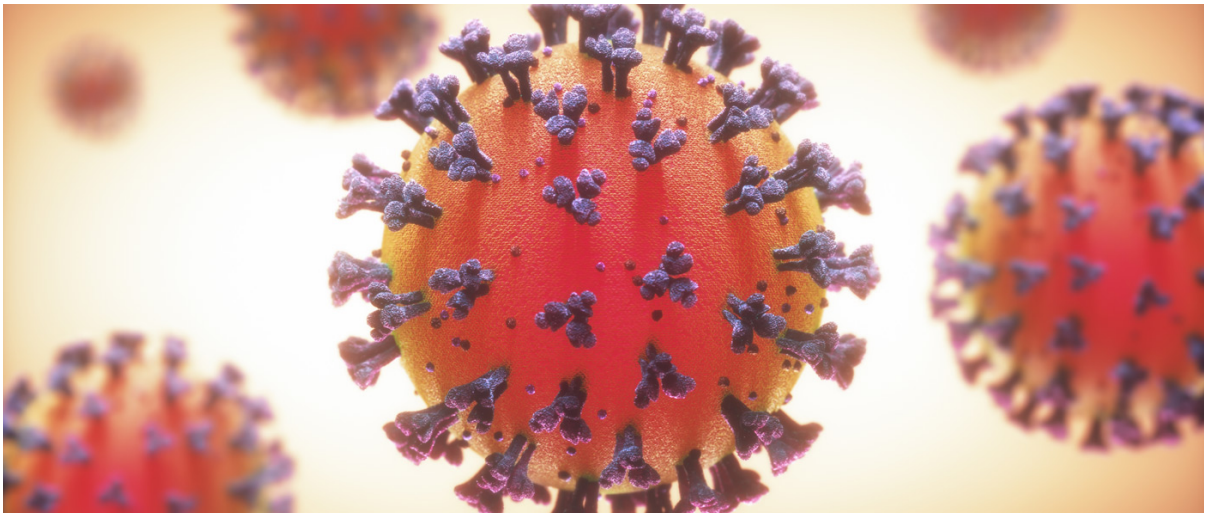
INFO 6105 Data Science Eng Methods and Tools, Northeastern University
Dino Konstantopoulos, 2 March 2020

This notebook is dedicated to Li Wenliang, "*a healthy society should have more than one voice*".



We evaluate a possible range for the infectivity, or R_0, of the Coronavirus Covid19. Specifically, we focus on how to extrapolate existing data from China in order to get a complete dataset that we can use for estimation.

We need to keep in mind that R_0 is not an absolute metric, it is one based on the virus as much as the society it infects. So different countries may experience distinct R_0s for the same epidemic. This notebook is a framework for estimating R_0 from data.

Data from China

This website (https://github.com/BlankerL/DXY-COVID-19-Data) is a result of good citizenship: People posting real data that can help other people do science, learn from mistakes, and fare better.

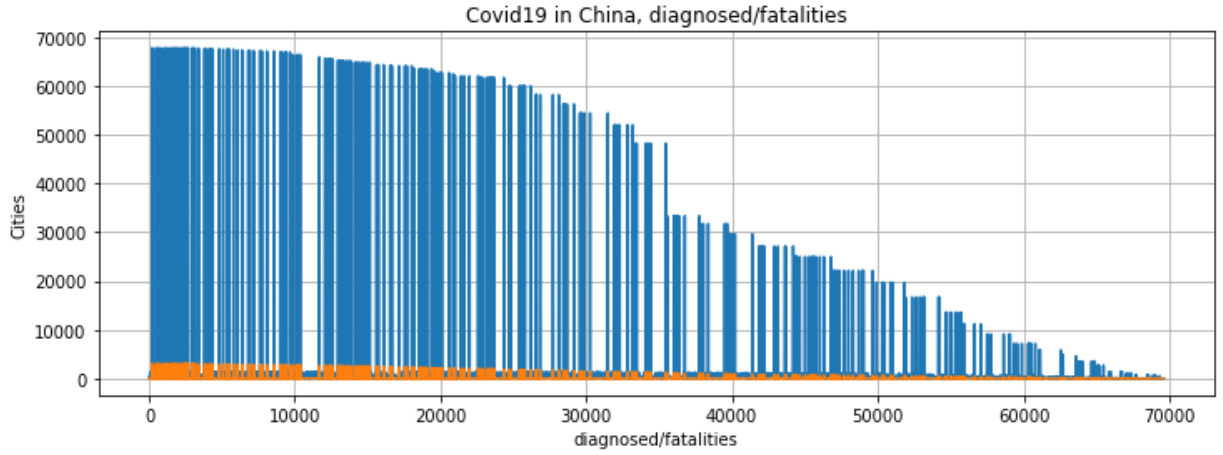
Let's download this data, import `DXYArea.csv` as a pandas dataframe, and plot some interesting plots.

In [1]:
```
import numpy as np
import pandas as pd
covid19_china = pd.read_csv('data/DXYArea.csv')
covid19_china.head(50)
```

Out[1]:

	provinceName	provinceEnglishName	province_zipCode	cityName	cityEnglishName	city_zip
0	辽宁省	Liaoning	210000	丹东	Dandong	210
1	辽宁省	Liaoning	210000	沈阳	Shenyang	210
2	辽宁省	Liaoning	210000	大连	Dalian	210
3	辽宁省	Liaoning	210000	葫芦岛	Huludao	211
4	辽宁省	Liaoning	210000	朝阳	Chaoyang	211
5	辽宁省	Liaoning	210000	锦州	Jinzhou	210
6	辽宁省	Liaoning	210000	盘锦	Panjin	211

