The Relationships Between PM2.5 Concentration and Meteorology Factors in Beijing

Analysis and Prediction

Peter He (UPI: qhe231)

qhe231@aucklanduni.ac.nz

GitHub Link for Datasets and PySpark Code:
https://github.com/qhe231/Iteration4_qhe231_Peter_He

Contents

1. Situation Understanding

1.1 Situation Objectives

PM2.5, which refers to the particular matter with a diameter less than 2.5 micrometres, causes premature death from diseases that related to lung and heart (Bliss Air, n.d.). Especially in China, PM2.5 pollution has attracted public attention due to its seriousness. In 2013, it caused the death of 1.37 million adults (Ding, Xing, Wang, Liu, & Hao, 2019). Wang et al. (2020) considered PM2.5 as the primary component of air pollutants, although it only occupies a little of the atmosphere.

United Nations (2016) appealed that the mortality rate from atmospheric pollution should be adequately reduced by 2030. Since PM2.5 is the main part of air pollutants, lowering its concentration can lead to better air quality, thereby reducing the related premature death.

Beijing government has made an effort to reduce PM2.5 by blocking the source. For example, power stations based on coal fire are shut down, while citizens are forbidden from burning coal for Heat. These actions effectively reduced PM2.5 concentration in the city. However, they are controversial and expensive (Hao, 2018).

Therefore, this study aims to achieve the following objectives:

- Find out meteorology elements that affect PM2.5 concentration in Beijing.
- Design an effective solution to control PM2.5 concentration in Beijing based on the find so that we can reduce, thereby helping lower the premature mortality rate related to air pollution in this city.

Briefly, the study will be estimated as an accomplishment if:

- Find the meteorology elements that influence PM2.5 concentration in Beijing.
- Use the model to predict PM2.5 concentration.
- Finish the study within two weeks.

1.2 Situation Assessment

1.2.1 Resource Inventory

1.2.1.1 Research Hardware Resources

In this project, I am going to use PySpark to analyze the data and to submit my code and datasets in GitHub. Therefore, I need access to AWS EC2, SSH, Jupyter Notebook and GitHub. My original plan was to use the lab machines in the university as they already have the software. However, due to the spread of COVID-19, all of us have to study and work from home. Fortunately, this course provides the EC2 instance. Besides, I have a MacBook whose terminal can be used as SSH to access Jupyter Notebook. I also have a GitHub account that can be used to submit the code and the datasets. Everything seems working well. In case my laptop may go wrong, I prepared another Microsoft PC which I can use to access these systems.

Besides, the internet connection is necessary since everyone is working from home. Luckily, the internet at my place works fine and allows me to search for essential information which I need for this project from the website, and to ask the lecturer and tutors questions through Piazza or Emails.

1.2.1.2 Data Sources and Knowledge Stores

Data will be collected from websites that provide datasets to the public, which is the requirement of this course. The tutors have already provided us with a variety range of websites where we can find useful datasets for our study, such as kaggle.com or data.govt.nz.

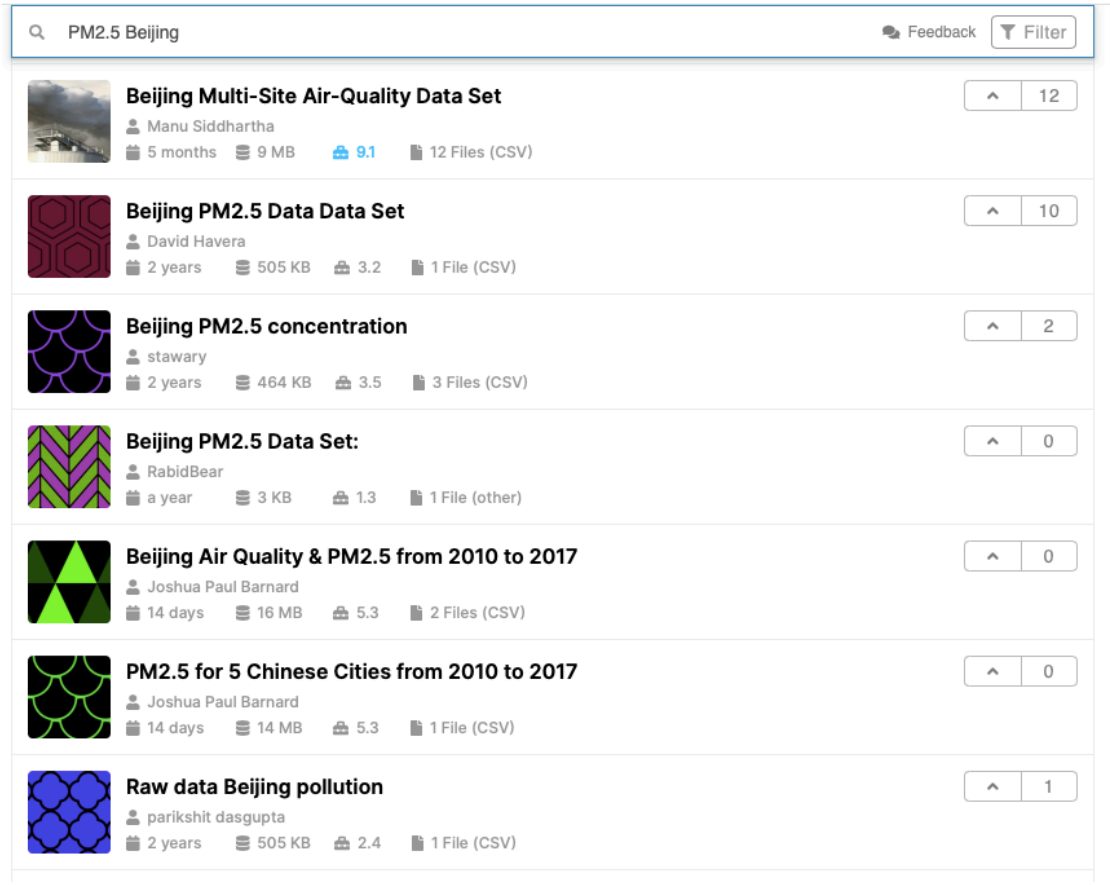
After Exploring these websites, I have found some datasets related to Beijing PM2.5 concentration on UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets.php) as shown in *Diagram 1.2.1*. Also, *Diagram 1.2.2* listed Beijing PM2.5 related datasets in Kaggle (https://www.kaggle.com/datasets). These datasets are mostly regression problem type and are formatted in .csv or .xlsx files, which can be processed by PySpark. Therefore, there are enough open-sourced datasets

for me to do data mining in this iteration. Later in the data understanding chapter, I will choose the most appropriate datasets among them.

Diagram 1.2.1 Related Datasets in UCI Machine Learning Repository

Name	Data Types	Default Task	Attribute Types	# Instances	# Attributes	Year
Beijing Multi-Site Air-Quality Data	Multivariate, Time-Series	Regression	Integer, Real	420768	18	2019
Beijing PM2.5 Data	Multivariate, Time-Series	Regression	Integer, Real	43824	13	2017
PM2.5 Data of Five Chinese Cities	Multivariate, Time-Series	Regression	Integer, Real	52854	86	2017

Diagram 1.2.2 Related Datasets in Kaggle



Since all these datasets are open to the public and are free for any users to download, there will be no security or confidential problems. I will download useful datasets and store them in my laptop for data mining. Besides, there is no plan to buy extra datasets as this project is for study only.

1.2.1.3 Personnel Resources

This is a personal project to meet the requirement of the course, so I will do all the jobs individually from situation understanding to interpretation. However, as I am new to the data mining area and have no experience with PySpark before, I will seek support from the lecturer, tutors and classmates when necessary. Piazza, Zoom and email will be the tools due to the closure of the university, which is required.

1.2.2 Requirements, Assumptions and Constraints

1.2.2.1 Requirements

As discussed in the situation understanding chapter, the basic requirements of this iteration are to find out the relationships between PM2.5 concentration and the meteorology information in Beijing, and to design useful solution to decrease PM2.5 concentration so that air-pollution-related premature mortality can be reduced.

There is no requirement that restricts the data and the results since the data are open to the public on websites without security or confidential concern. The results will only be submitted for assessment. There is no plan to publish it anywhere.

1.2.2.2 Assumptions

Since this is a project for study purpose, no economic element is going to affect the data mining process.

As for data quality, I assume that the datasets are with different qualities as they are open online resources. However, I assume some of them are high-quality that I can use for data mining. Anyway, the data quality will be assessed in the data quality verification chapter. If the quality of the datasets is not enough for the data mining process, the data will be cooked for the study purpose.

I assume the marker wants to see the whole iteration being clearly presented in this paper, including the models, the results and the entire process. Therefore, I will make this report as detailed as possible to illustrate everything I have done.

1.2.2.3 Constraints

There is no legal or confidential constraint for this project because all the data are free to download from the websites. The only constraint for this project is that we are not allowed to collect our own data (The University of Auckland, n.d.)

1.2.3 Risks and Contingencies

1.2.3.1 Scheduling

As I am taking four courses this semester, there will be many other assignments besides INFOSYS 722. Also, the spread of COVID-19 in the country makes everything unsure. Therefore, it is slightly possible that I may not finish the project on time. However, doing proper time management and setting an appropriate plan can effectively help deal with that.

1.2.3.2 Data

Insufficient data quality and quantity is another risk for this iteration. Since the data are from online resources, its quality and quantity cannot be guaranteed. Fortunately, cooking data is allowed in this course. Therefore, if the qualities of all original datasets are poor, I will cook up the data to make them more suitable for the business and data mining objectives in the data preparation chapter.

1.2.3.3 Results

Another risk is the results. PM2.5 concentration may not be affected by the meteorology factors. If this situation happened, it means that the direction of the entire project is wrong. I will have to find some other datasets to do

another data mining. Besides, the models may give inaccurate detection. In this case, I will try other models to improve the correctness.

1.3 Data Mining Objectives

As indicated in the situation objectives chapter, this study is trying to find out the meteorology factors that influencing PM2.5 concentration in Beijing. In order to achieve this, the data mining process will need to use models to find out whether they have relationships and how the meteorology factors are affecting PM2.5 concentration in the city. For example, the wind from one combined direction with faster combined wind speed may cause PM2.5 concentration lower.

Besides, the purposes of this study include designing an effective solution to lower PM2.5 concentration to protect people's health. To keep PM2.5 concentration low, we have to detect the concentration first based on meteorology information and find out the main reasons why it increases or decreases. Then, according to the main reasons, we can find out the solution to affect the meteorology factor, thereby lowering PM2.5 concentration. Therefore, the related premature mortality rate can be better controlled.

Based on the situation objectives discussed above, the data mining objectives for this study are:

- Use the meteorology information and the existed data to predict Beijing PM2.5 concentration.
- Use the historical data to analyze how the meteorology elements influence PM2.5 concentration in Beijing.
- Use the meteorology information to analyze which elements will be the main reasons for the increase or decrease of PM2.5 concentration in Beijing.

1.4 Project Plan

Diagram 1.4.1 shows the brief plan of duration (in days) for each step of this study. Meanwhile, while *Diagram 1.4.2* demonstrated the details of the study plan, including the content, the duration, the resources that help cater for the iteration, and the risks of each step.

Diagram 1.4.1 Brief Plan in Gantt Chart

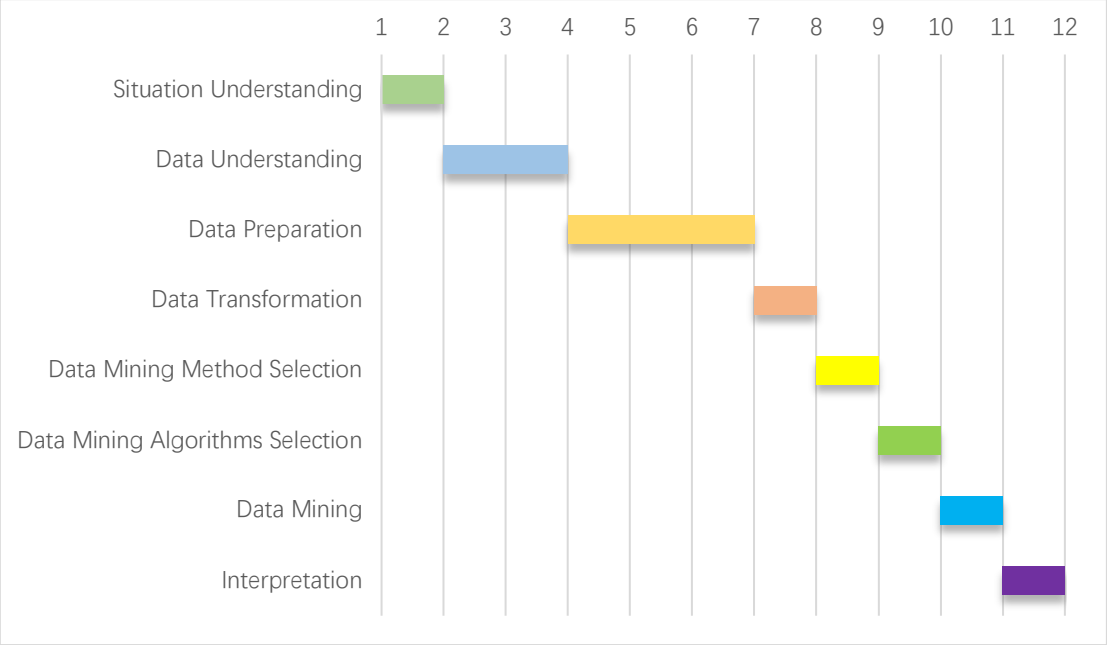


Diagram 1.4.2 Detailed Project Plan

Phrase	Time	Resources	Risks
Situation Understanding: Understand and assess the situation, followed by setting up goals and plan	1 day	Literature Reviews, Websites Providing Data Sets, Lectures	Situation change
Data Understanding: Collect, describe and explore the data, then verify their quality	2 days	Datasets, EC2, SSH, Jupyter Notebook, PySpark Libraries, Tutors, Lectures	Data Problems, Technology Problems

Diagram 1.4.2 Detailed Project Plan – Continued

Phrase	Time	Resources	Risks
Data Preparation: Select and construct the data to make sure they are clean and integrated as required	3 days	Datasets, EC2, SSH, Jupyter Notebook, PySpark Libraries, GitHub, Tutors, Lectures	Data Problems, Technology Problems
Data Transformation: Reduce and project the data	1 day	Datasets, EC2, SSH, Jupyter Notebook, PySpark Libraries, GitHub, KDD Process Model, Tutors, Lectures	Data Problems, Technology Problems
Data Mining Method Selection: Select data mining methods to match the discussed data mining objectives	1 day	Datasets, Literature Review, Tutors, Lectures	Wrong Methods Selection, Inability to Find Appropriate Methods
Data Mining Algorithms Selection: Choose data mining algorithms based on exploratory analysis, followed by building suitable models and choosing appropriate arguments	1 day	Datasets, Literature Review, Machine Learning Cheat-sheet, Tutors, Lectures	Wrong Algorithms Selection, Inability to Find Appropriate Algorithms

Diagram 1.4.2 Detailed Project Plan – Continued

Phrase	Time	Resources	Risks
Data Mining: Create and justify test designs, then conduct data mining	1 day	Datasets, EC2, SSH, Jupyter Notebook, PySpark Libraries, GitHub, KDD Process Model, Tutors, Lectures	Technology Problems
Interpretation: study, visualise and interpret the data, results, models and patterns, also appraise the results	1 day	Datasets, EC2, SSH, Jupyter Notebook, PySpark Libraries, GitHub, Literature Review, KDD Process Model, Tutors, Lectures	Inability to Implement results

2. Data Understanding

2.1 Initial Data Collection

After overviewing many datasets from the recommended websites, I chose two datasets related to PM2.5 in Beijing to be used for this study. They both include useful hourly PM2.5 concentration in Beijing from 1/1/2010 to 31/12/2014, as well as relevant meteorology information. They will be merged into one table, which will have more detailed meteorology information.

2.1.1 Beijing PM2.5 Data

(https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data)

This dataset, which was provided by Chen (2017) from Peking University, contains hourly PM2.5 concentration at the US Embassy in Beijing, as well as the meteorology information for each hour, including dew point, temperature, pressure, combined wind direction, cumulated wind speed, and cumulated hours of snow and rain from 1/1/2010 to 31/12/2014.

2.1.2 PM2.5 Data of Five Chinese Cities

(https://archive.ics.uci.edu/ml/datasets/PM2.5+Data+of+Five+Chinese+Cities)

This is another dataset provided by Chen (2017). It includes similar attributes with the first dataset. However, it has some information which the former one does not, like season, humidity and precipitation. This dataset also contains PM2.5 concentration in other three areas in Beijing beside the US Embassy. Specifically, only the data of Beijing from 1/1/2010 to 31/12/2014 in this dataset will be used, while the data of the other four cities will be removed for this study.

2.2 Data Description

As mentioned above, the two datasets contain hourly PM2.5 concentration together with corresponding meteorology information. Therefore, it is suitable for the study which aims to analyze the relationships between them and to predict PM2.5 concentration from the meteorology information.

Diagrams 2.2.1 and 2.2.2 demonstrates overviews of the two datasets, respectively.

Diagram 2.2.1 Overview of *Beijing PM2.5 Data*

```
#read "Beijing PM2.5 Data" and show it.
dataset1 = spark.read.csv('./Datasets/PRSA_data_2010.1.1-2014.12.31.csv',
                          inferSchema = True, header = True)
dataset1.show()
```

```
+---+----+-----+---+----+-----+----+-----+------+----+-----+---+---+
| No|year|month|day|hour|pm2.5|DEWP| TEMP|  PRES|cbwd|  Iws| Is| Ir|
+---+----+-----+---+----+-----+----+-----+------+----+-----+---+---+
|  1|2010|    1|  1|   0|   NA| -21|-11.0|1021.0|  NW| 1.79|  0|  0|
|  2|2010|    1|  1|   1|   NA| -21|-12.0|1020.0|  NW| 4.92|  0|  0|
|  3|2010|    1|  1|   2|   NA| -21|-11.0|1019.0|  NW| 6.71|  0|  0|
|  4|2010|    1|  1|   3|   NA| -21|-14.0|1019.0|  NW| 9.84|  0|  0|
|  5|2010|    1|  1|   4|   NA| -20|-12.0|1018.0|  NW|12.97|  0|  0|
|  6|2010|    1|  1|   5|   NA| -19|-10.0|1017.0|  NW| 16.1|  0|  0|
|  7|2010|    1|  1|   6|   NA| -19| -9.0|1017.0|  NW|19.23|  0|  0|
|  8|2010|    1|  1|   7|   NA| -19| -9.0|1017.0|  NW|21.02|  0|  0|
|  9|2010|    1|  1|   8|   NA| -19| -9.0|1017.0|  NW|24.15|  0|  0|
| 10|2010|    1|  1|   9|   NA| -20| -8.0|1017.0|  NW|27.28|  0|  0|
| 11|2010|    1|  1|  10|   NA| -19| -7.0|1017.0|  NW| 31.3|  0|  0|
| 12|2010|    1|  1|  11|   NA| -18| -5.0|1017.0|  NW|34.43|  0|  0|
| 13|2010|    1|  1|  12|   NA| -19| -5.0|1015.0|  NW|37.56|  0|  0|
| 14|2010|    1|  1|  13|   NA| -18| -3.0|1015.0|  NW|40.69|  0|  0|
| 15|2010|    1|  1|  14|   NA| -18| -2.0|1014.0|  NW|43.82|  0|  0|
| 16|2010|    1|  1|  15|   NA| -18| -1.0|1014.0|  cv| 0.89|  0|  0|
| 17|2010|    1|  1|  16|   NA| -19| -2.0|1015.0|  NW| 1.79|  0|  0|
| 18|2010|    1|  1|  17|   NA| -18| -3.0|1015.0|  NW| 2.68|  0|  0|
| 19|2010|    1|  1|  18|   NA| -18| -5.0|1016.0|  NE| 1.79|  0|  0|
| 20|2010|    1|  1|  19|   NA| -17| -4.0|1017.0|  NW| 1.79|  0|  0|
+---+----+-----+---+----+-----+----+-----+------+----+-----+---+---+
only showing top 20 rows
```

Diagram 2.2.2 Overview of *PM2.5 Data of Five Chinese City* (The Part of Beijing)

```
#read "PM2.5 Data of Five Chinese Cities (the Part of Beijing)" and show it.
dataset2 = spark.read.csv('./Datasets/BeijingPM20100101_20151231.csv',
                          inferSchema = True, header = True)
dataset2.show(5)
```

```
+---+----+-----+---+----+------+---------+-------------+--------------+----------+----+----+
----+----+----+-----+-------------+-----+
| No|year|month|day|hour|season|PM_Dongsi|PM_Dongsihuan|PM_Nongzhanguan|PM_US Post|DEWP|HUMI|
PRES|TEMP|cbwd|  Iws|precipitation|Iprec|
+---+----+-----+---+----+------+---------+-------------+--------------+----------+----+----+
----+----+----+-----+-------------+-----+
|  1|2010|    1|  1|   0|     4|       NA|           NA|            NA|        NA| -21|  43|
1021| -11|  NW| 1.79|          0.0|  0.0|
|  2|2010|    1|  1|   1|     4|       NA|           NA|            NA|        NA| -21|  47|
1020| -12|  NW| 4.92|          0.0|  0.0|
|  3|2010|    1|  1|   2|     4|       NA|           NA|            NA|        NA| -21|  43|
1019| -11|  NW| 6.71|          0.0|  0.0|
|  4|2010|    1|  1|   3|     4|       NA|           NA|            NA|        NA| -21|  55|
1019| -14|  NW| 9.84|          0.0|  0.0|
|  5|2010|    1|  1|   4|     4|       NA|           NA|            NA|        NA| -20|  51|
1018| -12|  NW|12.97|          0.0|  0.0|
+---+----+-----+---+----+------+---------+-------------+--------------+----------+----+----+
----+----+----+-----+-------------+-----+
only showing top 5 rows
```

2.2.1 Amount of Data

Diagrams 2.2.3 and 2.2.4 shows the total amounts of the data in the two datasets, which are 43,824 and 52,584, respectively.

Diagram 2.2.3 Total Amount of data in *Beijing PM2.5 Data*

```
#Show the amount of the instances of "Beijing PM2.5 Data".
dataset1.count()
```

```
43824
```

Diagram 2.2.4 Total Amount of data in

PM2.5 Data of Five Chinese Cities (The Part of Beijing)

```
#Show the amount of the instances of "PM2.5 Data of Five Chinese Cities (the Part of Beijing)".
dataset2.count()
```

```
52584
```

However, there are some invalid data with "NA" values or whitespaces in both datasets, which are unwanted for data mining. Since PySpark cannot recognize "NA"s or whitespaces as null, I used the function in *Diagram 2.2.5* to convert all the "NA" values and whitespaces to null. Afterwards, the function in *Diagram 2.2.6* was used to check the amount of valid data in both datasets.

Diagram 2.2.5 Function to Convert Invalid Values to Null

```
#Function to set all the "NA" values to null.
def set_NA_to_null(dataset):
    attributes = dataset.columns
    for attr in attributes:
        dataset = dataset.withColumn(attr, when(col(attr) == "NA", None).otherwise(col(attr)))
        dataset = dataset.withColumn(attr, when(col(attr) == " ", None).otherwise(col(attr)))
    return dataset
```

Diagram 2.2.6 Function to Check the Amount of Valid Data

```
#Function to check and print the amount of valid data in each column in the dataset.
def check_validation(dataset):
    attributes = dataset.columns
    for attr in attributes:
        valid_data = dataset.select(attr).na.drop()
        amount = valid_data.count()
        print(attr,"\t", amount)
```

To avoid inaccuracy caused by insufficiency of the data, the columns with less than 40,000 valid data will be reduced. Therefore, according to *Diagrams 2.2.7 and 2.2.8*, the combined dataset will have over 40,000 instances, each of which represents the PM2.5 concentration and the climatic and meteorology information in an hour. However, it is expected that some of them will be dismissed because of the values missing.

Also, the initially combined dataset will have 18 columns. As shown in *Diagrams 2.2.7* and *2.2.8*, four of them will be the time and season. Another four of them will be PM2.5 concentrations in different areas in Beijing. The last nine is going to be meteorology information.

