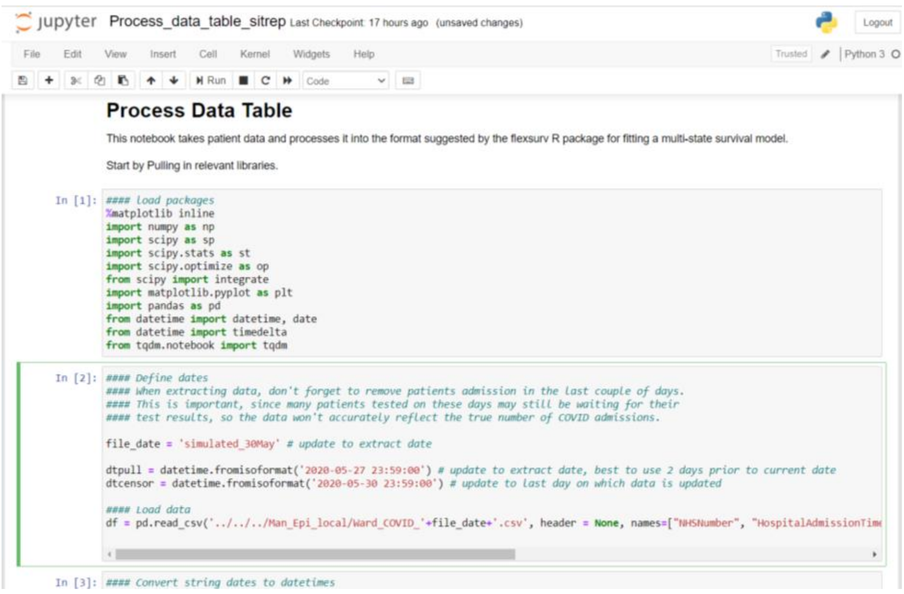
Process_data_table.ipynb

Setup: Amend all file paths to the required directories.

Step 1: Amend *file_date* to the indexing method of your input data.

Step 2: Amend *dtpull* to the last **admission** date included in the data file. For example, in the simulated data provided, the data was extracted on 30/05/2020, but any admissions after 27/05/2020 have been removed.

Step 3: Amend *dtcensor* to the last date when data is updated. For example, in the simulated data, although no admissions are included after 27/05/2020, we still include discharge/ICU data the occurs after this date (for patients admitted before this).


jupyter Process_data_table_sitrep Last Checkpoint: 17 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

Process Data Table

This notebook takes patient data and processes it into the format suggested by the flexsurv R package for fitting a multi-state survival model.

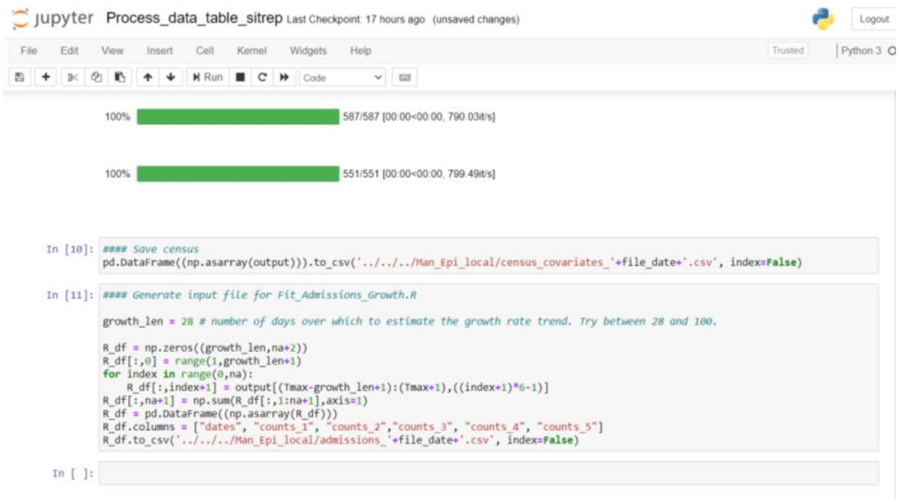
Start by Pulling in relevant libraries.

```
In [1]: #### Load packages
%matplotlib inline
import numpy as np
import scipy as sp
import scipy.stats as st
import scipy.optimize as op
from scipy import integrate
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime, date
from datetime import timedelta
from tqdm.notebook import tqdm
```

```
In [2]: #### Define dates
#### When extracting data, don't forget to remove patients admission in the last couple of days.
#### This is important, since many patients tested on these days may still be waiting for their
#### test results, so the data won't accurately reflect the true number of COVID admissions.

file_date = 'simulated_30May' # update to extract date

dtpull = datetime.fromisoformat('2020-05-27 23:59:00') # update to extract date, best to use 2 days prior to current date
dtcensor = datetime.fromisoformat('2020-05-30 23:59:00') # update to last day on which data is updated

#### Load data
df = pd.read_csv('../../../Man_Epi_local/Ward_COVID_'+file_date+'.csv', header = None, names=["NHSNumber", "HospitalAdmissionTim
```

```
In [3]: #### Convert string dates to datetimes
```

Step 4: Amend growth_len to be equal to the number of days over which we wish to estimate growth trends. For the simulated data, use 28 days. For real data, it might be worth testing 100 days. The difference depends on how much the historic data is needed to accurately capture changes over time. For example, if there is a peak/trough in the admission data between 20-28 days ago, the 28 day trend will not capture this accurately, whereas the 100 day trend should.


jupyter Process_data_table_sitrep Last Checkpoint: 17 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
100% 587/587 [00:00<00:00, 790.03it/s]

