Data Science Project1 PM2.5 Kaggle (Redo) Report
One night miracle

Present to

Peerapon Vateekul, Ph.D.
Natawut Nupairoj, Ph.D.
Veera Muangsin, Ph.D.

By

6230556021	Sirawit	Makoo
6230563321	Suchakree	Maneepakorn

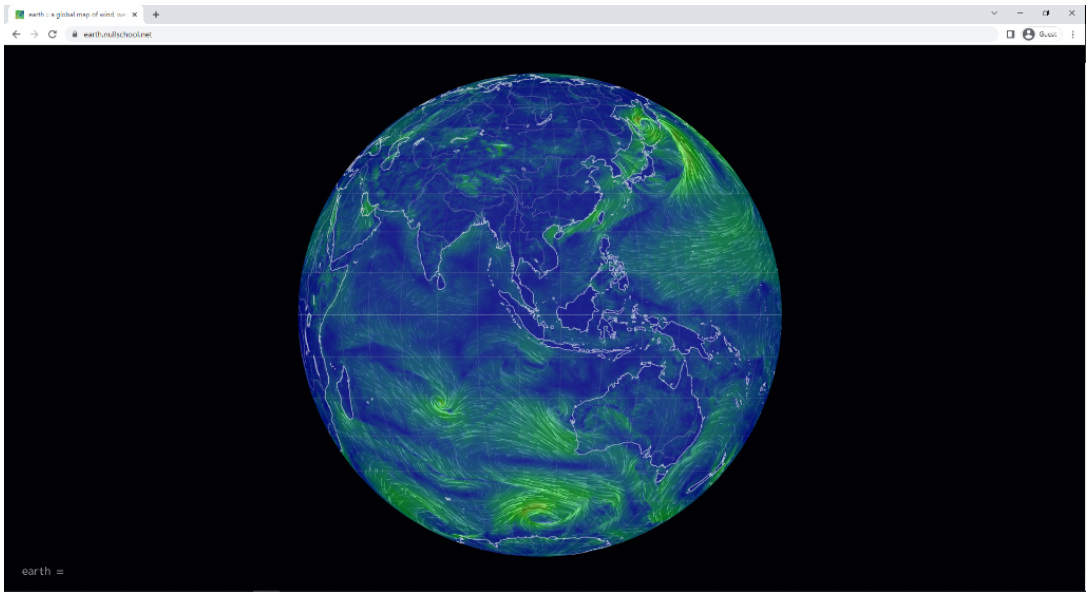
2110446, semester 2/2021

Department of Computer Engineering,
Faculty of Engineering, Chulalongkorn University

Table of Contents

Data Scraping

To scrape the test data of wind and temperature from https://earth.nullschool.net/, we used Selenium for such purposes.




```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.edge.service import Service
from webdriver_manager.microsoft import EdgeChromiumDriverManager
```

```
service = Service(executable_path=EdgeChromiumDriverManager().install())
driver = webdriver.Edge(service=service)
```

```
[WDM] - ====== WebDriver manager ======
[WDM] - Current edge version is 99.0.1150
[WDM] - Get LATEST edgedriver version for 99.0.1150 Edge
[WDM] - Trying to download new driver from https://msedgedriver.azureedge.net/99.0.1150.55/edgedriver_win64.zip
[WDM] - Driver has been saved in cache [C:\Users\FACT-PC\.wdm\drivers\edgedriver\win64\99.0.1150.55]
```

First, we read the position of Bangkok, Chiangmai, Khonkaen, Rayong, Saraburi, and Surat from the given files.

```
f_bkk = open('./datasci_dataset_2022/BKK/position.txt', 'r')
f_cm = open('./datasci_dataset_2022/Chiangmai/position.txt', 'r')
f_kk = open('./datasci_dataset_2022/Khonkaen/position.txt', 'r')
f_ry = open('./datasci_dataset_2022/Rayong/position.txt', 'r')
f_sb = open('./datasci_dataset_2022/Saraburi/position.txt', 'r')
f_sr = open('./datasci_dataset_2022/Surat/position.txt', 'r')

```

```
BKK: {'lat': '13.729984', 'long': '100.536443'}
Chiangmai: {'lat': '18.840633', 'long': '98.969661'}
Khonkaen: {'lat': '16.445329', 'long': '102.835251'}
Rayong: {'lat': '12.671521', 'long': '101.275875'}
Saraburi: {'lat': '14.685833', 'long': '100.871996'}
Surat: {'lat': '9.126057', 'long': '99.325355'}
```

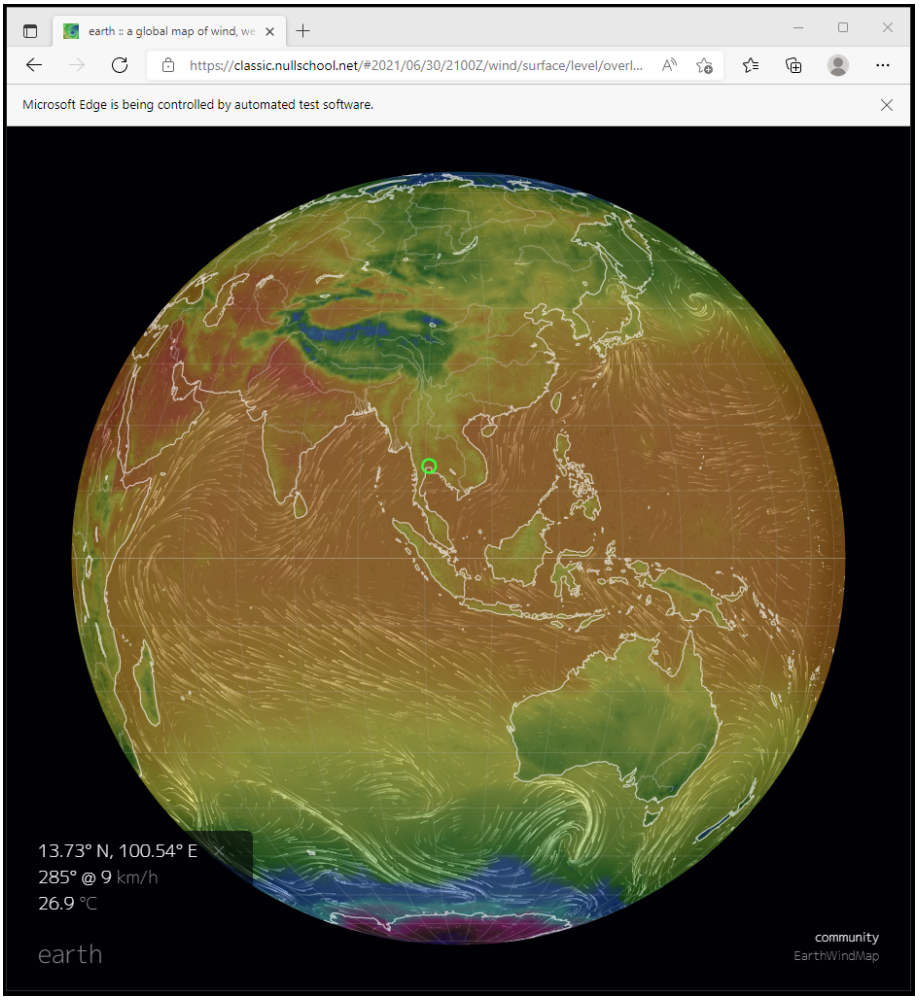
Second, we defined functions (scrape_wind, scrape_wind_data, scrape_temp, scrape_temp_data) used to scrape the data from the website, format the scraped data into the appropriate form, and save the data to .csv files.

```python
def scrape_temp(date, lat, long):
    url =  f'https://classic.nullschool.net/{date}/wind/surface/level/overlay=temp/orthographic/loc={long},{lat}'
    driver.get(url)
    driver.refresh()
    temp_info = driver.find_element(by=By.CSS_SELECTOR, value='#location-value').get_attribute('innerText')
    return temp_info

def scrape_temp_data(start_date, end_date, position, f_dir):
    path = f'./datasci_dataset_2022/{f_dir.capitalize()}/test/{f_dir}_temp_test.csv'

    if (not os.path.exists(path) or os.stat(path).st_size==0):
        f_w = open(path, 'a')
        f_w.write('date_time,temp\n')
        f_w.close()
    else:
        f_r = open(path, 'r')
        last_line = f_r.readlines()[-1][:-1]
        # print(f"lastLine: {last_line}")
        if (last_line == 'date_time,temp'):
            pass
        else:
            last_date = pd.to_datetime([last_line[:16] + " +0700"], utc=True)
            start_date = last_date[0] + pd.Timedelta(hours=3)

    counter = 0
    f_w = open(path, 'a')
    while start_date <= end_date :
        scrape_date = start_date.strftime('#%Y/%m/%d/%H%MZ')
        save_date = start_date.tz_convert('Asia/Bangkok').strftime('%Y-%m-%d %H:%M:%S')
        while True:
            temp_info = scrape_temp(scrape_date, position['lat'], position['long'])
            if temp_info: # if we have got the data, break the loop
                break
        start_date += pd.Timedelta(hours=3)
        f_w.write(f"{save_date},{temp_info}\n")
        if counter == 7:   # Save the data once a day
            counter = 0
            f_w.close()
            f_w = open(path, 'a')
        else:
            counter += 1
        print(f"{save_date},{temp_info}")
    f_w.close()
```