Diagram 2.2.7 Amount of Valid Data in *Beijing PM2.5 Data*

```
#Call the function to check the amount of valid data.
check_validation(dataset1)
```

```
No       43824
year     43824
month    43824
day      43824
hour     43824
pm       41757
DEWP     43824
TEMP     43824
PRES     43824
cbwd     43824
Iws      43824
Is       43824
Ir       43824
```

Diagram 2.2.8

Amount of Valid Data in *PM2.5 Data of Five Chinese Cities* (The Part of Beijing)

```
#Call the function to check the amount of valid data.
check_validation(dataset2)
```

```
No              52584
year            52584
month           52584
day             52584
hour            52584
season          52584
PM_Dongsi       25052
PM_Dongsihuan   20508
PM_Nongzhanguan         24931
PM_US Post      50387
DEWP            52579
HUMI            52245
PRES            52245
TEMP            52579
cbwd            52579
Iws             52579
precipitation   52100
Iprec           52100
```

2.2.2 Value Types

Initially, since PySpark cannot recognize "NA" as null, it recognized all the attributes with "NA" as string type, which was not accurate, as shown in *Diagrams 2.2.9 and 2.2.10.*

Diagram 2.2.9 Inaccurately recognized data types in *Beijing PM2.5 Data*

```
#Show the data types of the instances of "Beijing PM2.5 Data".
dataset1.printSchema()
```

```
root
 |-- No: integer (nullable = true)
 |-- year: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- day: integer (nullable = true)
 |-- hour: integer (nullable = true)
 |-- pm: string (nullable = true)
 |-- DEWP: integer (nullable = true)
 |-- TEMP: double (nullable = true)
 |-- PRES: double (nullable = true)
 |-- cbwd: string (nullable = true)
 |-- Iws: double (nullable = true)
 |-- Is: integer (nullable = true)
 |-- Ir: integer (nullable = true)
```

Diagram 2.2.10 Inaccurately recognized data types

in *PM2.5 Data of Five Chinese Cities* (The Part of Beijing)

```
#Show the data types of the instances of "PM2.5 Data of Five Chinese Cities (the Part of Bei
jing)".
dataset2.printSchema()
```

```
root
 |-- No: integer (nullable = true)
 |-- year: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- day: integer (nullable = true)
 |-- hour: integer (nullable = true)
 |-- season: integer (nullable = true)
 |-- PM_Dongsi: string (nullable = true)
 |-- PM_Dongsihuan: string (nullable = true)
 |-- PM_Nongzhanguan: string (nullable = true)
 |-- PM_US Post: string (nullable = true)
 |-- DEWP: string (nullable = true)
 |-- HUMI: string (nullable = true)
 |-- PRES: string (nullable = true)
 |-- TEMP: string (nullable = true)
 |-- cbwd: string (nullable = true)
 |-- Iws: string (nullable = true)
 |-- precipitation: string (nullable = true)
 |-- Iprec: string (nullable = true)
```

Therefore, I checked the detailed datasets in Excel to find out their right value types for the data, then change the types to the right ones with the code shown in *Diagrams 2.2.11 and 2.2.12*.

As shown in *Diagrams 2.2.11 and 2.2.12,* Most of the value types in the combined dataset are integer, such as PM2.5 concentration, year, month, date, hour, season, dew point, and cumulated hours of rain and snow. Combined wind direction is in string type, while cumulated wind speed is double type.

Diagram 2.2.11 Data Types in *Beijing PM2.5 Data*

```
#Change data type of pm2.5 to integer type.
dataset1 = dataset1.withColumn("pm",dataset1["pm"].cast(IntegerType()))
dataset1.printSchema()
```

```
root
 |-- No: integer (nullable = true)
 |-- year: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- day: integer (nullable = true)
 |-- hour: integer (nullable = true)
 |-- pm: integer (nullable = true)
 |-- DEWP: integer (nullable = true)
 |-- TEMP: double (nullable = true)
 |-- PRES: double (nullable = true)
 |-- cbwd: string (nullable = true)
 |-- Iws: double (nullable = true)
 |-- Is: integer (nullable = true)
 |-- Ir: integer (nullable = true)
```

Diagram 2.2.12 Data Types in *PM2.5 Data of Five Chinese City* (The Part of Beijing)

```
#Change data type of numeric data to integer or double type.
dataset2 = dataset2.withColumn("PM_Dongsi",dataset2["PM_Dongsi"].cast(IntegerType()))
dataset2 = dataset2.withColumn("PM_Dongsihuan",dataset2["PM_Dongsihuan"].cast(IntegerType()))
dataset2 = dataset2.withColumn("PM_Nongzhanguan",dataset2["PM_Nongzhanguan"].cast(IntegerType()))
dataset2 = dataset2.withColumn("PM_US Post",dataset2["PM_US Post"].cast(IntegerType()))
dataset2 = dataset2.withColumn("DEWP",dataset2["DEWP"].cast(IntegerType()))
dataset2 = dataset2.withColumn("HUMI",dataset2["HUMI"].cast(IntegerType()))
dataset2 = dataset2.withColumn("PRES",dataset2["PRES"].cast(IntegerType()))
dataset2 = dataset2.withColumn("TEMP",dataset2["TEMP"].cast(IntegerType()))
dataset2 = dataset2.withColumn("Iws",dataset2["Iws"].cast(DoubleType()))
dataset2 = dataset2.withColumn("precipitation",dataset2["precipitation"].cast(IntegerType()))
dataset2 = dataset2.withColumn("Iprec",dataset2["Iprec"].cast(IntegerType()))
dataset2.printSchema()
```

```
root
 |-- No: integer (nullable = true)
 |-- year: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- day: integer (nullable = true)
 |-- hour: integer (nullable = true)
 |-- season: integer (nullable = true)
 |-- PM_Dongsi: integer (nullable = true)
 |-- PM_Dongsihuan: integer (nullable = true)
 |-- PM_Nongzhanguan: integer (nullable = true)
 |-- PM_US Post: integer (nullable = true)
 |-- DEWP: integer (nullable = true)
 |-- HUMI: integer (nullable = true)
 |-- PRES: integer (nullable = true)
 |-- TEMP: integer (nullable = true)
 |-- cbwd: string (nullable = true)
 |-- Iws: double (nullable = true)
 |-- precipitation: integer (nullable = true)
 |-- Iprec: integer (nullable = true)
```

2.2.3 Coding Schemes

Since both initial datasets are from the same author, they use the same code schema, as shown in *Diagram 2.2.13*. Therefore, there should be no problem when combining these two datasets into one.

Diagram 2.2.13 Code Explanation

Code	Explanation
No	Instance number
year	Year of data in this instance.
month	Month of data in this instance.
day	Day of data in this instance.
hour	Hour of data in this instance.
pm2.5	PM2.5 concentration (ug/m^3).
DEWP	Dew Point (°C).
TEMP	Temperature (°C).
PRES	Pressure (hPa).
cbwd	Combined wind direction (N-north, S-south, W-west, E-east, cv-calm or varies).
Iws	Cumulated wind speed (m/s).
Is	Cumulated hours of snow (hour).
Ir	Cumulated hours of rain (hour).
season	Season of data in this row (1-Spring, 2-Summer, 3-Autumn, 4-Winter).
HUMI	Humidity (%).
precipitation	Hourly precipitation (mm).
Iprec	Cumulated precipitation (mm).

2.3 Data Exploration

In this part, I have visualized some exploration, since PySpark does not have any build-in functions to do visualization, I used the libraries seaborn and matploy.pylot to visualize my exploration. Since these two visualization libraries require the input data frames as Pandas version, I converted the data frames to Pandas data frames before visualization. After the visualization of data exploration, I changed them back to Apache Spark data frames for latter processes such as data cleaning and data mining.

All the latter visualizations in this report, including the ones in the interpretation step, will use these libraries as there are no build-in PySpark methods to do visualization.

2.3.1 PM2.5 Data of Five Chinese Cities (the Part of Beijing)

Diagram 2.3.1 illustrated the PySpark code I used to explore the data of PM2.5 Concentration at the US Embassy in Beijing.

Diagram 2.3.1 Code to Explore PM2.5 Concentration

```
#Explore the data of PM2.5 concentration.
dataset2 = dataset2.toPandas()
pm25 = dataset2["PM_US Post"]
sns.distplot(pm25, kde=False)
plt.ylabel("Count")
plt.xlabel("PM2.5")
plt.show()
```

As a result, most values of the target field - PM2.5 concentration at the US Embassy centralized between 0 to 103 ug/m^3, as shown in *Diagram 2.3.2*. It also demonstrates that the instances of PM2.5 concentration higher than 200 ug/m^3 are much fewer than the instances that PM2.5 concentration lower than 200 ug/m^3.

Diagram 2.3.2 Distribution PM2.5 Concentration

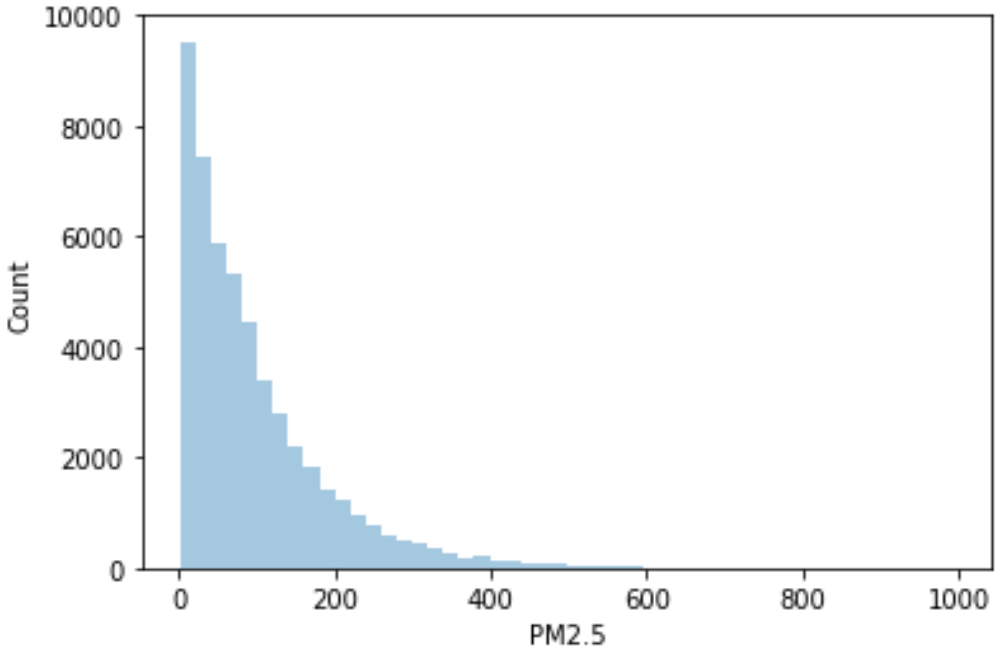


Diagram 2.3.3 illustrated the PySpark code I used to explore the data of combined wind direction.

Diagram 2.3.3 Code to Explore Combined Wind Direction

```
#Explore the data of combined wind speed.
cbwd_values = dataset2["cbwd"].value_counts()
plt.pie(cbwd_values, labels = cbwd_values.index)
plt.show()
```

Diagram 2.3.4 describes the data of combined wind direction. The amount of instance of southeast wind and northwest wind each took around 1/3, while about 1/5 of the time the wind is calm or variable. In the rest of the days, Beijing has northeast wind.

Diagram 2.3.4 Combined Wined Direction Pie Chart

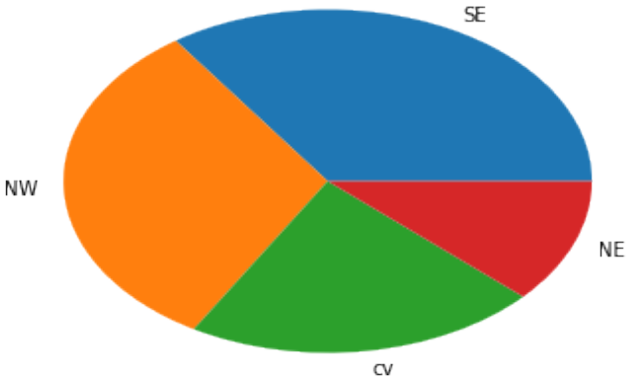


Diagram 2.3.5 demonstrates the PySpark code I used to explore the humidity data. For the exploration of the rest of the meteorology factors, I used almost the same code frame but changing the arguments to the other attribute names of the meteorology factors. The code provided me with the distribution of the data in each column, as well as the plot of them versus PM2.5 concentration.

Diagram 2.3.5 Code to Explore Humidity

```
#Explore the data of humidity.
humi = dataset2["HUMI"]
sns.distplot(humi, kde=False)
plt.ylabel("Count")
plt.xlabel("Humidity")
plt.show()

plt.scatter(humi,pm25)
plt.ylabel("PM2.5")
plt.xlabel("Humidity")
plt.show()
```

Diagram 2.3.6 shows the distribution of the humidity data, while *Diagram 2.3.7* displays the plot of humidity versus PM2.5 concentration. Generally, higher PM2.5 concentration case happens more frequently when the humidity is high.

Diagram 2.3.6 Distribution of Humidity Data

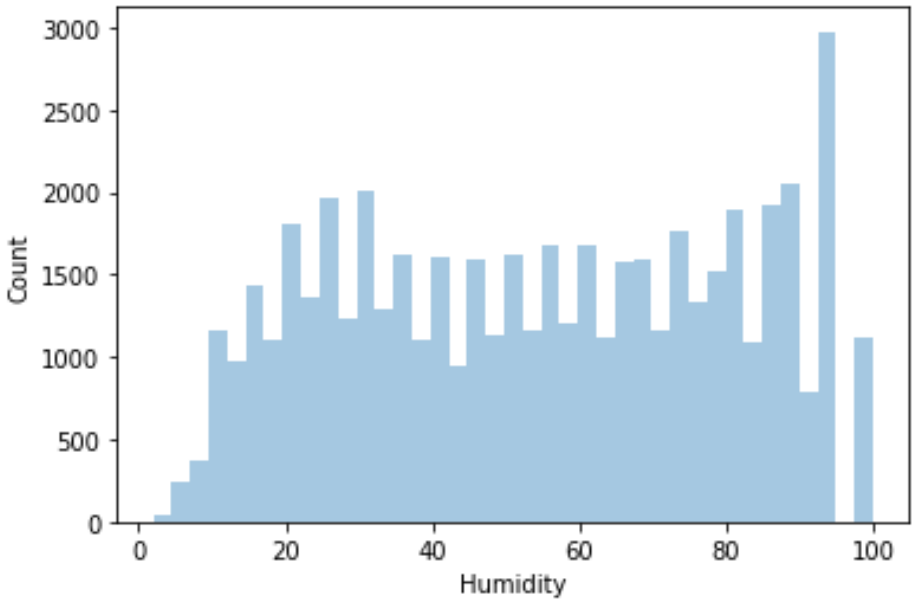


Diagram 2.3.7 Plot of Humidity versus PM2.5 Concentration

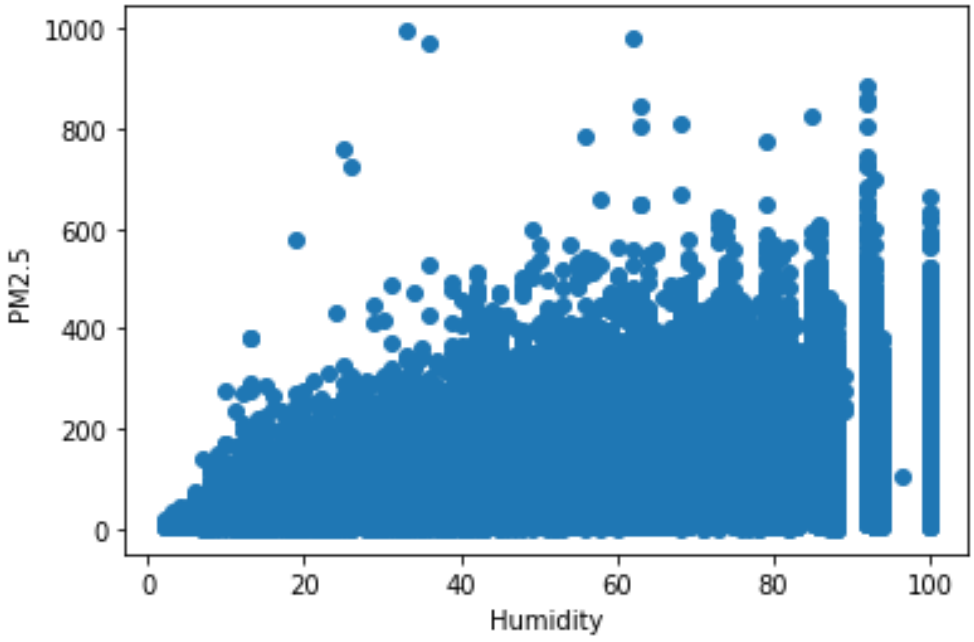


Diagram 2.3.8 describes that in most time, the cumulated wind speed in Beijing is lower than 13 m/s. Besides, *Diagram 2.3.9* indicates that higher cumulated wind speed can potentially make PM2.5 concentration go lower. An exception is that when the cumulated wind speed is between 230 to 305 m/s, PM2.5 concentration can go higher.

Diagram 2.3.8 Distribution of Cumulated Wind Speed Data

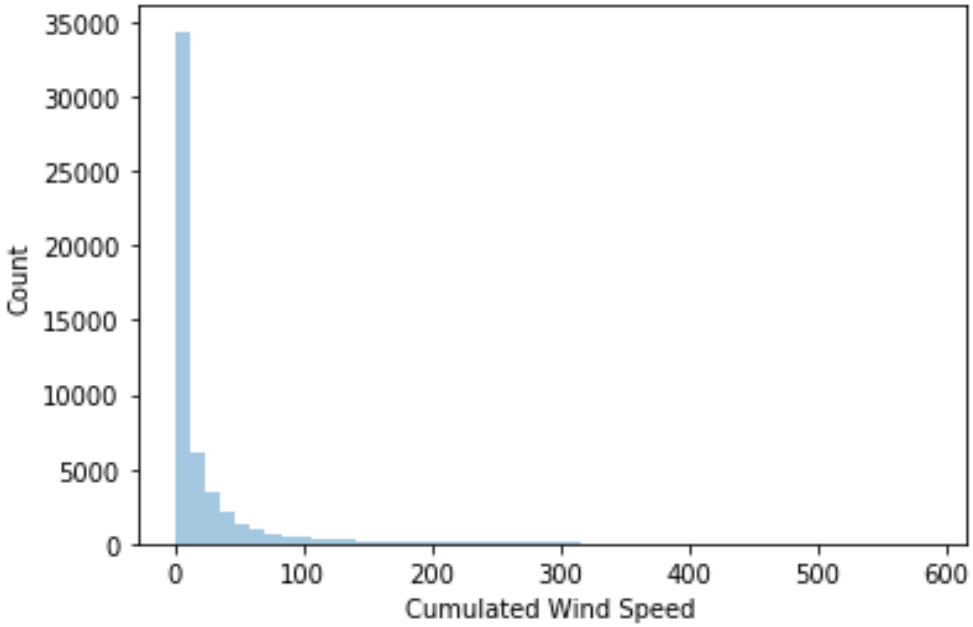


Diagram 2.3.9 Plot of Cumulated Wind Speed versus PM2.5 Concentration

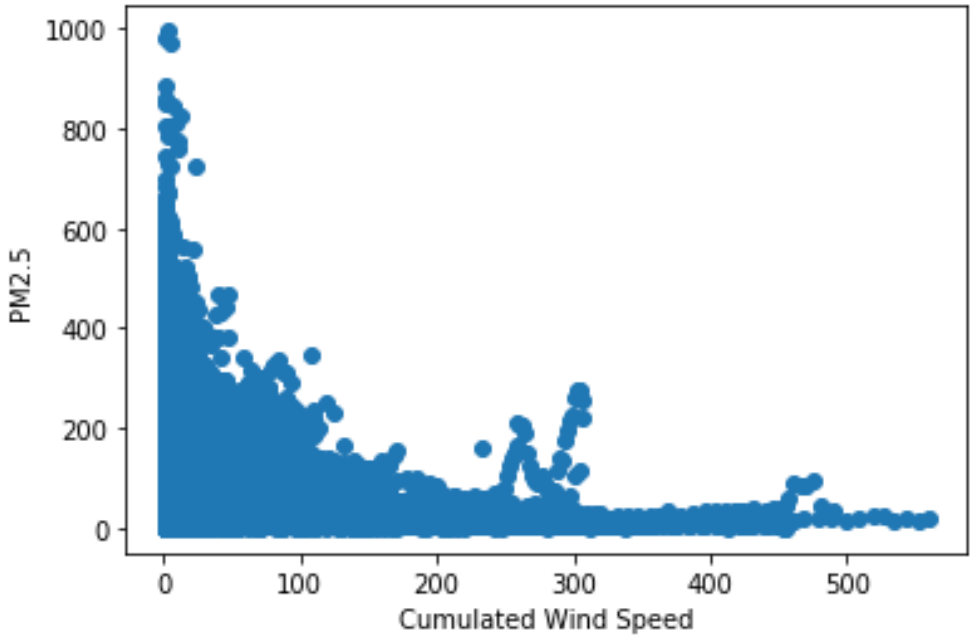


Diagram 2.3.10 shows that the temperatures in Beijing are mostly between -8 to 32 °C. Besides, *Diagram 2.3.11* indicates that very low or very high temperature can possibly stop PM2.5 concentration going high.

Diagram 2.3.10 Distribution of Temperature Data

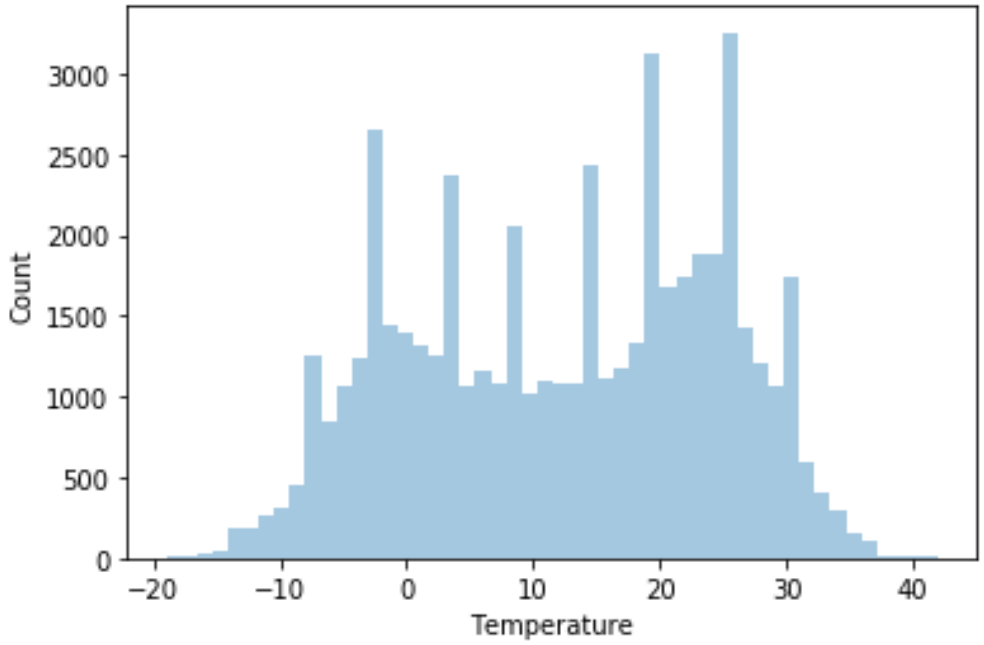


Diagram 2.3.11 Plot of Temperature versus PM2.5 Concentration

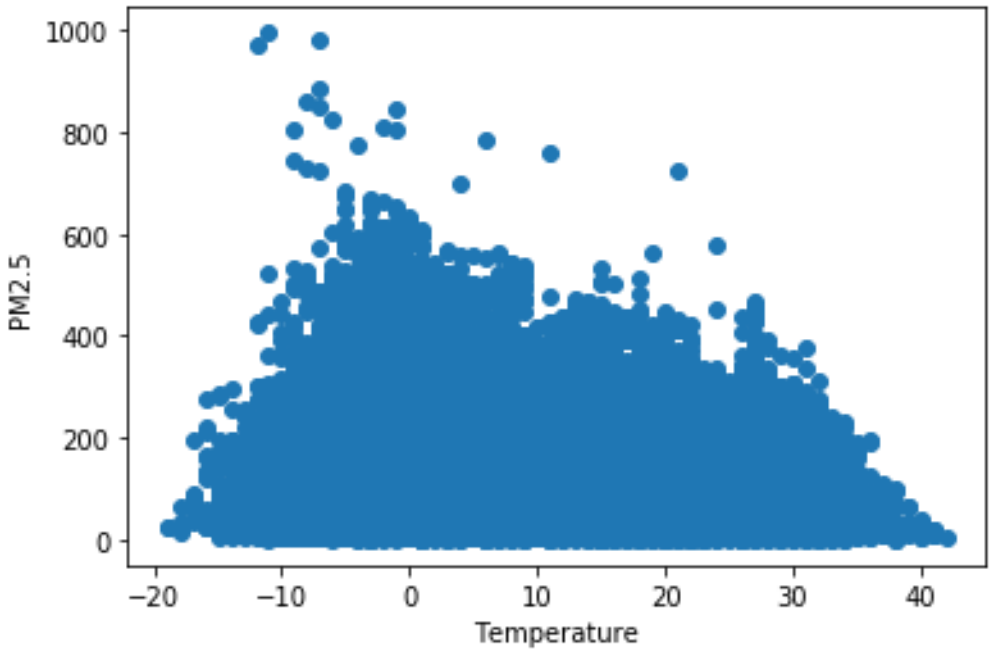


Diagram 2.3.12 describes that the dew points are mostly between -22 to 25 °C. Besides, *Diagram 2.3.13* indicates that very low dew point may affect PM2.5 concentration to be lower. When the dew point is between -10 to -7 °C, PM2.5 concentration can be extremely high.