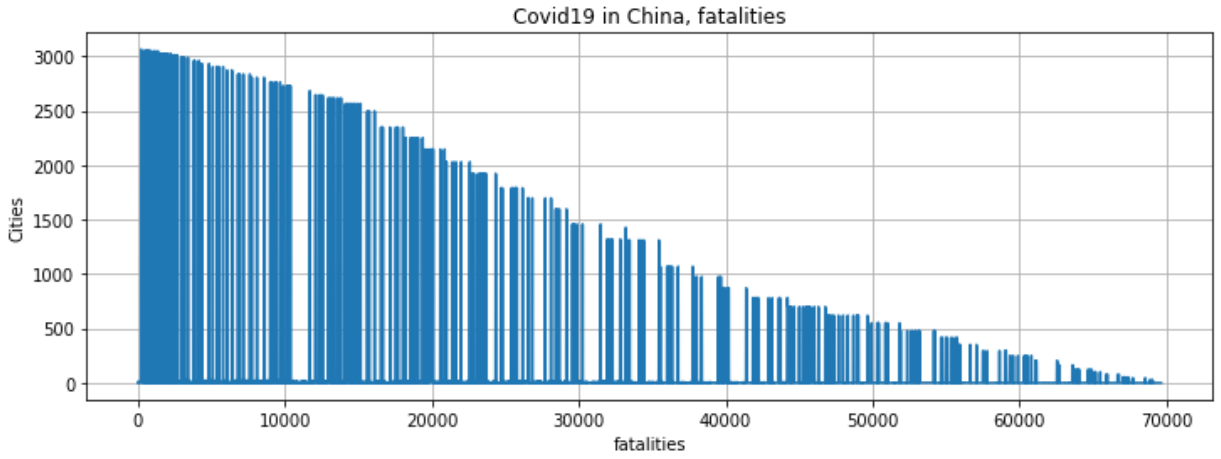
In [3]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

plt.figure(figsize=(12, 4))

plt.plot(covid19_china.province_confirmedCount)
plt.plot(covid19_china.province_deadCount)

plt.title('Covid19 in China, diagnosed/fatalities')
plt.ylabel('Cities')
plt.xlabel('diagnosed/fatalities')
plt.grid(True)
```



It looks like the data is sorted by hardest hit cities.

This is just the fatalities:

In [42]:
```
plt.figure(figsize=(12, 4))

plt.plot(covid19_china.province_deadCount)

plt.title('Covid19 in China, fatalities')
plt.ylabel('Cities')
plt.xlabel('fatalities')
plt.grid(True)
```



Let's drop some columns:

In [3]:
```
drop_cols = ['province_suspectedCount', 'city_suspectedCount', 'province_zip