100% 551/551 [00:00<00:00, 799.49it/s]
```

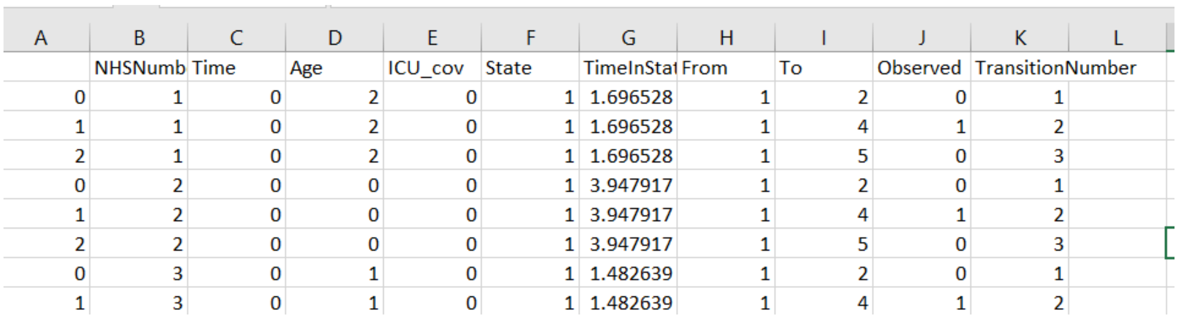
```
In [10]: #### Save census
pd.DataFrame((np.asarray(output))).to_csv('../../../Man_Epi_local/census_covariates_'+file_date+'.csv', index=False)
```

```
In [11]: #### Generate input file for Fit_Admissions_Growth.R

growth_len = 28 # number of days over which to estimate the growth rate trend. Try between 28 and 100.

R_df = np.zeros((growth_len,na+2))
R_df[:,0] = range(1,growth_len+1)
for index in range(0,na):
    R_df[:,index+1] = output[(Tmax-growth_len+1):(Tmax+1),((index+1)*6-1)]
R_df[:,na+1] = np.sum(R_df[:,1:na+1],axis=1)
R_df = pd.DataFrame((np.asarray(R_df)))
R_df.columns = ["dates", "counts_1", "counts_2","counts_3", "counts_4", "counts_5"]
R_df.to_csv('../../../Man_Epi_local/admissions_'+file_date+'.csv', index=False)
```