Diagram 2.3.12 Distribution of Dew Point Data

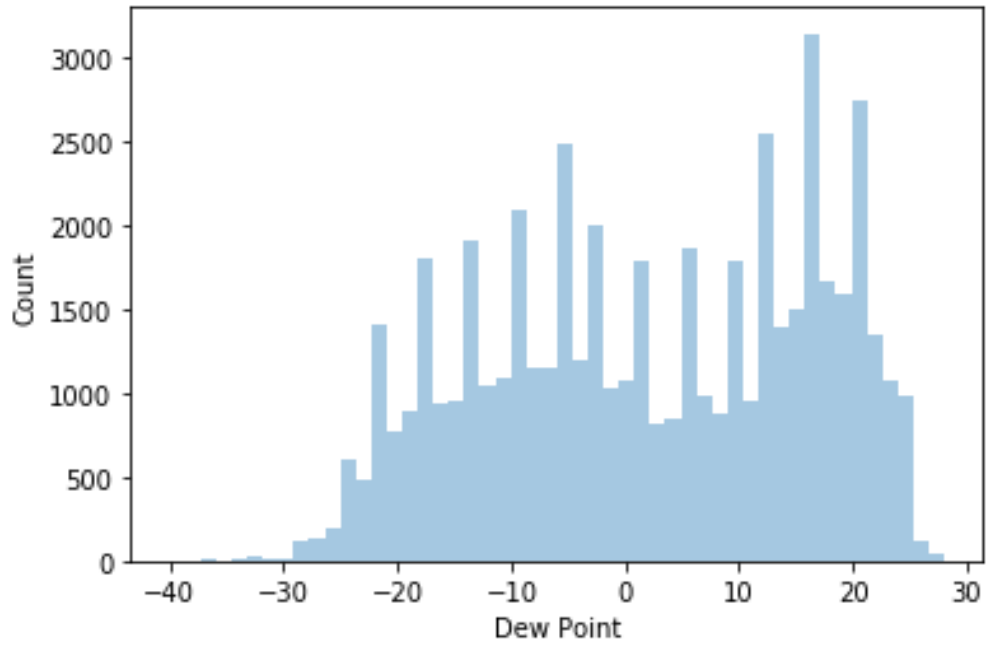


Diagram 2.3.13 Plot of Dew Point versus PM2.5 Concentration

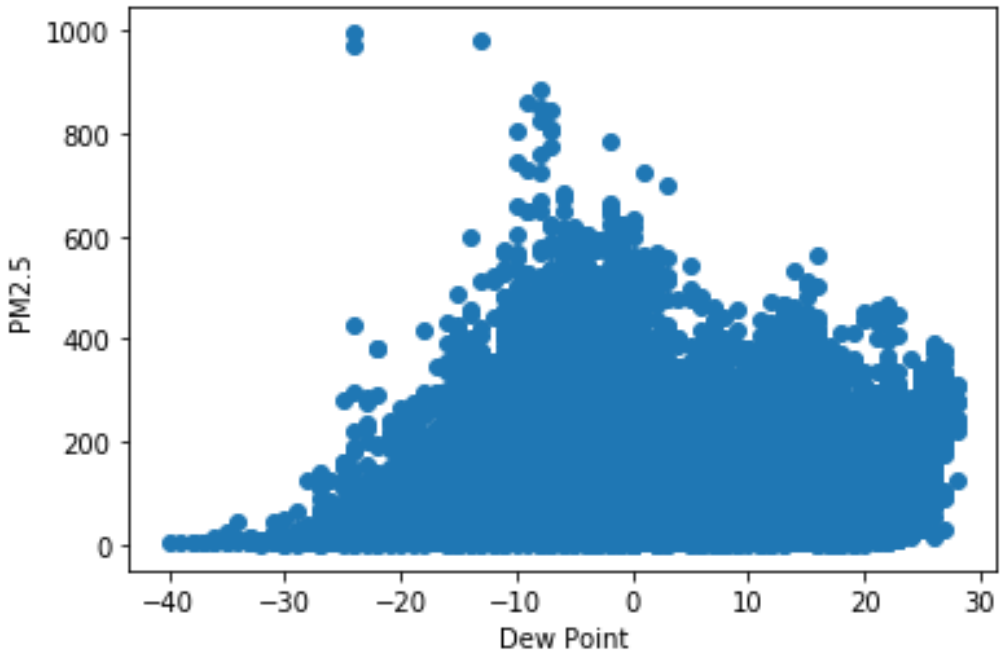


Diagram 2.3.14 describes that the pressure data are consistent with normal distribution, while *Diagram 2.3.15* indicates that very low and very high pressure may control PM2.5 in a low concentration.

Diagram 2.3.14 Distribution of Pressure Data

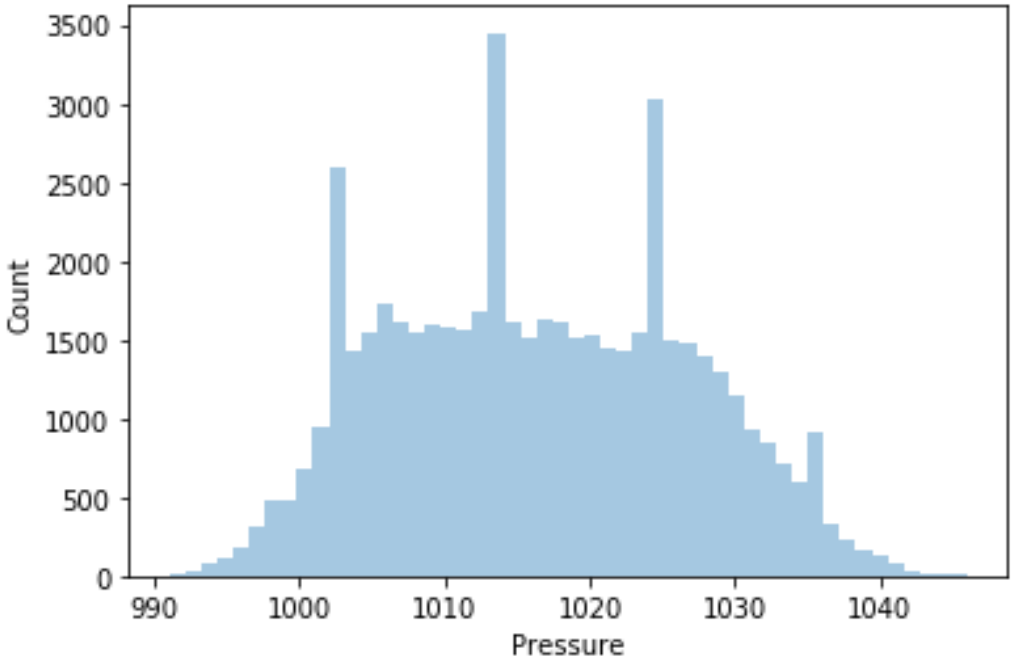


Diagram 2.3.15 Plot of Pressure versus PM2.5 Concentration

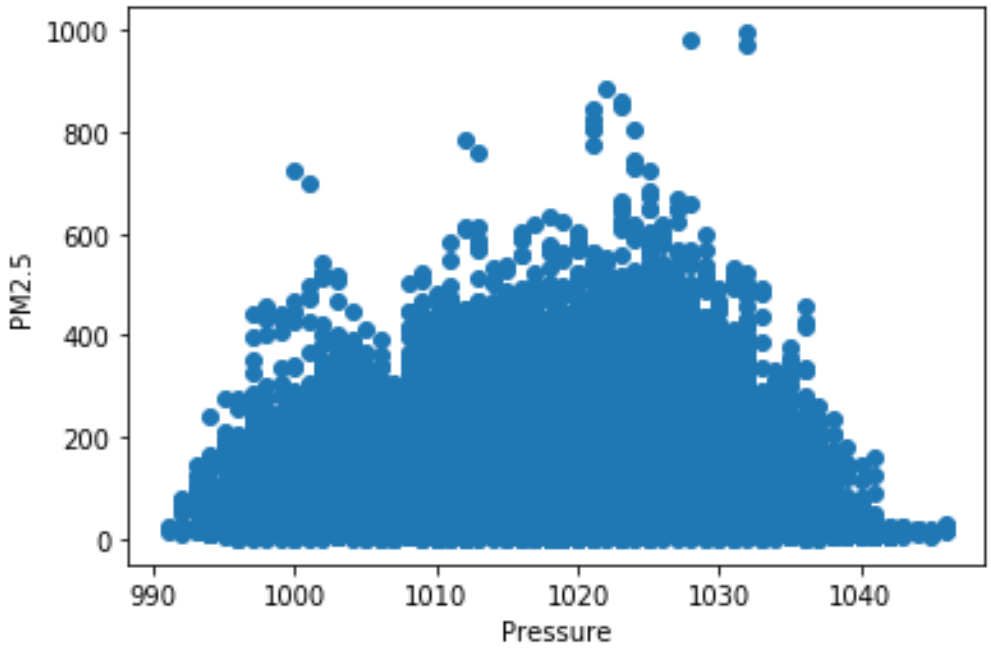


Diagram 2.3.16 describes that PM2.5 concentration and the meteorology factors which may affect it are all in reasonable ranges except hourly precipitation and cumulated precipitation. In order to have a clearer illustration, I separated them into three tables.

Diagram 2.3.16 Statistic Data of PM 2.5 concentration and the Meteorology Factors

```
#Get statistical data for "PM2.5 Data of Five Chinese Cities (the Part of Beijing)".
dataset2.select('PM_US Post').describe().show()
dataset2.select('DEWP','HUMI','PRES','TEMP').describe().show()
dataset2.select('Iws','precipitation','Iprec').describe().show()
```

```
+-------+-----------------+
|summary|       PM_US Post|
+-------+-----------------+
|  count|            50387|
|   mean|95.90424117331851|
| stddev|91.64377236804653|
|    min|                1|
|    max|              994|
+-------+-----------------+

+-------+------------------+-----------------+------------------+------------------+
|summary|              DEWP|             HUMI|              PRES|              TEMP|
+-------+------------------+-----------------+------------------+------------------+
|  count|             52579|            52245|             52245|             52579|
|   mean|2.0745544799254456|54.60231601110154| 1016.465403387884|12.587021434412978|
| stddev|14.222058604097723| 25.9913550673888|10.295032148686303|12.098526776991708|
|    min|               -40|                2|               991|               -19|
|    max|                28|              100|              1046|                42|
+-------+------------------+-----------------+------------------+------------------+

+-------+------------------+------------------+------------------+
|summary|               Iws|     precipitation|             Iprec|
+-------+------------------+------------------+------------------+
|  count|             52579|             52100|             52100|
|   mean| 23.26182886704009|19.245028790786947|19.503282149712092|
| stddev|49.281705508541435| 4381.035586211772| 4381.036077990199|
|    min|              0.45|                 0|                 0|
|    max|             585.6|            999990|            999990|
+-------+------------------+------------------+------------------+
```

2.3.2 Beijing PM2.5 Data

Since this dataset has many attributes same as *PM2.5 Data of Five Chinese Cities (the part of Beijing)*, I did not explore the duplicated parts in depth.

Diagram 2.3.17 shows that Beijing does not rain in most time. However, *Diagram 2.3.18* indicates that the longer it rains, the lower PM2.5 concentration may be.

Diagram 2.3.17 Distribution of Cumulated Hours of Rain

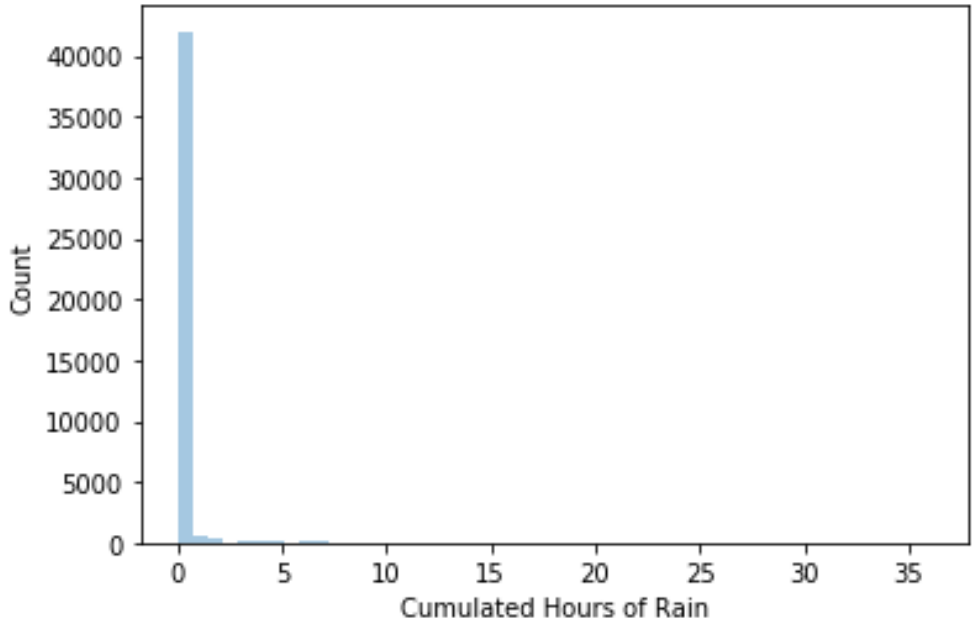
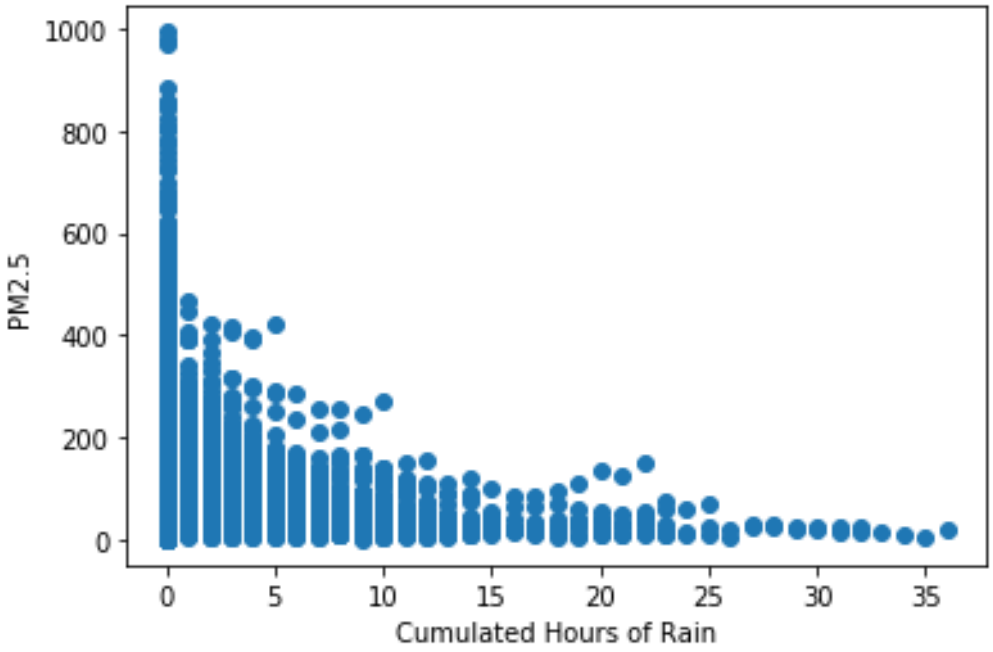


Diagram 2.3.18 Plot of Cumulated Hours of Rain versus PM2.5 Concentration



Diagrams 2.3.19 and *2.3.20* show that the situation of cumulated hours of snow is quite similar to the situation of cumulated hours of rain. Beijing mostly does not snow, while longer snowing time seems lowering PM2.5 concentration in the city.

Diagram 2.3.19 Distribution of Cumulated Hours of Snow

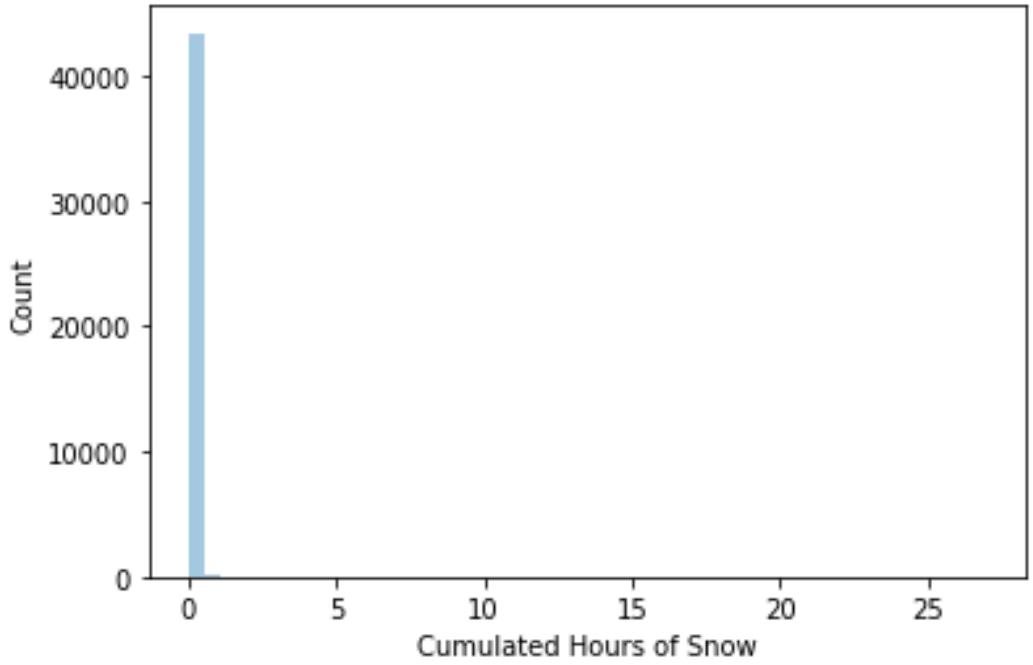


Diagram 2.3.20 Plot of Cumulated Hours of Snow versus PM2.5 Concentration

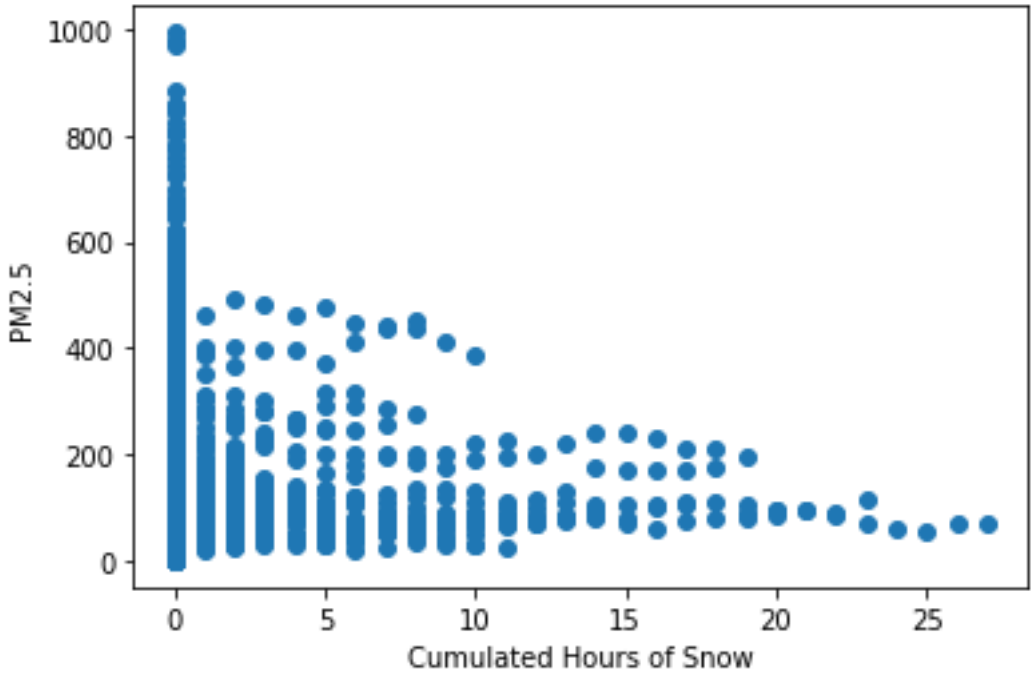


Diagram 2.3.21 describes that cumulated hours of rain and snow, which may affect PM2.5 concentration, are both in reasonable ranges.

Diagram 2.3.21 Statistic Data of cumulated hours of rain and snow

```
#Get statistical data for "Is" and "Ir".
dataset1.select('Is','Ir').describe().show()
```

```
+-------+-------------------+-------------------+
|summary|                 Is|                 Ir|
+-------+-------------------+-------------------+
|  count|              43824|              43824|
|   mean|0.05273366192040891|0.19491602774735306|
| stddev| 0.7603747364300517| 1.4158667068673554|
|    min|                  0|                  0|
|    max|                 27|                 36|
+-------+-------------------+-------------------+
```

2.4 Data Quality Verification

2.4.1 Missing Data

Diagram 2.4.1 shows the function that was used to evaluate the completion of each column in the dataset.

Diagram 2.4.1 Function to Evaluate Completion

```
#Function to check and print the completion of data in each column in the dataset.
def check_completion(dataset):
    attributes = dataset.columns
    instance_no = dataset.select("No").count()
    for attr in attributes:
        completion = dataset.select(attr).na.drop().count() / instance_no
        print(attr, "\t", round(completion *100,2), "%")
```

As shown in *Diagram 2.4.2*, only a few data are missing in *Beijing PM2.5 Data*. The instances with missing data will be removed later in the data preparation chapter as the remaining instances are enough for data mining.

Diagram 2.4.2 Data Completion of *Beijing PM2.5 Data*

```
#Call the function to check the completion of each column.
check_completion(dataset1)
```

```
No        100.0 %
year      100.0 %
month     100.0 %
day       100.0 %
hour      100.0 %
pm        95.28 %
DEWP      100.0 %
TEMP      100.0 %
PRES      100.0 %
cbwd      100.0 %
Iws       100.0 %
Is        100.0 %
Ir        100.0 %
```

However, *Diagram 2.4.3* shows that in *PM2.5 Data of Five Chinese Cities* (the part of Beijing), there are too many missing values of PM2.5 concentration in all the areas except the US Embassy. To guarantee there are enough instances to ensure the accuracy of the study, PM2.5 concentration in those places are decided not to be used after data exploration. The instances that miss PM2.5 concentration at the US Embassy will be left out as well. Besides, the columns all have enough valid data, but the related instances will be removed if the value is missing in the column as the remaining instances will still be sufficient.

Diagram 2.4.3 Data Completion of *PM2.5 Data of Five Chinese Cities* (the Part of Beijing)

```
#Call the function to check the completion of each column.
check_completion(dataset2)
```

```
No              100.0 %
year            100.0 %
month           100.0 %
day             100.0 %
hour            100.0 %
season          100.0 %
PM_Dongsi       47.64 %
PM_Dongsihuan   39.0 %
PM_Nongzhanguan          47.41 %
PM_US Post      95.82 %
DEWP      99.99 %
HUMI      99.36 %
PRES      99.36 %
TEMP      99.99 %
cbwd      99.99 %
Iws       99.99 %
precipitation    99.08 %
Iprec     99.08 %
```

2.4.2 Data Error

Since the initial dataset are provided in UCL Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets.php), which is a main source of machine learning and widely used in the academic community all over the world (UCI Machine Learning Repository, n.d.), the correctness of the datasets should be trustworthy.

However, when I used the functions in *Diagram 2.4.4* to check the maximum and minimum value for the numeric attributes, I found the maximum values of hourly precipitation and cumulated precipitation are abnormal as shown in *Diagram 2.4.6*. They might be typographical mistakes. Later in the data preparation chapter, instances with these mistakes will be deleted before data mining. Otherwise, I have not found any typographical errors. *Diagrams 2.4.5 and 2.4.6* demonstrate that all the other data are in reasonable ranges.

