Predict Important Feature of Influencing the Concentration of PM2.5 Pollution in Python

ITERATION 3 OSAS

Name: Ziteng Li

Student ID: 733922565

Table of Content

Abstract

In the industrialization era, air pollution levels and associated problems rose rapidly. Evidence shows that air pollution is one of the leading avoidable causes of disease and death globally. Even relatively low levels of air pollution pose health risks. One of the most important sources of air pollution is PM2.5. For the sustainable development of the human living environment, the international community needs to explore the core factors affecting air quality to reduce air pollution. This project used data mining methods and Python programming language to study air quality in Beijing. The three algorithms used in this project are linear Regression, Random Forest, and XGBoost, and the leading software is Spyder. The results of the three models are generally consistent. Compounds in the air have a significant effect on the concentration of PM2.5; the influence of the external environment also impacts the concentration of PM2.5.The results of the analysis can be used as a reference to improve air quality. To reduce the emission of related compounds in the air could become the ultimate goal of reducing the concentration of PM2.5.

Keywords: Sustainable development goals, PM2.5, Python

1. Business or Situation understanding.

1.1. Identify the objectives of the business/situation

1.1.1. Background

Air pollution may be the significant driver of NCDs(non-communicable diseases), including lung cancer, heart attacks, strokes, and accelerating climate change. Air pollution could make a global death increase of 7,000,000 per year. Global must try to prevent diseases and reduce air pollution.

1.1.2. Associate with sustainable problems

The target of 17 Sustainable Development Goals is to stop extreme poverty and create a healthy and sustainable world by 2030. There are five goals associated with the severe air pollution problems, including Goal 3 (Good Health and Well-being), Goal 7 (Affordable and Clean Energy), Goal 11 (Sustainable Cities and Communities), Goal 13 (Climate Action), and Goal 17 (Partnerships For The Goals). To deal with the high concentration of PM2.5 in the air is harmful to human beings, we should understand what PM2.5 is. “PM (Particulate matter) is a standard proxy indicator for air pollution. It affects more people than any other pollutant. PM’s significant components are sulfate, nitrates, ammonia, sodium chloride, black carbon, mineral dust, and water” (Team Airveda, 2017). It consists of a complex mixture of solid and liquid particles of organic and inorganic substances suspended in the air. “While particles with a diameter of 10 microns or less, can penetrate and lodge deep inside the lungs, the even more health-damaging particles are those with a diameter of 2.5 microns or less” (WHO, 2018).

1.1.3. Defining business objectives

The problem that need to solve by using data mining is that we need to find out which feature is influencing the concentration of PM2.5, and how this feature affected the concentration of PM2.5, is there any solutions for reduction the concentration of PM2.5, and how could we provide some actions for human beings to have a better life.

1.1.4. Business Success Criteria

Finding the main reasons for high PM2.5 concentration, use methods and models to predict the relative inferiority of PM2.5 concentration, find out solutions to reduce PM2.5 pollution by human activities, such as car emissions and factory pollution.

1.2. Assess the situation

1.2.1. General situation

While PM2.5 impacts everyone, people with breathing and heart problems, children, and the elderly are most sensitive. Due to particulate matter’s omnipresence, ambient particulate matter has proven to be more potent than alcohol and diabetes.

Exposure to PM2.5 has multiple short terms and long-term health impacts. The short term includes irritation in the eyes, nose, and throat, coughing, sneezing, and shortness of breath. Prolonged exposure to PM2.5 can cause permanent respiratory problems such as asthma, chronic bronchitis, and heart disease.

In low- and middle- income countries, exposure to pollutants in and around homes from

the household combustion of polluting fuels on open fires or traditional stoves for cooking, heating and lighting further increases the risk for air pollution-related diseases, including acute lower respiratory infections, cardiovascular disease, chronic obstructive pulmonary disease, and lung cancer.

1.2.2. Area evaluation

There are severe risks to health not only from exposure to PM but also from exposure to ozone (O3), nitrogen dioxide (NO2), carbon monoxide (CO) and sulfur dioxide (SO2). As with PM, concentrations are often highest, mainly in low- and middle-income countries. “Ozone is a significant factor in asthma morbidity and mortality. At the same time, nitrogen dioxide, and sulfur dioxide also can play a role in asthma, bronchial symptoms, lung inflammation, and reduced lung function” (WHO, 2018).

There are currently more than 30,000 available air quality monitoring stations in the world, out of which more than 12,000 are published on the World Air Quality Index project. In this dataset, the data came from 11 different monitoring stations in Beijing, mainly focus on the air quality, including different air types mentioned above.

1.2.3. Requirements

The data and project results are security and legal restrictions. For example, the dataset, Kaggle, is a data modeling and data analysis competition platform. There were lots of reliable dada set on the Kaggle website and the UCI-Machine Learning Repository website. In this project, the data set was chosen from the Kaggle website. It is a useful dataset for evaluating and predicting the concentration of PM2.5.

1.2.4 Feasibility study

● Hardware feasibility analysis

The project implementation in the hardware without too many requirements, only the general hardware configuration can be built. Upon investigation, the computers used by most companies are entirely achievable, so the hardware feasibility passes.

● Software feasibility analysis

This project uses Python as the programming language, Spyder, as the development environment. Python is a very concise language with a rich database and algorithms for training models. Spyder is an open-source, cross-platform integrated development environment that uses Python language for scientific programming. This free software has been widely used in the development and research of data mining.

● Limitation

The result of this project is simply to predict the concentration of PM2.5 through the existing data set (air quality in Beijing Changping). However, the dataset does not reflect the general situation of PM2.5 concentrations. In addition to the existing factors affecting PM2.5 concentrations, many human activities affect PM2.5 emissions that cannot be recorded. Therefore, these factors are not taken into account in the prediction.

1.2.5. Assumptions

In this project, the dataset was published in 2018. It is not hard to find the dataset used by many reports or articles during the research. There is no need for data quality assumptions. The project result would be analyzed by the Spyder (by using Python) and would report in later steps.

1.2.6. Constraints

This project could only predict the concentration of PM2.5 by the collected weather index. There are many uncertainties such as vehicle restrictions, factory closures, increase and decrease in human carbon travel, and another situation that cannot be recorded are not considered when predicting the concentration of PM2.5.

1.3. Determine data mining objectives

1.3.1. Data Mining Goals

The focus of this project is to study the changes in the concentration of PM2.5, conduct data mining on the data to predict the occurrence of PM2.5 in the later period effectively, and the government researches the exhaust emissions industry and automobile exhaust that lead to an increase in PM2.5. Control and help individuals prevent air pollution from harming the human body and reduce the occurrence of diseases. We also need to figure out which feature is influencing the concentration of PM2.5, and how this feature affected the concentration of PM2.5, is there any solutions for reduction the concentration of PM2.5, and how could we provide some actions for human beings to have a better life.

1.3.2. Data Mining Success Criteria

Finding the main reasons of high PM2.5 concentration base on the dataset. Use methods and functions in Python and discover the result in Spyder to predict the relative inferiority of PM2.5 concentration. Find out solutions to reduce PM2.5 pollution by human activities.

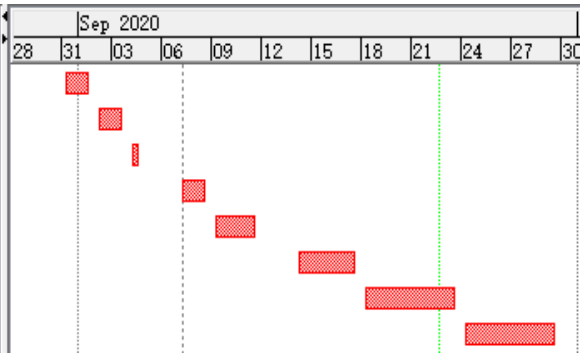
The success criteria for this project should be described by the data mining model and algorithms' results, for example, we need to find the suitable models and algorithms, find out is the accuracy score or error satisfied in our project, find out which feature has the most significant influence on the concentration of PM2.5. The analysis results can be used as a reference to improve air quality and achieve the ultimate goal of reducing PM2.5 concentration by reducing the emission of these related compounds in the air.

1.4. Produce a project plan

The overview plan for the study of this project is as shown in the Gantt Chart below.

Figure 1. Gantt Chart – Evidence from ProjectLibre

	Name	Duration	Start	Finish
1	Business/Situation understanding	2 days	8/31/20 8:00 AM	9/1/20 5:00 PM
2	Data understanding	2 days	9/2/20 8:00 AM	9/3/20 5:00 PM
3	Data preparation	1 day	9/4/20 8:00 AM	9/4/20 5:00 PM
4	Data Transformation	2 days	9/5/20 8:00 AM	9/8/20 5:00 PM
5	Data Mining Method Selection	3 days	9/9/20 8:00 AM	9/11/20 5:00 PM
6	Data Mining Algorithm Selection	4 days	9/12/20 8:00 AM	9/17/20 5:00 PM
7	Data Mining	4 days	9/18/20 8:00 AM	9/23/20 5:00 PM
8	Interpretation	4 days	9/24/20 8:00 AM	9/29/20 5:00 PM



The last three steps (Data Mining Algorithm Selection, Data Mining, and Interpretation) may take longer time than the previous steps, because I need to learn the Python packages and functions, and also do some research on Python programming language. During this period of time, I have learnt the how to use Python in machine learning, and the mainly software for this project is Tableau and Spyder.

Using the Gantt Chart can easily schedule the project, help me work more efficiently. In this project, the primary technique is the Spyder. It can be used to analyze and process the data by using Python programming language. Python is a strong and useful programming language in machine learning.

2. Data Understanding.

2.1. Collect initial data

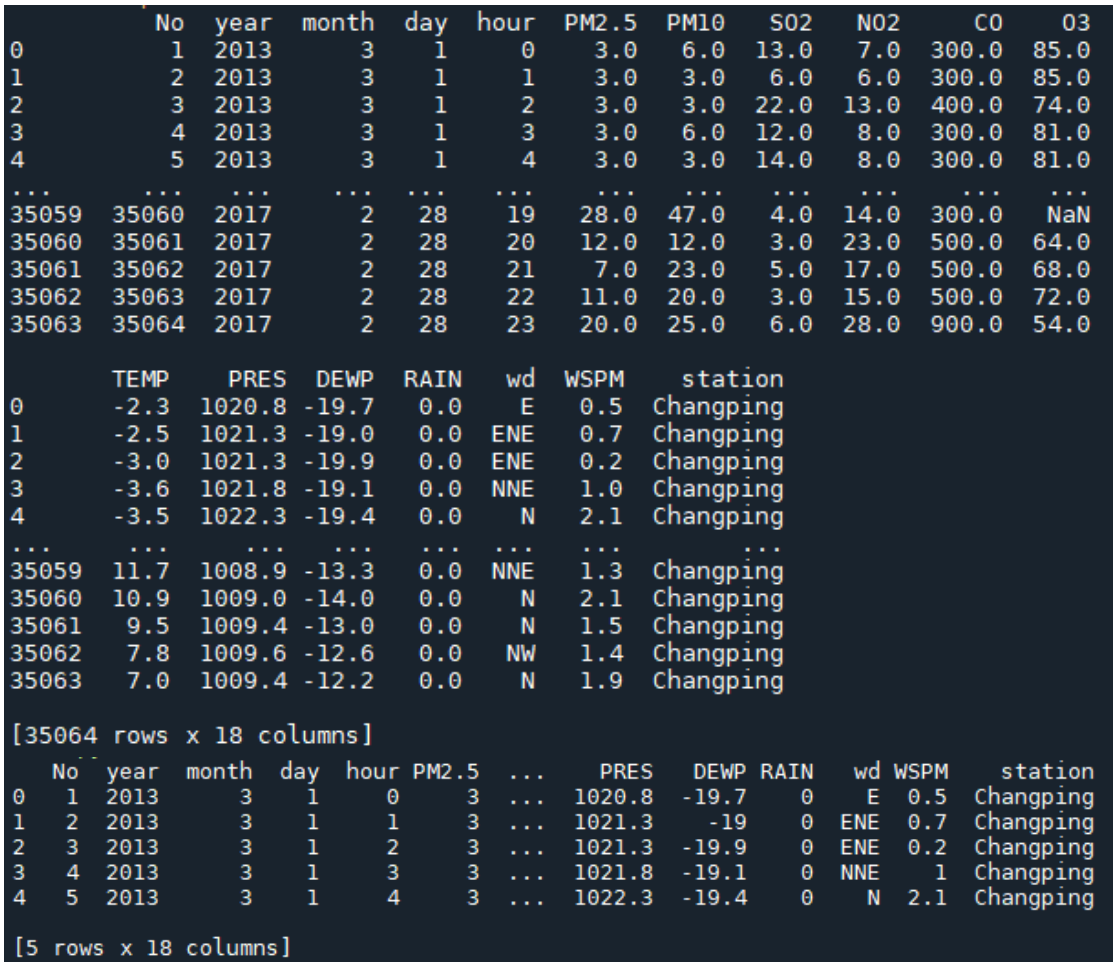
Asian regions are the typical areas that suffered from PM2.5 pollution, especially in Beijing, China. In order to find the analytical and usable data for PM2.5 pollution, the data set of this project has been found from Kaggle (https://www.kaggle.com/sid321axn/beijing-multisite-airquality-data-set), and UCI-Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Beijing+Multi-Site+Air-Quality+Data#). However, there are 12 different regions have been tested in this data set, research has been done for finding a moderate area (Changping in Beijing) in this data set to make the results of this project is significant and usable.

2.2. Describe the data

For the whole data set, there are 12 Beijing regions under-tested, respectively. The data set was recorded every hour while testing the air quality in the Beijing Changping district, which was in total 35064 hours' air-quality report.

There are 18 columns, 35064 rows in the data set, containing two data types: numeric and string.

Figure 2. Examples of data in the dataset – Evidence from Spyder



```
          No  year  month  day  hour  PM2.5  PM10   SO2   NO2     CO    O3
0          1  2013      3    1     0    3.0   6.0  13.0   7.0  300.0  85.0
1          2  2013      3    1     1    3.0   3.0   6.0   6.0  300.0  85.0
2          3  2013      3    1     2    3.0   3.0  22.0  13.0  400.0  74.0
3          4  2013      3    1     3    3.0   6.0  12.0   8.0  300.0  81.0
4          5  2013      3    1     4    3.0   3.0  14.0   8.0  300.0  81.0
...      ...   ...    ...  ...   ...    ...   ...   ...   ...    ...   ...
35059  35060  2017      2   28    19   28.0  47.0   4.0  14.0  300.0   NaN
35060  35061  2017      2   28    20   12.0  12.0   3.0  23.0  500.0  64.0
35061  35062  2017      2   28    21    7.0  23.0   5.0  17.0  500.0  68.0
35062  35063  2017      2   28    22   11.0  20.0   3.0  15.0  500.0  72.0
35063  35064  2017      2   28    23   20.0  25.0   6.0  28.0  900.0  54.0

       TEMP    PRES  DEWP  RAIN   wd  WSPM    station
0      -2.3  1020.8 -19.7   0.0    E   0.5  Changping
1      -2.5  1021.3 -19.0   0.0  ENE   0.7  Changping
2      -3.0  1021.3 -19.9   0.0  ENE   0.2  Changping
3      -3.6  1021.8 -19.1   0.0  NNE   1.0  Changping
4      -3.5  1022.3 -19.4   0.0    N   2.1  Changping
...     ...     ...   ...   ...  ...   ...        ...
35059  11.7  1008.9 -13.3   0.0  NNE   1.3  Changping
35060  10.9  1009.0 -14.0   0.0    N   2.1  Changping
35061   9.5  1009.4 -13.0   0.0    N   1.5  Changping
35062   7.8  1009.6 -12.6   0.0   NW   1.4  Changping
35063   7.0  1009.4 -12.2   0.0    N   1.9  Changping

[35064 rows x 18 columns]
   No  year  month  day  hour PM2.5  ...    PRES   DEWP RAIN   wd WSPM    station
0   1  2013      3    1     0     3  ...  1020.8  -19.7    0    E  0.5  Changping
1   2  2013      3    1     1     3  ...  1021.3    -19    0  ENE  0.7  Changping
2   3  2013      3    1     2     3  ...  1021.3  -19.9    0  ENE  0.2  Changping
3   4  2013      3    1     3     3  ...  1021.8  -19.1    0  NNE    1  Changping
4   5  2013      3    1     4     3  ...  1022.3  -19.4    0    N  2.1  Changping

[5 rows x 18 columns]
```

