)
 |-- TEMP: double (nullable = true)
 |-- PRES: double (nullable = true)
 |-- DEWP: double (nullable = true)
 |-- RAIN: double (nullable = true)
 |-- WSPM: double (nullable = true)
 |-- PM2-5_Grade: string (nullable = true)
```

Figure 44: Attributes Types



```
#Change data type
from pyspark.sql.types import IntegerType
df = df.withColumn("Year", df["Year"].cast(IntegerType()))
df = df.withColumn("Month", df["Year"].cast(IntegerType()))
df = df.withColumn("day", df["Year"].cast(IntegerType()))
df = df.withColumn("Hour", df["Year"].cast(IntegerType()))
```

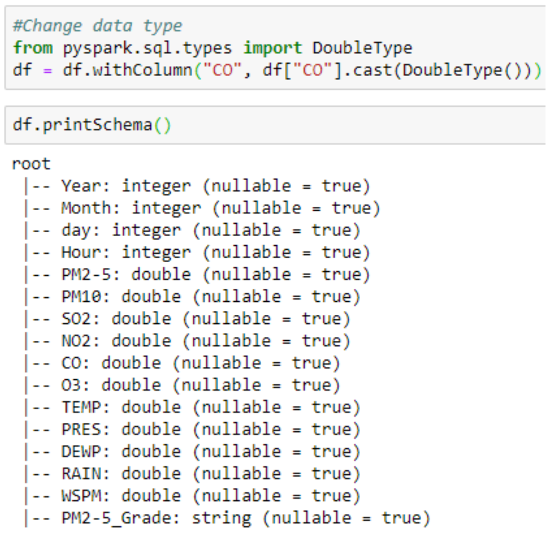
Figure 45: Code of Converting Data Type



```
df.printSchema()

root
 |-- Year: integer (nullable = true)
 |-- Month: integer (nullable = true)
 |-- day: integer (nullable = true)
 |-- Hour: integer (nullable = true)
 |-- PM2-5: double (nullable = true)
 |-- PM10: double (nullable = true)
 |-- SO2: double (nullable = true)
 |-- NO2: double (nullable = true)
 |-- CO: long (nullable = true)
 |-- O3: double (nullable = true)
 |-- TEMP: double (nullable = true)
 |-- PRES: double (nullable = true)
 |-- DEWP: double (nullable = true)
 |-- RAIN: double (nullable = true)
 |-- WSPM: double (nullable = true)
 |-- PM2-5_Grade: string (nullable = true)
```

Figure 46: Attributes Types (After converting)

In addition, I will change datatype for CO. the value of CO could be decimal. Therefore, I convert long to double.



```
#Change data type
from pyspark.sql.types import DoubleType
df = df.withColumn("CO", df["CO"].cast(DoubleType()))
```

```
df.printSchema()

root
 |-- Year: integer (nullable = true)
 |-- Month: integer (nullable = true)
 |-- day: integer (nullable = true)
 |-- Hour: integer (nullable = true)
 |-- PM2-5: double (nullable = true)
 |-- PM10: double (nullable = true)
 |-- SO2: double (nullable = true)
 |-- NO2: double (nullable = true)
 |-- CO: double (nullable = true)
 |-- O3: double (nullable = true)
 |-- TEMP: double (nullable = true)
 |-- PRES: double (nullable = true)
 |-- DEWP: double (nullable = true)
 |-- RAIN: double (nullable = true)
 |-- WSPM: double (nullable = true)
 |-- PM2-5_Grade: string (nullable = true)
```

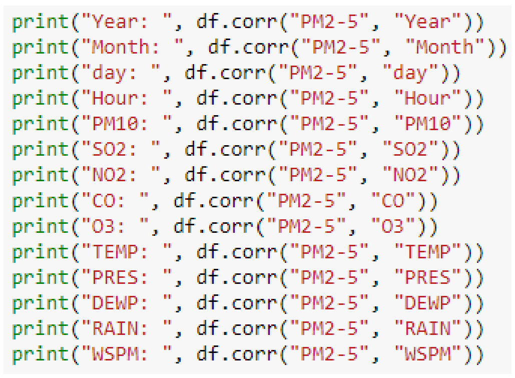
Figure 47: Attributes Types (After converting)

In conclusion, every attribute has an appropriate data type. This step will increase the speed of prediction, so it is necessary to use correct and reasonable data type. If I use proper datatype, the program may not have an exception. However, using proper datatype can increase the speed of execution.

## 4 Data transformation

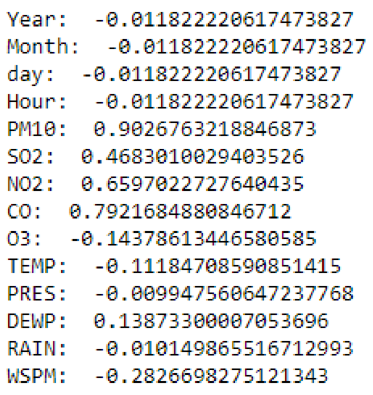
### 4.1 Reduce the Data

At this stage, we reduce the data in order to make the dataset tidier, which will make this dataset easy to analyse. The original dataset contains 35,064 rows and 18 attributes. There are 30,587 rows and 16 attributes after cleaning the data. In this step, I will use feature selection to reduce data. Feature selection can help me have a basic understanding about the important of all fields. In other word, I can figure out what features are important and what features are irrelevant. Then, I can remove irrelative attributes according to the result of feature selection in order to reduce the data. In Jupyter Notebook, the pyspark support corr() function, it is able to help me rank attributes according to their importance. Figure 48 shows the code of feature selection. Figure 49 shows the number of correlations. According to Figure 49, PM10, SO2, NO2, CO have strong positive correlation with PM2.5. In addition, DEWP has a weak positive correlation with PM2.5. Meanwhile, O3, WSPM and TEMP have a weak negative correlation with PM2.5. In contrast, other attributes have no relationship with PM2.5. Therefore, I will preserve relative attributes which include PM2.5, PM10, SO2, NO2, CO, DEWP, O3, WSPM, TEMP and WSPM.



```
print("Year: ", df.corr("PM2-5", "Year"))
print("Month: ", df.corr("PM2-5", "Month"))
print("day: ", df.corr("PM2-5", "day"))
print("Hour: ", df.corr("PM2-5", "Hour"))
print("PM10: ", df.corr("PM2-5", "PM10"))
print("SO2: ", df.corr("PM2-5", "SO2"))
print("NO2: ", df.corr("PM2-5", "NO2"))
print("CO: ", df.corr("PM2-5", "CO"))
print("O3: ", df.corr("PM2-5", "O3"))
print("TEMP: ", df.corr("PM2-5", "TEMP"))
print("PRES: ", df.corr("PM2-5", "PRES"))
print("DEWP: ", df.corr("PM2-5", "DEWP"))
print("RAIN: ", df.corr("PM2-5", "RAIN"))
print("WSPM: ", df.corr("PM2-5", "WSPM"))
```

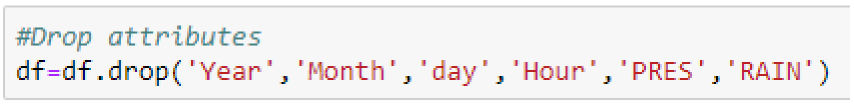
Figure 48: Code of Feature Selection



```
Year:  -0.011822220617473827
Month:  -0.011822220617473827
day:  -0.011822220617473827
Hour:  -0.011822220617473827
PM10:  0.9026763218846873
SO2:  0.4683010029403526
NO2:  0.6597022727640435
CO:  0.7921684880846712
O3:  -0.14378613446580585
TEMP:  -0.11184708590851415
PRES:  -0.009947560647237768
DEWP:  0.13873300007053696
RAIN:  -0.010149865516712993
WSPM:  -0.2826698275121343
```

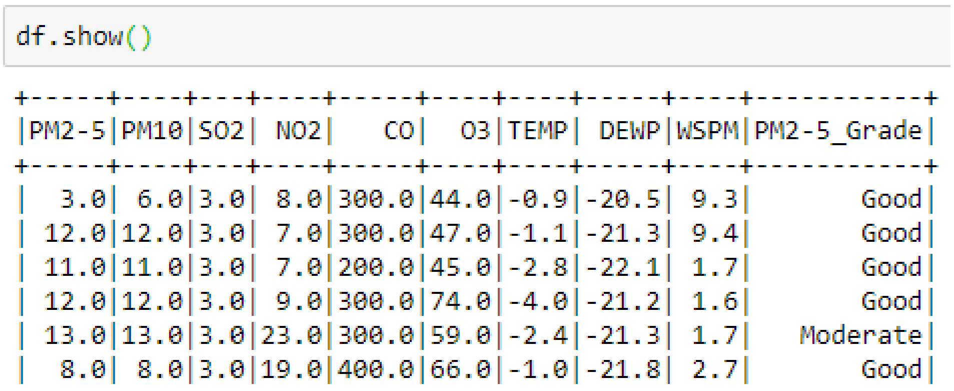
Figure 49: Correlation

Then, I will use python to remove all irrelative attributes in Jupyter Notebook. I will use drop() function to get rid of irrelative attributes. Figure 50 shows the code of removing attributes. Figure 51 shows the table after removing attributes.



```
#Drop attributes
df=df.drop('Year','Month','day','Hour','PRES','RAIN')
```

Figure 50: Code of Removing Attributes



```
df.show()
```

```
+-----+----+---+----+-----+----+----+-----+----+-----------+
|PM2-5|PM10|SO2| NO2|   CO|  O3|TEMP| DEWP|WSPM|PM2-5_Grade|
+-----+----+---+----+-----+----+----+-----+----+-----------+
|  3.0| 6.0|3.0| 8.0|300.0|44.0|-0.9|-20.5| 9.3|       Good|
| 12.0|12.0|3.0| 7.0|300.0|47.0|-1.1|-21.3| 9.4|       Good|
| 11.0|11.0|3.0| 7.0|200.0|45.0|-2.8|-22.1| 1.7|       Good|
| 12.0|12.0|3.0| 9.0|300.0|74.0|-4.0|-21.2| 1.6|       Good|
| 13.0|13.0|3.0|23.0|300.0|59.0|-2.4|-21.3| 1.7|   Moderate|
|  8.0| 8.0|3.0|19.0|400.0|66.0|-1.0|-21.8| 2.7|       Good|
```

Figure 51: Table (After Removing Attributes)

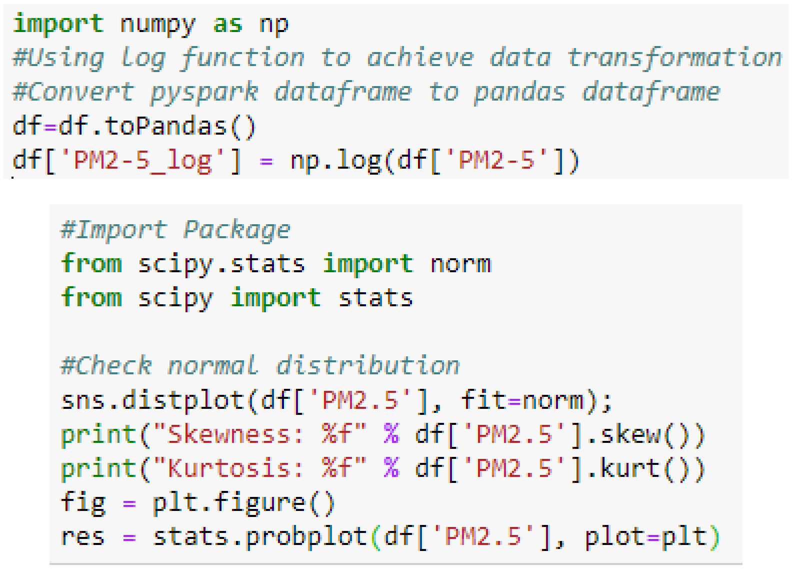
In conclusion, I remove all irrelative attributes in this step, I have 30,587 rows and 10 attributes after the process of removing. Reducing irrelative attribute can increase the speed of prediction, and we will not consider irrelative attributes.

## 4.2 Project the data

Data transformation is one of the most important steps in my project. Data transformation can help me get a more accurate and more reliable result. It also can make data easier to visualize, and the points will be spread more uniformly in a graph. I will use log() function to achieve the data transformation. The log transformation can not only reduce skew but also makes patterns more interpretable.

PM2.5:

Figure 53 shows a very skew distribution, which might lead to problems in analysis. Therefore, I use log() function in order to produce a normal distribution, it may make the data more proper and suitable for statistical analysis. Figure 54 show the new distribution after using log() function. The new distribution is not a perfect normal distribution, but the skewness decrease and the data will be easy to analyse. Then, I will display the distribution for other attributes in the following steps.



```
import numpy as np
#Using log function to achieve data transformation
#Convert pyspark dataframe to pandas dataframe
df=df.toPandas()
df['PM2-5_log'] = np.log(df['PM2-5'])
```

```
#Import Package
from scipy.stats import norm
from scipy import stats

#Check normal distribution
sns.distplot(df['PM2.5'], fit=norm);
print("Skewness: %f" % df['PM2.5'].skew())
print("Kurtosis: %f" % df['PM2.5'].kurt())
fig = plt.figure()
res = stats.probplot(df['PM2.5'], plot=plt)
```

Figure 52: Code

```
Skewness: 1.947186
Kurtosis: 5.349506
```

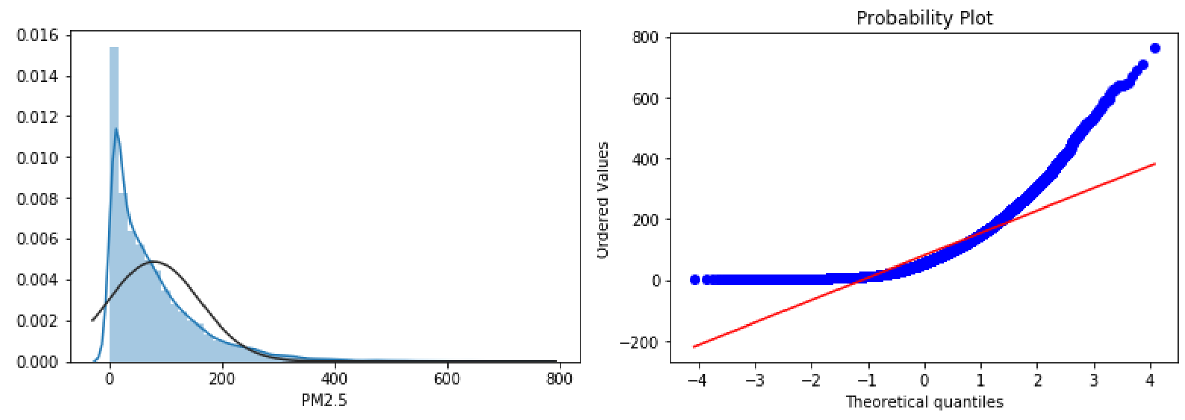


Figure 53: Distribution of PM2.5

```
Skewness: -0.395030
Kurtosis: -0.611573
```

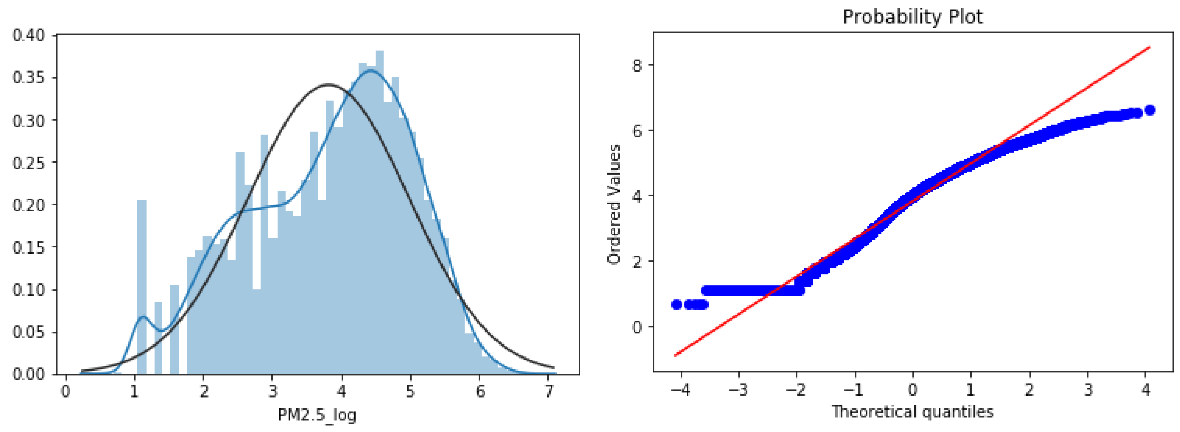


Figure 54: Distribution of PM2.5_log

PM10:

```
Skewness: 1.873400
Kurtosis: 6.349925
```

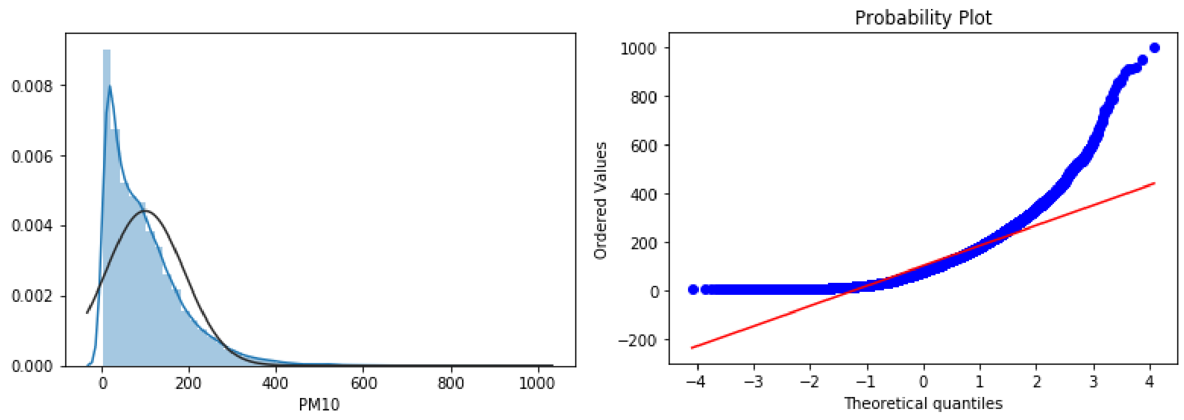


Figure 55: Distribution of PM10