Diagram 2.4.4 Functions to Get Maximum and Minimum Values

```
#Function to check and print the maximum and minumum value of each numeric column in the dataset.
def check_max_and_min(dataset):
    numeric_sub_dataset = dataset.drop("cbwd")
    sub_dataset = numeric_sub_dataset.toPandas()
    attributes = pd.Series(sub_dataset.columns)
    max_values = []
    min_values = []
    for attr in attributes:
        min_value = sub_dataset[attr].min()
        min_values.append(min_value)
        max_value = sub_dataset[attr].max()
        max_values.append(max_value)
    max_df = pd.Series(max_values, index = attributes.tolist(), name = 'Max')
    min_df = pd.Series(min_values, index = attributes.tolist(), name = 'Min')
    attrs = pd.Series(attributes.tolist(), index = attributes.tolist(), name = 'Attributes')
    max_min_df = pd.concat([attrs, min_df, max_df], axis = 1)
    return max_min_df
```

Diagram 2.4.5 Minimum and Maximum values of the Attributes in *Beijing PM2.5 Data*

```
#Call the function to get the maximum and minimum values for each column.
max_min_df = spark.createDataFrame(check_max_and_min(dataset1))
max_min_df.show()
```

```
+----------+------+-------+
|Attributes|   Min|    Max|
+----------+------+-------+
|        No|   1.0|43824.0|
|      year|2010.0| 2014.0|
|     month|   1.0|   12.0|
|       day|   1.0|   31.0|
|      hour|   0.0|   23.0|
|        pm|   0.0|  994.0|
|      DEWP| -40.0|   28.0|
|      TEMP| -19.0|   42.0|
|      PRES| 991.0| 1046.0|
|       Iws|  0.45|  585.6|
|        Is|   0.0|   27.0|
|        Ir|   0.0|   36.0|
+----------+------+-------+
```

Diagram 2.4.6 Minimum and Maximum values of the Attributes in *PM2.5 Data of Five Chinese Cities* (the Part of Beijing)

```
#Call the function to get the maximum and minimum values for each column.
max_min_df = spark.createDataFrame(check_max_and_min(dataset2))
max_min_df.show()
```

```
+---------------+------+--------+
|     Attributes|   Min|     Max|
+---------------+------+--------+
|             No|   1.0| 52584.0|
|           year|2010.0|  2015.0|
|          month|   1.0|    12.0|
|            day|   1.0|    31.0|
|           hour|   0.0|    23.0|
|         season|   1.0|     4.0|
|      PM_Dongsi|   3.0|   737.0|
|  PM_Dongsihuan|   3.0|   672.0|
|PM_Nongzhanguan|   3.0|   844.0|
|     PM_US Post|   1.0|   994.0|
|           DEWP| -40.0|    28.0|
|           HUMI|   2.0|   100.0|
|           PRES| 991.0|  1046.0|
|           TEMP| -19.0|    42.0|
|            Iws|  0.45|   585.6|
|  precipitation|   0.0|999990.0|
|          Iprec|   0.0|999990.0|
+---------------+------+--------+
```

2.4.3 Measurement

The data has already been used by the author and some other scholars in published researches in academic journals which are Journal of Geophysical Research: Atmospheres (Liang, Zhang, Huang, & Chen, 2016) and Proceedings of the Royal Society A: Mathematical, Physical & Engineering Sciences (Liang et al., 2015), so the measurement error should be checked already by the scholars and not be worried too much.

However, during data Exploration, the function in Diagram 2.4.7 has found some outliers, as shown in *Diagrams 2.4.8* and *2.4.9*. After checking the values of the data in *Diagrams 2.4.5* and *2.4.6*, most of the values are in reasonable ranges, even though they seem extreme. For example, *Diagram 2.4.9* points out 933 outliers in the field *PM_US Post*. Nevertheless, after checking the values of the data in *Diagram 2.4.6*, the minimum value – 1 ug/m^3 and maximum value – 994 ug/m^3 are both reasonable as values of PM2.5 concentration.

However, a few unreasonable values still exist. *Diagram 2.4.6* illustrates that the values of hourly precipitation and cumulated precipitation include some unreasonably big value. These will be deleted before data mining. Otherwise, the dataset is high-quality in measurement.

Diagram 2.4.7 Functions to Look for Outliers

```
#Function to check and print the amount of outliers in each numeric column in the dataset.
def check_outliers(dataset):
    numeric_sub_dataset = dataset.drop("cbwd")
    attributes = numeric_sub_dataset.columns
    for attr in attributes:
        df_stats = numeric_sub_dataset.select( _mean(col(attr)).alias('mean'),
                                               _stddev(col(attr)).alias('std')).collect()
        mean = df_stats[0]['mean']
        std = df_stats[0]['std']
        lower_limit = mean - 3 * std
        upper_limit = mean + 3 * std
        lower_outliers = numeric_sub_dataset.filter(numeric_sub_dataset[attr] < lower_limi
t).select(attr)
        upper_outliers = numeric_sub_dataset.filter(numeric_sub_dataset[attr] > upper_limi
t).select(attr)
        amount_of_outliers = lower_outliers.count() + upper_outliers.count()
        print(attr,"\t", amount_of_outliers)
```

Diagram 2.4.8 Amounts of Outliers in *Beijing PM2.5 Data*

```
#Call the function to check the amount of outliers for each column.
check_outliers(dataset1)
```

```
No       0
year     0
month    0
day      0
hour     0
pm       769
DEWP     0
TEMP     0
PRES     0
Iws      1163
Is       256
Ir       613
```

Diagram 2.4.9 Amounts of Outliers in

PM2.5 Data of Five Chinese Cities (the Part of Beijing)

```
#Call the function to check the amount of outliers for each column.
check_outliers(dataset2)
```

```
No              0
year            0
month           0
day             0
hour            0
season          0
PM_Dongsi         413
PM_Dongsihuan     394
PM_Nongzhanguan           470
PM_US Post        933
DEWP            0
HUMI            0
PRES            0
TEMP            0
Iws            1388
precipitation     1
Iprec             1
```

2.4.4 Coding Quality

Both datasets are from the same author and use the same coding style. For example, numbers from one to four are used to represent the season, while characters E, W, S and N are used for wind directions east, west, south and

north, as shown in *Diagram 2.2.13*. As a result, the coding is high-quality. Therefore, the concern of inconsistencies can be ignored.

2.4.5 Metadata

Attribute types have already been discussed in the data types chapter where *Diagrams 2.2.11, 2.2.12* and *2.2.13* together proved that the attributes in both datasets are well-defined and understandable. Also, the data values match their field names in each column except two abnormal values in hourly precipitation and cumulated precipitation, as shown in *Diagram 2.4.5* and *2.4.6*. Otherwise, there is no problem with the high-quality metadata.

3. Data Preparation

3.1 Selecting Data

Data selection is based on the consideration of business and data-mining objectives discussed in the situation understanding chapter, data quality concern discussed in the data understanding chapter, the technical constraint of the library pyspark, which is used to undertake the data mining process in PySpark. Details are discussed in the following subsections.

3.1.1 Selecting Items

As shown in *Diagrams 2.4.5 and 2.4.6*, data in *Beijing PM2.5 Data* are from 2010 to 2014, while *PM2.5 Data of Five Chinese Cities* (the Part of Beijing) has data from 2010 to 2015. If we keep all the data, the data of 2015 will have too many missing values in some meteorology factors.

To ensure the data quality of the combined dataset, data of 2015 are discarded by the Select node, as shown in *Diagram 3.1.1*. After the discard, there will still be 43,824 instances, which is enough for data mining and under the processing capacity of the library.

Diagram 3.1.1 Code to Discard Data of 2015

```
#Select the items from the dataset.
#Only the data before 2015 are kept for the further progress.
dataset2 = dataset2.filter('year<2015')
dataset2.count()
```

```
43824
```

3.1.2 Selecting Attributes

As discussed in the data description chapter, *Diagrams 2.2.11* and *2.2.12* illustrate that there are some duplicated attributes in the two datasets. Therefore, discarding the duplicated columns will improve the quality of the combined dataset. Besides, in the data quality verification chapter, *Diagram 2.4.3* clarified that PM2.5 concentration data in the areas except the US Embassy include too many missing data. Therefore, these PM2.5 concentration data will be deleted to ensure the quality of data. *Diagram 3.1.2* shows the code I used the keep only the unduplicated and high-quality attributes in *PM2.5 Data of Five Chinese Cities* (the Part of Beijing).

Diagram 3.1.2 Code to Keep Only the Unduplicated and High-Quality Attributes

```
#Select the attributes from the dataset.
#Only high-quality and unique data are kept.
#Those duplicated and low-quality attributes are discarded.
dataset2 = dataset2.select("year","month","day","hour","HUMI","precipitation","Iprec")
dataset2.show(5)
```

```
+----+-----+---+----+----+-------------+-----+
|year|month|day|hour|HUMI|precipitation|Iprec|
+----+-----+---+----+----+-------------+-----+
|2010|    1|  1|   0|43.0|          0.0|  0.0|
|2010|    1|  1|   1|47.0|          0.0|  0.0|
|2010|    1|  1|   2|43.0|          0.0|  0.0|
|2010|    1|  1|   3|55.0|          0.0|  0.0|
|2010|    1|  1|   4|51.0|          0.0|  0.0|
+----+-----+---+----+----+-------------+-----+
only showing top 5 rows
```

Also, as the business and data mining perspectives for this project focus on the relationships between PM2.5 concentration and meteorology factors, those data, which are not related to meteorology elements, are excluded before data cleaning, such as season and time. *Diagram 3.1.3* shows the code I used to keep only the meteorology-related attributes and PM2.5 concentration for the latter process.

Diagram 3.1.3 Code to Keep the Final Attributes

```
#Drop the attributes from the datasets again.
#Only the target and metoerology-related attributes are kept.
dataset_final = dataset_final.drop('year','month','day','hour')
dataset_final.show(5)
```

```
+---+----+----+-----+------+----+-----+---+---+----+----+----+-------------+-----+
| No|  pm|DEWP| TEMP|  PRES|cbwd|  Iws| Is| Ir|rain|snow|HUMI|precipitation|Iprec|
+---+----+----+-----+------+----+-----+---+---+----+----+----+-------------+-----+
|  1|null| -21|-11.0|1021.0|  NW| 1.79|  0|  0|  No|  No|43.0|          0.0|  0.0|
|  2|null| -21|-12.0|1020.0|  NW| 4.92|  0|  0|  No|  No|47.0|          0.0|  0.0|
|  3|null| -21|-11.0|1019.0|  NW| 6.71|  0|  0|  No|  No|43.0|          0.0|  0.0|
|  4|null| -21|-14.0|1019.0|  NW| 9.84|  0|  0|  No|  No|55.0|          0.0|  0.0|
|  5|null| -20|-12.0|1018.0|  NW|12.97|  0|  0|  No|  No|51.0|          0.0|  0.0|
+---+----+----+-----+------+----+-----+---+---+----+----+----+-------------+-----+
only showing top 5 rows
```

3.2 Cleaning Data

3.2.1 Missing Data

In the selecting data chapter, *Diagram 3.1.2* illustrated the discard of the attributes that with too many missing data. After that, there are still some small amount of missing data in some useful instances, as shown in *Diagram 3.2.1*. It shows that the valid records are enough for data mining. I will discard the record with missing values instead of imputing data, because imputing data may randomly give inaccurate values which may affect the data mining results.

Diagram 3.2.1 Data Completion of Selected Data

```
#Call the function to check the completion of each column.
du.check_completion(dataset_final)
```

```
No          100.0 %
pm          95.28 %
DEWP        100.0 %
TEMP        100.0 %
PRES        100.0 %
cbwd        100.0 %
Iws         100.0 %
Is          100.0 %
Ir          100.0 %
rain        100.0 %
snow        100.0 %
HUMI        100.0 %
precipitation    99.94 %
Iprec    99.94 %
```

As shown in *Diagram 3.2.2*, I discarded the instances with null values in it and to recheck the completion of all the attributes. Afterwards, all the attributes have the completion of 100%.

Diagram 3.2.2 Discard Instances with Null Values

```
#Drop the instances with missing values.
dataset_final = dataset_final.na.drop()
du.check_completion(dataset_final)
```

```
No            100.0 %
pm            100.0 %
DEWP          100.0 %
TEMP          100.0 %
PRES          100.0 %
cbwd          100.0 %
Iws           100.0 %
Is            100.0 %
Ir            100.0 %
rain          100.0 %
snow          100.0 %
HUMI          100.0 %
precipitation    100.0 %
Iprec         100.0 %
```

3.2.2 Data Error and Measurement

Diagram 2.4.6 in the data quality verification chapter indicated that unreasonable large values exist in hourly and cumulated precipitations, which are supposed to be cleaned at this stage. However, *Diagram 3.2.3* shows that all the data for hourly and cumulated precipitations are reasonable now. It probably happened at selecting data stage. Therefore, data errors and measurement errors are all cleaned now.

Diagram 3.2.3 Cleaned Values of Hourly and Cumulated Precipitations

```
#Double-check the measurement error.
max_min_df = du.check_max_and_min(dataset_final.select('precipitation','Iprec'))
spark.createDataFrame(max_min_df).show()
```

```
+-------------+---+-----+
|   Attributes|Min|  Max|
+-------------+---+-----+
|precipitation|0.0| 69.2|
|        Iprec|0.0|223.0|
+-------------+---+-----+
```

3.3 Construct the Data

In the data exploration chapter, *Diagrams 2.3.18* and *2.3.20* indicated that longer snowing or raining hours might help control PM2.5 concentration. Therefore, I suspect that PM2.5 concentration may be decreased as long as it rains or snows. To find out whether my suspicion is right, I used the code in *Diagram 3.3.1* to constructed two columns – rain and snow with values "Yes" or "No" to show whether it rained or snowed in that specific hour. The values are constructed by the values of cumulated hours of rain and snow. The value of the constructed attribute will be "Yes" if the value of hours is greater than zero, otherwise "No".

Diagram 3.3.1 Construct Attributes – Rain and Snow

```
#Construct two new attributes - rain and snow, then show the dataset.
dataset1 = dataset1.withColumn('rain',when(col('Ir')>0, "Yes").otherwise("No"))
dataset1 = dataset1.withColumn('snow',when(col('Is')>0, "Yes").otherwise("No"))
dataset1.show(5)
```

```
+---+----+-----+---+----+-----+----+-----+------+----+-----+---+---+----+----+
| No|year|month|day|hour|pm2.5|DEWP| TEMP|  PRES|cbwd|  Iws| Is| Ir|rain|snow|
+---+----+-----+---+----+-----+----+-----+------+----+-----+---+---+----+----+
|  1|2010|    1|  1|   0|   NA| -21|-11.0|1021.0|  NW| 1.79|  0|  0|  No|  No|
|  2|2010|    1|  1|   1|   NA| -21|-12.0|1020.0|  NW| 4.92|  0|  0|  No|  No|
|  3|2010|    1|  1|   2|   NA| -21|-11.0|1019.0|  NW| 6.71|  0|  0|  No|  No|
|  4|2010|    1|  1|   3|   NA| -21|-14.0|1019.0|  NW| 9.84|  0|  0|  No|  No|
|  5|2010|    1|  1|   4|   NA| -20|-12.0|1018.0|  NW|12.97|  0|  0|  No|  No|
+---+----+-----+---+----+-----+----+-----+------+----+-----+---+---+----+----+
only showing top 5 rows
```

3.4 Integrating Data

3.4.1 Appending

As Shahzad suggested, I separated *PM2.5 Data of Five Chinese Cities* (the part of Beijing) into two datasets, *BeijingPM2010_2012* and *BeijingPM2013_2015*, to make sure that I can use appending function. *Diagram 3.4.1* illustrates the code of the appending process. The two datasets are appended based on their same attribute names. After appending, *PM2.5 Data of Five Chinese Cities* (the part of Beijing) has 52,584 instances.

Diagram 3.4.1 Code to Append Two Datasets

```
#Append two dataset with same attributes names.
dataset2_a = spark.read.csv('./Datasets/BeijingPM2010_2012.csv',
                            inferSchema = True, header = True)
dataset2_b = spark.read.csv('./Datasets/BeijingPM2013_2015.csv',
                            inferSchema = True, header = True)
dataset2 = dataset2_a.union(dataset2_b)
dataset2.count()
```

```
52584
```

3.4.2 Merging

Diagram 3.4.2 illustrates the code to merge *Beijing PM2.5 Data* and *PM2.5 Data of Five Chinese Cities* (the part of Beijing). The keys for merging are year, month, day and hour, which means the instance under the same time will be merged. As a result, only matching records will be retained.

Diagram 3.4.2 Code to Merge the Two Datasets

```
#Merge two datasets based on the key ('year','month','day','hour').
dataset_final = dataset1.join(dataset2,['year','month','day','hour'])
dataset_final = dataset_final.withColumnRenamed("pm2.5","pm")

#Call the function to set "NA" values to null and change the type of PM2.5 concentration to Integer type for further cl
dataset_final = du.set_NA_to_null(dataset_final)
dataset_final = dataset_final.withColumn("pm",dataset_final["pm"].cast(IntegerType()))
dataset_final.show(5)
```

```
+----+-----+---+----+---+----+----+-----+------+----+-----+---+---+----+----+----+-------------+-----+
|year|month|day|hour| No|  pm|DEWP| TEMP|  PRES|cbwd|  Iws| Is| Ir|rain|snow|HUMI|precipitation|Iprec|
+----+-----+---+----+---+----+----+-----+------+----+-----+---+---+----+----+----+-------------+-----+
|2010|    1|  1|   0|  1|null| -21|-11.0|1021.0|  NW| 1.79|  0|  0|  No|  No|43.0|          0.0|  0.0|
|2010|    1|  1|   1|  2|null| -21|-12.0|1020.0|  NW| 4.92|  0|  0|  No|  No|47.0|          0.0|  0.0|
|2010|    1|  1|   2|  3|null| -21|-11.0|1019.0|  NW| 6.71|  0|  0|  No|  No|43.0|          0.0|  0.0|
|2010|    1|  1|   3|  4|null| -21|-14.0|1019.0|  NW| 9.84|  0|  0|  No|  No|55.0|          0.0|  0.0|
|2010|    1|  1|   4|  5|null| -20|-12.0|1018.0|  NW|12.97|  0|  0|  No|  No|51.0|          0.0|  0.0|
+----+-----+---+----+---+----+----+-----+------+----+-----+---+---+----+----+----+-------------+-----+
only showing top 5 rows
```

After merging, the “NA” values are set to null, and PM2.5 concentration is set to integer type to make sure the dataset is cleaned to format the data.

3.5 Formatting Data

Later on, I will use pyspark library to build the models and to conduct the data mining process. Therefore, I need to format the data as the library requires. In pyspark data mining, string data are required to be processed before data mining, as only numeric variables are supported in its models. Thus, all the data will be encoded into numeric codes.

In the merged dataset, there are three columns with categorical data, which are combined wind direction, rain, and snow, as illustrated in *Diagram 3.4.2*. Therefore, I changed their format to number with the code in *Diagram 3.5.1*. After formatting the data, they are able to be processed in the models in the pyspark library.

Diagram 3.5.1 Format the Data

```
#Format the string-typed data to numeric type.
indexer = StringIndexer(inputCol = "cbwd", outputCol = "cbwd_trans")
dataset_final = indexer.fit(dataset_final).transform(dataset_final)
indexer = StringIndexer(inputCol = "rain", outputCol = "rain_trans")
dataset_final = indexer.fit(dataset_final).transform(dataset_final)
indexer = StringIndexer(inputCol = "snow", outputCol = "snow_trans")
dataset_final = indexer.fit(dataset_final).transform(dataset_final)
dataset_final = dataset_final.drop('cbwd','rain','snow')
dataset_final = dataset_final.withColumnRenamed('cbwd_trans','cbwd')
dataset_final = dataset_final.withColumnRenamed('rain_trans','rain')
dataset_final = dataset_final.withColumnRenamed('snow_trans','snow')
dataset_final.show()
```

```
+---+---+----+----+------+-----+---+---+----+-------------+-----+----+----+----+
| No| pm|DEWP|TEMP|  PRES|  Iws| Is| Ir|HUMI|precipitation|Iprec|cbwd|rain|snow|
+---+---+----+----+------+-----+---+---+----+-------------+-----+----+----+----+
| 25|129| -16|-4.0|1020.0| 1.79|  0|  0|38.0|          0.0|  0.0| 0.0| 0.0| 0.0|
| 26|148| -15|-4.0|1020.0| 2.68|  0|  0|42.0|          0.0|  0.0| 0.0| 0.0| 0.0|
| 27|159| -11|-5.0|1021.0| 3.57|  0|  0|63.5|          0.0|  0.0| 0.0| 0.0| 0.0|
| 28|181|  -7|-5.0|1022.0| 5.36|  1|  0|85.0|          0.0|  0.0| 0.0| 0.0| 1.0|
| 29|138|  -7|-5.0|1022.0| 6.25|  2|  0|85.0|          0.0|  0.0| 0.0| 0.0| 1.0|
| 30|109|  -7|-6.0|1022.0| 7.14|  3|  0|92.0|          0.0|  0.0| 0.0| 0.0| 1.0|
| 31|105|  -7|-6.0|1023.0| 8.93|  4|  0|92.0|          0.0|  0.0| 0.0| 0.0| 1.0|
| 32|124|  -7|-5.0|1024.0|10.72|  0|  0|85.0|          0.0|  0.0| 0.0| 0.0| 0.0|
| 33|120|  -8|-6.0|1024.0|12.51|  0|  0|85.0|          0.0|  0.0| 0.0| 0.0| 0.0|
| 34|132|  -7|-5.0|1025.0| 14.3|  0|  0|85.0|          0.0|  0.0| 0.0| 0.0| 0.0|
| 35|140|  -7|-5.0|1026.0|17.43|  1|  0|85.0|          0.0|  0.0| 0.0| 0.0| 1.0|
| 36|152|  -8|-5.0|1026.0|20.56|  0|  0|79.0|          0.0|  0.0| 0.0| 0.0| 0.0|
| 37|148|  -8|-5.0|1026.0|23.69|  0|  0|79.0|          0.0|  0.0| 0.0| 0.0| 0.0|
| 38|164|  -8|-5.0|1025.0|27.71|  0|  0|79.0|          0.0|  0.0| 0.0| 0.0| 0.0|
| 39|158|  -9|-5.0|1025.0|31.73|  0|  0|73.0|          0.0|  0.0| 0.0| 0.0| 0.0|
| 40|154|  -9|-5.0|1025.0|35.75|  0|  0|73.0|          0.0|  0.0| 0.0| 0.0| 0.0|
| 41|159|  -9|-5.0|1026.0|37.54|  0|  0|73.0|          0.0|  0.0| 0.0| 0.0| 0.0|
| 42|164|  -8|-5.0|1027.0|39.33|  0|  0|79.0|          0.0|  0.0| 0.0| 0.0| 0.0|
| 43|170|  -8|-5.0|1027.0|42.46|  0|  0|79.0|          0.0|  0.0| 0.0| 0.0| 0.0|
| 44|149|  -8|-5.0|1028.0|44.25|  0|  0|79.0|          0.0|  0.0| 0.0| 0.0| 0.0|
+---+---+----+----+------+-----+---+---+----+-------------+-----+----+----+----+
only showing top 20 rows
```

4. Data Transformation

4.1 Reducing Data

To make sure the good quality of data mining, it is important to drop the unrelated data. However, I cannot find any build-in feature selection method in PySpark. Therefore, I wrote a function to get the important attributes. As shown in *Diagram 4.1.1*, the amount of the most repeated value in each column will be counted and assessed. If the amount of one value in the feature takes up more than 90% of the instances, it means that the feature is not variant enough, thus being unimportant to predict the target value.

Diagram 4.1.1 Function to Select Important Features

```
#Function to select and keep the useful attributes.
def feature_selection(dataset):
    predictors = get_predictors(dataset)
    instance_no = dataset.select("No").count()
    unimportant_features =[]
    for predictor in predictors:
        dist_value = dataset.select(predictor).groupBy(predictor).count()
        most_repeated = dist_value.agg({"count": "max"}).collect()[0]["max(count)"]
        if most_repeated / instance_no > 0.9:
            unimportant_features.append(predictor)
            dataset.drop(predictor)
    print(unimportant_features)
    return dataset
```

However, Jupyter Notebook was not able to process it when I put it into the whole .ipynb document for data cleaning and data mining. It caused the exception as shown in *Diagram 4.1.2,* which would later lead to the disconnection to Jupyter Notebook. As a result, I could not process the latter steps. Nevertheless, when I put it in a separate file with only itself in it, it worked well and selected the proper features. I reckon that putting it in the whole .ipynb file made the file too big, which exceeded the processability, though I do not know whether it was because of the memory size of my laptop being insufficient, or it was caused by the processability of Jupyter Notebook being insufficient.

Diagram 4.1.2 Exception caused by the feature selection function

```
In [25]: #Call the function to do feature selection.
         dataset_final  = feature_selection(dataset_final)
         dataset_final.show()

         ERROR:root:Exception while sending command.
         Traceback (most recent call last):
           File "/home/ubuntu/spark-2.1.1-bin-hadoop2.7/python/lib/py4j-0.10.4-src.zip/py4j/java_gatew
         ay.py", line 1035, in send_command
             raise Py4JNetworkError("Answer from Java side is empty")
         py4j.protocol.Py4JNetworkError: Answer from Java side is empty

         During handling of the above exception, another exception occurred:

         Traceback (most recent call last):
           File "/home/ubuntu/spark-2.1.1-bin-hadoop2.7/python/lib/py4j-0.10.4-src.zip/py4j/java_gatew
         ay.py", line 883, in send_command
             response = connection.send_command(command)
           File "/home/ubuntu/spark-2.1.1-bin-hadoop2.7/python/lib/py4j-0.10.4-src.zip/py4j/java_gatew
         ay.py", line 1040, in send_command
```

Once again, the code in *Diagram 4.1.1*, which deals with Apache Spark data frame, works well when the file size is small but caused exception in this large-sized file. Therefore, I had to use the code in *Diagram 4.1.3*, which convert the dataset the Pandas format to process feature selection, so I could process the latter steps. After feature selection, the data frame was converted back to Apache Spark format for latter processes.