Figure 3. Examples of data in the dataset – Evidence from Tableau

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	6	13.000	7.000	300	85.000	-2.3000	1,020.8000	-19.7000	0.00000	E	0.50000	Changping
2	2013	3	1	1	3	3	6.000	6.000	300	85.000	-2.5000	1,021.3000	-19.0000	0.00000	ENE	0.70000	Changping
3	2013	3	1	2	3	3	22.000	13.000	400	74.000	-3.0000	1,021.3000	-19.9000	0.00000	ENE	0.20000	Changping
4	2013	3	1	3	3	6	12.000	8.000	300	81.000	-3.6000	1,021.8000	-19.1000	0.00000	NNE	1.00000	Changping
5	2013	3	1	4	3	3	14.000	8.000	300	81.000	-3.5000	1,022.3000	-19.4000	0.00000	N	2.10000	Changping
6	2013	3	1	5	3	3	10.000	17.000	400	71.000	-4.5000	1,022.6000	-19.5000	0.00000	NNW	1.70000	Changping
7	2013	3	1	6	4	6	12.000	22.000	500	65.000	-4.5000	1,023.4000	-19.5000	0.00000	NNW	1.80000	Changping
8	2013	3	1	7	3	6	25.000	39.000	600	48.000	-2.1000	1,024.6000	-20.0000	0.00000	NW	2.50000	Changping
9	2013	3	1	8	9	25	13.000	42.000	700	46.000	-0.2000	1,025.2000	-20.5000	0.00000	NNW	2.80000	Changping
10	2013	3	1	9	11	29	5.000	18.000	500	73.000	0.6000	1,025.3000	-20.4000	0.00000	NNW	3.80000	Changping
11	2013	3	1	10	9	10	3.000	10.000	300	83.000	2.0000	1,025.1000	-21.3000	0.00000	N	2.20000	Changping
12	2013	3	1	11	3	3	4.000	9.000	300	81.000	3.6000	1,024.8000	-20.7000	0.00000	NNE	2.70000	Changping
13	2013	3	1	12	3	6	4.000	8.000	300	90.000	4.8000	1,023.8000	-19.7000	0.00000	N	3.00000	Changping
14	2013	3	1	13	3	101	5.000	9.000	300	89.000	5.8000	1,022.8000	-20.6000	0.00000	NE	4.40000	Changping
15	2013	3	1	14	9	60	5.000	6.000	300	94.000	5.9000	1,022.6000	-20.5000	0.00000	N	3.60000	Changping
16	2013	3	1	15	3	34	6.000	7.000	300	94.000	6.0000	1,022.6000	-20.4000	0.00000	N	3.20000	Changping
17	2013	3	1	16	3	28	5.000	9.000	300	92.000	5.4000	1,022.8000	-20.0000	0.00000	NE	2.20000	Changping
18	2013	3	1	17	6	25	19.000	12.000	300	88.000	3.5000	1,023.4000	-20.8000	0.00000	ENE	2.00000	Changping
19	2013	3	1	18	4	17	40.000	22.000	600	77.000	2.7000	1,024.2000	-20.0000	0.00000	NE	3.60000	Changping
20	2013	3	1	19	7	19	37.000	22.000	600	75.000	2.0000	1,025.1000	-19.3000	0.00000	ENE	1.70000	Changping
21	2013	3	1	20	8	21	23.000	14.000	300	80.000	1.1000	1,025.9000	-18.3000	0.00000	NE	1.70000	Changping
22	2013	3	1	21	7	16	34.000	19.000	300	74.000	1.6000	1,026.7000	-17.4000	0.00000	E	2.30000	Changping

Table 1. Data details – Evidence from the dataset

Data Name	Unit	Data Type	Description
NO		Numeric	Rows of data
Year		Numeric	The observation time (year)
Month		Numeric	The observation time (month)
Day		Numeric	The observation time (day)
Hour		Numeric	The observation time (hour)
PM2.5	ug/m^3	Numeric	PM2.5 (Particulate matter) concentration
PM10	ug/m^3	Numeric	PM10 (Particulate matter) concentration
SO2	ug/m^3	Numeric	SO2 (Sulfur dioxide) concentration
NO2	ug/m^3	Numeric	NO2 (Nitrogen dioxide) concentration
CO	ug/m^3	Numeric	CO (Carbon monoxide) concentration
O3	ug/m^3	Numeric	O3 (Ozone) concentration
TEMP	degree Celsius	Numeric	Hourly temperature in particular area
PRES	hPa	Numeric	Pressure
DEWP	Celsius	Numeric	Dew point temperature
RAIN	mm	Numeric	Precipitation
WSPM	m/s	Numeric	Wind speed
wd	N S E W	String	Wind direction
station		String	Air-quality monitoring site (Changping)

2.3. Explore the data

This data set includes hourly air pollutants data from 12 nationally controlled air-quality monitoring sites. The air-quality data are from the Beijing Municipal Environmental Monitoring Center. The meteorological data in each air-quality area are matched with the nearest weather station from the China Meteorological Administration. The period is from March 1st, 2013 to February 28th, 2017.

Figure 4. The Python code of reading the dataset – Evidence from Spyder

```
# Read the dataset
dataset=pandas.read_csv("D:/1UOA MIT/Semester Two/INFOSYS 722/ASSIGNMENT/zli786-Iteration3-OSAS/dataset.csv")
print(dataset.head())
print(dataset.columns)
# The size of the dataset
print(dataset.shape)
```

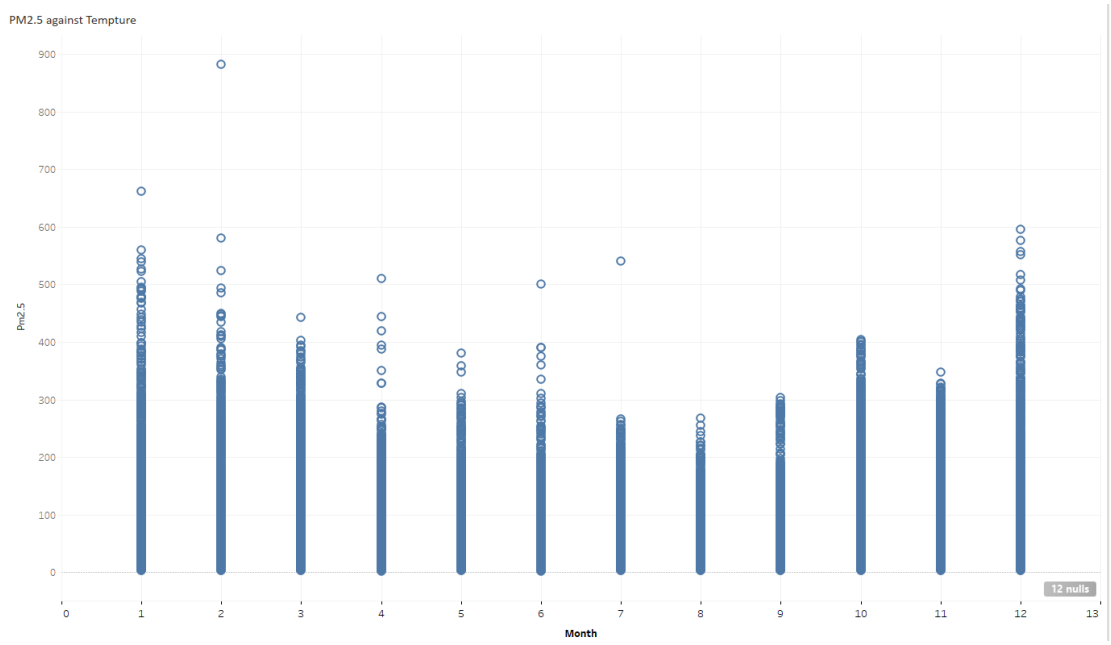
Figure 5. The name of columns and the size of data - Evidence from Spyder

```
Index(['No', 'year', 'month', 'day', 'hour', 'PM2.5', 'PM10', 'SO2', 'NO2',
       'CO', 'O3', 'TEMP', 'PRES', 'DEWP', 'RAIN', 'wd', 'WSPM', 'station'],
      dtype='object')
(35064, 18)
```

In the Python code (figure 4), dataset.head() function gives the result in figure 2, which shows some examples of our dataset. The dataset.columns method gives the result in figure 5, shows the head and name of each columns in the dataset. The .shape function shows the size of our dataset (35064) and the column number of our dataset (18 columns).

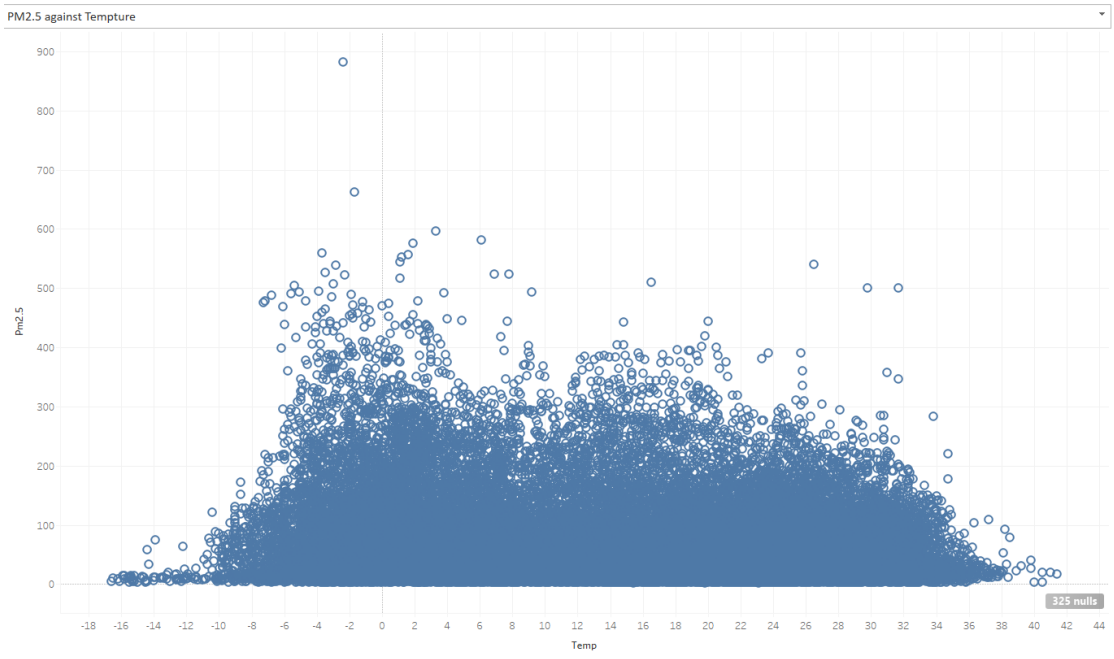
It can be known that the chemical element such as SO2, NO2, O3, CO; the TEMP (Hourly temperature in particular area); DEWP (Dew point temperature); RAIN (Precipitation); and WSPM (Wind speed) may have effects on the PM2.5 concentration, so these fields should be the useful data. However, the results of time (including the year and hour) and location (the station) may be useless, so they would not affect the PM2.5 concentration. The charts below show the distribution of PM2.5 concentration against month, temperature, pressure, dew point temperature, precipitation, and wind speed.

Figure 5. PM2.5 concentration against hour – Evidence from Tableau



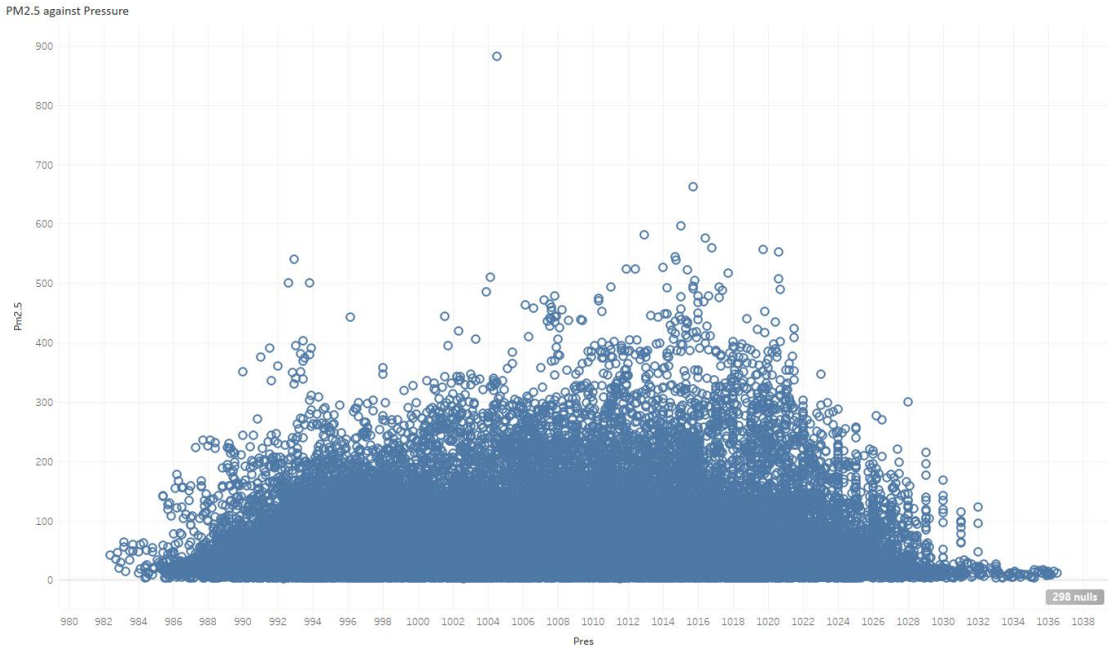
In figure 5, the PM2.5 concentration (>500) appears in the winter (December, January, February) and spring (March, April, May) in Beijing.

Figure 6. PM2.5 concentration against temperature – Evidence from Tableau



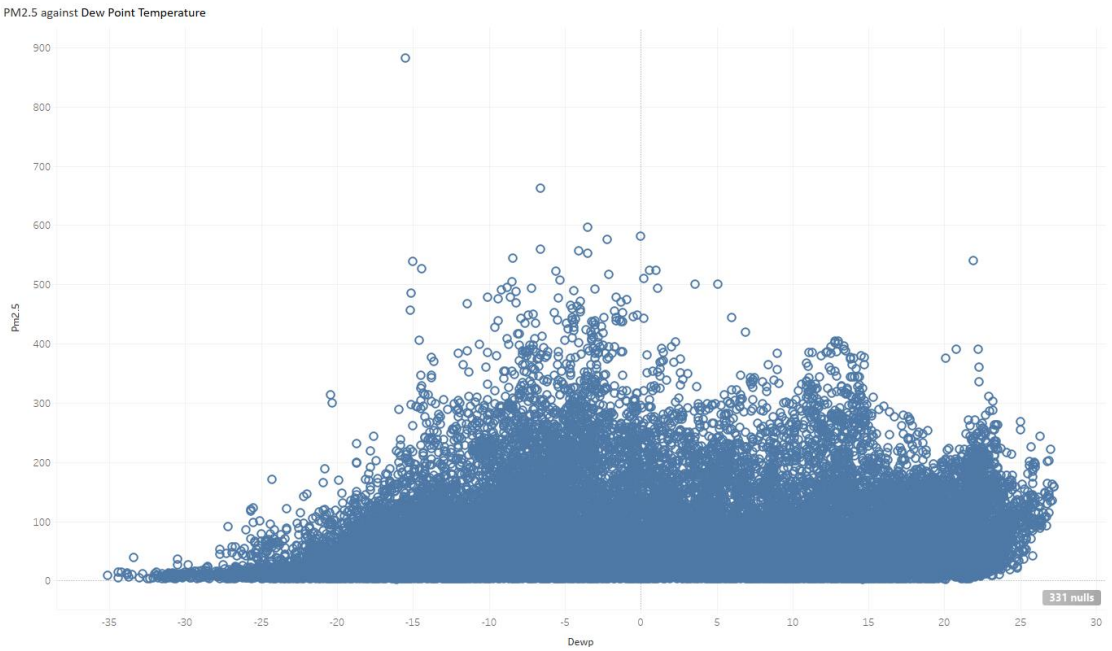
In figure 6, the PM2.5 concentration is affected by temperature. In this chart, the temperature between -5 to 10 degrees has a greater concentration of PM2.5(around 200) than the temperature between 20 with 40 degrees.

Figure 7. PM2.5 concentration against pressure – Evidence from Tableau



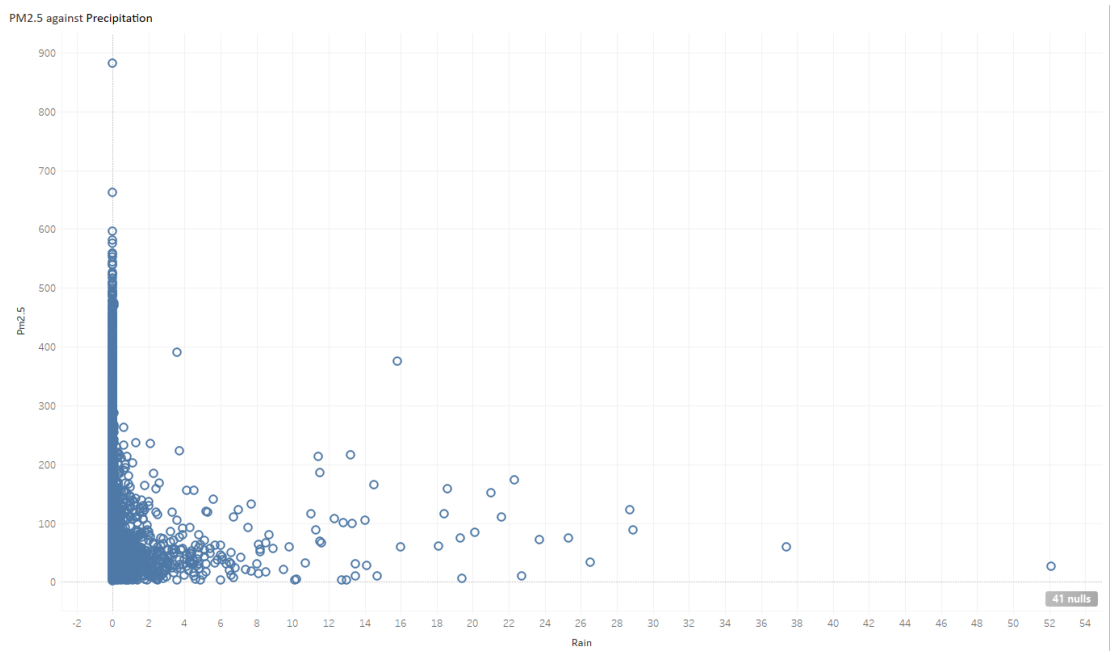
In figure 7, the PM2.5 concentration is not affected by the pressure. According to normal atmosphere pressure (around 1000 hPa), the concentration is slightly centralized near 200 ug/m^3.

