Supplementary materials

-------------------------------------------------------------------------------------------------------------------------

\*R code\*

-------------------------------------------------------------------------------------------------------------------------

\*\*\*\*SIR model\*\*\*\*

library(deSolve)

sir\_equations <- function(time, variables, parameters) {

with(as.list(c(variables, parameters)), {

dS <- -beta \* I \* S

dI <- beta \* I \* S - gamma \* I

dR <- gamma \* I

return(list(c(dS, dI, dR)))

})

}

parameters\_values <- c(

beta = 0.378, # infectious contact rate (/person/day)

gamma = 0.14 # recovery rate (/day)

)

initial\_values <- c(

S = 25000000, # number of susceptibles at time = 0

I = 1, # number of infectious at time = 0

R = 0 # number of recovered (and immune) at time = 0

)

time\_values <- seq(0, 400)

#ls()

#sir\_equations

#parameters\_values

#initial\_values

sir\_values\_1 <- ode(

y = initial\_values,

times = time\_values,

func = sir\_equations,

parms = parameters\_values

)

#sir\_values\_1

sir\_values\_1 <- as.data.frame(sir\_values\_1)

#sir\_values\_1

with(sir\_values\_1, {

# plotting the time series of susceptibles:

plot(time, S, type = "l", col = "blue",

xlab = "time (days)", ylab = "number of people")

# adding the time series of infectious:

lines(time, I, col = "red")

# adding the time series of recovered:

lines(time, R, col = "green")

})

# adding a legend:

legend("right", c("susceptibles", "infectious", "recovered"),

col = c("blue", "red", "green"), lty = 1, bty = "n")

#(999 + 1) \* parameters\_values["beta"] / parameters\_values["gamma"]

sir\_1 <- function(beta, gamma, S0, I0, R0, times) {

require(deSolve) # for the "ode" function

# the differential equations:

sir\_equations <- function(time, variables, parameters) {

with(as.list(c(variables, parameters)), {

dS <- -beta \* I \* S

dI <- beta \* I \* S - gamma \* I

dR <- gamma \* I

return(list(c(dS, dI, dR)))

})

}

# the parameters values:

parameters\_values <- c(beta = beta, gamma = gamma)

# the initial values of variables:

initial\_values <- c(S = S0, I = I0, R = R0)

# solving

out <- ode(initial\_values, times, sir\_equations, parameters\_values)

# returning the output:

as.data.frame(out)

}

sir\_1(beta = 0.378, gamma = 0.14, S0 = 25000000, I0 = 1, R0 = 0, times = seq(0, 400))

library(lubridate)

Australia2<-read.csv("E:/Australia2.csv")

df<- subset(Australia2, select = c(cases,day))

with(df, plot(day, cases, pch = 19, col = "blue", ylim = c(0, 600)))

predictions <- sir\_1(beta = 0.378, gamma = 0.14, S0 = 25000000, I0 = 1, R0 = 0, times = df$day)

with(predictions, lines(time, I, col = "red"))

predictions <- sir\_1(beta = 0.378, gamma = 0.14, S0 = 25000000, I0 = 1, R0 = 0, times = df$day)

RMSE=sqrt(mean((predictions$I - df$cases)^2)) / sd(df$cases)

RMSE

-------------------------------------------

\*\*Improved version of SIR model

-------------------------------------------

library(deSolve)

sir\_equations <- function(time, variables, parameters) {

with(as.list(c(variables, parameters)), {

dS <- -beta \* I \* S

dI <- beta \* I \* S - gamma \* I

dR <- gamma \* I

return(list(c(dS, dI, dR)))

})

}

parameters\_values <- c(

beta = 0.378, # infectious contact rate (/person/day)

gamma = 0.14 # recovery rate (/day)

)

initial\_values <- c(

S = 60000000, # number of susceptibles at time = 0

I = 1, # number of infectious at time = 0

R = 0 # number of recovered (and immune) at time = 0

)

time\_values <- seq(0, 400)

#ls()

#sir\_equations

#parameters\_values

#initial\_values

sir\_values\_1 <- ode(

y = initial\_values,

times = time\_values,

func = sir\_equations,

parms = parameters\_values

)

#sir\_values\_1

sir\_values\_1 <- as.data.frame(sir\_values\_1)

