AI Capstone Project 1 Report – Crowd Flow Prediction

111550108，吳佳諭

1. Link to my dataset

Check this link to access my dataset: https://github.com/chia-yuu/AI-Capstone

2. Research question: THSR crowed flow prediction

I always go home by taking Taiwan High Speed Rail (THSR). Sometimes it's very easy to buy a discount ticket. If I forgot to buy tickets in advance, I can take the non-reserved seats. It's not difficult to take THSR to go home. However, sometimes I just cannot buy any ticket. And there are lots of people in the station, so it's really hard to get a non-reserved seat, unless you are super lucky. Therefore, in this project, I want to build a model that can predict the crowed flow in THSR station. If the model can give a good prediction, maybe in the next time when I want to go home, I can avoid the time that are lots of people taking trains, so it's easier for me to buy tickets, and I don't have to squeeze in the crowd!

In this project, I focus on ***how many people will enter a specific station in a specific month in the future***. For example, how many people are entering Hsinchu station in 2024.8. The definition of "how many people" means the number of people "entering" the station, but not "leaving" or "total" because I just want to know how many people will enter the same station (and maybe take the same train) with me when I want to take the train.

3. Dataset

(1) Data source

I used web crawlers to get the passenger traffic of High-Speed Rail Station data from THSR official website (https://www.thsrc.com.tw/corp). The website provides the number of arrival and departure passengers every month in the 12 stations from 2017-01 to 2025-01. I get the data from 2022-01 to 2025-01 to train and test the model.

In addition, specific holidays (疏運期間) and number of train schedule every week will affect the passenger traffic, so I also include them in the training data. The information can also be found on THSR website.

As for number of people diagnosed with COVID-19, it's from government open data website and then calculate to get the correct time range.

(2) Data collection and preprocess

- **Web crawlers**

I use web crawlers to collect the data and save it in thsr_raw.csv. The related program is in data.py. Run this file to get the raw data.

- **Factors that may affect the crowd number**

Holiday: people tend to go home or go out to play in holidays. THSR also have some policies regarding holidays. There are always much more people in the station when it's holidays, so the number of holidays in a month is definitely one of the factors that will affect the crowd number.

Diseases (ex: COVID-19): if there's some serious disease recently, people tend not to take public transportation. In 2021.5, when lots of people were diagnosed with COVID-19, the number of people taking THSR has a significantly decrease.

Number of available trains: the more train available, the more passenger it can take. THSR has increased the number of train schedule every week five times since 2022 to meet the need of so many passengers.

- **Preprocess**

When training the model, the program will first call Data_preprocess function in utils.py to get the processed data (store as data frame and return to main function).

Add number of holidays in the month to the data: as I mentioned above, holiday is an important factor that will affect the passenger traffic, so I add a new column in the dataset to record the days of the holiday.

Data splitting: I split the dataset into training data, validation data, and testing data. Each contain few months and all stations' crowd number. The splitting is continuous, not random, because time is meaningful in crowd flow prediction. We can use the past few months' data to predict the next month, but not using future data to determine the historical data. If we split the data randomly, the time will become meaningless.

Data	Range	Amount
Training	2022-01 ~ 2023-12	24 months (288 rows, 65% of the data)
Validation	2024-01 ~ 2024-06	6 months (72 rows, 16% of the data)
Testing	2024-07 ~ 2025-01	7 months (84 rows, 19% of the data)

Station id: at first, I use 1~12 to represent the stations, but later I found that using the average crowd number in that station can represent the stations better. Some stations such as Taipei have more passengers than others. Using average crowd number in that station as station id can provide more information to the model.

(3) Data description

The dataset consists of 37 months, 12 stations, and a total of 444 rows. thsr_raw0.csv, which includes only [Date, Station id, Crowd number], delivers better performance. In

contrast, thsr_raw1.csv, which contains [Date, Station id, Crowd number, Schedule number, Disease number], results in poorer predictions.

Column name	Data type	Description	Example
Date	Integer	Month from 2022-01 to 2025-01. Format: yyyymm	202501
Station id	String	Chinese name of the 12 stations	南港
Crowd number	Integer	Passenger traffic of that month	360796
Schedule number	Integer	Number of train schedule every week	1103
Disease number	Integer	Number of people diagnosed with COVID-19	0

After preprocessing, the final data looks like the following.

Column name	Data type	Description	Example
Date	Integer	Month from 2022-01 to 2025-01.	202312
Station id	Float	Average crowd flow in that station	247475.9
Crowd number	Integer	Passenger traffic of that month	332100
Schedule number	Integer	Number of train schedule every week	1060
Disease number	Integer	Number of people diagnosed with COVID-19	14
Special days	Integer	Number of specific holidays in that month, ex: new year, Double Tenth	3

Data example.

Raw data (first five)

date	station id	crowd number	schedule number	disease number
202312	南港	332100	1060	14
202312	台北	1442644	1039	14
202312	板橋	515867	1039	14
202312	桃園	785005	1039	14
202312	新竹	631655	1039	14

After preprocess (first five training data)

	date	station id	crowd number	schedule number	disease number	special days
202312	2.474759e+05	332100	1060	14	3	
202312	1.155741e+06	1442644	1039	14	3	
202312	3.962060e+05	515867	1039	14	3	
202312	5.766890e+05	785005	1039	14	3	
202312	4.912195e+05	631655	1039	14	3	

4. Methods

(1) Supervised learning 1: Random Forest

Random forest is an easy algorithm because it's idea and implementation are easy to understand. It is also a brute force solution because it's an ensemble method. It is made up of many decision trees, so it's computation may be large.

There are two main reasons I chose random forest for the task. First, it can handle non-linear relationship. The crowd number may depend on many factors like time and holidays. These factors may not have linear relationship with the crowd number, but random forest

can effectively capture these features. Secondly, it is robust to outliers. The outliers will fall into leaves and won't affect the result too much. Crowd flow in different stations can be very different, but random forest can still give a good prediction.

To build the model, I import RandomForestRegressor from sklearn.

(2) Supervised learning 2: LSTM (Long Short-term Memory)

LSTM is a kind of RNN. Its memory cell has three gates, controlling what data can be written, read, or forgot from the memory cell. As a result, it can store information for long periods and forget unrelated details, making it suitable for time-series data.