```
Skewness: -0.528633
Kurtosis: -0.425809
```

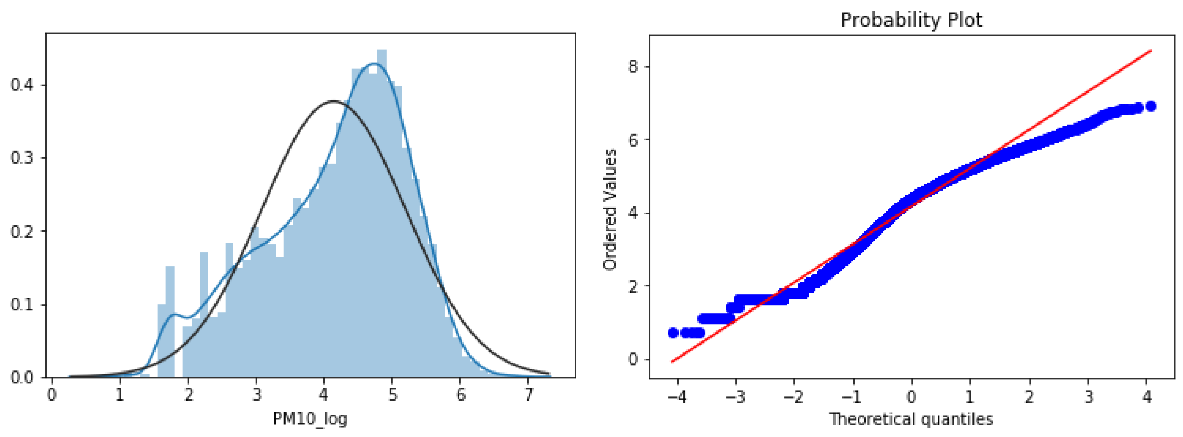


Figure 56: Distribution of PM10_log

SO2:

```
Skewness: 2.827960
Kurtosis: 10.896737
```

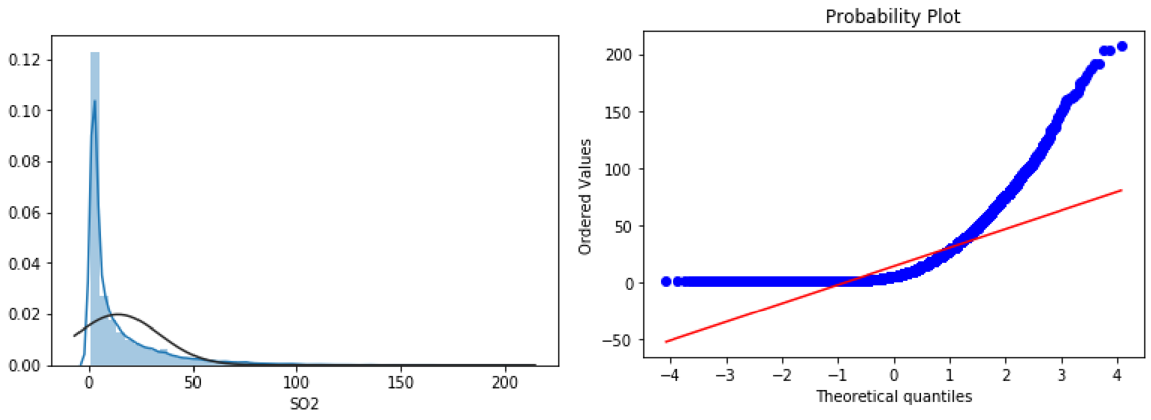


Figure 57: Distribution of SO2

```
Skewness: 0.480741
Kurtosis: -0.947704
```

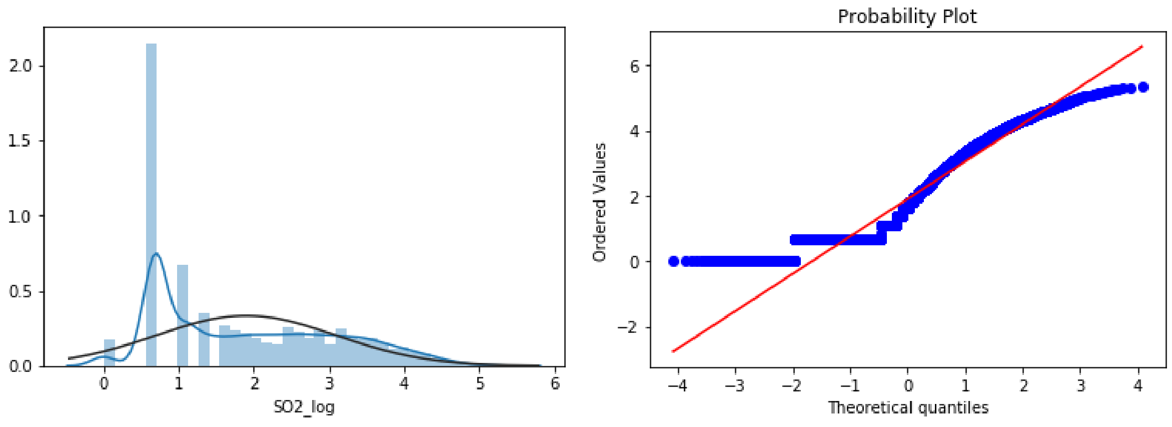


Figure 58: Distribution of SO2_log

NO2:

```
Skewness: 1.030570
Kurtosis: 1.038336
```

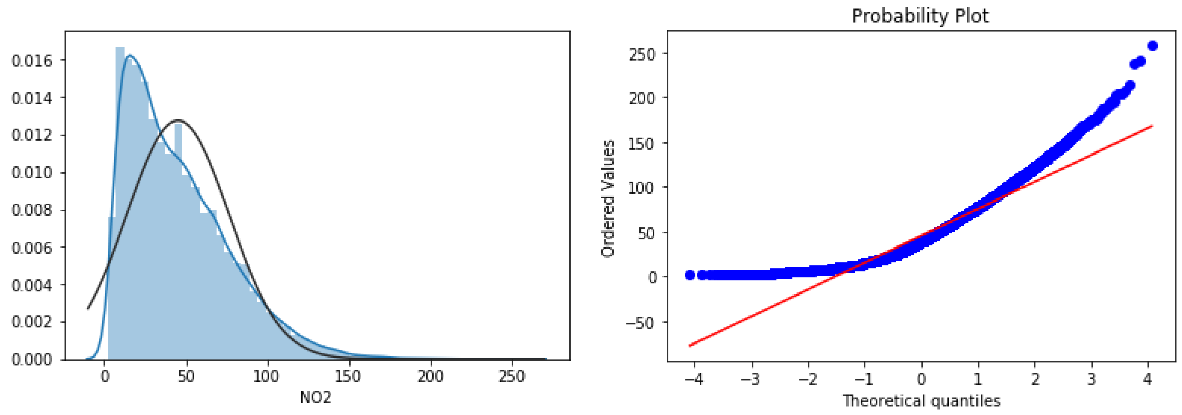


Figure 59: Distribution of NO2

```
Skewness: -0.529943
Kurtosis: -0.186013
```

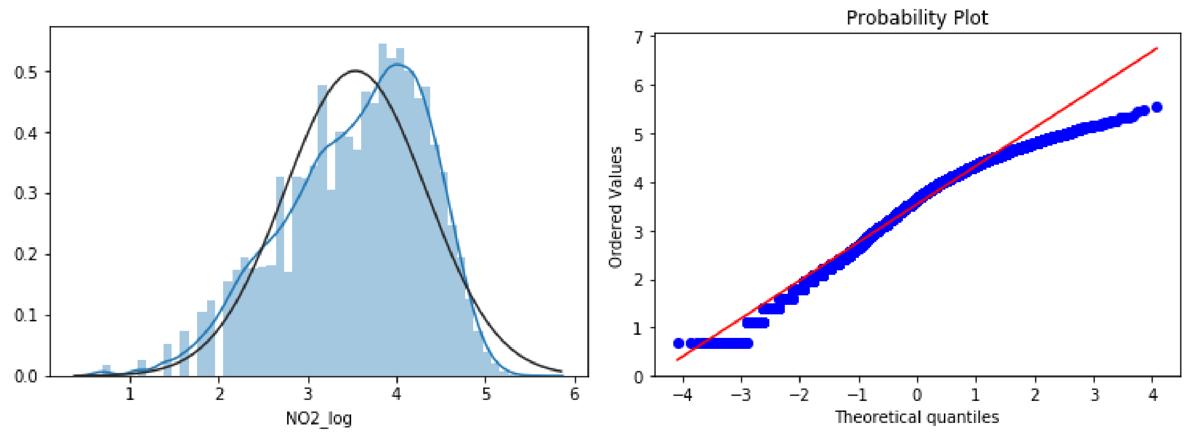


Figure 60: Distribution of NO2_log

CO:

```
Skewness: 2.556916
Kurtosis: 9.454768
```

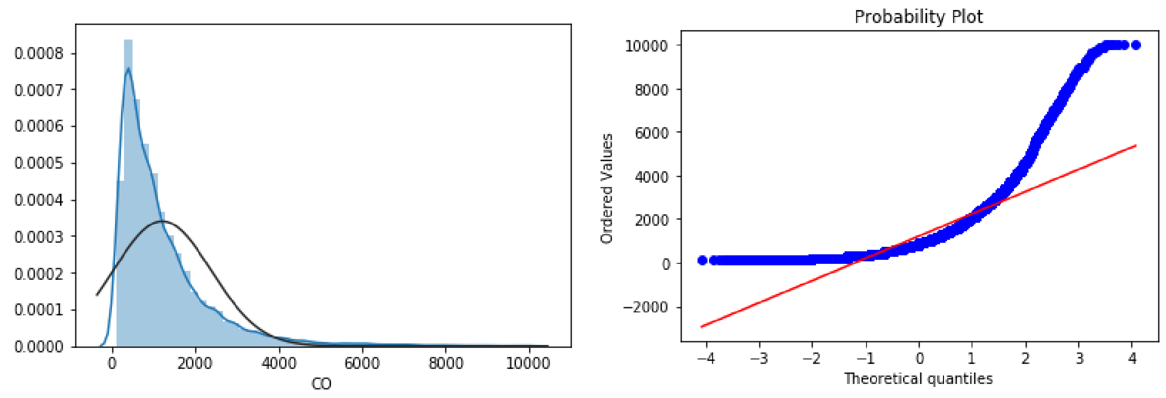


Figure 61: Distribution of CO

```
Skewness: -0.056549
Kurtosis: -0.336176
```

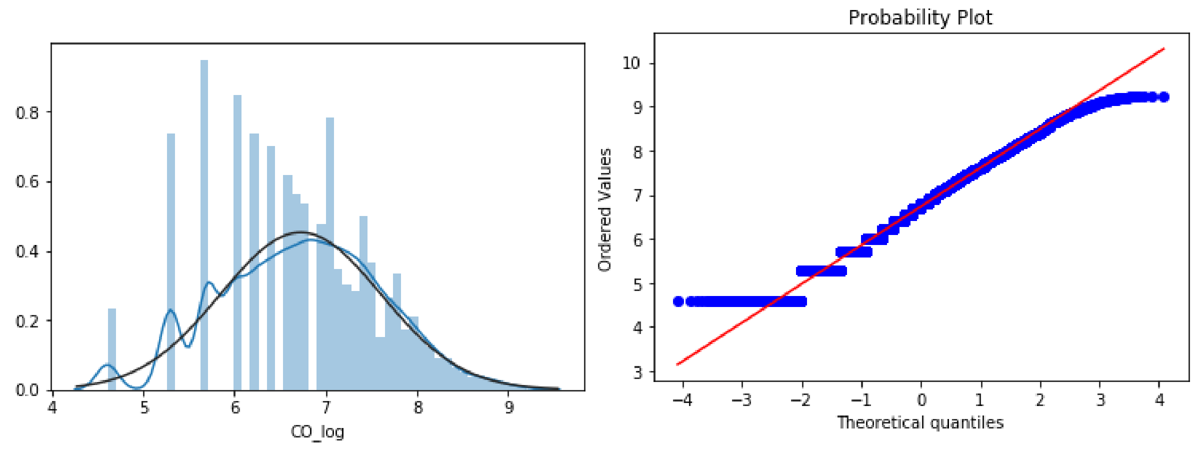


Figure 62: Distribution of CO_log

O3:

```
Skewness: 1.510587
Kurtosis: 2.542162
```

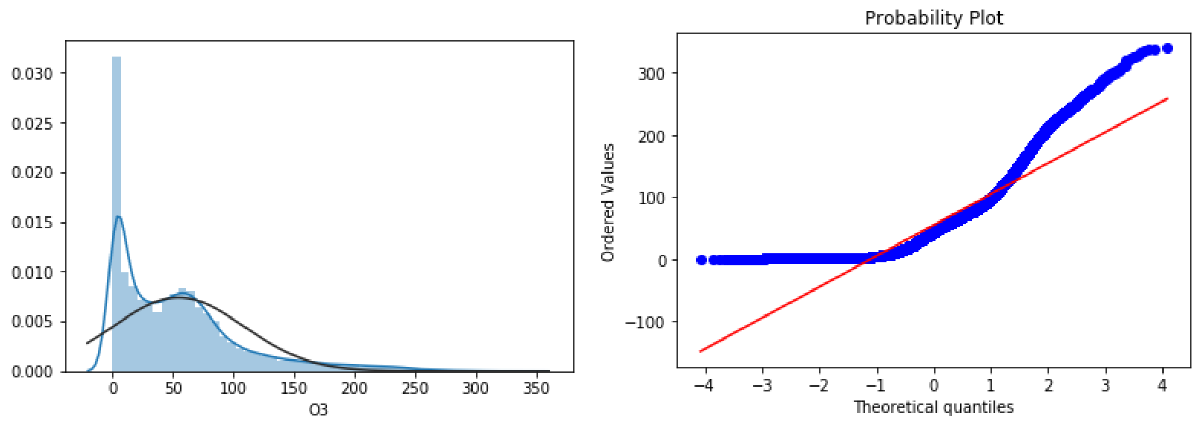


Figure 63: Distribution of O3

```
Skewness: -0.699523
Kurtosis: -0.571416
```

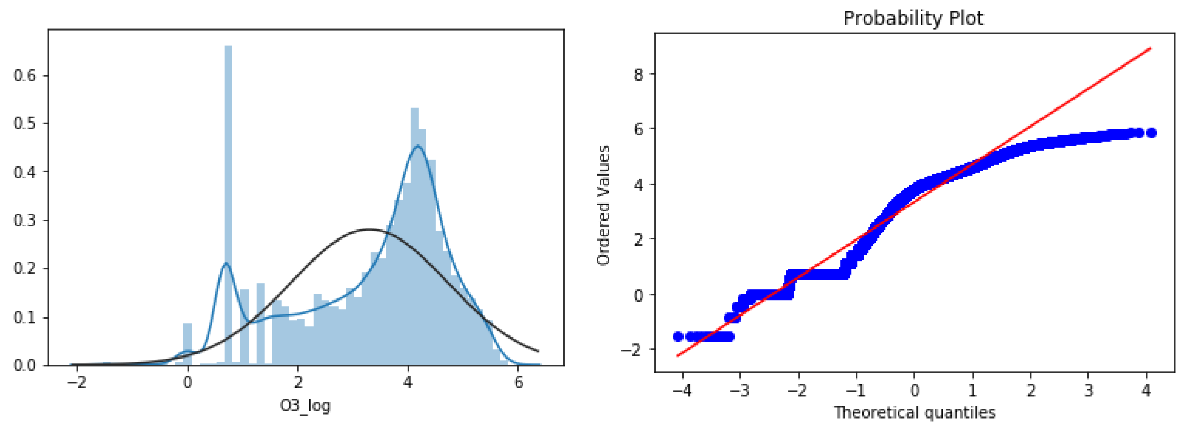


Figure 64: Distribution of O3_log

TEMP: Figure 65 shows the distribution of TEMP, it is very close to normal distribution. Therefore, I will not use log() function to TEMP.

```
Skewness: -0.022123
Kurtosis: -1.171961
```

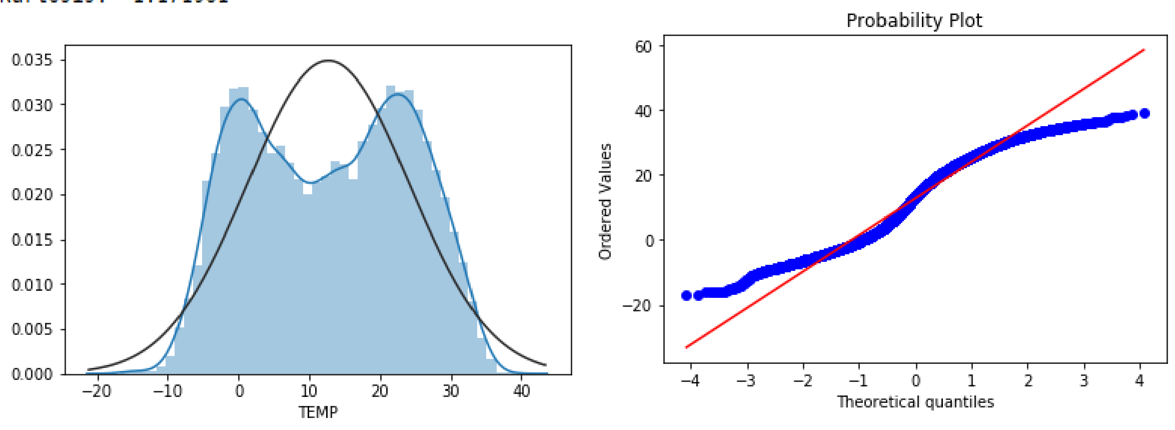


Figure 65: Distribution of TEMP

DEWP: Figure 66 shows the distribution of DEWP, it is very close to normal distribution. Therefore, I will not use log() function to DEWP.