covid19_china.drop(drop_cols, axis=1, inplace=True)
covid19_china.head()
```

Out[3]:

	provinceName	provinceEnglishName	cityName	cityEnglishName	province_confirmedCount	prov
0	辽宁省	Liaoning	丹东	Dandong	125	
1	辽宁省	Liaoning	沈阳	Shenyang	125	
2	辽宁省	Liaoning	大连	Dalian	125	
3	辽宁省	Liaoning	葫芦岛	Huludao	125	
4	辽宁省	Liaoning	朝阳	Chaoyang	125	

Original number of rows:

In [4]:
```
1 len(covid19_china)
```

Out[4]: 69635

Let's drop the N/As:

In [5]:
```
1 covid19_china.dropna(inplace=True)
```

New number of rows:

In [6]:
```
1 len(covid19_china)
```

Out[6]: 67997

This column looks like a good candidate for an index:

In [26]:
```
1 set(covid19_china.updateTime.values)
```

Out[26]:
```
{'2020-02-08 12:00:32.912',
 '2020-02-12 09:33:11.975',
 '2020-02-07 15:27:38.565',
 '2020-02-10 09:38:47.820',
 '2020-02-15 15:38:43.493',
 '2020-02-18 17:37:59.292',
 '2020-02-08 20:44:39.842',
 '2020-02-15 11:50:01.777',
 '2020-02-11 15:24:24.035',
 '2020-02-03 21:27:58.202',
 '2020-01-25 11:06:28.924',
 '2020-02-05 10:09:54.167',
 '2020-01-28 15:50:34.892',
 '2020-02-13 08:53:10.157',
 '2020-02-19 17:33:34.704',
 '2020-01-26 19:58:39.132',
 '2020-02-08 19:15:23.007',
 '2020-01-27 09:42:20.485',
 '2020-01-31 00:21:21.496',
```

In [14]:
```
1 covid19_china.cityEnglishName.values
```

Out[14]:
```
array(['Dandong', 'Shenyang', 'Dalian', ..., 'Changchun', 'Shijiazhuang',
       'Yinchuan'], dtype=object)
```

Wuhan

How many rows refer to `Wuhan`, the epicenter of the pandemic?

In [30]:
```
1 list(covid19_china.cityEnglishName.values).count('Wuhan')
```

Out[30]: 360

Let's get these rows:

In [7]:
```
1 covid19_wuhan = covid19_china[covid19_china['cityEnglishName']=='Wuhan']
2 covid19_wuhan
```

Out[7]:

	provinceName	provinceEnglishName	cityName	cityEnglishName	province_confirmedCount
206	湖北省	Hubei	武汉	Wuhan	67786
474	湖北省	Hubei	武汉	Wuhan	67781
600	湖北省	Hubei	武汉	Wuhan	67781
775	湖北省	Hubei	武汉	Wuhan	67781
1035	湖北省	Hubei	武汉	Wuhan	67773
...	...	...	...	...	...
69075	湖北省	Hubei	武汉	Wuhan	549
69331	湖北省	Hubei	武汉	Wuhan	549
69339	湖北省	Hubei	武汉	Wuhan	549
69345	湖北省	Hubei	武汉	Wuhan	549
69349	湖北省	Hubei	武汉	Wuhan	549

360 rows × 11 columns

Let's sort chronologically:

In [8]:
```
covid19_wuhan2 = covid19_wuhan.sort_values(by=['updateTime'])
covid19_wuhan2.head(30)
```

Out[8]:

	provinceName	provinceEnglishName	cityName	cityEnglishName	province_confirmedCount
69349	湖北省	Hubei	武汉	Wuhan	549
69345	湖北省	Hubei	武汉	Wuhan	549
69339	湖北省	Hubei	武汉	Wuhan	549
69331	湖北省	Hubei	武汉	Wuhan	549
69075	湖北省	Hubei	武汉	Wuhan	549
69052	湖北省	Hubei	武汉	Wuhan	549
69044	湖北省	Hubei	武汉	Wuhan	549
68992	湖北省	Hubei	武汉	Wuhan	549
68979	湖北省	Hubei	武汉	Wuhan	549
68971	湖北省	Hubei	武汉	Wuhan	549
68963	湖北省	Hubei	武汉	Wuhan	549
68871	湖北省	Hubei	武汉	Wuhan	549
68863	湖北省	Hubei	武汉	Wuhan	549
68847	湖北省	Hubei	武汉	Wuhan	549
68565	湖北省	Hubei	武汉	Wuhan	729
68551	湖北省	Hubei	武汉	Wuhan	729
68543	湖北省	Hubei	武汉	Wuhan	729
68522	湖北省	Hubei	武汉	Wuhan	729
68512	湖北省	Hubei	武汉	Wuhan	729
68500	湖北省	Hubei	武汉	Wuhan	729
67740	湖北省	Hubei	武汉	Wuhan	730
67722	湖北省	Hubei	武汉	Wuhan	730

	provinceName	provinceEnglishName	cityName	cityEnglishName	province_confirmedCount
67672	湖北省	Hubei	武汉	Wuhan	730
67658	湖北省	Hubei	武汉	Wuhan	761
67387	湖北省	Hubei	武汉	Wuhan	1053
67253	湖北省	Hubei	武汉	Wuhan	1052
67096	湖北省	Hubei	武汉	Wuhan	1052
67081	湖北省	Hubei	武汉	Wuhan	1052
67053	湖北省	Hubei	武汉	Wuhan	1058
66760	湖北省	Hubei	武汉	Wuhan	1423

Let's set the index:

In [30]:
```
covid19_wuhan3 = covid19_wuhan2.set_index('updateTime')
covid19_wuhan3
```

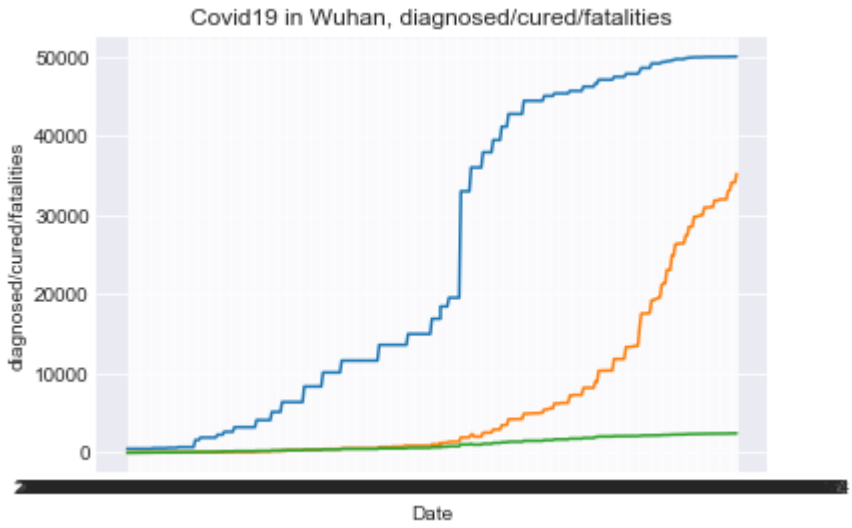
Out[30]:

updateTime	provinceName	provinceEnglishName	cityName	cityEnglishName	province_confirmedC
2020-01-24 09:47:38.698	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:48:39.253	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:49:39.772	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:50:40.357	湖北省	Hubei	武汉	Wuhan	
2020-01-24 11:49:48.584	湖北省	Hubei	武汉	Wuhan	
...	...	...	...	...	
2020-03-11 18:49:50.822	湖北省	Hubei	武汉	Wuhan	6
2020-03-12 09:21:37.890	湖北省	Hubei	武汉	Wuhan	6
2020-03-12 14:27:43.370	湖北省	Hubei	武汉	Wuhan	6
2020-03-12 17:33:24.482	湖北省	Hubei	武汉	Wuhan	6
2020-03-13 11:08:59.974	湖北省	Hubei	武汉	Wuhan	6

360 rows × 10 columns

Let's plot diagnosed, cured, and fatalities for Wuhan, chronologically:

In [31]:
```
plt.plot(covid19_wuhan3.city_confirmedCount)
plt.plot(covid19_wuhan3.city_curedCount)
plt.plot(covid19_wuhan3.city_deadCount)