```
In [ ]:
```

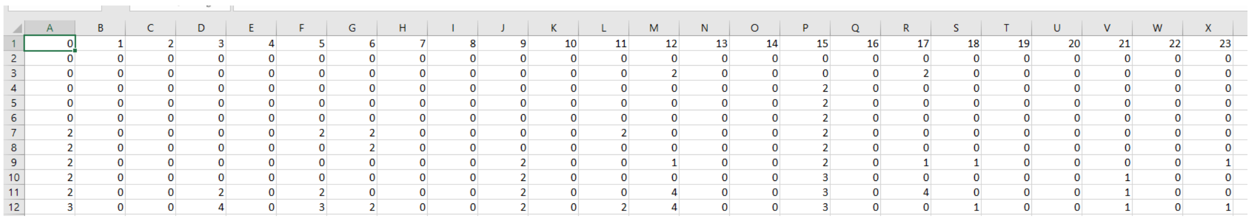
This code will generate 3 csv output files: *Current_COVID*; *census_covariates*; and *admissions.*

Current_COVID:



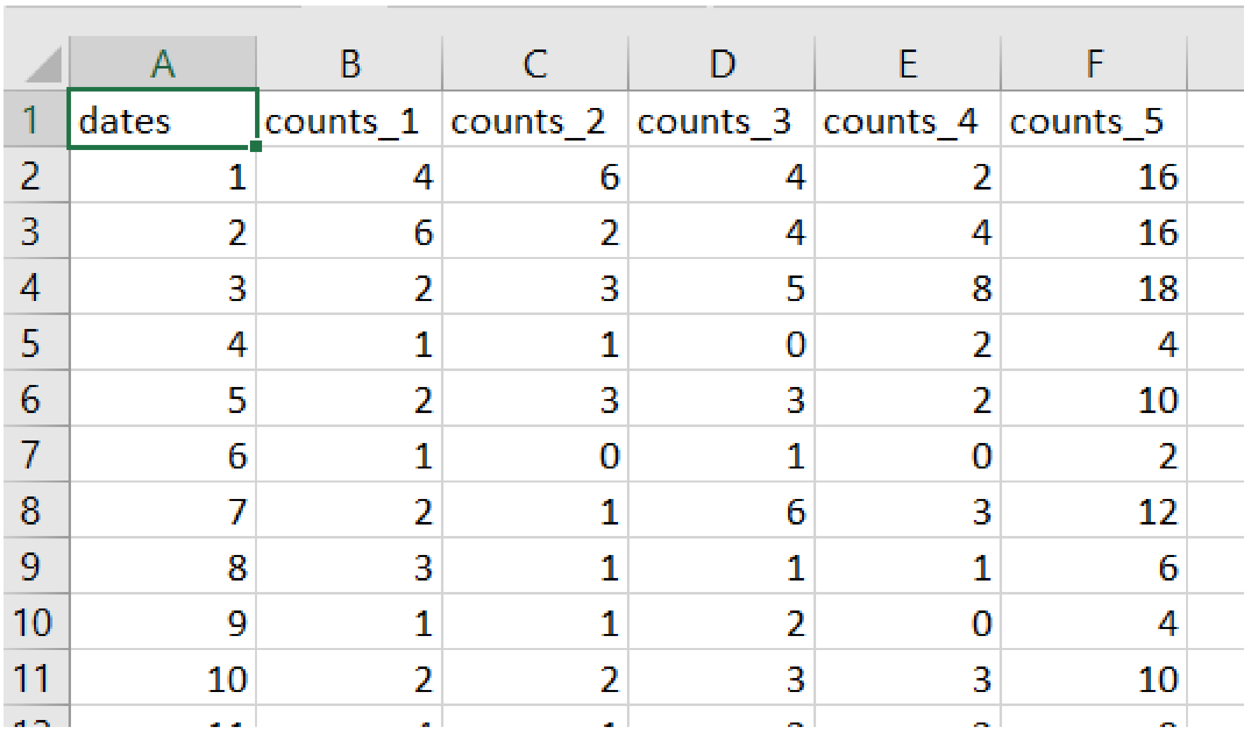
A	B	C	D	E	F	G	H	I	J	K	L
	NHSNumb	Time	Age	ICU_cov	State	TimeInStat	From	To	Observed	TransitionNumber	
0	1	0	2	0	1	1.696528	1	2	0	1	
1	1	0	2	0	1	1.696528	1	4	1	2	
2	1	0	2	0	1	1.696528	1	5	0	3	
0	2	0	0	0	1	3.947917	1	2	0	1	
1	2	0	0	0	1	3.947917	1	4	1	2	
2	2	0	0	0	1	3.947917	1	5	0	3	
0	3	0	1	0	1	1.482639	1	2	0	1	
1	3	0	1	0	1	1.482639	1	4	1	2	

census_covariates:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	2	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
7	2	0	0	0	0	2	2	0	0	0	0	2	0	0	0	2	0	0	0	0	0	0	0	0
8	2	0	0	0	0	0	2	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
9	2	0	0	0	0	0	0	0	0	2	0	0	1	0	0	2	0	1	1	0	0	0	0	1
10	2	0	0	0	0	0	0	0	0	2	0	0	0	0	0	3	0	0	0	0	0	1	0	0
11	2	0	0	2	0	2	0	0	0	2	0	0	4	0	0	3	0	4	0	0	0	1	0	0
12	3	0	0	4	0	3	2	0	0	2	0	2	4	0	0	3	0	0	1	0	0	1	0	1

admissions:



	A	B	C	D	E	F
1	dates	counts_1	counts_2	counts_3	counts_4	counts_5
2	1	4	6	4	2	16
3	2	6	2	4	4	16
4	3	2	3	5	8	18
5	4	1	1	0	2	4
6	5	2	3	3	2	10
7	6	1	0	1	0	2
8	7	2	1	6	3	12
9	8	3	1	1	1	6
10	9	1	1	2	0	4
11	10	2	2	3	3	10

Fit_Admissions_Growth.R

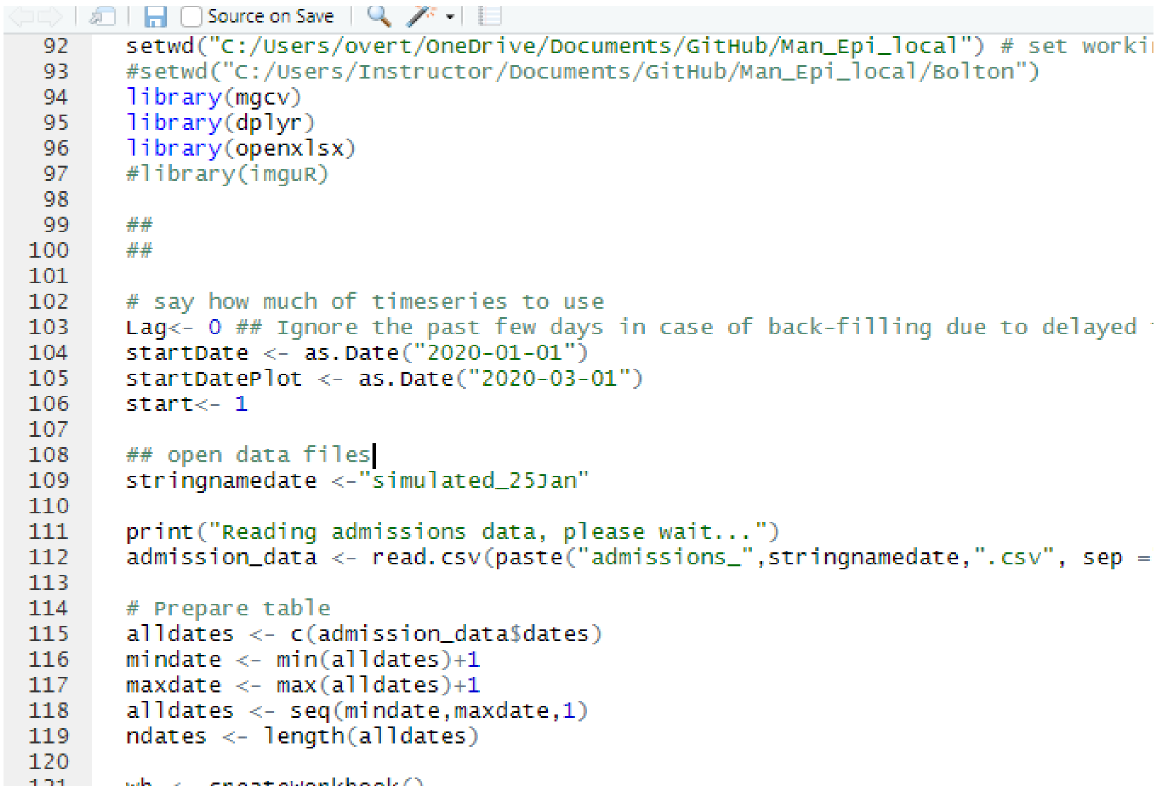
Setup: Amend all file paths to the required directories.

Step 1: Update working directory to relevant directory

Step 2: Update *stringnamedate* to be the same indexing used in the python script. That is:

stringnamedate = file_date

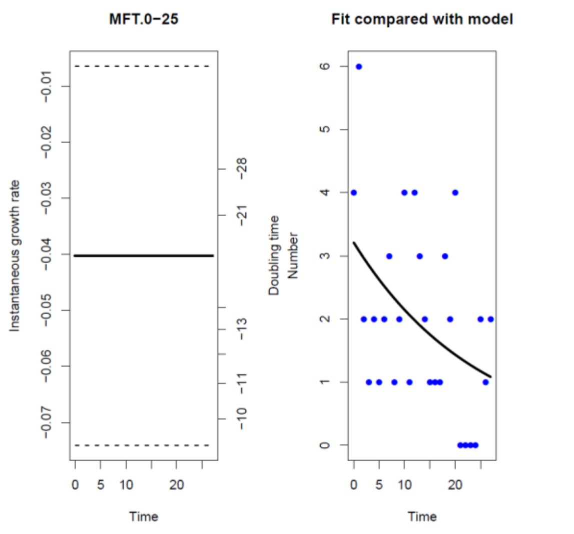
Step 3: Run the full script.