```
Skewness: -0.123685
Kurtosis: -1.122988
```

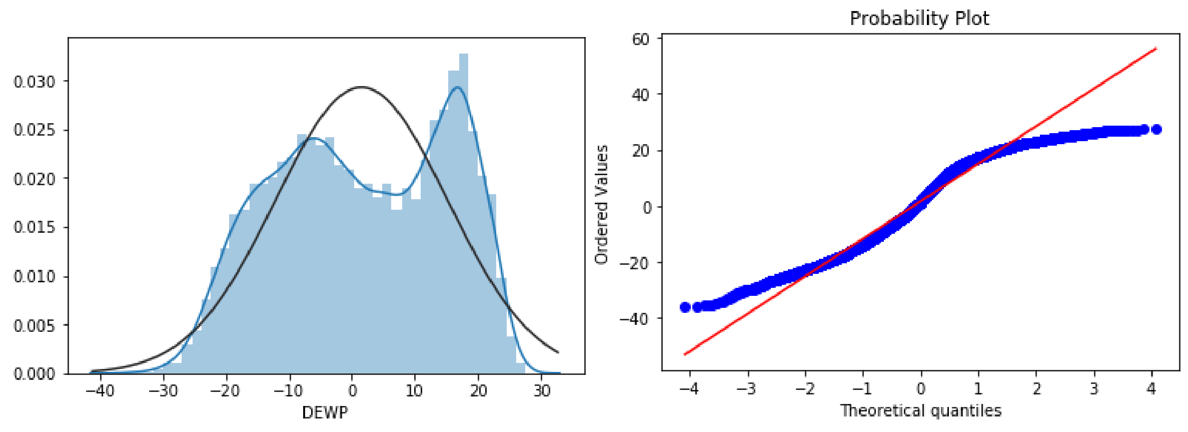


Figure 66: Distribution of DEWP

WSPM:

```
Skewness: 1.733997
Kurtosis: 4.152576
```

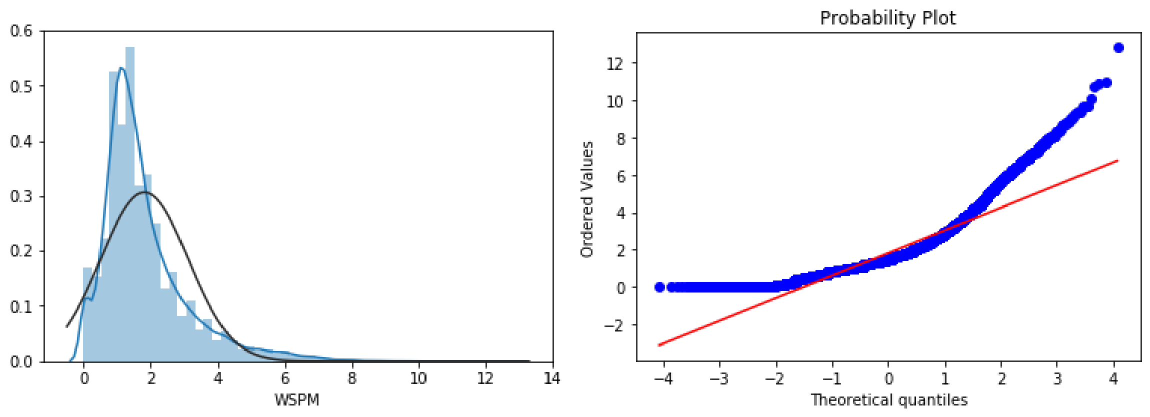


Figure 67: Distribution of WSPM

```
Skewness: -0.669760
Kurtosis: 1.912976
```

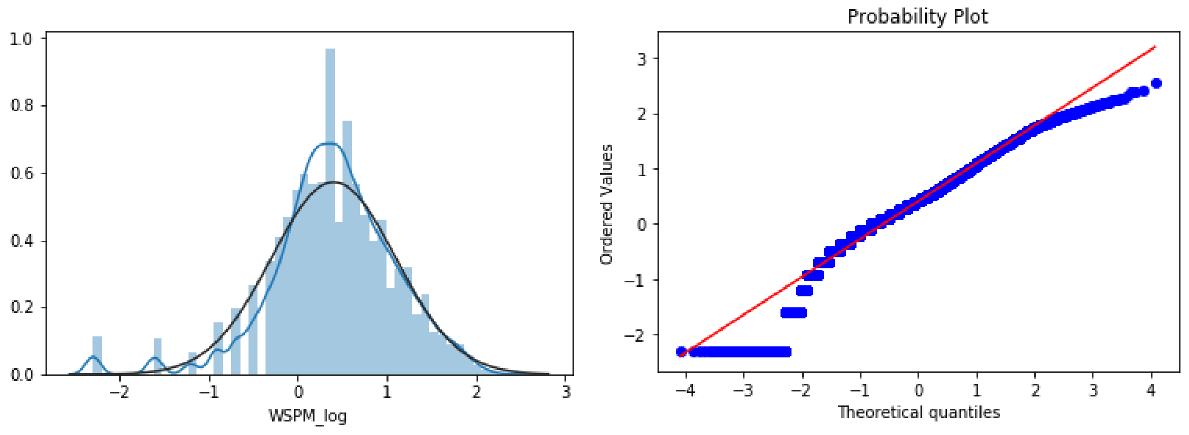


Figure 68: Distribution of WSPM_log

Figure 69 shows the table after transforming data. This table contains some old attributes, I need remove them using drop() function.

```
+-----+----+---+----+-----+----+----+-----+----+-----------+-----------------+-----------------+-----------------+---------
--------+-----------------+-----------------+------------------+
|PM2-5|PM10|SO2| NO2|   CO|  O3|TEMP| DEWP|WSPM|PM2-5_Grade|        PM2-5_log|         PM10_log|          SO2_log|
NO2_log|           CO_log|           O3_log|          WSPM_log|
+-----+----+---+----+-----+----+----+-----+----+-----------+-----------------+-----------------+-----------------+---------
--------+-----------------+-----------------+------------------+
|  3.0| 6.0|3.0| 8.0|300.0|44.0|-0.9|-20.5| 9.3|       Good|1.0986122886681098| 1.791759469228055|1.0986122886681098|2.07944154
16798357|5.703782474656201| 3.784189633918261| 2.2300144001592104|
```

Figure 69: Table(With Old Attributes)

Figure 70 shows the new table with no old attributes. However, name should be in same format. I will change them in next graph.

```
#Drop attributes
df=df.drop('PM2.5','PM10','SO2','NO2','CO','O3', 'WSPM')
```

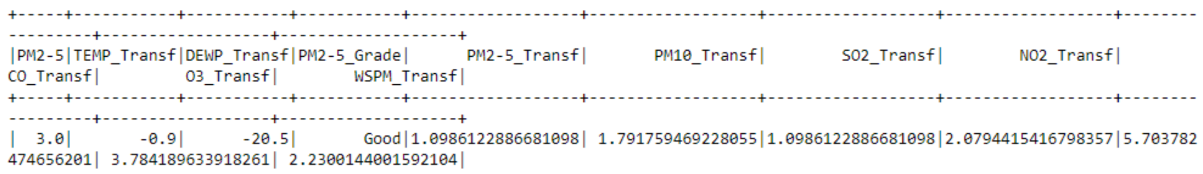
```
+-----+----+-----+-----------+-----------------+-----------------+-----------------+-----------------+-----------------+---
---------------+------------------+
|PM2-5|TEMP| DEWP|PM2-5_Grade|        PM2-5_log|         PM10_log|          SO2_log|          NO2_log|           CO_log|
        O3_log|          WSPM_log|
+-----+----+-----+-----------+-----------------+-----------------+-----------------+-----------------+-----------------+---
---------------+------------------+
|  3.0|-0.9|-20.5|       Good|1.0986122886681098| 1.791759469228055|1.0986122886681098|2.0794415416798357|5.703782474656201| 3.
784189633918261| 2.2300144001592104|
```

Figure 70: New Table

Figure 72 shows the final version of my table.

```
df = df.withColumnRenamed("TEMP","TEMP_Transf")
df = df.withColumnRenamed("DEWP","DEWP_Transf")
df = df.withColumnRenamed("PM2-5_log","PM2-5_Transf")
df = df.withColumnRenamed("PM10_log","PM10_Transf")
df = df.withColumnRenamed("SO2_log","SO2_Transf")
df = df.withColumnRenamed("NO2_log","NO2_Transf")
df = df.withColumnRenamed("CO_log","CO_Transf")
df = df.withColumnRenamed("O3_log","O3_Transf")
df = df.withColumnRenamed("WSPM_log","WSPM_Transf")
```

Figure 71: Code of Changing Name



```
+-----+-----------+-----------+-----------+------------------+------------------+------------------+------------------+--------
----------+------------------+------------------+
|PM2-5|TEMP_Transf|DEWP_Transf|PM2-5_Grade|      PM2-5_Transf|        PM10_Transf|        SO2_Transf|        NO2_Transf|
CO_Transf|         O3_Transf|       WSPM_Transf|
+-----+-----------+-----------+-----------+------------------+------------------+------------------+------------------+--------
----------+------------------+------------------+
|  3.0|       -0.9|      -20.5|       Good|1.0986122886681098| 1.791759469228055|1.0986122886681098|2.0794415416798357|5.703782
474656201| 3.784189633918261| 2.2300144001592104|
```

Figure 72: Final Table

# 5 Data-mining Method(s) Selection

## 5.1 Match and discuss the objectives of data mining (1.1) to data mining methods

Objectives of data mining:

**Prediction**: This project will use the dataset of air-quality in Shunyi to build a data model in order to predict the air-quality in the future.

**Relationship**: This project will use the dataset of air-quality in Shunyi to find relationships between PM 2.5 and other attributes.

There are five main data mining methods, they are anomaly detection, association learning, cluster detection, classification and regression.

**Anomaly Detection (unsupervised):** Mckell (2018, March 27) stated “Anomaly detection is all about trying to figure out when something is no longer in its normal behaviour”. Anomaly detection can be used to detect if something is abnormal. In other word, anomaly detection means that finding out unexpected patterns or events in regular patterns.

**Association Learning (unsupervised):** Mckell (2018, March 27) stated “Association learning, or market-basket analysis, is used to analyse which things tend to occur together either in pairs or larger groups”. For example, customers who buy mikes will buy eggs, or customers who purchase infant formula will also buy diapers. Then, the manager may display baby mike and diapers in same area. Therefore, this method can be used to find the connections between two to more goods or items. People can use this method to make some business plan.

**Cluster Detection (unsupervised):** The cluster detection (also clustering) is an effective data-mining method which can deal with a larger number of records. The cluster detection will separate a dataset in different groups according to the characteristic of each group.

**Classification (supervised):** Mckell (2018, March 27) stated “Unlike cluster detection, classification deals with things that already have labels. This is referred to as training data — when there is information existing that can be trained on, or rather easily classified with an algorithm”. Classification models are used to predict categorical values.

**Regression (supervised):** The regression analysis is the most popular method in big data analysis. Foley (2018, February 14) said “Regression analysis is a powerful statistical method that allows you to examine the relationship between two or more variables of interest”. Regression is used to predict numerical values, which includes identifying distribution trends based on available data.

There are two main types of data mining techniques, they are supervised data mining and unsupervised data mining. For the supervised data mining, it is predictive and directed. For the unsupervised data mining, it is descriptive and undirected. Then I will discuss which method is suitable to match my objective.

**Objective One:**

In my opinion, I think that I should use supervised data mining methods for my objective one. I will use predictive modelling in this objective, the predictive modelling is a process of using historical data to predict future trends or events. In my project, I collect the attributes of air-quality from 2013 to 2017, these are historical data. Then, I will use these data and select an appropriate model to analyse and predict the air-quality in the future.

**Objective Two:**

In my opinion, I think that I still can use supervised data mining methods for this objective. In this objective, I still use historical data to predict future trends. I will find a relationship between PM2.5 and other attributes. Therefore, supervised data mining methods are very appropriate to deal with this objective.

### 5.2 Select the appropriate data-mining method(s) based on discussion

I decided to use supervised data mining technique previous step. There are two main methods, they are regression and classification. This this step, I will choose the most appropriate data-mining methods. According to the basic definition of regression, it is used to predict numerical values and it can also identify distribution trends based on historical data. According to the definition of classification, the classification models are mainly used to predict categorical values. In my dataset, all the attributes are described by numerical values. In addition, the first objective is predicting the air-quality in the future. Therefore, the first objective can use regression as the data-mining method. Furthermore, regression is a statistical method which can determine the strength of relationships between one dependent variable and independent variables, so the second objective can use regression as the data-mining method.

In conclusion, I will select regression as the data-mining method for my objectives. Regression includes a series of algorithms, so I will select proper algorithms in next step.

# 6 Data-mining Algorithm(s) Selection(s)

### 6.1 Conduct exploratory analysis and discuss

In the previous step, I choose regression as my data-mining method. Regression has various data-mining algorithms, like random forest regression, linear regression, logistic regression, etc. In this step, I will explore the characteristic of these regression algorithms. Then, I will choose appropriate algorithms based on discussion.

**Linear regression:** Linear regression is the most common predictive modelling algorithm; it predicts future scores of a dependent variable based on one or more independent variables. Linear regression describes a relationship between target and predictors. For the linear regression there are also requirements. For example, all variables should be numerical in linear regression models. In addition, linear regression only allows exactly one dependent variable, but it allows one or more independent variables.

**Logistic regression:** Logistic regression (also known as nominal regression) is mainly used to describe the relationships between a binary response variable. The main usage of logistic regression is for classification. For example, the probability of a specific event is true (yes, 0) or false (no, 1). In addition, linear regression does not need to establish a linear relationship.

**Random Forest Regression:** Chakure (2019, June 29) stated "Random forest is a **Supervised Learning algorithm** which uses ensemble learning method for **classification and regression**. **Random forest** is a **bagging** technique and **not a boosting** technique. The trees in **random forests** are run in parallel. There is no interaction between these trees while building the trees. It operates by constructing a multitude of decision trees at training time and outputting the class that is the **mode** of the **classes (classification)** or **mean prediction (regression)** of the individual trees".

**CART Tree:** Decision tree are widely used in data-mining, it will create a predictive modelling which is able to predict the value of one dependent variable based on the values of one or more independent variables. CART Tree also knowns as Classification and Regression Trees, which means it can not only construct a classification tree but also can construct a regression tree. For the regression tree, the target variable should be continuous. The CART Tree require one target variable and one or more input fields.

## 6.2 Select data-mining algorithms based on discussion

In the 6.1, I discuss four data-mining algorithms. I will choose appropriate data-mining algorithms based on discussion in this step. In this project, I would like to find a relationship or trends between PM2.5 and other attributes. In my opinion, I think I can use more than one algorithm to achieve my data-mining goal.

First of all, I will select linear regression as my data-mining algorithm, because my data-mining is explore and predict a relationship between PM2.5 and other attributes. According to the definition of linear regression, it mainly used to predict the future scores of a dependent variable (target variable) based on one or more independent variables (input fields), so linear regression is an appropriate data-mining algorithm.

Secondly, I also want to use random forest regression as my data-mining algorithm, because it is one of the most accurate algorithms in data-mining learning. In addition, it can estimate of which input fields that are significant in the regression. Therefore, I will also use random forest regression in the following steps.

In conclusion, I selected linear regression and random forest regression as my data-mining algorithms. I will select appropriate models and choose relevant parameters in next step.

## 6.3 Build / Select appropriate models(s) and choose relevant parameters(s)

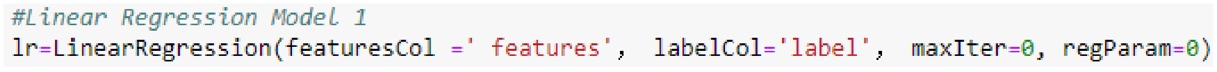
**Linear Regression:**

Parameter:

maxIter: the max value of iteration

regParam: regularization parameter

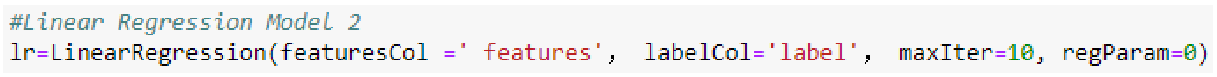
Model 1: (maxIter=0, regParam=0)



```
#Linear Regression Model 1
lr=LinearRegression(featuresCol =' features',  labelCol='label',  maxIter=0, regParam=0)
```

Figure 73: Linear Regression M1

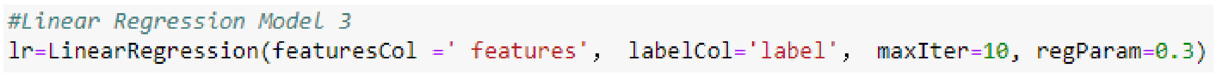
Model 2: (maxIter=10, regParam=0)



```
#Linear Regression Model 2
lr=LinearRegression(featuresCol =' features',  labelCol='label',  maxIter=10, regParam=0)
```

Figure 74: Linear Regression M2

Model 3: (maxIter=10, regParam=0.3)



```
#Linear Regression Model 3
lr=LinearRegression(featuresCol =' features',  labelCol='label',  maxIter=10, regParam=0.3)
```

Figure 75: Linear Regression M3

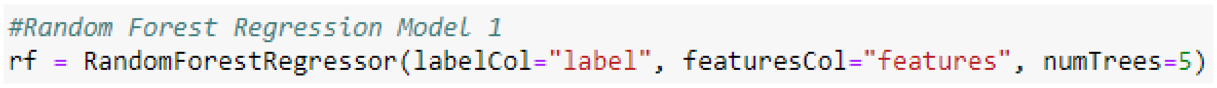
For the linear regression, I want to use three different models for linear regression. I choose maxIter and regParam as my selected parameters. The value of maxIter is the max time of iteration, and the regParam means regularization parameter. I would like to explore how maxIter influence the result of prediction and how regParam influence the result of prediction.

**Random Forest Regression:**

Parameter:

numTrees: it is the number of trees to be used in the forest.

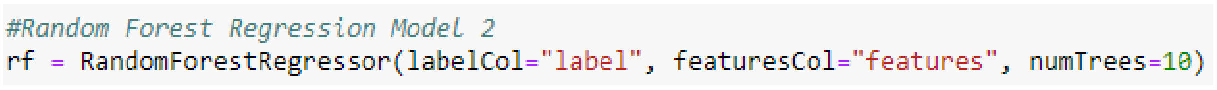
Model 1:



```
#Random Forest Regression Model 1
rf = RandomForestRegressor(labelCol="label", featuresCol="features", numTrees=5)
```

Figure 76: Random Forest Regression M1

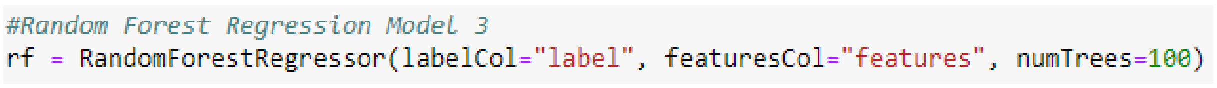
Model 2:



```
#Random Forest Regression Model 2
rf = RandomForestRegressor(labelCol="label", featuresCol="features", numTrees=10)
```

Figure 77: Random Forest Regression M2

Model 3:



```
#Random Forest Regression Model 3
rf = RandomForestRegressor(labelCol="label", featuresCol="features", numTrees=100)
```

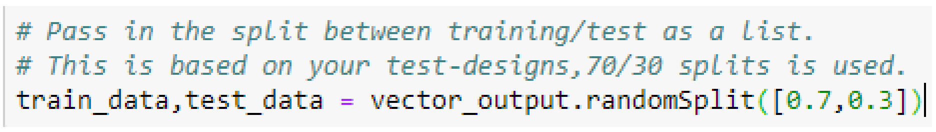
Figure 78: Random Forest Regression M3

For the random forest regression, I will display three different models. I will set different value to numTrees. I want to figure out how the number of trees influence my result.

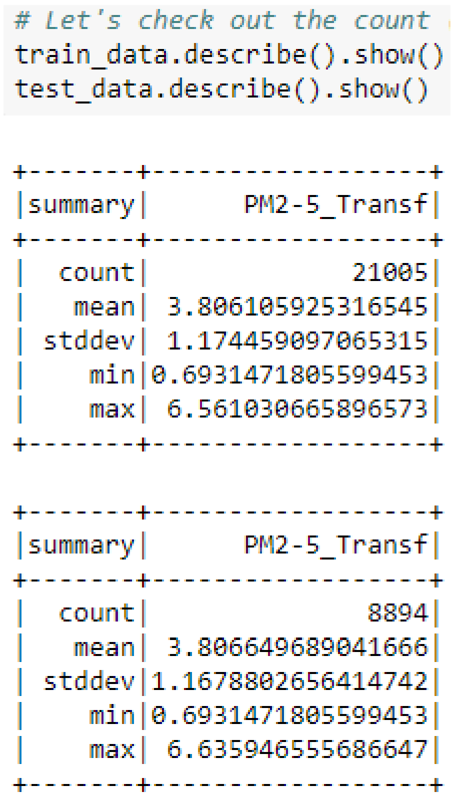
# 7 Data Mining

## 7.1 Create and justify tests designs

At this stage, I will create a logical test design which will separate my data in two group. This process is a very important step in data mining. Some rows will be assigned to group one which named as training group. Others will be assigned to group two which named as testing group. In my project, 70% of data will be assigned to the training group, and 30% of data will be assigned to testing group. In the Spyder, I can use random_split() function to achieve this goal. Figure 79 shows the code of separate data in training and testing group. The 0.3 means that there are 30% of data was assigned to testing group, and the 0.7 means 70% of data was assigned to training group.



```
# Pass in the split between training/test as a list.
# This is based on your test-designs,70/30 splits is used.
train_data,test_data = vector_output.randomSplit([0.7,0.3])
```



```
# Let's check out the count
train_data.describe().show()
test_data.describe().show()
```

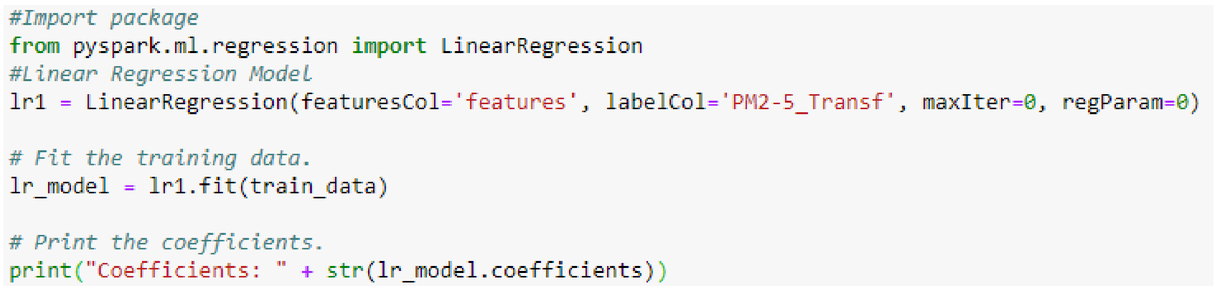
```
+-------+------------------+
|summary|       PM2-5_Transf|
+-------+------------------+
|  count|             21005|
|   mean| 3.806105925316545|
| stddev| 1.174459097065315|
|    min|0.6931471805599453|
|    max| 6.561030665896573|
+-------+------------------+

+-------+------------------+
|summary|       PM2-5_Transf|
+-------+------------------+
|  count|              8894|
|   mean| 3.806649689041666|
| stddev|1.1678802656414742|
|    min|0.6931471805599453|
|    max| 6.635946555686647|
+-------+------------------+
```

Figure 79: Code of Spliting Data

## 7.2 Conduct data mining – classify, regress, cluster, etc. (Models must execute)

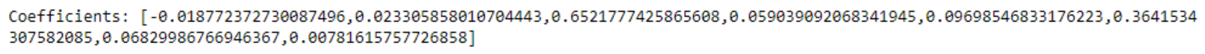
**Linear Regression:**

Model 1: (maxIter=0, regParam=0)



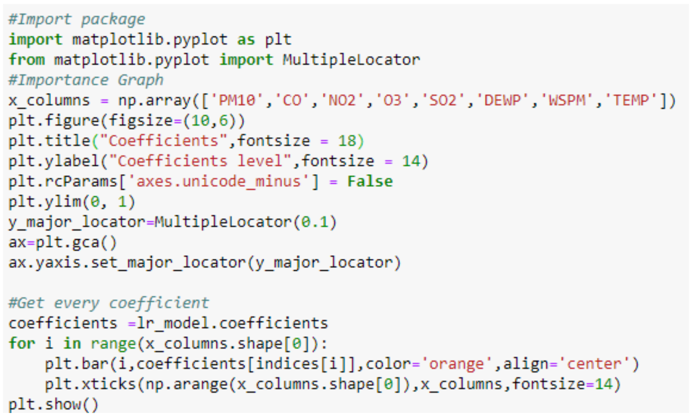
```
#Import package
from pyspark.ml.regression import LinearRegression
#Linear Regression Model
lr1 = LinearRegression(featuresCol='features', labelCol='PM2-5_Transf', maxIter=0, regParam=0)

# Fit the training data.
lr_model = lr1.fit(train_data)

# Print the coefficients.
print("Coefficients: " + str(lr_model.coefficients))
```

Figure 80: Code of Displaying Coefficients



```
Coefficients: [-0.018772372730087496,0.023305858010704443,0.6521777425865608,0.059039092068341945,0.09698546833176223,0.3641534
307582085,0.06829986766946367,0.00781615757726858]
```

Figure 81: Describe of Coefficients



```
#Import package
import matplotlib.pyplot as plt
from matplotlib.pyplot import MultipleLocator
#Importance Graph
x_columns = np.array(['PM10','CO','NO2','O3','SO2','DEWP','WSPM','TEMP'])
plt.figure(figsize=(10,6))
plt.title("Coefficients",fontsize = 18)
plt.ylabel("Coefficients level",fontsize = 14)
plt.rcParams['axes.unicode_minus'] = False
plt.ylim(0, 1)
y_major_locator=MultipleLocator(0.1)
ax=plt.gca()
ax.yaxis.set_major_locator(y_major_locator)

#Get every coefficient
coefficients =lr_model.coefficients
for i in range(x_columns.shape[0]):
    plt.bar(i,coefficients[indices[i]],color='orange',align='center')
    plt.xticks(np.arange(x_columns.shape[0]),x_columns,fontsize=14)
plt.show()
```

Figure 82: Code of Coefficients Graph

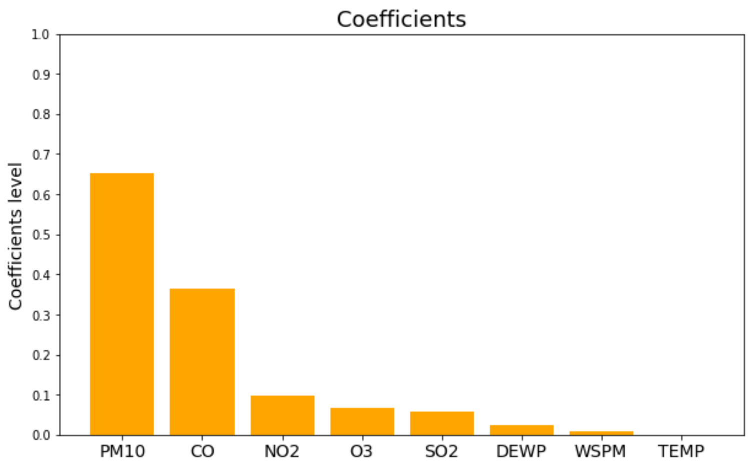
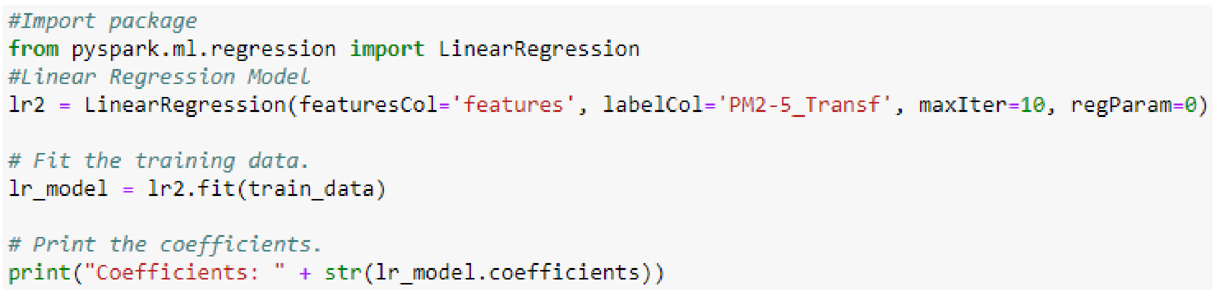


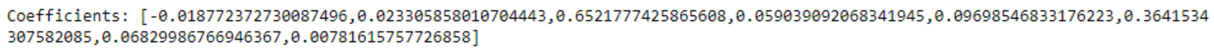
Figure 83: Coefficients Graph

Model 2: (maxIter=10, regParam=0)



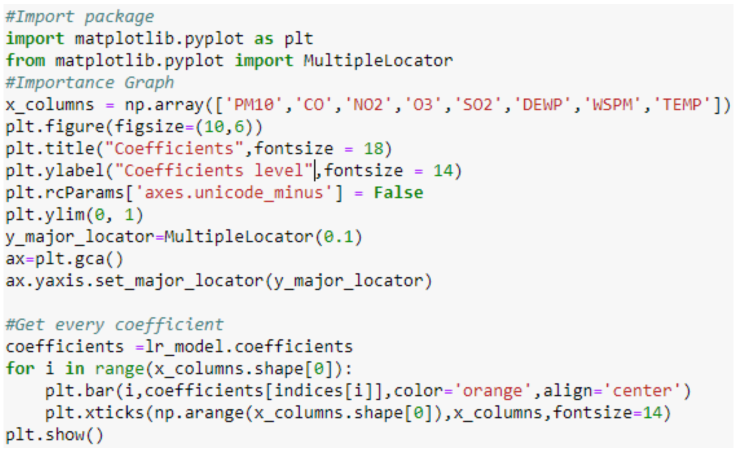
```
#Import package
from pyspark.ml.regression import LinearRegression
#Linear Regression Model
lr2 = LinearRegression(featuresCol='features', labelCol='PM2-5_Transf', maxIter=10, regParam=0)

# Fit the training data.
lr_model = lr2.fit(train_data)

# Print the coefficients.
print("Coefficients: " + str(lr_model.coefficients))
```

Figure 84: Code of Displaying Coefficients



```
Coefficients: [-0.018772372730087496,0.023305858010704443,0.6521777425865608,0.059039092068341945,0.09698546833176223,0.3641534
307582085,0.06829986766946367,0.00781615757726858]
```

Figure 85: Describe of Coefficients



```
#Import package
import matplotlib.pyplot as plt
from matplotlib.pyplot import MultipleLocator
#Importance Graph
x_columns = np.array(['PM10','CO','NO2','O3','SO2','DEWP','WSPM','TEMP'])
plt.figure(figsize=(10,6))
plt.title("Coefficients",fontsize = 18)
plt.ylabel("Coefficients level",fontsize = 14)
plt.rcParams['axes.unicode_minus'] = False
plt.ylim(0, 1)
y_major_locator=MultipleLocator(0.1)
ax=plt.gca()
ax.yaxis.set_major_locator(y_major_locator)

#Get every coefficient
coefficients =lr_model.coefficients
for i in range(x_columns.shape[0]):
    plt.bar(i,coefficients[indices[i]],color='orange',align='center')
    plt.xticks(np.arange(x_columns.shape[0]),x_columns,fontsize=14)
plt.show()
```

Figure 86: Code of Coefficients Graph

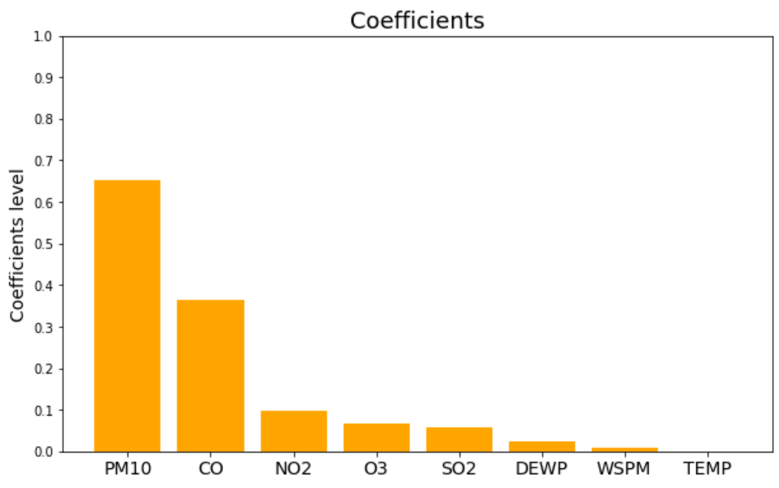
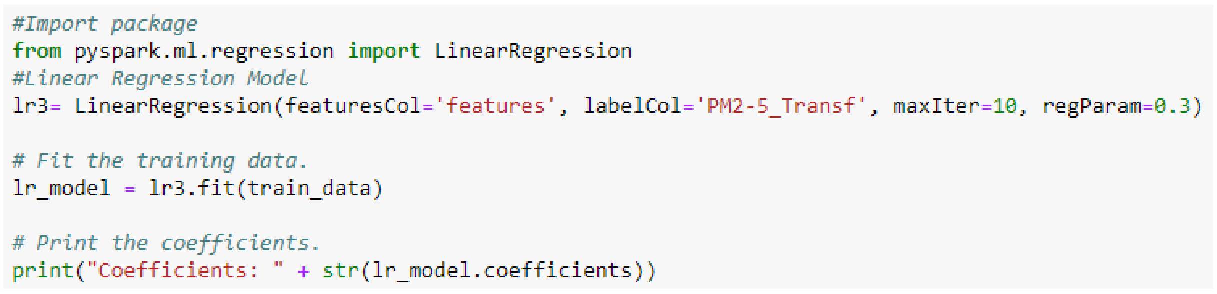


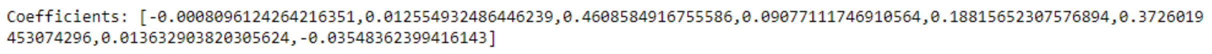
Figure 87: Coefficients Graph

Model 2: (maxIter=10, regParam=0)



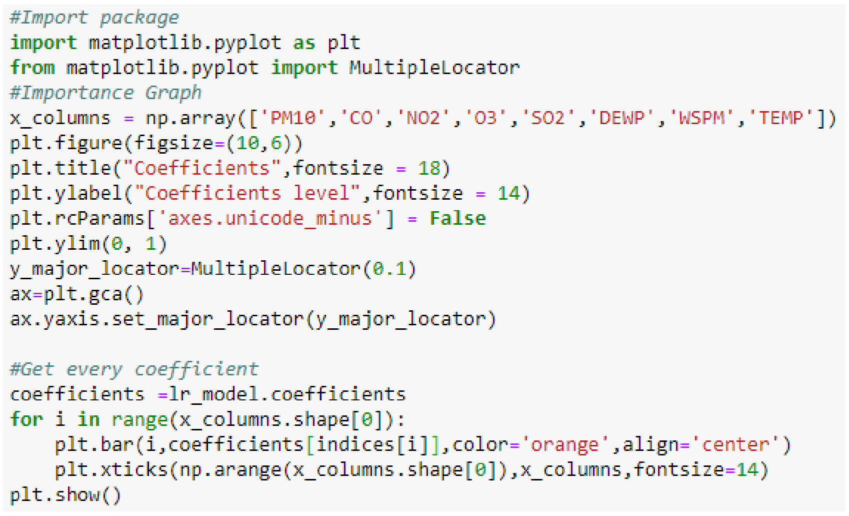
```
#Import package
from pyspark.ml.regression import LinearRegression
#Linear Regression Model
lr3= LinearRegression(featuresCol='features', labelCol='PM2-5_Transf', maxIter=10, regParam=0.3)