Figure 8. PM2.5 concentration against dew point temperature – Evidence from Tableau



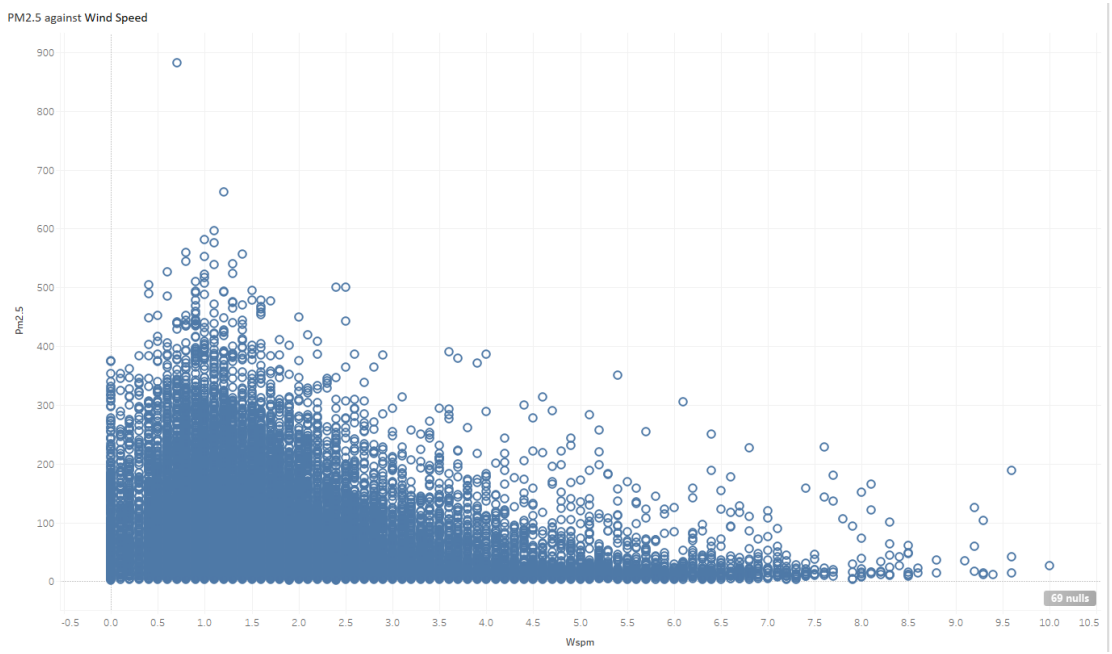
In figure 8, the PM2.5 concentration increased when the dew point temperature grows. The dew point temperature may influence the concentration of PM2.5.

Figure 9. PM2.5 concentration against precipitation – Evidence from Tableau



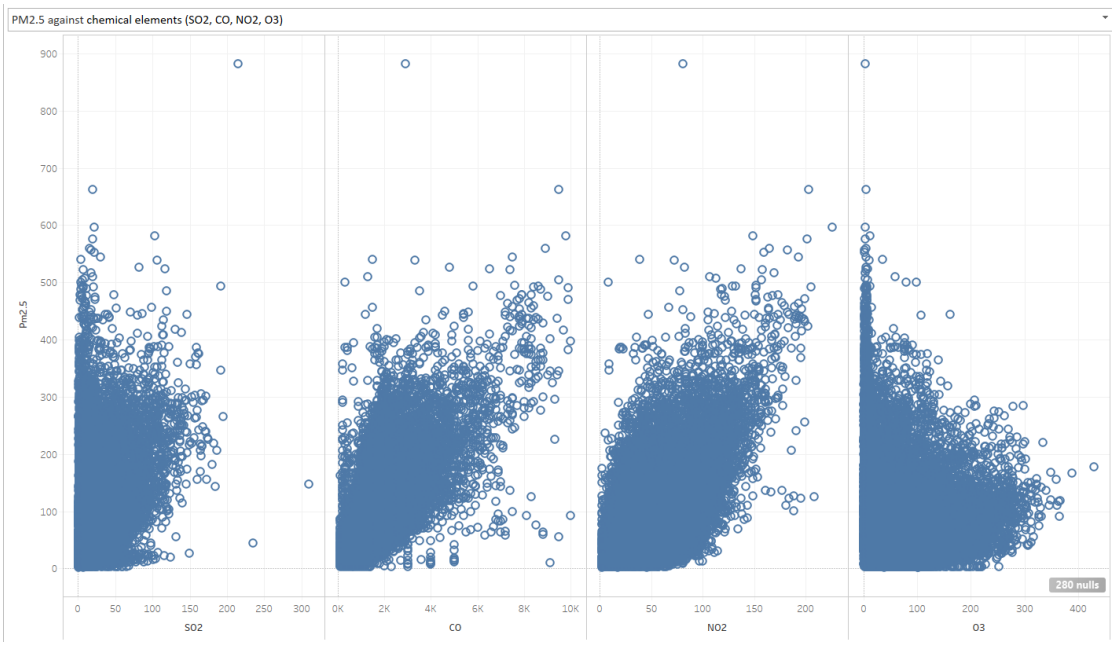
In figure 9, the concentration of PM2.5 in 500-100 ug/m^3 when 0 precipitation is more generous than other precipitation levels. The concentration of PM2.5 shows higher level when there is no precipitation, which means rainy days may affect the PM2.5.

Figure 10. PM2.5 concentration against Wind speed – Evidence from Tableau



In figure 10, the concentration of PM2.5 decreased when the wind speed become faster, which means the wind weather may influence the concentration of PM2.5.

Figure 11. PM 2.5 concentration against chemical elements (SO2, CO, NO2, O3) – Evidence from Tableau



In figure 11, the result shows that when the concentration of chemical elements (SO2, CO, NO2, O3) increase the concentration of PM2.5 may increase.

In the *Tableau*, Plots show values of a *Y* field against values of an *X* field. Often, these fields correspond to a dependent variable and an independent variable, respectively.

It is clear that the PM2.5 concentration is affected significantly by the precipitation and wind speed; the concentration would decrease while the weather is rainy or windy. To demonstrate the concentration of PM2.5 affected by different variables by using the plot chart. Meanwhile, the PM2.5 concentration is higher than average while under the lower temperature (-20 to 10).

The PM2.5 concentration in winter (December, January, February) and spring (March, April, May) is much higher than the other two seasons. Because most weather in summer and autumn in Beijing is rainy and windy, the concentration of PM2.5 decreased clearly in these two seasons.

2.4. Verify the data quality

The following charts show that how to verify the data quality in this project by using Python. In figure 9, using the dataset.info() function to check the details of the dataset, as we can see the output shows the name of columns, the number of the data, and the data type which default by python. There are 5 integer type (including the time variables and the id of the data), 11 float type (including the target PM2.5, and other variables that may influence the target), and 2 object type (the wd- wind direction, and the station).

Figure 12. The code of showing details in dataset – Evidence from Spyder

```
# The details of dataset
print(dataset.info())
```

Figure 13. The details of dataset – Evidence from Spyder

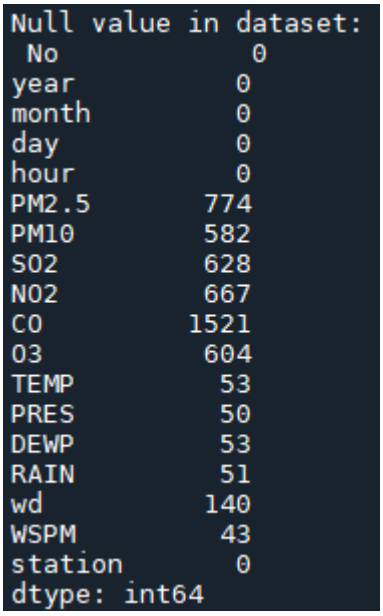
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35064 entries, 0 to 35063
Data columns (total 18 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   No       35064 non-null  int64
 1   year     35064 non-null  int64
 2   month    35064 non-null  int64
 3   day      35064 non-null  int64
 4   hour     35064 non-null  int64
 5   PM2.5    34290 non-null  float64
 6   PM10     34482 non-null  float64
 7   SO2      34436 non-null  float64
 8   NO2      34397 non-null  float64
 9   CO       33543 non-null  float64
 10  O3       34460 non-null  float64
 11  TEMP     35011 non-null  float64
 12  PRES     35014 non-null  float64
 13  DEWP     35011 non-null  float64
 14  RAIN     35013 non-null  float64
 15  wd       34924 non-null  object
 16  WSPM     35021 non-null  float64
 17  station  35064 non-null  object
dtypes: float64(11), int64(5), object(2)
memory usage: 4.8+ MB
None
```

Use isnull() function to check if there is the null value in the dataset, the use sum() function to calculate the numbers of null value, then print them in the output (as shown in figure 14).

Figure 14. The null value (missing value) in the dataset – Evidence from Spyder

```
#count the null value
null=dataset.isnull().sum()
print("Null value in dataset: \n",null)
```

Figure 15. The missing value in the dataset – Evidence from Spyder



In the output of figure 15, the numbers of null value (empty value) in each column are shown in the chart. In particular, the number of the missing value of CO has shown the highest number (1521), the target attribute of our project (PM2.5) has 774 missing value.

Some of the fields contain the Outliers and Extremes, these data should be analyzed and process in the later steps. The data in this particular situation may depend on the day's weather, so the data may not be removable.

3. Data preparation

3.1. Select the data

By observing complete data and the distribution of missing data, I decided to select all the data in the dataset. The data set got an entire record of the weather conditions for every hour from March 1st, 2013 to February 28th, 2017. The missing data of each attribute is irregularly distributed every hour during this period.

Therefore, all the data in the data set should be considered. There are 18 fields in this data set. 'No' has marked as the record id. 'wd' is the wind direction, was recorded in abbreviation. 'Station' has been marked as a Flag value, because there is only one monitor station(Changping) in this data set. Other fields are all numeric values.

The describe() function shows the descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN (null or empty) values. It can be used to analyzes both numeric and object series, as well as dataset column sets of mixed data types. The output will vary depending on what is provided. Refer to the notes below for more detail.

Figure 16. The code of selected data – Evidence from Spyder

```
# Generate descriptive statistics of the dataset
print(dataset.describe(include="all"))
```

Figure 17. The output of generating descriptive statistics of the data – Evidence from Spyder

	PM2.5	PM10	SO2	NO2	CO
count	34290.000000	34482.000000	34436.000000	34397.000000	33543.000000
unique	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	NaN	NaN	NaN
mean	71.099743	94.657871	14.958906	44.182086	1152.301345
std	72.326926	83.441738	20.975331	29.519796	1103.056282
min	2.000000	2.000000	0.285600	1.847700	100.000000
25%	18.000000	34.000000	2.000000	22.000000	500.000000
50%	46.000000	72.000000	7.000000	36.000000	800.000000
75%	100.000000	131.000000	18.000000	60.358200	1400.000000
max	882.000000	999.000000	310.000000	226.000000	10000.000000

	O3	TEMP	PRES	DEWP	RAIN
count	34460.000000	35011.000000	35014.000000	35011.000000	35013.000000
unique	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	NaN	NaN	NaN
mean	57.940003	13.686111	1007.760278	1.505495	0.060366
std	54.316674	11.365313	10.225664	13.822099	0.752899
min	0.214200	-16.600000	982.400000	-35.100000	0.000000
25%	15.636600	3.400000	999.300000	-10.200000	0.000000
50%	46.000000	14.700000	1007.400000	1.800000	0.000000
75%	80.000000	23.300000	1016.000000	14.200000	0.000000
max	429.000000	41.400000	1036.500000	27.200000	52.100000

	wd	WSPM	station
count	34924	35021.000000	35064
unique	16	NaN	1
top	NNW	NaN	Changping
freq	4776	NaN	35064
mean	NaN	1.853836	NaN
std	NaN	1.309808	NaN
min	NaN	0.000000	NaN
25%	NaN	1.000000	NaN
50%	NaN	1.500000	NaN
75%	NaN	2.300000	NaN
max	NaN	10.000000	NaN

In figure 17, this chart shows the output of generating descriptive statistics of the data, particularly, the count means the count number of non-NA/null observations; max means the maximum of the values in the object; min means the minimum of the values in the object; mean is to calculate the mean of the values; std shows the standard deviation of the observations.

3.2. Clean the data

There are blank values, outliers, and extreme values in the data, affecting the analysis results. As mentioned in 2.4, verify the data, the missing value should be replaced.

This data set's total record is 35064, and there is the null or empty value show in the

chart below. In terms of replacing or handling the missing value, figure 18 shows the numbers of null value or missing value in the dataset, figure 20 shows the columns that need to be treaded to handle the missing value.

The code in figure 19 is using to find the missing values, base on the numbers of missing value is greater than 50% of the total data in the dataset.

Figure 18. The numbers of null value in dataset – Evidence from Spyder

```
Null value in dataset:
 No            0
year           0
month          0
day            0
hour           0
PM2.5        774
PM10         582
SO2          628
NO2          667
CO          1521
O3           604
TEMP          53
PRES          50
DEWP          53
RAIN          51
wd           140
WSPM          43
station        0
dtype: int64
```

Figure 19. Find the missing value of more than 50% - Evidence from Spyder

```
#Find the Missing values of more than 50%
remove = []
for i in dataset.columns:
    print("The columns need to be treated: ", i)
    if dataset[i].isnull().sum()>(dataset.shape[0]*0.5):
        remove.append(i)
```

Figure 20. The output of the columns needs to be treated – Evidence from Spyder

```
The columns need to be treated:  No
The columns need to be treated:  year
The columns need to be treated:  month
The columns need to be treated:  day
The columns need to be treated:  hour
The columns need to be treated:  PM2.5
The columns need to be treated:  PM10
The columns need to be treated:  SO2
The columns need to be treated:  NO2
The columns need to be treated:  CO
The columns need to be treated:  O3
The columns need to be treated:  TEMP
The columns need to be treated:  PRES
The columns need to be treated:  DEWP
The columns need to be treated:  RAIN
The columns need to be treated:  wd
The columns need to be treated:  WSPM
The columns need to be treated:  station
```

Figure 21. Remove the missing value – Evidence from Spyder

```
# Remove the variable
dataset1=dataset.drop(remove,axis=1)
# Remove the missing value
dataset1=dataset1.dropna()
print("The columns after removed the missing value:\n",dataset1.columns)
```

Figure 22. The code to check the missing value – Evidence from Spyder

```
# Check the dataset if the missing value has been removed
check_null = dataset1.isnull().any()
print(check_null)
print(dataset1.shape)
```

In figure 21, use drop(remove, axis=1) to remove the variables that found in pervious code, then use dropna() function to remove the missing value.

Figure 23. The output of columns after removed missing value – Evidence from Spyder

```
The columns after removed the missing value:
 Index(['No', 'year', 'month', 'day', 'hour', 'PM2.5', 'PM10', 'SO2', 'NO2',
       'CO', 'O3', 'TEMP', 'PRES', 'DEWP', 'RAIN', 'wd', 'WSPM', 'station'],
      dtype='object')
```

Figure 24. The output of check the missing value – Evidence from Spyder

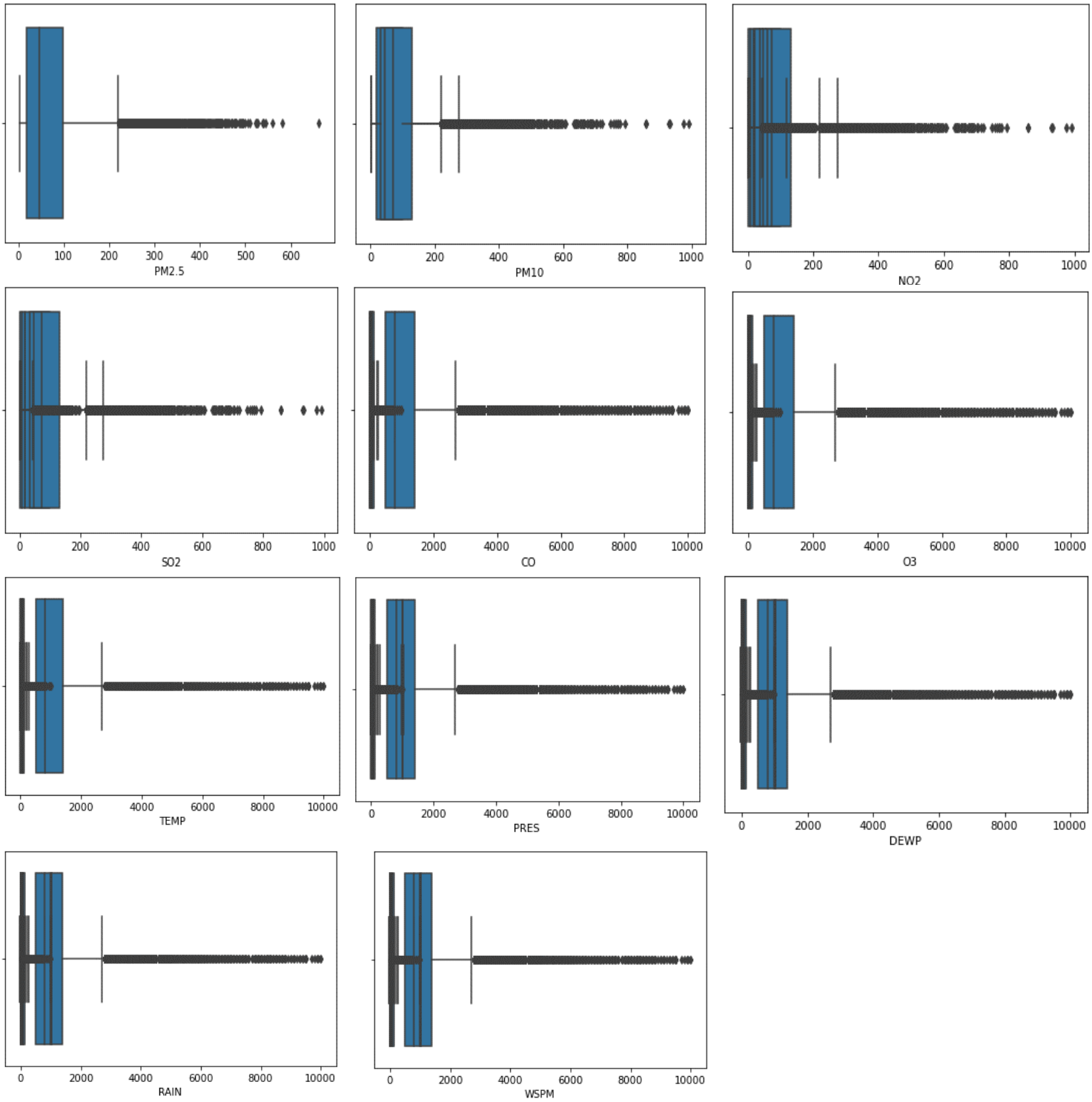
```
No          False
year        False
month       False
day         False
hour        False
PM2.5       False
PM10        False
SO2         False
NO2         False
CO          False
O3          False
TEMP        False
PRES        False
DEWP        False
RAIN        False
wd          False
WSPM        False
station     False
dtype: bool
(32681, 18)
```

Figure 23 shows the output after cleaning the dataset, its layout the columns after

removing the null value from the dataset, all the selected columns as shown in figure 20 have been treated. At the bottom of figure 24, shows the total numbers of the data has been reduced from 35064 to 32681.

The outliers and extremes value in this dataset may shows the specific time that the concentration of PM2.5 or other chemical elements is extremely high, so that we check the outliers and keep them in the dataset.

Figure 25. The box plot of each float type date – Evidence from Spyder



3.3. Construct the data

Python is a great language for doing data analysis, primarily because of the fantastic ecosystem of data-centric python packages. Pandas is one of those packages and makes importing and analyzing data much easier.

Pandas dataframe.append() function is used to append rows of other dataframe to the end of the given dataframe, returning a new dataframe object. Columns not in the original dataframes are added as new columns and the new cells are populated with NaN value.

However, in this project, the primary purpose is to predict the concentration of PM2.5, and the dataset already contains all the required indicators (including all the attributes that may influence the target); as a result, it is not necessary to add more datasets or tables.

3.4. Integrate various data sources

Figure 24. The merge function to integrate the various data source – Evidence from Python API reference