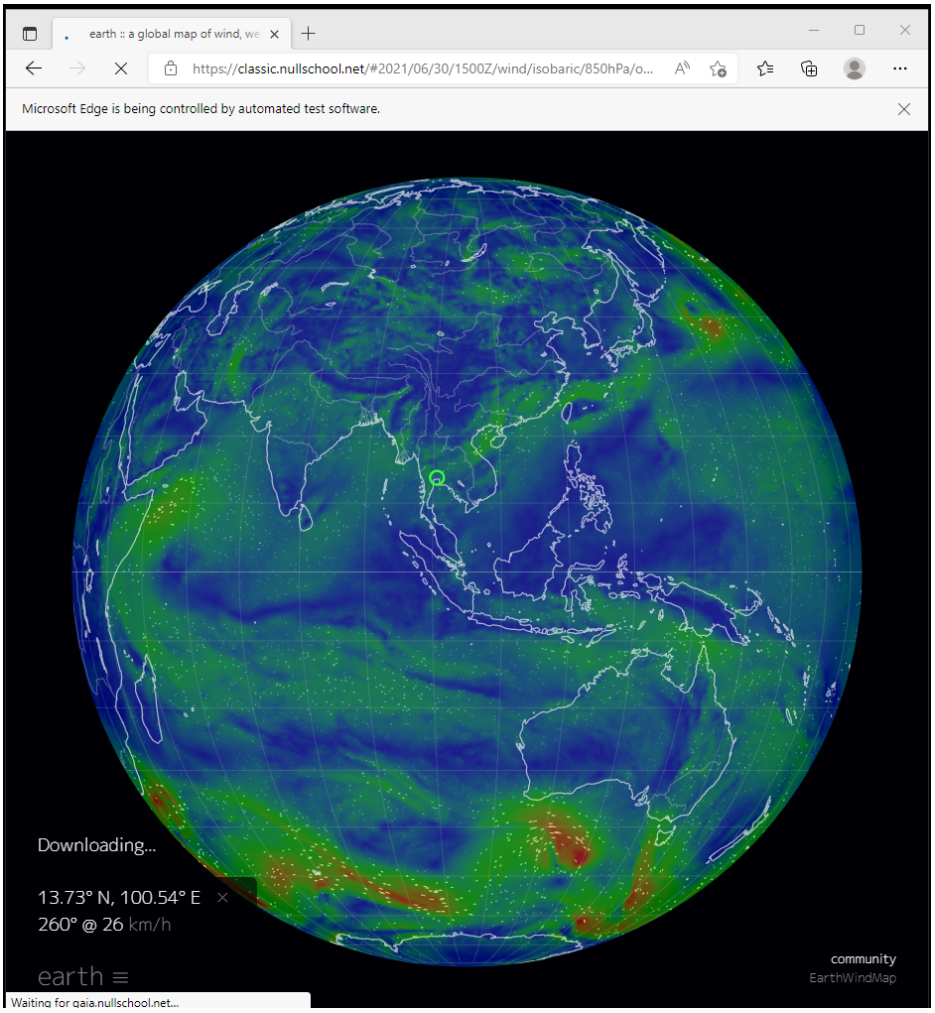
```python
def scrape_wind(date, lat, long):
    url =  f'https://classic.nullschool.net/{date}/wind/isobaric/850hPa/orthographic/loc={long},{lat}'
    driver.get(url)
    driver.refresh()
    wind_el = driver.find_element(by=By.CSS_SELECTOR, value='#location-wind')
    wind_info = wind_el.get_attribute('innerText').split('° @ ')
    return wind_info

def scrape_wind_data(start_date, end_date, position, f_dir):
    path = f'./datasci_dataset_2022/{f_dir.capitalize()}/test/{f_dir}_wind_test.csv'

    if (not os.path.exists(path) or os.stat(path).st_size==0):
        f_w = open(path, 'a')
        f_w.write('date_time,wind speed,wind dir\n')
        f_w.close()
    else:
        f_r = open(path, 'r')
        last_line = f_r.readlines()[-1][:-1]
        # print(f"Lastline: {last_line}")
        if (last_line == 'date_time,wind speed,wind dir'):
            pass
        else:
            last_date = pd.to_datetime([last_line[:16] + " +0700"], utc=True)
            start_date = last_date[0] + pd.Timedelta(hours=3)

    counter = 0
    f_w = open(path, 'a')
    while start_date <= end_date :
        scrape_date = start_date.strftime('#%Y/%m/%d/%H%MZ')
        save_date = start_date.tz_convert('Asia/Bangkok').strftime('%Y-%m-%d %H:%M:%S')
        while True:
            wind_info = scrape_wind(scrape_date, position['lat'], position['long'])
            if len(wind_info) == 2: # if we have got the data, break the loop
                break
        start_date += pd.Timedelta(hours=3)
        f_w.write(f"{save_date},{wind_info[1]},{wind_info[0]}\n")
        if counter == 7:   # Save the data once a day
            counter = 0
            f_w.close()
            f_w = open(path, 'a')
        else:
            counter += 1
        print(f"{save_date},{wind_info[1]},{wind_info[0]}")
    f_w.close()
```

Then, we called the defined function in order to scrape the data which will be used in the following steps.

```
# Scrape temp data for BKK
scrape_temp_data(start_date, end_date, position_bkk, 'bkk')
# Scrape temp data for Chiangmai
scrape_temp_data(start_date, end_date, position_cm, 'chiangmai')
# Scrape temp data for Khonkaen
scrape_temp_data(start_date, end_date, position_kk, 'khonkaen')
# Scrape temp data for Rayong
scrape_temp_data(start_date, end_date, position_ry, 'rayong')
# Scrape temp data for Saraburi
scrape_temp_data(start_date, end_date, position_sb, 'saraburi')
# Scrape temp data for Surat
scrape_temp_data(start_date, end_date, position_sr, 'surat')
```



```python
# Scrape wind data for BKK
scrape_wind_data(start_date, end_date, position_bkk, 'bkk')
# Scrape wind data for Chiangmai
scrape_wind_data(start_date, end_date, position_cm, 'chiangmai')
# Scrape wind data for Khonkaen
scrape_wind_data(start_date, end_date, position_kk, 'khonkaen')
# Scrape wind data for Rayong
scrape_wind_data(start_date, end_date, position_ry, 'rayong')
# Scrape wind data for Surat
scrape_wind_data(start_date, end_date, position_sr, 'surat')
```



Data Preparation & Analysis

Data Preparation

For data preparation, we wrote 2 functions: toDF(), and toDFtest() in custom_function/csv_df.py in order to handle the duplicate rows, set the indexes, interpolate and do forward/backward filling to eliminate missing values. Then, the functions merge the 4 columns into one dataframe and return as train_set and test_set for each province.

```
bkk_train = toDF('BKK/train/bkk_train.csv', 'BKK/train/bkk_temp_surface.csv', 'BKK/train/bkk_weather_wind.csv')
bkk_test = toDFtest('BKK/test/bkk_test.csv', 'BKK/test/bkk_temp_test.csv', 'BKK/test/bkk_wind_test.csv')

cnx_train = toDF('Chiangmai/train/chiangmai_train.csv', 'Chiangmai/train/chiangmai_temp_surface.csv', 'Chiangmai/train/chiangmai_weather_wind.csv')
cnx_test = toDFtest('Chiangmai/test/chiangmai_test.csv', 'Chiangmai/test/chiangmai_temp_test.csv', 'Chiangmai/test/chiangmai_wind_test.csv')

kkc_train = toDF('Khonkaen/train/khonkhan_train.csv', 'Khonkaen/train/khonkaen_temp_surface.csv', 'Khonkaen/train/khonkaen_weather_wind.csv')
kkc_test = toDFtest('Khonkaen/test/khonkhan_test.csv', 'Khonkaen/test/khonkaen_temp_test.csv', 'Khonkaen/test/khonkaen_wind_test.csv')

rayong_train = toDF('Rayong/train/rayong_train.csv', 'Rayong/train/rayong_temp_surface.csv', 'Rayong/train/rayong_weather_wind.csv')
rayong_test = toDFtest('Rayong/test/rayong_test.csv', 'Rayong/test/rayong_temp_test.csv', 'Rayong/test/rayong_wind_test.csv')

saraburi_train = toDF('Saraburi/train/saraburi_train.csv', 'Saraburi/train/saraburi_temp_surface.csv', 'Saraburi/train/saraburi_weather_wind.csv')
saraburi_test = toDFtest('Saraburi/test/saraburi_test.csv', 'Saraburi/test/saraburi_temp_test.csv', 'Saraburi/test/saraburi_wind_test.csv')

surat_train = toDF('Surat/train/surat_train.csv', 'Surat/train/surat_temp_surface.csv', 'Surat/train/surat_weather_wind.csv')
surat_test = toDFtest('Surat/test/surat_test.csv', 'Surat/test/surat_temp_test.csv', 'Surat/test/surat_wind_test.csv')
```

Second, we checked null values of each province dataframe. We have all rows fully filled with ffill() and bfill() functions of pandas

Number of null values

Before Dropping

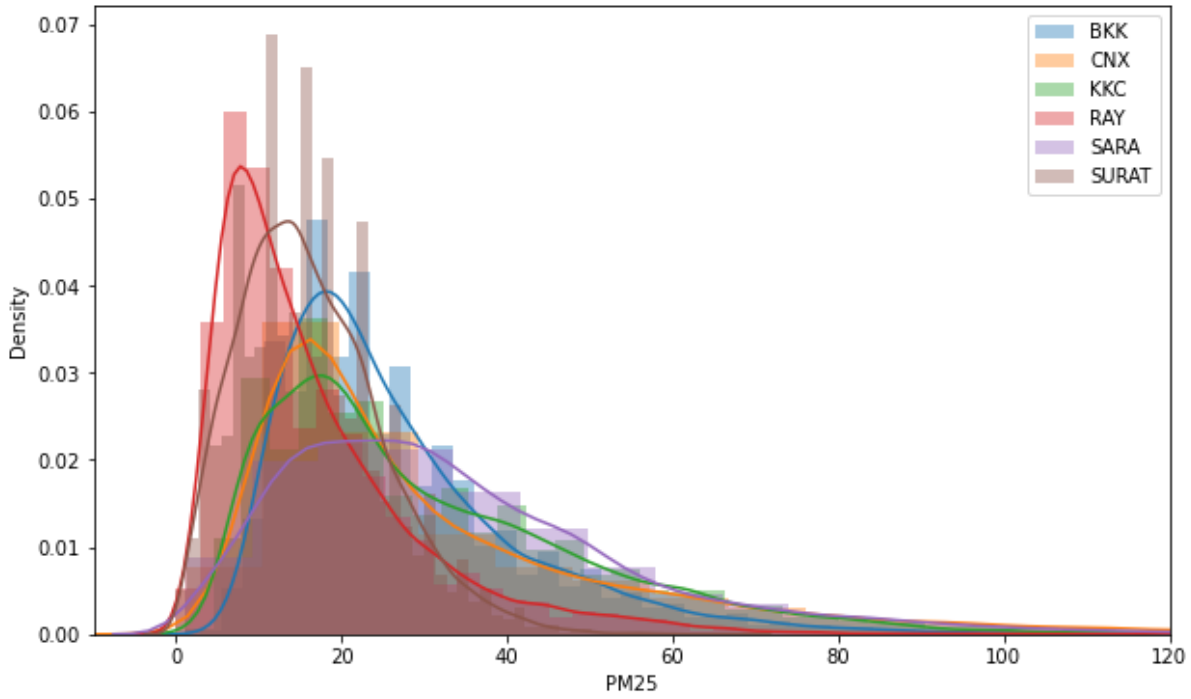
```
null_counts_bkk_train = bkk_train.isnull().sum()
print("[bkk_train] Number of null values in each column:\n{}".format(null_counts_bkk_train))
print('\n')

null_counts_cnx_train = cnx_train.isnull().sum()
print("[cnx_train] Number of null values in each column:\n{}".format(null_counts_cnx_train))
print('\n')

null_counts_kkc_train = kkc_train.isnull().sum()
print("[kkc_train] Number of null values in each column:\n{}".format(null_counts_kkc_train))
print('\n')

null_counts_rayong_train = rayong_train.isnull().sum()
print("[rayong_train] Number of null values in each column:\n{}".format(null_counts_rayong_train))
print('\n')

null_counts_saraburi_train = saraburi_train.isnull().sum()
print("[saraburi_train] Number of null values in each column:\n{}".format(null_counts_saraburi_train))
print('\n')

null_counts_surat_train = surat_train.isnull().sum()
print("[surat_train] Number of null values in each column:\n{}".format(null_counts_surat_train))
```