# Fit the training data.
lr_model = lr3.fit(train_data)

# Print the coefficients.
print("Coefficients: " + str(lr_model.coefficients))
```

Figure 88: Code of Displaying Importance



```
Coefficients: [-0.0008096124264216351,0.012554932486446239,0.4608584916755586,0.09077111746910564,0.18815652307576894,0.3726019
453074296,0.013632903820305624,-0.03548362399416143]
```

Figure 89: Describe of Importance



```
#Import package
import matplotlib.pyplot as plt
from matplotlib.pyplot import MultipleLocator
#Importance Graph
x_columns = np.array(['PM10','CO','NO2','O3','SO2','DEWP','WSPM','TEMP'])
plt.figure(figsize=(10,6))
plt.title("Coefficients",fontsize = 18)
plt.ylabel("Coefficients level",fontsize = 14)
plt.rcParams['axes.unicode_minus'] = False
plt.ylim(0, 1)
y_major_locator=MultipleLocator(0.1)
ax=plt.gca()
ax.yaxis.set_major_locator(y_major_locator)

#Get every coefficient
coefficients =lr_model.coefficients
for i in range(x_columns.shape[0]):
    plt.bar(i,coefficients[indices[i]],color='orange',align='center')
    plt.xticks(np.arange(x_columns.shape[0]),x_columns,fontsize=14)
plt.show()
```

Figure 90: Code of Importance Graph

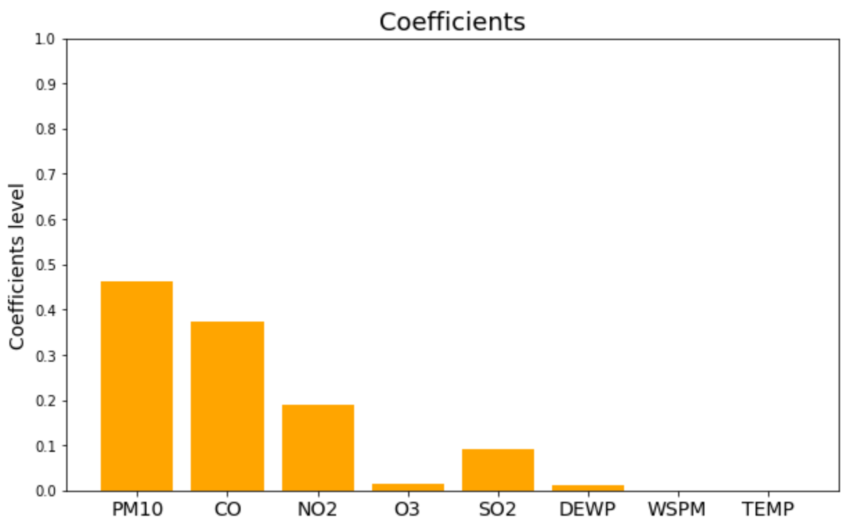
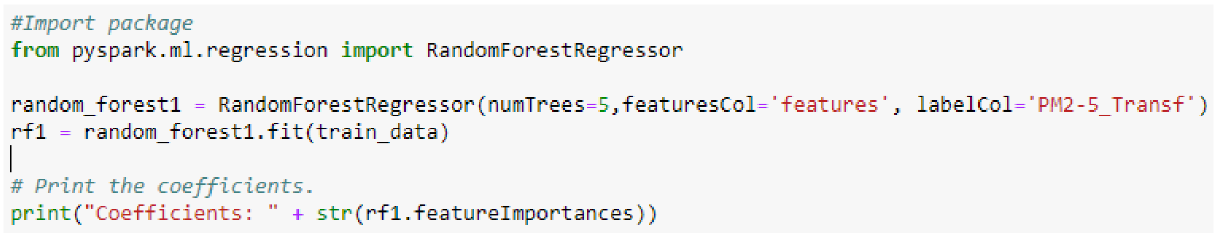


Figure 91: Importance Graph

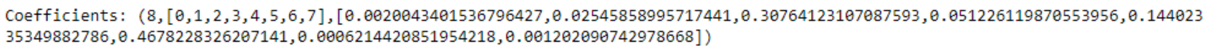
**Random Forest Regression:**

Model 1: (numTree= 5)



```
#Import package
from pyspark.ml.regression import RandomForestRegressor

random_forest1 = RandomForestRegressor(numTrees=5,featuresCol='features', labelCol='PM2-5_Transf')
rf1 = random_forest1.fit(train_data)

# Print the coefficients.
print("Coefficients: " + str(rf1.featureImportances))
```

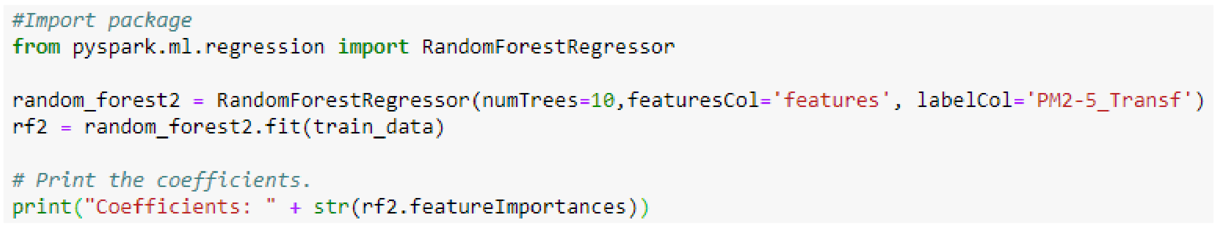
Figure 92: Code of Displaying Importance



```
Coefficients: (8,[0,1,2,3,4,5,6,7],[0.0020043401536796427,0.02545858995717441,0.30764123107087593,0.051226119870553956,0.144023
35349882786,0.4678228326207141,0.0006214420851954218,0.001202090742978668])
```

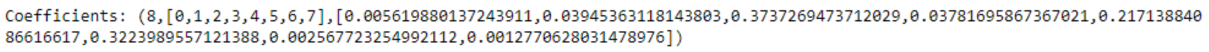
Figure 93: Describe of Importance

Model 2: (numTree= 10)



```
#Import package
from pyspark.ml.regression import RandomForestRegressor

random_forest2 = RandomForestRegressor(numTrees=10,featuresCol='features', labelCol='PM2-5_Transf')
rf2 = random_forest2.fit(train_data)

# Print the coefficients.
print("Coefficients: " + str(rf2.featureImportances))
```

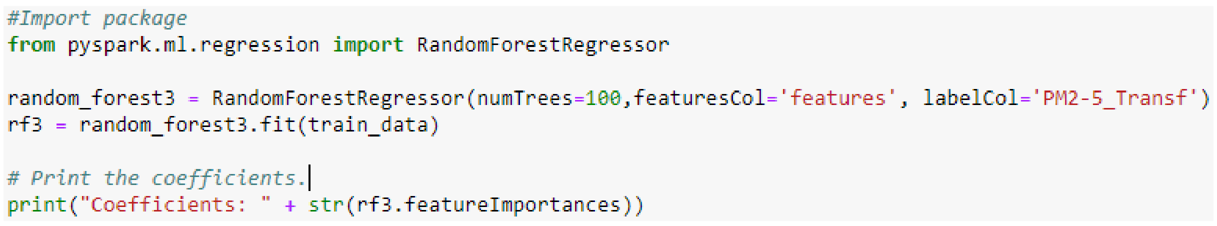
Figure 94: Code of Displaying Importance



```
Coefficients: (8,[0,1,2,3,4,5,6,7],[0.005619880137243911,0.03945363118143803,0.3737269473712029,0.03781695867367021,0.217138840
86616617,0.3223989557121388,0.002567723254992112,0.0012770628031478976])
```

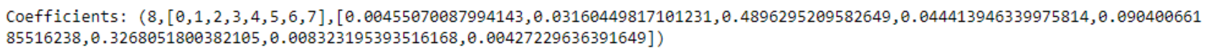
Figure 95: Describe of Importance

Model 3: (numTree= 100)



```
#Import package
from pyspark.ml.regression import RandomForestRegressor

random_forest3 = RandomForestRegressor(numTrees=100,featuresCol='features', labelCol='PM2-5_Transf')
rf3 = random_forest3.fit(train_data)

# Print the coefficients.
print("Coefficients: " + str(rf3.featureImportances))
```

Figure 96: Code of Displaying Importance

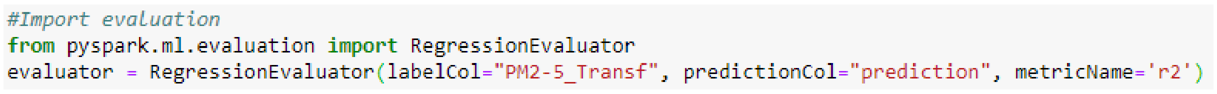


```
Coefficients: (8,[0,1,2,3,4,5,6,7],[0.00455070087994143,0.03160449817101231,0.4896295209582649,0.044413946339975814,0.090400661
85516238,0.3268051800382105,0.008323195393516168,0.00427229636391649])
```

Figure 97 Describe of Importance

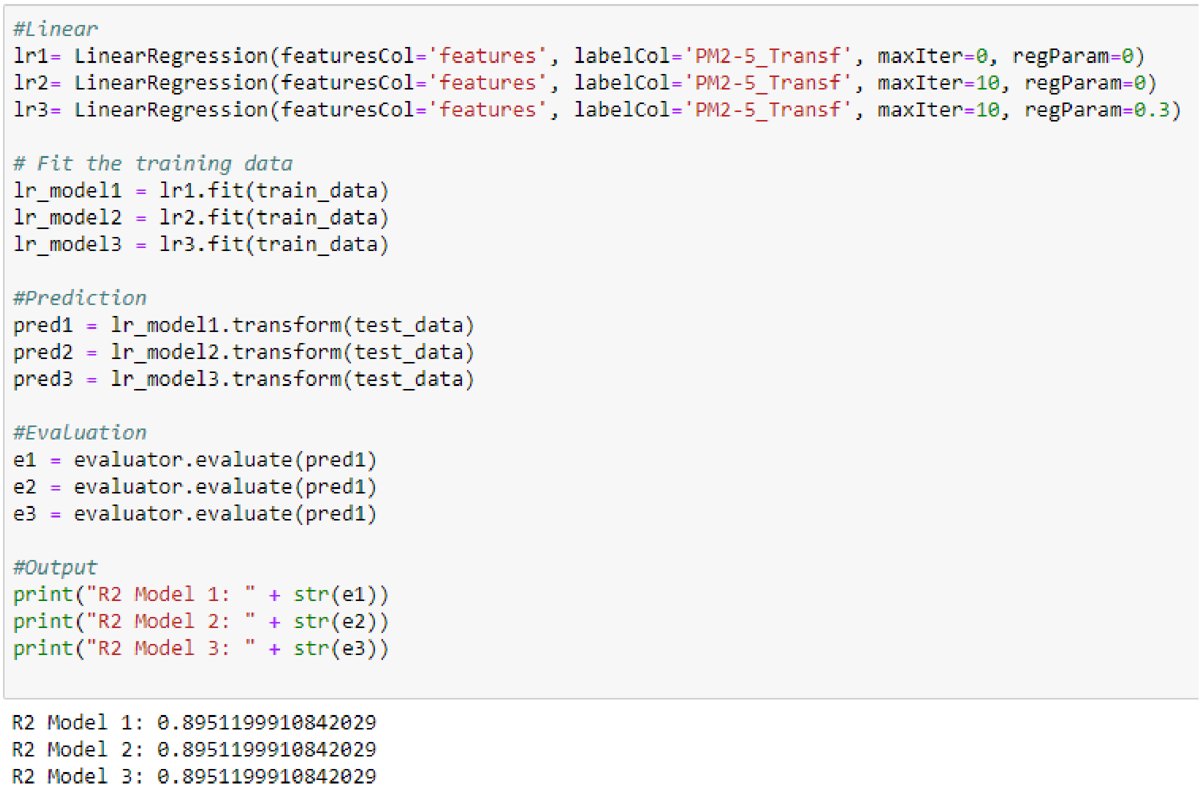
## 7.3 Search for patterns

In this step, I will choose the most appropriate pattern according to their r2. In Jupyter Notebook, I will use the value of r2 to evaluate each model. Then, I can select the most appropriate models according to r2. In this step, I user RegressionEvaluator() function to evaluate these models. I set the metricName to r2 in order to explore r2 value for each model.



```
#Import evaluation
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(labelCol="PM2-5_Transf", predictionCol="prediction", metricName='r2')
```

**Linear Regression:**

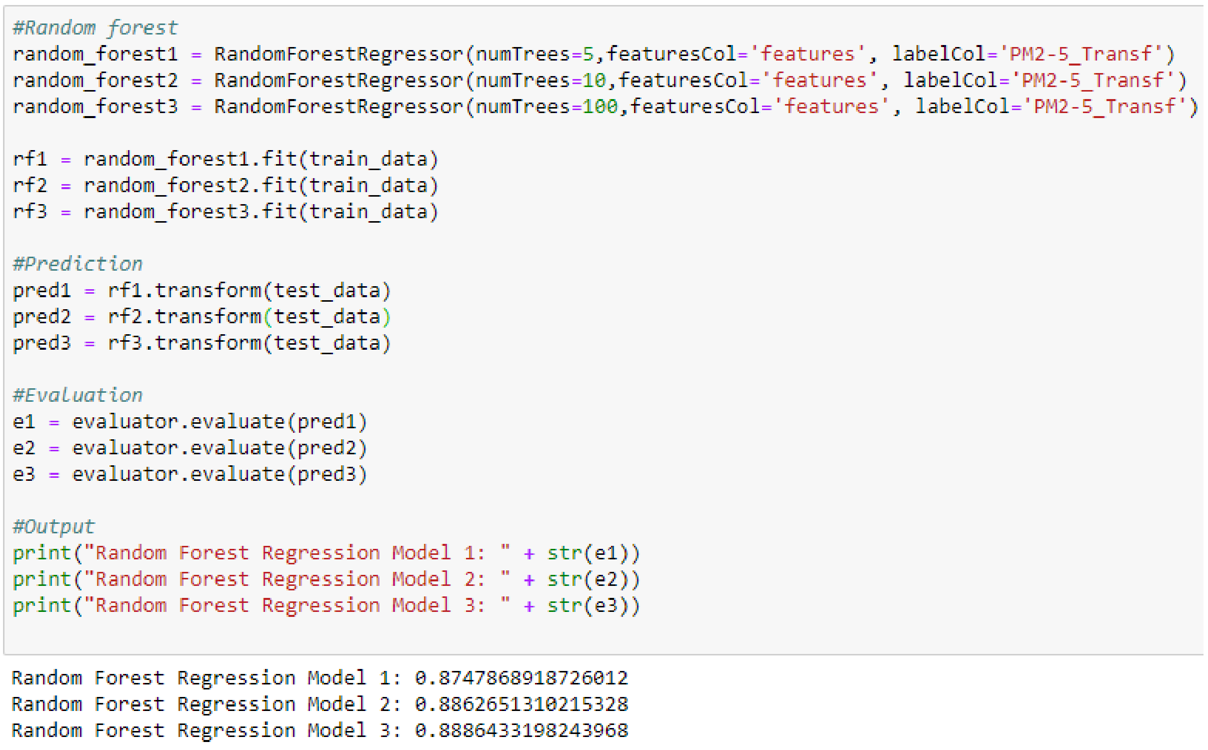


```
#Linear
lr1= LinearRegression(featuresCol='features', labelCol='PM2-5_Transf', maxIter=0, regParam=0)
lr2= LinearRegression(featuresCol='features', labelCol='PM2-5_Transf', maxIter=10, regParam=0)
lr3= LinearRegression(featuresCol='features', labelCol='PM2-5_Transf', maxIter=10, regParam=0.3)

# Fit the training data
lr_model1 = lr1.fit(train_data)
lr_model2 = lr2.fit(train_data)
lr_model3 = lr3.fit(train_data)

#Prediction
pred1 = lr_model1.transform(test_data)
pred2 = lr_model2.transform(test_data)
pred3 = lr_model3.transform(test_data)

#Evaluation
e1 = evaluator.evaluate(pred1)
e2 = evaluator.evaluate(pred1)
e3 = evaluator.evaluate(pred1)

#Output
print("R2 Model 1: " + str(e1))
print("R2 Model 2: " + str(e2))
print("R2 Model 3: " + str(e3))
```

```
R2 Model 1: 0.8951199910842029
R2 Model 2: 0.8951199910842029
R2 Model 3: 0.8951199910842029
```

Figure 98: Linear evaluation

**Random Forest Regression:**



```
#Random forest
random_forest1 = RandomForestRegressor(numTrees=5,featuresCol='features', labelCol='PM2-5_Transf')
random_forest2 = RandomForestRegressor(numTrees=10,featuresCol='features', labelCol='PM2-5_Transf')
random_forest3 = RandomForestRegressor(numTrees=100,featuresCol='features', labelCol='PM2-5_Transf')

rf1 = random_forest1.fit(train_data)
rf2 = random_forest2.fit(train_data)
rf3 = random_forest3.fit(train_data)

#Prediction
pred1 = rf1.transform(test_data)
pred2 = rf2.transform(test_data)
pred3 = rf3.transform(test_data)

#Evaluation
e1 = evaluator.evaluate(pred1)
e2 = evaluator.evaluate(pred2)
e3 = evaluator.evaluate(pred3)

#Output
print("Random Forest Regression Model 1: " + str(e1))
print("Random Forest Regression Model 2: " + str(e2))
print("Random Forest Regression Model 3: " + str(e3))
```