#sir\_values\_1

with(sir\_values\_1, {

# plotting the time series of susceptibles:

plot(time, S, type = "l", col = "blue",

xlab = "time (days)", ylab = "number of people")

# adding the time series of infectious:

lines(time, I, col = "red")

# adding the time series of recovered:

lines(time, R, col = "green")

})

# adding a legend:

legend("right", c("susceptibles", "infectious", "recovered"),

col = c("blue", "red", "green"), lty = 1, bty = "n")

#(999 + 1) \* parameters\_values["beta"] / parameters\_values["gamma"]

sir\_1 <- function(beta, gamma, S0, I0, R0, times) {

require(deSolve) # for the "ode" function

# the differential equations:

sir\_equations <- function(time, variables, parameters) {

with(as.list(c(variables, parameters)), {

dS <- -beta \* I \* S

dI <- beta \* I \* S - gamma \* I

dR <- gamma \* I

return(list(c(dS, dI, dR)))

})

}

# the parameters values:

parameters\_values <- c(beta = beta, gamma = gamma)

# the initial values of variables:

initial\_values <- c(S = S0, I = I0, R = R0)

# solving

out <- ode(initial\_values, times, sir\_equations, parameters\_values)

# returning the output:

as.data.frame(out)

}

library(lubridate)

UK2<-read.csv("E:/UK2.csv")

df<- subset(UK2, select = c(cases,day))

mLL <- function(beta, gamma, sigma, day, cases, N = 67000000) {

beta <- exp(beta) # to make sure that the parameters are positive

gamma <- exp(gamma)

sigma <- exp(sigma)

I0 <- cases[1] # initial number of infectious

observations <- cases[-1] # the fit is done on the other data points

predictions <- sir\_1(beta = beta, gamma = gamma,

S0 = N - I0, I0 = I0, R0 = 0, times = day)

predictions <- predictions$I[-1] # removing the first point too

# returning minus log-likelihood:

-sum(dnorm(x = observations, mean = predictions, sd = sigma, log = TRUE))

}

library(bbmle)

starting\_param\_val <- list(beta = 0.378, gamma = 0.14, sigma = 1)

estimates <-mle2(minuslogl = mLL, start = lapply(starting\_param\_val, log),

method = "Nelder-Mead","optim", data = c(df, N = 67000000))

exp(coef(estimates))

N <- 67000000 # total population size

time\_points <- seq(min(df$day), max(df$day), le = 100) # vector of time points

I0 <- df$cases[1] # initial number of infected

param\_hat <- exp(coef(estimates)) # parameters estimates

# model's best predictions:

best\_predictions <- sir\_1(beta = param\_hat["beta"], gamma = param\_hat["gamma"],

S0 = N - I0, I0 = I0, R0 = 0, time\_points)$I

# confidence interval of the best predictions:

cl <- 0.99 # confidence level

cl <- (1 - cl) / 2

lwr <- qnorm(p = cl, mean = best\_predictions, sd = param\_hat["sigma"])

upr <- qnorm(p = 1 - cl, mean = best\_predictions, sd = param\_hat["sigma"])

# layout of the plot:

plot(time\_points, time\_points, ylim = c(0, max(upr)), type = "n",

xlab = "time (days)", ylab = "prevalence")

# adding the predictions' confidence interval:

sel <- time\_points >= 1 # predictions start from the second data point

polygon(c(time\_points[sel], rev(time\_points[sel])), c(upr[sel], rev(lwr[sel])),

border = NA, col = adjustcolor("red", alpha.f = 0.1))

# adding the model's best predictions:

lines(time\_points, best\_predictions, col = "red")

# adding the observed data:

with(df, points(day, cases, pch = 19, col = "blue"))

RMSE=sqrt(mean((best\_predictions - df$cases)^2)) / sd(df$cases)

RMSE

-------------------------- --------------------------------------------

\*\*Improved version of SEIR model\*\*

--------------------------- -------------------------------------------

#Writing a simulator

t<-seq(0,125)

seir\_1 <- function(beta,sigma, gamma,mu, S0, E0,I0, R0, times) {

require(deSolve) # for the "ode" function

# the differential equations:

seir\_ode<-function(t,Y,par){

S<-Y[1]

E<-Y[2]