```
DataFrame.merge(right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False,
sort=False, suffixes='_x', '_y', copy=True, indicator=False, validate=None) ¶
```

[source]

The DataFrame.merge() function (as shown in figure ?) that under pandas' package could be used to combine two datasets into one dataset, and merge DataFrame or named Series objects with a database-style join. The join is done on columns or indexes. If joining columns on columns, the DataFrame indexes will be ignored. Otherwise if joining indexes on indexes or indexes on a column or columns, the index will be passed on.

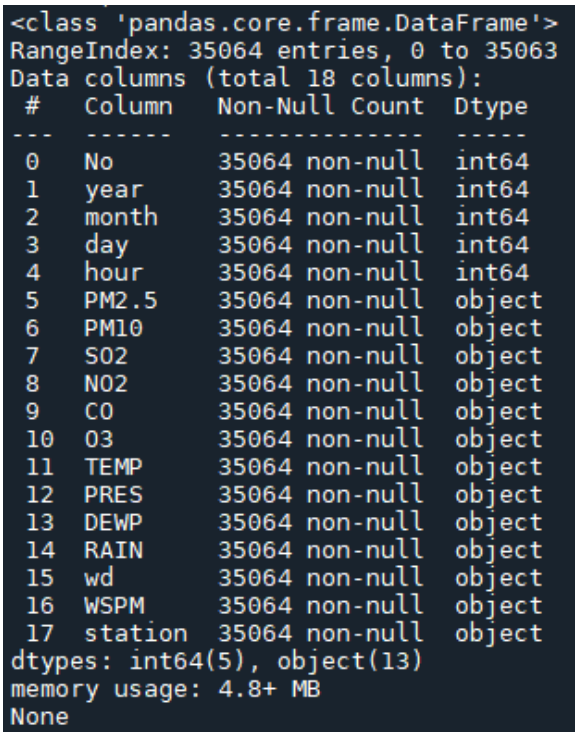
However, in this project, the dataset contains all valid data. It is an unnecessary need to collect data and does not need to integrate any tables or attributes.

3.5. Format the data as required

In this step talking about the data formatting in Python, there are some useful method and function could be used, including using dataframe.dtypes() to identify data type, and using dataframe.astype() to convert the data type. For example, in this project, the dataset were have 13 object type of data by default in python, however, the PM2.5, PM10, SO2, CO, O3, should be float data type, because they are recorded number of the air quality. After using the astype() function, some of the data type has converted to the correct data type as shown in figure , there are 11 float data (including the target PM2.5, and other variables that may influence the target), 5 integer data (including the time variables and the id of the data), 2 object data (the wd- wind direction, and the station) type in the dataset.

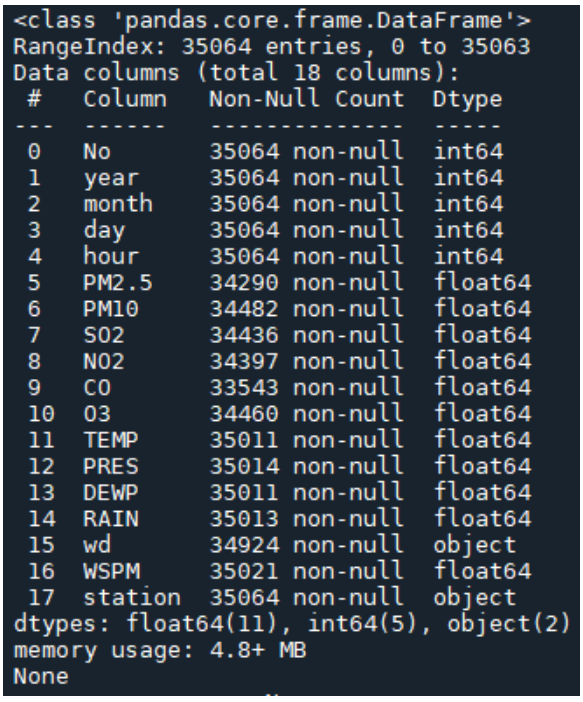
The figure 25 shows the data type before formatted, and the figure shows the data type after formatted.

Figure 25. The data type information before formatted – Evidence from Spyder



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35064 entries, 0 to 35063
Data columns (total 18 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   No       35064 non-null  int64
 1   year     35064 non-null  int64
 2   month    35064 non-null  int64
 3   day      35064 non-null  int64
 4   hour     35064 non-null  int64
 5   PM2.5    35064 non-null  object
 6   PM10     35064 non-null  object
 7   SO2      35064 non-null  object
 8   NO2      35064 non-null  object
 9   CO       35064 non-null  object
 10  O3       35064 non-null  object
 11  TEMP     35064 non-null  object
 12  PRES     35064 non-null  object
 13  DEWP     35064 non-null  object
 14  RAIN     35064 non-null  object
 15  wd       35064 non-null  object
 16  WSPM     35064 non-null  object
 17  station  35064 non-null  object
dtypes: int64(5), object(13)
memory usage: 4.8+ MB
None
```

Figure 26. The data type information after formatted – Evidence from Spyder



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35064 entries, 0 to 35063
Data columns (total 18 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   No       35064 non-null  int64
 1   year     35064 non-null  int64
 2   month    35064 non-null  int64
 3   day      35064 non-null  int64
 4   hour     35064 non-null  int64
 5   PM2.5    34290 non-null  float64
 6   PM10     34482 non-null  float64
 7   SO2      34436 non-null  float64
 8   NO2      34397 non-null  float64
 9   CO       33543 non-null  float64
 10  O3       34460 non-null  float64
 11  TEMP     35011 non-null  float64
 12  PRES     35014 non-null  float64
 13  DEWP     35011 non-null  float64
 14  RAIN     35013 non-null  float64
 15  wd       34924 non-null  object
 16  WSPM     35021 non-null  float64
 17  station  35064 non-null  object
dtypes: float64(11), int64(5), object(2)
memory usage: 4.8+ MB
None
```

The data set comes from a professional statistical team, and All data were obtained from the Meteorological Monitoring Center. The format of data is now suitable for this project to predict the PM2.5 concentration.

4. Data transformation

4.1. Reduce the data

After cleaned the data, replaced the missing value, some of the useless attributes should be reduced. Additionally, the data may influence accuracy by making the model overfit the data set.

Figure 27. The code of testing the correlation with target – Evidence from Spyder

```
# Test the columns of the correlation with PM2.5
print(dataset1.corr()["PM2.5"])
```

Figure 28. The correlation of columns with target (PM2.5) – Evidence from Spyder

```
No       -0.055993
year     -0.049505
month    -0.021399
day      -0.006403
hour      0.034190
PM2.5     1.000000
PM10      0.865804
SO2       0.466832
NO2       0.679889
CO        0.767194
O3       -0.095132
TEMP     -0.111434
PRES      0.004944
DEWP      0.114330
RAIN     -0.012783
WSPM     -0.276370
Name: PM2.5, dtype: float64
```

Figure 29. Remove the columns with low correlation – Evidence from Spyder

```
# Get the columns from the dataframe.
columns = dataset1.columns.tolist()
# Remove the columns with low correlation
columns = [c for c in columns if c not in ['No','year','PM10','station','wd','day','PRES']]
# Print the valid columns
print("Valid columns name: \n", columns)
```

Figure 30. The results after remove the columns – Evidence from Spyder

```
Valid columns name:
 ['month', 'hour', 'PM2.5', 'SO2', 'NO2', 'CO', 'O3', 'TEMP', 'DEWP', 'RAIN', 'WSPM']
```

Reasons for deletion

- 'No', 'year' and 'station' are the useless fields in predicting the PM2.5.
- 'No' is the record number, there are 35064 rows of data.
- 'day' is to record the particular day, has low correlation with the concentration of PM2.5 (0.006403), which may not affect the concentration of PM2.5.
- 'year' is the year from 2013 to 2017, which did not affect the PM2.5 concentration, only for recording the time.
- 'station' is the region that had been monitored(Changping, Beijing), there is only one station in this dataset.
- 'PM10' is the concentration of PM10, which is also a particulate matter, and it may affect the role of other air attributes.
- 'PRES' is the air pressure when recording the data, this has low correlation (0.04944) with the concentration of PM2.5.

4.2. Project the data

Data normalization and data standardization of datasets is a common requirement for many machine learning estimators implemented in scikit-learn; they might behave badly if the individual features do not more or less look like standard normally distributed data: Gaussian with zero mean and unit variance.

As we already reduced the low correlation or useless data to predict the concentration of PM2.5, in this step, the dataset needs to be transformed and standardized. The charts below show the steps and output of the data standardization.

Figure 31.The code of feature scaling – Evidence from Spyder

```
# Remove the useless columns
dataset1 = dataset1.drop(['No','year','PM10','station','wd','day','PRES'],axis=1)
# Scale the data
dataset2 = pd.DataFrame(mpl.scale(dataset1.drop(['PM2.5'],axis=1)))
# Set the target column - PM2.5
target_column = dataset1['PM2.5']
# Reset the index of the columns starting from 0
target_column = target_column.reset_index()
target_column = target_column.iloc[:,1:]
new_dataset = pd.concat([dataset2,target_column],axis=1)
# Standardization the target
new_dataset = new_dataset[new_dataset['PM2.5']>=0]
new_dataset['PM2.5'] = np.log2(new_dataset['PM2.5']+1)
# print the new dataset
print(new_dataset)
```

Figure 32. The new dataset after data standardization – Evidence from Spyder

```
          month      hour       SO2       NO2        CO        O3      TEMP  \
0     -1.008212 -1.657953 -0.097893 -1.261165 -0.770345  0.512633 -1.378151
1     -1.008212 -1.513871 -0.430320 -1.294959 -0.770345  0.512633 -1.395706
2     -1.008212 -1.369789  0.329513 -1.058400 -0.679899  0.308141 -1.439591
3     -1.008212 -1.225706 -0.145383 -1.227371 -0.770345  0.438273 -1.492254
4     -1.008212 -1.081624 -0.050403 -1.227371 -0.770345  0.438273 -1.483477
...         ...       ...       ...       ...       ...       ...       ...
32676 -1.295659  0.791445 -0.430320 -0.517694 -0.679899 -0.119434  0.113955
32677 -1.295659  1.223692 -0.572789 -0.720459 -0.589453  0.122239 -0.219575
32678 -1.295659  1.367774 -0.477810 -0.923224 -0.589453  0.196600 -0.342454
32679 -1.295659  1.511856 -0.572789 -0.990812 -0.589453  0.270961 -0.491665
32680 -1.295659  1.655938 -0.430320 -0.551488 -0.227668 -0.063663 -0.561881

           DEWP      RAIN      WSPM     PM2.5
0     -1.505330 -0.079364 -1.043998  2.000000
1     -1.454756 -0.079364 -0.891116  2.000000
2     -1.519780 -0.079364 -1.273322  2.000000
3     -1.461981 -0.079364 -0.661793  2.000000
4     -1.483656 -0.079364  0.179058  2.000000
...         ...       ...       ...       ...
32676 -0.992362 -0.079364  1.784319  4.247928
32677 -1.093511 -0.079364  0.179058  3.700440
32678 -1.021262 -0.079364 -0.279588  3.000000
32679 -0.992362 -0.079364 -0.356029  3.584963
32680 -0.963463 -0.079364  0.026176  4.392317
```

In figure 31, the code shows the steps of standardization dataset, the function scale (from sklearn.preprocessing package) provides a quick and easy way to perform the operation on dataset, then we reset the columns name and index for standardization the dataset, and create a new dataset for later analysis.

Figure 32 shows the new dataset after data standardization, it may help improving data quality.

5. Data-mining methods selection

5.1. Match and discuss the objectives of data mining to data mining methods

5.1.1. Supervised learning and unsupervised learning

The training data are labeled, which is supervised learning. Training data without a label is unsupervised learning (Sathya & Abraham, 2013).

The supervised learning uses the algorithm or model to understand the relationship of the data input and output, including classification and regression. The unsupervised learning is that only have input data and no corresponding output, for example, clustering.

5.1.2. Discuss the method with the Data mining objective

Base on the evidence above supervised learning is suitable for this project. Meanwhile, learning the existing function from a given dataset is useful to predict new data results.

In this project, the data mining objective is to predict the concentration of PM2.5 by the given fields, including the effect of weather and other pollute air. The target is to predict the PM2.5 concentration and consider all the other attributes.

The classification method is used to divide a group of data objects in the database into different classes according to the classification model and find out the similar characteristics of the data objects (Data for Data Mining, n.d.).

The regression analysis method reflects the temporal features of attributes in the dataset. At the same time, the regression method generates a function that maps data elements to a real value prediction variable and discovers the dependence between variables or attributes (Data for Data Mining, n.d.).

5.2. Select the appropriate data-mining methods based on discussion

The clustering method is not suitable for the project, both input data and output data are required in this project. In this case, unsupervised learning method is not allowed to be used.

The classification method is also not suitable for this objective. The classification algorithms estimate the collection from inputs data to the categorical output variables. On the other hand, the data type of classification output might be a category. However, this project's prediction data is numeric, which is continuous data, but not discrete.

Regarding the data-mining objectives, there are both input variables and output variables; thus, it is necessary to choose the supervised learning methods. Because the predicted target is PM2.5, it is the continuous value, and the data mining method should be decided as the regression method.

6. Data-mining algorithms selection

6.1. Conduct exploratory analysis and discuss

In terms of selecting algorithms, there are three main data-mining algorithms for solving regression problems, including “Linear and Polynomial Regression, Neural Networks, and Regression Trees and Random Forests” (Seif, 2018).

To choose a suitable algorithm for the project, it is necessary to find out the definition and function of the algorithm and discover the advantages and disadvantages.

- Linear and Polynomial Regression

  To begin with the simple case, the single variable linear regression is one of the techniques to function the relationship between the independent input variable and the dependent output variable, which would use a linear model. On the other hand, the multivariable linear regression is a model for multiple input independent attributes (feature variables) and an output dependent variable. This linear model would combine various input variables (Seif, 2018). Another general case is called polynomial regression; this model becomes a non-linear combination of the independent variables and would require knowledge of how data relates to the output.

  Advantages:

  Linear regression is useful for small and straightforward size dataset, fast to model.

  Linear regression is easy to understand the valuable business decisions.

  Limitations:

  Because the polynomial regression model must contain the data structure and the feature variables’ relationship, it is hard for non-linear data to design.

  Linear and Polynomial Regression do not have enough benefits during analysis of complex data.

  For data mining objective:

  The most appropriate algorithm for the project should be the multivariable linear regression. We set the PM2.5 concentration as the target, which is the output, and combine all other attributes as the input, which is the independent variables.

- Neural Networks

  An interrelated group of nodes is called neurons; it consists of a neural network. For each neuron, the independent input variables would model as a multi-variable linear combination, and all the feature variables may multiply a value called weights. A non-linearity gives the neural network the ability to model complex non-linear relationships, as is applied to the linear combination of neurons. A neural network can have multiple layers where one layer’s output would pass other results in the same way. There is no generally non-linearity applied to the output. “Neural Networks are trained using Stochastic Gradient Descent (SGD) and the backpropagation algorithm” (Seif, 2018).

Advantages:

Neural networks could effectively model complex non-linear relationships because it contains many non-linearities' parameters.

We generally do not need to worry about learning the data structure and the feature variables' relationships in neural networks.

Limitations:

The neural network models are not easy to understand and translate.

Other machine learning algorithms commonly exceed neural networks in "small data" cases.

They demand an amount of data to accomplish high performance.

For data mining objective:

As the neural networks could effectively model complex non-linear relationships and achieve high performance, the dataset's quantity could train the neural network. Therefore, set the concentration of PM2.5 as the target, the neural network in SPSS Modeler could be a suitable algorithm for the project.

- Regression Trees and Random Forests

Regarding the decision tree's base case, it is an intuitional model where one traverses down the tree branches, then selects the next chapter to keep going down based on the decision at the node. The induction of tree is input training instances, decide the best attributes to split on. All training instances should be categorized after splitting the dataset and recurring on the resulting split datasets. While building the tree, the goal is to break into the attributes that create the purest child nodes possible, keeping to a minimum the number of splits that would need to be made to classify all instances in our dataset. The purity relates to a previously unseen example for it to be appropriately organized.

Decision trees are a part of random forests, and the input elements are run through several decision trees. For classifications, a voting system is used to establish the final class. For regression, the output value of all trees is averaged (Seif, 2018).

Advantages:

Regression trees could easily handle complex non-linear relationships. They could often accomplish high performance, better than polynomial regression, and the same functions with neural networks.

Very easy to understand and translate. The decision boundaries are realistic and easy to understand.

Limitations:

The training decision trees can be capable of major overfitting because of its nature. The over complex and unnecessary structure may occur in a completed decision tree model.

It would require more memory and time to reach higher performance while using more extraordinary random forest.

For data mining objective:

Because the relationships between the target(PM 2.5) and other attributes are uncertain, the tree regression model may successfully find the connection. The 35064 instances should be able to prevent overfitting.

- XGBoost

  According to the XGBoost algorithm steps, first initialized to a constant, GB is based on the first derivative RI, XGBoost is based on the first derivative GI, and the second derivative HI, iteratively generate a base learner, then add to update the learner. The primary learner in XGBoost could be either CART (GBtree) or linear classifier (GBlinear) (Regression Trees, 2013).

  Advantages:

  XGBoost model could enumerate several candidates, thus finding the best split point from the candidates based on the formula (Random Forest, GBDT, and XGBoost, n.d.).

  For data mining objective:

  In this project, the dataset contains many data, and the XGBoost model can segment and manage the data efficiently because it could find the best split point.

All the algorithms above show the advantages and disadvantages. Because we want to find out the prediction of PM2.5, I would select the Linear Regression algorithm, Random Forest algorithm, and the XGBoost Tree algorithm for this project to study the final results.

6.2. Select data-mining algorithms based on discussion

After understanding the algorithms associated with the regression, the main training set and test set are continuous data, it better to use the regression model to generate the result. Base on the discovery in Iteration two, some of the models could also be used in this project.

Base on the 6.1 discussion and the function's generated results, the project will analyze the "Linear Regression," "Random Forest," and "XGBoost Tree" model for the prediction of PM2.5.

6.3. Build/Select appropriate models and choose relevant parameters

Base on the discussion in 6.2 to select the algorithms, the Linear regression, Random forest, and XGBoost model have been selected. I would recommend an analysis of how I build the appropriate models and discuss the relevant parameters in these models.

Because the large values would affect more on prediction results than the smaller amounts. As a result, without generate the significant value after the "train_test_split()" function in the Python (as shown in figure 33). The dataset has been separated into two part, one is training set, the other one is testing set. The test size for testing size is 0.33, which is 33%, the size of training set should be 67% automatically. The random state has

been set equal to 123, is to controls the shuffling applied to the data before applying the split. Pass 123 for reproducible output across multiple function calls. Base on the research, when random state is 123, the outputs would not change while running the code multiple times.

Figure 33. Feature selection in python – Evidence from Spyder