Diagram 4.1.3 Another Function to Select Important Features

```
#Function to select and keep the useful attributes.
#Those attributes who has more than 90% of the instances with the same value will be deleted.
def feature_selection(dataset):
    predictors = get_predictors(dataset)
    dataset = dataset.toPandas()
    instance_no = dataset.shape[0]
    unimportant_features =[]
    for predictor in predictors:
        predictor_values = dataset[predictor].value_counts()
        for value_amount in predictor_values:
            if value_amount/instance_no > 0.9 :
                unimportant_features.append(predictor)
                dataset = dataset.drop(columns = [predictor])
    print("Unimportant features: ", unimportant_features)
    return dataset
```

The result is shown in *Diagram 4.1.4*, indicating that cumulated hours of snow, cumulated hours of rain, rain, snow, hourly precipitation, and cumulated precipitation are not important. Therefore, I discarded these features in the function in *Diagram 4.1.3*. The remained features are kept for the data mining process.

Diagram 4.1.4 Feature Selection Result for Prepared Data

```
#Call the function to do feature selection.
selected_data = feature_selection(dataset_final)
dataset_final = spark.createDataFrame(selected_data)
dataset_final.show(5)
```

```
Unimportant features:  ['Is', 'Ir', 'precipitation', 'Iprec', 'rain', 'snow']
+---+---+----+----+------+----+----+----+
| No| pm|DEWP|TEMP|  PRES| Iws|HUMI|cbwd|
+---+---+----+----+------+----+----+----+
| 25|129| -16|-4.0|1020.0|1.79|38.0| 0.0|
| 26|148| -15|-4.0|1020.0|2.68|42.0| 0.0|
| 27|159| -11|-5.0|1021.0|3.57|63.5| 0.0|
| 28|181|  -7|-5.0|1022.0|5.36|85.0| 0.0|
| 29|138|  -7|-5.0|1022.0|6.25|85.0| 0.0|
+---+---+----+----+------+----+----+----+
only showing top 5 rows
```

4.2 Projecting Data

Lakshmanan (2019) demonstrated that the field with larger value would inherently affect the data mining result more. Therefore, to ensure the accuracy, it is required to normalize the data in different ranges to a common scale.

In the data exploration chapter, *Diagram 2.4.6* illustrated that the data values in different fields are in different ranges. For example, temperature values are from -19 to 42, while pressure values are between 991 and 1046. This means that pressure will intrinsically put more weight in the data mining process if the data are not projected.

As a result, I normalized all the input data with continuous type by the function in *Diagram 4.2.1*, to make sure the data mining results are as accurate as possible. After normalization, the dataset included many duplicated columns in different data formats. Therefore, I dropped all the duplicated ones. Finally, the continuous input data are between zero and one, as shown in *Diagram 4.2.2*.

Diagram 4.2.1 Function to Normalize Data

```
#Function to project the data of numeric predictors by normalizing them and return the proje
cted dataset.
def normalization(dataset):
    predictors = get_predictors(dataset)
    if "cbwd" in predictors:
        predictors.remove("cbwd")
    columns_to_scale = predictors
    assemblers = [VectorAssembler(inputCols=[col],
                                  outputCol=col + "_vec") for col in columns_to_scale]
    scalers = [MinMaxScaler(inputCol=col + "_vec",
                            outputCol=col + "_scaled") for col in columns_to_scale]
    pipeline = Pipeline(stages=assemblers + scalers)
    scalerModel = pipeline.fit(dataset)
    scaledData = scalerModel.transform(dataset)
    return scaledData
```

Diagram 4.2.2 Data Range After Projecting Data

```
#Drop duplicated columns after normalization.
dataset_final = dataset_final.drop("No","DEWP", "TEMP", "PRES","Iws","HUMI","DEWP_vec",
                                   "TEMP_vec", "PRES_vec","Iws_vec","HUMI_vec")
dataset_final.show(5)
```

```
+---+----+--------------------+--------------------+--------------------+--------------------+--------------------+
| pm|cbwd|         DEWP_scaled|         TEMP_scaled|         PRES_scaled|          Iws_scaled|         HUMI_scaled|
+---+----+--------------------+--------------------+--------------------+--------------------+--------------------+
|129| 0.0|[0.35294117647058...|[0.2459016393442623]|[0.5272727272727272]|[0.00237151352116...|[0.3673469387755102]|
|148| 0.0|[0.36764705882352...|[0.2459016393442623]|[0.5272727272727272]|[0.00394662324791...|[0.40816326530612...|
|159| 0.0|[0.4264705882352941]|[0.22950819672131...|[0.5454545454545454]|[0.00552173297465...|[0.6275510204081632]|
|181| 0.0|[0.4852941176470588]|[0.22950819672131...|[0.5636363636363636]|[0.00868965029024...|[0.8469387755102041]|
|138| 0.0|[0.4852941176470588]|[0.22950819672131...|[0.5636363636363636]|[0.01026476001698...|[0.8469387755102041]|
+---+----+--------------------+--------------------+--------------------+--------------------+--------------------+
only showing top 5 rows
```

5. Data-Mining Method Selection

5.1 Matching and Discussing Data-Mining Objectives

In the situation understanding chapter, I have discussed that the purpose for this study is to control PM2.5 concentration by controlling the main related meteorology factor, so that the premature mortality based on air pollution can be reduced.

In order to achieve this, the first data mining objective is to use related meteorology data to predict PM2.5 concentration. In order to achieve this, the data-mining method must be able to give a prediction for the target field – PM2.5 concentration based on the input data.

Besides, the data mining objectives include analyzing how the meteorology factors affect PM2.5 concentration, which requires the data-mining method to be able to explain how the model get the prediction. In other words, the method should include algorithms that can illustrate the calculation process.

The last data mining objective is to find out which meteorology elements affect PM2.5 concentration more, so that an effective solution to control PM2.5 concentration by controlling these elements can be made. This gives the data-mining method a similar requirement as the second data mining objective, which is the ability to display the details of the predicting process in the model.

As we discussed above, it is evident that PM2.5 concentration is the target of this data mining project. Therefore, it is confirmed that this data mining project is a supervised machine learning project, which means that we use the input fields – the meteorology data to get the output – PM2.5 concentration. Supervised methods can find out the function that is able to give out a prediction by taking the input data (Brownlee, 2019).

Since this project is confirmed as a supervised machine learning problem, the method is going to be chosen between the two main groups of supervised learning methods – classification method and regression method (Brownlee, 2019). As classification method is to predict a categorical or ordinal output in a finite class (Simplilearn, n.d.), it is not suitable for this project, whose target – PM2.5 concentration is a continuous value.

Therefore, regression method should be chosen for this data mining project which needs to predict a continuous value. Regression method, as a supervised method, is to use historical data to find out a prediction for a continuous value from the input values (Dave, 2018).

Previously, the data understanding chapter affirmed that a PM2.5 concentration is a continuous value between 1 and 994 ug/m^3, as shown in *Diagrams 2.2.12 and 2.3.16*, so it can be predicted by regression method. Also, the predicting process can be analyzed in regression method so that details of the calculation will be illustrated to meet the data mining objectives which aim to find out the most influential meteorology and how they affect PM2.5 concentration.

Besides, the situation understanding chapter stated that this study would be estimated as successful if PM2.5 concentration can be predicted, and the meteorology elements influencing PM2.5 concentration can be found. With regression method, I am confident that PM2.5 concentration will be accurately predicted, and the meteorology elements will be found during the predicting process.

5.2 Selecting Data-Mining Method

In conclusion, based on these objectives and success criteria of this study, this data mining project aims to use meteorology data to predict PM2.5 concentration, so it is a supervised machine learning problem. Besides, Since PM2.5 concentration is a numeric value, this project needs regression method to give a continuous-typed prediction. Therefore, I chose regression method, which is able to predict PM2.5 concentration and to illustrate the detailed data mining process.

6. Data-Mining Algorithms Selection

6.1 Exploratory Analysis and Discuss

After selecting regression method for this study, I started exploring the algorithms for regression problems, that are available in pyspark libraries, which will be used to build models and to mine the data in the latter steps.

The first area I explored is statistical regression algorithms, which include logistic regression algorithm, multivariate regression algorithm, multiple regression algorithm, linear regression algorithm, and so on.

Logistic regression algorithm is often used to predict a binary result (Statistics Solutions, n.d.). However, the data mining objective of this study is to predict PM2.5 concentration, which is a numeric value. Therefore, logistic regression algorithm is not suitable for the study. Besides, multivariate regression algorithm is used to calculate at least two targets from more than one input values (Vaga & Rai, n.d.), which is not appropriate for this project whose goal only has one target value – PM2.5 concentration. Also, multiple regression algorithm contains linear and non-linear regression learnings that take nominal predictors. Nevertheless, the data mining objective is to predict PM2.5 concentration from many continuous predictors and to find how these numeric data affect PM2.5

concentration. Thus, it is not proper to choose multiple regression algorithm for this project.

However, linear regression algorithm, which aims to predict a single continuous variable from the predictors and to find out the relationships between them (Bhatia, 2017), is appropriate for this data mining project which aims to predict PM2.5 concentration – a single continuous target from the meteorology data which include many meteorology factors. Therefore, I chose linear regression algorithm for this project.

Afterwards, I looked into tree-building algorithms, which can list the details of each step. As the data mining objectives request to find out how the meteorology factors influence PM2.5 concentration and which factors have more effect, the details of each step provided by regression trees can explain this clearly. Besides, using the tree to predict is easy and effective, because each step will reduce some data by the filter and leave fewer data to work on (Srivastava, 2018).

In pyspark libraries, there are two main kinds of decision trees, which are the decision tree classifier and the decision tree regressor (Apache Spark, n.d). Although decision tree classifier has successful performance in many problems, it is designed for projects that aim to predict a flag or a categorical result (Pantazi, Moshou, & Bochtis, 2020). Therefore, decision tree classifier is not suitable for this study, which is going to predict PM2.5 concentration – a continuous variable.

Nevertheless, the decision tree regressor is suitable for this project, as it allows the target to be a numeric value. The decision tree regressor has the advantages of tree algorithms, that can be useful to naturally visualise the interactions between the attributes in the dataset. Also, it can explain the process of the algorithm by providing the tree structure. Therefore, I chose the decision tree regressor algorithm for this project.

6.2 Selecting Data-Mining Algorithms

In conclusion, based on the data mining objectives, the project needs to predict PM2.5 concentration from data of meteorology factors and to find out how the meteorology elements influence PM2.5 concentration. The linear regression is used to predict a single continuous value from multiple predictors, while the decision tree regressor can generate a regression tree to predict the numeric result from filtering out the data effectively. Therefore, I chose the linear regression and the decision tree regressor as the algorithms to conduct the data mining process.

6.3 Selecting Models and Choosing Parameters

As discussed in the selecting data-mining algorithms chapter, I have decided to use the linear regression algorithm and the decision tree regressor algorithm for this study. Fortunately, pyspark libraries have the linear models which include the linear regression model and the tree models which include the decision tree regressor model.

Firstly, I set the predictors for both algorithms, as shown in *Diagram 6.3.1*. The predictors are dew point, temperature, pressure, cumulated wind speed, humidity and combined wind direction.

Diagram 6.3.1 Code to Set Target and Predictors for Both Models

```
#Set predictors as input columns.
assembler = VectorAssembler(
    inputCols=['DEWP_scaled','TEMP_scaled','PRES_scaled','Iws_scaled','HUMI_scaled','cbwd'],
    outputCol="predictors")
dataset_final = assembler.transform(dataset_final)
dataset_final.head(1)
```

```
[Row(pm=129, cbwd=0.0, DEWP_scaled=DenseVector([0.3529]), TEMP_scaled=DenseVector([0.2459]),
PRES_scaled=DenseVector([0.5273]), Iws_scaled=DenseVector([0.0024]), HUMI_scaled=DenseVector
([0.3673]), predictors=DenseVector([0.3529, 0.2459, 0.5273, 0.0024, 0.3673, 0.0]))]
```

After selecting the predictors, I chose the dense column with all predictor values and the target – PM2.5 concentration as the final dataset for the data mining process in latter steps, as shown in *Diagram 6.3.2*.

Diagram 6.3.2 Select Final Dataset for Data Mining

```
#Select predictors and the target for data mining.
dataset_final = dataset_final.select("predictors","pm")
dataset_final.show()
```

```
+--------------------+---+
|          predictors| pm|
+--------------------+---+
|[0.35294117647058...|129|
|[0.36764705882352...|148|
|[0.42647058823529...|159|
|[0.48529411764705...|181|
|[0.48529411764705...|138|
|[0.48529411764705...|109|
|[0.48529411764705...|105|
|[0.48529411764705...|124|
|[0.47058823529411...|120|
|[0.48529411764705...|132|
|[0.48529411764705...|140|
|[0.47058823529411...|152|
|[0.47058823529411...|148|
|[0.47058823529411...|164|
|[0.45588235294117...|158|
|[0.45588235294117...|154|
|[0.45588235294117...|159|
|[0.47058823529411...|164|
|[0.47058823529411...|170|
|[0.47058823529411...|149|
+--------------------+---+
only showing top 20 rows
```

6.3.1 Linear Regression Model

I built the linear regression model and chose the parameters by the code in *Diagram 6.3.3*.

Diagram 6.3.3 Build Linear Regression Model

```
#Build and choose the parameters for Linear Regression Model.
lr = LinearRegression(featuresCol='predictors', labelCol='pm',
                      fitIntercept=True, standardization=True)
```

For the first conduction, I built a standard model to see a general result, that most of the parameters are set as default, as shown in *Diagram 6.3.3*, because it is easier and faster to get the result for the first time. Later in the reiteration chapter, I will change the parameters to see if the results are going to be changed.

As illustrated in *Diagram 6.3.3*, I set the dense column with all predictors as "featureCol", while PM2.5 concentration – the target as "labelCol".

Besides, I set "*fitIntercept*" as true. As a result, the model will calculate an intercept value, which is the point where the result line crosses the y-axis. In the first round, I will not force the line to run through the origin point. Also, I set "*standardization*" as true. Although I have done normalization carefully in the data transformation chapter, there might still be some area that computer can better deal with rather than human.

6.3.2 Decision Tree Regressor Model

I built the decision tree regressor model and chose the parameters by the code in *Diagram 6.3.4*.

Diagram 6.3.4 Code to Build Decision Tree Regressor Model

```
#Build and choose the parameters for Desicion Tree Regressor Model.
tree = DecisionTreeRegressor(featuresCol="predictors", labelCol='pm',
                             maxDepth=5, minInstancesPerNode=1)
```

Same as in the linear model, in the decision tree regression model, a standard model was built for the first conduction, that most of the parameters were set as default. Later in the reiteration chapter, I will change the parameters to see if the results are going to be changed.

As illustrated in *Diagram 6.3.4*, I set the dense column with all predictors as "featureCol", while PM2.5 concentration – the target as "labelCol". Also, "*maxDepth*" is set as five, which is neither too deep that will take a very long time, nor too shallow that not enough to find trustworthy relationships between PM2.5 concentration and the meteorology factors. Lastly, "minInstancesPerNode" is set as 1. The set of this parameter is based on the find in the data exploration chapter. In my dataset, there are some attributes with data which are not fit the normal distribution. For example, *Diagram 2.3.8* illustrated that most values for cumulated wind speed are smaller than 13. Therefore, when the cumulated wind speed is high, they have very few data. However, they may still need to be split.

7. Data Mining

7.1 Creating and Justifying Test Design

Information Gain Ltd (2012) explained that in a data mining project, selecting 70% of the data to do the training and analysis, while leaving the rest 30% for testing is a good choice. This can make sure that the model will have enough data to conduct the training process. Meanwhile, the portion of the testing data is sufficient for the accuracy of the error estimate.

Also, Information Gain Ltd (2012) indicated that the ratio 70:30 is not randomly chosen. Research has found that if the testing data is less than 10%, the result can be very inaccurate. Besides, when the percentage of training data is between 40% and 80%, the performance of the data mining process tends to be better.

Therefore, I decided to set 70% of the data in this study for training, while using the rest 30% to test the training result. *Diagram 7.1.1* illustrates the code I used to set training data and testing data and displays the descriptions of training data and testing data.

Diagram 7.1.1 Create Test Design

```
#Split the dataset into two parts, 70% for training and 30% for testing.
data_train,data_test = dataset_final.randomSplit([0.7,0.3])
data_train.describe().show()
data_test.describe().show()
```

```
+-------+-----------------+
|summary|               pm|
+-------+-----------------+
|  count|            29229|
|   mean|98.43949502206712|
| stddev|91.88988295826178|
|    min|                0|
|    max|              994|
+-------+-----------------+

+-------+-----------------+
|summary|               pm|
+-------+-----------------+
|  count|            12503|
|   mean|99.01015756218507|
| stddev|92.48507889615608|
|    min|                0|
|    max|              980|
+-------+-----------------+
```

7.2 Conducting Data Mining

After creating the test design, I conducted the data mining with both algorithms respectively by the code in *Diagrams 7.2.1 and 7.2.2*.

Diagram 7.2.1 Code to Conduct the Linear Regression Model

```
#Conduct the Linear Regression Model.
lr_model = lr.fit(data_train)
lr_test = lr_model.evaluate(data_test)
```

Diagram 7.2.2 Code to Conduct the Decision Tree Regressor Model

```
#Conduct the Desicion Tree Regressor Model.
tree_model = tree.fit(data_train)
tree_train = tree_model.transform(data_train)
tree_test = tree_model.transform(data_test)
```

Afterwards, because these two models have different build-in attributes, I analyzed the linear regression model by the function in *Diagram 7.2.3* and the decision tree regressor model by the function in *Diagram 7.2.4*. These two functions calculated the statistical data of predictions versus actual records. Both training data and testing data are analyzed.

Diagram 7.2.3 Function to Analyze the Results for Linear Regression Model

```
#Function to analyse the result of the linear regression model by calculating the statistical data
of predictions versus actual record.
def analyze_result_lr(model):
    rmse = model.rootMeanSquaredError
    r2 = model.r2
    mae = model.meanAbsoluteError
    print("Root Mean Squared Error:","\t",rmse)
    print("R2 Score:","\t",r2)
    print("Mean Absolute Error:","\t",mae)
    pred_data = model.predictions.select("pm","prediction")
    res = model.residuals
    max_e = res.agg({"residuals":"max"}).collect()[0]
    print("Maximum Error:","\t",max_e["max(residuals)"])
    min_e = res.agg({"residuals":"min"}).collect()[0]
    print("Minimum Error:","\t",min_e["min(residuals)"])
    mean_pm = pred_data.agg({"pm":"mean"}).collect()[0]
    mean_pred = pred_data.agg({"prediction":"mean"}).collect()[0]
    print("Mean Error:","\t", abs(mean_pm["avg(pm)"] - mean_pred["avg(prediction)"]))
```

Diagram 7.2.4 Function to Analyze the Results for Decision Tree Regressor Model

```
#Function to analyse the result of the decision tree regressor model by calculating the statistica
l data of predictions versus actual record.
def analyze_result_tree(model):
    rmse_evaluator = RegressionEvaluator(labelCol="pm", predictionCol="prediction",
                                         metricName="rmse")
    r2_evaluator = RegressionEvaluator(labelCol="pm", predictionCol="prediction",
                                         metricName="r2")
    mae_evaluator = RegressionEvaluator(labelCol="pm", predictionCol="prediction",
                                         metricName="mae")
    rmse = rmse_evaluator.evaluate(model)
    r2 = r2_evaluator.evaluate(model)
    mae = mae_evaluator.evaluate(model)
    print("Root Mean Squared Error:","\t",rmse)
    print("R2 Score:","\t",r2)
    print("Mean Absolute Error:","\t",mae)
    pred_data = model.select("pm","prediction")
    res = pred_data.withColumn('residuals',pred_data['pm']-pred_data['prediction'])
    max_e = res.agg({"residuals":"max"}).collect()[0]
    print("Maximum Error:","\t",max_e["max(residuals)"])
    min_e = res.agg({"residuals":"min"}).collect()[0]
    print("Minimum Error:","\t",min_e["min(residuals)"])
    mean_pm = pred_data.agg({"pm":"mean"}).collect()[0]
    mean_pred = pred_data.agg({"prediction":"mean"}).collect()[0]
    print("Mean Error:","\t", abs(mean_pm["avg(pm)"] - mean_pred["avg(prediction)"]))
```

The analysis function in *Diagrams* 7.2.3 and 7.2.4 gave me the results of both algorithms, which are shown in *Diagrams 7.2.5, 7.2.6, 7.2.7 and 7.2.8*. From the analysis, both models are acceptable. Firstly, all the statistical values in the training data and the testing data are similar in the same model. Besides, as the values of PM2.5 concentration vary from 1 to 994, as mentioned in *Diagram 2.4.6* in the data understanding chapter, all the errors from both models are in the reasonable range. The R2 scores of the linear regression model are imperceptibly smaller than 0.25, while the R2 scores of the decision tree regressor model are greater than 0.25, which means the models are meaningful.

Diagram 7.2.5 Results of Training Data in Linear Regression Algorithm

```
#Call the function to analyze the training data of Linear Regression Model.
analyze_result_lr(lr_model.summary)
```

```
Root Mean Squared Error:        79.78121392496766
R2 Score:       0.2461572852603613
Mean Absolute Error:    57.360686405664545
Maximum Error:  885.7038771113768
Minimum Error:  -175.94883782134025
Mean Error:     7.105427357601002e-12
```

Diagram 7.2.6 Results of Testing Data in Linear Regression Algorithm

```
#Call the function to analyze the testing data of Linear Regression Model.
analyze_result_lr(lr_test)
```

```
Root Mean Squared Error:          80.35582338243803
R2 Score:         0.24503624011009872
Mean Absolute Error:      57.541803619214285
Maximum Error:    854.1134380188765
Minimum Error:    -164.04099493493368
Mean Error:       0.2581468556162605
```

Diagram 7.2.7 Results of Training Data in Decision Tree Regressor Algorithm

```
#Call the function to analyze the training data of Desicion Tree Regressor
analyze_result_tree(tree_train)
```

```
Root Mean Squared Error:          75.54769924808232
R2 Score:         0.3240385212329048
Mean Absolute Error:      52.84662247797476
Maximum Error:    922.8030303030303
Minimum Error:    -225.79834024896266
Mean Error:       1.6143530956469476e-11
```

Diagram 7.2.8 Results of Testing Data in Decision Tree Regressor Algorithm

```
#Call the function to analyze the testing data of Desicion Tree Regressor M
analyze_result_tree(tree_test)
```

```
Root Mean Squared Error:          76.59038462237937
R2 Score:         0.31413302020280465
Mean Absolute Error:      53.57778568631454
Maximum Error:    816.1850631521143
Minimum Error:    -219.79834024896266
Mean Error:       0.2124201689506009
```

Comparing these results in the two models in *Diagrams 7.2.5, 7.2.6, 7.2.7* and *7.2.8*, I have found that the decision tree regressor model is better than the linear regression model in this case. It has slighter errors and better R2 score, which means it is more accurate and meaningful. Later in the reiteration chapter, I will change the model parameters to see if the results are going to be different.

7.3 Searching for Patterns

After conducting the data mining process, I have found some interesting patterns.

Diagrams 7.3.1 and 7.3.2 show part of residuals of the two models.

Diagram 7.3.1 Residuals of Linear Regression Model

```
#Show the residuals of Linear Regression Model.
lr_test.residuals.show()
```

```
+-------------------+
|          residuals|
+-------------------+
|-30.479418828320263|
| -34.59632699864652|
|-29.878398645887785|
|-27.113125942454232|
|-38.786275041515665|
| -31.69835600638838|
|-28.013051030370406|
| -40.86852558946346|
| -40.39780534895549|
|  -41.3213620129016|
|-48.270278070786446|
| -51.17020360013538|
|-34.390806354620395|
| -32.86714136886937|
| -43.53007422122397|
|  -54.7016970065644|
| -56.55963521961331|
| -26.11874752651383|
| -36.32922266096875|
| 10.561795977916773|
+-------------------+
only showing top 20 rows
```

Diagram 7.3.2 Residuals of Decision Tree Regressor Model

```
#Show the residuals of Desicion Tree Regressor Model.
predict_data = tree_test.select("pm","prediction")
residuals = predict_data.withColumn('residuals',
            predict_data['pm']-predict_data['prediction']).select("residuals")
residuals.show()
```

```
+-------------------+
|          residuals|
+-------------------+
|-12.026182432432432|
|-12.026182432432432|
|-10.026182432432432|
| -8.026182432432432|
| -24.22249093107618|
| -8.026182432432432|
| -6.026182432432432|
| -25.22249093107618|
| -23.22249093107618|
|-26.706521739130437|
| -26.22249093107618|
|-21.657534246575345|
|-12.026182432432432|
| -8.026182432432432|
|-31.706521739130437|
| -26.22249093107618|
| -26.22249093107618|
| -5.026182432432432|
|-10.026182432432432|
| 12.973817567567568|
+-------------------+
only showing top 20 rows
```

To have a better view of the overall residuals, I used the function in *Diagram 7.3.3* to visualize the pattern of the residuals to assess the success of the models that conducted the data mining process.

Diagram 7.3.3 Function to visualize the residuals

```
#Function to visualize the residuals of the model.
def visualize_residuals(test_result):
    predictions = test_result.select("prediction").toPandas()
    observations = test_result.select("pm").toPandas()
    sns.residplot(predictions, observations)
    plt.show()
    sns.distplot(observations, kde=False, norm_hist=True, color='blue', label='observation')
    sns.distplot(predictions, kde=False, norm_hist=True, color='red', label='prediction')
    plt.title('Comparison of distributions', fontsize=16)
    plt.legend()
    plt.show()
```

Diagrams 7.3.4 and 7.3.5 show the residuals plot and the comparison of distributions in the linear regression model, while *Diagrams 7.3.6 and 7.3.7* illustrate the plot and the comparison in the decision tree regressor model. The plots and comparisons show that the predictions are quite in accordance with the actual record except for the outliers.