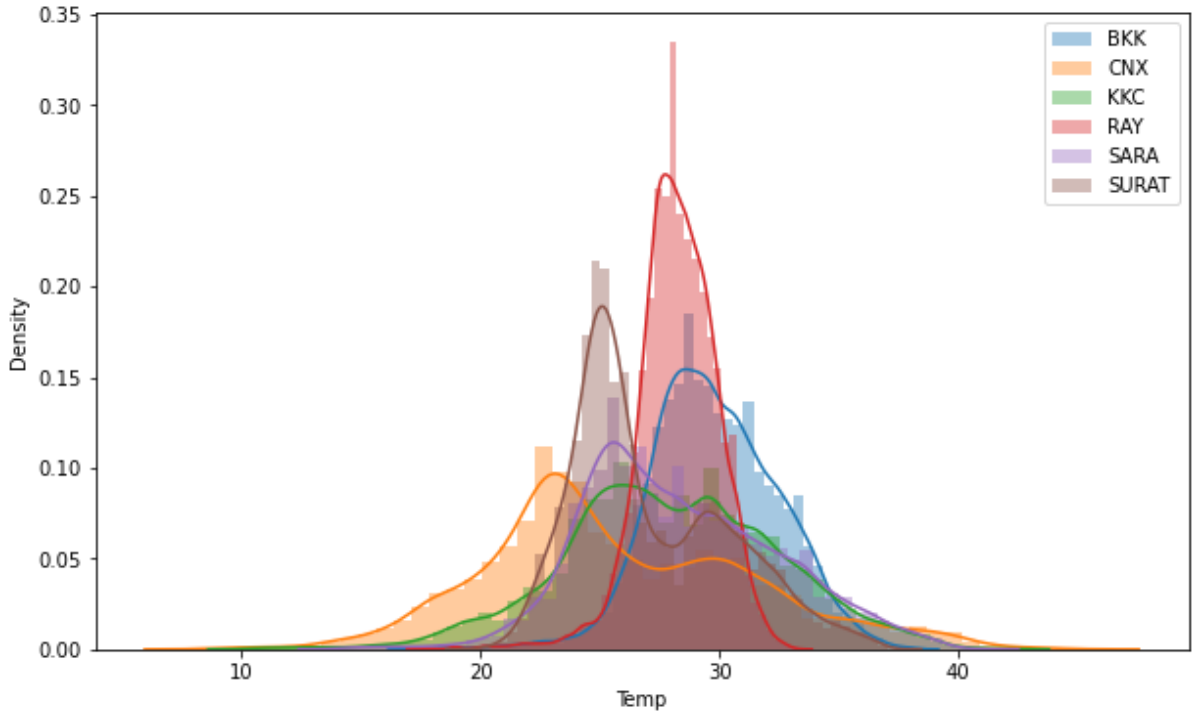
```
[bkk_train] Number of null values in each column:
Temp         0
WindSpeed    0
WindDir      0
PM25         0
dtype: int64


[cnx_train] Number of null values in each column:
Temp         0
WindSpeed    0
WindDir      0
PM25         0
dtype: int64


[kkc_train] Number of null values in each column:
Temp         0
WindSpeed    0
WindDir      0
PM25         0
dtype: int64
```

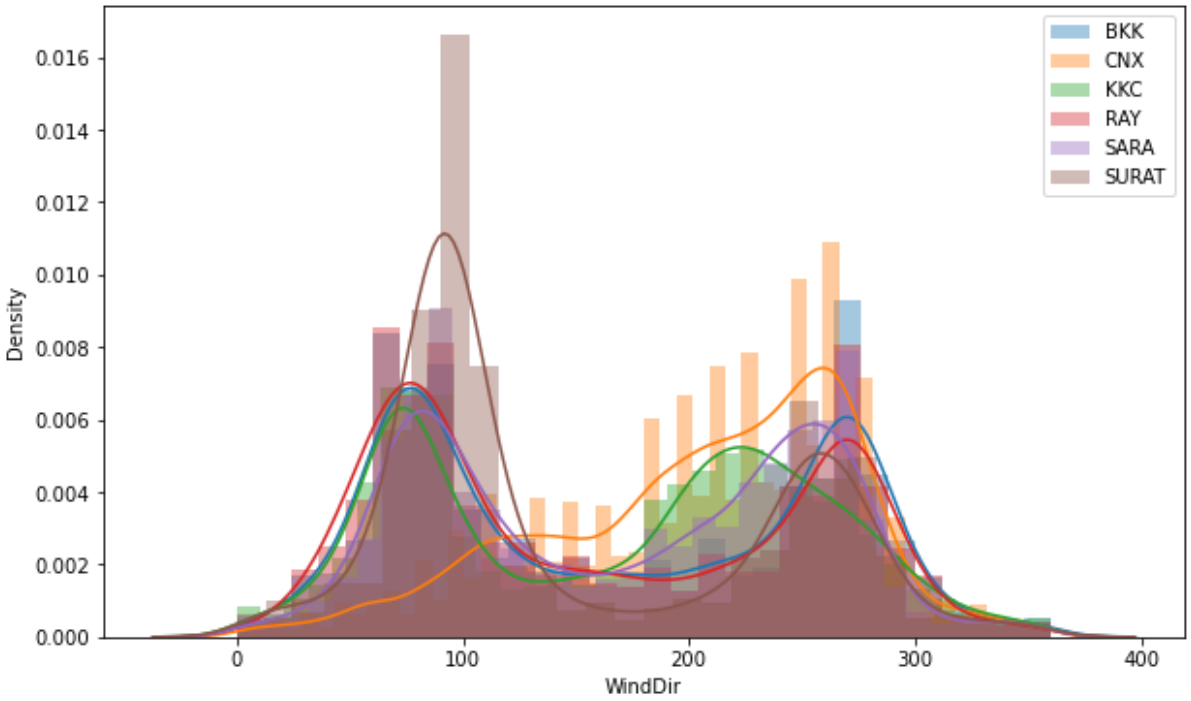
```
[rayong_train] Number of null values in each column:
Temp         0
WindSpeed    0
WindDir      0
PM25         0
dtype: int64


[saraburi_train] Number of null values in each column:
Temp         0
WindSpeed    0
WindDir      0
PM25         0
dtype: int64


[surat_train] Number of null values in each column:
Temp         0
WindSpeed    0
WindDir      0
PM25         0
dtype: int64
```

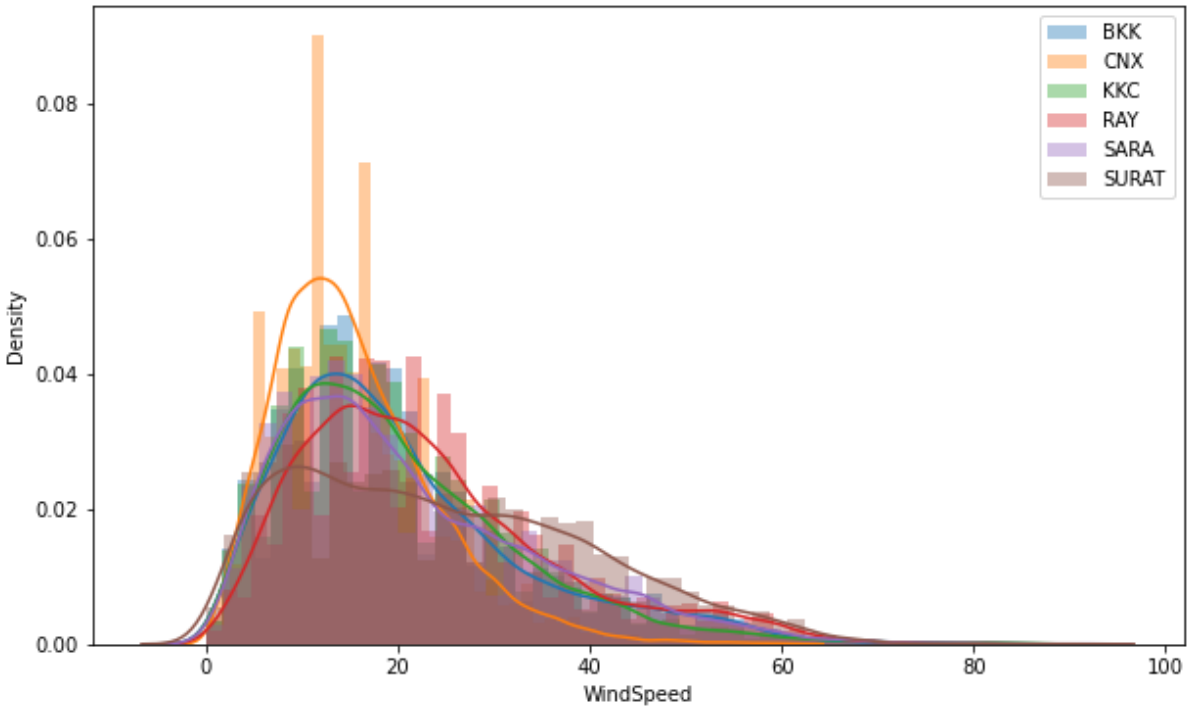
Data Analysis

Next, we implemented a function to do multiplots on each column from all provinces to display the trends of different provinces on the same types of data.

