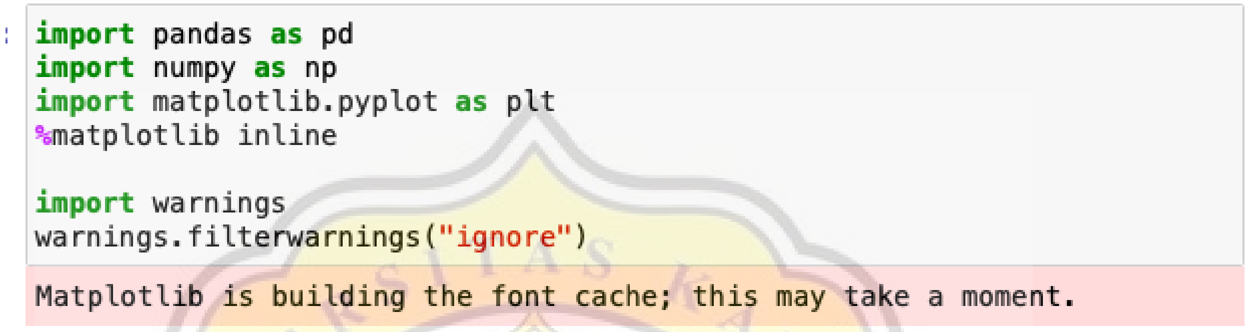
CHAPTER 5
IMPLEMENTATION AND TESTING

5.1. Implementation of formula and Designs



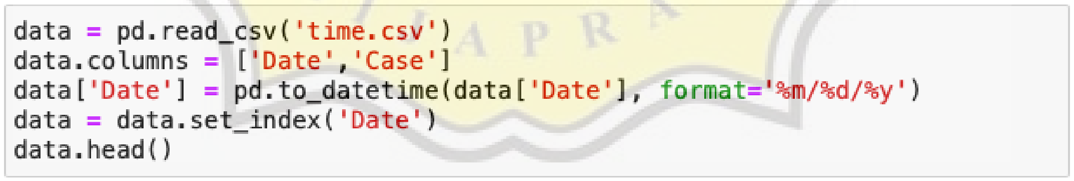
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")
```

```
Matplotlib is building the font cache; this may take a moment.
```

Figure 5-1 Import Libraries

The programming language used in this project is Python as it is a general-purpose programming language. In this project, Python 3.8.8 is used for data analysis and also data visualization with vast choice of libraries that helps the user to code. The libraries used for forecasting are pandas 1.2.4, numpy 1.20.1, and matplotlib.pyplot 3.3.4. This project is done on macOs Catalina Version 10.15.7, with i5 Intel Core Processor, on Anaconda Navigator using Jupyter Notebook for its clean output of integrating code live and visualizations.

Pandas is used to perform data manipulation, analysis and also data operations for numerical tables and time series data. Numpy is the fundamental library for scientific computing, and matplotlib is used for plotting and creates figures.

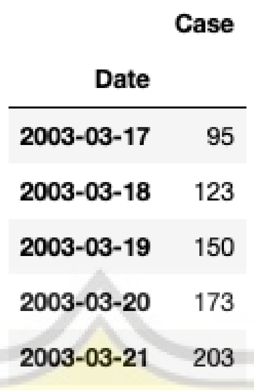


```
data = pd.read_csv('time.csv')
data.columns = ['Date','Case']
data['Date'] = pd.to_datetime(data['Date'], format='%m/%d/%y')
data = data.set_index('Date')
data.head()
```

Figure 5-2 Import data

After importing the necessary libraries, the data is imported and formatted according to need. Pd.read_csv is one of the usage of Panda libraries. The data used are in comma-separated values(csv) format and it is loaded into python by using Pandas. The format is changed to date and formatted as month-day-year.

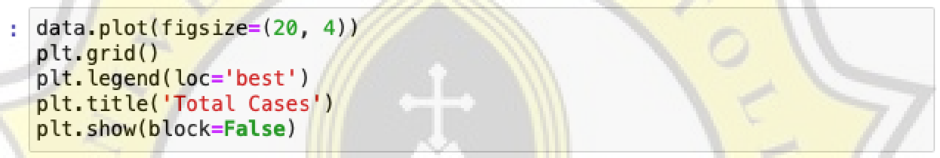
The results are as follow:



	Case
Date	
2003-03-17	95
2003-03-18	123
2003-03-19	150
2003-03-20	173
2003-03-21	203

Figure 5-3 Results of Imported Data

After data is imported and formatted, data is plotted using matplotlib libraries.



```
data.plot(figsize=(20, 4))
plt.grid()
plt.legend(loc='best')
plt.title('Total Cases')
plt.show(block=False)
```

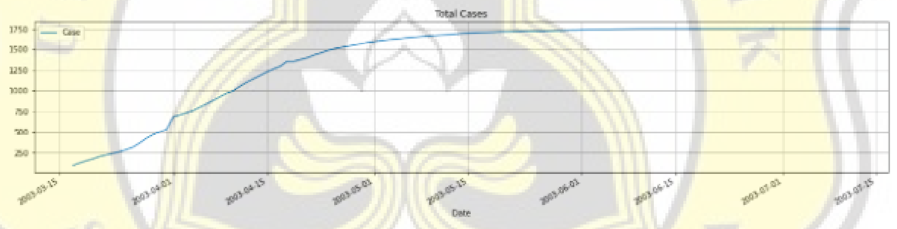
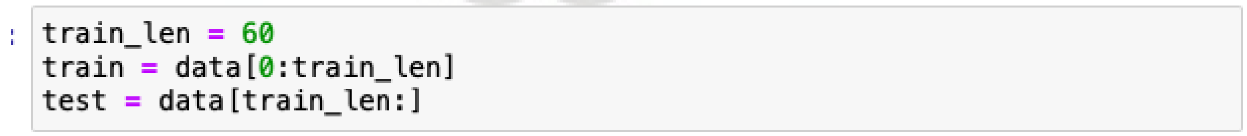


Figure 5-4 Plot Data

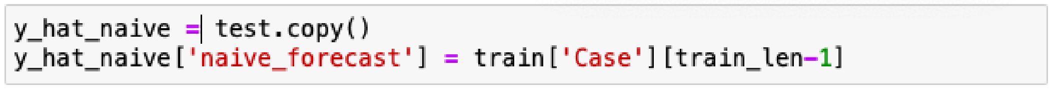
The data is plotted into figure size with 20,4 sizing. The grid () function sets visibility of grid inside the figure to be on or off. Plt.legend is used to specify location of the legend. The figure is titled 'Total Case' and block=False is to ensure the window will not be closed when the script is finished.