plt.title('Covid19 in Wuhan, diagnosed/cured/fatalities')
plt.ylabel('diagnosed/cured/fatalities')
plt.xlabel('Date')
plt.grid(True)
```



How about the entire province of `Hubei`, whose capital is `Wuhan`?

Hubei

In [48]:
```
covid19_Hubei = covid19_china[covid19_china['provinceEnglishName']=='Hubei']
covid19_Hubei
```

Out[48]:

	provinceName	provinceEnglishName	cityName	cityEnglishName	province_confirmedCount
206	湖北省	Hubei	武汉	Wuhan	67786
207	湖北省	Hubei	孝感	Xiaogan	67786
208	湖北省	Hubei	鄂州	Ezhou	67786
209	湖北省	Hubei	随州	Suizhou	67786
210	湖北省	Hubei	荆州	Jingzhou	67786
...	...	...	...	...	...
69346	湖北省	Hubei	孝感	Xiaogan	549
69347	湖北省	Hubei	黄冈	Huanggang	549
69348	湖北省	Hubei	荆州	Jingzhou	549
69349	湖北省	Hubei	武汉	Wuhan	549
69350	湖北省	Hubei	孝感	Xiaogan	549

5903 rows × 11 columns

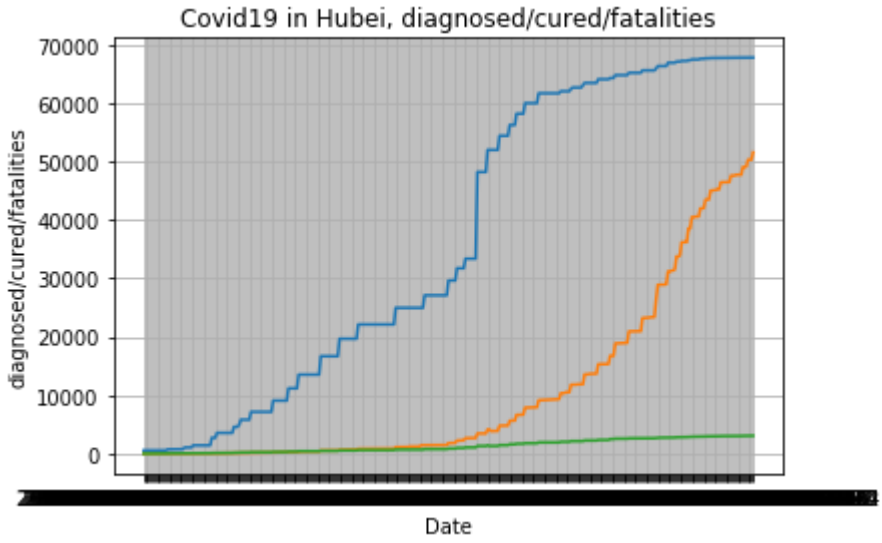
In [54]:
```
covid19_Hubei2 = covid19_Hubei.drop_duplicates(subset=['updateTime'])
covid19_Hubei3 = covid19_Hubei2.sort_values(by=['updateTime'])
covid19_Hubei4 = covid19_Hubei3.set_index('updateTime')
covid19_Hubei4
```

Out[54]:

updateTime	provinceName	provinceEnglishName	cityName	cityEnglishName	province_confirmedC
2020-01-24 09:47:38.698	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:48:39.253	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:49:39.772	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:50:40.357	湖北省	Hubei	武汉	Wuhan	
2020-01-24 11:49:48.584	湖北省	Hubei	武汉	Wuhan	
...	...	...	...	...	
2020-03-11 18:49:50.822	湖北省	Hubei	武汉	Wuhan	6
2020-03-12 09:21:37.890	湖北省	Hubei	武汉	Wuhan	6
2020-03-12 14:27:43.370	湖北省	Hubei	武汉	Wuhan	6
2020-03-12 17:33:24.482	湖北省	Hubei	武汉	Wuhan	6
2020-03-13 11:08:59.974	湖北省	Hubei	武汉	Wuhan	6

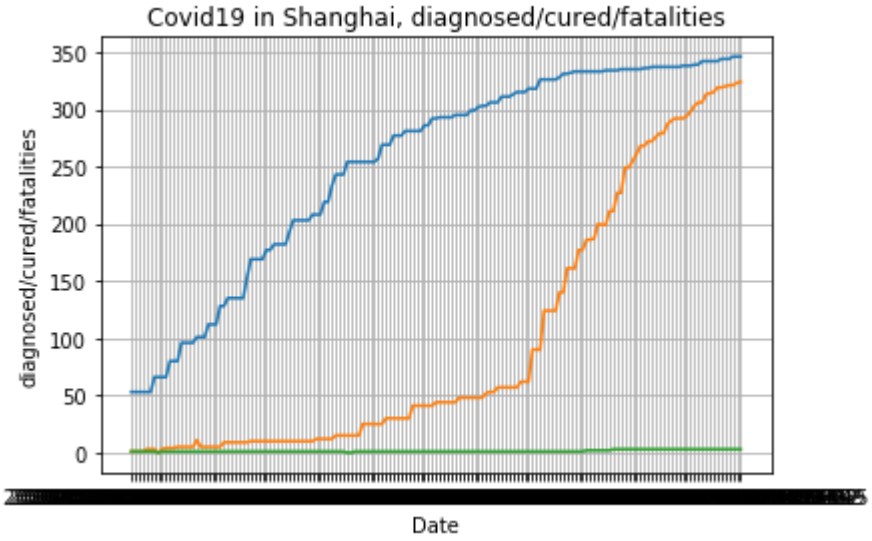
359 rows × 10 columns

In [56]:
```
plt.plot(covid19_Hubei4.province_confirmedCount)
plt.plot(covid19_Hubei4.province_curedCount)
plt.plot(covid19_Hubei4.province_deadCount)

plt.title('Covid19 in Hubei, diagnosed/cured/fatalities')
plt.ylabel('diagnosed/cured/fatalities')
plt.xlabel('Date')
plt.grid(True)
```



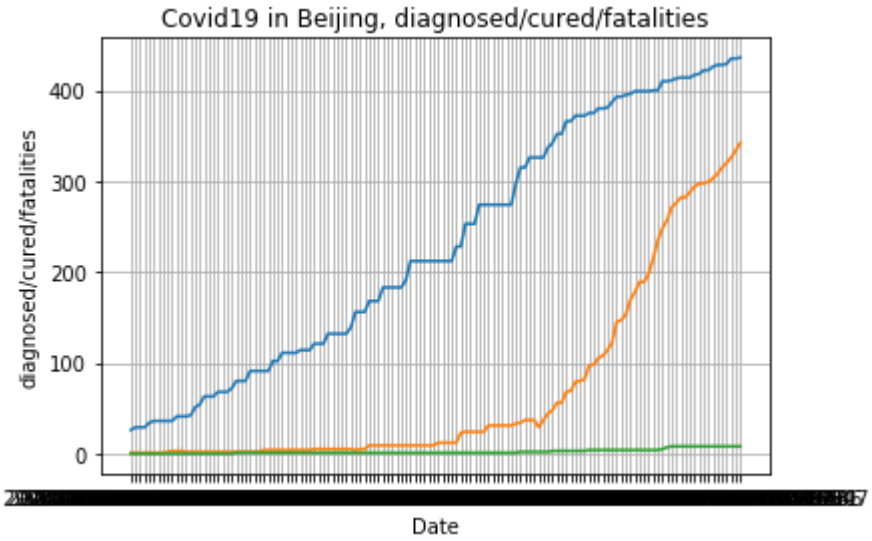
Shanghai

In [60]:
```
covid19_shanghai = covid19_china[covid19_china['provinceEnglishName']=='Shan
covid19_shanghai2 = covid19_shanghai.sort_values(by=['updateTime'])
covid19_shanghai3 = covid19_shanghai2.set_index('updateTime')