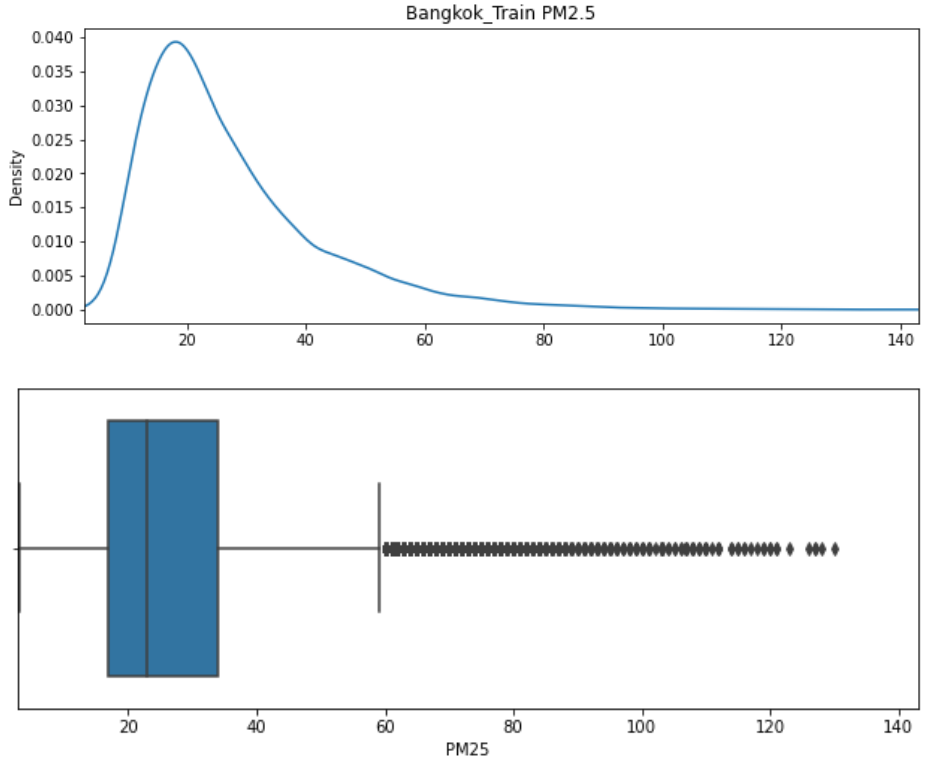


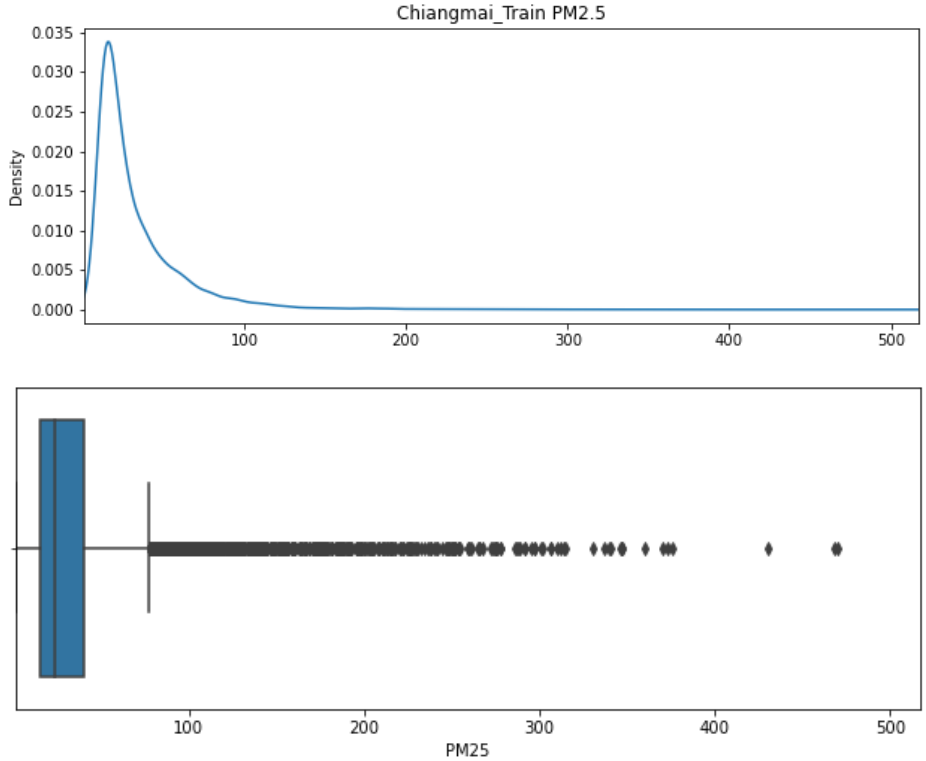


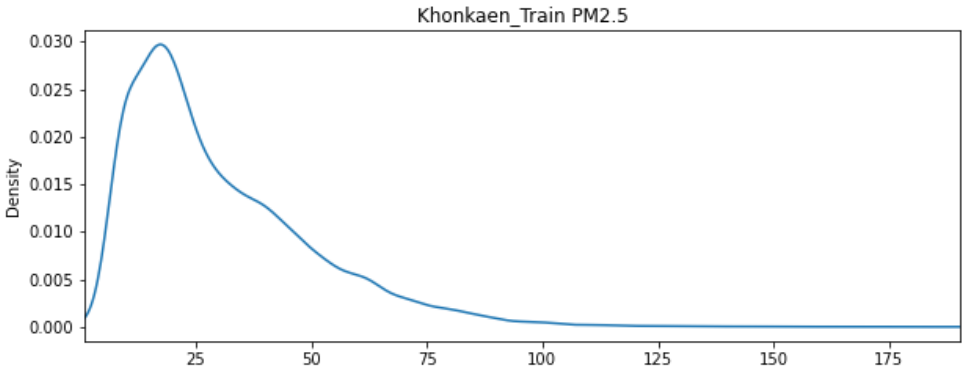

BKK
CNX
KKC
RAY
SARA
SURAT
Density
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
100
200
300
400
WindDir

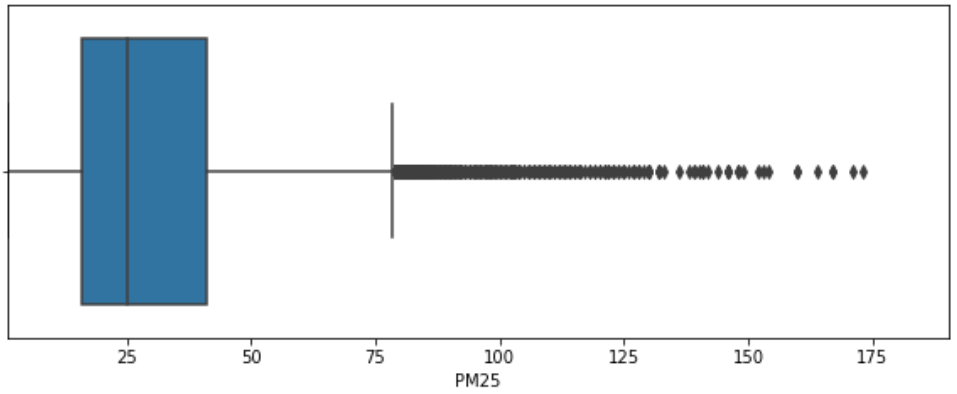

BKK
CNX
KKC
RAY
SARA
SURAT
Density
0.08
0.06
0.04
0.02
0.00
0
20
40
60
80
100
WindSpeed

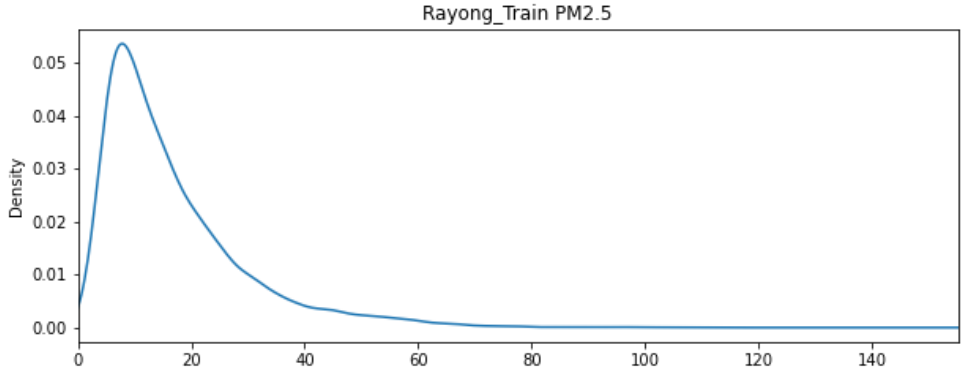
We implemented a new class, named ‘HandleOutliers’, to handle the outliers of the data columns. We have selected the IQR Box plot to display how the outliers of each province are distant from the range of the box plot.

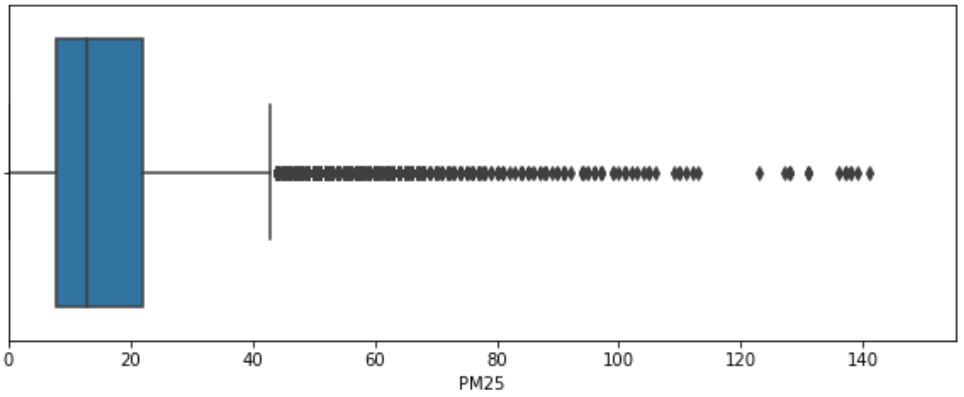


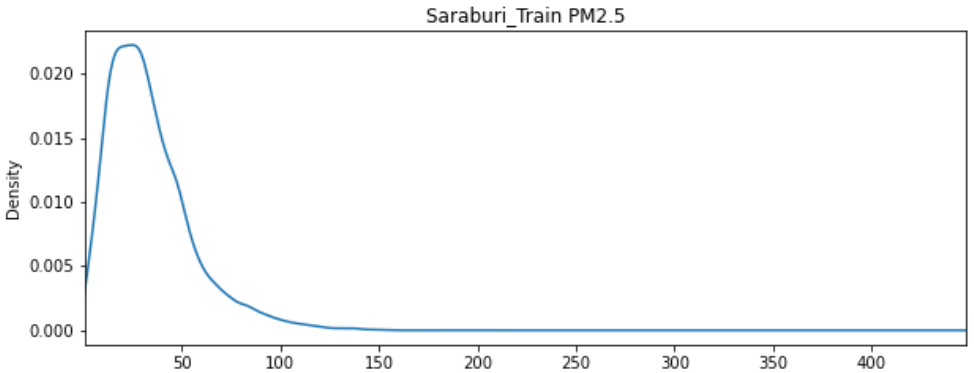


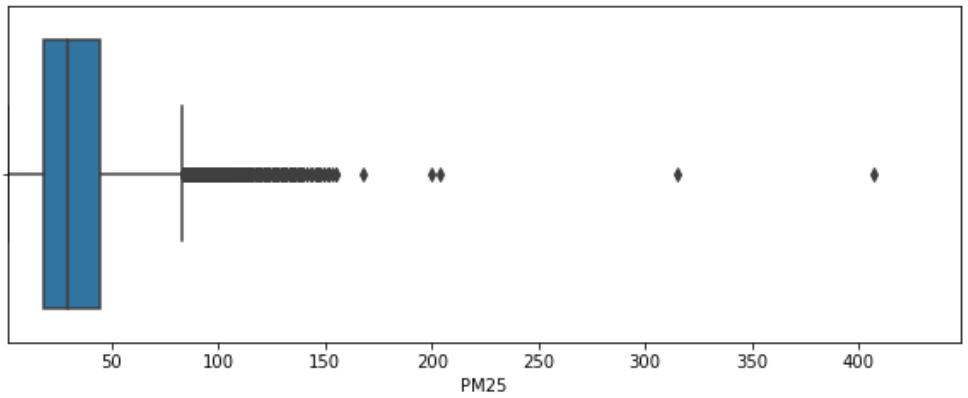

Khonkaen_Train PM2.5
Density
0.030
0.025
0.020
0.015
0.010
0.005
0.000
25
50
75
100
125
150
175