```r
setwd("C:/Users/overt/OneDrive/Documents/GitHub/Man_Epi_local") # set worki
#setwd("C:/Users/Instructor/Documents/GitHub/Man_Epi_local/Bolton")
library(mgcv)
library(dplyr)
library(openxlsx)
#library(imguR)

##
##

# say how much of timeseries to use
Lag<- 0 ## Ignore the past few days in case of back-filling due to delayed
startDate <- as.Date("2020-01-01")
startDatePlot <- as.Date("2020-03-01")
start<- 1

## open data files
stringnamedate <-"simulated_25Jan"

print("Reading admissions data, please wait...")
admission_data <- read.csv(paste("admissions_",stringnamedate,".csv", sep =

# Prepare table
alldates <- c(admission_data$dates)
mindate <- min(alldates)+1
maxdate <- max(alldates)+1
alldates <- seq(mindate,maxdate,1)
ndates <- length(alldates)

wb <- createWorkbook()
```

This code generates two output files: *MFT_cases_trends_full_(file_date).pdf*; *admissions_conditions_(file_date).csv.*

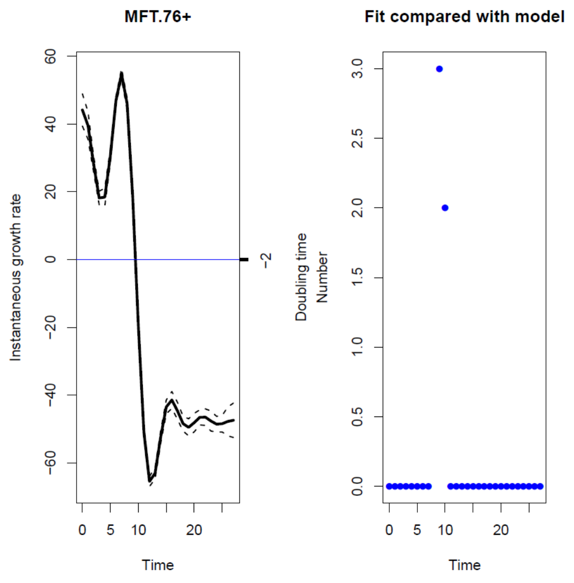
MFT_cases_trends_full_(file_date).pdf:



This figure compares the fitted admissions trends to the data. This generate 5 figures like the one on the left: one for each age group and one for all ages aggregated.

The left hand plot shows the exponential growth rate over time, and the right plot shows the corresponding admissions curve generated by that growth rate (black) compared to the raw data (blue).

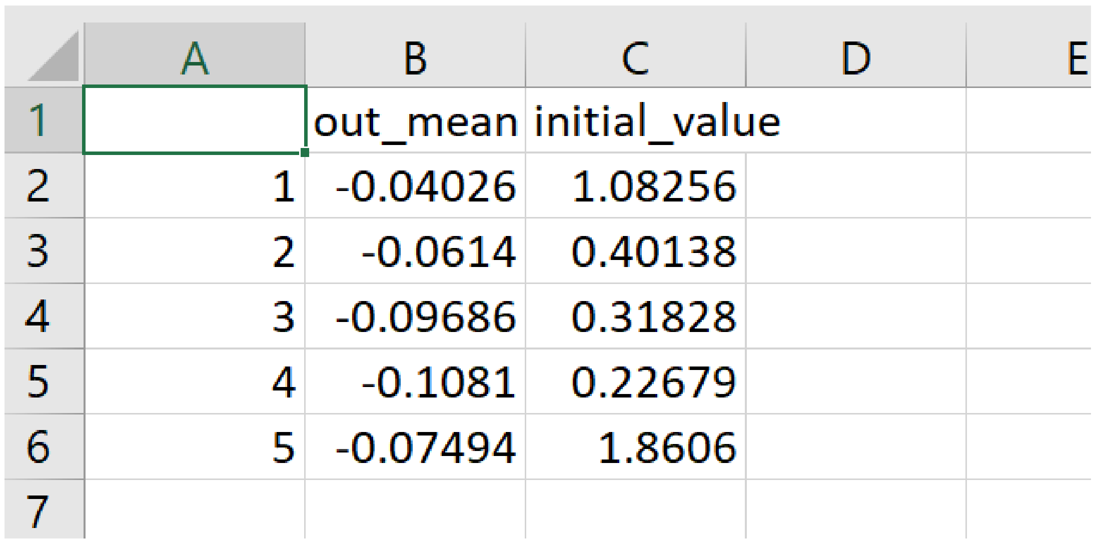
The purpose of this figures is to sense check that the fit looks sensible. Poor quality fits can occur if the admissions in any age groups are particularly sparse.



For example, the figure on the left shows a poor quality fit. If any such fits arise, we do not want to use the age specific growth rates when modelling the trends, as this will lead to poor forecasts.

In such a case, there is an option “approx._growth” in the Fit_And_Model.ipynb script. If this is set to “1”, then the model will use the growth rate from the overall admissions rather than the age specific admissions, which should be more stable.

admissions_conditions_(file_date).csv:



	A	B	C	D	E
1		out_mean	initial_value		
2	1	-0.04026	1.08256		
3	2	-0.0614	0.40138		
4	3	-0.09686	0.31828		
5	4	-0.1081	0.22679		
6	5	-0.07494	1.8606		
7					

Fit_And_Model.ipynb

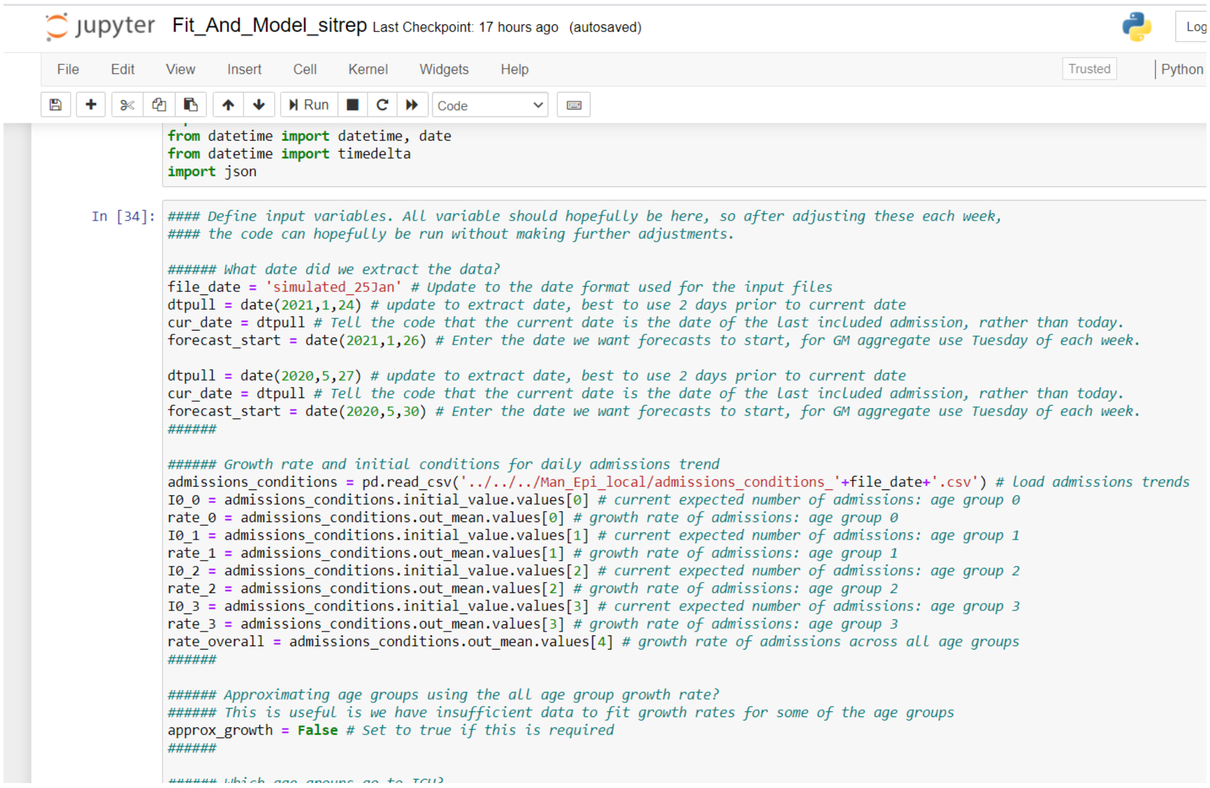
Setup: Amend all file paths to the required directories.

Step 1: Amend *file_date* to the indexing method of your input data.

Step 2: Amend *dtpull* to the last **admission** date included in the data file. For example, in the simulated data provided, the data was extracted on 30/05/2020, but any admissions after 27/05/2020 have been removed.

Step 3: Amend *forecast_start* to the date on which you wish to generate output files. The code currently outputs forecast tables with forecasts on a weekly basis, so this will be the date of the first forecast.

Step 4: Amend *approx_growth* to either **False** or **True**. By default, this should be set to **False**, unless you have growth rate issues as described above.


jupyter Fit_And_Model_sitrep Last Checkpoint: 17 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python
Run
Code