I<-Y[3]

R<-Y[4]

beta<-par[1]

sigma<-par[2]

gamma<-par[3]

mu<-par[4]

dYdt<-vector(length=4)

dYdt[1]=mu-beta\*I\*S-mu\*S

dYdt[2]=beta\*I\*S-(sigma+mu)\*E

dYdt[3]=sigma\*E-(gamma+mu)\*I

return(list(dYdt))

}

# the parameters values:

parameters\_values <- c(beta = beta, sigma=sigma,gamma = gamma,mu=mu)

# the initial values of variables:

initial\_values <- c(S = S0, E=E0,I = I0, R = R0)

# solving

out <- ode(initial\_values,times,seir\_ode,parameters\_values)

# returning the output:

as.data.frame(out)

}

seir\_1(beta=520/365 ,sigma=1/60,gamma =1/30,mu=774835/(65640000\*365),S0 = 67000000,E0=1, I0 = 1, R0 = 0,times=seq(0,10))

data1<-read.csv("C:/Users/user pc/Desktop/UK3.csv")

with(data1, plot(day, cases, pch = 19, col = "red", ylim = c(0, 600)))

predictions <- seir\_1(beta=520/365 ,sigma=1/60,gamma =1/30,mu=774835/(65640000\*365),S0 = 67000000,E0=1, I0 = 1, R0 = 0, times = data1$day)

with(predictions, lines(time, I, col = "red"))

model\_fit <- function(beta, sigma,gamma,mu, data, N = 67000000, ...) {

I0 <- data1$cases[1] # initial number of infected (from data)

times <- data1$day # time points (from data)

# model's predictions:

predictions <- seir\_1(beta = beta, sigma=sigma,gamma = gamma,mu=mu, # parameters

S0 = N - I0, I0 = I0, R0 = 0,E0=1, # variables' intial values

times = times) # time points

# plotting the observed prevalences:

with(data1, plot(day, cases, ...))

# adding the model-predicted prevalence:

with(predictions, lines(time, I, col = "red"))

}

model\_fit(beta=520/365 ,sigma=1/60,gamma =1/30,mu=774835/(65640000\*365), data1, pch = 19, col = "red", ylim = c(0, 600))

predictions <- seir\_1(beta=520/365 ,sigma=1/60,gamma =1/30,mu=774835/(65640000\*365), S0 = 67000000, I0 = 1, R0 = 0, E0=1,times = data1$day)

#sum((predictions$I - data1$cases)^2)

# the observed prevalences:

#with(data1, plot(day, cases, pch = 19, col = "red", ylim = c(0, 600)))

# the model-predicted prevalences:

#with(predictions, lines(time, I, col = "red", type = "o"))

# the "errors":

#segments(data1$day, data1$cases, predictions$time, predictions$I)

ss <- function(beta, sigma,gamma, mu,data = data1, N = 67000000) {

I0 <- data$cases[1]

times <- data$day

predictions <- seir\_1(beta = beta, sigma=sigma,gamma = gamma,mu=mu, # parameters

S0 = N - I0, I0 = I0, R0 = 0,E0=1, # variables' intial values

times = times) # time points

sum((predictions$I[-1] - data$cases[-1])^2)

}

ss(beta=520/365 ,sigma=1/60,gamma =1/30,mu=774835/(65640000\*365))

#beta\_val <- seq(from = 0.0016, to = 0.004, le = 100)

#ss\_val <- sapply(beta\_val, ss, gamma = 0.5,sigma=1/60,mu=774835/(65640000\*365))

#min\_ss\_val <- min(ss\_val)

#beta\_hat <- beta\_val[ss\_val == min\_ss\_val]

ss(beta=520/365 ,sigma=1/60,gamma =1/30,mu=774835/(65640000\*365))

ss2 <- function(x) {

ss(beta = x[1],sigma=x[2], gamma = x[3],mu=x[4])

}

#ss2(c(0.004, 1/60,0.5,774835/(65640000\*365)))

starting\_param\_val <- c(0.004, 1/60,0.5,774835/(65640000\*365))

ss\_optim <- optim(starting\_param\_val, ss2)

mLL <- function(beta, sigma,gamma,mu, siigma, day, cases, N = 67000000) {

beta <- exp(beta) # to make sure that the parameters are positive

sigma<-exp(sigma)

gamma <- exp(gamma)

mu<-exp(mu)

siigma <- exp(siigma)