Before training data, I use StandardScaler from sklearn to normalize the data. Then I use tensorflow.keras to build my mode. The model starts with two layers of LSTM, each has 128 units. And then a Dense layer with 32 units and rule as activation function. Finally, there is another Dense layer with 1 unit to output the prediction.

After try and error testing, the time window is set to 6 and epoch is 100.

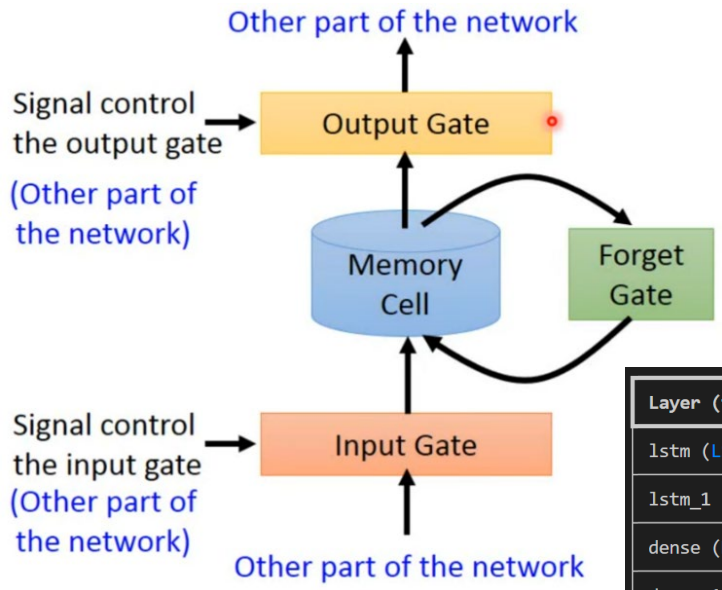

Fig. 1. Four components in LSTM (from Hung-yi Lee ML lecture 21-1 video)

Fig. 2. My model architecture

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 6, 128)	67,584
lstm_1 (LSTM)	(None, 128)	131,584
dense (Dense)	(None, 32)	4,128
dense_1 (Dense)	(None, 1)	33

(3) Unsupervised learning: K-means

K-means clustering is an unsupervised learning method. It partitions data into k clusters by similarity. Each data is assigned to the nearest cluster center and iteratively update the centers to minimize the variance in each cluster.

I import StandardScaler and KMeans from sklearn. The first one is used to normalize data, while the second one is my model. It groups data into several groups, and predict which group is the new data in. The prediction is the group's average crowd flow. After doing some experiments, I found that the model can reach the best performance with k = 100.

5. Experiment

(1) Prediction result

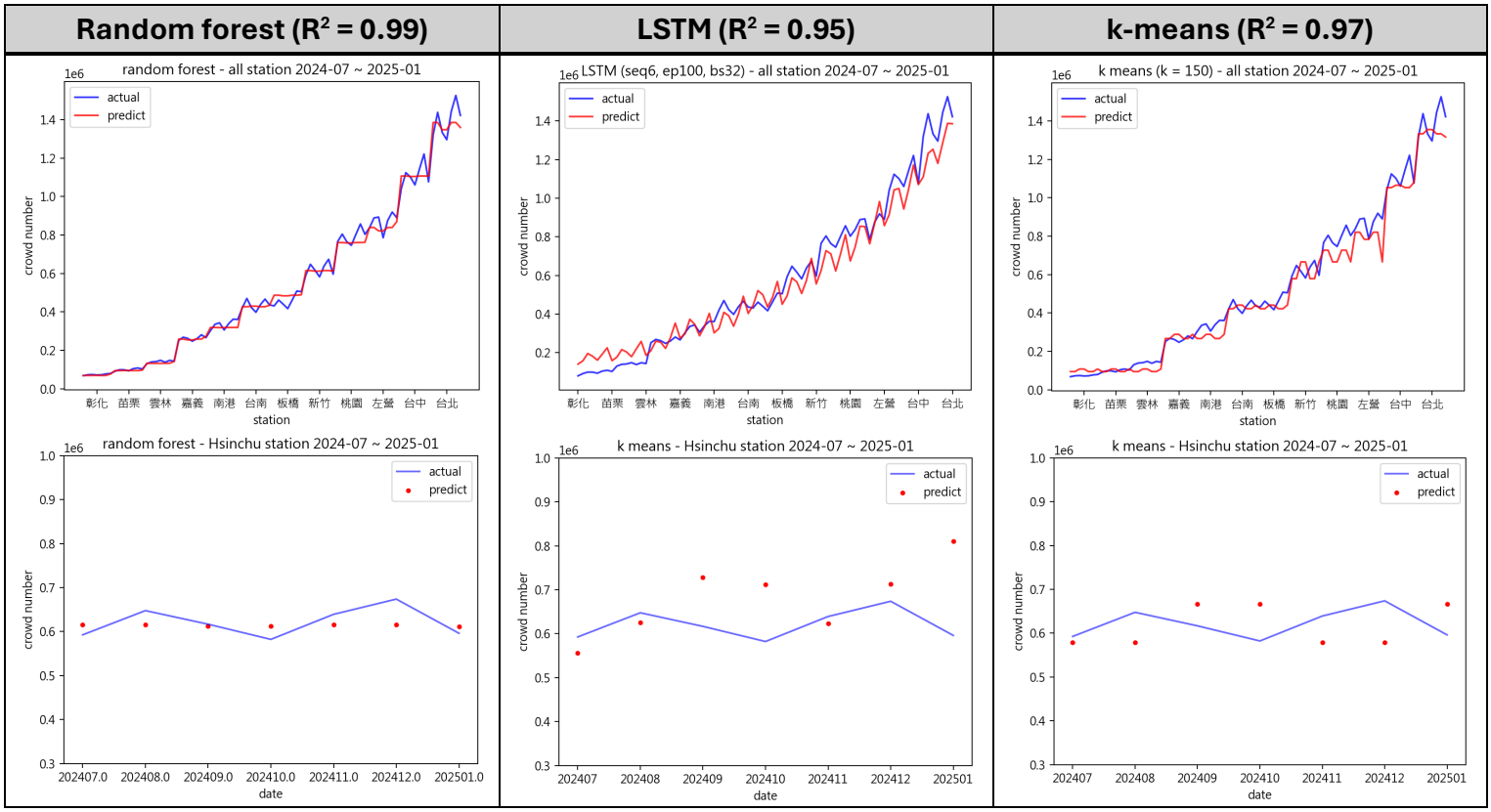

Fig. 3. Prediction result of three methods. Data = [date, station id, crowd number, specific holiday] sorted by station. Upper ones are prediction for all station, lower ones are prediction for Hsinchu station.