```
# Store the variable we'll be predicting on
# Generate the training set.  Set random_state to be able to replicate results.
x_train, x_test, y_train, y_test = train_test_split(
    new_dataset.drop(['PM2.5'], axis=1), new_dataset['PM2.5'], test_size=0.33, random_state=123)
# Print the shapes of both sets.
print(x_train.shape)
print(x_test.shape)
```

Figure 34. The code of printing independent variables and dependent variables for regression model – Evidence from Spyder

```
# Print the independent variables name in the trainning set
print(x_train.columns)
# Print the dependent variables name in the trainning set
print(y_train.name)
```

Figure 35. The output of figure 34, the target and predictors in training set – Evidence from Spyder

```
In [8]:
   ...: print(x_train.columns)
Index(['month', 'hour', 'SO2', 'NO2', 'CO', 'O3', 'TEMP', 'DEWP', 'RAIN',
       'WSPM'],
      dtype='object')

In [9]:
   ...: print(y_train.name)
PM2.5
```

Model 1: Linear Regression

For the Linear Regression model, we should import the package first (in figure 36), the sklearn.linear_model module implements a variety of linear models, Linear Regression model is one of them.

Figure 36. Import the package for Linear regression in python – Evidence from Spyder

```
import pandas as pd
import seaborn as sns
import numpy as np
import sklearn.preprocessing as mpl
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error,explained_variance_score
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
```

Figure 37. Initialization of Linear Regression model – Evidence from Spyder

```
#### linear regression model
# Initialize the model class.
linear_model = LinearRegression(fit_intercept=True,normalize=False)
```

Figure 37 shows the initialization of linear regression model in python, we need to initialize the model class first, then fit the model to the training data. There are some parameters has been set for the linear regression model, including fit_intercept, by default is true, which means the system would calculate the intercept for this model; normalize has been set to false, this is because we have already done the data standardize in step 4.2 (see figure 31 and 32), so that we do not need to do normalization again in this model; other parameters such as copy_X, n_jobs has been set as default.

Figure 38. Fit the Linear Regression model – Evidence from Spyder

```
# Fit the model to the training data.
linear_model.fit(x_train, y_train)
```

In figure 38, use .fit() function to fit the Linear regression model with the independent variables and dependent variables, x_train and y_train is the predictors and target after use the feature select function of this dataset (shown in figure 33), the details of the predictors and target are shown in figure 35, the target is 'PM2.5', predictors are 'month', 'hour', 'SO2', 'NO2', 'CO', 'O3', and so on.

Model 2: Random Forest

For the Random Forest model, we should import the package first (in figure 39), the sklearn.ensemble module includes ensemble-based methods for classification, regression and anomaly detection., Random Forest model is the one of the regression models that we choose to use in this project.

Figure 39. Import the package for Random Forest in python – Evidence from Spyder

```
import pandas as pd
import seaborn as sns
import numpy as np
import sklearn.preprocessing as mpl
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error,explained_variance_score
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
```

Figure 40. Initialization in the Random Forest model– Evidence from Spyder

```
####Random forest model
# Initialize the model class.
random_forest_model = RandomForestRegressor(n_estimators=10,max_features="auto",random_state=123)
```

In figure 40, the chart shows the initialization of random forest model in python, we need to initialize the model class first, then fit the model to the training data. There are some parameters has been set for the random forest model, including n_estimators has been set to 10, which is the number of trees in the forest; the max_features has been set auto, which is the number of features to consider when looking for the best split, the max feature is equal to the numbers of feature in this model; random_state has been set to 123, which random_state is the seed used by the random number generator; other parameters such as the maximum depth of the tree(max_depth) is default to none; the minimum number of samples required to be at a leaf node (min_samples_leafs) is default to 1.

Figure 41. Fit the Random Forest model – Evidence from Spyder

```
# Fit the model to the training data.
random_forest_model.fit(x_train, y_train)
```

In figure 41, use .fit() function to fit the Random forest model with the training input samples and target value, x_train and y_train is the predictors and target after use the feature select function of this dataset (shown in figure 33), the details of the training input samples and target are shown in figure 35, the target is 'PM2.5', predictors are 'month', 'hour', 'SO2', 'NO2', 'CO', 'O3', and so on.

Model 3: XGBoost

For the Random Forest model, we should import the package first (in figure 39), the xgboost module is the eXtreme Gradient Boosting library, which is an open source library. XGBRFRegressor model is the one of the XGBoost models in this library.

Figure 42. Import the package for XGBoost in python – Evidence from Spyder

```
import pandas as pd
import seaborn as sns
import numpy as np
import sklearn.preprocessing as mpl
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error,explained_variance_score
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
```

Figure 43. Initialization of XGBoost model – Evidence from Spyder

```
###XGBoost model
# Initialize the model class.
xgboost_model = xgb.XGBRFRegressor(n_estimators=10, seed=123)
```

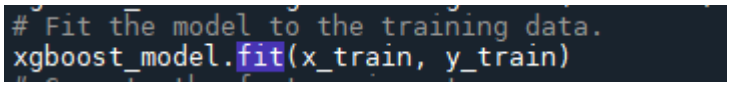
In figure 43, the chart shows the initialization of XGBoost model in python, we need to initialize the model class first, then fit the model to the training data. There are some parameters has been set for the random forest model, including n_estimators have been set to 10, which is the number of trees in the forest; the seed parameter is the same as the random_states, which has been set to 123, other default parameters are show in figure 44.

Figure 44. Documentation of XGBRFRegressor – Evidence from Spyder

```
XGBRFRegressor(learning_rate=1, subsample=0.8,
               colsample_bynode=0.8, reg_lambda=1e-5, *,
               objective="reg:squarederror", max_depth=None,
               n_estimators=100, verbosity=None,
               booster=None, tree_method=None, n_jobs=None,
               gamma=None, min_child_weight=None,
               max_delta_step=None, colsample_bytree=None,
               colsample_bylevel=None, reg_alpha=None,
               scale_pos_weight=None, base_score=None,
               random_state=None, missing=np.nan,
               num_parallel_tree=None,
               monotone_constraints=None,
               interaction_constraints=None,
               importance_type="gain", gpu_id=None,
               validate_parameters=None, **kwargs)

No documentation available
```

Figure 45. Fit the XGBoost model – Evidence from Spyder



In figure 45, use .fit() function to fit the XGBoost model with the training input samples and target value, x_train and y_train is the predictors and target after use the feature select function of this dataset (shown in figure 33), the details of the training input samples and target are shown in figure 35, the target is 'PM2.5', predictors are 'month', 'hour', 'SO2', 'NO2', 'CO', 'O3', and so on.

7. Data Mining

7.1. Create and justify test designs

We want to be able to figure out how accurate an algorithm is using our error metrics. However, evaluating the algorithm on the same data it has been trained on will lead to overfitting. We want the algorithm to learn generalized rules to make predictions, not memorize how to make specific predictions.

In order to prevent overfitting, we'll train our algorithm on a set consisting of 67% of the data and test it on another set consisting of 33% of the data. To do this, we first import the train_test_split function from the sklearn.model_selection package (shown in figure 46). Then, set the parameters for train_test_split function, use the dataset that has been modified in step 4 (data transformation), set the test size equal to 0.33, random state equal to 123. Set the random_state equal to 123, the test result would not change after running serval times, the system selected the same random data.

Figure 46. Import the train_test_split function in python – Evidence from Spyder

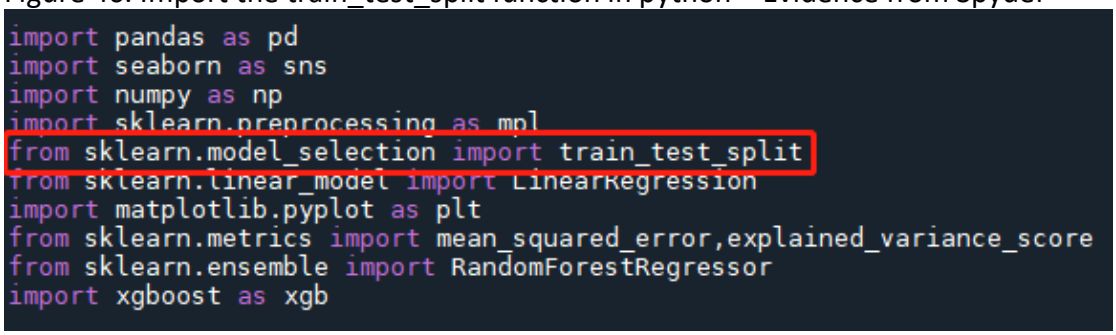


Figure 47. Split the data into training dataset and test dataset – Evidence from Spyder

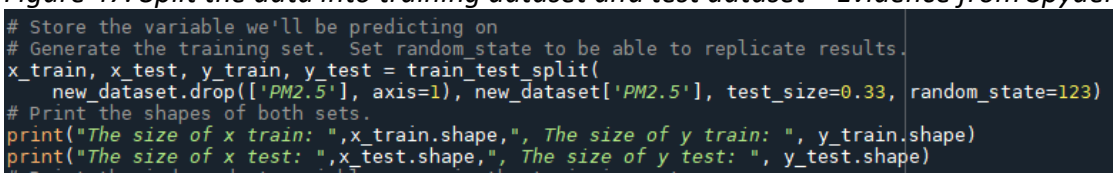
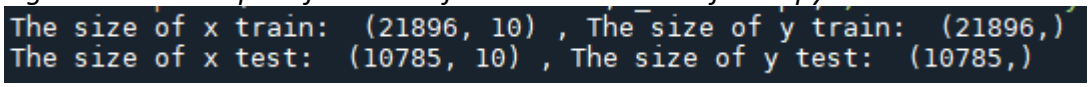


Figure 48. The output of the size of dataset – Evidence from Spyder



The dataset randomly sample 67% of the rows to be in the training set, then put everything else in the testing set, the output of this function has shown in figure 48, where the training set contains 21896 rows of data, and 10 columns, the testing set contains 10785 rows of data, and 10 columns.

7.2. Conduct data mining

After completing the preparations, the three models run successfully, and the results would show up, respectively.

Model 1: Linear Regression

Figure 49. The code of running the Linear Regression model – Evidence from Spyder

```
#### linear regression model
# Initialize the model class.
linear_model = LinearRegression()
# Fit the model to the training data.
linear_model.fit(x_train, y_train)
importances = linear_model.coef_
# Returns the indices that would sort an array.
indices = np.argsort(importances)[::-1]
# Generate predictions for the test set
predictions = linear_model.predict(x_test)
# compute the explained variance score for the model
score = explained_variance_score(y_test,predictions)
# print the score
print("Accuracy score: ", score)
# Compute error between our test predictions and the actual values
linear_model_mse = mean_squared_error(predictions, y_test)
# print the mean squared error
print("Mean squared error: ", linear_model_mse)

# show the result in bar chart
plt.figure()
plt.title("Feature importances(Linear Regression)")
plt.barh(range(x_train.shape[1]), importances[indices]
         , color='r', align="center")
plt.yticks(range(x_train.shape[1]), x_train.columns[indices])
plt.ylim([-1, x_train.shape[1]])
plt.show()
```

Figure 50. The output of the Linear Regression model – Evidence from Spyder

```
    ...: predictions = linear_model.predict(x_test)
    ...: # compute the explained variance score for the model
    ...: score = explained_variance_score(y_test,predictions)
    ...: # print the score
    ...: print("Accuracy score: ", score)
    ...: # Compute error between our test predictions and the actual values
    ...: linear_model_mse = mean_squared_error(predictions, y_test)
    ...: # print the mean squared error
    ...: print("Mean squared error: ", linear_model_mse)
Accuracy score:  0.6725590704043392
Mean squared error:  0.7670261150269483
```

Model 2: Random Forest model

Figure 51. The code of running the Random Forest model – Evidence from Spyder

```
####Random forest model
# Initialize the model class.
random_forest_model = RandomForestRegressor(random_state=123)
# Fit the model to the training data.
random_forest_model.fit(x_train, y_train)
# Compute the feature importance
importances = random_forest_model.feature_importances_
# Returns the indices that would sort an array.
indices = np.argsort(importances)[::-1]
# Generate predictions for the test set
predictions = random_forest_model.predict(x_test)
# Compute the accuracy score for the model
accuracy = explained_variance_score(y_test,predictions)
# Print the score
print("Accuracy: ", accuracy)
# Compute error between our test predictions and the actual values
random_forest_model_mse = mean_squared_error(predictions, y_test)
# Print the mean squared error
print("Mean squared error: ", random_forest_model_mse)
# Show the result in bar chart
plt.figure()
plt.title("Feature importances(Random Forest)")
plt.barh(range(x_train.shape[1]), importances[indices]
         , color='r', align="center")
plt.yticks(range(x_train.shape[1]), x_train.columns[indices])
plt.ylim([-1, x_train.shape[1]])
plt.show()
```

Figure 52. The output of the Random Forest model – Evidence from Spyder

```
    ...: # Generate predictions for the test set
    ...: predictions = random_forest_model.predict(x_test)
    ...: # Compute the accuracy score for the model
    ...: accuracy = explained_variance_score(y_test,predictions)
    ...: # Print the score
    ...: print("Accuracy: ", accuracy)
    ...: # Compute error between our test predictions and the actual values
    ...: random_forest_model_mse = mean_squared_error(predictions, y_test)
    ...: # Print the mean squared error
    ...: print("Mean squared error: ", random_forest_model_mse)
Accuracy:  0.8396240967137185
Mean squared error:  0.37602003577903165
```

Model 3: XGBoost model

Figure 53. The code of running the XGBoost model – Evidence from Spyder

```
###XGBoost model
# Initialize the model class.
xgboost_model = xgb.XGBRFRegressor(seed=123)
# Fit the model to the training data.
xgboost_model.fit(x_train, y_train)
# Compute the feature importance
importances = xgboost_model.feature_importances_
# Returns the indices that would sort an array.
indices = np.argsort(importances)[::-1]
# Generate predictions for the test set
predictions = xgboost_model.predict(x_test)
# Compute the accuracy score for the model
accuracy = explained_variance_score(y_test,predictions)
# Print the score
print("Accuracy: ", accuracy)
# Compute error between our test predictions and the actual values
xgboost_model_mse = mean_squared_error(predictions, y_test)
#  Print the mean squared error
print("Mean squared error: ", xgboost_model_mse)
# plot the feature importances of the XGBoost model
plt.figure()
plt.title("Feature importances(XGBoost model)")
plt.barh(range(x_train.shape[1]), importances[indices]
         , color='r', align="center")
plt.yticks(range(x_train.shape[1]), x_train.columns[indices])
plt.ylim([-1, x_train.shape[1]])
plt.show()
```

Figure 54. The output of the XGBoost model – Evidence from Spyder