I0 <- cases[1] # initial number of infectious

observations <- cases[-1] # the fit is done on the other data points

predictions <- seir\_1(beta = beta,sigma=sigma, gamma = gamma,mu=mu,

S0 = N - I0, I0 = I0, R0 = 0,E0=1, times = day)

predictions <- predictions$I[-1] # removing the first point too

# returning minus log-likelihood:

-sum(dnorm(x = observations, mean = predictions, sd = siigma, log = TRUE))

}

library(bbmle)

starting\_param\_val <- list(beta=520/365 ,sigma=1/60,gamma =1/30,mu=774835/(65640000\*365),siigma=1)

estimates <- mle2(minuslogl = mLL, start = lapply(starting\_param\_val, log),

method = "L-BFGS-B", data = c(data1, N = 67000000))

summary(estimates)

exp(coef(estimates))

N <- 67000000 # total population size

time\_points <- seq(min(data1$day), max(data1$day), le = 100) # vector of time points

I0 <- data1$cases[1] # initial number of infected

param\_hat <- exp(coef(estimates)) # parameters estimates

# model's best predictions:

best\_predictions <- seir\_1(beta = param\_hat["beta"], sigma=param\_hat["sigma"],gamma = param\_hat["gamma"],mu=param\_hat["mu"],

S0 = N - I0, I0 = I0, R0 = 0,E0=1, time\_points)$I

# confidence interval of the best predictions:

cl <- 0.95 # confidence level

cl <- (1 - cl) / 2

lwr <- qnorm(p = cl, mean = best\_predictions, sd = param\_hat["siigma"])

upr <- qnorm(p = 1 - cl, mean = best\_predictions, sd = param\_hat["siigma"])

# layout of the plot:

plot(time\_points, time\_points, ylim = c(0, max(upr)), type = "n",

xlab = "time (days)", ylab = "prevalence")

# adding the predictions' confidence interval:

sel <- time\_points >= 1 # predictions start from the second data point

polygon(c(time\_points[sel], rev(time\_points[sel])), c(upr[sel], rev(lwr[sel])),

border = NA, col = adjustcolor("red", alpha.f = 0.1))

# adding the model's best predictions:

lines(time\_points, best\_predictions, col = "red")

# adding the observed data:

with(data1, points(day, cases, pch = 19, col = "red"))

RMSE=sqrt(mean((best\_predictions - data1$cases)^2)) / sd(data1$cases)

RMSE

---------------------------------------------

\*\*\*Prophet algorithm\*\*

---------------------------------------------

library(lubridate)

#Call data

UK<-read.csv("E:/UK3.csv")

#Call the Prophet library

library(prophet)

#Convert dataframe to Prophet object

m <- prophet(UK)

class(m)

#Forcast data for one year

future <- make\_future\_dataframe(m, periods = 365)

#Add intervals

forecast <- predict(m, future)

df.cv <- cross\_validation(m, initial = 50, period = 365, horizon = 50, units = 'days')

df.cv

df.p <- performance\_metrics(df.cv)

df.p

plot\_cross\_validation\_metric(df.cv, metric = 'rmse')

tail(forecast[c("ds", "yhat", "yhat\_lower", "yhat\_upper")])

plot(m, forecast)

#Decompose the time

prophet\_plot\_components(m, forecast)

dyplot.prophet(m, forecast)

-------------------------------------------------------------------------------------------------------------------------

\*Python code\*

-------------------------------------------------------------------------------------------------------------------------