plt.plot(covid19_shanghai3.province_confirmedCount)
plt.plot(covid19_shanghai3.province_curedCount)
plt.plot(covid19_shanghai3.province_deadCount)

plt.title('Covid19 in Shanghai, diagnosed/cured/fatalities')
plt.ylabel('diagnosed/cured/fatalities')
plt.xlabel('Date')
plt.grid(True)
```



Beijing

In [64]:
```
covid19_beijing = covid19_china[covid19_china['provinceEnglishName']=='Beiji
covid19_beijing2 = covid19_beijing.sort_values(by=['updateTime'])
covid19_beijing3 = covid19_beijing2.set_index('updateTime')

plt.plot(covid19_beijing3.province_confirmedCount)
plt.plot(covid19_beijing3.province_curedCount)
plt.plot(covid19_beijing3.province_deadCount)

plt.title('Covid19 in Beijing, diagnosed/cured/fatalities')
plt.ylabel('diagnosed/cured/fatalities')
plt.xlabel('Date')
plt.grid(True)
```



And there are a lot more cities in the data!

In [20]:
```
', '.join([str(x) for x in covid19_china.cityEnglishName.values])
```

Out[20]:
```
"Dandong, Shenyang, Dalian, Huludao, Chaoyang, Jinzhou, Panjin, Fuxin, Tielin
g, Anshan, Benxi, Liaoyang, Yingkou, Harbin, Shuangyashan, Suihua, Qiqihar, D
aqing, Jixi, Qitaihe, Mudanjiang, Heihe, Hegang, Daxinganling, Jiamusi, Yichu
n, Qijiang District, Changshou District, Dazu District, Rongchang District, W
anzhou District, Jiangbei District, Yunyang County, Hechuan District, Fengjie
County, Jiulongpo District, Kaizhou District, Zhong County, Yuzhong District,
Dianjiang County, Tongnan District, Yubei District, Liangjiang New Area, Na
n'an District, Shizhu Tujia Autonomous County, Wuxi County, Tongliang Distric
t, Fengdu County, Wushan County, Shapingba District, Bishan District, Dadukou
District, Banan District, Fuling District, Yongchuan District, Jiangjin Distr
ict, Liangping District, Chongqing High-tech Zone, Qianjiang Tujia and Miao A
utonomous County, Chengkou County, Pengshui Miao and Tujia Autonomous County,
Wulong District, Xiushan Tujia and Miao Autonomous County, Youyang Tujia and
Miao Autonomous County, Wansheng District, Shenzhen, Guangzhou, Foshan, Zhong
shan, Dongguan, Shantou, Zhuhai, Huizhou, Jiangmen, Zhanjiang, Zhaoqing, Meiz
hou, Yangjiang, Maoming, Qingyuan, Shaoguan, Jieyang, Shanwei, Chaozhou, Heyu
an, Chengdu, Bazhong, Garzê Tibetan Autonomous Prefecture, Guang'an, Meishan,
Dazhou, Nanchong, Luzhou, Mianyang, Neijiang, Deyang, Suining, Panzhihua, Lia
ngshan Yi Autonomous Prefecture, Yibin, Zigong, Ya'an, Guangyuan, Ziyang, Les
```

A possible model of infection

We're going to need version `3.8` of `PyMC3` for our modeling.

In [9]:
```
import pymc3 as pm
pm.__version__
```

Out[9]:
```
'3.8'
```

In [10]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

plt.style.use('seaborn-darkgrid')
```

Acquired immunity controls the infection

The **Susceptible-Infected-Recovered** model of infection describes time dynamics of an [infectious disease (https://en.wikipedia.org/wiki/Compartmental_models_in_epidemiology)](https://en.wikipedia.org/wiki/Compartmental_models_in_epidemiology) spreading through a homogenous closed population (no births or deaths). The population is divided into three categories: **Susceptible** `S`, **Infective** `I`, or **Recovered/Dead** `R`. We can further refine our model by having a different category `D` for dead, but let's keep it simple for now and include dead people as recovered. When people die, if properly buried, they cannot infect anymore and so they are equivalent to people that have recovered and are immune to the infection.

Susceptible individuals are those that have not acquired immunity yet and are susceptible to becoming infected.

Infected individuals have been infected with the disease.

Recovered individuals are cured and not susceptible anymore to the disease.

> **NOTE**: In China, some people have been described as cured and released from hospitals and yet they test positive again for Covid19 later on. This is actually impossible because if you recover, you *have to be immune* to the disease, and is most likley assocaited with false positives and false negatives (patients where either impoperly labeled as cured, or improperly tested as infected). It is also possible that we are dealing with multiple Covid19 strains, in which case it is possible to become infected with another strain after being cured from one. It may also be possible that the virus mutates so fast that it essentially becomes a different strain in a few days and can reinfect an immune individual. God help us all if that is the case.

Differential equations (https://en.wikipedia.org/wiki/Ordinary_differential_equation) are a mathematical framework for modelling temporal dynamics of a system. The differential equations for the SIR model of infection (https://www.maa.org/press/periodicals/loci/joma/the-sir-model-for-spread-of-disease-the-differential-equation-model) are:

$$\frac{dS}{dt} = -\beta S I \quad S(0) = S_0$$

$$\frac{dI}{dt} = \beta S I - \gamma I \quad I(0) = I_0$$

$$\frac{dR}{dt} = \gamma I \quad R(0) = R_0$$

With the constraint that:

$$S(t) + I(t) + R(t) = 1 \quad , \forall t$$

Other models are available for modeling infectious diseases, such as the IBM (https://arxiv.org/ftp/arxiv/papers/1902/1902.02784.pdf) model.

If we know R(t) and I(t) then we can determine S(t): $S(t) = 1 - I(t) - R(t)$, so we can work only with the two unknowns: R(t) and I(t). We prefer to work with these because that is what the China Covid19 dataset gives us!

So we write:

$$\frac{dI}{dt} = \beta(1 - I - R)I - \gamma I \quad I(0) = I_0$$

$$\frac{dR}{dt} = \gamma I \quad R(0) = R_0$$

Simplifying:

$$\frac{dI}{dt} = \beta(1 - I - R - \gamma/\beta)I \quad I(0) = I_0$$

$$\frac{dR}{dt} = \gamma I \quad R(0) = R_0$$

We have two equations in two unknowns, so we're good mathematicaly speaking. What are the parameters?