Diagram 7.3.4 Residuals Plot of the Linear Regression Model

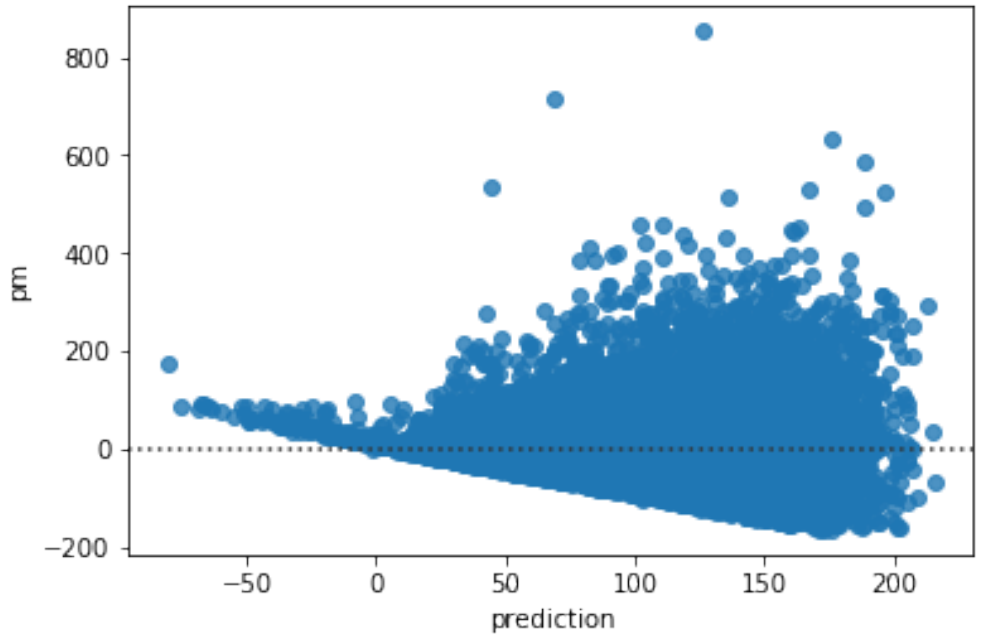


Diagram 7.3.5 Comparison of Distribution in the Linear Regression Model

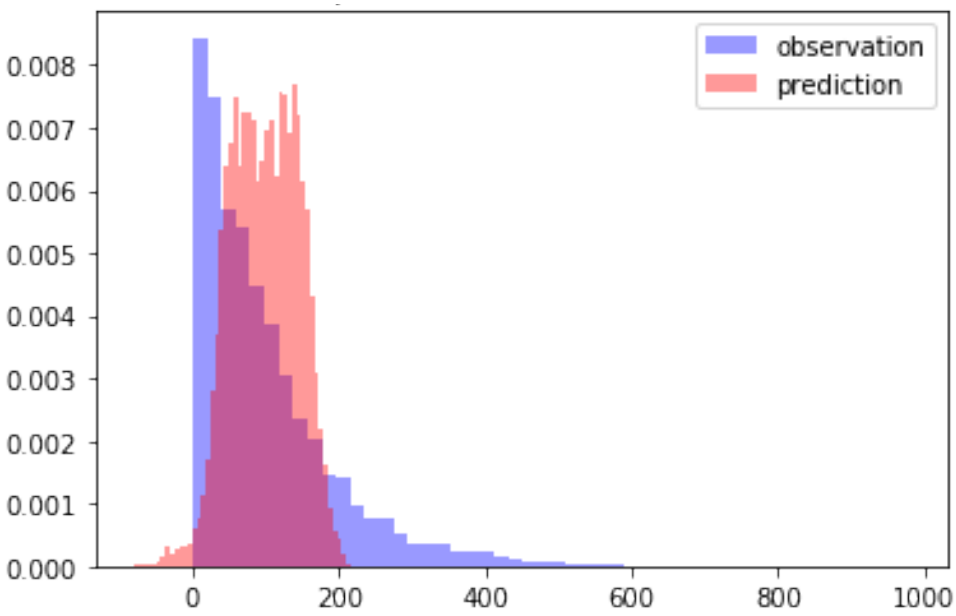


Diagram 7.3.6 Residuals Plot of the Decision Tree Regressor Model

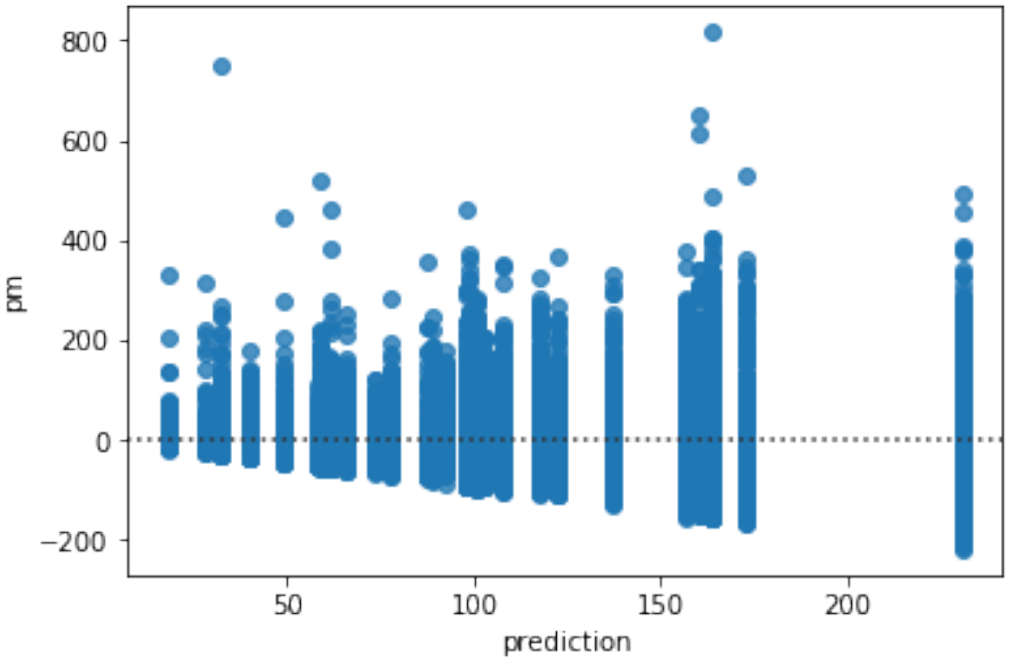
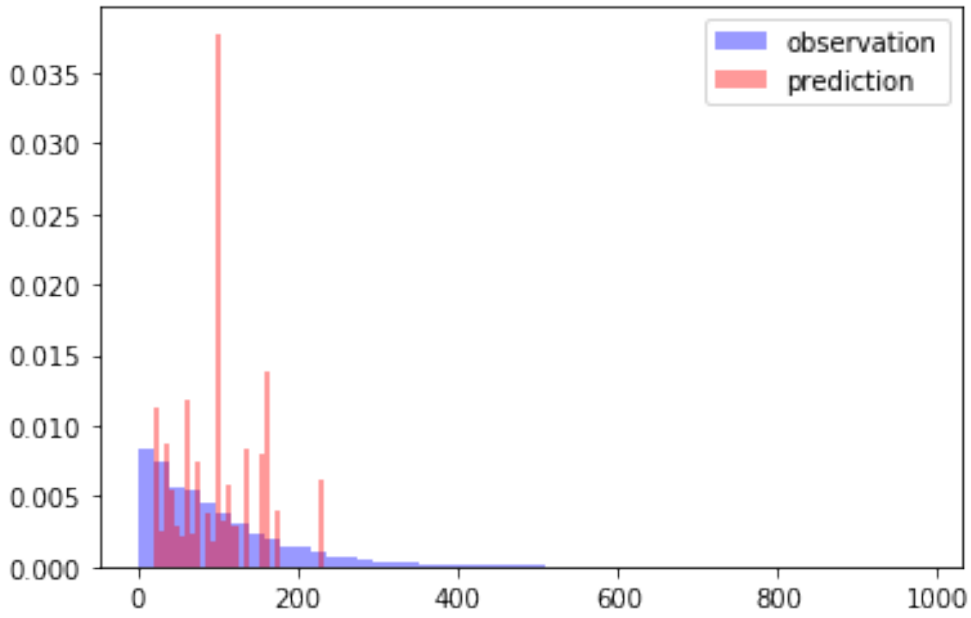


Diagram 7.3.7 Comparison of Distribution in the Decision Tree Regressor Model



To further explore the patterns of predictions and the actual records, I used the function in *Diagram 7.3.8* to show the comparison of actual PM2.5 concentration values and the predictions.

Diagram 7.3.8 Function to Compare Actual Values with Predictions

```
#Function to compare and visualize the values of actual record and predictions.
def prediction_actual_comparison_visulisation(test_result):
    predictions = test_result.select("prediction").toPandas()
    observations = test_result.select("pm").toPandas()
    plt.scatter(observations,predictions)
    plt.xlabel('Actual Values')
    plt.ylabel('Predicted Values')
    plt.show()
```

Diagrams 7.3.9 shows the comparison of predicted PM2.5 concentration and the real record in the linear regression model, while *Diagram 7.3.10* illustrate the comparison in the decision tree regressor model. In both models, the predicted values excluded all the outliers in the real record.

Diagram 7.3.9 Predicted Target with Actual Records in Linear Regression Model

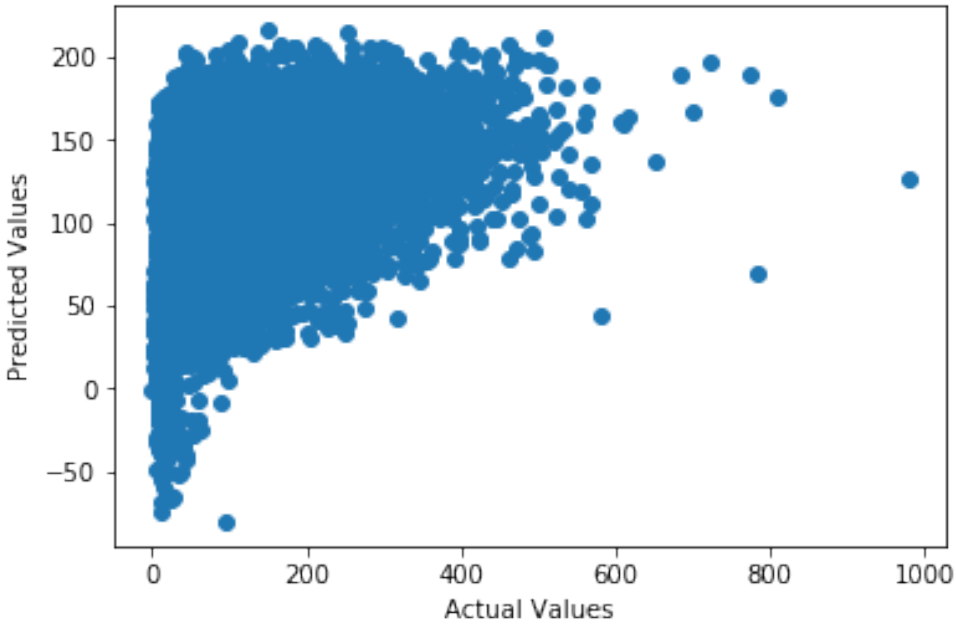
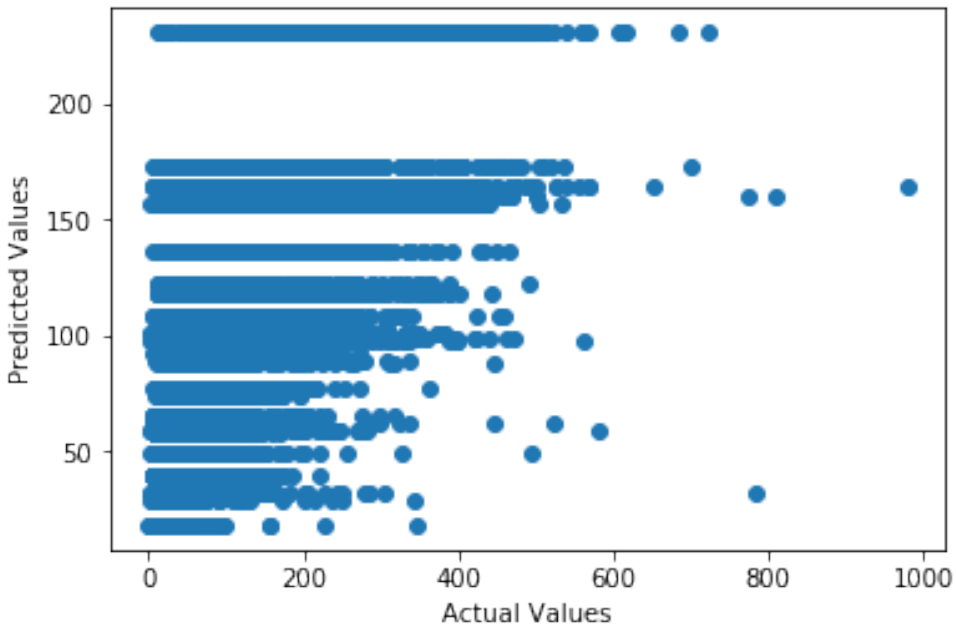


Diagram 7.3.10 Predicted Target with Actual Records in Decision Tree Regressor Model



The depth of the decision tree is five, while it has 63 nodes, as shown in *Diagram 7.3.11*.

Diagram 7.3.11 Depth and Node Numbers of the Decision Tree

```
#Show the depth of the Desicion Tree.
tree_model.depth
```

```
5
```

```
#Show the amount of the nodes of the Desicion Tree.
tree_model.numNodes
```

```
63
```

Diagram 7.3.12 demonstrates part of the decision tree. As we can see, for most of the time, it did not stop until reaching the depth of five.

Diagram 7.3.12 Part of the Decision Tree

```
#Show the detailed Desision Tree.
tree_detail = tree_model.toDebugString
for i, feat in enumerate(predictors):
    tree_detail = tree_detail.replace('feature ' + str(i), feat)
print(tree_detail)
```

```
DecisionTreeRegressionModel (uid=DecisionTreeRegressor_42c7a819721
6810e26c8) of depth 5 with 63 nodes
  If (HUMI <= 0.4489795918367347)
   If (cbwd <= 0.0)
    If (HUMI <= 0.2857142857142857)
     If (HUMI <= 0.16326530612244897)
      If (HUMI <= 0.09183673469387756)
       Predict: 39.657534246575345
      Else (HUMI > 0.09183673469387756)
       Predict: 58.699095022624434
     Else (HUMI > 0.16326530612244897)
      If (TEMP <= 0.6885245901639344)
       Predict: 77.64772727272727
      Else (TEMP > 0.6885245901639344)
       Predict: 57.84183673469388
    Else (HUMI > 0.2857142857142857)
     If (TEMP <= 0.6065573770491803)
      If (Iws <= 0.02690075038935297)
       Predict: 122.74798387096774
      Else (Iws > 0.02690075038935297)
       Predict: 92.6225352112676
```

8. Interpretation

8.1 Studying and Discussing the Mined Patterns

In the patterns of the comparisons of distributions, *Diagrams 7.3.5 and 7.3.7* showed that the distributions of the residuals of the predictions in both models are quite in accordance with actual records except for the outliers. Although the predicted results of the linear regression model are most in accordance with the normal distribution, which means that the distribution of the predicted results is mainly ideal. As for the decision tree model, the distribution is not very ideal. Therefore, later in the reiteration step, I will adjust the parameters of the decision tree regressor model to see if it can have a better performance.

With regard to the residuals plots of the two models, *Diagrams 7.3.4 and 7.3.6* both illustrated that the actual values are in a wider range compared to the predictions. Since both of the models gave similar patterns, it means that the outliers in the data of PM2.5 concentration may not affect the result much. Therefore, later in the reiteration part, I will delete these outliers when recleaning the data in the reiteration chapter to keep all the values of PM2.5 concentration lower than 374 ug/m^3. The value 374 ug/m^3, which is the upper limit of the normal values of PM2.5 concentration, is calculated by the function in *Diagram 2.4.7*.

As for the comparison patterns, *Diagrams 7.3.9 and 7.3.10 both* showed that the range of the real values of PM2.5 concentration is much wider than the predicted values of PM2.5 concentration, which is similar with the residuals plot patterns. However, the patterns here also indicated that the amount of real values in the outlier range is small. This proves that the outliers did not affect the result much. Therefore, as mentioned above, later in the reiteration chapter, these outliers will be removed from the dataset at the data cleaning step.

Lastly, the pattern of the tree depth, which illustrated in *Diagram 7.3.12*, showed that most of the branches did not stop until it reached the depth of

five. This means that it may be better to have a deeper tree depth to make the model have better performance. Therefore, later in the reiteration chapter, the max depth of the decision tree will be set to ten to see if the results of the model can be better.

8.2 Visualizing Data, Results, Models, and Patterns

Function in *Diagram 8.2.1* was used to visualize the predictor importance of the model with a bar graph.

Diagram 8.2.1 Function to Visualize the Predictor Importance

```
#Function visualizes the feature importance after conducting the model.
def feature_importance(importance_,predictors_):
    importance_value = []
    for value in importance_:
        importance_value.append(abs(value))
    sum_values = sum(importance_value)
    i=0
    while i< len(importance_value):
        importance_value[i] = importance_value[i] / sum_values
        i+=1
    importance_df = pd.Series(importance_value, index= predictors_)
    importance_df.plot(kind='bar')
    plt.show()
```

Diagram 8.2.2 indicates that the linear regression model thinks humidity and cumulated wind speed are the two most important factors that affect PM2.5 concentration, followed by dew point and pressure. Combined wind direction seems to affect PM2.5 concentration least.

Diagram 8.2.2 Predictor Importance Analyzed by Linear Regression Model

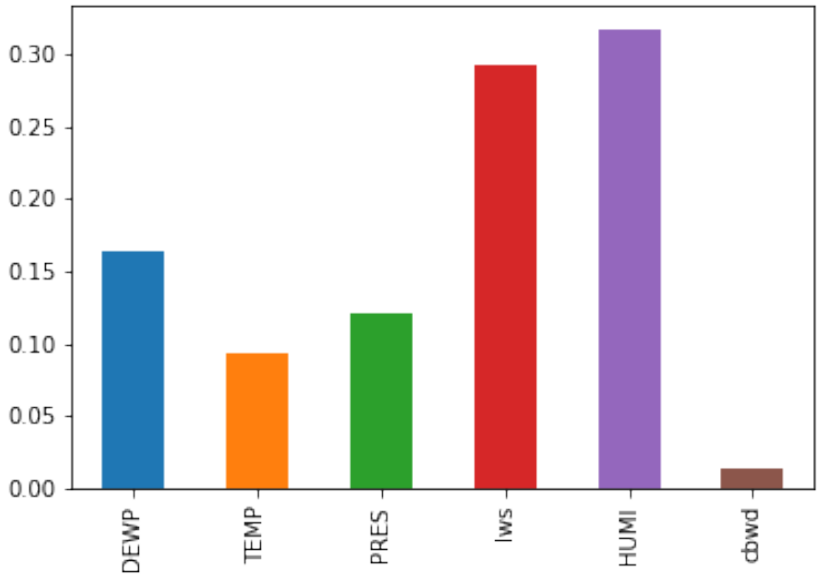


Diagram 8.2.3 shows that the decision tree regressor model believes that humidity is much more important than the other meteorology factors in predicting PM2.5 concentration, followed by temperature. The others do not affect the result that much. Pressure affects PM2.5 concentration very little.

This result is a bit different from the linear regression result. However, they both agree that humidity is the top reason that affects PM2.5 concentration.

Diagram 8.2.3 Predictor Importance Analyzed by Decision Tree Regressor Model

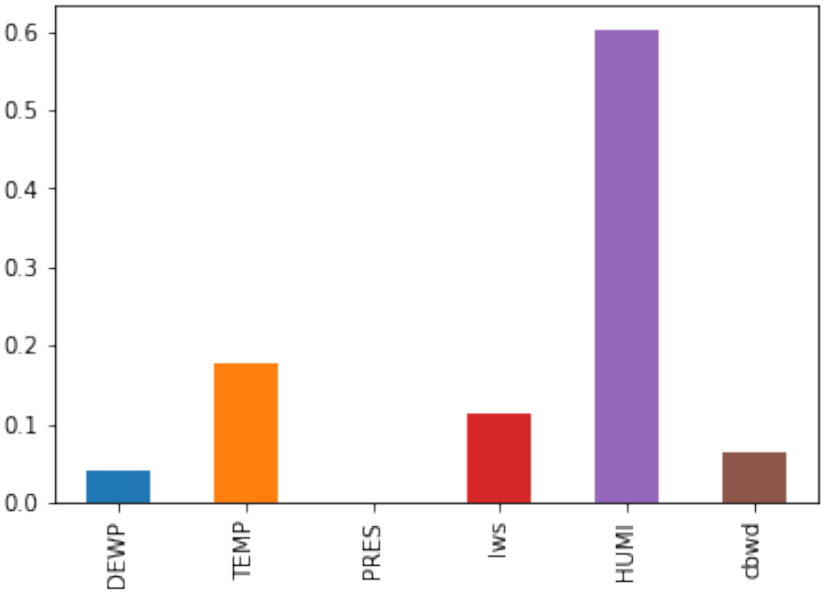


Diagram 8.2.4 illustrates the function I used to visualize the coefficients of the meteorology factors.

Diagram 8.2.4 Function to Visualize the Coefficients of Meteorology Factors

```
#Function to visualize whether the predictors affected the target positively or negatively.
def coefficients(coef_, predictors_):
    colors = []
    related = [1] * len(predictors_)
    for value in coef_:
        if value > 0:
            colors.append('blue')
        else:
            colors.append('orange')
    related_df = pd.Series(related, index = predictors_)
    related_df.plot(kind='bar',color = colors).legend(['Negative','Positive'])
    plt.gca().axes.get_yaxis().set_visible(False)
    plt.show()
```

Diagram 8.2.5 indicates that when humidity is higher, PM2.5 concentration tends to be higher. Other meteorology factors with continuous values influence PM 2.5 in a negative way. When they go lower, PM2.5 concentration tends to go higher, including cumulated wind speed, pressure, temperature, and dew point. As for the combined wind direction, after label encoding, 0, 1, 2 and 3 represent SE, NW, cv and NE respectively. Therefore, according to *Diagram 8.2.5*, when the wind is from the northeast, PM 2.5 concentration can go lower. However, when it is the southeast wind, PM2.5 tends to go higher.

Diagram 8.2.5 Coefficients of Meteorology Factors

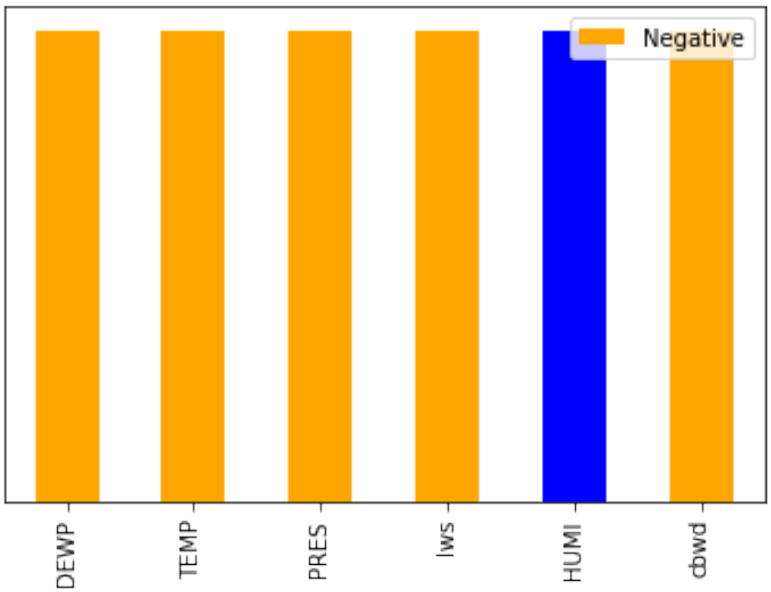


Diagram 8.2.6 illustrates the function I used to visualize how each predictor affects the target respectively in details.

Diagram 8.2.6 Function to Visualize the Effect from Each Predictor

```
#Function to visualize how each attribute affects the target respectively.
def effect_graph(coef_, predictors_, intercept_):
    pred = predictors_
    pred.remove('cbwd')
    coeff = coef_[:-1]
    i=0
    for value in coeff:
        x=np.linspace(0,1,100)
        y=value * x + intercept_
        plt.plot(x,y)
        plt.xlabel(pred[i])
        plt.ylabel('pm2.5')
        plt.grid()
        plt.show()
        i+=1
```

Diagram 8.2.7 shows how cumulated wind speed affects PM2.5 averagely when the other meteorology factors are not changing.