(2) Performance

The prediction performance is shown in Table 1 and Fig 4. Baseline is to predict using each station's average crowd flow. At first, I only use "date", "station id", "crowd number", and "specific holidays" to train the models. Random forest has the best performance, while LSTM the worst. I thought k-means will perform the worst because I use the average crowd flow in each cluster to be the prediction result. I thought it can only give a rough estimation and therefore cannot have a good performance. In contrast, I thought LSTM is deep learning method, so it could perform the best, but it turns out it is not that case.

Then I remove "specific holidays" in the dataset. It turns out that without the number of specific holidays, the performance of random forest and k-means has a better performance, while LSTM perform worse than original. I think it's because my data's time unit is in month, and the number of the specific holidays in each month has not much difference. Hence, it won't affect the crowd flow too much. On the other hand, if the data is the crowd flow per day. Maybe the specific holidays will affect the result more.

Finally, I add “number of people diagnosed COVID-19” and “number of train schedule” to the dataset. To my surprise, adding more features doesn’t improve the models’ performance, instead, they give a worse prediction. I think it’s because in k-means, giving too much features will not help the training, but mislead the model to learn a wrong relationship between data and put data to wrong cluster. This experiment shows how important the training data is. If we use inappropriate data, our model cannot capture the correct features and relationship and give a good performance.

Table 1: comparison of prediction performance in different models and different data

Data	Data 1**			Data 2**			Data 3**		
	RMSE	**R^2**	**MAPE**	**RMSE**	**R^2**	**MAPE**	**RMSE**	**R^2**	**MAPE**
Baseline	129844.7	0.86	23.4%	129844.7	0.86	23.4%	129844.7	0.86	23.4%
RF*	41570.5	0.99	4.2%	38396.1	0.98	8.0%	38636.3	0.99	5.09%
LSTM	118867.8	0.95	20.1%	89684.5	0.94	13.6%	251519.7	0.59	61.24%
kmeans	66831.4	0.97	12.1%	51407.9	0.98	9.0%	404200.3	0.01	114.3%

*RF means random forest.

** Data 1 = “date”, “station id”, “crowd number”, “number of specific holidays”. Data 2 = data 1 without “number of specific holidays”. Data 3 = data 1 plus “number of train schedule”, “number of people diagnosed with COVID-19”

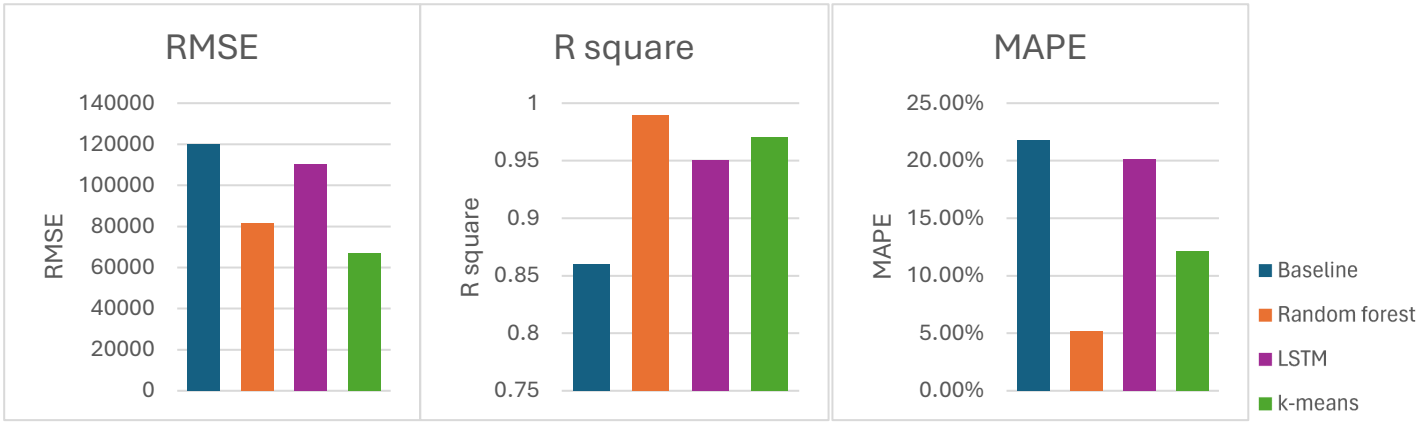

Fig. 4. comparison of prediction performance in different models using data1

(3) Data sorting

Table 2: prediction result with and without data sorted in time

	Data sorted in time			Data sorted in station		
Model	**RMSE**	**R square**	**MAPE**	**RMSE**	**R square**	**MAPE**
Baseline	129844.7	0.86	23.4%	129844.7	0.86	23.4%
Random forest	41570.5	0.99	4.2%	38214.8	0.99	5.1%
k-means	70004.7	0.97	13.4%	66831.4	0.97	12.1%
LSTM	177561.9	**0.78**	**27.1%**	118867.8	**0.95**	**20.1%**

LSTMs can capture time dependencies and temporal patterns in time-series data, while random forest and k-means don't care about the time series. Therefore, how the data is sorted will affect the prediction performance. The raw data is sorted in time (same date but different station's data are put together), and then I tried to do the prediction with data sorted in station (same station but different date's data are put together). The result is shown in Tabe2. When data has temporal dependencies (sorted in station), LSTM can capture the feature better. In contrast, due to the algorithm and how they do the prediction, random forest and k-means don't affect by the data order as much as LSTM.

(4) Adjust the number of data

I tried to use different amount of data, that is, data from 2022-01, 2023-01, and 2020-01, to train the model. The result in Table 3 shows that using less data make the performance worse, which is same as my expect. But using more data neither improve the performance. I think it's because crowd flow between 2020 to 2022 is affected by COVID-19. Later in the testing data, the crowd flow has back to normal, but the model will under estimate the crowd flow. This also indicates that it is important to choose a reasonable range of training data. Different training data will lead to different result.