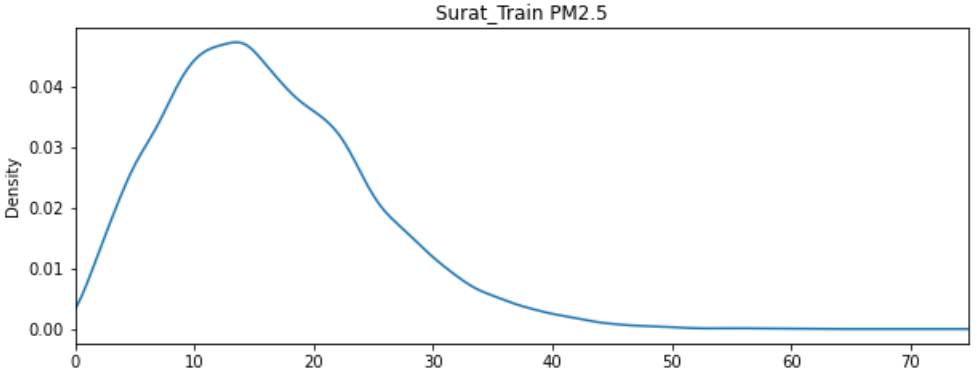

25
50
75
100
125
150
175
PM25

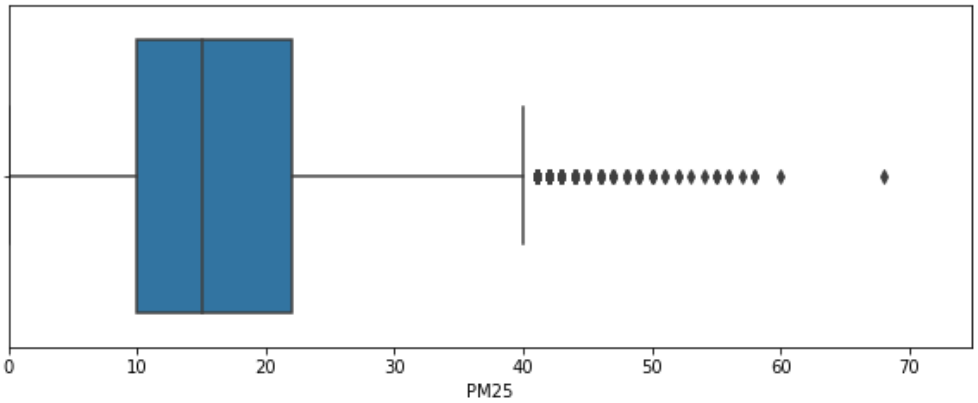

Rayong_Train PM2.5
Density
0.05
0.04
0.03
0.02
0.01
0.00
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
PM25

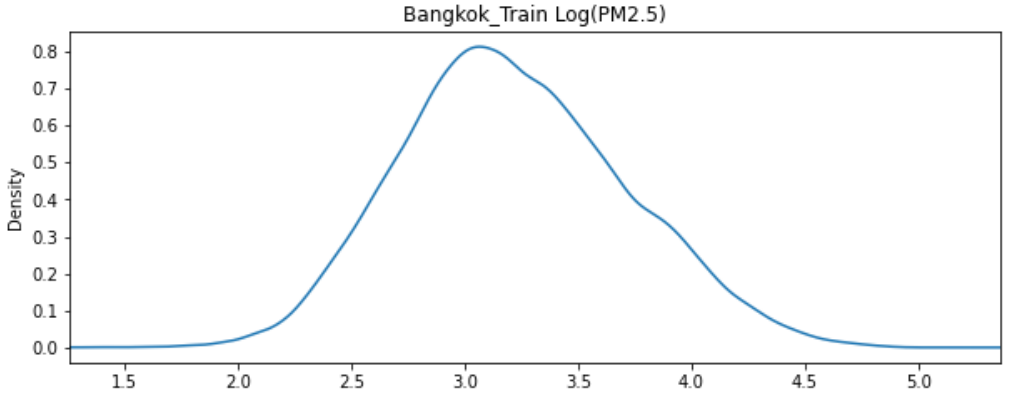

Saraburi_Train PM2.5
Density
0.020
0.015
0.010
0.005
0.000
50
100
150
200
250
300
350
400

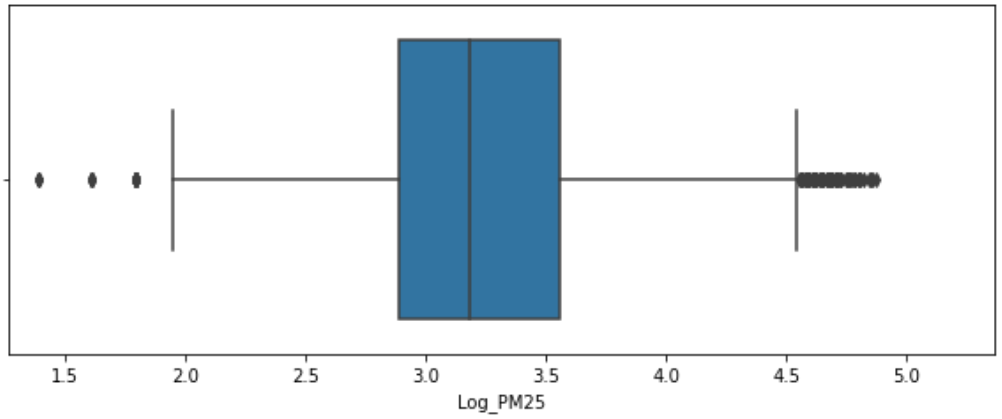

50
100
150
200
250
300
350
400
PM25

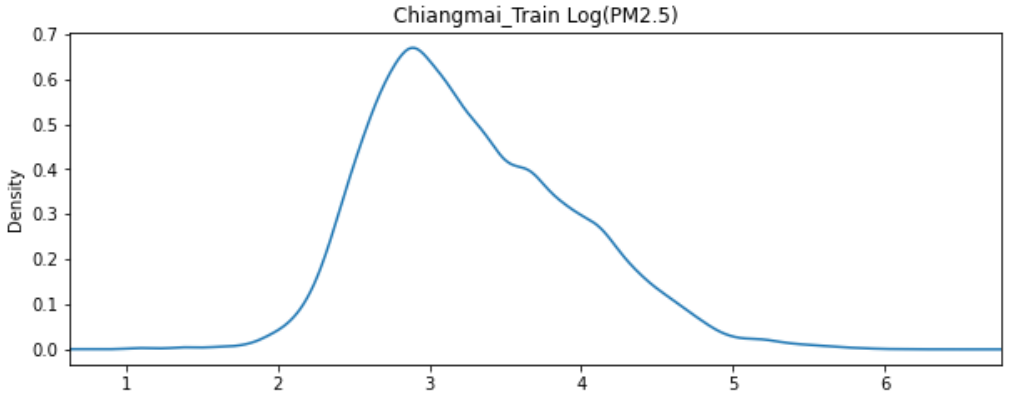

Surat_Train PM2.5
Density
0.04
0.03
0.02
0.01
0.00
0
10
20
30
40
50
60
70

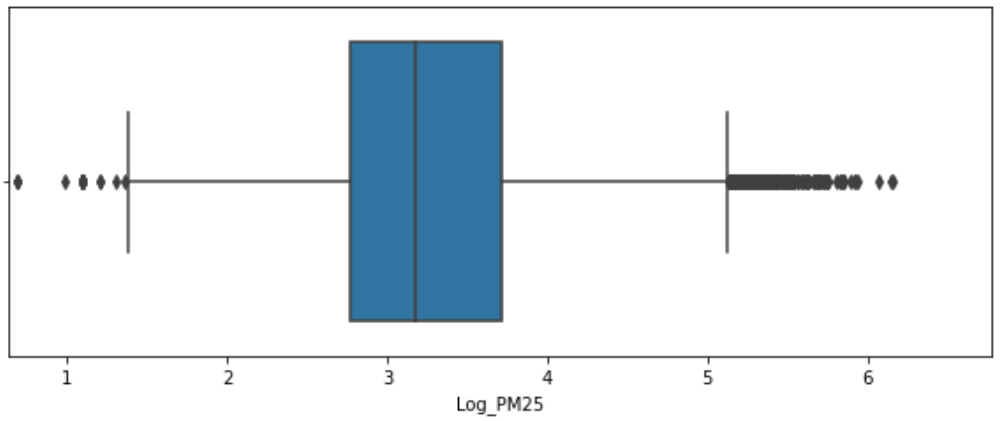

0
10
20
30
40
50
60
70
PM25

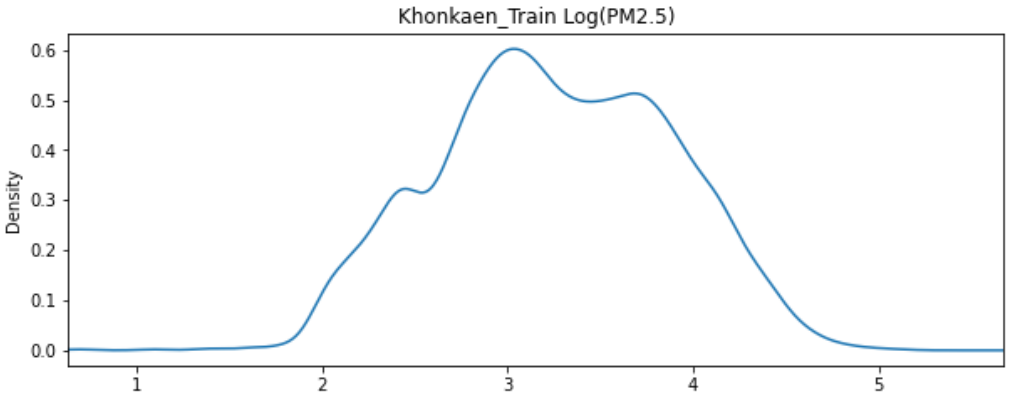
And we found that the kurtosis and skewness of all the data are not similar to normal distribution at all. Due to many outliers lying on the distances, we add a log column to change the ‘PM2.5’ to be distributed as Gaussian on logarithm function and display the IQR box plots.

