Appendix D. Python code

The study considers Python as an application to establish the forecasting model and implement the external parameters such as vaccination rate, covid-variant, and control measures into the final model. The following figures describe the detailed code, explanation of the code is directly provided next to the coding sentences:

1. Import libraries into Python file:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA
from pandas import DataFrame
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import warnings; warnings.filterwarnings(action='once')
import math
```

2. Import excel file, this excel file contains the COVID-19 confirmed cases have been recorded daily from 27th April to 24th November 2021. After which, the data was plotted in order to visualize the distribution of the collected data

```
WB = pd.read_excel("Data import for ADF.xlsx")
# Plot data into a graph:
y = list(WB['Data value'])
x = list(WB['Date'])
plt.title('Recorded confirmed cases time series plot')
plt.plot(x, y)
plt.show()
```

3. The first step in model ARIMA development is considering whether the data is stationary. Hence, an ADF test is provided in the code. From which, d parameters will then be determined based on the stationarity of data in which differencing stage. The COVID-19 confirmed cases in this study is considered to be stationary on the first differencing, which means d = 1 in ARIMA model:

```python
# ADF test
# Considering the stationarity of the original data
X = WB['Data value'].values
result = adfuller(X)
print('The result of ADF test of the original value is:')
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print(key, value)

if result[0] < result[4]["5%"]:
    print("Reject Ho - Time Series is Stationary")
else:
    print("Failed to Reject Ho - Time Series is Non-Stationary")
```

```python
# Considering the 1st differencing
WB['1st_Differencing'] = WB['Data value'] - WB['Data value'].shift(1)
plt.plot(x, WB['1st_Differencing'])
plt.title('First differencing of recorded confirmed cases')
plt.show()
result2 = adfuller(WB['1st_Differencing'].dropna())
# Note we are dropping na values because the first value of the first difference is NA
print('-----------------------------------------------------------------')
print('The result of ADF test of 1st Differencing is:')
print('Critical Values:')
for key, value in result2[4].items():
    print(key, value)
print('ADF Statistic: %f' % result2[0])
print('p-value: %f' % result2[1])
if result2[0] < result2[4]["5%"]:
    print("Reject Ho - Time Series is Stationary")
else:
    print("Failed to Reject Ho - Time Series is Non-Stationary")
```

4. AR(p) and MA(q) were determined using ACF and PACF method, the code is as following:

```
# ACF and PACF method
WB['1st_Differencing'] = np.asarray(WB['1st_Differencing'])
fig, (ax1, ax2) = plt.subplots(1,2,figsize=(16,6), dpi= 80)
ACF_plot = plot_acf(WB['1st_Differencing'].dropna(),ax=ax1, lags=60)
PACF_plot = plot_pacf(WB['1st_Differencing'].dropna(),ax=ax2, lags=60, method='ywm')
# Decoration:
# 1.lighten the borders
ax1.spines["top"].set_alpha(.3); ax2.spines["top"].set_alpha(.3)
ax1.spines["bottom"].set_alpha(.3); ax2.spines["bottom"].set_alpha(.3)
ax1.spines["right"].set_alpha(.3); ax2.spines["right"].set_alpha(.3)
ax1.spines["left"].set_alpha(.3); ax2.spines["left"].set_alpha(.3)
# 2.Font size of tick labels
ax1.tick_params(axis='both', labelsize=12)
ax2.tick_params(axis='both', labelsize=12)
plt.show()

```

5. To implement the external parameters into the model, the study uses Pearson's correlation to determine the relationships between Confirmed cases and external data defined. However, since the correlation can easily be calculated through excel, the following code only considers when the Beta parameters of external data have been defined and implement these parameters into the ARIMA model for simplicity:

```
# Implementing external parameters, the coefficients were performed in excel file
WB2 = pd.read_excel("External Parameters.xlsx")
Vaccination = WB2['Vaccination coverage (PPM)'].values
Social_distance = WB2['Social distancing'].values
Incubation = WB2['Incubation period'].values
Generation_time = WB2['Generation time'].values
beta_vaccination = 0.361
beta_distancing = 0.577
beta_incubation = -0.03
beta_generation = 0.092
X1 = beta_vaccination*Vaccination
X2 = beta_distancing*Social_distance
X3 = beta_incubation*Incubation
X4 = beta_generation*Generation_time
WB['External'] = X1 + X2 + X3 + X4

```

6. The ARIMA parameters and external parameters were used to forecast the COVID-19 confirmed cases in this stage, the following code provides a way to plot these data and include the real data for fitting. Noted that in this stage, a 95% confidence level is applied for COVID-19 confirmed cases collected

```
# ARIMA model: provide a forecasting model of ARIMA => ARIMA (1,1,0)
model = ARIMA(WB['Data value'], order=(1,1,0))
RESULT = model.fit()
print(RESULT.summary())

fig2, ax = plt.subplots(1,2,figsize=(16,6), dpi= 80)
residuals = DataFrame(RESULT.resid) # Line plot of residuals
residuals.plot(title="Residuals", ax=ax[0])
residuals.plot(title="Density", ax=ax[1], kind='kde') # Density plot of residuals
plt.show()

# Summary stats of residuals
print(residuals.describe())
# Forecast ARIMA (1,1,0)
WB['ARIMA(1,1,0)'] = RESULT.predict(start=0, end=207, dynamic=False)
WB['Final Forecast result'] = WB['ARIMA(1,1,0)'] + WB['External']

```

```

# 95% Confidence interval:
CI = 1.96*np.std(WB['Data value'])/np.sqrt(len(x)) # since Confidence level = 95%, z-value = 1.96
# Plot
fig3, ax3 = plt.subplots(1,1)
plt.title('Forecast of COVID-19 Confirmed Cases in HCMC', alpha = 0.5)
plt.plot(x, WB['Data value'])
plt.plot(x, WB['Final Forecast result'])
ax3.fill_between(x, (WB['Data value'] - CI), (WB['Data value'] + CI), color = "b", alpha = 0.2)
plt.legend(["COVID-19 Confirmed Cases","ARIMA(1,1,0) Forecast"])
plt.show()
```

7. After providing forecast results, the study continued with calculating and plotting a 7-day moving average between forecasting results and real data. MSE and RMSE calculation was also included in this part for error comparison in this study:

```python
# Calculate the 7-day moving average of both confirmed cases and forecasting results:
# For confirmed cases:
window_size1 = 7
i = 0
moving_avg1 = [] # Provide an empty list to store moving averages
# Loop through the array to consider every window of size 7:
while i < len(WB['Data value']) - window_size1 + 1:
    list1 = list(WB['Data value'])
    window1 = list1[i:i+window_size1] # Store elements from i to i+window_size in list to get the current window
    window_average1 = round(sum(window1) / window_size1, 2) # Calculate the average of current window
    moving_avg1.append(window_average1) # Store the average of current window in moving average list
    i = i+1
```

```python
# for forecasting results:
window_size2 = 7
j = 0
moving_avg2 = [] # Provide an empty list to store moving averages
# Loop through the array to consider every window of size 7:
while j < len(WB['Final Forecast result']) - window_size2 + 1:
    list2 = list(WB['Final Forecast result'])
    window2 = list2[j:j+window_size2] # Store elements from i to i+window_size in list to get the current window
    window_average2 = round(sum(window2) / window_size2, 2) # Calculate the average of current window
    moving_avg2.append(window_average2) # Store the average of current window in moving average list
    j = j+1

x1 = np.linspace(0,201,201) # Specify the number of observation in 7-day moving average => 201 observations
plt.plot(x1, moving_avg1)
plt.plot(x1, moving_avg2)
plt.legend(["COVID-19 Confirmed Cases","ARIMA(1,1,0) Forecast"])
plt.title('7-day moving average of Covid-19 confirmed cases & forecasting result', alpha=0.5)
plt.show()
```

```python
# RMSE Calculation:
MSE = np.square(np.subtract(WB['Data value'], WB['Final Forecast result'])).mean()
RMSE = math.sqrt(MSE)
print("Mean Square Error: ", MSE)
print("Root Mean Square Error: ", RMSE)
```