```
    ...: # Generate predictions for the test set
    ...: predictions = xgboost_model.predict(x_test)
    ...: # Compute the accuracy score for the model
    ...: accuracy = explained_variance_score(y_test,predictions)
    ...: # Print the score
    ...: print("Accuracy: ", accuracy)
    ...: # Compute error between our test predictions and the actual values
    ...: xgboost_model_mse = mean_squared_error(predictions, y_test)
    ...: #  Print the mean squared error
    ...: print("Mean squared error: ", xgboost_model_mse)
Accuracy:  0.7803440431587867
Mean squared error:  0.5145518676592501
```

In the Linear regression model, the explained variance score shows 0.67, and the mean squared error is 0.767 approximately (in figure 50).

In the Random Forest model, the explained variance score is estimated to 0.839, and the mean squared error is estimated to 0.376 (in figure 52).

In the XGBoost model, the explained variance score shows 0.78, and the mean squared error is 0.514 approximately (in figure 54).

The results from the three models are satisfying and straightforward for further analysis.

7.3. Search for patterns

- Data mining mode – mined pattern

Python is a powerful high-level dynamic language. Many developers prefer Python because of its exact syntax, well-structured modules, software packages, and great flexibility and extensive modern capabilities (Boyanov, 2016). In particular, many of Python's schema features help beginners, even though it is a programming language. Python's syntax is so close to English that reading Python code is like reading an article. Python also has a rich tool library that provides an excellent foundation for a wide variety of uses. In general, programming language patterns limit the programming process. Different programming languages may have various limitations, but Python's programming patterns combine well with the advantages of other programming language patterns. Python is developed across multiple programming languages and is an advanced development result based on practice. Some design patterns have been built into Python, and many of them have been simplified. People do not need to think about the deep thoughts of realizing functions in a complicated way. They just need to determine their goals and use a few code lines to call functions directly. Because of this, it is very convenient to write each step in Python. Besides, when drawing in Python, use the Python visualization class tool.

Using the .corr() function to automatically select the essential and relevant predictors, as mentioned in step 4, provides the features' valuable and high correlation attributes. Use .train_test_split() function to separate the dataset and avoid overfitting. Use .scale() function to do the data standardization to finish the data transformation. In step 6, access the sklearn package, metrics module to calculate the accuracy score and mean squared error. Simultaneously, use sklearn.linear_model and sklearn.ensemble and xgboost module to import the most effective algorithms: The Linear Regression, the Random Forest model, the XGBoost tree model, for the regression problems.

- The output and results of the three models could be found in the following graphs.

*Figure 55. Feature importance of **Linear Regression model** – Evidence from Spyder*

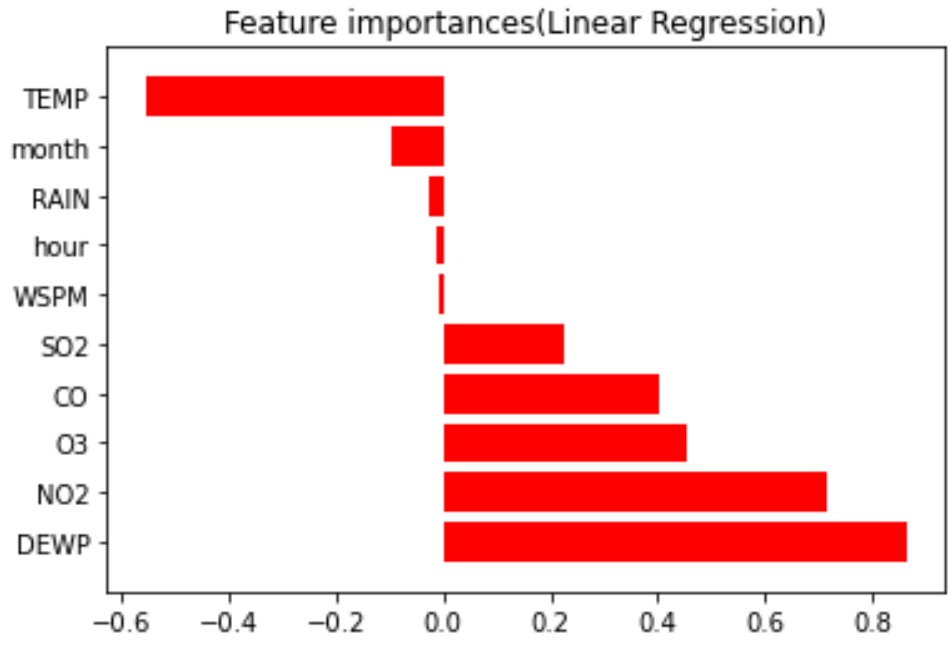
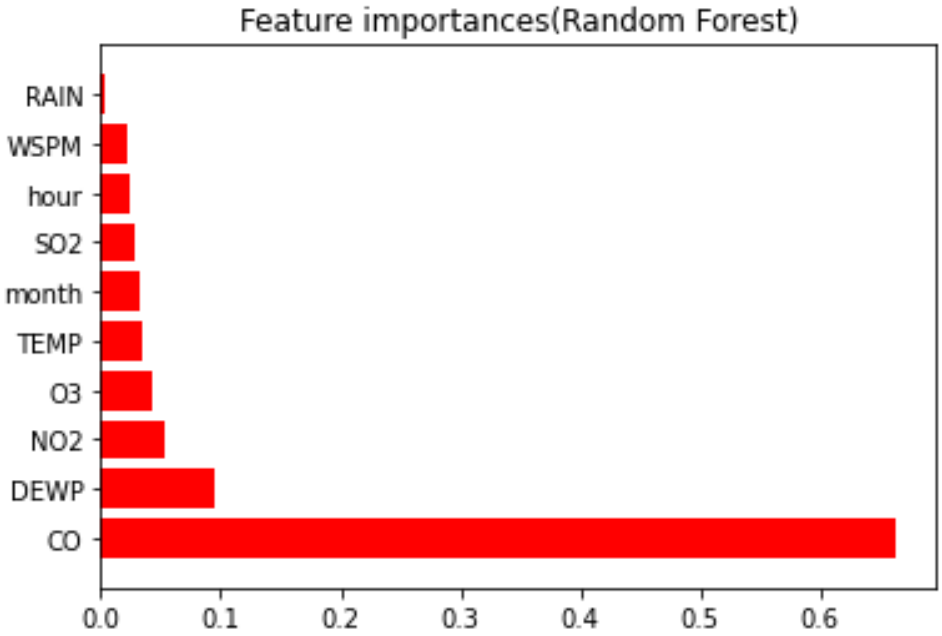


*Figure 56. Feature importance of **Random Forest model** – Evidence from Spyder*

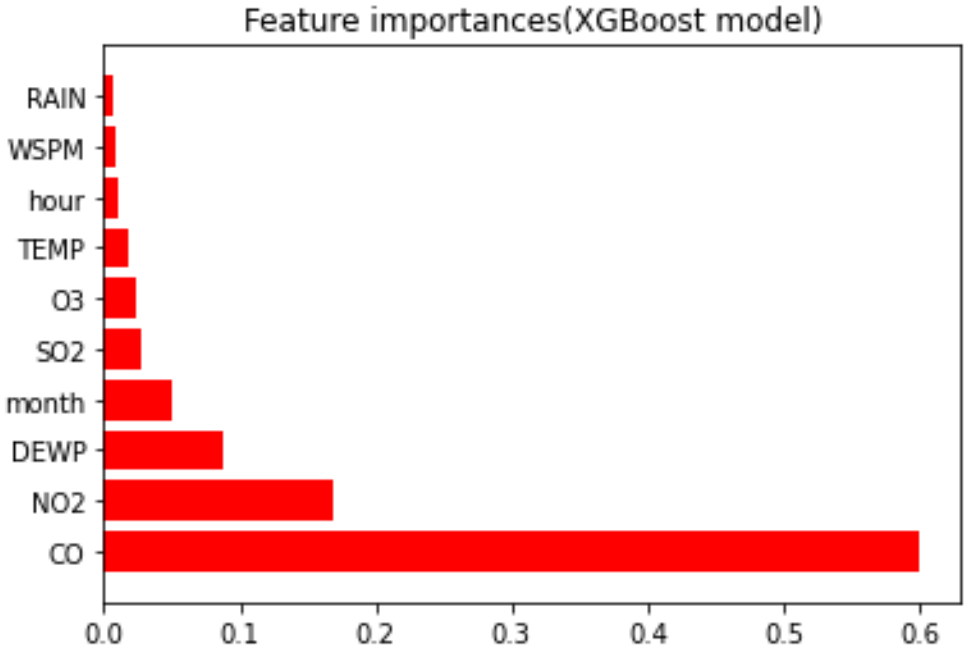


The essential predictor based Linear Regression model's output is the dew point temperature ('DEWP'), with the number of importance is 0.83 approximately (Figure 55).

Additionally, based on the output of the XGBoost Tree model, the most significant predictor is the Carbon monoxide ('CO'), with 0.67 importance; the second is Dew point temperature ('DEWP') (Figure 56).

Meanwhile, based on the output of the Neural Network model, the most important predictor is the concentration of Carbon monoxide ('CO'), with the figure is 0.6; the second significant predictor is the nitrogen dioxide('NO2') (Figure 57).

Figure 57. *Feature importance of **XGBoost model** – Evidence from Spyder*

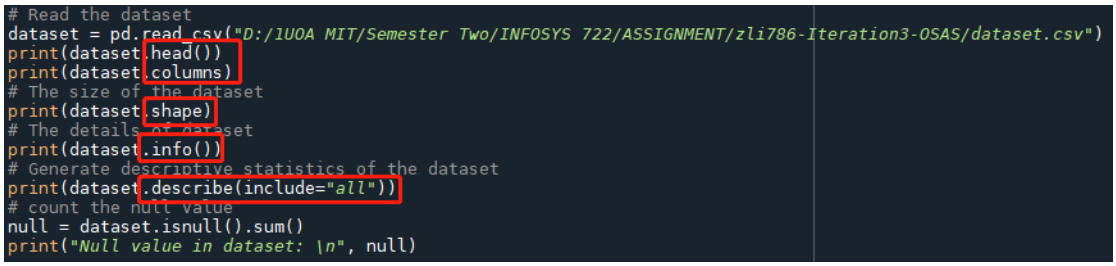


8. Interpretation

8.1. Study and discuss the mined patterns

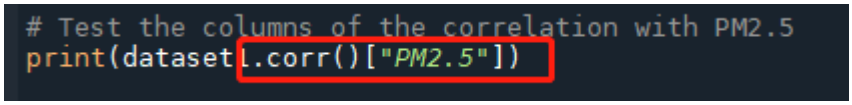
The missing data's replacement helped construct the complete relationship between the independent variables and the dependent variable. Before the data preparation, there are a few portions of space or NA value data for some fields. Replacing the missing data helps us to delete the noise and make every instance useful.

Figure 58. The code from pandas package – Evidence from Spyder(Python)



Using the pandas package for the basic analysis of the dataset, to see the columns, the size, the descriptive statistics, missing value and the outliers in the dataset (in figure 58).

Figure 59. The code of checking correlations – Evidence from Spyder(Python)



While selecting the relevant features, the correlation of each feature and the target (PM2.5) is shown in figure , after the analysis by .corr() function under the pandas package. These high correlations help us to decide the features that using in the algorithms and keep moving on the study (in figure 59).

Figure 60. The code of data standardization – Evidence from Spyder(Python)

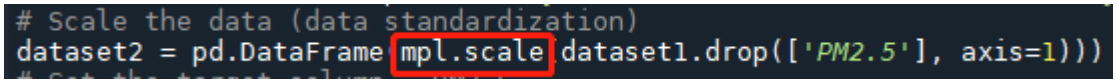
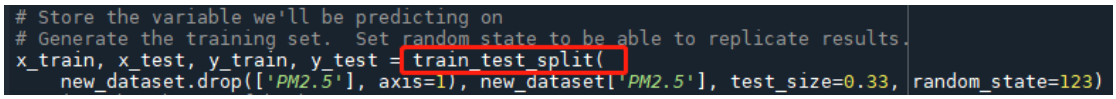


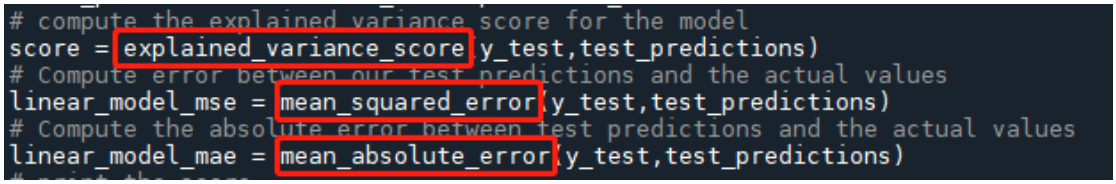
Figure 61. The code of preventing overfitting – Evidence from Spyder(Python)



Use .scale() function to do the data standardization to finish the data transformation (figure 60).

Use .train_test_split() function to separate the dataset and avoid overfitting (figure 61).

Figure 62. The code from pandas package – Evidence from Spyder(Python)



In step 6, access the sklearn package, metrics module to calculate the explained variance score, mean squared error, mean absolute error (figure 62).

Simultaneously, use sklearn.linear_model and sklearn.ensemble and xgboost module to import the most effective algorithms: The Linear Regression, the Random Forest model, the XGBoost tree model, for the regression problems.

Based on the discussions in step five(Data-mining methods selection) and step six(Data-mining algorithms selection), this project's main pattern is **regression**. By using the algorithms in Python, some significant predictors with high feature importance have shown in the result, including the Dew point temperature ('DEWP'), shows in figure 55; the concentration of Carbon monoxide ('CO'), shows in figure 56 and figure 57.

Overall, the predictor importance rate in each model shows significant results. Dew point temperature and the concentration of Carbon monoxide could be the primary input of predicting the concentration of PM2.5.

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

8.2.1. Data visualization:

All the data has been modified after step three(Data preparation) and step four(Data transformation). The steps have shown in the following charts, the main code of initialize the dataset, clean the dataset, process the data in figure 63 to 65.

Figure 63. Initialization of the dataset – Evidence from Spyder

```
# Read the dataset
dataset = pd.read_csv("D:/1UOA MIT/Semester Two/INFOSYS 722/ASSIGNMENT/zli786-Iteration3-OSAS/dataset.csv")
print(dataset.head())
print(dataset.columns)
# The size of the dataset
print(dataset.shape)
# The details of dataset
print(dataset.info())
# Generate descriptive statistics of the dataset
print(dataset.describe(include="all"))
# count the null value
null = dataset.isnull().sum()
print("Null value in dataset: \n", null)
```

Figure 64. Clean the dataset and remove the missing value – Evidence from Spyder

```
# Find the Missing values of more than 50%
remove = []
for i in dataset.columns:
    print("The columns need to be treated: ", i)
    if dataset[i].isnull().sum() > (dataset.shape[0] * 0.5):
        remove.append(i)
# Remove the variable
dataset1 = dataset.drop(remove, axis=1)
# Remove the missing value
dataset1 = dataset1.dropna()
print("The columns after removed the missing value:\n", dataset1.columns)
# Check the dataset if the missing value has been removed
check_null = dataset1.isnull().any()
print(check_null)
print(dataset1.shape)
```

Figure 65. Processing the data – Evidence from Spyder

```
# Remove the useless columns
dataset1 = dataset1.drop(['No', 'year', 'PM10', 'station', 'wd', 'day', 'PRES'], axis=1)
# Scale the data
dataset2 = pd.DataFrame(mpl.scale(dataset1.drop(['PM2.5'], axis=1)))
# Set the target column - PM2.5
target_column = dataset1['PM2.5']
# Reset the index of the columns starting from 0
target_column = target_column.reset_index()
target_column = target_column.iloc[:, 1:]
new_dataset = pd.concat([dataset2, target_column], axis=1)
# Standardization the target
new_dataset = new_dataset[new_dataset['PM2.5'] >= 0]
new_dataset['PM2.5'] = np.log2(new_dataset['PM2.5'] + 1)
new_dataset.rename(columns={0: 'month', 1: 'hour', 2: 'SO2', 3: 'NO2', 4: 'CO', 5: 'O3',
                            6: 'TEMP', 7: 'DEWP', 8: 'RAIN', 9: 'WSPM'}, inplace=True)
# print the new dataset
print(new_dataset)