Table 3: train the model using different amount of data (*RF means random forest)

Data	2022-01 ~ 2025-01			2023-01 ~ 2025-01			2020-01 ~ 2025-01		
	RMSE	**R^2**	**MAPE**	**RMSE**	**R^2**	**MAPE**	**RMSE**	**R^2**	**MAPE**
RF*	41570.5	0.99	4.2%	79103.3	0.96	10.7%	96790.2	0.94	13.5%
LSTM	118867.8	0.95	20.1%	925625.1	-4.46	356.3%	195659.9	0.74	23.82%
kmeans	66831.4	0.97	12.1%	90647.1	0.95	13.9%	84785.2	0.96	14.0%

6. Discussion

(1) Based on your experiments, are the results and observed behaviors what you expect?

For the performance part (R squared), the results actually out of my expect. Random forest is better than my expect, and k-means and LSTM is different from what I expect. I'm really surprised that random forest can get such a high R squared score (0.99), while LSTM (0.93) is lower than other methods. I thought LSTM is RNN, so it can give a better prediction, but it doesn't.

The most surprising thing is the result of k-means. I spent lots of time thinking about how to solve the regression problem such as crowd flow prediction using unsupervised learning methods. I thought these methods are more suitable for classification problems.

I didn't know how to do, so I tried to use the average crowd flow in each cluster to be the predicted number. I thought this is a bad idea because it can only give a rough estimation but not a precise number. Nevertheless, k-means can give a nice performance (R squared = 0.96), and LSTM perform the worst.

And when I tried to add more data to train the model, all the performance decrease. This also surprised me. When we look at the crowd flow data, it is clear that the crowd flow decreases or increases due to COVID-19 and more schedule per week, so I think adding these number can contribute to the prediction. In spite of that, the experiment shows that these data cannot improve the performance but make it worse.

The behavior about how the data is sorted is same as my expect. When the data is sorted by date, the neighbor of each data is other station's crowd flow in the same date (ex: 2024-01 南港、2024-01 台北、2024-01 板橋…). LSTM is hard to detect the relationship between data and time, so it'll have a worse performance. On the other hand, if the data is sorted by station, same station is group together (ex: 2024-01 南港、2024-02 南港、2024-03 南港…). LSTM can better capture the information of time, and therefore give a better prediction. By contrast, random forest and k-means group the data based on all features. The time is not so importance for the models. Hence, the order of the training data will not affect so much like LSTM.

(2) Discuss factors that affect the performance, including dataset characteristics.

Data order and amount: the data has temporal dependencies, so its order will affect the performance, especially in LSTM. If the data is sorted in time sequence, the model can perform better. Besides, if there's more data, the model can observe more relationship between data, and therefore produce a better prediction.

Station id encoding (and other preprocessing): using ordinal encoding or one hot encoding can of course represent the stations. However, using the average crowd flow of each station can better indicate the station's feature. It suggests the station's "popularity". The more popular is the station, the more people go to that station. As the result, it can give a better prediction.

External factor: the models are build using historical data. If there are new factors that affect the crowd flow, the prediction may not be correct. For example, new station, new festival, new transportation, price increase. All of these may change the crowd flow and affect the performance.

Parameters (epoch, time window size, k): we should find suitable parameters. If the epoch

is too large or too small, it may overfit or underfit. If the time window for LSTM is not suitable, the model cannot capture enough feature between data and time. In k-means, if k is small, it can only give a rough estimation. On the other hand, if k is large, it will cause a huge computation cost.

Model complexity: if the dataset is not too large, or the features are simple, a simple model can capture all the feature and give a good performance. In contrast, if the data is complex, may we can consider a more complex model to capture all the features. For example, a neural network with more units and layers, or combine multiple models together to predict the result.

(3) Describe experiments that you would do if there were more time available.

I want to adapt more factors that may affect the crowd flow in the dataset, such as rain fall, temperature, or air quality. I don't select them in my dataset because I "think" they won't affect the result too much, but it just my guess. In addition, earth quake (many earth quakes in this winter vacation), public security (killing people on the train), and other many reasons are also factors I didn't consider when I construct the dataset. If I have more time, maybe I can find more factors to improve the prediction.

The second thing I want to try is to find more detailed data. The current data is the crowd flow per month, but I actually want to predict the crowd flow in each day or each hour is even better. If I can predict the crowd flow in a smaller unit, I can use the model to predict what time does the station has fewer people, so I can take the train at that time. I did find such data (crowd flow per hour) for Taipei MRT, but not for THSR. If I have more time, I will try to find if there's such data for THSR.

(4) Indicate what you have learned from the experiments as well as your remaining questions.

First, I learned how to construct my dataset, including collecting data, cleaning, etc. In the pervious courses, like intro to AI and intro to ML, we always get the data that can be used directly. When we did our project, we also find datasets on the internet. We had never start training model from constructing dataset. In this project, I realized how many things I have to consider when I am collecting data. How many and what kind of data do I need? Will this factor affect the crowd flow? Where can I find this data? There are so many things to think about when collecting data.

Secondly, I knew more about time series data. I learned a model called LSTM, which is good at handling data with time dependencies. When I implementing it, I clearly saw how

the size of the time window helps the model to learn and affects the result. Also, CNN can also be used to deal with the task. I thought CNN is used to process and make prediction from images, but I found that it can also be used in time series data. In addition, I found that k-means can also give a good performance on predicting the crowd flow. This is really out of my expect.

Last but not least, I learned how to demonstrate the experiment result. In the pervious homework, we only had to past the terminal screen shot (such as accuracy or loss) in the report or the figure that TAs assigned. I knew these metrics' meaning and why we use them to evaluate the performance. I also knew plotting the result can let us have a better feeling to the number and result. Nevertheless, I just follow the spec and do what TAs tell me to do. However, in this project, I have to learn how to demonstrate the result. What metric and figure should I use? What should be my baseline? How to explain these numbers? That is, how to write a good report, but not just pasting screen shots.

All in all, I learn many things from this project, but I still have some questions. I am curious about the performance of the three model. I thought LSTM is deep learning method, so it can perform better than other two methods. However, it turns out that LSTM has the worst result among the three methods. Why will this happen? Is it because the model is too simple? Maybe I should build a more complex model. In addition, I am really confused about why adding more features cannot help the model and even decrease the performance. In the crowd flow data, we can see a huge drop in 2022-05, when there's lots of people diagnose with COVID-19. And other evidences also show number of people diagnose with COVID-19 and number of train schedule per week will affect the crowd flow. So I really cannot understand why including these two data in the dataset decrease the performance.