```
train_len = 60
train = data[0:train_len]
test = data[train_len:]
```

Figure 5-5 Separating the Data

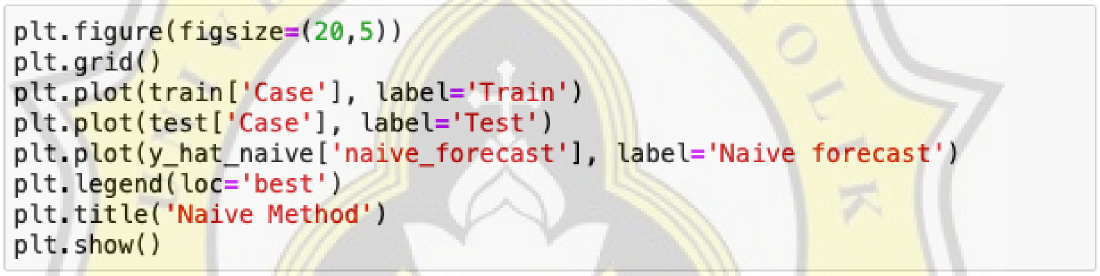
The data is then separated into 2 sections train data and test data. There is a total of 96 entry of data inside the data frame used. From the first day, the infected person is identified until the last day of entry for recovered person. The data is separated as 60 days as training data and the remaining day is used for testing data.



```
y_hat_naive = test.copy()
y_hat_naive['naive_forecast'] = train['Case'][train_len-1]
```

Figure 5-6 Training the data

As the simple naive model is used in this project, it used the previous observation directly as the forecast therefore the code used is [train_len-1]. There are 3 experiment results of this method, using t-3, t-2 and t-1. The time used in these experiment method is to show the reliability of using naïve formulas for time forecasting. Thus, the farthest time used is time minus 3. The results will be compared to show the best time to use this method. Results will be compared by the calculation of MAPE and RMSE.



```
plt.figure(figsize=(20,5))
plt.grid()
plt.plot(train['Case'], label='Train')
plt.plot(test['Case'], label='Test')
plt.plot(y_hat_naive['naive_forecast'], label='Naive forecast')
plt.legend(loc='best')
plt.title('Naive Method')
plt.show()
```

Figure 5-7 Plotting the Results

This is the code to plot the results of the training data, test data and the result of Naïve calculation.

5.2. Method Testing and Data Results

This is the result of t-1:

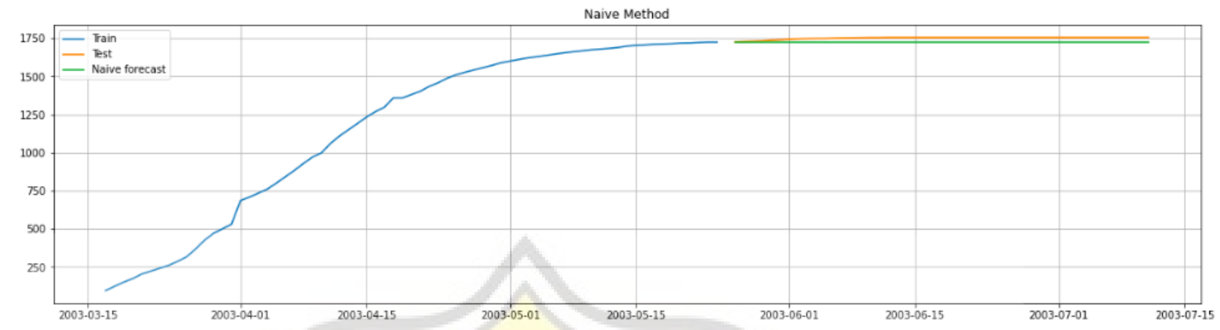
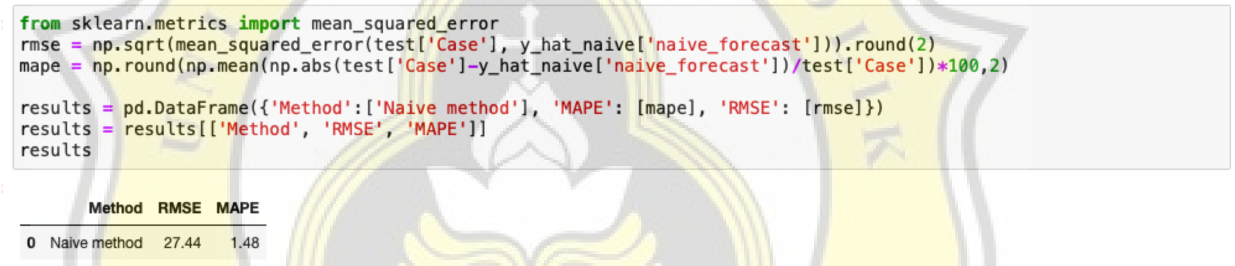


Figure 5-8 Result of t-1

The results of the forecast are not much different from the real data.