```
Random Forest Regression Model 1: 0.8747868918726012
Random Forest Regression Model 2: 0.8862651310215328
Random Forest Regression Model 3: 0.8886433198243968
```

Figure 99: Linear evaluation

| | Model 1 | Model 2 | Model 3 |
|---|---|---|---|
| Linear Regression | 89.51% | 89.51% | 89.51% |
| Random Forest Regression | 87.47% | 88.62% | 88.86% |

In conclusion, the model 3 of random forest regression has a high score, so I will choose this pattern as my data-mining model. For the linear regression, I observe that all models have the same score, so I choose model 1 as my selected model. In this step, I choose the most appropriate model for each algorithm. I will analyse the whole code and results in section 8. In addition, I conclude that using a big number of numTree will get a higher score.

# 8 Interpretation

## 8.1 Study and discuss the mined patterns

Since my data-mining goal is focus on relationships and predictions, so I will use regression algorithms to conduct my data-mining. In section 5, I choose linear regression random forest regression as my data-mining models. I can compare results of two models in order to get accurate prediction. In the previous selection, I select one model for each data-mining algorithm. In this step, I will analysis the result of two model. Figure 101 shows the importance graph of random forest regression. I can observe that there are two main factors that influence the value of PM2.5, they are PM10, and CO. Figure 100 shows the importance graph of linear regression. The PM10 and CO are still main factors that affect the value of PM2.5. I will visualize the data, results, models and patterns in next step.

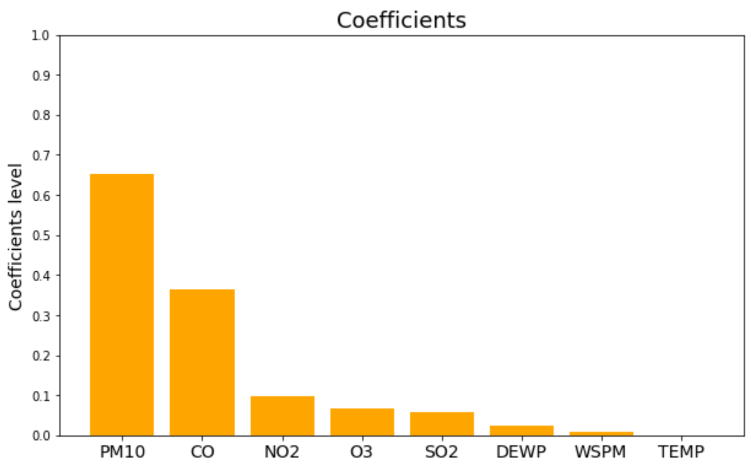
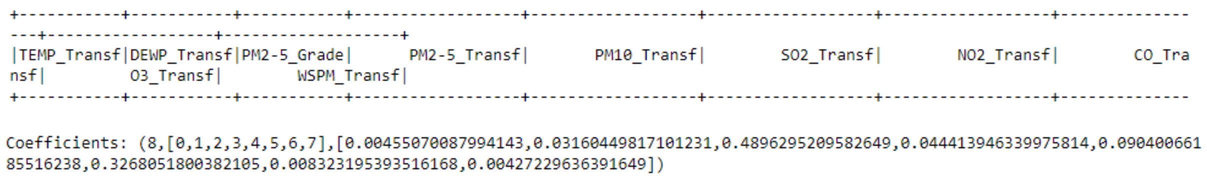


Figure 100: Linear Regression

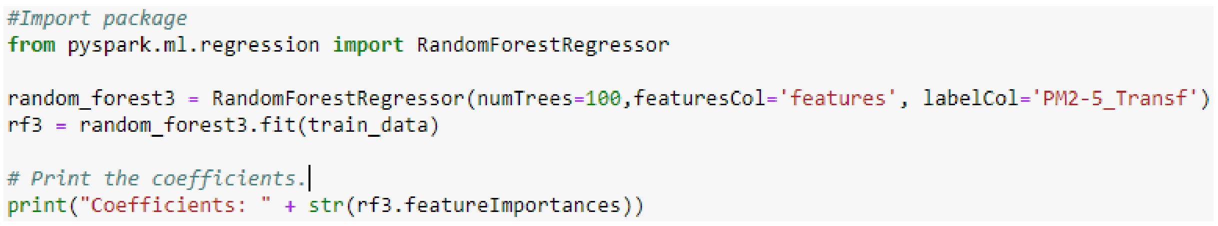


```
+-----------+----------+----------+-----------------+-----------------+-----------------+-----------------+-------------
---+-----------------+-----------------+
|TEMP_Transf|DEWP_Transf|PM2-5_Grade|      PM2-5_Transf|        PM10_Transf|         SO2_Transf|         NO2_Transf|        CO_Tra
nsf|          O3_Transf|        WSPM_Transf|
+-----------+----------+----------+-----------------+-----------------+-----------------+-----------------+-------------

Coefficients: (8,[0,1,2,3,4,5,6,7],[0.00455070087994143,0.03160449817101231,0.4896295209582649,0.044413946339975814,0.090400661
85516238,0.3268051800382105,0.008323195393516168,0.00427229636391649])
```

Figure 101: Linear Regression

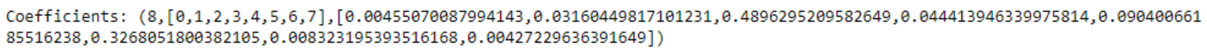
## 8.2 Visualize the data, results, models, and patterns

**Random Forest Regression:**



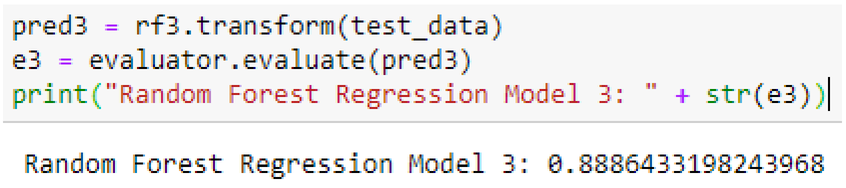
```
#Import package
from pyspark.ml.regression import RandomForestRegressor

random_forest3 = RandomForestRegressor(numTrees=100,featuresCol='features', labelCol='PM2-5_Transf')
rf3 = random_forest3.fit(train_data)

# Print the coefficients.
print("Coefficients: " + str(rf3.featureImportances))
```

Figure 102: Code of Displaying Importance



```
Coefficients: (8,[0,1,2,3,4,5,6,7],[0.00455070087994143,0.03160449817101231,0.4896295209582649,0.044413946339975814,0.090400661
85516238,0.3268051800382105,0.008323195393516168,0.00427229636391649])
```

Figure 103 Describe of Importance



```
pred3 = rf3.transform(test_data)
e3 = evaluator.evaluate(pred3)
print("Random Forest Regression Model 3: " + str(e3))
```

```
Random Forest Regression Model 3: 0.8886433198243968
```

Figure 104: Score

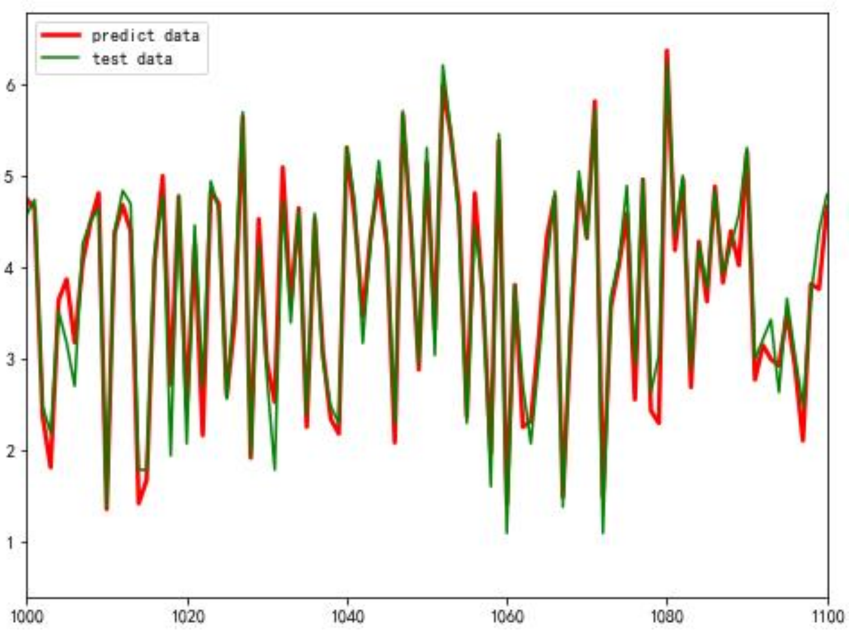


Figure 105: Predicted Data vs Tested data (X range from 1000 to 1100)

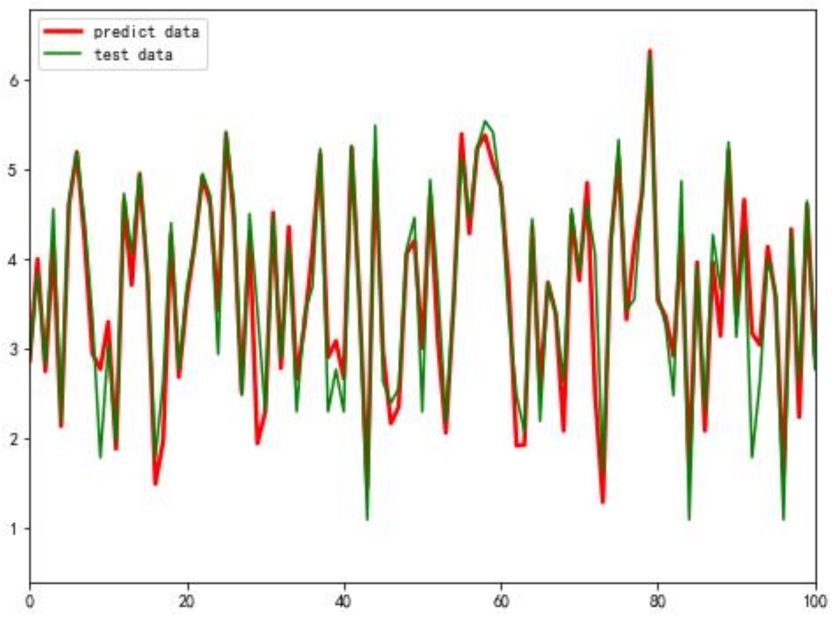


Figure 106: Predicted Data vs Tested data (X range from 0 to 100)

**Random Forest Regression:**

```
#Import package
from pyspark.ml.regression import LinearRegression
#Linear Regression Model
lr1 = LinearRegression(featuresCol='features', labelCol='PM2-5_Transf', maxIter=0, regParam=0)

# Fit the training data.
lr_model = lr1.fit(train_data)

# Print the coefficients.
print("Coefficients: " + str(lr_model.coefficients))
```

Figure 107: Code of Displaying Coefficients

```
Coefficients: [-0.018772372730087496,0.023305858010704443,0.6521777425865608,0.059039092068341945,0.09698546833176223,0.3641534
307582085,0.06829986766946367,0.00781615757726858]
```

Figure 108: Describe of Coefficients

```
#Import package
import matplotlib.pyplot as plt
from matplotlib.pyplot import MultipleLocator
#Importance Graph
x_columns = np.array(['PM10','CO','NO2','O3','SO2','DEWP','WSPM','TEMP'])
plt.figure(figsize=(10,6))
plt.title("Coefficients",fontsize = 18)
plt.ylabel("Coefficients level",fontsize = 14)
plt.rcParams['axes.unicode_minus'] = False
plt.ylim(0, 1)
y_major_locator=MultipleLocator(0.1)
ax=plt.gca()
ax.yaxis.set_major_locator(y_major_locator)

#Get every coefficient
coefficients =lr_model.coefficients
for i in range(x_columns.shape[0]):
    plt.bar(i,coefficients[indices[i]],color='orange',align='center')
    plt.xticks(np.arange(x_columns.shape[0]),x_columns,fontsize=14)
plt.show()
```

Figure 109: Code of Coefficients Graph

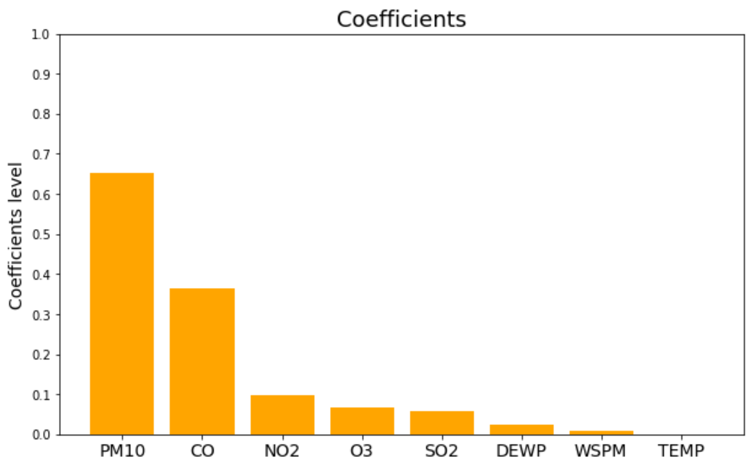


Figure 110: Coefficients Graph

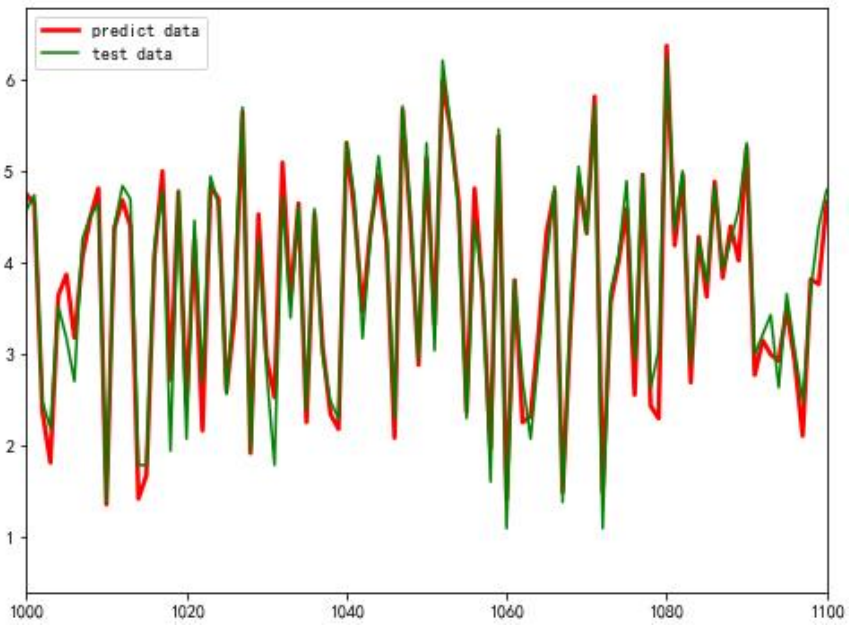


Figure 111: Predicted Data vs Tested data (X range from 1000 to 1100)

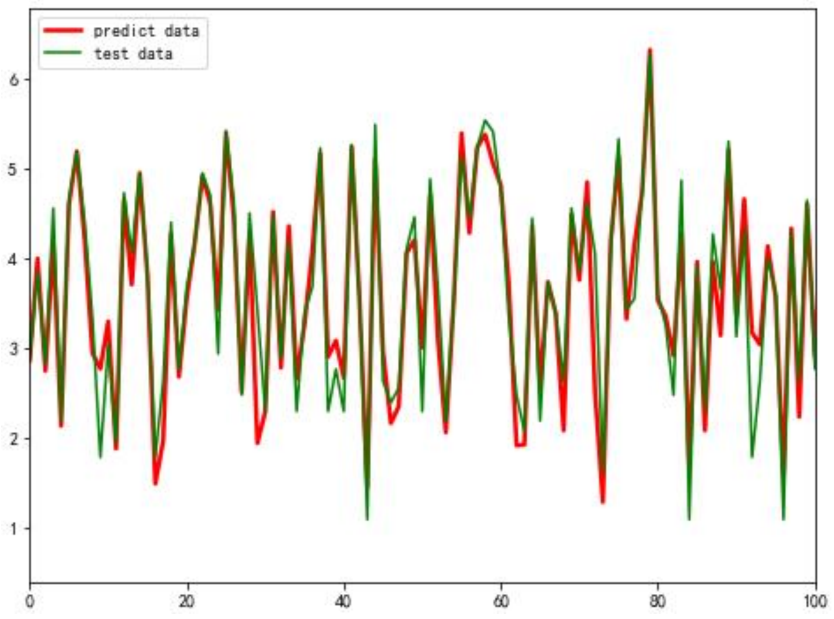
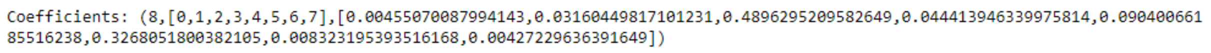


Figure 112: Predicted Data vs Tested data (X range from 0 to 100)

## 8.3 Interpret the result, models

**Random Forest Regression:**

Figure 113 shows the importance graph of random forest regression. I can clearly observe the PM10 and CO are the most significant feature that influence the value of PM2.5. Then, I can tell PM10 becomes main factor which have a huge influence to PM2.5, and secondary factor is CO. DEWP only have a small influence to the amount of PM2.5. For the rest of features, I will not consider. Therefore, my assumption is true. There is a relationship between PM2.5 and PM10, CO. There is also a weak relationship between PM2.5 and DEWP.