7. Reference

- Taiwan High Speed Rail https://www.thsrc.com.tw/corp
- Government open data https://data.gov.tw/
- Construct a machine learning model for predicting Taiwan Taoyuan International Airport Access MRT passenger flow https://ndltd.ncl.edu.tw
- Prediction of Passenger Flow Based on CNN-LSTM Hybrid Model https://ieeexplore.ieee.org
- 機器學習預測北捷人流--隨機森林 https://harrychengz.medium.com
- Deep learning Architecture for Short-term Passenger Flow Forecasting in Urban Rail Transit https://arxiv.org/pdf/1912.12563

8. Appendix: program code

data.py

```
# get raw data from website and save as csv
import requests
import pandas as pd
from lxml import etree
import numpy as np

station_name = ['南港', '台北', '板橋', '桃園', '新竹', '苗栗', '台中', '彰化', '雲林', '嘉義', '台南', '左營']

url = 'https://www.thsrc.com.tw/corp/9571df11-8524-4935-8a46-0d5a72e6bc7c'
req = requests.get(url)
if(req.status_code == 200):
    # get content
    html = etree.HTML(req.text)
    passenger = html.xpath('//table[@id="fixTable"]//td/text()')
    month = html.xpath('//table[@id="fixTable"]//th/text()')

    num = []
    date = []
    station_id = []
    schedule_n = []     # 高鐵每周總班次
    j = 14
    id = 0
    k = 4
    for i in range(len(passenger)):
        # from 2022-1 to 2025-1
        if (month[j] == '2021-12'):
            break
        # only get each station's data, no total
        if(i + 1) % 13 == 0:
            j += 1
            id = 0
            continue
```

```
        # number of schedule
        if(k == 0):
            schedule_n.append(1016)
        elif(k == 1):
            schedule_n.append(1025)
            if(month[j] == '2023-06'):
                k -= 1
        elif(k == 2):
            schedule_n.append(1039)
            if(month[j] == '2023-09'):
                k -= 1
        elif(k == 3):
            schedule_n.append(1060)
            if(month[j] == '2023-12'):
                k -= 1
        else:
            schedule_n.append(1103)
            if(month[j] == '2024-06'):
                k -= 1

        tmp = int(passenger[i].replace(",", ""))
        num.append(tmp)
        date.append(month[j])
        station_id.append(station_name[id])
        id += 1

else:
    print('request error!')

# save as csv
data = list(zip(date, station_id, num, schedule_n))
df = pd.DataFrame(data, columns=['date', 'station id', 'crowd number',
'schedule number'])
df.to_csv('thsr_exp.csv', index=False)
df = pd.read_csv('thsr_exp.csv')
df['date'] = pd.to_datetime(df['date'], format='%Y-
%m').dt.strftime('%Y%m').astype(int)
```

```
# COVID-19
cov = pd.read_csv('19CoV.csv')
cov['date'] = pd.to_datetime(cov['date']).dt.strftime('%Y%m').astype(int)
cov_sum = cov.groupby(['date'])['disease number'].sum().reset_index()
new_row = [
    {'date': 202310, 'disease number': 0},
    {'date': 202402, 'disease number': 0},
    {'date': 202406, 'disease number': 0},
    {'date': 202407, 'disease number': 0},
    {'date': 202408, 'disease number': 0},
    {'date': 202409, 'disease number': 0},
    {'date': 202411, 'disease number': 0},
    {'date': 202412, 'disease number': 0},
    {'date': 202501, 'disease number': 0}
]
for r in new_row:
    cov_sum.loc[len(cov_sum)] = r
df = pd.merge(df, cov_sum[['date', 'disease number']], on='date',
how='left')
df.to_csv('thsr_exp.csv', index=False)
```

utils.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def Data_preprocess():
    data = pd.read_csv('thsr_raw0.csv')
    data['date'] = pd.to_datetime(data['date'], format='%Y-
%m').dt.strftime('%Y%m').astype(int)

    # 疏運期間天數
    special = {
        202201: 8, 202202: 11, 202203: 1, 202204: 8, 202205: 7,
        202206: 5, 202207: 0, 202208: 0, 202209: 5, 202210: 5,
        202211: 0, 202212: 2, 202301: 16, 202302: 5, 202303: 2,
        202304: 9, 202305: 6, 202306: 6, 202307: 0, 202308: 0,
        202309: 4, 202310: 8, 202311: 0, 202312: 3, 202401: 2,
        202402: 10, 202403: 0, 202404: 6, 202405: 4, 202406: 5,
        202407: 0, 202408: 0, 202409: 6, 202410: 6, 202411: 0,
        202412: 0, 202501: 9
    }
    data['special days'] = data['date'].map(special)

    # split data
    # train data = 2022-01 - 2023-12
    # validate data = 2024-01 - 2024-06
    # test data = 2024-07 - 2025-01
    train_data = data[data['date'] <= 202312].reset_index(drop=True)
    validate_data = data[(data['date'] >= 202401) & (data['date'] <=
202406)].reset_index(drop=True)
    test_data = data[data['date'] >= 202407].reset_index(drop=True)

    # 車站編號 = 每月平均人數 (from train data)
    station_avg = train_data.groupby(['station id'])['crowd
number'].mean().reset_index()
    station_avg.columns = ['station id', 'avg']
```

```
    train_data = pd.merge(train_data, station_avg, on='station id',
how='left')
    train_data['station id'] = train_data['avg']
    train_data.drop(columns=['avg'], inplace=True)

    validate_data = pd.merge(validate_data, station_avg, on='station id',
how='left')
    validate_data['station id'] = validate_data['avg']
    validate_data.drop(columns=['avg'], inplace=True)

    test_data = pd.merge(test_data, station_avg, on='station id',
how='left')
    test_data['station id'] = test_data['avg']
    test_data.drop(columns=['avg'], inplace=True)