```
from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(test['Case'], y_hat_naive['naive_forecast'])).round(2)
mape = np.round(np.mean(np.abs(test['Case']-y_hat_naive['naive_forecast'])/test['Case'])*100,2)

results = pd.DataFrame({'Method':['Naive method'], 'MAPE': [mape], 'RMSE': [rmse]})
results = results[['Method', 'RMSE', 'MAPE']]
results
```

	Method	RMSE	MAPE
0	Naive method	27.44	1.48

Figure 5-9 RMSE and MAPE of t-1

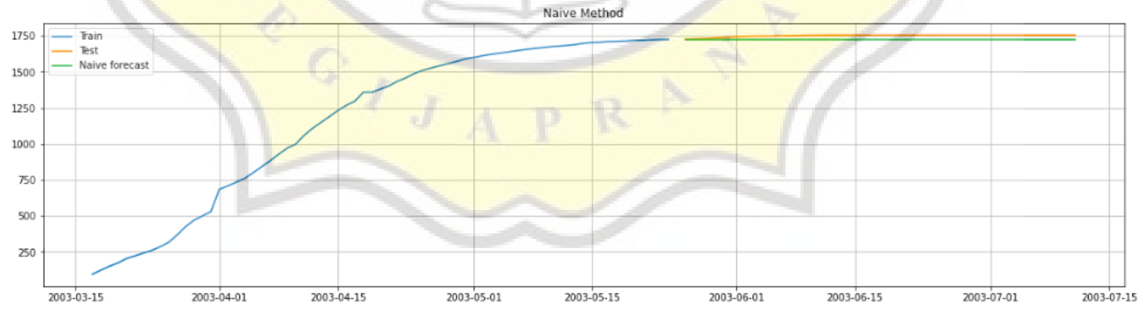


Figure 5-10 Result of t-2

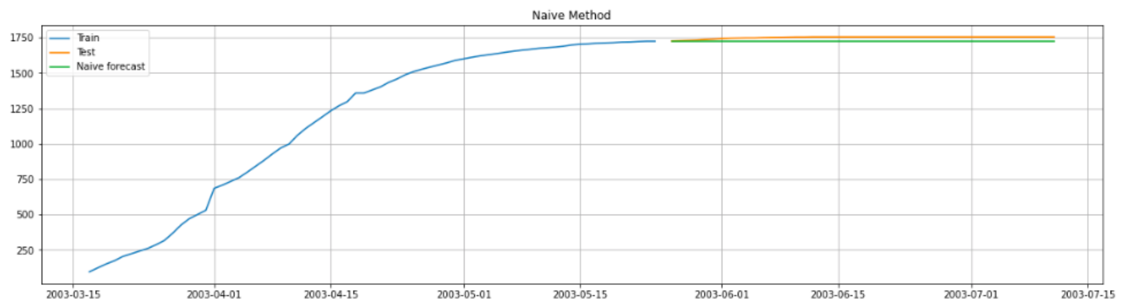


Figure 5-11 Result of t-3

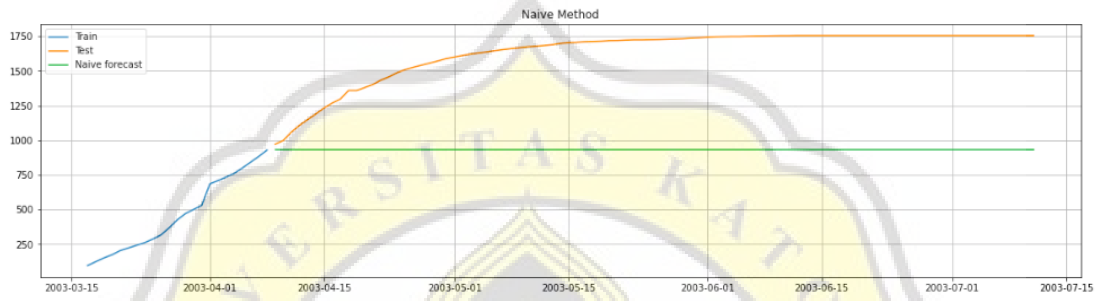
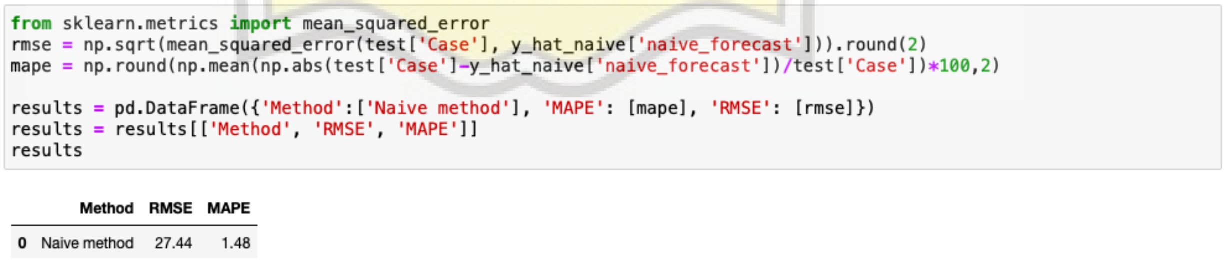


Figure 5-12 Result of train_len= 20, t-1

On figure 5-12, the graphic doesn't show promising result since the train_len is 20. The prediction method only trained on 20 days and have to predict the rest of the days hence the results aren't as acceptable as using 60 days to train the algorithm.

For measuring the performance, the existing data is compared with the prediction models. The difference between the data and the predicted values is known as forecast error and the lesser forecast error, the more accurate the model is. RMSE is Root Mean Squared Error, it's useful for calculating forecast accuracy. As we can see the RMSE is 27.44 and MAPE is 1.48.

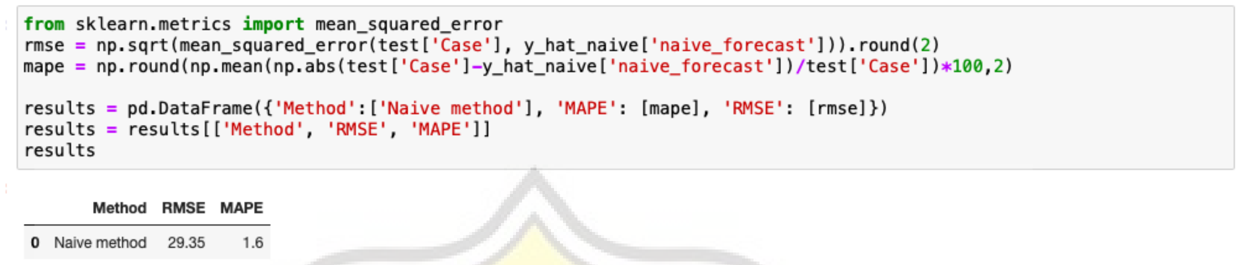