```
from datetime import datetime, date
from datetime import timedelta
import json
```

```
In [34]: #### Define input variables. All variable should hopefully be here, so after adjusting these each week,
#### the code can hopefully be run without making further adjustments.

###### What date did we extract the data?
file_date = 'simulated_25Jan' # Update to the date format used for the input files
dtpull = date(2021,1,24) # update to extract date, best to use 2 days prior to current date
cur_date = dtpull # Tell the code that the current date is the date of the last included admission, rather than today.
forecast_start = date(2021,1,26) # Enter the date we want forecasts to start, for GM aggregate use Tuesday of each week.

dtpull = date(2020,5,27) # update to extract date, best to use 2 days prior to current date
cur_date = dtpull # Tell the code that the current date is the date of the last included admission, rather than today.
forecast_start = date(2020,5,30) # Enter the date we want forecasts to start, for GM aggregate use Tuesday of each week.
######

###### Growth rate and initial conditions for daily admissions trend
admissions_conditions = pd.read_csv('../../../Man_Epi_local/admissions_conditions_'+file_date+'.csv') # load admissions trends
I0_0 = admissions_conditions.initial_value.values[0] # current expected number of admissions: age group 0
rate_0 = admissions_conditions.out_mean.values[0] # growth rate of admissions: age group 0
I0_1 = admissions_conditions.initial_value.values[1] # current expected number of admissions: age group 1
rate_1 = admissions_conditions.out_mean.values[1] # growth rate of admissions: age group 1
I0_2 = admissions_conditions.initial_value.values[2] # current expected number of admissions: age group 2
rate_2 = admissions_conditions.out_mean.values[2] # growth rate of admissions: age group 2
I0_3 = admissions_conditions.initial_value.values[3] # current expected number of admissions: age group 3
rate_3 = admissions_conditions.out_mean.values[3] # growth rate of admissions: age group 3
rate_overall = admissions_conditions.out_mean.values[4] # growth rate of admissions across all age groups
######