# Store the variable we'll be predicting on
# Generate the training set.  Set random_state to be able to replicate results.
x_train, x_test, y_train, y_test = train_test_split(
    new_dataset.drop(['PM2.5'], axis=1), new_dataset['PM2.5'], test_size=0.33, random_state=123)
# Print the shapes of both sets.
print(x_train.shape)
print(x_test.shape)
```

Figure 66. Visualization of the data (the initial dataset) – Evidence from Spyder

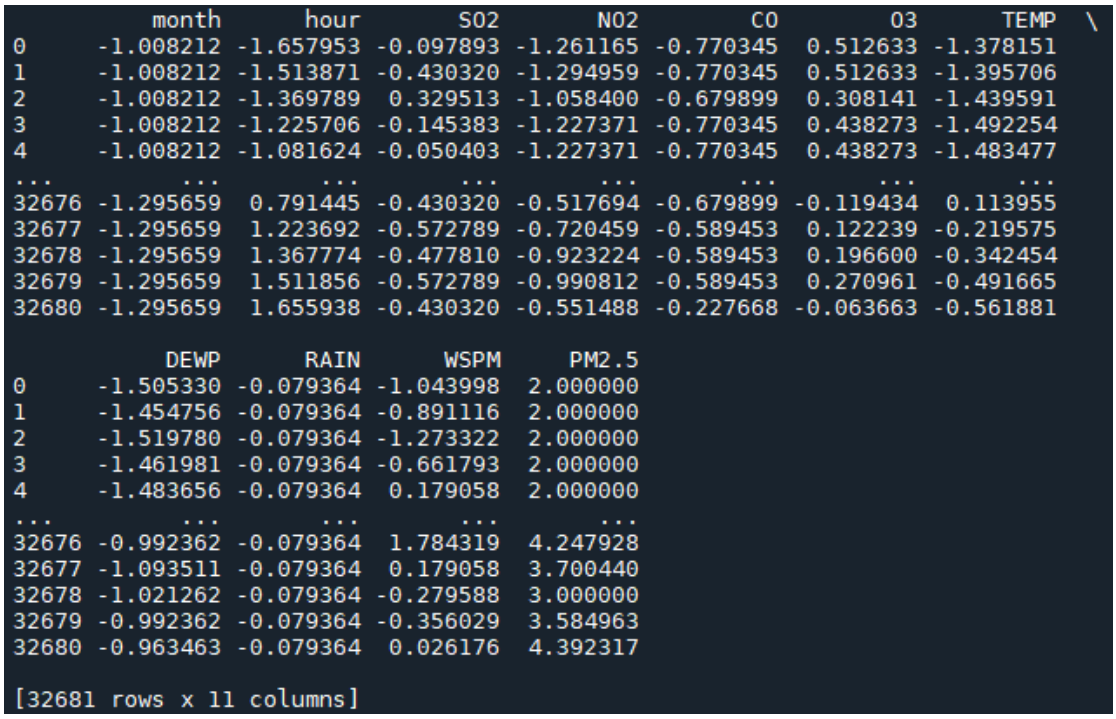
```
          No  year  month  day  hour  PM2.5  PM10   SO2   NO2     CO    O3
0          1  2013      3    1     0    3.0   6.0  13.0   7.0  300.0  85.0
1          2  2013      3    1     1    3.0   3.0   6.0   6.0  300.0  85.0
2          3  2013      3    1     2    3.0   3.0  22.0  13.0  400.0  74.0
3          4  2013      3    1     3    3.0   6.0  12.0   8.0  300.0  81.0
4          5  2013      3    1     4    3.0   3.0  14.0   8.0  300.0  81.0
...      ...   ...    ...  ...   ...    ...   ...   ...   ...    ...   ...
35059  35060  2017      2   28    19   28.0  47.0   4.0  14.0  300.0   NaN
35060  35061  2017      2   28    20   12.0  12.0   3.0  23.0  500.0  64.0
35061  35062  2017      2   28    21    7.0  23.0   5.0  17.0  500.0  68.0
35062  35063  2017      2   28    22   11.0  20.0   3.0  15.0  500.0  72.0
35063  35064  2017      2   28    23   20.0  25.0   6.0  28.0  900.0  54.0

       TEMP    PRES  DEWP  RAIN   wd  WSPM    station
0      -2.3  1020.8 -19.7   0.0    E   0.5  Changping
1      -2.5  1021.3 -19.0   0.0  ENE   0.7  Changping
2      -3.0  1021.3 -19.9   0.0  ENE   0.2  Changping
3      -3.6  1021.8 -19.1   0.0  NNE   1.0  Changping
4      -3.5  1022.3 -19.4   0.0    N   2.1  Changping
...     ...     ...   ...   ...  ...   ...         ...
35059  11.7  1008.9 -13.3   0.0  NNE   1.3  Changping
35060  10.9  1009.0 -14.0   0.0    N   2.1  Changping
35061   9.5  1009.4 -13.0   0.0    N   1.5  Changping
35062   7.8  1009.6 -12.6   0.0   NW   1.4  Changping
35063   7.0  1009.4 -12.2   0.0    N   1.9  Changping

[35064 rows x 18 columns]
```

In figure 66, shows the visualization of the initial dataset, after reducing and transforming the dataset, the new dataset has shown in figure 67. The changes has shown in figure 68 and 69, it is clear to see the changes.

Figure 67. Visualization of the data (after data standardization) – Evidence from Spyder

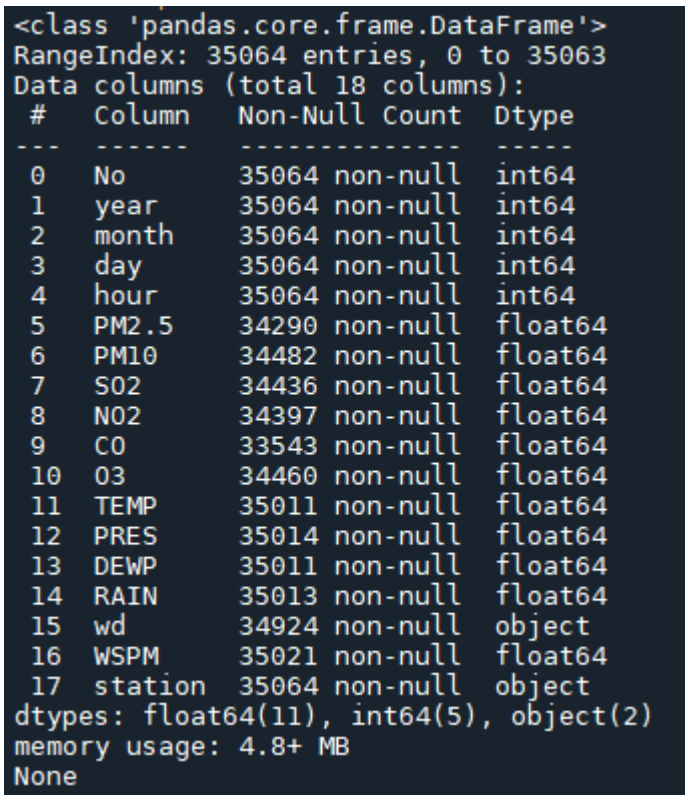


	month	hour	SO2	NO2	CO	O3	TEMP	\
0	-1.008212	-1.657953	-0.097893	-1.261165	-0.770345	0.512633	-1.378151	
1	-1.008212	-1.513871	-0.430320	-1.294959	-0.770345	0.512633	-1.395706	
2	-1.008212	-1.369789	0.329513	-1.058400	-0.679899	0.308141	-1.439591	
3	-1.008212	-1.225706	-0.145383	-1.227371	-0.770345	0.438273	-1.492254	
4	-1.008212	-1.081624	-0.050403	-1.227371	-0.770345	0.438273	-1.483477	
...	...	...	...	...	...	...	...	
32676	-1.295659	0.791445	-0.430320	-0.517694	-0.679899	-0.119434	0.113955	
32677	-1.295659	1.223692	-0.572789	-0.720459	-0.589453	0.122239	-0.219575	
32678	-1.295659	1.367774	-0.477810	-0.923224	-0.589453	0.196600	-0.342454	
32679	-1.295659	1.511856	-0.572789	-0.990812	-0.589453	0.270961	-0.491665	
32680	-1.295659	1.655938	-0.430320	-0.551488	-0.227668	-0.063663	-0.561881	

	DEWP	RAIN	WSPM	PM2.5
0	-1.505330	-0.079364	-1.043998	2.000000
1	-1.454756	-0.079364	-0.891116	2.000000
2	-1.519780	-0.079364	-1.273322	2.000000
3	-1.461981	-0.079364	-0.661793	2.000000
4	-1.483656	-0.079364	0.179058	2.000000
...	...	...	...	...
32676	-0.992362	-0.079364	1.784319	4.247928
32677	-1.093511	-0.079364	0.179058	3.700440
32678	-1.021262	-0.079364	-0.279588	3.000000
32679	-0.992362	-0.079364	-0.356029	3.584963
32680	-0.963463	-0.079364	0.026176	4.392317

[32681 rows x 11 columns]

Figure 68. Information of the dataset before analysis – Evidence from Spyder



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35064 entries, 0 to 35063
Data columns (total 18 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   No       35064 non-null  int64
 1   year     35064 non-null  int64
 2   month    35064 non-null  int64
 3   day      35064 non-null  int64
 4   hour     35064 non-null  int64
 5   PM2.5    34290 non-null  float64
 6   PM10     34482 non-null  float64
 7   SO2      34436 non-null  float64
 8   NO2      34397 non-null  float64
 9   CO       33543 non-null  float64
 10  O3       34460 non-null  float64
 11  TEMP     35011 non-null  float64
 12  PRES     35014 non-null  float64
 13  DEWP     35011 non-null  float64
 14  RAIN     35013 non-null  float64
 15  wd       34924 non-null  object
 16  WSPM     35021 non-null  float64
 17  station  35064 non-null  object
dtypes: float64(11), int64(5), object(2)
memory usage: 4.8+ MB
None
```

Figure 69. Information of the dataset after processing – Evidence from Spyder

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 32681 entries, 0 to 32680
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   month   32681 non-null  float64
 1   hour    32681 non-null  float64
 2   SO2     32681 non-null  float64
 3   NO2     32681 non-null  float64
 4   CO      32681 non-null  float64
 5   O3      32681 non-null  float64
 6   TEMP    32681 non-null  float64
 7   DEWP    32681 non-null  float64
 8   RAIN    32681 non-null  float64
 9   WSPM    32681 non-null  float64
 10  PM2.5   32681 non-null  float64
dtypes: float64(11)
memory usage: 3.0 MB
None
```

8.2.2. Model visualization:

Figure 70, 71, 72 show the model visualization of the models using in this project. The following graphs show the code of running the linear regression, random forest, and XGBoost model.

Figure 70. The code visualizations of Linear Regression model – Evidence from Spyder

```
#### linear regression model
# Initialize the model class.
linear_model = LinearRegression()
# Fit the model to the training data.
linear_model.fit(x_train, y_train)
importances = linear_model.coef_
# Returns the indices that would sort an array.
indices = np.argsort(importances)[::-1]
# Generate predictions for the test set
predictions = linear_model.predict(x_test)
# compute the explained variance score for the model
score = explained_variance_score(y_test,predictions)
# print the score
print("Accuracy score: ", score)
# Compute error between our test predictions and the actual values
linear_model_mse = mean_squared_error(predictions, y_test)
# print the mean squared error
print("Mean squared error: ", linear_model_mse)

# show the result in bar chart
plt.figure()
plt.title("Feature importances(Linear Regression)")
plt.barh(range(x_train.shape[1]), importances[indices]
         , color='r', align="center")
plt.yticks(range(x_train.shape[1]), x_train.columns[indices])
plt.ylim([-1, x_train.shape[1]])
plt.show()
```

Figure 71. The code visualizations of Random Forest model – Evidence from Spyder

```
####Random forest model
# Initialize the model class.
random_forest_model = RandomForestRegressor(random_state=123)
# Fit the model to the training data.
random_forest_model.fit(x_train, y_train)
# Compute the feature importance
importances = random_forest_model.feature_importances_
# Returns the indices that would sort an array.
indices = np.argsort(importances)[::-1]
# Generate predictions for the test set
predictions = random_forest_model.predict(x_test)
# Compute the accuracy score for the model
accuracy = explained_variance_score(y_test,predictions)
# Print the score
print("Accuracy: ", accuracy)
# Compute error between our test predictions and the actual values
random_forest_model_mse = mean_squared_error(predictions, y_test)
# Print the mean squared error
print("Mean squared error: ", random_forest_model_mse)
# Show the result in bar chart
plt.figure()
plt.title("Feature importances(Random Forest)")
plt.barh(range(x_train.shape[1]), importances[indices]
         , color='r', align="center")
plt.yticks(range(x_train.shape[1]), x_train.columns[indices])
plt.ylim([-1, x_train.shape[1]])
plt.show()
```

Figure 72. The code visualizations of XGBoost model – Evidence from Spyder

```
###XGBoost model
# Initialize the model class.
xgboost_model = xgb.XGBRFRegressor(seed=123)
# Fit the model to the training data.
xgboost_model.fit(x_train, y_train)
# Compute the feature importance
importances = xgboost_model.feature_importances_
# Returns the indices that would sort an array.
indices = np.argsort(importances)[::-1]
# Generate predictions for the test set
predictions = xgboost_model.predict(x_test)
# Compute the accuracy score for the model
accuracy = explained_variance_score(y_test,predictions)
# Print the score
print("Accuracy: ", accuracy)
# Compute error between our test predictions and the actual values
xgboost_model_mse = mean_squared_error(predictions, y_test)
#  Print the mean squared error
print("Mean squared error: ", xgboost_model_mse)
# plot the feature importances of the XGBoost model
plt.figure()
plt.title("Feature importances(XGBoost model)")
plt.barh(range(x_train.shape[1]), importances[indices]
         , color='r', align="center")
plt.yticks(range(x_train.shape[1]), x_train.columns[indices])
plt.ylim([-1, x_train.shape[1]])
plt.show()
```

8.2.3. Results and pattern visualization:

Figure 73 shows the packages and modules that used in this project.

In figure 74, 75, the details show the output of the Linear Regression model. The work from the Random Forest model has shown in figure 76, 77. The output from the XGBoost model has shown in figure 78, 79.

Figure 73. The packages import to code in python (pattern visualization) – Evidence from Spyder

```
import pandas as pd
import seaborn as sns
import numpy as np
import sklearn.preprocessing as mpl
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error,explained_variance_score
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
```

*Figure 74. Result visualization of **Linear Regression** model – Evidence from SPSS Modeler*

```
Linear Regression model evaluation - testing set:
Explained variance score:  0.6725590704043392
Mean squared error:  0.7670261150269483
Mean absolute error:  0.6747270652128363
Linear Regression model evaluation - training set:
Explained variance score:  0.667645309548268
Mean squared error:  0.7681235298866673
Mean absolute error:  0.6751280389355763
```

*Figure 75. Result visualization of **Linear Regression** model – Evidence from SPSS Modeler*

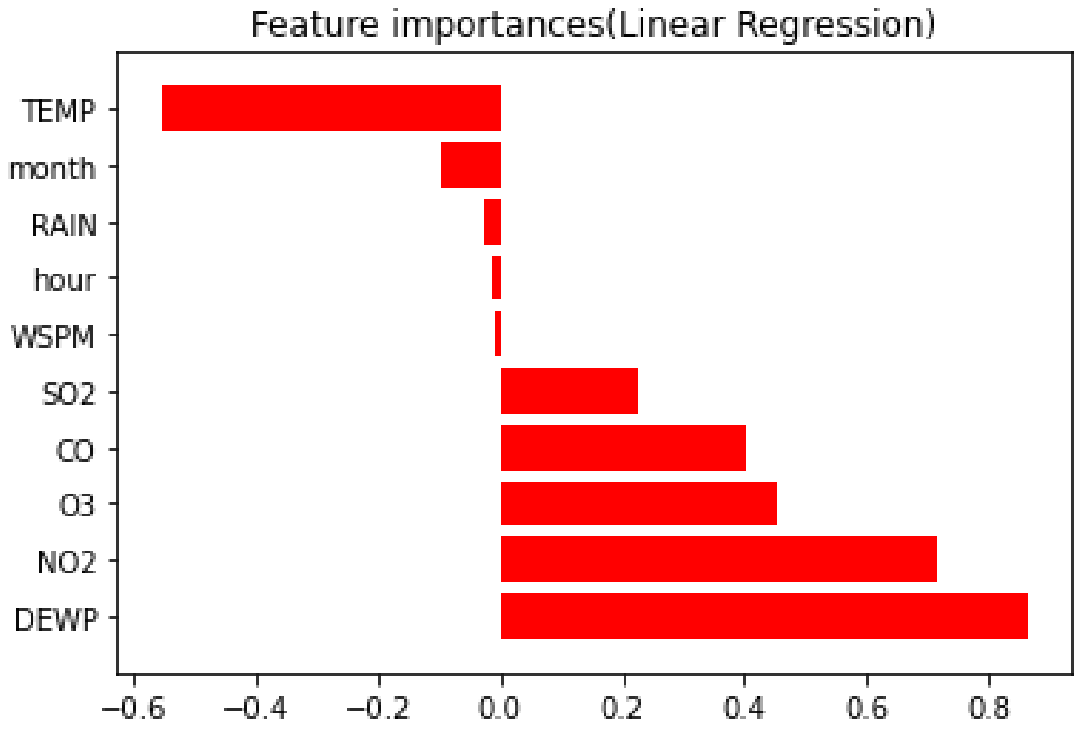


*Figure 76. Results visualization of **Random Tree** model – Evidence from SPSS Modeler*