```
from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(test['Case'], y_hat_naive['naive_forecast'])).round(2)
mape = np.round(np.mean(np.abs(test['Case']-y_hat_naive['naive_forecast'])/test['Case'])*100,2)

results = pd.DataFrame({'Method':['Naive method'], 'MAPE': [mape], 'RMSE': [rmse]})
results = results[['Method', 'RMSE', 'MAPE']]
results
```

	Method	RMSE	MAPE
0	Naive method	27.44	1.48

Figure 5-13 RMSE and MAPE of t-2

The results between t-1 and t-2 are identical because the data from t-1 and t-2 is the same so the calculation does not differ.



```
from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(test['Case'], y_hat_naive['naive_forecast'])).round(2)
mape = np.round(np.mean(np.abs(test['Case']-y_hat_naive['naive_forecast'])/test['Case'])*100,2)

results = pd.DataFrame({'Method':['Naive method'], 'MAPE': [mape], 'RMSE': [rmse]})
results = results[['Method', 'RMSE', 'MAPE']]
results
```

	Method	RMSE	MAPE
0	Naive method	29.35	1.6

Figure 5-14 RMSE and MAPE of t-3

The result between t-1, t-2 and t-3 has shown that the bigger (t) used, the more different the prediction accuracies are. The smaller (t) used in the experiment results in a smaller value of RMSE, meaning the differences between the value predicted by the experiment are not much different from the values observed. The bigger (t) used in the experiment has shown a bigger value in RMSE, thus the results aren't much more promising than when the smaller (t) is used.

5.3. RMSE, MAPE and Cross Validation

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} \|y(i) - \hat{y}(i)\|^2}{N}}$$

Equation 5-1 RMSE Equation

$$MAPE = \frac{100}{n} \sum_{t=1}^{n} \frac{|A_t - F_t|}{A_t} \%$$

Equation 5-2 MAPE Equation

As shown on the equation above, RMSE and MAPE have a similarity in some aspect as both measures averaged model prediction error, both can range from 0 to infinity and lower values are essentially better. High values between RMSE and MAPE indicates that the values of predicted data are too different compared to the actual data point and lower values means that predicted values are much more similar with the actual data point.

One of the key differences between RMSE and MAPE are errors. Errors on RMSE are squared root before averaged and RMSE is highly weighted to larger errors while MAPE are highly sensitive to smaller errors.

By using two measurements to predict error, cross-validation between the two measurements are needed to test the result of the experiment. Cross Validation in a time series model is on a rolling basis.

Steps needed in cross validating starts with a small subset of data for the first training purposes and it is used to forecast the later data point and it will be checked for the accuracy of the forecasted data point. After that, the same forecasted data are then included as part of the next training. The example of cross validation by rolling basis are shown on the image below.

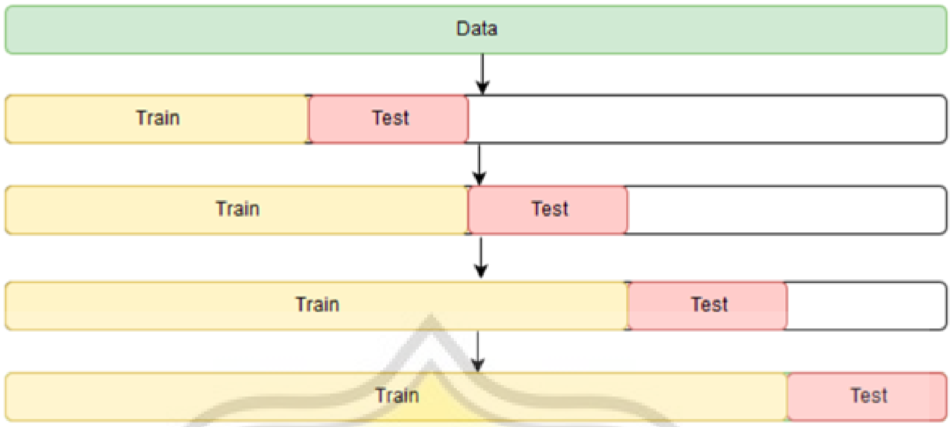


Figure 5-15 Rolling Basis Cross Validation

As shown on the table below, data are divided for the rolling basis cross validation. The first test will use 20 train_len and it will be tested on the next 5 data point, the rest of the data are nulled. Then for the next test, the 5 tests data will be included in the train_len and the next 5 data are tested until the whole data are tested. Every rolling basis test will be tested using t-1, t-2, and t-3 and the results of the value difference are also mentioned on the table below.