β is the rate of infection per susceptible and per infective individual: β is an average infected to non-infected individual contact (e.g. 4 individuals) and hinges on the underlying society. Arguably, it is higher in China where the population is denser. But it also high in very social Mediterranean countries like Italy and Spain. γ is the rate of recovery: It can be interpreted as an average period of infectiousness (e.g. 3 to 5 days for the common flu). For Covid19, experts are astonished by how long an individual can infect others and say that it can be up to 30 days!

> For an example, in the Hong Kong flu in New York City in the late 1960's, hardly anyone was immune at the beginning of the epidemic, so almost everyone was susceptible. This (https://www.maa.org/press/periodicals/loci/joma/the-sir-model-for-spread-of-disease-the-differential-equation-model) paper gives an estimate for the parameters. Assuming a trace level of infection in the population, say, 10 people, population variables are S(0) = 7,900,000, I(0) = 10, R(0) = 0. In terms of scaled variables, these initial conditions are s(0) = 1, i(0) = 1.27 x 10-6, r(0) = 0. We don't know values for the parameters β and γ yet, but they can be estimated and then adjusted as necessary to fit the excess death data. The average period of infectiousness is estimated at three days, so that would suggest $\gamma = 1/3$. *Guessing* that each infected individual would make a possibly infecting contact with another individual every two days, then β would be $1/2$. So $\beta/\gamma = 3/2 > 1$.

The quantity β/γ is called R-Nought (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1181873/) (R_0). Its interpretation is that if we were to drop a *single* infected person into a population of suceptible individuals, we would expect R_0 new infections. If $R_0 > 1$, then an epidemic will take place. If $R_0 \leq 1$ then there will be no epidemic.

> When $R_0 < 1$, each person who contracts the disease will infect fewer than one person before dying or recovering, so the outbreak will fizzle ($dI/dt < 0$). When $R_0 > 1$, each person who gets the disease will infect more than one person, so the epidemic will spread ($dI/dt > 0$). R_0 is the most important quantity in epidemiology.

Let's model our SIR equations using a python function. `y` is our unknown, it's 2D: $y[0] = I$ and $y[1] = R$. . The parameters `p` are 2D: $p[0] = \beta$ and $p[1] = \gamma$.

In accordance with the `odeint` package of the `scipy.integrate` library for integrating (solving) ordinary differential equations, we'll write our `SIR` function thusly, assuming that the constant time step `dt` is equal to 1.

In [11]:
```
def SIR(y, t, p):
    di = p[0] * (1. - y[0] - y[1] - p[1]/p[0]) * y[0]
    dr = p[1] * y[0]
    return [di, dr]
```

In [12]:
```
def susceptible(i, r):
    return 1 - i - r
```

This will be our time discretization:

In [13]:
```
times = np.arange(0, 5, 0.25)
```

Let's compute *exact* values for `y` (y[0] = I and y[1] = R). Let's assume that I(0) = 0.01 and R(0) = 0. In other words, we start with 1% infected individuals in the population, 0 recovered, and so 1 - 0.01 - 0 = 99% susceptible individuals.

To prepare for our data estimation, let's start by picking some values for β and γ from a hat (we will model them based on China's data leter on), to guarantee an epidemic ($\beta > \gamma$), and see if we can actually solve our ODE.

In [14]:
```
beta, gamma = 4.0, 1.0
```

Realistically, an individual may be in infectious contact with less or more than 4 other individuals per day, and Covid19's infectivity period is much bigger than 1 day per individual, but let's go with these numbers.

We use the library `odeint` from the package `scipy.integrate` to solve the ordinary differential equations for SIR:

In [15]:
```
from scipy.integrate import odeint

# Compute true curves
y = odeint(SIR, t=times, y0=[0.01, 0.0], args=((beta, gamma),), rtol=1e-8)
```

The data has uncertainty (or ***noise***) because of our measurement tools (aleatoric uncertainty (https://en.wikipedia.org/wiki/Uncertainty_quantification)), but also because the modeling equations are ot perfect (epistemic uncertainity (https://en.wikipedia.org/wiki/Uncertainty_quantification)).

We pick a log-normal distribution (https://en.wikipedia.org/wiki/Log-normal_distribution), a continuous probability distribution whose logarithm is *normally distributed* (thus, if the random variable X is log-normally distributed, then Y = ln(X) has a normal distribution) as our observations pdf, with a mean equal to the log of `y`. The idea for picking this distribution was Dimitri Pananos' idea, on a blog post highlighting new functionality for `PyMC3`. That post can be found in the references section in this notebook.

> **NOTE**: Recall that taking the logarithm of a histogram of a dataset makes the histogram look "*prettier*" (可爱极了), i.e. closer to a normal distribution, so that we can indeed model it as a normal distribution. So this is a common preprocessing operation in data science.

So, these may be our observations: Data from the true curves for I and R with errors 10% and 10% respectively

In [16]:
```
1 yobs = np.random.lognormal(mean=np.log(y[1::]), sigma=[0.1, 0.1])
```

Let's plot these observations as datapoints on top of the exact values for S and I. Note that R is not observed but estimated from the exact curves for S and I:

In [17]:
```
1 yobs
```

Out[17]:
```
array([[0.02152122, 0.00371614],
       [0.03747861, 0.01306067],
       [0.08530631, 0.02434567],
       [0.1248505 , 0.06808261],
       [0.24997992, 0.09022543],
       [0.34884741, 0.17233984],
       [0.38146798, 0.27370836],
       [0.40200712, 0.37542981],
       [0.34821233, 0.46835361],
       [0.33305082, 0.5562813 ],
       [0.34586895, 0.69002402],
       [0.26367482, 0.69526107],
       [0.1971184 , 0.77343443],
       [0.15749306, 0.96260052],
       [0.11611039, 0.82150439],
       [0.1019318 , 0.81480331],
       [0.08411772, 0.97402679],
       [0.06459255, 0.84040777],
       [0.05251203, 0.88166701]])
```