###### Approximating age groups using the all age group growth rate?
###### This is useful is we have insufficient data to fit growth rates for some of the age groups
approx_growth = False # Set to true if this is required
######
```

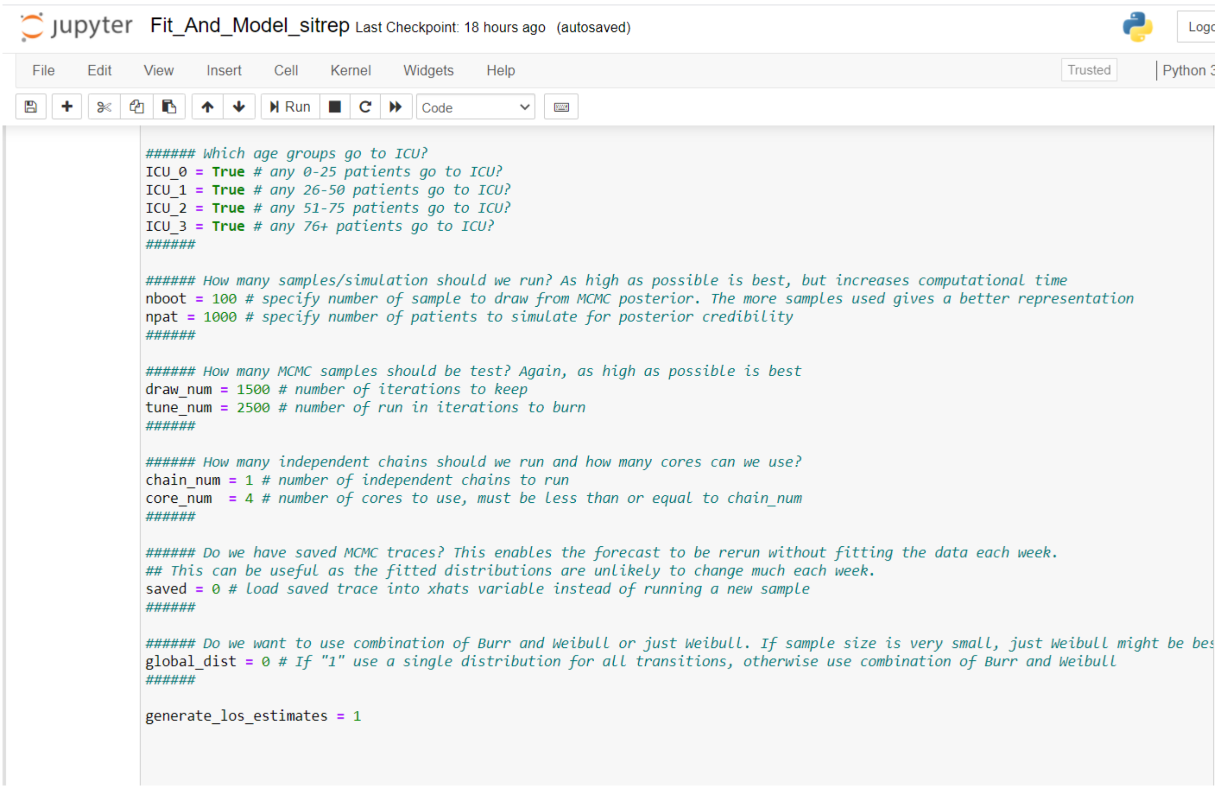
Step 5: Amend the ICU indicator variable. These determine whether the model allows individuals in each age group to go to ICU, Age group 0: 0-25, age group 1: 26-50, age group 2: 51-75, age group 3: 76+. Some hospitals see very few admissions to ICU in under 25s and over 75s. In such cases, it may be better to ban the model from allowing such transitions, since the parameters may be poorly estimated from the small sample size. If the sample size is zero, these should definitely be set to **False**. By default, these values should be **True**.

Step 6: There are various parameters that can be tuned, in order to run more or fewer simulations. Running more simulations may improve accuracy but will also increase the time the model takes to run. The default values should be sufficient, but can be changed if required.

Step 7: If you are rerunning analysis on raw data that you have already fitted before, you can set "saved = 1", which will tell the code to load existing parameters rather than rerunning the entire process.

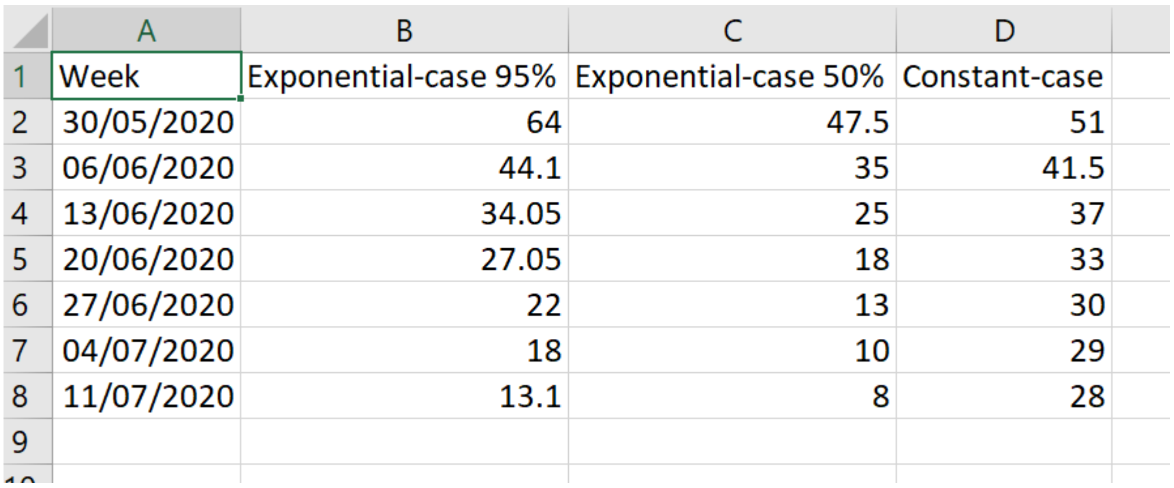
Step 8: If you have a very small sample size, setting "global_dist = 1" might give more stable estimates. By default, setting this to zero gives more accurate results.

Step 9: The final option "generate_los_estimates" specifies whether you want to only run the forecasting model or also generate length of stay estimates and patient outcome probabilities. This is useful for monitoring length of stay and looking at how outcomes might be changing over time, but is not necessary for generating the forecasts. If equal to "1", we generate this output, if set to "0" we do not.


jupyter Fit_And_Model_sitrep Last Checkpoint: 18 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
###### Which age groups go to ICU?
ICU_0 = True # any 0-25 patients go to ICU?
ICU_1 = True # any 26-50 patients go to ICU?
ICU_2 = True # any 51-75 patients go to ICU?
ICU_3 = True # any 76+ patients go to ICU?
######

###### How many samples/simulation should we run? As high as possible is best, but increases computational time
nboot = 100 # specify number of sample to draw from MCMC posterior. The more samples used gives a better representation
npat = 1000 # specify number of patients to simulate for posterior credibility
######

###### How many MCMC samples should be test? Again, as high as possible is best
draw_num = 1500 # number of iterations to keep
tune_num = 2500 # number of run in iterations to burn
######

###### How many independent chains should we run and how many cores can we use?
chain_num = 1 # number of independent chains to run
core_num  = 4 # number of cores to use, must be less than or equal to chain_num
######

###### Do we have saved MCMC traces? This enables the forecast to be rerun without fitting the data each week.
## This can be useful as the fitted distributions are unlikely to change much each week.
saved = 0 # load saved trace into xhats variable instead of running a new sample
######

###### Do we want to use combination of Burr and Weibull or just Weibull. If sample size is very small, just Weibull might be bes
global_dist = 0 # If "1" use a single distribution for all transitions, otherwise use combination of Burr and Weibull
######

generate_los_estimates = 1
```