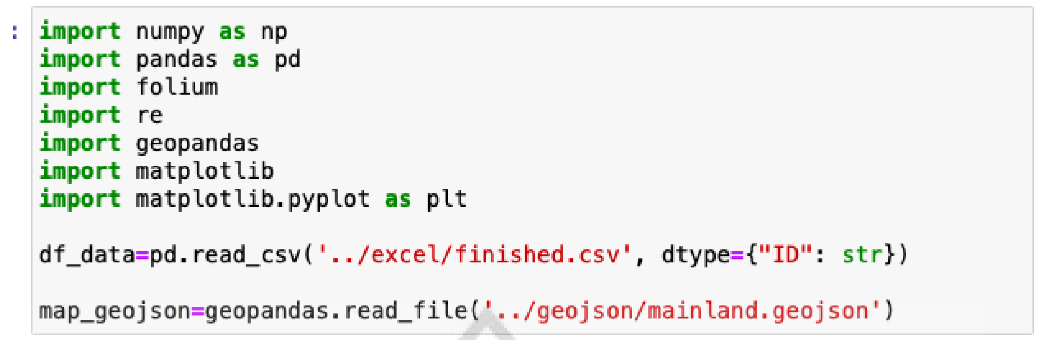
Based on the table of results below, less time used for training the method has shown much greater values in both RMSE and MAPE with RMSE weighing much more than MAPE as it is much sensitive to large errors, furthermore the time(t) variable used in testing the method also affects the results of the test as lesser time(t) used has shown a much more acceptable results between the predicted value and the actual data point, showing the sensitivity of MAPE as MAPE is much more sensitive to smaller errors. In hindsight, greater train_len and smaller (t) such as t-1 variable have better results for the test compared to other combination.

The results listed on the table below can be used to measure acceptable method to forecast a time-series data and these results can be greatly improved by combining other method for forecasting data. Naïve Method used in this test is solely used as a benchmark for forecasting and leaves much room for improvement for other forecasting method.

Table 5-1 Rolling Basis Validation

			t-1		t-2		t-3	
Train _len	Test	Total Data	RMSE	MAPE	RMSE	MAPE	RMSE	MAPE
20	5	25	137.98	10.53	175.43	14.41	217.41	18.56
25	5	30	96.53	6.6	136	9.78	215.82	16
30	5	35	76.19	4.75	118.09	7.74	118.09	7.74
35	5	40	55	3.31	71.4	4.38	92.95	5.78
40	5	45	52.41	2.88	62.66	3.54	73.13	4.21
45	5	50	19.19	1.02	25.67	1.44	33.33	1.91
50	5	55	11.06	0.59	19.69	1.12	25.59	1.47
55	5	60	4.72	0.24	8.5	0.47	12.42	0.71
60	5	65	6.24	0.31	8.06	0.43	8.06	0.43
65	5	70	8.29	0.47	11.28	0.64	15.27	0.87
70	5	75	4.06	0.22	6.04	0.34	6.04	0.34
75	5	80	0	0	0	0	1	0.05
80	5	85	0	0	0	0	0	0
85	5	90	0	0	0	0	0	0
90	5	95	0	0	0	0	0	0

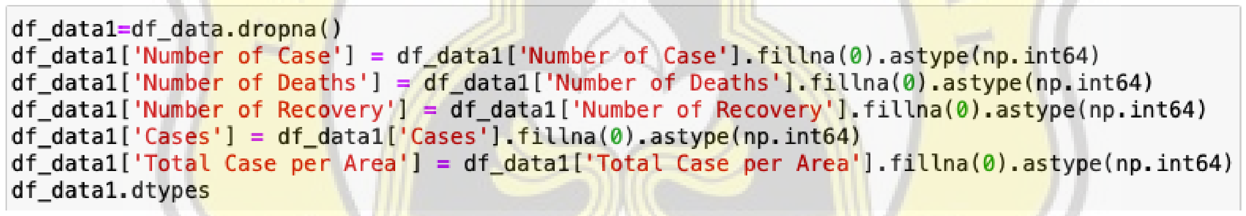
5.4. Implementation of Map Formula and Designs



```
: import numpy as np
  import pandas as pd
  import folium
  import re
  import geopandas
  import matplotlib
  import matplotlib.pyplot as plt

  df_data=pd.read_csv('../excel/finished.csv', dtype={"ID": str})

  map_geojson=geopandas.read_file('../geojson/mainland.geojson')
```

Figure 5-16 Importing Libraries

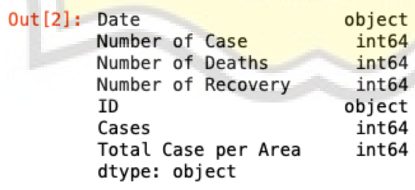
Importing necessaries libraries are the first step to configure the maps. Python have vast choices of libraries that can be accessed and used. After importing libraries, the next step is to import the data and configure the data. The data is processed by pandas and the geojson file is processed by geopandas.



```
df_data1=df_data.dropna()
df_data1['Number of Case'] = df_data1['Number of Case'].fillna(0).astype(np.int64)
df_data1['Number of Deaths'] = df_data1['Number of Deaths'].fillna(0).astype(np.int64)
df_data1['Number of Recovery'] = df_data1['Number of Recovery'].fillna(0).astype(np.int64)
df_data1['Cases'] = df_data1['Cases'].fillna(0).astype(np.int64)
df_data1['Total Case per Area'] = df_data1['Total Case per Area'].fillna(0).astype(np.int64)
df_data1.dtypes
```

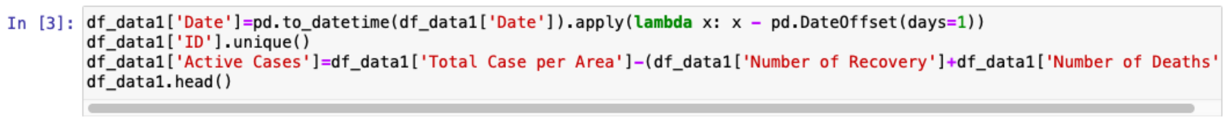
Figure 5-17 Pre-Processing Data

The next step is very crucial which is to preprocess the data to accommodate the map. Any blank data are instantly dropped and the data type which is initially recognized as float are changed to int.