```
Coefficients: (8,[0,1,2,3,4,5,6,7],[0.00455070087994143,0.03160449817101231,0.4896295209582649,0.044413946339975814,0.090400661
85516238,0.3268051800382105,0.008323195393516168,0.00427229636391649])
```

Figure 113 Describe of Importance

**Linear Regression:**

Figure 114 shows the importance graph of linear regression. I can clearly observe the PM10 and CO are the most significant feature that influence the value of PM2.5. Then, I can tell PM10 becomes main factor which have a huge influence to PM2.5, and secondary factor is CO. In addition, NO2 is the third factor that influence the value of PM2.5. For the rest of features, I will not consider. Therefore, my assumption is true. There is a relationship between PM2.5 and PM10, CO. There is also are relationship between NO2 and PM2.5.

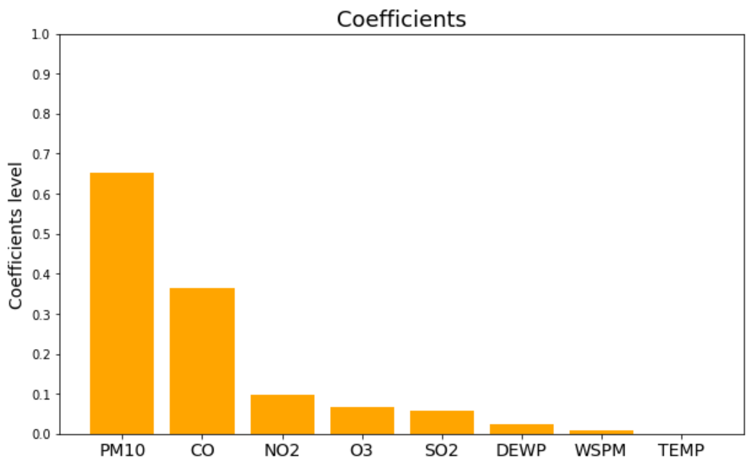


Figure 114: Coefficients Graph

In conclusion, these are very effective and useful information for the environment department. The environment department need to focus on the PM10 and CO, they do not need to pay more attention about the influence of SO2, O3, etc. Figure 115 shows the relationship between PM2.5 and PM10. It is obvious that the value of PM2.5 will rapidly increase with the increase of PM10. In addition, Figure 116 shows the trends between PM2.5 and CO. According to the Figure 116, I can tell PM2.5 will have a high value when CO has a high value. If they want to decrease the amount of PM2.5, they should find some approaches to decrease the value of PM10 and CO. For example, government can encourage people take public transports to travel. This is an effective way to decrease the emission of CO. In addition, I think the value of PM2.5 which measured in summer is lower than the value that measured in winter. Therefore, government need pay more attention to decrease the value of PM2.5 in winter. All in all, if government release the emission of PM10 and CO, the air-quality will become better. In this report I use two models, they are linear regression and random tree regression. Linear regression is a powerful model, it can help people figure out the relationships between independent variable and dependent variable. This model is a good choice, because the measurement of predicate fields is continuous. The random tree regression is also an effective algorithm to deal with prediction problems.

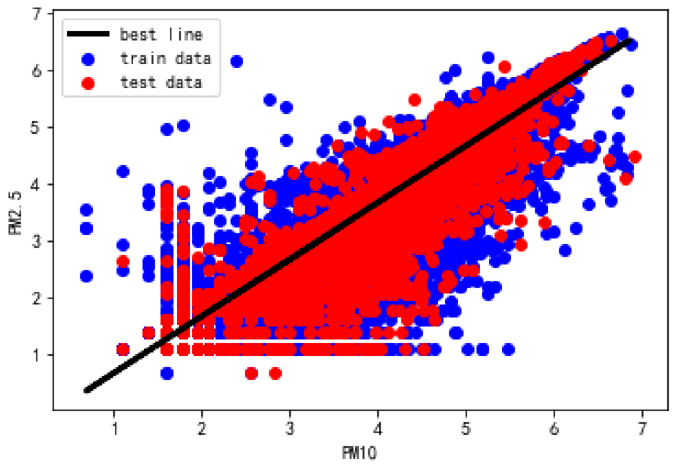


Figure 115: PM10

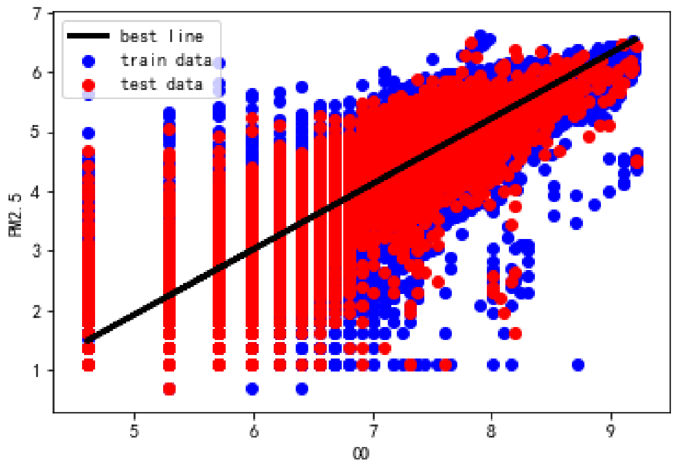


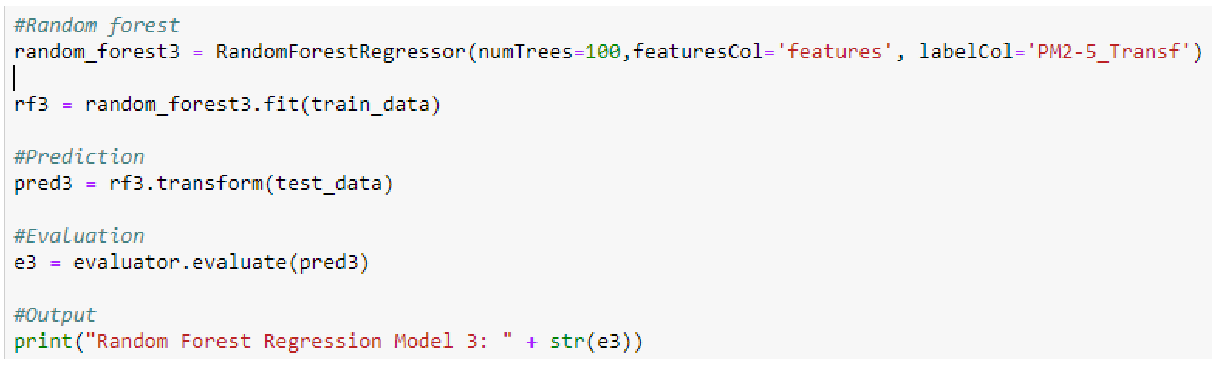
Figure 116: CO

## 8.4 Assess and evaluate results, models and patterns

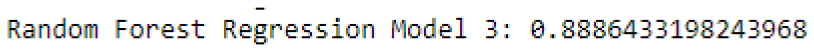
```
#Linear
lr1= LinearRegression(featuresCol='features', labelCol='PM2-5_Transf', maxIter=0, regParam=0)

# Fit the training data
lr_model1 = lr1.fit(train_data)

#Prediction
pred1 = lr_model1.transform(test_data)

#Evaluation
e1 = evaluator.evaluate(pred1)

#Output
print("R2 Model 1: " + str(e1))
```

```
R2 Model 1: 0.8877997113167807
```

Figure 117: Linear Regression



```
#Random forest
random_forest3 = RandomForestRegressor(numTrees=100,featuresCol='features', labelCol='PM2-5_Transf')

rf3 = random_forest3.fit(train_data)

#Prediction
pred3 = rf3.transform(test_data)

#Evaluation
e3 = evaluator.evaluate(pred3)

#Output
print("Random Forest Regression Model 3: " + str(e3))
```



```
Random Forest Regression Model 3: 0.8886433198243968
```

Figure 118: Random Forest Regression

In this step, I use four statistical measures to evaluate these two regression models. I use R square to evaluate my data-mining model. R square is a statistical measure of how close the data that fit regression line. The value of R square in random forest regression is 0.887, and this value in linear regression is 0.888. According to the definition, a R square value that more than 0.7 will consider a strong effect size. Therefore, these two models are appropriate and reliable.

In conclusion, I believe that Linear regression model and random forest regression are appropriate approaches to achieve my data mining goal. The government is able to use these models to predict the air-quality in the future. Additionally, the government also can figure out the main factor which increase the amount of PM2.5. Then, they may make appropriate rules or solutions in order to release air pollution.

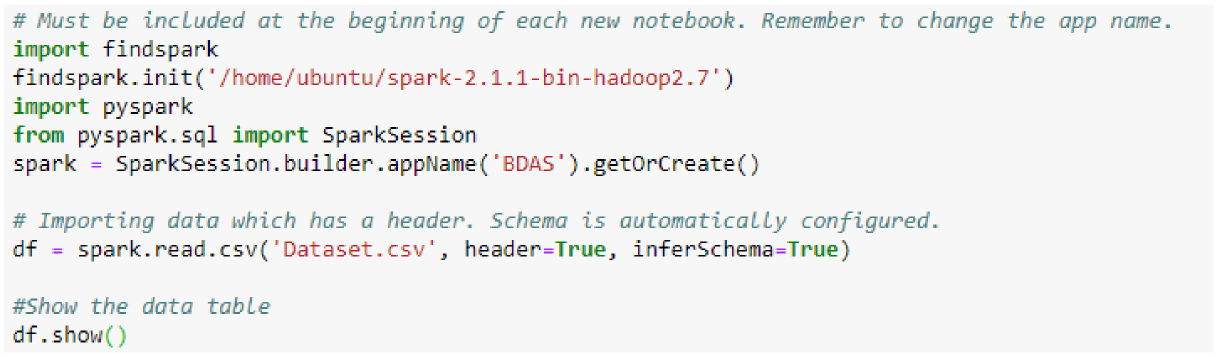
## 8.5 Iterate prior steps (1-7) as required

**Business Understanding:**

In this step, the goal is finding an interesting topic from the 17 Sustainable development goals of the UN. I did some research. I find that I'm interested in environment, so I choose ensuring sustainable consumption and production patterns as my data-mining topic. First of all, I did some research about air pollution, then I discuss some problems that may influence air quality. Then, I construct business objectives and business goals. In addition, I construct some standards of data mining success. I also make some assumptions, and I will discuss them in the following section. Finally, I produce a project plan, which can help me reasonably manage tasks.

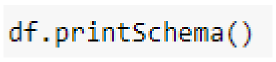
**Data Understanding:**

In this step, I collect the dataset from Kaggle which is an open-source platform for data-mining scientists. This website allows people to publish and download datasets in order to build and explore models for data mining. I downloaded a dataset about air-quality in Shunyi. After getting the dataset, I need to explore the data. Firstly, I will look though the whole dataset. Then, I will decide the target variable and relative features that may influence the value of target. I can also check the format for each feature. In addition, I can also check whether it has null value by using isnull() function. Furthermore, I use boxplot to explore outliers and extremes. Then, I will deal with them in next section.



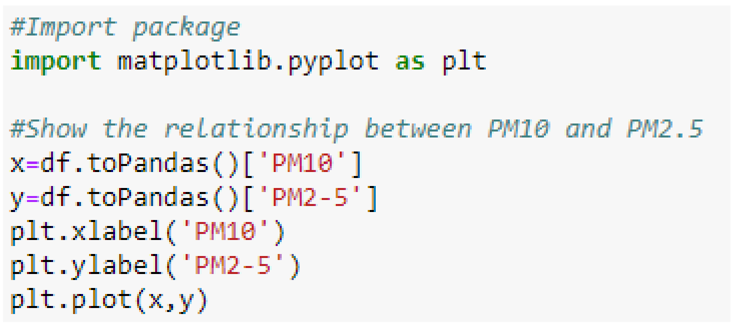
```
# Must be included at the beginning of each new notebook. Remember to change the app name.
import findspark
findspark.init('/home/ubuntu/spark-2.1.1-bin-hadoop2.7')
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('BDAS').getOrCreate()

# Importing data which has a header. Schema is automatically configured.
df = spark.read.csv('Dataset.csv', header=True, inferSchema=True)

#Show the data table
df.show()
```

Figure 119: Explore data (1)



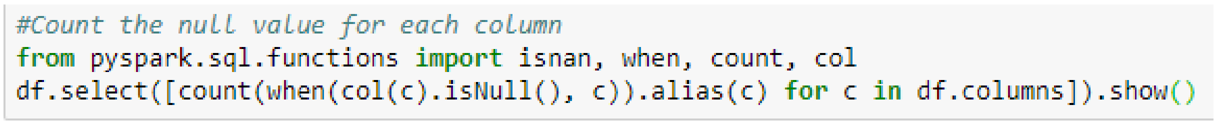
```
df.printSchema()
```

Figure 120: Explore data (2)



```
#Import package
import matplotlib.pyplot as plt

#Show the relationship between PM10 and PM2.5
x=df.toPandas()['PM10']
y=df.toPandas()['PM2-5']
plt.xlabel('PM10')
plt.ylabel('PM2-5')
plt.plot(x,y)
```

Figure 121: Explore data (3)

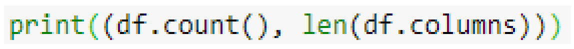


```
#Count the null value for each column
from pyspark.sql.functions import isnan, when, count, col
df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns]).show()
```

Figure 122: Explore data (4)

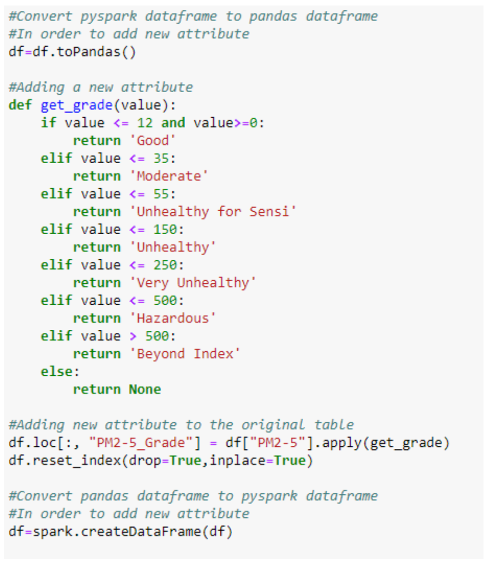
**Data Preparation:**

In this section, I will do the process of data preparation. First, I will import the file and display all the attributes. Then, I will draw a heatmap according to correlation of each attributes. When I finish to construct heatmap, I can pick up the attributes that have high correlation. Then, I will drop irrelative attributes in order to reduce data. In addition, I also add an extra attribute, because I think the new attribute will make the data clearer and more readable. Furthermore, I will remove all rows that have null value in order to make an accurate prediction. If an attribute has many null records, I would drop the whole attributes. In this step, I will also consider outliers and extremes. However, I find some attributes have outliers and extremes, but they all in a reasonable data range. If I remove outlier and extremes, results of prediction will be influenced. Therefore, I decided to keep these outlier and extremes. Secondly, I change the data types. For example, all the float attributes use float64 as their data type. Float64 will occupy more memory and it will also slow the program, so I change the float64 to float32. Finally, I try to integrate various data sources.



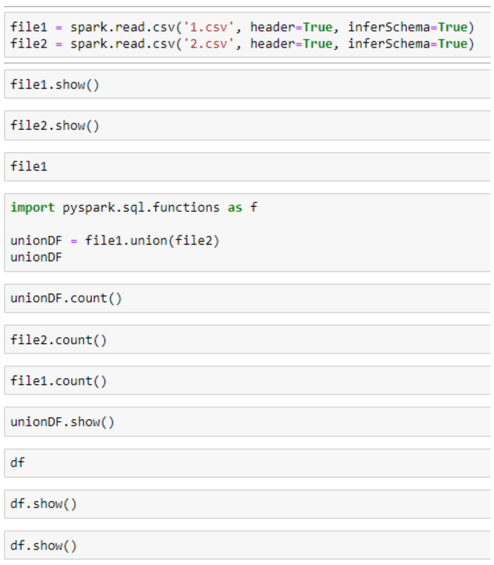
```
print((df.count(), len(df.columns)))
```

Figure 123: Data Preparation (1)



```
#Convert pyspark dataframe to pandas dataframe
#In order to add new attribute
df=df.toPandas()

#Adding a new attribute
def get_grade(value):
    if value <= 12 and value>=0:
        return 'Good'
    elif value <= 35:
        return 'Moderate'
    elif value <= 55:
        return 'Unhealthy for Sensi'
    elif value <= 150:
        return 'Unhealthy'
    elif value <= 250:
        return 'Very Unhealthy'
    elif value <= 500:
        return 'Hazardous'
    elif value > 500:
        return 'Beyond Index'
    else:
        return None

#Adding new attribute to the original table
df.loc[:, "PM2-5_Grade"] = df["PM2-5"].apply(get_grade)
df.reset_index(drop=True,inplace=True)

#Convert pandas dataframe to pyspark dataframe
#In order to add new attribute
df=spark.createDataFrame(df)
```

Figure 123: Data Preparation (2)



```
file1 = spark.read.csv('1.csv', header=True, inferSchema=True)
file2 = spark.read.csv('2.csv', header=True, inferSchema=True)
```

```
file1.show()
```

```
file2.show()
```

```
file1
```

```
import pyspark.sql.functions as f

unionDF = file1.union(file2)
unionDF
```

```
unionDF.count()
```

```
file2.count()
```

```
file1.count()
```

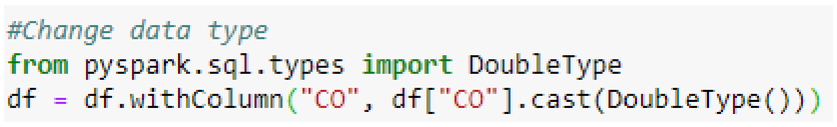
```
unionDF.show()
```

```
df
```

```
df.show()
```

```
df.show()
```

Figure 124: Data Preparation (3)

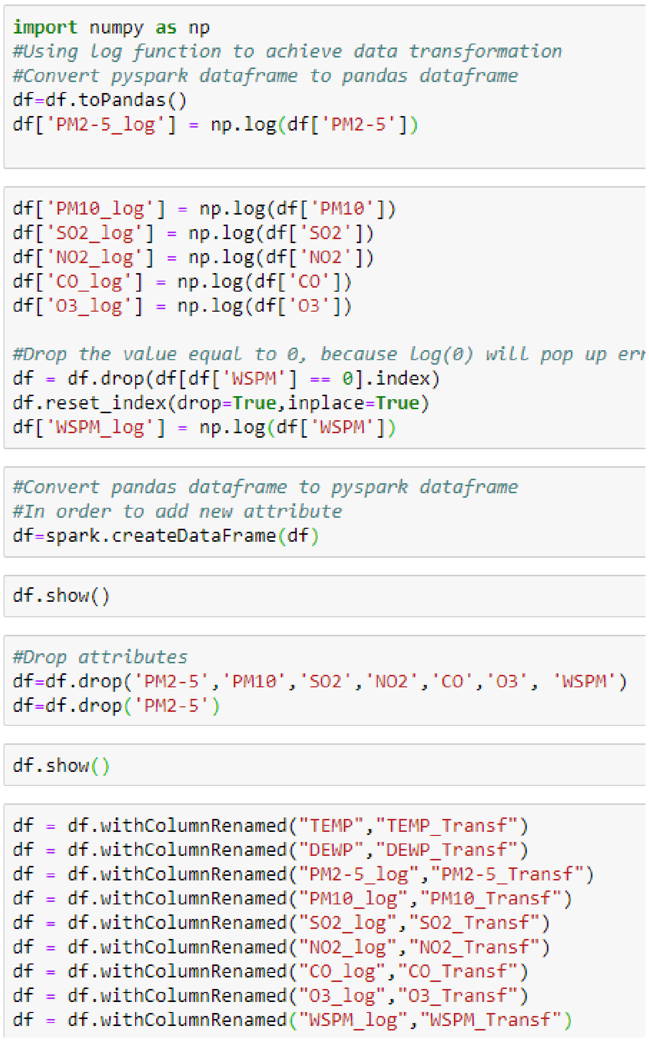


```
#Change data type
from pyspark.sql.types import DoubleType
df = df.withColumn("CO", df["CO"].cast(DoubleType()))
```

Figure 124: Data Preparation (4)

**Data Transformation:**

In this step, I use distplot() function to construct a distribution graph for each attributes. Then, I find that the majority of attributes do not have normal distribution. Therefore, I use log() function to achieve data transformation, which will help me to receive an accurate result.