    # record avg -> station name mapping
    mp = dict(zip(station_avg['station id'], station_avg['avg']))
    avg_to_name = {v:k for k, v in mp.items()}

    return train_data, validate_data, test_data, avg_to_name

# Data_preprocess()
def Data_analyze():
    data = pd.read_csv('thsr_exp.csv')
    # data['date'] = pd.to_datetime(data['date'], format='%Y-
%m').dt.strftime('%Y%m').astype(int)
    mp = {'南港': 0, '台中': 1, '台北': 2, '台南': 3, '嘉義': 4, '左營': 5,
'彰化': 6, '新竹': 7, '板橋': 8, '桃園': 9, '苗栗': 10, '雲林': 11}
    data['station id'] = {v:k for k,v in mp.items()}

    plt.rc('font', family='Microsoft JhengHei')     # show chinese in plt

    # different month's crowd flow
    month_sum = data.groupby(['date'])['crowd number'].sum().reset_index()
    plt.figure(1)
    plt.plot(month_sum['crowd number'])
    plt.xticks(range(len(month_sum['date'])), month_sum['date'],
rotation=45)
```

```
    plt.title("crowd flow in different month")
    plt.show()

    # different station crowd flow
    station_sum = data.groupby(['station id'])['crowd
number'].sum().reset_index()
    plt.figure(2)
    plt.plot(station_sum['crowd number'])
    plt.xticks(range(len(station_sum['station id'])), station_sum['station
id'])
    plt.title("crowd flow in different station")
    plt.show()

def Covid(data):
    cov = pd.read_csv('19CoV.csv')
    cov['date'] =
pd.to_datetime(cov['date']).dt.strftime('%Y%m').astype(int)
    cov_sum = cov.groupby(['date'])['disease number'].sum().reset_index()
    new_row = [
        {'date': 202310, 'disease number': 0},
        {'date': 202402, 'disease number': 0},
        {'date': 202406, 'disease number': 0},
        {'date': 202407, 'disease number': 0},
        {'date': 202408, 'disease number': 0},
        {'date': 202409, 'disease number': 0},
        {'date': 202411, 'disease number': 0},
        {'date': 202412, 'disease number': 0},
        {'date': 202501, 'disease number': 0}
    ]
    for r in new_row:
        cov_sum.loc[len(cov_sum)] = r
    data = pd.merge(data, cov_sum[['date', 'disease number']], on='date',
how='left')
    return data

# train_data, validate_data, test_data, _ = Data_preprocess()
# Covid(train_data)
```

random_forest.py

```
import utils
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error,
mean_absolute_percentage_error
import math
import statsmodels.api as sm
import pandas as pd
import matplotlib.pyplot as plt

if __name__ == '__main__':
    train_data, validate_data, test_data, avg_to_name =
utils.Data_preprocess() # data type =  pd.df
    # uncommand to use covid data
    # train_data = utils.Covid(train_data)
    # validate_data = utils.Covid(validate_data)
    # test_data = utils.Covid(test_data)

    # uncommand to sort data
    # train_data = train_data.sort_values(by=['station id', 'date'])
    # validate_data = validate_data.sort_values(by=['station id', 'date'])
    # test_data = test_data.sort_values(by=['station id', 'date'])

    train_y = train_data['crowd number']
    train_x = train_data.drop(columns=['crowd number'])
    val_y = validate_data['crowd number']
    val_x = validate_data.drop(columns=['crowd number'])
    test_y = test_data['crowd number']
    test_x = test_data.drop(columns=['crowd number'])
    baseline = test_data['station id']      # baseline = avg of each
station

    train_x = np.array(train_x)
    train_y = np.array(train_y)
    val_x = np.array(val_x)
    val_y = np.array(val_y)
    test_x = np.array(test_x)
```

```
    test_y = np.array(test_y)
    baseline = np.array(baseline)

    # train
    print("start training...")
    model = RandomForestRegressor()
    model.fit(train_x, train_y)

    # pred
    print("pred...")
    pred = model.predict(test_x)

    # MSE / RMSE
    mse = mean_squared_error(test_y, pred)
    rmse = math.sqrt(mse)
    mse_base = mean_squared_error(baseline, pred)
    rmse_base = math.sqrt(mse_base)
    print("\n### random forest result ###")
    print("MSE / RMSE")
    print(f"model: MSE = {mse}, RMSE = {rmse}")
    print(f"baseline: MSE = {mse_base}, RMSE = {rmse_base}")
    print(f"improvement(base line rmse - predict rmse): {rmse_base -
rmse}")

    # R square
    r_square = model.score(test_x, test_y)
    test_np = np.array(baseline)
    tot_mean = np.mean(baseline)
    ss_tot = np.sum((baseline - tot_mean) ** 2)
    ss_res = np.sum((baseline - pred) ** 2)
    r_base = 1 - (ss_res / ss_tot)
    print("\nR square")
    print(f"model: R square = {r_square}")
    print(f"baseline: R square = {r_base}")
    print(f"improvement(predict - base line): {r_square - r_base}")

    # MAPE
    mape = mean_absolute_percentage_error(test_y, pred)
```

```
    mape_base = mean_absolute_percentage_error(baseline, pred)
    print("\nMAPE")
    print(f"model: MAPE = {mape}")
    print(f"baseline: MAPE = {mape_base}")
    print(f"improvement(base line - predict): {mape_base - mape}")
    # raise NotImplementedError()

    # save result to csv & visualize
    test_date = test_x[:, 0]
    data = list(zip(pd.Series(test_date), pd.Series(test_x[:,
1]).map(avg_to_name), pred, test_y))
    # data = list(zip(pred, test_y))
    df = pd.DataFrame(data, columns=['date', 'station', 'predict',
'actual'])
    df.to_csv('random forest result.csv', index=False)

    plt.rc('font', family='Microsoft JhengHei')     # show chinese in plt

    # 畫全部
    plt.figure(1)
    test_station = pd.Series(test_x[:, 1]).map(avg_to_name)
    draw_x = df
    test_station = np.array(test_station)
    draw_x = np.array(draw_x)
    plt.plot(draw_x[:, 3], label='actual', color='blue', alpha=0.8)
    # plt.scatter(range(len(draw_x[:, 2])), draw_x[:, 2], label='predict',
color='red', s=10)
    plt.plot(draw_x[:, 2], label='predict', color='red', alpha=0.8)
    # tic_range = range(3, len(test_station), 7)
    # tic_label = test_station[tic_range]
    # plt.xticks(tic_range, tic_label)
    t = np.array(df['date'])
    t = np.round(t/100, 2)
    tic_range = range(6, len(t), 12)
    tic_label = t[tic_range]
    plt.xticks(tic_range, tic_label)
    # plt.xticks(range(len(test_station)), test_station, rotation=90)
    plt.title("random forest - all station 2024-07 ~ 2025-01")
```