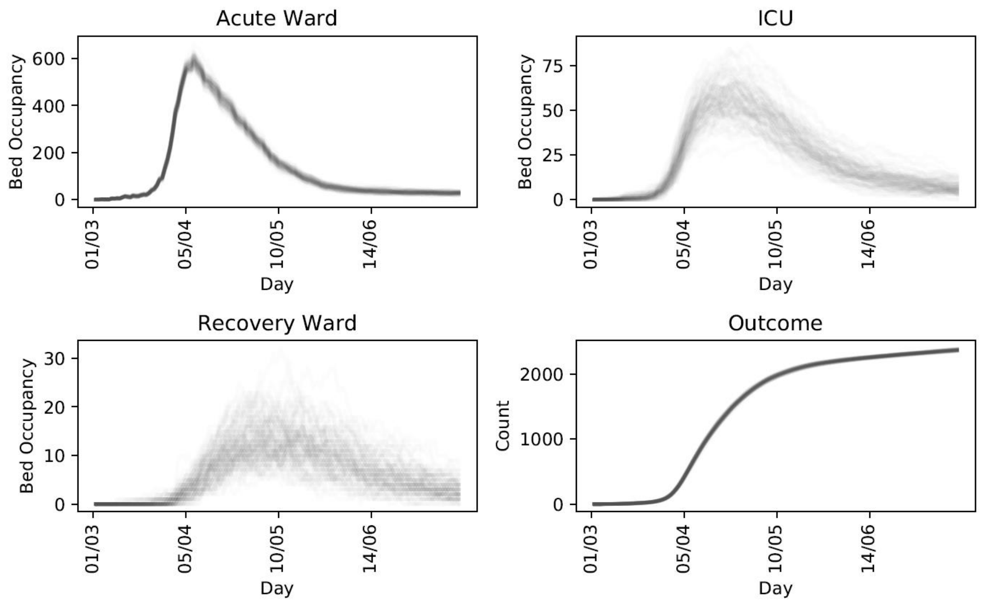
This code generates multiple outputs. Firstly, we have four output csv files: *Output_Acute_(file_date).csv, Output_ICU_(file_date).csv, Output_Deaths_(file_date).csv, Output_Discharge_(file_date).csv.* These provide estimates for Acute beds occupancy, ICU bed occupancy, daily deaths and daily discharges, reported on a weekly basis:



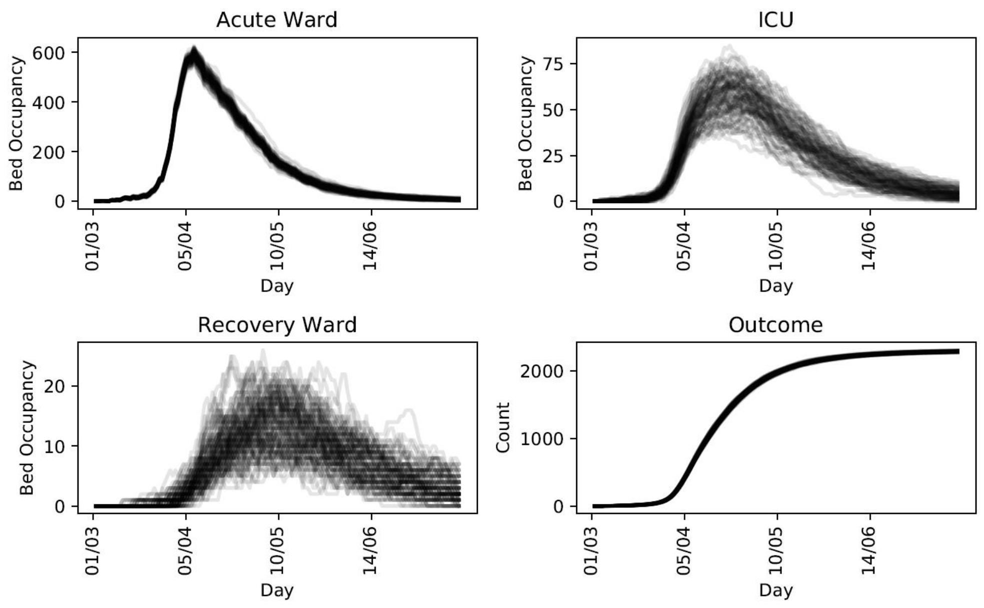
	A	B	C	D
1	Week	Exponential-case 95%	Exponential-case 50%	Constant-case
2	30/05/2020	64	47.5	51
3	06/06/2020	44.1	35	41.5
4	13/06/2020	34.05	25	37
5	20/06/2020	27.05	18	33
6	27/06/2020	22	13	30
7	04/07/2020	18	10	29
8	11/07/2020	13.1	8	28
9				

The model then generates some output figures, which can be used to visualise the forecasts and analyse model performance. These are: *indicative_stable_(file_date).pdf*, *indicative_growth_(file_date).pdf*, *modelfit_(file_date).pdf*.