```
Out[2]: Date                   object
        Number of Case          int64
        Number of Deaths        int64
        Number of Recovery      int64
        ID                     object
        Cases                   int64
        Total Case per Area     int64
        dtype: object
```

Figure 5-18 Output of Data Pre-processing

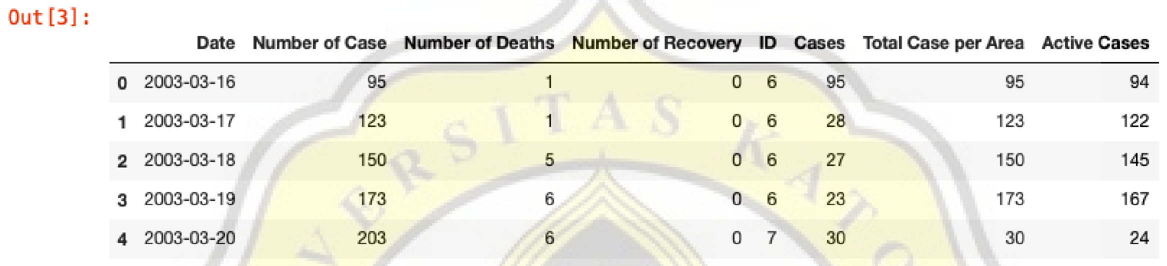


```
In [3]: df_data1['Date']=pd.to_datetime(df_data1['Date']).apply(lambda x: x - pd.DateOffset(days=1))
        df_data1['ID'].unique()
        df_data1['Active Cases']=df_data1['Total Case per Area']-(df_data1['Number of Recovery']+df_data1['Number of Deaths'
        df_data1.head()
```

Figure 5-19 Data Adjustment

Datas collected in the database are the data from the day before and thus accumulated on the day after. Thus, preprocessing data to adjust the active cases are necessary. Firstly, date is imported as object so it needed to be changed to date time, and then the data are put a day before.

Using ID in both geojson and dataframe are crucial to pinpoint the exact location of the cases. So ID is classified as unique, and then to calculate the active cases in each location, Total case per area are subtracted by number of recovery and also number of deaths, and the result are put in a new row as "Active Cases".

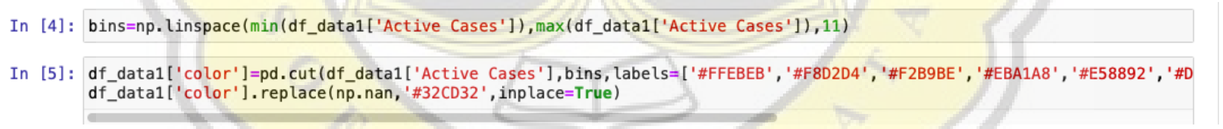


Out[3]:

	Date	Number of Case	Number of Deaths	Number of Recovery	ID	Cases	Total Case per Area	Active Cases
0	2003-03-16	95	1	0	6	95	95	94
1	2003-03-17	123	1	0	6	28	123	122
2	2003-03-18	150	5	0	6	27	150	145
3	2003-03-19	173	6	0	6	23	173	167
4	2003-03-20	203	6	0	7	30	30	24

Figure 5-20 Output of Data Adjustment

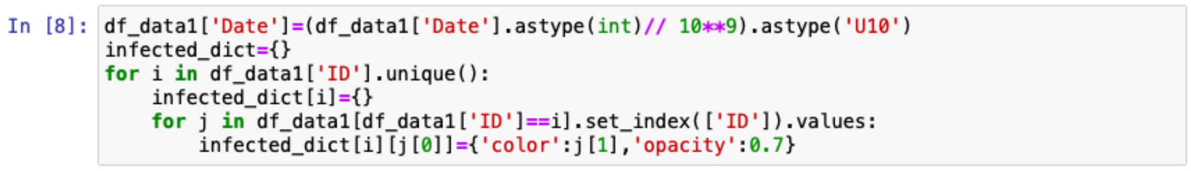
After calculating the sum of active cases in each area, the data is divided to 10 parts with minimal and maximal value to assign each data with different shade of colors. The data colors depend on the sum of active cases in each location, it will get darker shade of colors the higher the sum of data is.



```
In [4]: bins=np.linspace(min(df_data1['Active Cases']),max(df_data1['Active Cases']),11)