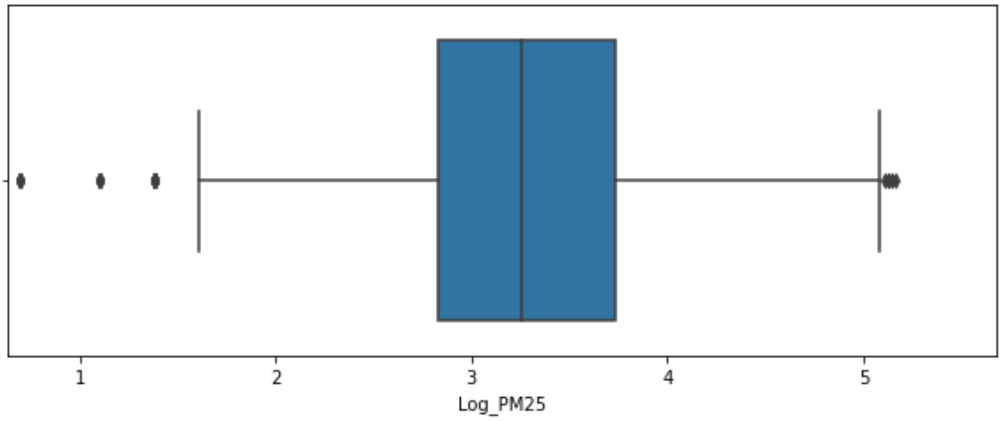


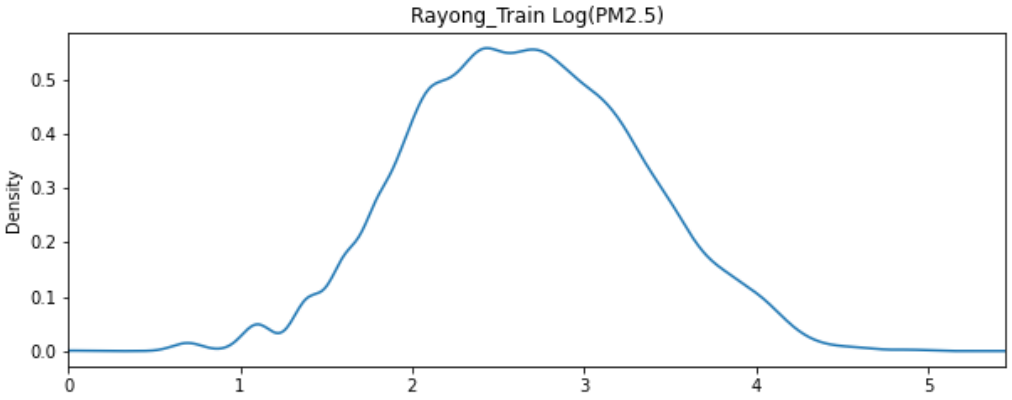


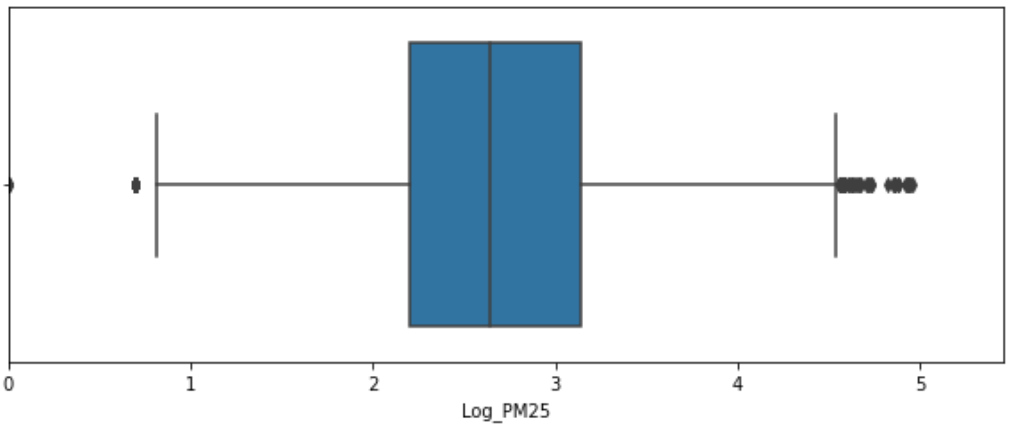


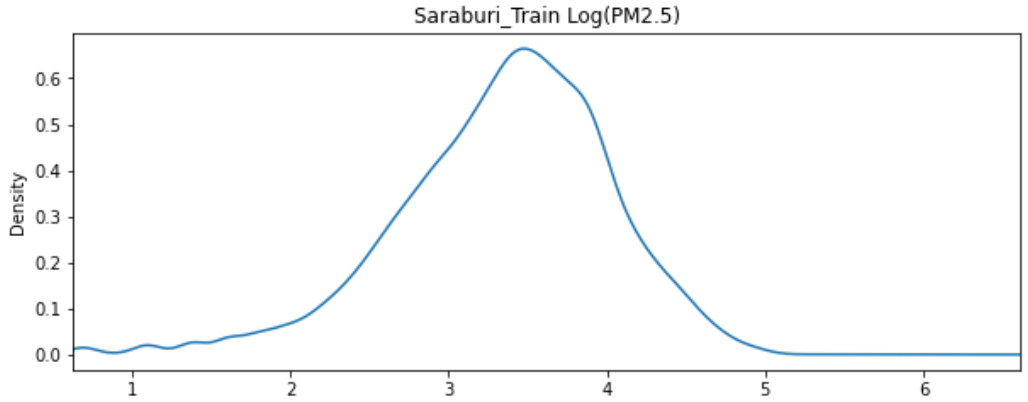


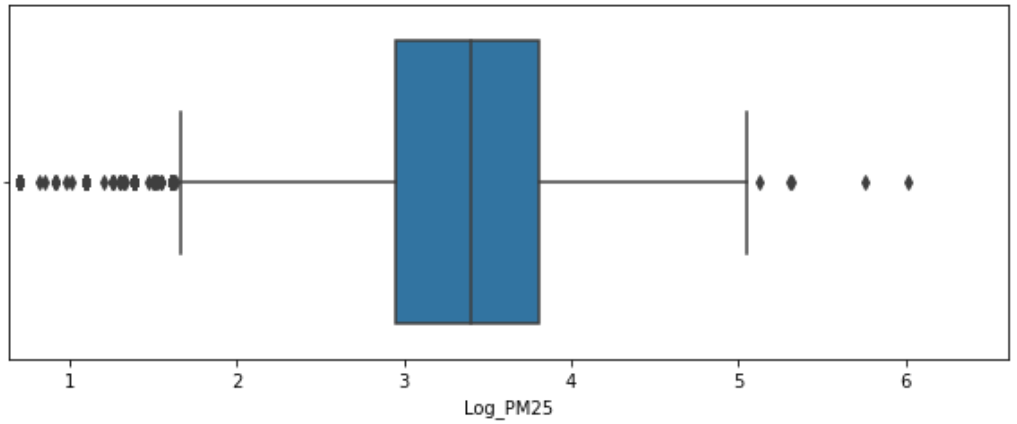

Khonkaen_Train Log(PM2.5)
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
1
2
3
4
5

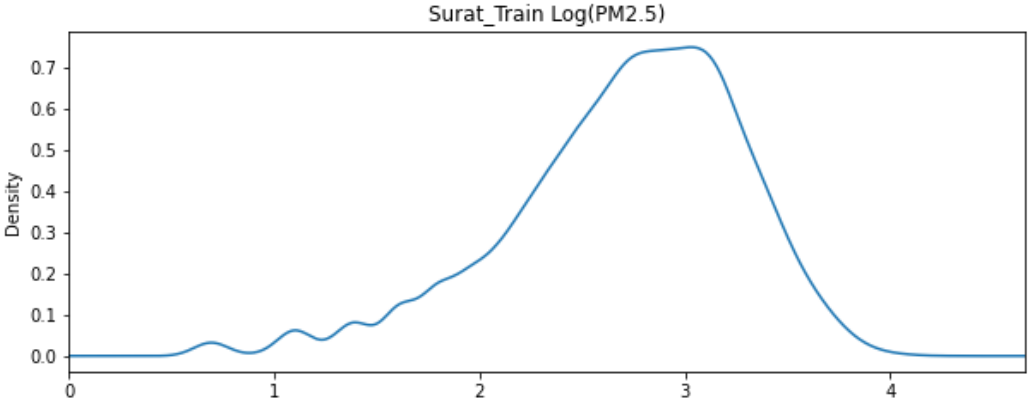

1
2
3
4
5
Log_PM25

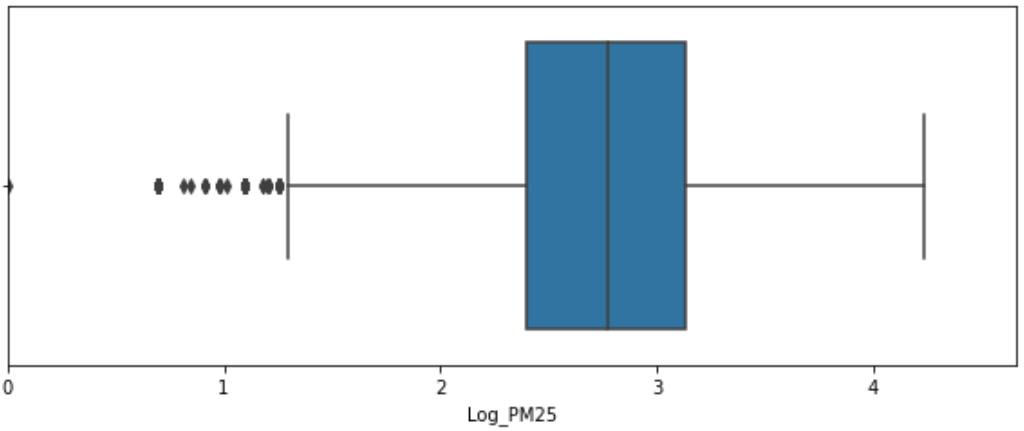

Rayong_Train Log(PM2.5)
Density
0.0
0.1
0.2
0.3
0.4
0.5
0
1
2
3
4
5


0
1
2
3
4
5
Log_PM25


Saraburi_Train Log(PM2.5)
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
1
2
3
4
5
6


1
2
3
4
5
6
Log_PM25


Surat_Train Log(PM2.5)
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
1
2
3
4

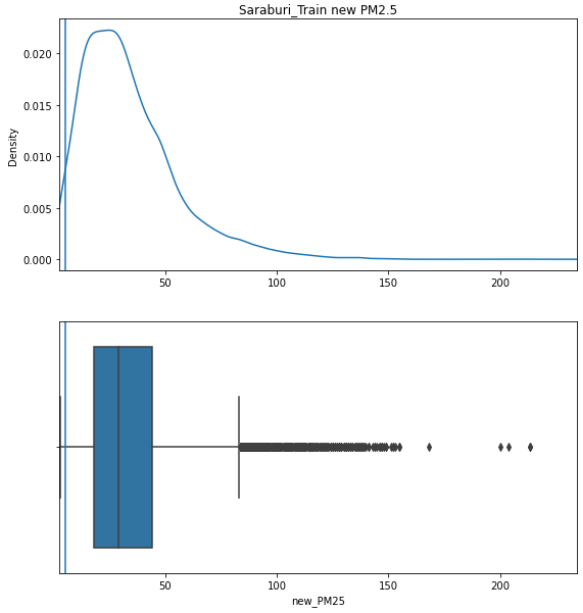

0
1
2
3
4
Log_PM25

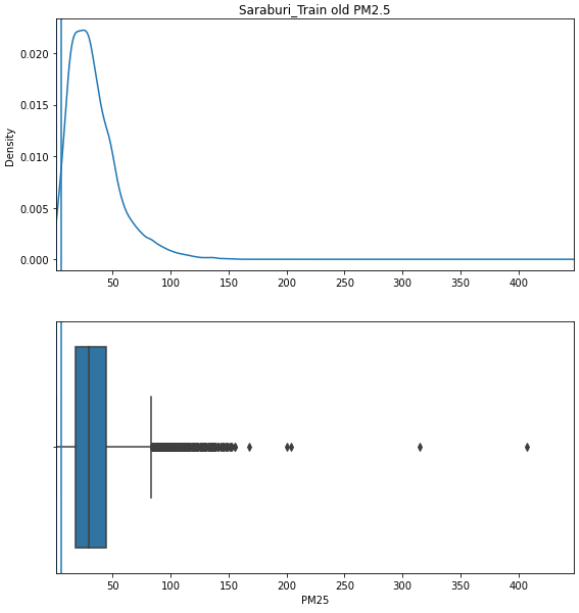
In the final stage, we set the min/max of the whole values to Log_PM25 to eliminate the outliers as we have intended. We showed the example of our outlier handling below.

```
thresh = {}

for prov, _ in train_set.items():
    thresh[prov] = {}
    thresh[prov]['min'], thresh[prov]['max'] = handle_outliers.calcOutliersMinMax(prov, 'Log_PM25', 3)
    handle_outliers.setOutliersThreshold(prov, 'Log_PM25', 'new_Log_PM25', thresh)

handle_outliers.addColumns('new_Log_PM25','new_PM25', lambda x: np.exp(x)-1)
```

```
handle_outliers.dataset['SARA'].describe()
```