In [18]:
```
1 yobs[:,0]
```

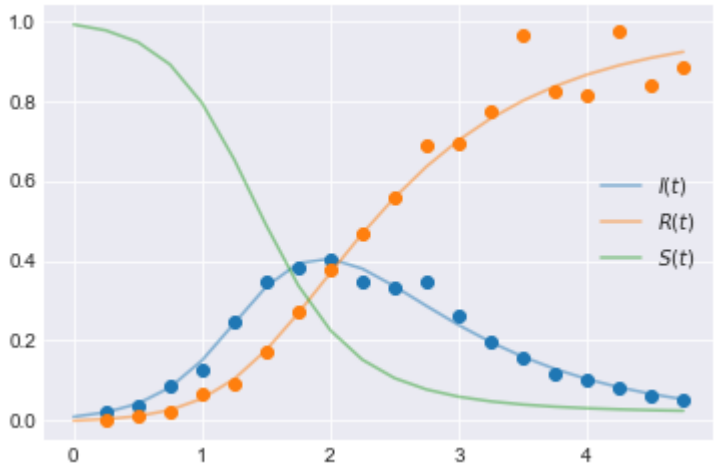
Out[18]:
```
array([0.02152122, 0.03747861, 0.08530631, 0.1248505 , 0.24997992,
       0.34884741, 0.38146798, 0.40200712, 0.34821233, 0.33305082,
       0.34586895, 0.26367482, 0.1971184 , 0.15749306, 0.11611039,
       0.1019318 , 0.08411772, 0.06459255, 0.05251203])
```

In [19]:
```
1 yobs[:,1]
```

Out[19]:
```
array([0.00371614, 0.01306067, 0.02434567, 0.06808261, 0.09022543,
       0.17233984, 0.27370836, 0.37542981, 0.46835361, 0.5562813 ,
       0.69002402, 0.69526107, 0.77343443, 0.96260052, 0.82150439,
       0.81480331, 0.97402679, 0.84040777, 0.88166701])
```

In [20]:
```
plt.plot(times[1::],yobs, marker='o', linestyle='none')
plt.plot(times, y[:,0], color='C0', alpha=0.5, label=f'$I(t)$')
plt.plot(times, y[:,1], color ='C1', alpha=0.5, label=f'$R(t)$')
plt.plot(times, susceptible(y[:,0], y[:,1]), color ='C2', alpha=0.5, label=f
plt.legend()
```

Out[20]: <matplotlib.legend.Legend at 0x1e960e56390>



Driven by rising infections, susceptible population diminishes due to acquired immunity, which controls the infection and brings it down to zero after it reaches a peak of 40% of the population, for the values of the parameters β and γ that we picked. Very sobering.

Our China data though is only up to timepoint 2 in the graph above. We have *yet* to observe the infection to come down. All we observe in China is that *the number of new infections is coming down to almost zero*, which means we are approaching the peak of the Infected curve $I(t)$.

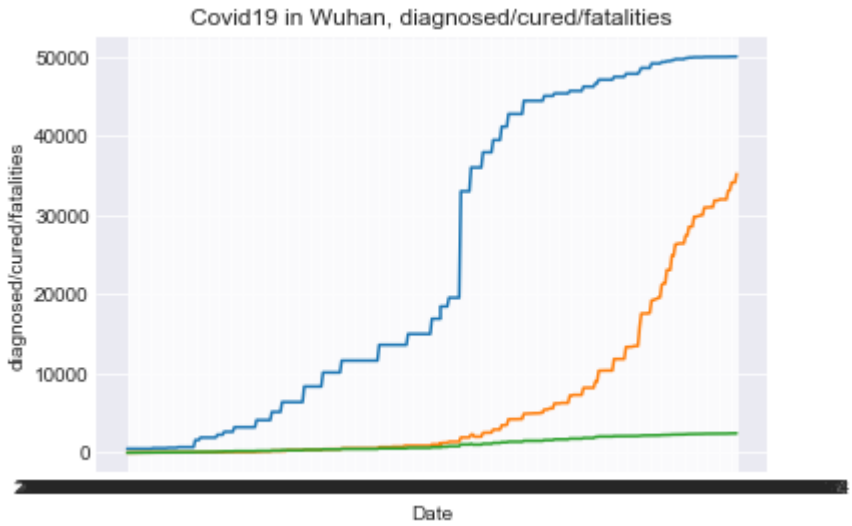
Modeling with China's data

Let's model China's infection data with Bayesian estimation in order to figure out the critical Chinese β and γ parameters. Based on social comparisons, we may then estimate our own β parameter, so that will give us a relative disease R_0, which will tell us how *dangerous* Covid19 is in our own society.

> Flyers about how the flu kills more people than Covid19 abounded around campus, and are probably partly responsible for the complacent response to the epidemic in Western societies. Let's see how much *more dangerous* than the flu it really is.

Let's plot Wuhan, Hubei's infected and cured populations once again:

In [41]:
```
plt.plot(covid19_wuhan3.city_confirmedCount)
plt.plot(covid19_wuhan3.city_curedCount)
plt.plot(covid19_wuhan3.city_deadCount)

plt.title('Covid19 in Wuhan, diagnosed/cured/fatalities')
plt.ylabel('diagnosed/cured/fatalities')
plt.xlabel('Date')
plt.grid(True)
```



Extrapolating Chinese data to the fizzling out of the pandemic

We will start with an upper range for R_0, by assuming that the virus has essentially impacted the entire city's population.

We have to complete the Chinese data since it only goes up to the peak. We will assume that what we hear from China is correct: The epidemic has peaked and is petering out, thanks to the draconian quarantine measures undertaken. So let's just mirror out the infected data so that it decreases at the same rate, and let's just continue the progression of the cured curve.