In [5]: df_data1['color']=pd.cut(df_data1['Active Cases'],bins,labels=['#FFEBEB','#F8D2D4','#F2B9BE','#EBA1A8','#E58892','#D
        df_data1['color'].replace(np.nan,'#32CD32',inplace=True)
```

Figure 5-21 Assigning Colors

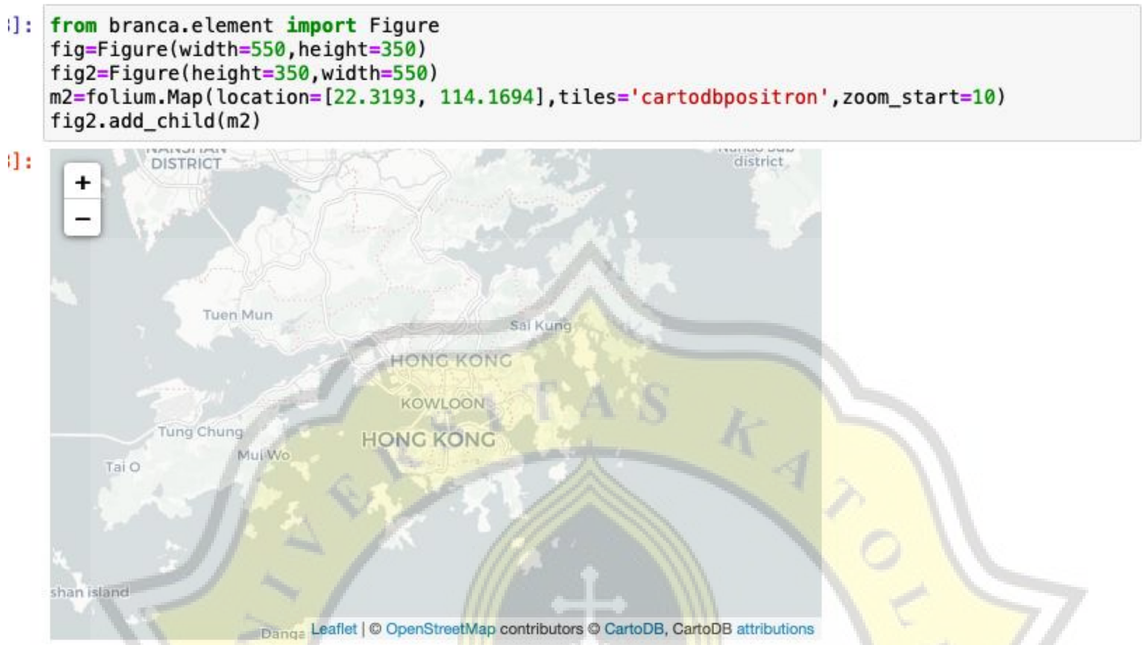
The libraries used in for mapping the time slider map is called TimeSliderChoropleth. This particular library uses times in millisecond and the data used are in the form of an indexed dictionary.



```
In [8]: df_data1['Date']=(df_data1['Date'].astype(int)// 10**9).astype('U10')
        infected_dict={}
        for i in df_data1['ID'].unique():
            infected_dict[i]={}
            for j in df_data1[df_data1['ID']==i].set_index(['ID']).values:
                infected_dict[i][j[0]]={'color':j[1],'opacity':0.7}
```

Figure 5-22 Creating dictionary for TimeSliderChoropleth

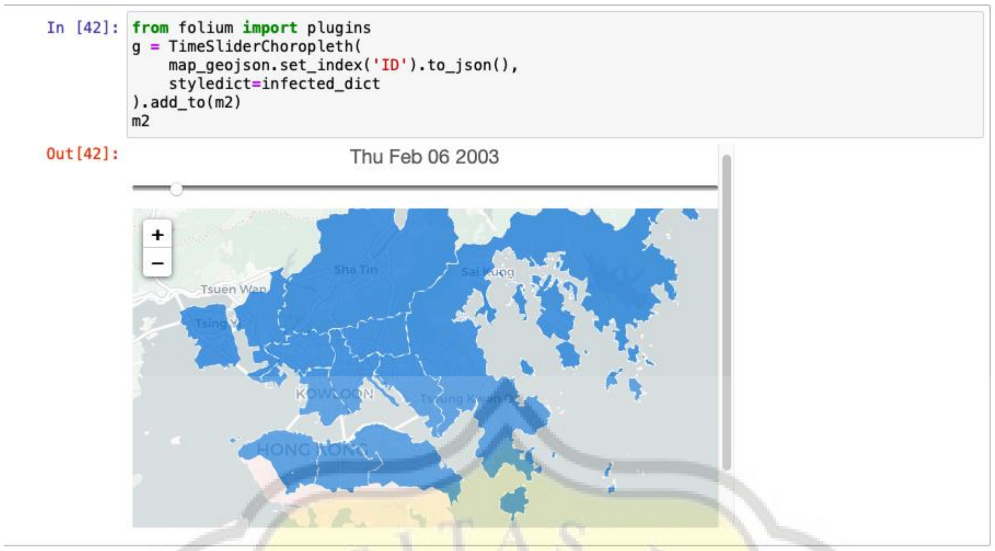
The process of preprocessing data used in this map are completed and the steps after this is to process the geojson data used in this map.



```
from branca.element import Figure
fig=Figure(width=550,height=350)
fig2=Figure(height=350,width=550)
m2=folium.Map(location=[22.3193, 114.1694],tiles='cartodbpositron',zoom_start=10)
fig2.add_child(m2)
```

Figure 5-23 Map Configuration

The steps in this section is to import the plugins for the map, creating the base map and adding the plugin into the map. The map uses Folium as it is easy to visualize data which has been formerly preprocessed in python and it is also interactive. The map style uses CartoDB thus it displays a clean set and the colors of each location will be contrasted. The zoom of the map is also set to prevent the map from over-zooming and clearly displays the whole map of Hong Kong.