-----------------------------------------------------

\*\*\*Logistic function in Python

-----------------------------------------------------

def main():

pass

if \_\_name\_\_ == '\_\_main\_\_':

main()

import numpy as np

from statsmodels.api import OLS

from statsmodels.tools.tools import add\_constant

from matplotlib import pyplot as plt

import pandas as pd

from statsmodels.api import OLS

from statsmodels.tools.tools import add\_constant

df=pd.read\_csv("E:/Global data/time\_series\_covid19\_confirmed\_global.csv")

df=df.loc[df['Country/Region']=='United Kingdom']

df=df.iloc[:,4:]

daily\_cases = df.sum(axis=0)

daily\_cases.index = pd.to\_datetime(daily\_cases.index)

daily\_cases = daily\_cases.diff().fillna(daily\_cases[0]).astype(np.int64)

confirmed=daily\_cases.to\_numpy()

confirmed=confirmed[0:80]

x=np.arange(len(confirmed))

x=add\_constant(x)

from scipy.optimize import curve\_fit

from sklearn.metrics import r2\_score

from sklearn.metrics import mean\_squared\_error

from math import sqrt

def logifunc(x,A,x0,k,off):

return A / (1 + np.exp(-k\*(x-x0)))+off

confirmed=np.array(confirmed)

x=x[:,1]

(a\_,b\_,c\_,d\_),\_=curve\_fit(logifunc,x,confirmed,p0=[10,0,0,10])

plt.plot(x,confirmed,".",label="Reported cases")

confirmed\_pred=logifunc(x,a\_,b\_,c\_,d\_)

plt.plot(x,confirmed\_pred,label="Fitted logistic function")

plt.title("Prediction of deaths cases in Italy by logistic function")

plt.legend()

plt.show()

rmse = np.sqrt(mean\_squared\_error(confirmed, confirmed\_pred))

print("rmse: ", sqrt(rmse))

print(r2\_score(confirmed,confirmed\_pred))

--------------------------------------------------------

\*\*\*Initial Analysis- Part 1\*\*\*

--------------------------------------------------------

def main():

pass

if \_\_name\_\_ == '\_\_main\_\_':

main()

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

confirmed=pd.read\_csv("E:/Global data/time\_series\_covid19\_confirmed\_global.csv")

deaths=pd.read\_csv("E:/Global data/time\_series\_covid19\_deaths\_global.csv")

recovered=pd.read\_csv("E:/Global data/time\_series\_covid19\_recovered\_global.csv")

confirmed=confirmed.drop(['Province/State','Lat','Long'],axis=1)

deaths=deaths.drop(['Province/State','Lat','Long'],axis=1)

recovered=recovered.drop(['Province/State','Lat','Long'],axis=1)

confirmed=confirmed.groupby(confirmed['Country/Region']).aggregate('sum')

deaths=deaths.groupby(deaths['Country/Region']).aggregate('sum')

recovered=recovered.groupby(recovered['Country/Region']).aggregate('sum')

confirmed=confirmed.T

deaths=deaths.T

recovered=recovered.T

#Extract new information

new\_cases=confirmed.copy()

for day in range(1,len(confirmed)):

new\_cases.iloc[day]=confirmed.iloc[day]-confirmed.iloc[day-1]

growth\_rate=confirmed.copy()

for day in range(1,len(confirmed)):

growth\_rate.iloc[day]=(new\_cases.iloc[day]/confirmed.iloc[day-1])\*100

for country in countries:

ax=plt.subplot()

ax.set\_facecolor('white')

ax.figure.set\_facecolor('#121212')

ax.tick\_params(axis='x',colors='white')

ax.tick\_params(axis='y',color='white')

ax.set\_title(f'covid-19 confirmed cases Growth rate in {country}', color="white")

plt.xlabel("Date")

plt.ylabel("%")

growth\_rate[country].plot.bar()

plt.show()

--------------------------------------------------------

\*\*\*Initial Analysis- Part 2\*\*\*

--------------------------------------------------------

def main():

pass

if \_\_name\_\_ == '\_\_main\_\_':

main()

#Import package

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

#Please check the location of data

df=pd.read\_csv("E:/Australia.csv")

cols=df.columns

x=df.loc[:,cols[1]]

df=df.drop(['Lat','Long','Recovered'],axis=1)

data=df

data["Date"]=pd.to\_datetime(data.Date)

Provinces=data["Province/State"].unique()

\_data=data[data["Province/State"]=="New South Wales"]

plt.figure(1,figsize=(16,16\*8))

plt.title("New South Wales",fontsize=12)

plt.plot(\_data.Date,\_data.Confirmed,color='grey',label="Confirmed")

plt.plot(\_data.Date,\_data.Deaths,color="r",alpha=0.7,label="Deaths")

plt.legend()

plt.ylabel("Numbers")

plt.xlabel("Date")

plt.show()