	Temp	WindSpeed	WindDir	PM25	Log_PM25	new_Log_PM25	new_PM25
count	26304.000000	26304.000000	26304.000000	26304.000000	26304.000000	26304.000000	26304.000000
mean	28.329087	21.333143	171.397506	33.851144	3.351229	3.354467	33.849761
std	4.062689	13.649308	83.836522	21.925713	0.671936	0.660645	21.777485
min	14.000000	0.000000	0.000000	1.000000	0.693147	1.335420	2.801593
25%	25.400000	11.000000	90.000000	18.000000	2.944439	2.944439	18.000000
50%	27.800000	18.000000	180.000000	29.000000	3.401197	3.401197	29.000000
75%	31.200000	29.000000	250.000000	44.000000	3.806662	3.806662	44.000000
max	40.900000	91.000000	360.000000	407.000000	6.011267	5.367038	213.227455





Model Train

MinimalSARIMAX

We decided to use the SARIMAX model for our PM2.5 prediction,but we wrote a class and functions of the model by ourselves since we thought that the models from libraries, such as statsmodels, would take too much time to be trained and predict the results (our big mistakes).

Describing the functions of our model in brief:

(1.) First, we initialized all parameters for the model such as weight parameters (‘p’, ‘d’, ‘q’, ‘P’, ‘D’, ‘Q’, ‘c’) in ‘__init__’ .

(2.) The functions of matrix multiplications were built to manipulate the format of matrix inputs for ‘p’, ‘d’, ‘q’, ‘P’, ‘D’, ‘Q’ as ‘p_prediction’, ‘d_prediction’, ‘q_prediction’, ‘P_prediction’, ‘D_prediction’, ‘Q_prediction’ respectively.

(3.) The ‘update_params’ function is to update the weight parameters, which are updated continuously every 6 hours. This function had been implemented based on a gradient descent algorithm minimizing the (mean) error between predicted values (at most 12, as 12 timestep forecasting) and the day arriving today. You might wonder if we put the future data before the prediction. Please be waiting to take a look at the ‘predict_step’ function.

(4.) The ‘fit’ function of our library is not fully functional as the well-known libraries do. The function receives the inputs of learning rate (‘lr’), and learning rate decay (‘lr_decay’) to train the model itself with the training data being inputted at the initialization step.

(5.) In order to fit our model, we need to predict new values of tomorrows again and again and minimize the error of the predicted ones.

Hence, we created the ‘predict_one’ function which calls the parameter prediction functions in (2.) to calculate and sum the result of matrix multiplication, then returns as a predicted value, ‘pred[‘y’]’. Moreover, the function returns ‘x’ which will be used to update the weight when the next day arrives.

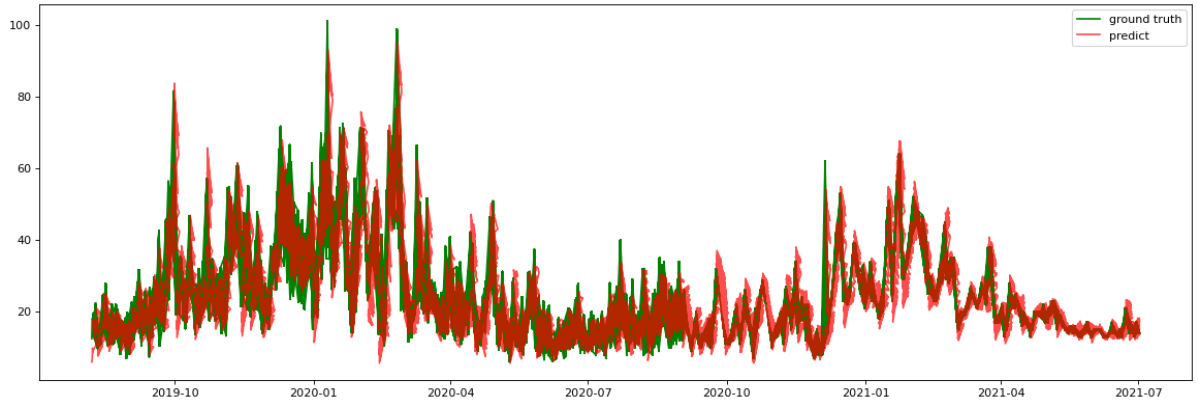
(6.) For the ‘predict’ function, we have used it to test on a normal prediction, which has no timestep parameter entered. Anyways, it is no longer used after we have implemented the prediction function with timesteps.

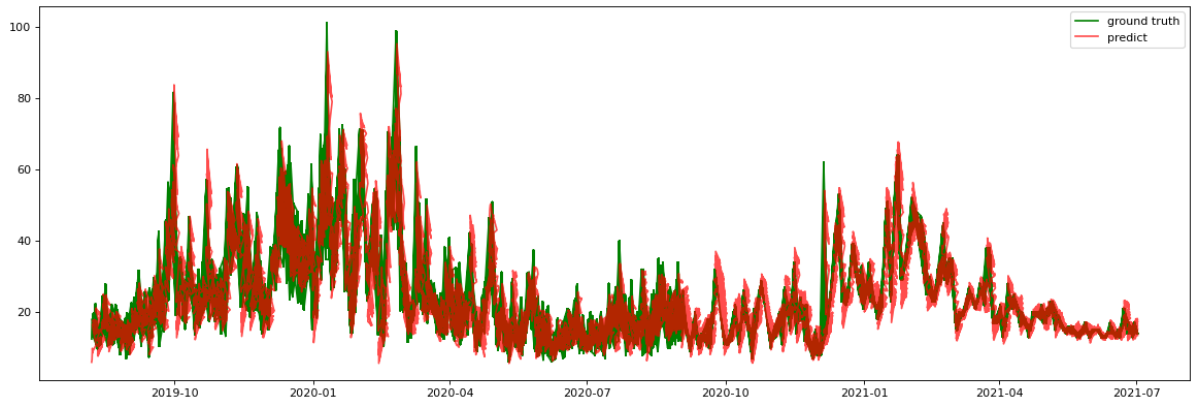
(7.) Let’s come to the main function of our prediction. The function, named ‘predict_step’, is to perform on the train/valid/test inputs (an exogenous column is optional) and returns the predicted values of all those data inputs over its range of the time period. The algorithm of this function is to predict the next time-step value based on the previous sequence of the days before, since we have known the values of all columns. On a new day coming, we recalled all the predicted values that we had predetermined on that day (12 values as usual) from the dataframe, named ‘twelve’ for the ‘PM25’ column and ‘twelve_x[i]’ for each exogenous column. The recalled values were calculated the mean values of each column, then the function computed the error of these values and the actual values. In the next step, the function called the ‘update_params’ function to adapt the weights of the model in the need to follow up-to-date with the current trends. To clarify, the model always updates its parameters after it finishes each 12 steps forecasting which performs on the incorrect values that the model itself has predicted previously. Thus, the model can learn over days passing to minimize the error which increases every time when it performs on its own false predicted values, and provide more accurate on time-step predictions.

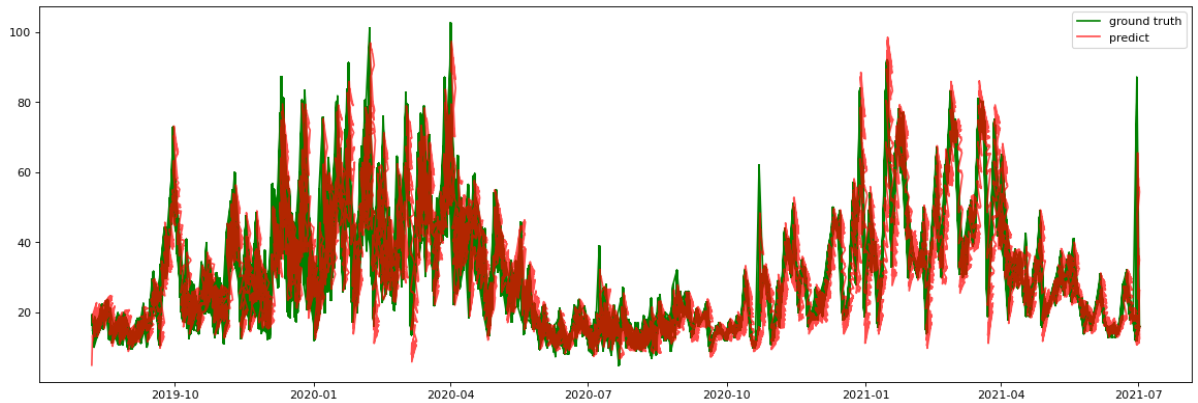
Let’s take a closer look at the results we received from this model.

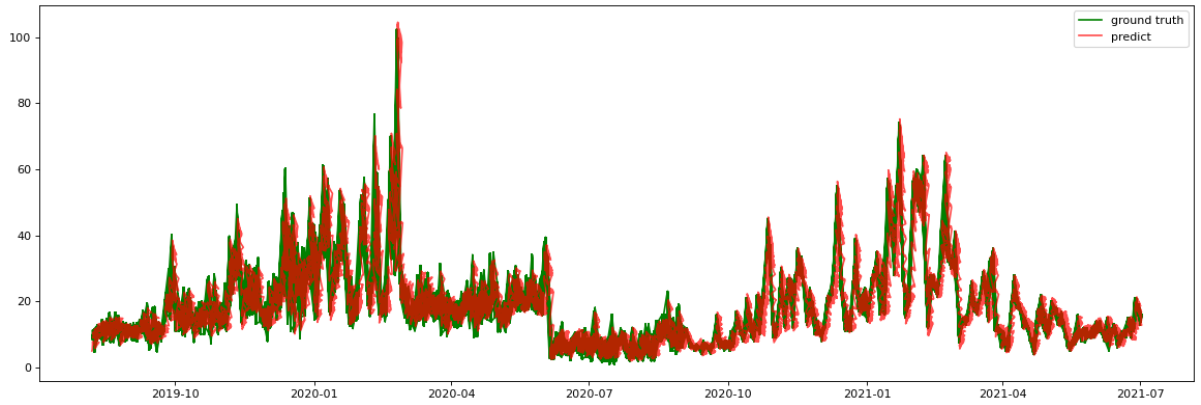
The graph is in order Bangkok, Chiangmai, Khonkaen, Rayong, Saraburi, Surat respectively.