```
In [42]: from folium import plugins
         g = TimeSliderChoropleth(
             map_geojson.set_index('ID').to_json(),
             styledict=infected_dict
         ).add_to(m2)
         m2
Out[42]:
```

Figure 5-24 Finished Map

After creating the base map, the plugins are imported and the dictionaries formerly created are imported inside the plugin for the map. The plugins are put inside the map and it overlays on the base map. The day and date are on top of the map along with the slider.

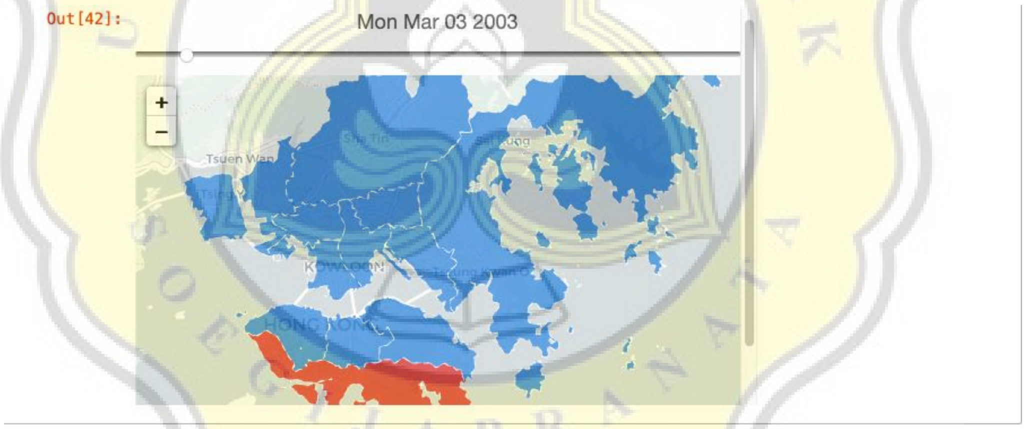


Figure 5-25 Comparison Between Dates and Data

The data from Feb 6 compared to March 3 increased thus the color gets darker to indicate growth of case in one area.

Out[25]:

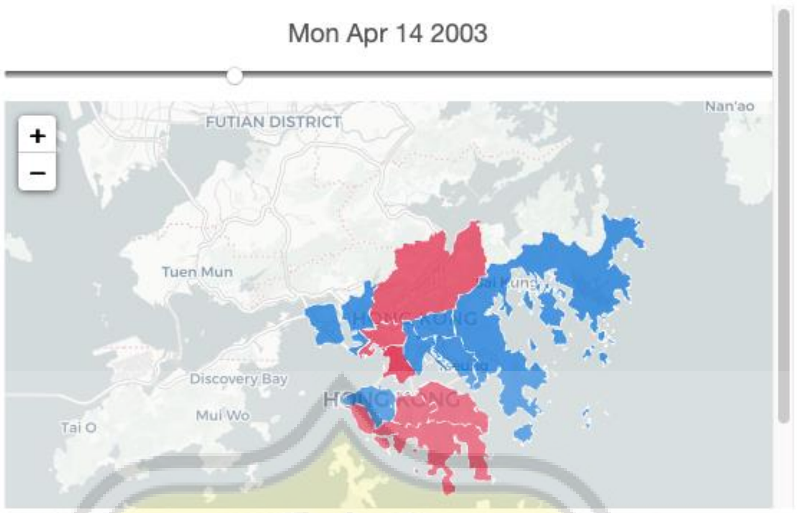


Figure 5-26 Comparison on another Area

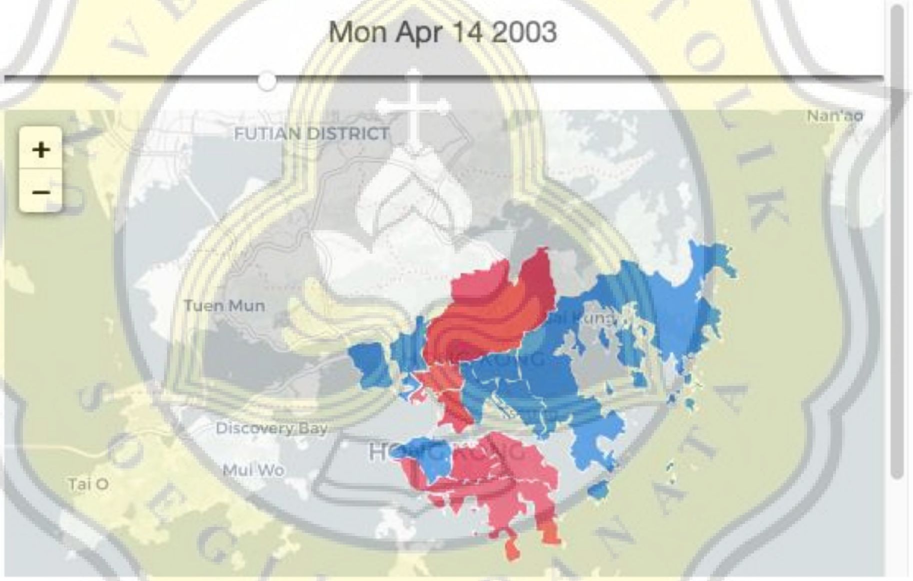


Figure 5-27 Forecasted Map

On April 14, 2003, a string of cases emerged on mainland Hongkong and it affected the area in-land, the color difference showed that where the 2-main area meets, cases number have spiked indicating the travel-point of the virus spread. This maps shows cases per area per state.

On the second image, the forecasted Map shows a spike of cases on the upper larger area of mainland Hong Kong. Mainly because forecast uses the data of time (t) before the stated date(t-1). Using these forecasts to visualize case travels and number of cases can prevent a much larger epidemic outbreak. Both map shows the same area affected yet the number differs, the first map shows the actual data point collected and the second map shows the forecasted data point using time variable (t-1), thus the different gradation of the affected area of the map.

Using the visualization on this map, it can be used as a benchmark to pinpoint which area is affected the most and also the growth rate of the virus furthermore results from the previous forecasting is put inside a .CSV document alongside the base data and integrated into these visualization maps.

This map can be developed even further by combining the travel data such as public transportation, public facilities and areas with high-number of civilian. Maps such as this can be used to prevent outspread of infectious virus in a way by learning how virus spreads and closing immediate area to prevent next outbreak.