Diagram 8.2.7 PM2.5 Affected by Cumulated Wind Speed

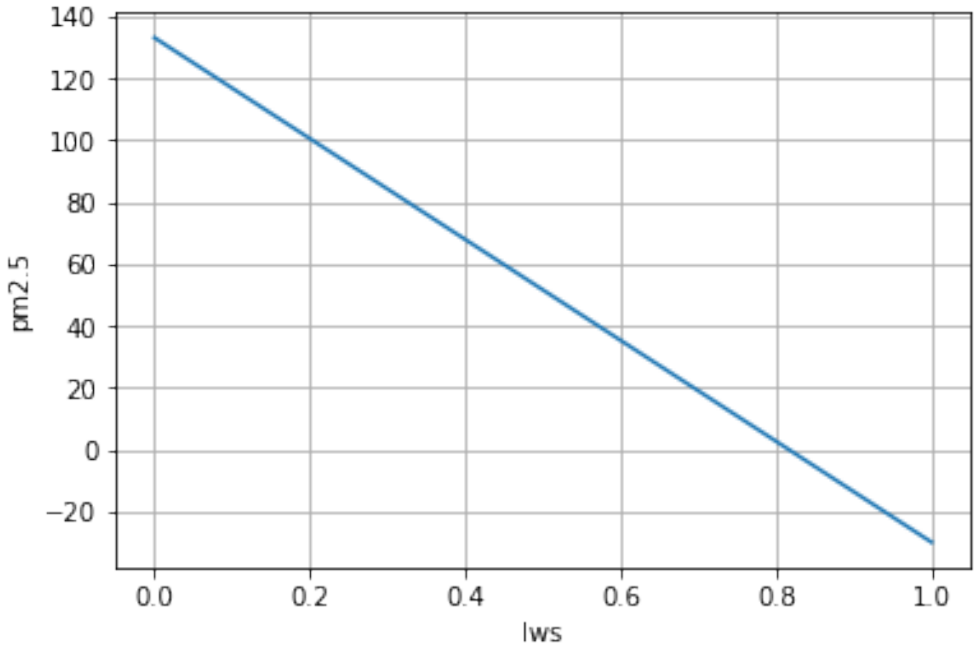


Diagram 8.2.8 shows how pressure affects PM2.5 averagely when the other meteorology factors are not changing.

Diagram 8.2.8 PM2.5 Affected by Pressure

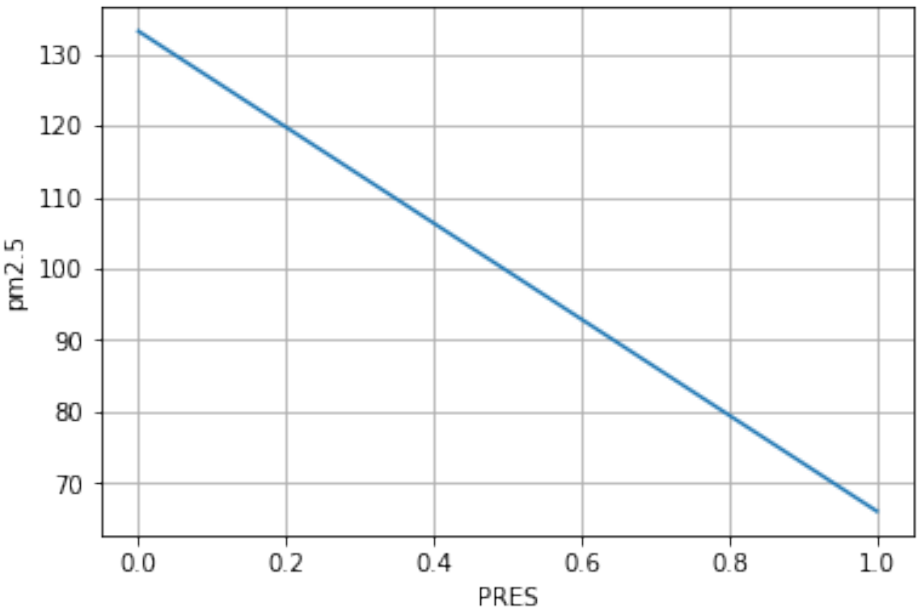


Diagram 8.2.9 shows how dew point affects PM2.5 averagely when the other meteorology factors are not changing.

Diagram 8.2.9 PM2.5 Affected by Dew Point

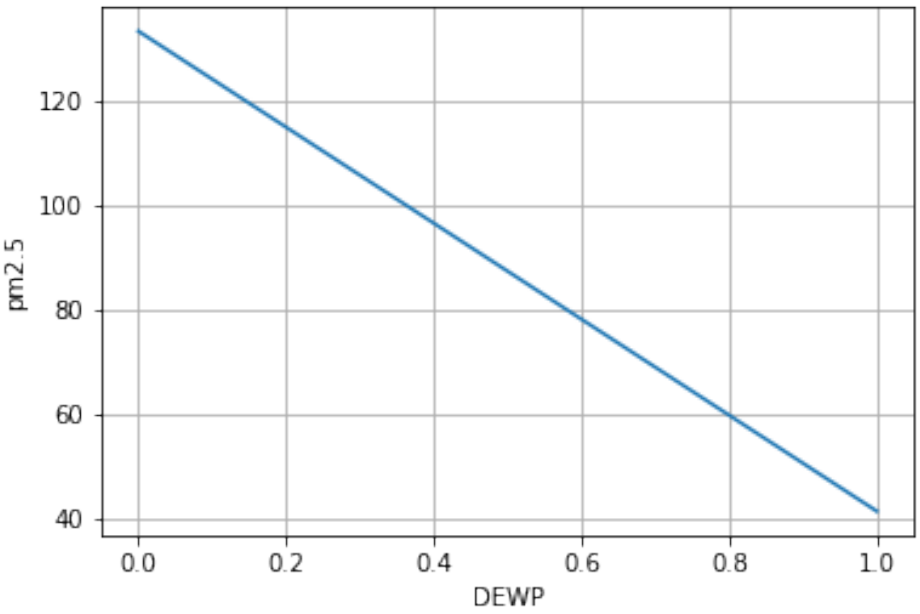


Diagram 8.2.10 shows how temperature affects PM2.5 averagely when the other meteorology factors are not changing.

Diagram 8.2.10 PM2.5 Affected by Temperature

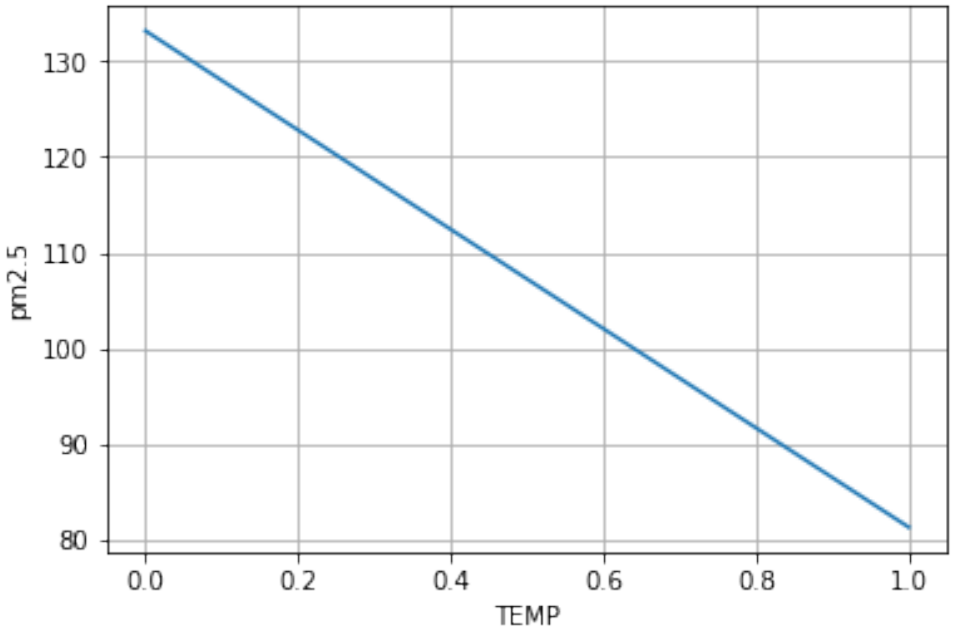
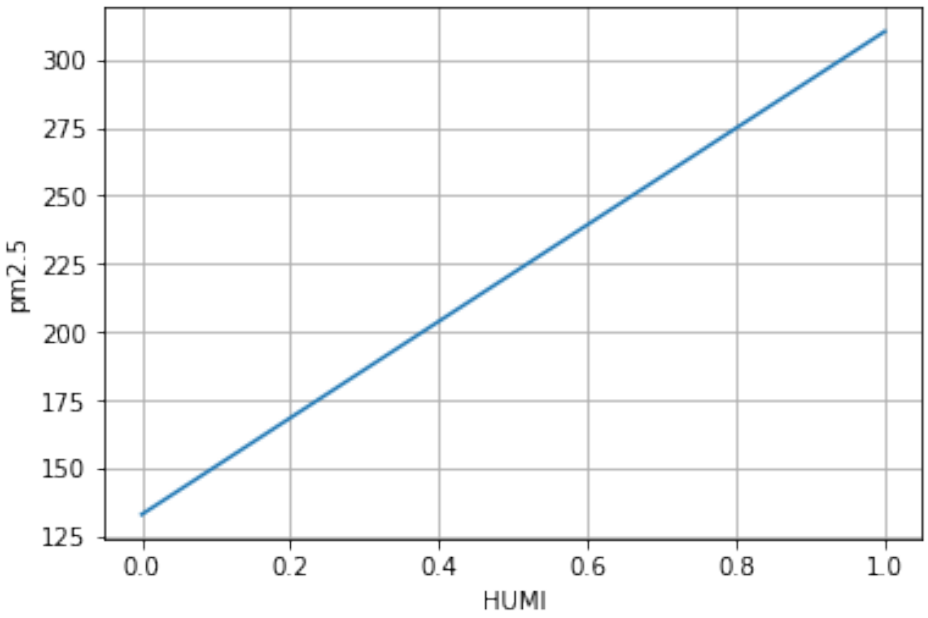


Diagram 8.2.11 shows how humidity affects PM2.5 averagely when the other meteorology factors are not changing.

Diagram 8.2.11 PM2.5 Affected by Humidity



8.3 Interpreting Result, Models, and Patterns

In the previous chapters, I have studied, discussed and visualized the results, models and patterns. In this chapter, I am going to interpret them from both technical and situational perspectives.

First, the models have found out the meteorology elements that affect PM2.5 concentration in Beijing. They are humidity, combined wind direction, cumulated wind speed, pressure, dew point, and temperature. Other meteorology factors, including hourly precipitation, cumulated precipitation, cumulated hours of snow, and cumulated hours of rain, are considered not affecting PM2.5 concentration in Beijing.

Second, the models have found out the main factor among these meteorology factors that influence PM2.5 concentration. Humidity is recognized as the most important factor which influences PM2.5 concentration more than the other factors in both models. Cumulated wind speed, pressure, temperature, combined wind direction and dew point are not as important as humidity, but they still play roles in PM2.5 concentration.

Third, the models have found out how these meteorology factors affect PM2.5 concentration. In the visualization chapter, *Diagram 8.2.11* showed that humidity, as the main factor, affects PM2.3 concentration positively. *Diagram 8.3.1*, as part of the decision tree regressor model, illustrates the same result. Besides, the other factors, including cumulated wind speed, pressure, dew point, and temperature, influence PM2.5 concentration negatively, as shown in *Diagrams 8.2.7, 8.2.8, 8.2.9,* and *8.2.10* in the visualization chapter. In other words, if other meteorology factors are stable, and one of them goes higher, PM2.5 concentration tends to go lower.

Diagram 8.3.1 How Humidity affects PM2.5 Concentration in Decision Tree Regressor

```
If (HUMI <= 0.3163265306122449)
 Predict: 32.22249093107618
Else (HUMI > 0.3163265306122449)
 Predict: 49.196969696969695
```

```
If (HUMI <= 0.5918367346938775)
 Predict: 61.142857142857146
Else (HUMI > 0.5918367346938775)
 Predict: 102.5880503144654
```

Fourth, both models are able to predict PM2.5 concentration based on the meteorology and can give acceptable results, as shown in *Diagrams 7.2.5, 7.2.6, 7.2.7* and *7.2.8* in the conducting data mining chapter. Besides, by comparing the results of these two models, it showed that the decision tree regressor model has slighter errors and higher R2 score. Therefore, it is better to use decision tree regressor models to make the prediction and compare the results.

Last but not least, as humidity is the main meteorology factor that affects PM2.5 concentration, a solution came up to protect people from high PM2.5 concentration in Beijing based on it, thereby reducing premature mortality rated related to air pollution. It is to build a green belt of trees around the border of the city. The tree belt can absorb moisture in the air (Liu & Sheng, 2017). Since the tree belt is in the border of the city, part of the moisture in the main area of the city will be absorbed by it, which can cause the humidity in the city lower. Also, Liu and Sheng (2017) described that Plants, especially trees, can effectively adsorb, filter and obstruct the air pollution. Therefore, the green belt can effectively control the PM2.5 matter.

8.4 Accessing and Evaluating Results, Models and Patterns

As discussed in the interpretation chapter, it is recognized that humidity, combined wind direction, cumulated wind speed, pressure, dew point, and temperature can affect PM2.5 concentration in Beijing. This meets one of the situation objectives and one of the success criteria – finding the meteorology elements that influence PM2.5 concentration in Beijing.

Besides, the models are able to predict PM2.5 concentration in Beijing by the meteorology information after studying the historical data, as mentioned in the interpretation chapter and the conducting data mining chapter. Therefore, the study has met the success criterion and the data mining objective of being able to predict PM2.5 concentration.

Also, the visualization chapter and the interpretation chapter has affirmed that humidity is the main meteorology reason for the increase or decrease of PM2.5 concentration in Beijing. This meets the data mining objective that analyzing the main influential meteorology element that affects PM2.5 concentration in Beijing.

Meanwhile, the ways the meteorology factors affect PM2.5 concentration are also clarified. Humidity affects PM2.5 concentration positively, while the other factors with numeric value affect it negatively. As for the combined wind direction, the southeast wind increases PM2.5 concentration, while the northeast wind reduces it, which was discussed in the visualization and interpretation chapter. Therefore, the last data mining objective, which required this study to find out how meteorology elements affect PM2.5 concentration in Beijing, is achieved.

In addition, I managed to do this project as I planned in the project plan chapter, so it meets the last success criterion that is to finish the project in two weeks.

Lastly, building a green belt of trees around the border of the city has come up as a solution to control PM2.5 concentration in Beijing. Therefore, the last situation objective, which aims to design an effective solution to reduce PM2.5 concentration, thereby reducing premature mortality rate related to air pollution, is met.

In conclusion, this study is estimated as a success as it achieves all the situation objectives and the data mining objectives, which I set in the situation understanding chapter. Also, this study has met the success criteria I set in that chapter.

8.5 Reiterating

8.5.1 Situation Understanding

Although the reiteration of this step did not change anything in this chapter, it gave me a deeper understanding of the importance of this study. It is a project that can help save people's life from atmospheric pollution.

8.5.2 Data Understanding

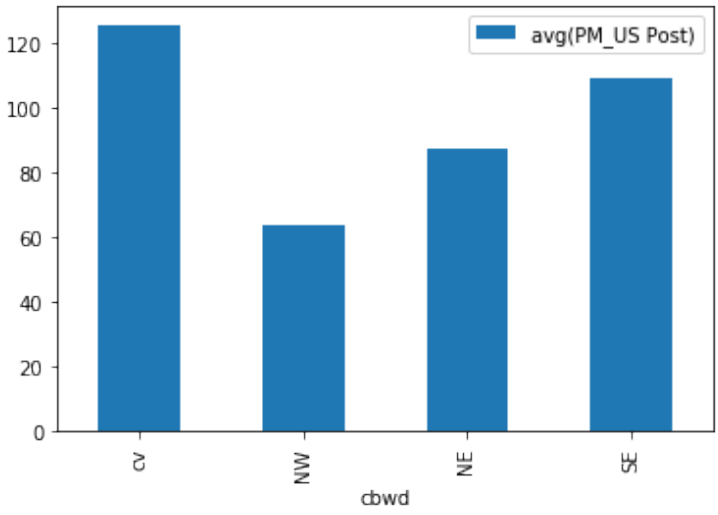
In this step, I explored the combined wind direction with PM2.5 concentration which I did not do previously.

Diagram 8.5.1 shows the code I used to explore the data, while *Diagram 8.5.2* illustrates that when the wind is calm or varies, the average PM2.5 concentration is the highest, followed by the southeast wind. The northwest wind seems to bring the lowest mean PM2.5 concentration.

Diagram 8.5.1 Code to Explore Combined Wind Direction

```
#Explore more in combined wind direction.
dataset2 = spark.read.csv('./Datasets/BeijingPM20100101_20151231.csv',
                          inferSchema = True, header = True)
dataset_cbwd = dataset2.select("PM_US Post","cbwd")
dataset_cbwd = du.set_NA_to_null(dataset_cbwd)
dataset_cbwd = dataset_cbwd.withColumn("PM_US Post",dataset_cbwd["PM_US Post"].cast(IntegerT
ype()))
dataset_cbwd = dataset_cbwd.groupBy('cbwd').mean().na.drop()
pm_mean_df = dataset_cbwd.toPandas()
pm_mean_df = pm_mean_df.set_index('cbwd')
pm_mean_df.plot(kind='bar')
plt.show()
```

Diagram 8.5.2 Combined Wind Direction versus PM2.5 Concentration



8.5.3 Data Preparation

In the studying and discussing the mined patterns chapter, it is discussed that the data mining results in the first round indicated that the outliers in PM2.5 concentration might have minimal effect on the results. Therefore, I deleted the outliers in this step, as shown in *Diagram 8.5.3*. After that, there are still 40,953 records in the dataset, which are enough for the data mining process.

Diagram 8.5.3 Delete the Outliers in PM2.5 Concentration

```
#Reprepare the data by dropping the instances including pm2.5 outliers.
dataset_final = spark.read.csv('./Datasets/cleaned_data.csv',
                               inferSchema = True, header = True)
print("Size of dataset before deleting outliers: ",dataset_final.count())
dataset_final = dataset_final.filter("pm<374")
print("Size of dataset after deleting outliers: ",dataset_final.count())
```

```
Size of dataset before deleting outliers:  41732
Size of dataset after deleting outliers:  40953
```

8.5.4 Data Transformation

After recleaning the data in the last step, I used the function in *Diagram 4.1.3* to check the importance of all the attributes again. The unimportant ones remained unimportant, while the important ones are also the same as before, as shown in *Diagram 8.5.4*. Therefore, the result of data transformation is the same as previously.

Diagram 8.5.4 Feature Selection Result for Re-prepared Data

```
#Call the function to redo feature selection.
selected_data = dm.feature_selection(dataset_final)
dataset_final = spark.createDataFrame(selected_data)
dataset_final.show(5)
```

```
Unimportant features:  ['Is', 'Ir', 'precipitation', 'Iprec', 'rain', 'snow']
+---+---+----+----+------+----+----+----+
| No| pm|DEWP|TEMP|  PRES| Iws|HUMI|cbwd|
+---+---+----+----+------+----+----+----+
| 25|129| -16|-4.0|1020.0|1.79|38.0| 0.0|
| 26|148| -15|-4.0|1020.0|2.68|42.0| 0.0|
| 27|159| -11|-5.0|1021.0|3.57|63.5| 0.0|
| 28|181|  -7|-5.0|1022.0|5.36|85.0| 0.0|
| 29|138|  -7|-5.0|1022.0|6.25|85.0| 0.0|
+---+---+----+----+------+----+----+----+
only showing top 5 rows
```

8.5.5 Data-Mining Method Selection

Since the situation objectives and data mining objectives are not changed, I still chose regression method for this project.

8.5.6 Data-Mining Algorithms Selection

Similarly, since the situation objectives and data mining objectives are not changed, I still chose the linear regression algorithm and the decision tree regressor algorithm for this study. However, I changed some parameters in the models to see if the performance of the models can be improved.

Diagram 8.5.5 illustrated that I chose to force the regression line to run through the original point by set *"fitIntercept"* as False this time to see if it will get a better result. Also, *"standardization"* is set to False as I have already normalized the data in the data transformation chapter.

Diagram 8.5.5 Adjust Parameters in Linear Regression Model

```
#Re-build and re-choose the parameters for Linear Regression Model.
lr = LinearRegression(featuresCol='predictors', labelCol='pm',
                      fitIntercept=False, standardization=False)
```

In the decision tree regressor model, since the last result showed that none of the branches stopped until they reached the depth of five, I adjust the max depth of the tree to ten to see if it is going to have a better performance, as shown in *Diagram 8.5.6*.

Diagram 8.5.6 Adjust Parameters in Decision Tree Regressor Model

```
#Re-build and re-choose the parameters for Desicion Tree Regressor Model.
tree = DecisionTreeRegressor(featuresCol="predictors", labelCol='pm',
                             maxDepth=10, minInstancesPerNode=1)
```

8.5.7 Data Mining

Comparing to the results in *Diagrams 7.2.5* and *7.2.6*, the results of the linear regression algorithm this time has most errors lower and R2 scores higher, as shown in *Diagrams 8.5.7* and *8.5.8*. Although the mean errors became a little bit higher, I still believe that the performance of the linear regression model is better at this time.

Diagram 8.5.7 New Result of Training Data in Linear Regression Algorithm

```
#Call the function to re-analyze the training data of Linear Regression
dm.analyze_result_lr(lr_model.summary)
```

```
Root Mean Squared Error:         68.63025716254447
R2 Score:        0.6761771609094065
Mean Absolute Error:     51.72493563331853
Maximum Error:   309.5153981586847
Minimum Error:   -159.86123230651975
Mean Error:      1.1388532080037095
```

Diagram 8.5.8 New Result of Testing Data in Linear Regression Algorithm

```
#Call the function to re-analyze the testing data of Linear Regression
dm.analyze_result_lr(lr_test)
```

```
Root Mean Squared Error:         69.30841045467224
R2 Score:        0.6735547601971704
Mean Absolute Error:     52.53581854121398
Maximum Error:   300.86339834054485
Minimum Error:   -157.68321728378078
Mean Error:      1.14996672061622
```

Comparing to the results in *Diagrams 7.2.7* and *7.2.8*, the results of the decision tree algorithm this time has lower errors and higher R2 score, as shown in *Diagrams 8.5.9* and *8.5.10*. Therefore, the performance of the decision tree regressor model is better at this time.

Diagram 8.5.9 New Result of Training Data in Decision Tree Regressor Algorithm

```
#Call the function to re-analyze the training data of Desicion Tree
dm.analyze_result_tree(tree_train)
```

```
Root Mean Squared Error:         57.23613157000377
R2 Score:        0.4649084898825109
Mean Absolute Error:     40.64250437782706
Maximum Error:   275.20754716981133
Minimum Error:   -215.35
Mean Error:      2.6858515411731787e-12
```

Diagram 8.5.10 New Result of Testing Data in Decision Tree Regressor Algorithm

```
#Call the function to re-analyze the testing data of Desicion Tree
dm.analyze_result_tree(tree_test)
```

```
Root Mean Squared Error:         63.31627894307097
R2 Score:        0.3515002585450827
Mean Absolute Error:     45.101610318184825
Maximum Error:   307.0
Minimum Error:   -280.5
Mean Error:      0.5062917447294666
```

Besides, comparing to *Diagrams 7.3.4*, *7.3.5, 7.3.6 and 7.3.7*, *Diagrams 8.5.11, 8.5.12, 8.5.13,* and *8.5.14* illustrate that the distributions of the predictions in both models this time are closer to the actual records.