```
 Random Forest model evaluation - testing set:
Explained variance score:  0.8396240967137185
Mean squared error:  0.37602003577903165
Mean absolute error:  0.4392414422322839
Random Forest model evaluation - training set:
Explained variance score:  0.9704067450401215
Mean squared error:  0.06840039528534199
Mean absolute error:  0.17477827421186698
```

*Figure 77. Results visualization of **Random Tree** model – Evidence from SPSS Modeler*

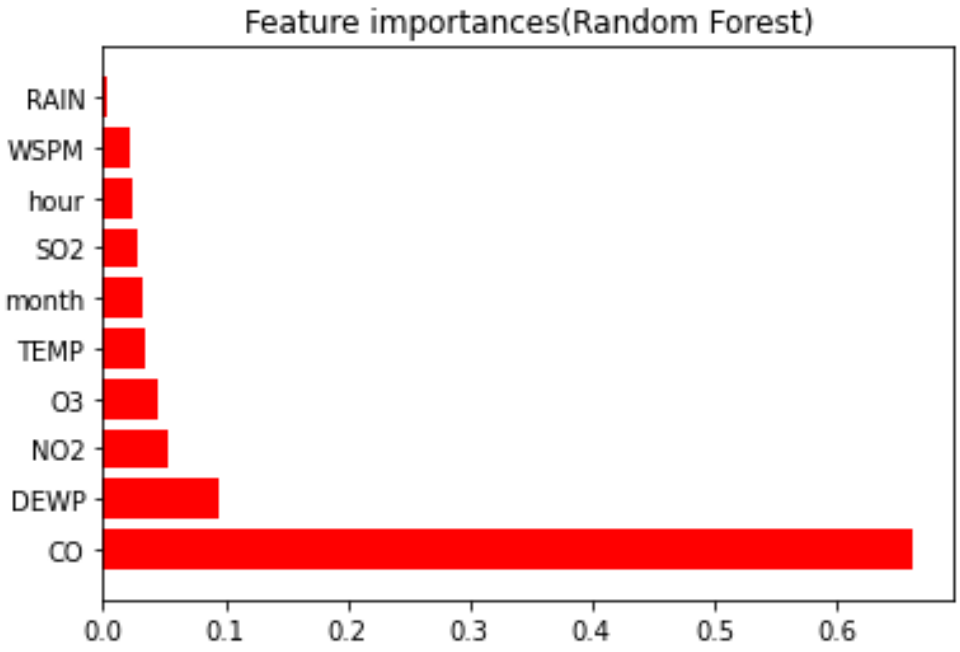
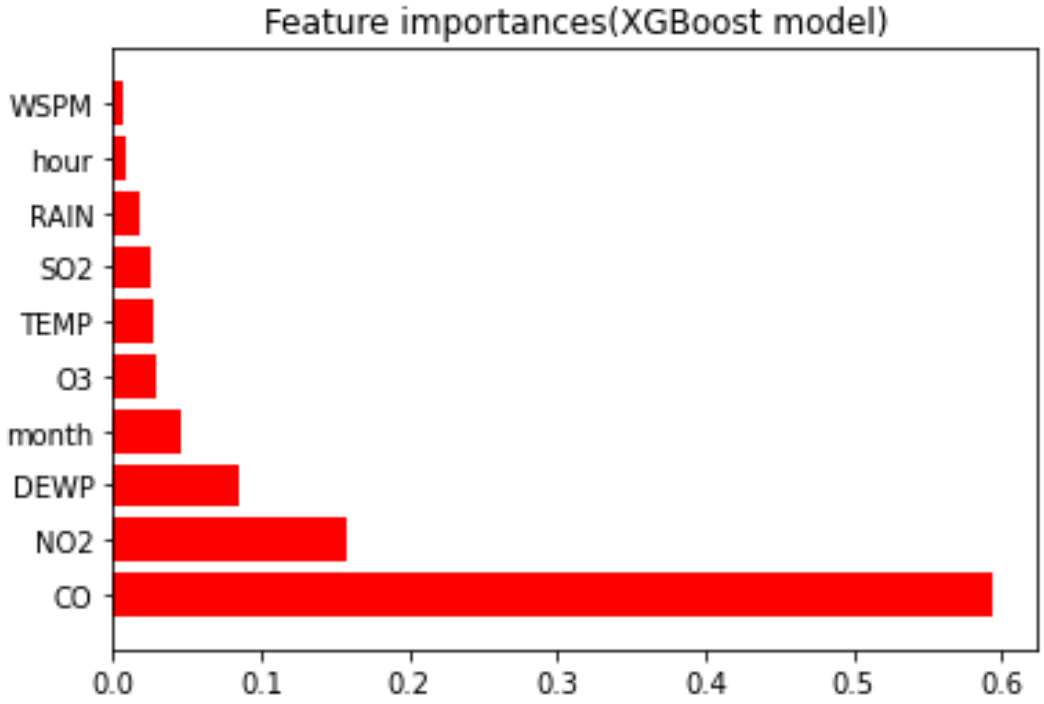


*Figure 78. Result visualization of **XGBoost Tree** model – Evidence from SPSS Modeler*

```
XGBoost model evaluation - testing set:
Explained variance score:  0.7803440431587867
Mean squared error:  0.5145518676592501
Mean absolute error:  0.5323117867954301
XGBoost model evaluation - training set:
Explained variance score:  0.7859695672965291
Mean squared error:  0.4946578492888901
Mean absolute error:  0.5179261294057668
```

*Figure 79. Result visualization of **XGBoost Tree** – Evidence from SPSS Modeler*



8.3. Interpret the results, models, and patterns

8.3.1. Results and models Interpretation

- Linear Regression model:

  The output from the Linear Regression model has shown in figure 74, 75.

  The model's accuracy score of the Linear Regression model (with the target is PM2.5) is 67%, which is a satisfying result (shown in figure 74). As regards to the predictor importance of this model (see figure 75), the highest-ranking is the DEWP (dew point temperature), with the percentage is 0.83, is the most important feature; the second highest-rated is NO2 (nitrogen dioxide), with the rate is 0.66; the third and fourth is the TEMP (temperature features) and O3 (Ozone), with the feature importance are -0.55 and 0.43 respectively; CO (Carbon monoxide) (0.40) and SO2 (Sulfur dioxide) (0.21) are the rest of chemical elements. It is worth noting that the time features do not show enough importance in this model, with only -0.1 and -0.02 for month and hour, respectively. The WSPM (Wind speed) has the lowest importance to the PM2.5, only have -0.01.

  In figure 75, the statistics result for the Linear Regression model shows the explained variance score is 66%, which is satisfied; Mean squared Error is 0.76, relatively big value for the error, and mean absolute error is 0.675, which is acceptable but the number is also big. The Occurrences of the dataset is 21896, which is 67% of the total number 32681 to avoid overfit.

- Random Forest model:

  In figure 76, 77, the details show the output of Random Forest model.

  In this model, the most important predictor is 'CO' (Carbon monoxide), with the figure is 0.67; the second significant element is 'DEWP' (Dew point temperature), with the figure is 0.09; the 'DEWP' is also on the second level of substantial predictor, with the model is 0.09 approximately. The effect of other chemical elements also plays a role, the value of NO2, O3, SO2 is 0.06, 0.05, 0.03, respectively. The time variables such as month and hours, does not show a big difference, month predictor becomes the less important, with the figure is 0.035; however, the value of hours predictor is estimated at 0.025.

  Figure 77 shows the statistics result for the Random Forest model shows the explained variance score is 97%, which is satisfied and is the highest score for the three models; Mean squared Error is 0.068, relatively low level for the error, and mean absolute error is 0.174, which is acceptable because the number is quite small. The Occurrences of the dataset is 21896, which is 67% of the total number 32681 to avoid overfit.

- XGBoost model:
  The output from the XGBoost model has shown in figure 78, 79.
  In this model, the most important predictor is same with the result in Random Forest model, CO (Carbon monoxide) become the most important feature for this project, with the figure is 0.6; the second most significant predictor is 'NO2' (nitrogen dioxide), with the figure is 0.17; the lowest figure is 0.01, which is the 'Rain' (Precipitation), most unimportant predictor in this model. The influence of chemical elements on predictions is not important enough than CO and NO2, the value of SO2 and O3 is 0.025, 0.03. Additionally, the time influencing makes some changes, the month predictor becomes the fourth in this model, with 0.045 predictor importance, however, the hour predictor does not change its' level against the other two models, with the figure 0.01.

  In figure 79, shows the statistics result for the XGBoost model shows the explained variance score is 78%, which is satisfied and is the second highest score for the three models; Mean squared Error is 0.494, relatively median level for the error, and mean absolute error is 0.517, which is acceptable but the number is also big. The Occurrences of the dataset is 21896, which is 67% of the total number 32681 to avoid overfit.

To sum up, the Linear Regression, Random Forest, and XGBoost models are generally consistent with this project's results. Compounds in the air have a significant effect on the concentration of PM2.5. Specifically, CO, NO2,and O3 are the most important influencing factors, and SO2, and other Sulfuric compounds have a moderate and weak influence on PM2.5 emissions. In addition, the impact of the external environment on PM2.5 concentration was more vulnerable than that of the compound, but DWEP was the most significant factor affecting PM2.5 emissions shows in Linear Regression model.

8.3.2. patterns Interpretation

Using the .corr() function to automatically select the essential and relevant predictors, as mentioned in step 4, provides the features' valuable and high correlation attributes. Use .train_test_split() function to separate the dataset and avoid overfitting. Use .scale() function to do the data standardization to finish the data transformation. In step 6, access the sklearn package, metrics module to calculate the accuracy score and mean squared error. Simultaneously, use sklearn.linear_model and sklearn.ensemble and xgboost module to import the most effective algorithms: The Linear Regression, the Random Forest model, the XGBoost tree model, for the regression problems.

(1) Linear Regression is a statistical analysis method that uses Regression analysis in mathematical statistics to determine the interdependent quantitative relationships between two or more variables (Multiple Linear Regression, 2009). In this technique, the dependent variable is continuous. The independent variable can be continuous or discrete. The most important thing to use this model is to ensure that the relationship between independent and dependent variables must be Linear.

(2) Based on the decision tree, Random Forest introduces Random feature selection in decision tree training. It can be summarized into four parts: random selection of samples, the random selection of feature attributes, construction of decision tree, unexpected forest results mean. As the number of learners increases, the random forest generally converges to a lower generalization error. The random forest training efficiency is also high because, in constructing a single decision tree, the random forest uses the 'random' feature number, and only the subset of feature is considered.

(3) XGBoost uses the model's negative gradient on the data to approximate the residual to fit the residual. At the same time, XGBoost improves the model's loss function and adds regular model complexity. Therefore, it can automatically use the multithreading of CPU to carry out a parallel computation, and at the same time, the accuracy of the algorithm is also improved. XGBoost also supports linear classifiers, equivalent to l1-l2 regularized logistic regression (classification) or linear regression (regression).

8.4. Assess and evaluate results, models, and patterns

According to Evaluation indicators for regression models (2019), the commonly used evaluation indexes in the regression model are:

- The Mean Squared Error

  The Mean Squared Error calculates the Squared Error between the predicted value and the actual value. The smaller the Mean Squared Error is, the better result we want.

- The Mean Absolute Error

  Mean Absolute Error is used to describe the difference between a predicted value and an actual value. The smaller Mean Absolute Error is, the better result we got.

- Explained variance score

  The definition of variance is the average squared distance from the average. Explained variance score means that we can use one or more variables to predict things more accurately than before. The maximum value of the Explained variance score is 1, and if the value is close to 1, the most accurate result we got.

Figure 80. Linear Regression model evaluation – Evidence from Spyder

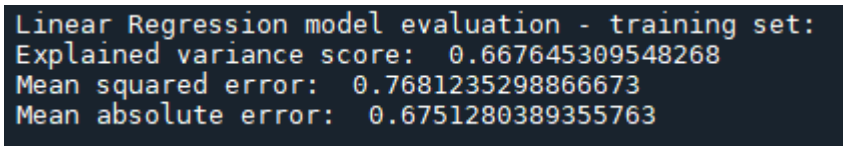


Figure 81. Random Forest model evaluation – Evidence from Spyder

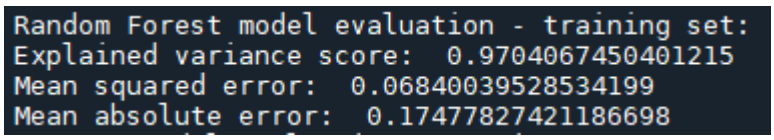
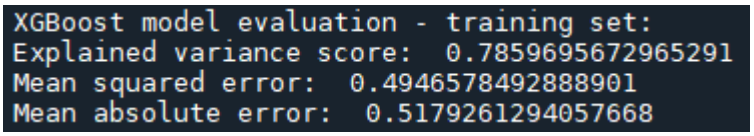


Figure 82. XGBoost model evaluation – Evidence from Spyder



In figure 80, the statistics result for the Linear Regression model shows the explained variance score is 66%, which is satisfied; Mean squared Error is 0.76, relatively big value for the error, and mean absolute error is 0.675, which is acceptable but the number is also big. The Occurrences of the dataset is 21896, which is 67% of the total number 32681 to avoid overfit.

Figure 81 shows the statistics result for the Random Forest model shows the explained variance score is 97%, which is satisfied and is the highest score for the three models; Mean squared Error is 0.068, relatively low level for the error, and mean absolute error is 0.174, which is acceptable because the number is quite small. The Occurrences of the dataset is 21896, which is 67% of the total number 32681 to avoid overfit.

In figure 82, shows the statistics result for the XGBoost model shows the explained variance score is 78%, which is satisfied and is the second highest score for the three

models; Mean squared Error is 0.494, relatively median level for the error, and mean absolute error is 0.517, which is acceptable but the number is also big. The Occurrences of the dataset is 21896, which is 67% of the total number 32681 to avoid overfit.

Base on the description of the Explained variance score, the Mean Squared Error, and the Mean Absolute Error, and the result shows in figure 80, 81, 82, we can see that The Random Forest model in this project has the most accuracy score which is 0.97, really close to 1; and it also have the lowest mean squared error, mean absolute error, with 0.068 and 0.174 respectively. The evaluation indicators make the Random Forest model becomes the most trusted model, the most important predictor is CO in this model, which is clearly showing the results. However, the impact of different models has demonstrated that the DEWP (dew point temperature) is an important predictor with PM2.5.

The time variables, month, and hour also play a role in the models' results.

In terms of the data mining success criteria, we have found that the DEWP temperature is one of the most significant feature associates with the concentration of PM2.5, the chemical elements in this dataset (SO2, O3, NO2, CO) also became the most critical predictor in some of the models, especially the CO. The Random Forest model is the most satisfied model in this project, as its high accuracy score and low error.

In conclusion, in my opinion, the PM2.5 concentration is associated with the DEWP temperature, as the lower DEWP is, the higher concentration of PM2.5 is. Similarly to the month and hour features, the concentration of PM2.5 is much higher in winter and spring (from December to May) than the other two seasons. Simultaneously, the temperature at night is lower than the temperature in the daytime; consequently, the concentration of PM2.5 would higher in the night. The concentration of CO is also affecting the concentration of PM2.5, the record with high concentration of CO also shows high concentration of PM2.5.

8.5. Iterate prior steps (1 – 7) as required

8.5.1. Iterate business/situation understanding

The PM2.5 weather situation in Changping(Beijing) would not change significantly for the time being, and the project prediction target would not change. Additionally, train the data set, build, and generate the model and predict the PM2.5 concentration according to the other relevant weather index.

8.5.2. data understanding

The data set is still the weather situation in Changping(Beijing). The data quality can be observed through the specific analysis and understanding of the data by processing the database.

8.5.3. Data preparation

Processing the missing values and finding the desired data column against the target could ensure data prediction quality.

8.5.4. Data transformation

According to the target, filter the unwanted rows, then convert the used columns to the agreed range to ensure fairness.

8.5.5. Data-mining method selection

Because the variables in the project are both input and output, supervised learning is required. The regression models are appropriate for our project because the prediction target is continuous.

8.5.6. Data-mining algorithms selection

Python sklearn package provides several reliable algorithms. After investigation, all these algorithms show their advantages. It is necessary to compare the results of each algorithm to verify our prediction.

8.5.7. Data-mining

Data mining results are as follows: for the random forest model, CO is the most critical factor affecting PM2.5; For the XGBoost Tree model, the temperature is the most crucial factor affecting PM2.5.

These steps have not changed compared to before; this project has been iterated many times and repeated various measures to ensure that the model is significant.

References

Boyanov, A. (2016, February 15). Python Design Patterns: For Sleek and Fashionable Code. Retrieved May 13, 2020, from https://www.toptal.com/python/python-design-patterns

Data for Data Mining. (n.d.). Principles of Data Mining, 11–21. Doi: 10.1007/978-1-84628-766-4_1.

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From Data Mining to Knowledge Discovery in Databases. *AI Magazine, 17*(3), 37-54.

IISD. (2018). *WHO Global Conference Recommends Reducing Deaths from Air Pollution by Two-Thirds by 2030.* Retrieved from https://sdg.iisd.org/news/who-global-conference-recommends-reducing-deaths-from-air-pollution-by-two-thirds-by-2030/

IQAIR. (2020). *2019 WORLD AIR QUALITY REPORT*. Retrieved from https://www.iqair.com/world-most-polluted-cities/world-air-quality-report-2019-en.pdf

Random Forest, GBDT and XGBoost. (n.d.). Retrieved from https://blog.csdn.net/yingfengfeixiang/article/details/80210145?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromBaidu-3

Regression Trees. (2013). *Regression Methods for Medical Research*, 204-235. doi:10.1002/9781118721957.ch9

Team Airveda. (2017). *What Is PM2.5 and Why Is It Important?* Retrieved from https://www.airveda.com/blog/what-is-pm2-5-and-why-is-it-important

World Health Organization. (2018). *Ambient (outdoor) air pollution.* Retrieved from https://www.who.int/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health

World-wide Air Quality Monitoring Data Coverage. (n.d.). Retrieved from https://aqicn.org/sources/

Zhang, S., Guo, B., Dong, A., He, J., Xu, Z. & Chen, S.X. (2017). Cautionary Tales on Air-Quality Improvement in Beijing. *Proceedings of the Royal Society A, 473*(2205), 20170457.

Sathya, R., & Abraham, A. (2013). Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification. *International Journal of Advanced Research in Artificial Intelligence, 2*(2). doi:10.14569/ijarai.2013.020206

Seif, G. (2018, March 5). Selecting the best Machine Learning algorithm for your regression problem. Retrieved from https://towardsdatascience.com/selecting-the-best-machine-learning-algorithm-for-your-regression-problem-20c330bad4ef

"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.htmlLinks to an external site.).

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."