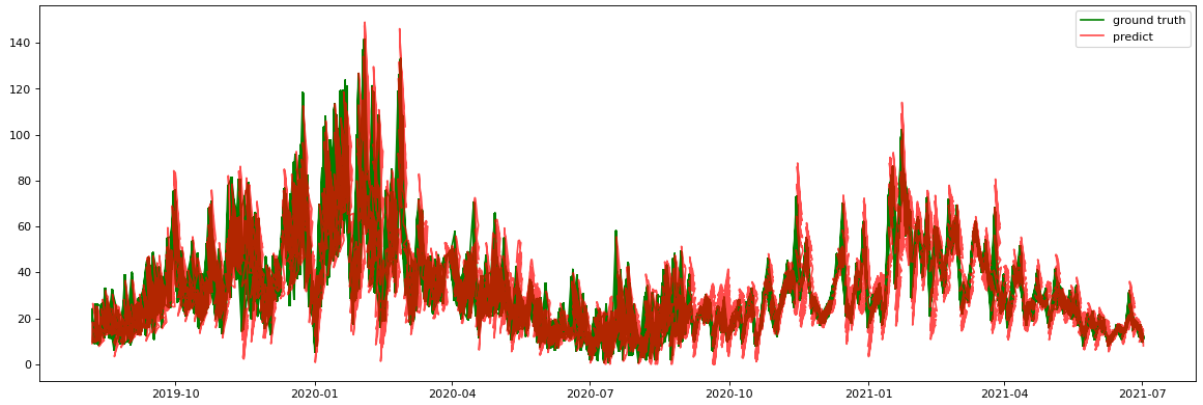
(Prediction on test set)

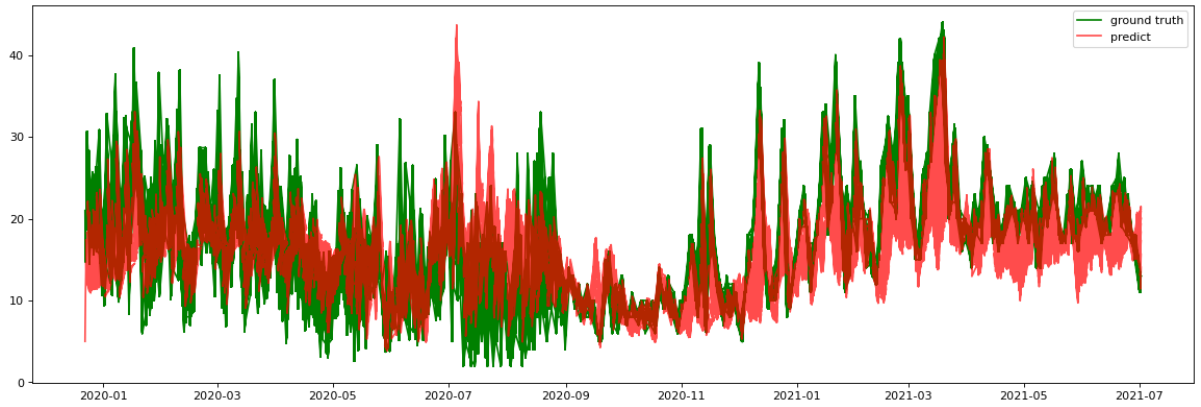







ground truth
predict
100
80
60
40
20
0
2019-10
2020-01
2020-04
2020-07
2020-10
2021-01
2021-04
2021-07


ground truth
predict
140
120
100
80
60
40
20
0
2019-10
2020-01
2020-04
2020-07
2020-10
2021-01
2021-04
2021-07


ground truth
predict
40
30
20
10
0
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07

Randomized Search

```
def sarimax_randomsearch(y_train, y_test, pdq, PDQs, y_val = None, x_train = None, x_test = None, x_val = None, model_exog=None, verbose=0, n_rand=10):
    '''
    Input:
        y_train: training data
        y_test: test data
        pdq : ARIMA combinations
        pdqs : seasonal ARIMA combinations
        x_train: exogenous training data
        x_test: exogenous test data

    Return:
        Returns dataframe of parameter combinations with the least RMSE
    '''

    ans_df = pd.DataFrame(columns=['pdq', 'pdqs', 'rmse'])

    save_comb = set()

    i = 0
    while i!=n_rand:
        comb = random.sample(pdq, 1)[0]
        combs = random.sample(PDQs, 1)[0]

        if (comb+combs) in save_comb:
            continue

        save_comb.add(comb+combs)

        p, d, q = comb[0], comb[1], comb[2]
        P, D, Q = combs[0], combs[1], combs[2]
        if (d <= 1) and (D <= 1) and (P <= 1) and (Q <= 1):
            model = MinimalSARIMAX(y_train, comb, combs, exog=x_train)
            model.fit(lr=1e-6, lr_decay=0.999, verbose=0)

            if (y_val is None):
                y_pred, err = model.predict(y_test, y_exog=x_test, verbose=verbose)
                rmse = model.scoring(y_pred, y_test)

            else:
                Result = model.predict_step(y_val, y_test, val_X_exog=x_val, y_exog=x_test,
                                            model_exog=model_exog, lr=np.array([1e-6, 1e-6, 1e-6, 1e-6, 1e-6]), lr_decay=0.9995,
                                            learn=True, verbose=verbose, verbose_rmse=0)

                _, y_pred_sav, _ = Result

                rmse = model.scoring(y_pred_sav.iloc[:,[1]], y_pred_sav.iloc[:,[2]])

            print(f"ITER#{i} {comb} {combs} {rmse}") ; i=i+1

            ans_df = ans_df.append({'pdq':comb, 'pdqs':combs, 'rmse':rmse}, ignore_index=True)
        else: continue

    # Sort and return a combination with the lowest RMSE
    ans_df = ans_df.sort_values(by=['rmse'],ascending=True)

    return ans_df
```

We then defined a custom randomizedSearch function for tuning the parameters, p, d, q, P, D, Q within a range between 0 and 3 (exclusively).

```
model_exog_surat = {}
exog_order_surat = {}
exog_seasonal_order_surat = {}

model_exog_surat['Temp'] = MinimalSARIMAX(temp_train_surat, (2, 0, 2), (0, 0, 0, 1461))
model_exog_surat['WindSpeed'] = MinimalSARIMAX(windSpeed_train_surat, (2, 1, 0), (1, 0, 1, 1461))
model_exog_surat['WindDirSin'] = MinimalSARIMAX(windDirSin_train_surat, (1, 0, 2), (0, 1, 0, 1461))
model_exog_surat['WindDirCos'] = MinimalSARIMAX(windDirCos_train_surat, (2, 0, 2), (0, 1, 0, 1461)  )

for exog in exog_columns:
    model_exog_surat[exog].fit(lr=1e-6, lr_decay=0.999, verbose=0)

result_surat = sarimax_randomsearch(pm_train_surat, pm_test_surat, pdq, pdqs, y_val=pm_valid_surat, x_t
                                    model_exog=model_exog_surat, n_rand=3, verbose=1)

display(result_surat)
```

✓ 480m 16.6s

```
100%|██████████| 16675/16675 [2:37:56<00:00,  1.76it/s]

ITER#0 (1, 0, 1) (0, 1, 1, 1461) 11.944792679813055

100%|██████████| 16675/16675 [2:41:48<00:00,  1.72it/s]

ITER#1 (0, 0, 2) (1, 0, 0, 1461) 11.70976219682504

100%|██████████| 16675/16675 [2:38:21<00:00,  1.76it/s]

ITER#2 (2, 0, 2) (0, 1, 0, 1461) 11.451204403766797
```

	pdq	pdqs	rmse
2	(2, 0, 2)	(0, 1, 0, 1461)	11.451204
1	(0, 0, 2)	(1, 0, 0, 1461)	11.709762
0	(1, 0, 1)	(0, 1, 1, 1461)	11.944793

```
gSearch_temp_sara = sarimax_randomsearch(temp_train_sara,
display(gSearch_temp_sara)
```

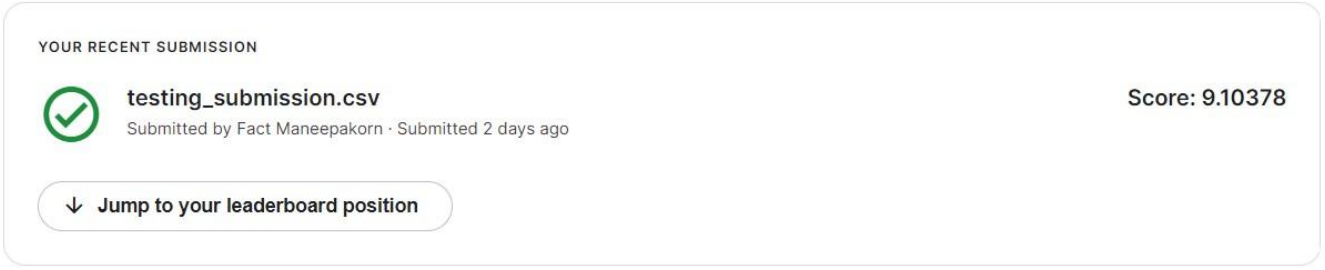
✓ 102m 6.5s

```
100%|██████████| 16675/16675 [19:29<00:00, 14.26it/s]

ITER#0 (2, 1, 2) (1, 0, 1, 1461) 4.372241089480689

100%|██████████| 16675/16675 [20:14<00:00, 13.73it/s]

ITER#1 (2, 0, 1) (0, 0, 0, 1461) 4.610449766166291

100%|██████████| 16675/16675 [20:34<00:00, 13.50it/s]

ITER#2 (0, 0, 2) (0, 0, 0, 1461) 24.94109239910067

100%|██████████| 16675/16675 [20:28<00:00, 13.57it/s]

ITER#3 (2, 0, 2) (0, 0, 1, 1461) 4.938988995499015

100%|██████████| 16675/16675 [20:18<00:00, 13.69it/s]

ITER#4 (2, 1, 1) (0, 0, 0, 1461) 4.644085648835342
```

	pdq	pdqs	rmse
0	(2, 1, 2)	(1, 0, 1, 1461)	4.372241
1	(2, 0, 1)	(0, 0, 0, 1461)	4.610450
4	(2, 1, 1)	(0, 0, 0, 1461)	4.644086
3	(2, 0, 2)	(0, 0, 1, 1461)	4.938989
2	(0, 0, 2)	(0, 0, 0, 1461)	24.941092

Kaggle submissions

Kaggle submission screenshot



RMSE (calculated on local notebook)

```
np.sqrt((np.array(final_answer['Predict'] - final_answer['Actual'])**2).mean())
```

```
9.103784095360165
```

RMSE for each province

RMSE

```
for sb in [sb_bkk, sb_cnx, sb_ray, sb_sara, sb_kkc, sb_surat]:
    sb[sb['Predict']<0] = 0
    diff = np.sqrt((np.array(sb['Predict'] - sb['Actual'])**2).mean())
    print(diff)
```

```
6.24731247042558
9.744932835576
7.602580886955378
11.75068202216614
10.763007935018155
7.2878734947065915
```