Diagram 8.5.11 New Residuals Plot of Linear Regression Model

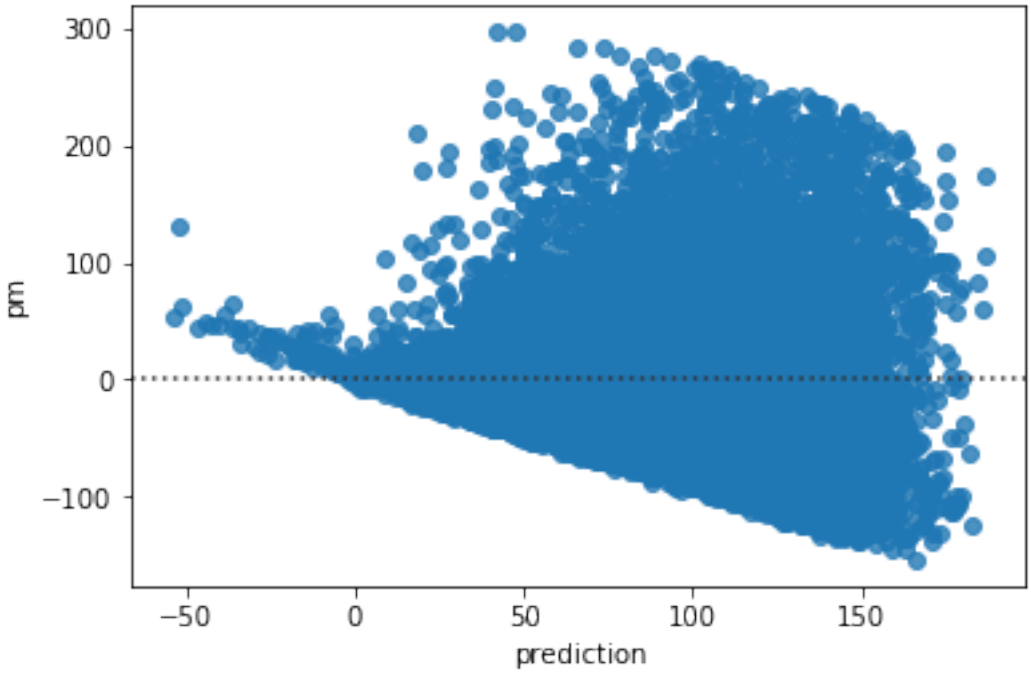


Diagram 8.5.12 New Comparison of Distribution in the Linear Regression Model

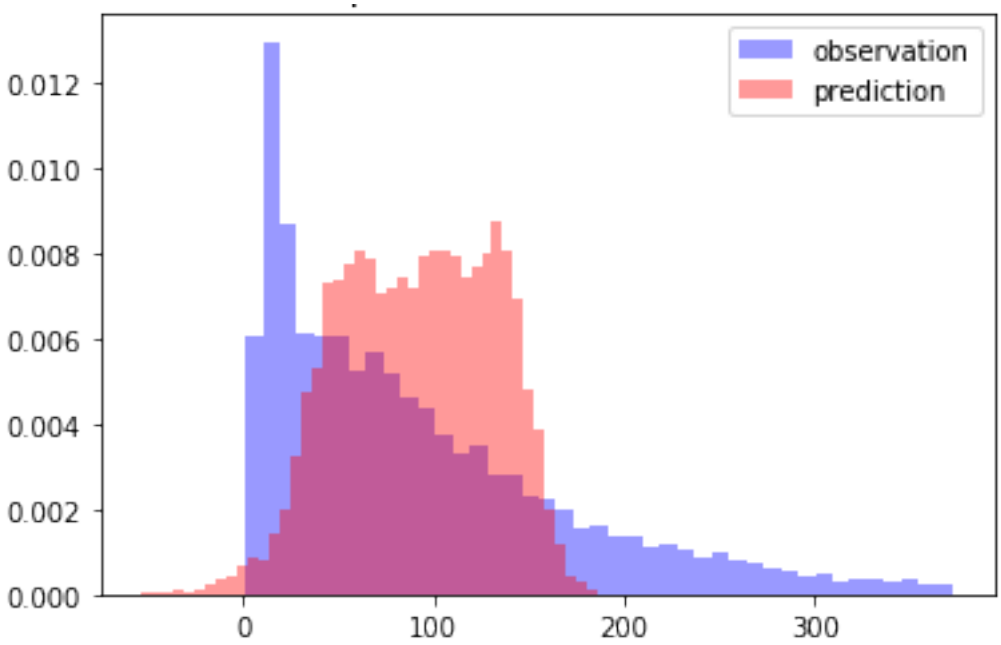


Diagram 8.5.13 New Residuals Plot of Decision Tree Regressor Model

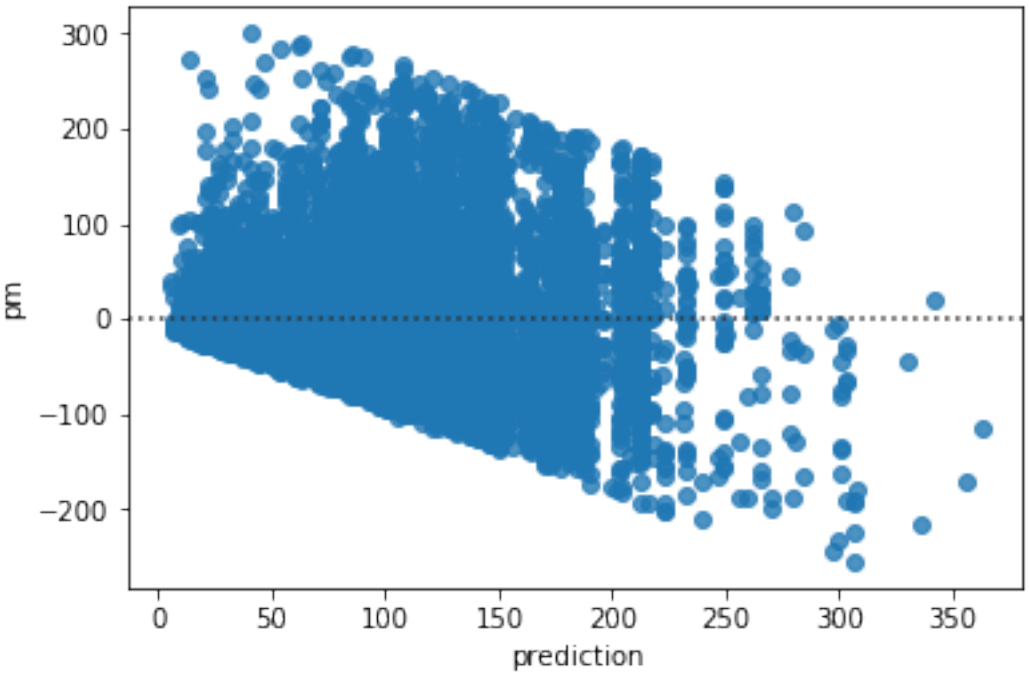
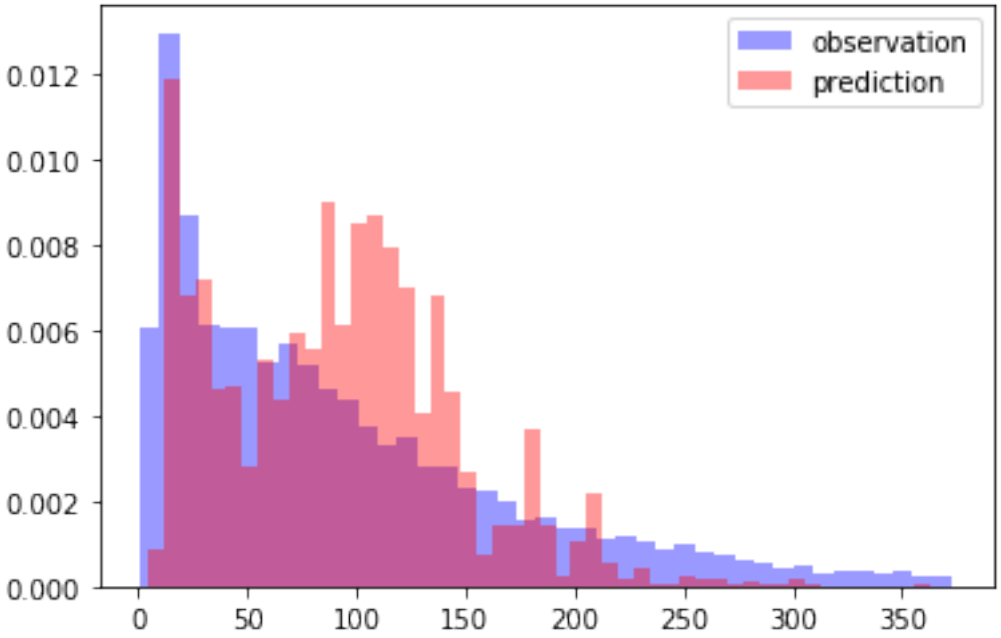


Diagram 8.5.14 New Comparison of Distribution in the Decision Tree Regressor Model



Besides, comparing to *Diagrams 7.3.9 and 7.3.10*, *Diagrams 8.5.15* and *8.5.16* illustrate that the values of predictions are closer to the values of actual records.

Diagram 8.5.15 New Predicted Target with Actual Records in Linear Regression Model

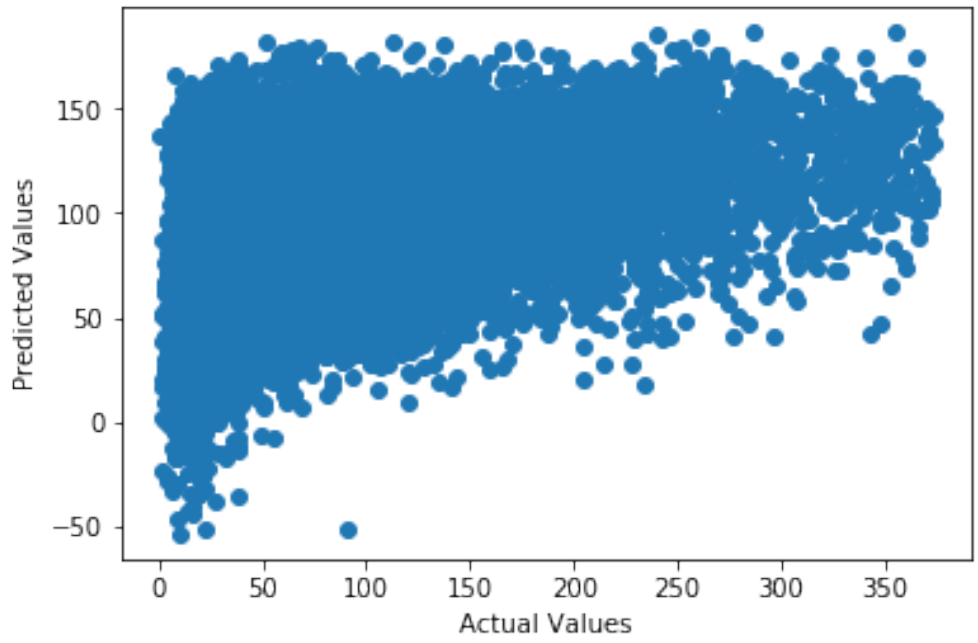
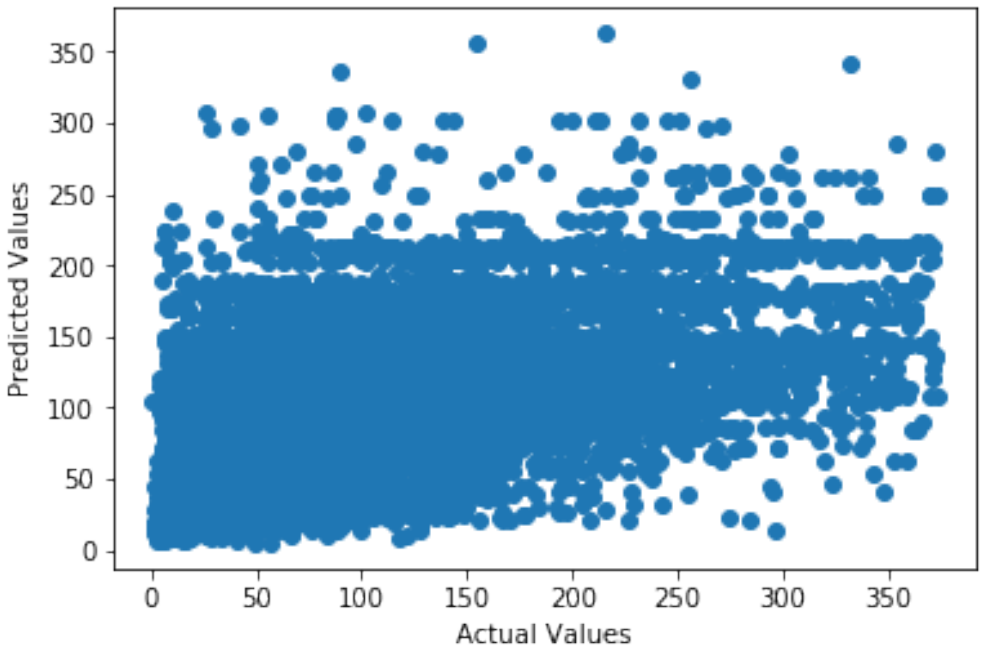


Diagram 8.5.16 New Predicted Target with Actual Records in Decision Tree Regressor Model

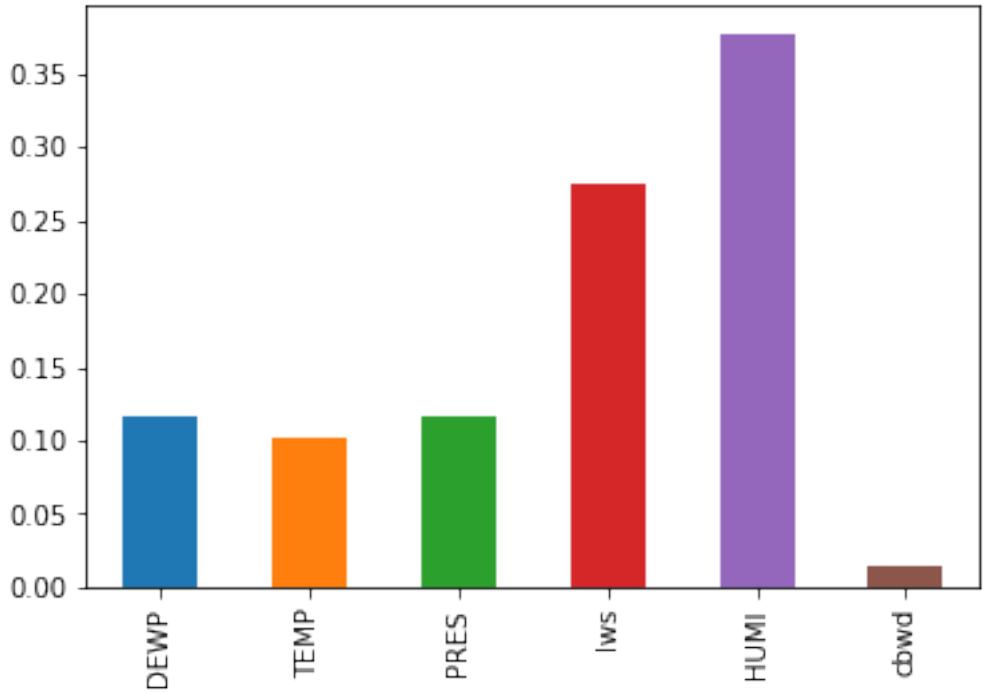


8.5.8 Interpretation

Overall, the results of the reiteration are similar to the results in the first round. Humidity is still the main meteorology element that influences PM2.5 concentration. The meteorology factors that were believed can affect PM2.5 concentration still remained influential. However, there are some interesting differences.

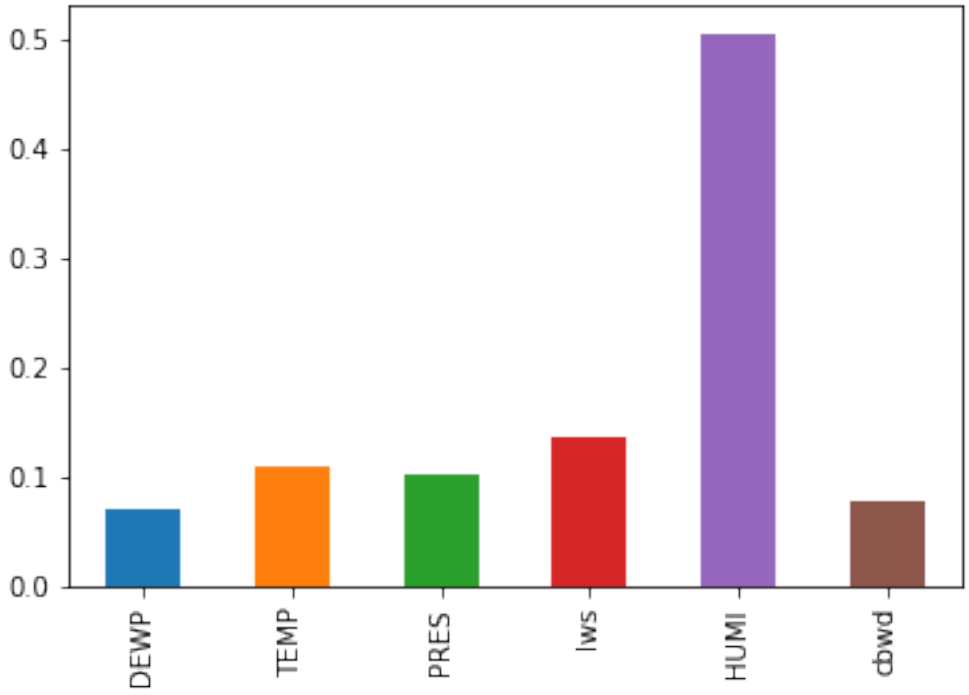
In the linear regression model, comparing to *Diagram 8.2.2*, *Diagram 8.5.17* shows that humidity has a greater influence than before.

Diagram 8.5.17 New Predictor Importance Analyzed by Linear Regression Model



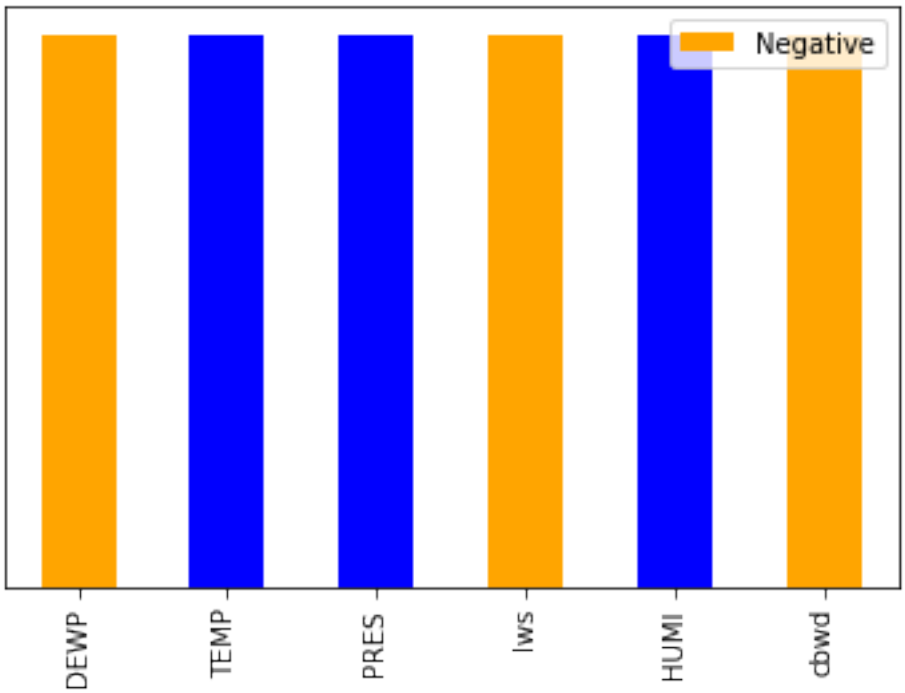
In the decision tree regressor model, comparing to *Diagram 8.2.3*, *Diagram 8.5.18* shows that although the importance of humidity is lower than before, it is still the most important feature. As for the other features, the importance of pressure becomes higher, while temperature became less important.

Diagram 8.5.18 New Predictor Importance Analyzed by Decision Tree Regressor Model



It is interesting that *Diagram 8.5.19* shows that temperature and pressure affect PM2.5 concentration positively, while in *Diagram 8.2.5*, they had negative influences.

Diagram 8.5.19 New Coefficients of Meteorology Factors



In conclusion, the linear regression model and the decision tree regressor model both can predict PM2.5 concentration in Beijing based on the meteorology data. From both models in different data mining processes, it is agreed that humidity is the main meteorology element that affects PM2.5 concentration in Beijing. When the humidity is higher, PM2.5 concentration tends to be higher, and vice versa. The other meteorology elements that affect PM2.5 concentration include cumulated wind speed, temperature, combined wind direction, pressure, and dew point. It is agreed that cumulated wind speed and dew point affect PM2.5 concentration negatively. With regard to combined wind direction, the northeast wind and the northeast wind can lower PM2.5 concentration, while the southeast wind makes PM2.5 concentration higher. According to the main influential meteorology factor, which is humidity, I suggest Beijing government to build a green belt of trees around the border of the city to help decrease PM2.5 concentration in the city, thereby protecting people from PM2.5 pollution and lower the premature rate related to air pollution in Beijing.

REFERENCES

Apache Spark. (n. d.) Classification and regression. Retrieved from https://spark.apache.org/docs/latest/ml-classification-regression.html#decision-trees

Bhatia, R. (2017). Top 6 Regression Algorithms Used in Data Mining and Their Application in Industry. Retrieved from https://analyticsindiamag.com/top-6-regression-algorithms-used-data-mining-applications-industry/

Bliss Air. (n.d.). What is PM2.5 and Why You Should Care. Retrieved from https://blissair.com/what-is-pm-2-5.htm

Brownlee, J. (2019). Supervised and Unsupervised Machine Learning Algorithms. Retrieved from https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

Chen, S. X. (2017). *FiveCitiePMData.rar* [rar file, PM2.5 Data of Five Chinese Cities]. Retrieved from https://archive.ics.uci.edu/ml/datasets/PM2.5+Data+of+Five+Chinese+Cities

Chen, S. X. (2017). *PRSA_data_2010.1.1-2014.12.31.csv* [csv file, Beijing PM2.5 Data]. Retrieved from https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data

Dave, A. (2020). Regression in Machine Learning. Retrieved from https://medium.com/datadriveninvestor/regression-in-machine-learning-296caae933ec

Ding, D., Xing, J., Wang, S., Liu, K., & Hao, J. (2019). Estimated Contributions of Emissions Controls, Meteorological Factors, Population Growth, and Changes in Baseline Mortality to Reductions in Ambient PM2.5 and PM2.5-Related Mortality in China, 2013-2017. *Environmental Health Perspectives, 127*(6). Retrieved from https://search-proquest-com.ezproxy.auckland.ac.nz/abicomplete/docview/2285241047/fulltextPDF/4B0EE766C4704715PQ/1?accountid=8424

Hao, F. (2018). China Releases 2020 Action Plan for Air Pollution. Retrieved from https://www.chinadialogue.net/article/show/single/en/10711-China-releases-2-2-action-plan-for-air-pollution

IBM. (2019). IBM SPSS Modeler 18.2.1 Modeling Nodes. Retrieved from ftp://public.dhe.ibm.com/software/analytics/spss/documentation/modeler/18.2.1/en/ModelerModelingNodes.pdf

Information Gain Ltd. (2012). Why Split Data in the Ratio 70:30. Retrieved from http://information-gain.blogspot.com/2012/07/why-split-data-in-ratio-7030.html

Lakshmanan, S. (2019). How, When and Why Should You Normalise / Standardize / Rescale Your Data. Retrieved from https://medium.com/@swethalakshmanan14/how-when-and-why-should-you-normalise-standardize-rescale-your-data-3f083def38ff

Liang, X., Li, S., Zhang, S., Huang, H., & Chen, S. X. (2016). PM2.5 data reliability, consistency, and air quality assessment in five chinese cities. *Journal of Geophysical Research.Atmospheres, 121*(17), 10,220-10,236. doi: 10.1002/2016JD024877

Liang, X., Zou, T., Guo, B., Li, S., Zhang, H., Zhang, S., ...Chen, S. X. (2015). Assessing Beijing's PM2.5 Pollution: Severity, Weather Impact, APEC and Winter Heating. *Proceedings of the Royal Society A: Mathematical, Physical & Engineering Sciences, 471*(2182), 1-20. doi: 10.1098/rspa.2015.0257

Liu, W., & Sheng, J. (2017). *Morden Environmentology*. Beijing, China: Beijing Book Co. Inc.

Molnar, C. (2020). *Interpretable Machine Learning: A Guild for Making Black Box Models Explainable.* Retrieved from https://christophm.github.io/interpretable-ml-book/

Pantazi, X. E., Moshou, D., & Bochtis, D. (2020). Intelligent Data Mining and Fushion Systems in Agriculture. Retrieved from https://www.sciencedirect.com/book/9780128143919/intelligent-data-mining-and-fusion-systems-in-agriculture#book-info

Simplilearn. (n.d.). Classification – Machine Learning. Retrieved from https://www.simplilearn.com/classification-machine-learning-tutorial

Srivastava, H. (2018). What is a Regression Tree. Retrieved from https://magoosh.com/data-science/what-is-a-regression-tree/

Statistics Solutions (n.d.) What is Logistic Regression. Retrieved from https://www.statisticssolutions.com/what-is-logistic-regression/

The University of Auckland. (n.d.) The University of Auckland Human Participants Ethics Committee (UAHPEC). Retrieved from https://www.auckland.ac.nz/en/research/about-our-research/human-ethics/human-participants-ethics-committee-uahpec.html

UCI Machine Learning Repository. (n.d.). About. Retrieved from https://archive.ics.uci.edu/ml/about.html

United Nations. (2016). Sustainable Development Goal 3: Ensure healthy lives and promote well-being for all at all ages. Retrieved from https://sustainabledevelopment.un.org/sdg3#targets

Vega, R. D. V., & Rai, A. G. (n. d.) Multivariate Regression. Retrieved from https://brilliant.org/wiki/multivariate-regression/

Wang, Y., Zhang, H., Zhai, J., Wu, Y., Cong, L., Yan, G., & Zhang, Z. (2020). Seasonal Variations and Chemical Characteristics of PM2.5 Aerosol in the Urban Green Belt of Beijing, China. *Polish Journal of Environmental Studies, 29*(1), 361-370. doi: 10.15244/pjoes/104358

"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."