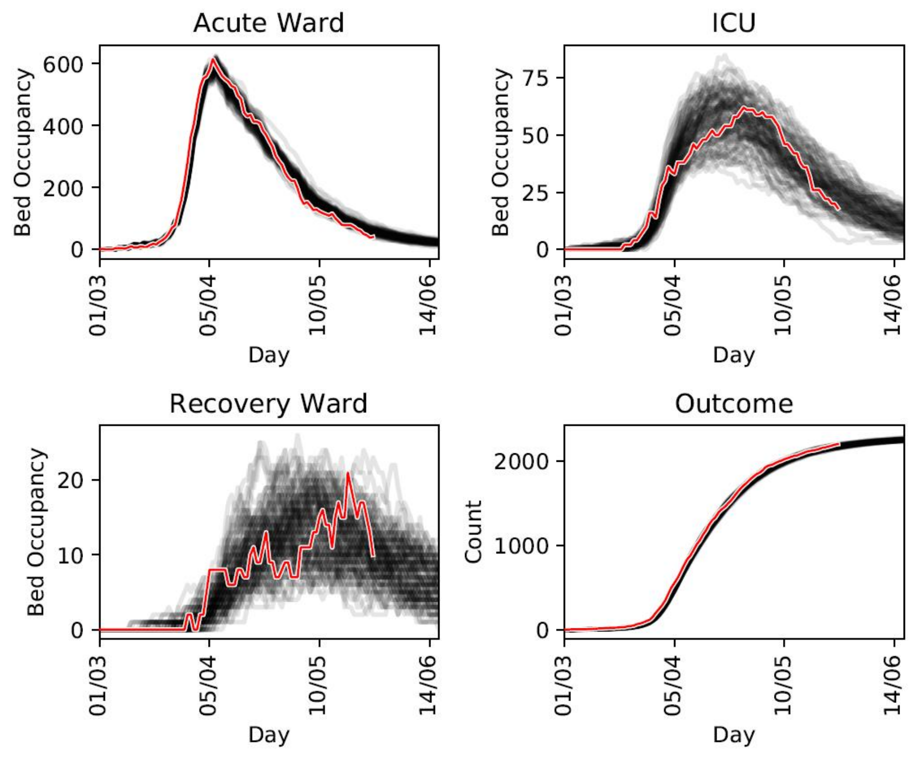
indicative_stable_(file_date).pdf: This plots forecasts under a constant admissions scenario, where admissions remain equal to their average over the last 10 days. Panes are: acute ward beds, ICU beds, recovery/stepdown beds, cumulative outcomes.



indicative_growth_(file_date).pdf: This plots forecasts under a constant admissions scenario, where admissions remain equal to their average over the last 10 days. Panes are: acute ward beds, ICU beds, recovery/stepdown beds, cumulative outcomes.



modelfit_(file_date).pdf: This compares the model simulations to the actual data. The black curves show the output of difference model simulation runs, and the red curve shows the actual data. This can be used to validate the model performance, by checking that the red curve fits within the cloud of black curves. Additionally, we can check the forecasts look sensible, by checking that the future black curves look like a sensible continuation of the red data.



Finally, if the *generate_los_estimates* option is set to “1”, the code will print some length of stay and outcome probability estimates to the screen, which can be copied and saved as a text file, if required. This output should look like as below. This reports the estimates for each age group. The output indicates the event taking place, the length of stay (in days), giving the median estimate first, followed by the 95% prediction interval. We then calculate the probability of this event, with the median probability followed by the 95% prediction interval.

```
*** Under 26 ***

Acute Ward to ICU: 3.8(2.2,6.9) days; probability 5.9(3.9,9.3)%
Acute Ward to Discharge: 8.5(7.1,9.8) days; probability 66.5(62.0,72.4)
%
Acute Ward to Death: 10.3(8.5,13.2) days; probability 27.3(21.9,31.1)%
ICU to Recovery Ward: 32.0(24.0,46.2) days; probability 69.7(52.4,84.7)
%
ICU to Death: 23.3(14.1,34.9) days; probability 30.4(15.3,47.6)%
Recovery Ward to Discharge: 7.5(5.5,11.5) days; probability 100.0(100.0
,100.0)%

*** 26-50 ***
Acute Ward to ICU: 2.5(1.3,4.4) days; probability 4.2(2.2,6.1)%
Acute Ward to Discharge: 7.9(6.9,9.4) days; probability 71.1(66.3,74.8)
%
Acute Ward to Death: 8.4(6.6,11.1) days; probability 25.0(21.4,29.8)%
ICU to Recovery Ward: 23.2(17.4,34.6) days; probability 72.1(49.7,89.8)
%
ICU to Death: 17.0(7.6,31.6) days; probability 27.9(10.2,50.3)%
Recovery Ward to Discharge: 7.5(3.8,15.7) days; probability 100.0(100.0
,100.0)%

*** 51-75 ***
Acute Ward to ICU: 3.6(2.0,5.7) days; probability 7.2(5.0,10.9)%
Acute Ward to Discharge: 8.4(7.4,10.3) days; probability 68.0(62.6,73.5
)%
Acute Ward to Death: 10.0(7.8,14.0) days; probability 24.5(20.8,29.3)%
ICU to Recovery Ward: 24.7(17.2,39.4) days; probability 74.2(61.6,84.2)
%
ICU to Death: 18.7(10.4,33.3) days; probability 25.8(15.8,38.4)%
Recovery Ward to Discharge: 12.6(9.2,22.9) days; probability 100.0(100.
0,100.0)%

*** Over 75 ***
Acute Ward to ICU: 3.6(1.5,7.3) days; probability 4.1(2.7,6.2)%
Acute Ward to Discharge: 9.6(8.1,11.0) days; probability 66.6(61.9,71.3
)%
Acute Ward to Death: 8.5(7.0,10.3) days; probability 29.0(24.8,33.6)%
ICU to Recovery Ward: 24.3(13.9,41.7) days; probability 62.8(38.3,84.1)
%
ICU to Death: 23.2(11.7,41.5) days; probability 37.2(15.9,61.7)%
Recovery Ward to Discharge: 9.8(5.4,24.9) days; probability 100.0(100.0
,100.0)%
```