```
import numpy as np
#Using log function to achieve data transformation
#Convert pyspark dataframe to pandas dataframe
df=df.toPandas()
df['PM2-5_log'] = np.log(df['PM2-5'])
```

```
df['PM10_log'] = np.log(df['PM10'])
df['SO2_log'] = np.log(df['SO2'])
df['NO2_log'] = np.log(df['NO2'])
df['CO_log'] = np.log(df['CO'])
df['O3_log'] = np.log(df['O3'])

#Drop the value equal to 0, because log(0) will pop up err
df = df.drop(df[df['WSPM'] == 0].index)
df.reset_index(drop=True,inplace=True)
df['WSPM_log'] = np.log(df['WSPM'])
```

```
#Convert pandas dataframe to pyspark dataframe
#In order to add new attribute
df=spark.createDataFrame(df)
```

```
df.show()
```

```
#Drop attributes
df=df.drop('PM2-5','PM10','SO2','NO2','CO','O3', 'WSPM')
df=df.drop('PM2-5')
```

```
df.show()
```

```
df = df.withColumnRenamed("TEMP","TEMP_Transf")
df = df.withColumnRenamed("DEWP","DEWP_Transf")
df = df.withColumnRenamed("PM2-5_log","PM2-5_Transf")
df = df.withColumnRenamed("PM10_log","PM10_Transf")
df = df.withColumnRenamed("SO2_log","SO2_Transf")
df = df.withColumnRenamed("NO2_log","NO2_Transf")
df = df.withColumnRenamed("CO_log","CO_Transf")
df = df.withColumnRenamed("O3_log","O3_Transf")
df = df.withColumnRenamed("WSPM_log","WSPM_Transf")
```

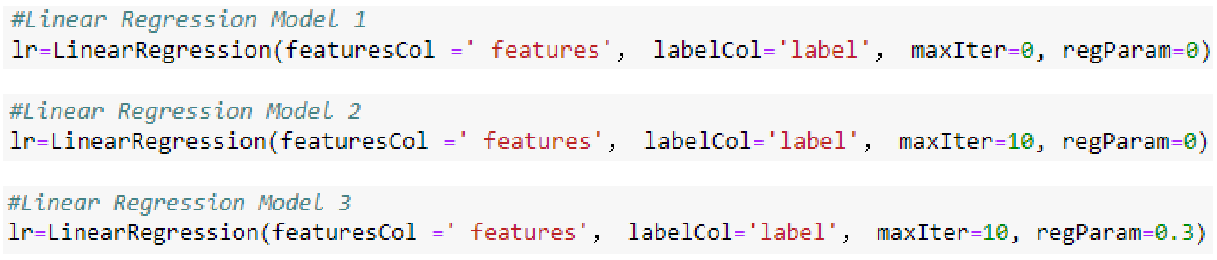
Figure 125: Data Transformation

**Data Mining Methods Selection:**

In this section, I will choose regression as my data-mining methods. I want to predict air-quality in the future, so regression is an appropriate selection. In addition, I also want to find the relationships between PM2.5 and others feature. According to the definition of regression, I decide to select regression as my data-mining methods.

**Data Mining Algorithm Selection:**

In this section, I select two data-mining algorithms, they are random forest regression and linear regression. Then, I construct three models for each algorithm in order to pick up the best models. These three models have different parameter, so I can pick up a model with high score.

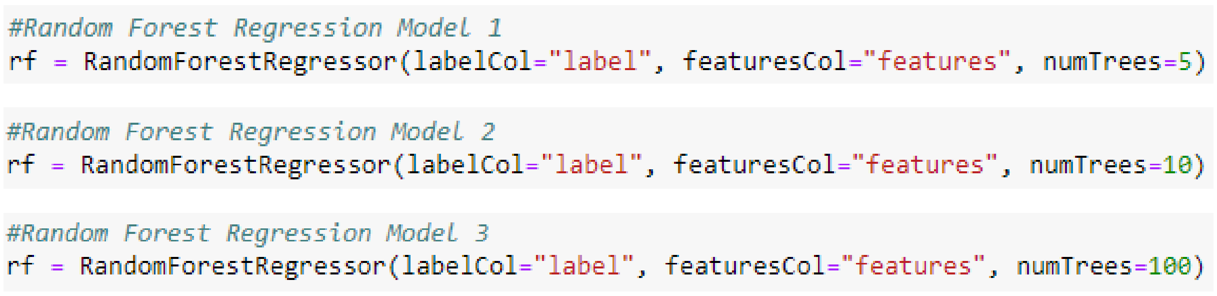


```
#Linear Regression Model 1
lr=LinearRegression(featuresCol =' features',  labelCol='label',  maxIter=0, regParam=0)
```

```
#Linear Regression Model 2
lr=LinearRegression(featuresCol =' features',  labelCol='label',  maxIter=10, regParam=0)
```

```
#Linear Regression Model 3
lr=LinearRegression(featuresCol =' features',  labelCol='label',  maxIter=10, regParam=0.3)
```

Figure 126: Linear Regression Models



```
#Random Forest Regression Model 1
rf = RandomForestRegressor(labelCol="label", featuresCol="features", numTrees=5)
```

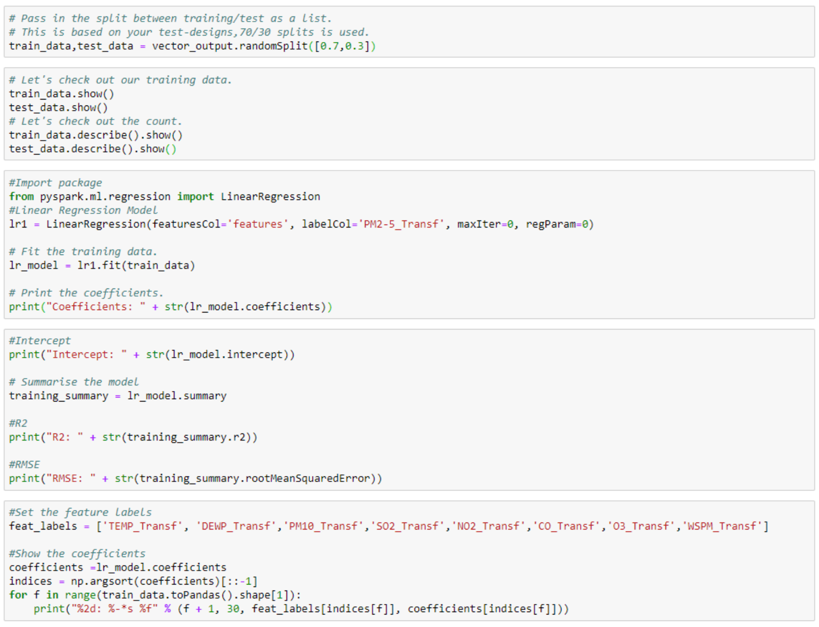
```
#Random Forest Regression Model 2
rf = RandomForestRegressor(labelCol="label", featuresCol="features", numTrees=10)
```

```
#Random Forest Regression Model 3
rf = RandomForestRegressor(labelCol="label", featuresCol="features", numTrees=100)
```

Figure 127: Random Forest Regression Models

**Data mining**:

In previous section, I successfully select data-mining algorithms and data-mining models . In this section, I will do the process of data-ming. First of all, I devide data into twp sets, they are training set and testing set. Then, I will use python to achieve forest random regression and linear regression. I will also use python to shoe the trends graphs, I can observe the relationship between the target variable and other features. The last step is evaluation, I will use statistical measures to evaluate these model.



```
# Pass in the split between training/test as a list.
# This is based on your test-designs,70/30 splits is used.
train_data,test_data = vector_output.randomSplit([0.7,0.3])
```

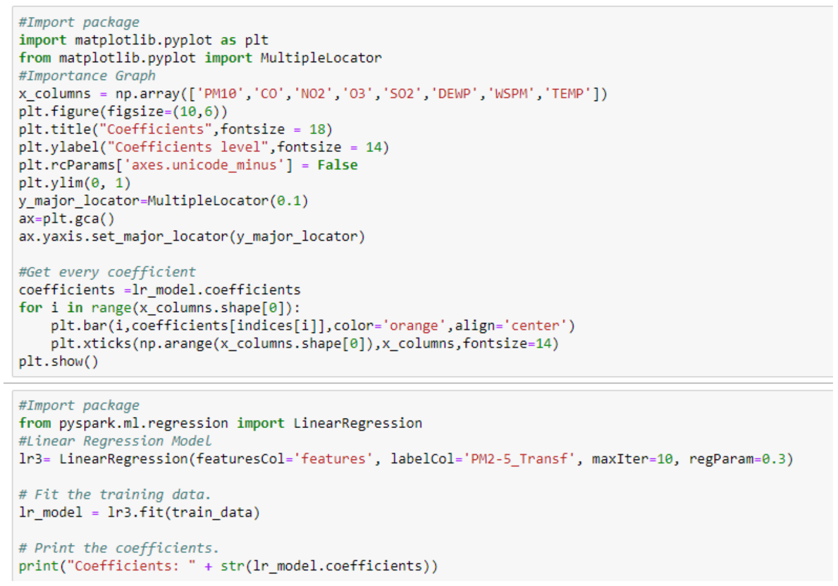
```
# Let's check out our training data.
train_data.show()
test_data.show()
# Let's check out the count.
train_data.describe().show()
test_data.describe().show()
```

```
#Import package
from pyspark.ml.regression import LinearRegression
#Linear Regression Model
lr1 = LinearRegression(featuresCol='features', labelCol='PM2-5_Transf', maxIter=0, regParam=0)

# Fit the training data.
lr_model = lr1.fit(train_data)

# Print the coefficients.
print("Coefficients: " + str(lr_model.coefficients))
```

```
#Intercept
print("Intercept: " + str(lr_model.intercept))

# Summarise the model
training_summary = lr_model.summary

#R2
print("R2: " + str(training_summary.r2))

#RMSE
print("RMSE: " + str(training_summary.rootMeanSquaredError))
```

```
#Set the feature labels
feat_labels = ['TEMP_Transf', 'DEWP_Transf','PM10_Transf','SO2_Transf','NO2_Transf','CO_Transf','O3_Transf','WSPM_Transf']

#Show the coefficients
coefficients =lr_model.coefficients
indices = np.argsort(coefficients)[::-1]
for f in range(train_data.toPandas().shape[1]):
    print("%2d: %-*s %f" % (f + 1, 30, feat_labels[indices[f]], coefficients[indices[f]]))
```

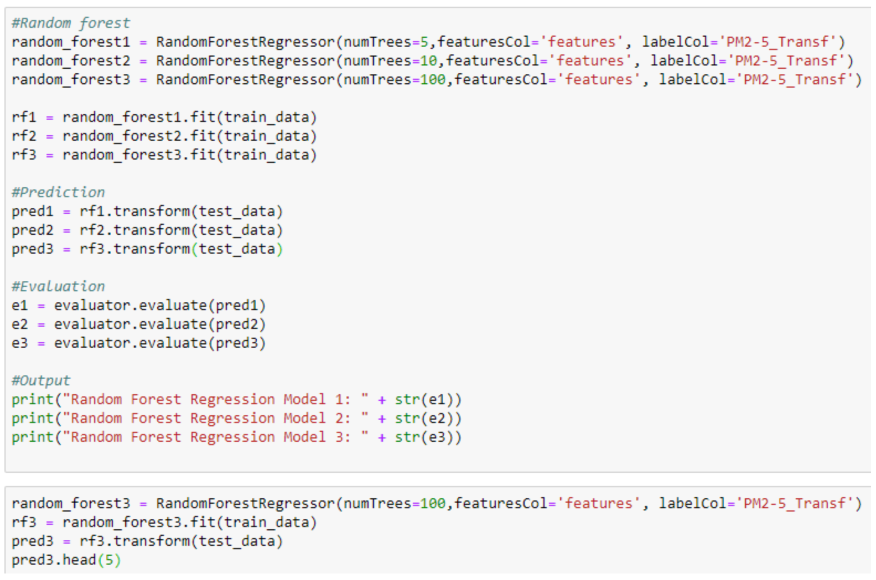
Figure 128: Data Mining (1)



```
#Import package
import matplotlib.pyplot as plt
from matplotlib.pyplot import MultipleLocator
#Importance Graph
x_columns = np.array(['PM10','CO','NO2','O3','SO2','DEWP','WSPM','TEMP'])
plt.figure(figsize=(10,6))
plt.title("Coefficients",fontsize = 18)
plt.ylabel("Coefficients level",fontsize = 14)
plt.rcParams['axes.unicode_minus'] = False
plt.ylim(0, 1)
y_major_locator=MultipleLocator(0.1)
ax=plt.gca()
ax.yaxis.set_major_locator(y_major_locator)

#Get every coefficient
coefficients =lr_model.coefficients
for i in range(x_columns.shape[0]):
    plt.bar(i,coefficients[indices[i]],color='orange',align='center')
    plt.xticks(np.arange(x_columns.shape[0]),x_columns,fontsize=14)
plt.show()
```

```
#Import package
from pyspark.ml.regression import LinearRegression
#Linear Regression Model
lr3= LinearRegression(featuresCol='features', labelCol='PM2-5_Transf', maxIter=10, regParam=0.3)

# Fit the training data.
lr_model = lr3.fit(train_data)

# Print the coefficients.
print("Coefficients: " + str(lr_model.coefficients))
```

Figure 129: Data Mining (2)



```
#Random forest
random_forest1 = RandomForestRegressor(numTrees=5,featuresCol='features', labelCol='PM2-5_Transf')
random_forest2 = RandomForestRegressor(numTrees=10,featuresCol='features', labelCol='PM2-5_Transf')
random_forest3 = RandomForestRegressor(numTrees=100,featuresCol='features', labelCol='PM2-5_Transf')

rf1 = random_forest1.fit(train_data)
rf2 = random_forest2.fit(train_data)
rf3 = random_forest3.fit(train_data)

#Prediction
pred1 = rf1.transform(test_data)
pred2 = rf2.transform(test_data)
pred3 = rf3.transform(test_data)

#Evaluation
e1 = evaluator.evaluate(pred1)
e2 = evaluator.evaluate(pred2)
e3 = evaluator.evaluate(pred3)

#Output
print("Random Forest Regression Model 1: " + str(e1))
print("Random Forest Regression Model 2: " + str(e2))
print("Random Forest Regression Model 3: " + str(e3))
```

```
random_forest3 = RandomForestRegressor(numTrees=100,featuresCol='features', labelCol='PM2-5_Transf')
rf3 = random_forest3.fit(train_data)
pred3 = rf3.transform(test_data)
pred3.head(5)
```

Figure 129: Data Mining (3)

## Reference List

1: Chakure, A. (2019, June 29). *Random Forest Regression: Along with its implementation in Python*. Retrieved from https://towardsdatascience.com/random-forest-and-its-implementation-71824ced454f

1: Foley, B. (2018, February 14). *What is Regression Analysis and Why Should I Use It?.* Retrieved from https://www.surveygizmo.com/resources/blog/regression-analysis/

2: Li, T., Guo, Y., Liu, Y., Wang, J., Wang, Q., Sun, Z., … Shi, X. (2019, April). Estimating mortality burden attributable to short-term PM2.5 exposure: A national observational study in China. *Environment International*, *125*, 245–251. doi: https://doi.org/10.1016/j.envint.2019.01.073

3: Mckell, K. (2018, March 27). *5 data mining methods*. Retrieved from https://universe.byu.edu/2018/03/27/data-mining-sidebar/

4: Nunez, C. (2019, February 4). *CLIMATE 101: AIR POLLUTION.* Retrieved from https://www.nationalgeographic.com/environment/global-warming/pollution/

5: World Health Organization. (n.d.). *Air pollution.* Retrieved from https://www.ovc.gov/about/victimsfund.html

Yuchen Deng
237381181

## Disclaimer

"I acknowledge that the submitted work is my own original work in accordance with the University of Auckland guidelines and policies on academic integrity and copyright.
(See: https://www.auckland.ac.nz/en/students/forms-policies-and-guidelines/student-policies-and-guidelines/academic-integrity-copyright.html.).
I also acknowledge that I have appropriate permission to use the data that I have utilised in this project. (For example, if the data belongs to an organisation and the data has not been published in the public domain then the data must be approved by the rights holder.) This includes permission to upload the data file to Canvas. The University of Auckland bears no responsibility for the student's misuse of data."