```
    plt.xlabel("station")
    plt.ylabel("crowd number")
    plt.legend()
    plt.savefig('random forest result.png')
    plt.show()

    # 只畫新竹
    plt.figure(2)
    draw_x = df[df['station'] == '新竹']
    draw_x = np.array(draw_x)
    plt.plot(draw_x[:, 3], label='actual', color='blue', alpha=0.6)
    # plt.plot(draw_x[:, 2], label='predict', color='red', alpha=0.6)
    plt.scatter(range(len(draw_x[:, 2])), draw_x[:, 2], s=10, color='red',
label='predict')
    plt.xticks(range(len(draw_x[:, 0])), draw_x[:, 0])   # date as x label
    plt.ylim(300000, 1000000)
    plt.title("random forest - Hsinchu station 2024-07 ~ 2025-01")
    plt.xlabel("date")
    plt.ylabel("crowd number")
    plt.legend()
    plt.savefig('random forest result (新竹).png')
    plt.show()
```

k_means.py

```
import os
os.environ["OMP_NUM_THREADS"] = "2"     # avoid warning: KMeans has a mem
leak

import utils
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error,
mean_absolute_percentage_error, confusion_matrix
import matplotlib.pyplot as plt

if __name__ == "__main__":
    k = 100
    train_data, validate_data, test_data, avg_to_name =
utils.Data_preprocess() # data type =  pd.df
    # uncommand to use covid data
    # train_data = utils.Covid(train_data)
    # validate_data = utils.Covid(validate_data)
    # test_data = utils.Covid(test_data)

    # uncommand to sort data
    # train_data = train_data.sort_values(by=['station id', 'date'])
    # validate_data = validate_data.sort_values(by=['station id', 'date'])
    # test_data = test_data.sort_values(by=['station id', 'date'])

    # train
    train_x = train_data.drop(columns=['crowd number'])
    scaler = StandardScaler()
    train_x_scale = scaler.fit_transform(train_x)

    kmeans = KMeans(n_clusters=k)
    train_data['station group'] = kmeans.fit_predict(train_x_scale)

    group_mean = train_data.groupby('station group')['crowd
number'].mean().reset_index()
```

```
    # pred
    fs = scaler.transform(test_data[["date", "station id", "special
days"]])
    test_data['station group'] = kmeans.predict(fs)
    pred = test_data['station group'].map(group_mean.set_index('station
group')['crowd number'])
    print(test_data)

    # save result to csv
    test_data['station id'] = test_data['station id'].map(avg_to_name)
    data = list(zip(test_data['date'], test_data['station id'], pred,
test_data['crowd number']))
    df = pd.DataFrame(data, columns=['date', 'station', 'predict',
'actual'])
    df.to_csv('k means result.csv', index=False)

    # MSE / RMSE
    mse = mean_squared_error(test_data['crowd number'], pred)
    rmse = np.sqrt(mse)
    mse_base = 14129571424.870806   # from random forest
    rmse_base = 118867.87381319987  # from random forest
    print(f"\n### k means result (k = {k}) ###")
    print("MSE / RMSE")
    print(f"model: MSE = {mse}, RMSE = {rmse}")
    print(f"baseline: MSE = {mse_base}, RMSE = {rmse_base}")
    print(f"improvement(base line rmse - predict rmse): {rmse_base -
rmse}")

    # R square
    test_np = np.array(test_data)
    tot_mean = np.mean(test_np[:, 2])
    ss_tot = np.sum((test_np[:, 2] - tot_mean) ** 2)
    ss_res = np.sum((test_np[:, 2] - pred) ** 2)
    r_square = 1 - (ss_res / ss_tot)
    r_base = 0.8668243182542058
    print("\nR square")
    print(f"model: R square = {r_square}")
    print(f"baseline: R square = {r_base}")
```

```
    print(f"improvement(predict - base line): {r_square - r_base}")

    # MAPE
    mape = mean_absolute_percentage_error(test_data['crowd number'], pred)
    mape_base = 0.21791360470522392
    print("\nMAPE")
    print(f"model: MAPE = {mape}")
    print(f"baseline: MAPE = {mape_base}")
    print(f"improvement(base line - predict): {mape_base - mape}")
    # raise NotImplementedError()

    # visualize
    plt.rc('font', family='Microsoft JhengHei')     # show chinese in plt

    # all station (line)
    plt.figure(1)
    plt.plot(np.array(test_data['crowd number']), label='actual',
color='blue', alpha=0.8)
    # plt.scatter(range(len(pred)), pred, label='predict', color='red')
    plt.plot(np.array(pred), label='predict', color='red', alpha=0.8)
    # tic_range = range(3, len(test_data['station id']), 7)
    # tic_label = test_data['station id'][0:len(test_data['station
id'])].iloc[tic_range]
    # plt.xticks(tic_range, tic_label)
    t = np.array(df['date'])
    t = np.round(t/100, 2)
    tic_range = range(6, len(t), 12)
    tic_label = t[tic_range]
    plt.xticks(tic_range, tic_label)
    plt.title("k means (k = 150) - all station 2024-07 ~ 2025-01")
    plt.xlabel('station')
    plt.ylabel('crowd number')
    plt.legend()
    plt.savefig('k means result.png')
    plt.show()

    # hsinchu station (point)
    plt.figure(2)
```

```
    draw_x = df[df['station'] == '新竹']
    draw_x = np.array(draw_x)
    plt.plot(draw_x[:, 3], label='actual', color='blue', alpha=0.6)
    plt.scatter(range(len(draw_x[:, 2])), draw_x[:, 2], s=10, color='red',
label='predict')
    plt.xticks(range(len(draw_x[:, 0])), draw_x[:, 0])   # date as x label
    plt.ylim(300000, 1000000)
    plt.title("k means - Hsinchu station 2024-07 ~ 2025-01")
    plt.xlabel("date")
    plt.ylabel("crowd number")
    plt.legend()
    plt.savefig('k means result (新竹).png')
    plt.show()
```

lstm.py

```
import utils
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Input
from sklearn.metrics import mean_squared_error,
mean_absolute_percentage_error
import matplotlib.pyplot as plt

if __name__ == "__main__":
    seq_sz = 5
    ep = 100
    bs = 32
    train_data, validate_data, test_data, avg_to_name =
utils.Data_preprocess() # data type =  pd.df
    train_data = train_data.sort_values(by=['station id', 'date'])
    validate_data = validate_data.sort_values(by=['station id', 'date'])
    test_data = test_data.sort_values(by=['station id', 'date'])

    train_y = train_data['crowd number']
    train_x = train_data.drop(columns=['crowd number'])
    val_y = validate_data['crowd number']
    val_x = validate_data.drop(columns=['crowd number'])
    test_y = test_data['crowd number']
    test_x = test_data.drop(columns=['crowd number'])
    baseline = test_data['station id']

    train_x = np.array(train_x)
    train_y = np.array(train_y)
    val_x = np.array(val_x)
    val_y = np.array(val_y)
    test_x = np.array(test_x)
    test_y = np.array(test_y)
    baseline = np.array(baseline)

    scaler_x = StandardScaler()
```

```
    train_x_scale = scaler_x.fit_transform(train_x)
    val_x_scale = scaler_x.transform(val_x)
    test_x_scale = scaler_x.transform(test_x)

    scaler_y = StandardScaler()
    train_y_scale = scaler_y.fit_transform(train_y.reshape(-1, 1))
    val_y_scale = scaler_y.transform(val_y.reshape(-1, 1))
    test_y_scale = scaler_y.transform(test_y.reshape(-1, 1))

    # time window
    def creat_seq(x, y, sz=3):
        seq_x, seq_y = [], []
        for i in range(len(x) - sz):
            seq_x.append(x[i:i+sz])
            seq_y.append(y[i+sz])
        return np.array(seq_x), np.array(seq_y)

    # prepare for LSTM
    train_x_seq, train_y_seq = creat_seq(train_x_scale, train_y_scale,
seq_sz)
    val_x_seq, val_y_seq = creat_seq(val_x_scale, val_y_scale, seq_sz)
    test_x_seq, _ = creat_seq(test_x_scale, test_y_scale, seq_sz)

    train_x_seq = train_x_seq.reshape(train_x_seq.shape[0], seq_sz,
train_x_seq.shape[2])
    val_x_seq = val_x_seq.reshape(val_x_seq.shape[0], seq_sz,
val_x_seq.shape[2])
    test_x_seq = test_x_seq.reshape(test_x_seq.shape[0], seq_sz,
test_x_seq.shape[2])

    # train
    model = Sequential([
        LSTM(128, activation='tanh', return_sequences=True),
        LSTM(128, activation='tanh'),
        Dense(32, activation='relu'),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')
```

```
    model.fit(train_x_seq, train_y_seq, epochs=ep, batch_size=bs,
validation_data=(val_x_seq, val_y_seq), verbose=0)

    # pred
    pred = model.predict(test_x_seq)
    pred = scaler_y.inverse_transform(pred)

    # MSE / RMSE
    # test_y = test_data['crowd number'][seq_sz:]
    test_y = test_y[seq_sz:].reshape(-1, 1)
    mse = mean_squared_error(test_y, pred)
    rmse = np.sqrt(mse)
    mse_base = 14129571424.870806   # from random forest
    rmse_base = 118867.87381319987  # from random forest
    print(f"\n### LSTM result (epoch = {ep}, batch = {bs}, seq_sz =
{seq_sz}) ###")
    print("MSE / RMSE")
    print(f"model: MSE = {mse}, RMSE = {rmse}")
    print(f"baseline: MSE = {mse_base}, RMSE = {rmse_base}")
    print(f"improvement(base line rmse - predict rmse): {rmse_base -
rmse}")

    # R square
    tot_mean = np.mean(test_y)
    ss_tot = np.sum((test_y - tot_mean) ** 2)
    ss_res = np.sum((test_y - pred) ** 2)
    r_square = 1 - (ss_res / ss_tot)
    print("\nR square")
    print(f"model: R square = {r_square}")

    # MAPE
    mape = mean_absolute_percentage_error(test_y, pred)
    mape_base = 0.21791360470522392
    print("\nMAPE")
    print(f"model: MAPE = {mape}")
    print(f"baseline: MAPE = {mape_base}")
    print(f"improvement(base line - predict): {mape_base - mape}")
    # raise NotImplementedError()
```

```
    # save res
    test_data['station id'] = test_data['station id'].map(avg_to_name)
    data = list(zip(test_data['date'], test_data['station id'],
pred.flatten(), test_data['crowd number']))
    df = pd.DataFrame(data, columns=['date', 'station', 'predict',
'actual'])
    df.to_csv(f'LSTM result (seq{seq_sz}, ep{ep}, bs{bs}).csv',
index=False)

    # visualize
    plt.rc('font', family='Microsoft JhengHei')     # show chinese in plt
    # 折線圖 all station
    plt.figure(1)
    plt.plot(np.array(test_y), label='actual', color='blue', alpha=0.8)
    # plt.scatter(range(len(pred)), pred, label='predict', color='red')
    plt.plot(np.array(pred), label='predict', color='red', alpha=0.8)
    # tic_range = range(0, len(test_y), 7)
    # tic_label = test_data['station id'][:len(test_y)].iloc[tic_range]
    # plt.xticks(tic_range, tic_label)
    t = np.array(df['date'])
    t = np.round(t/100, 2)
    tic_range = range(6, len(t), 12)
    tic_label = t[tic_range]
    plt.xticks(tic_range, tic_label)
    plt.title(f"LSTM (seq{seq_sz}, ep{ep}, bs{bs}) - all station 2024-07 ~
2025-01")
    plt.xlabel('station')
    plt.ylabel('crowd number')
    plt.legend()
    plt.savefig('LSTM result.png')
    plt.show()

    # scatter, hsinchu
    plt.figure(2)
    draw_x = df[df['station'] == '新竹']
    draw_x = np.array(draw_x)
    plt.plot(draw_x[:, 3], label='actual', color='blue', alpha=0.6)
```

```
    # plt.plot(draw_x[:, 2], label='predict', color='red', alpha=0.6)
    plt.scatter(range(len(draw_x[:, 2])), draw_x[:, 2], s=10, color='red',
label='predict')
    plt.xticks(range(len(draw_x[:, 0])), draw_x[:, 0])   # date as x label
    plt.ylim(300000, 1000000)
    plt.title("LSTM - Hsinchu station 2024-07 ~ 2025-01")
    plt.xlabel("date")
    plt.ylabel("crowd number")
    plt.legend()
    plt.savefig('LSTM result (新竹).png')
    plt.show()
```