In [21]:
```
covid19_wuhan = covid19_china[covid19_china['cityEnglishName']=='Wuhan']
```

In [22]:
```
covid19_wuhan.drop_duplicates('updateTime', inplace = True)
```

```
d:\Anaconda3.5.1\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy (http://pandas.pydat
a.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  """Entry point for launching an IPython kernel.
```

In [23]:
```
1 covid19_wuhan2 = covid19_wuhan.sort_values(by=['updateTime'])
```

In [24]:
```
1 covid19_wuhan3 = covid19_wuhan2.set_index('updateTime')
```

In [25]:
```
1 covid19_wuhan4 = covid19_wuhan3.copy()
```

In [26]:
```
1 covid19_wuhan4
```

Out[26]:

updateTime	provinceName	provinceEnglishName	cityName	cityEnglishName	province_confirmedC
2020-01-24 09:47:38.698	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:48:39.253	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:49:39.772	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:50:40.357	湖北省	Hubei	武汉	Wuhan	
2020-01-24 11:49:48.584	湖北省	Hubei	武汉	Wuhan	
...	...	...	...	...	
2020-03-11 18:49:50.822	湖北省	Hubei	武汉	Wuhan	6
2020-03-12 09:21:37.890	湖北省	Hubei	武汉	Wuhan	6
2020-03-12 14:27:43.370	湖北省	Hubei	武汉	Wuhan	6
2020-03-12 17:33:24.482	湖北省	Hubei	武汉	Wuhan	6
2020-03-13 11:08:59.974	湖北省	Hubei	武汉	Wuhan	6

359 rows × 10 columns

In [27]:
```
1 covid19_wuhan4.index
```

Out[27]:
```
Index(['2020-01-24 09:47:38.698', '2020-01-24 09:48:39.253',
       '2020-01-24 09:49:39.772', '2020-01-24 09:50:40.357',
       '2020-01-24 11:49:48.584', '2020-01-24 12:23:06.852',
       '2020-01-24 12:24:07.515', '2020-01-24 12:51:13.280',
       '2020-01-24 13:15:28.296', '2020-01-24 13:16:28.914',
       ...
       '2020-03-10 16:30:50.904', '2020-03-10 16:43:54.036',
       '2020-03-10 23:01:19.213', '2020-03-11 09:12:44.046',
       '2020-03-11 13:56:46.112', '2020-03-11 18:49:50.822',
       '2020-03-12 09:21:37.890', '2020-03-12 14:27:43.370',
       '2020-03-12 17:33:24.482', '2020-03-13 11:08:59.974'],
      dtype='object', name='updateTime', length=359)
```

Let's regularize the time index.

In [29]:
```
1 from datetime import datetime
2 datetime.strptime('2020-03-12 17:33:24.482', '%Y-%m-%d %H:%M:%S.%f')
```

Out[29]:
```
datetime.datetime(2020, 3, 12, 17, 33, 24, 482000)
```

In [30]:
```
covid19_wuhan4.index = covid19_wuhan4.index.map(
    lambda x: datetime.strptime(x, '%Y-%m-%d %H:%M:%S.%f').replace(second=0))
covid19_wuhan4
```

Out[30]:

updateTime	provinceName	provinceEnglishName	cityName	cityEnglishName	province_confirm
2020-01-24 09:47:00.698	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:48:00.253	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:49:00.772	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:50:00.357	湖北省	Hubei	武汉	Wuhan	
2020-01-24 11:49:00.584	湖北省	Hubei	武汉	Wuhan	
...	...	...	...	...	
2020-03-11 18:49:00.822	湖北省	Hubei	武汉	Wuhan	
2020-03-12 09:21:00.890	湖北省	Hubei	武汉	Wuhan	
2020-03-12 14:27:00.370	湖北省	Hubei	武汉	Wuhan	
2020-03-12 17:33:00.482	湖北省	Hubei	武汉	Wuhan	
2020-03-13 11:08:00.974	湖北省	Hubei	武汉	Wuhan	

359 rows × 10 columns

In [31]:
```
covid19_wuhan4.index = covid19_wuhan4.index.map(
    lambda x: x.replace(microsecond=0))
covid19_wuhan4
```

Out[31]:

updateTime	provinceName	provinceEnglishName	cityName	cityEnglishName	province_confirme
2020-01-24 09:47:00	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:48:00	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:49:00	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:50:00	湖北省	Hubei	武汉	Wuhan	
2020-01-24 11:49:00	湖北省	Hubei	武汉	Wuhan	
...	...	...	...	...	
2020-03-11 18:49:00	湖北省	Hubei	武汉	Wuhan	

In [32]:
```
1 covid19_wuhan4
```

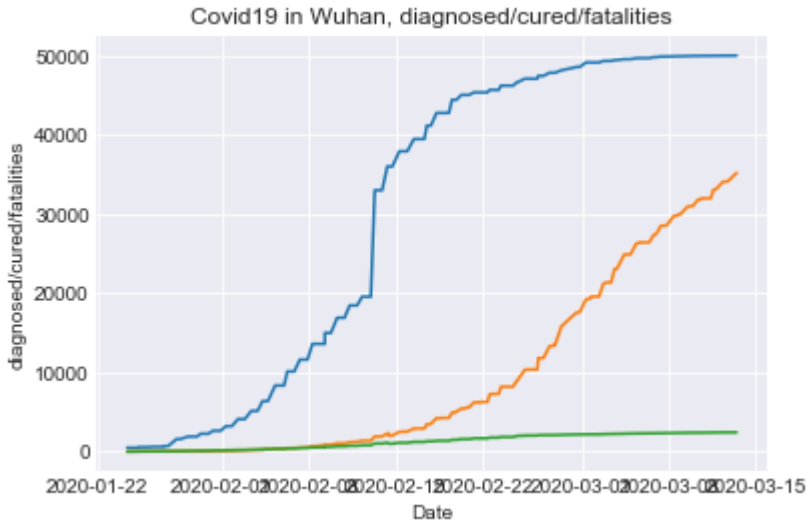
Out[32]:

updateTime	provinceName	provinceEnglishName	cityName	cityEnglishName	province_confirmedC
2020-01-24 09:47:00	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:48:00	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:49:00	湖北省	Hubei	武汉	Wuhan	
2020-01-24 09:50:00	湖北省	Hubei	武汉	Wuhan	
2020-01-24 11:49:00	湖北省	Hubei	武汉	Wuhan	
...	...	...	...	...	
2020-03-11 18:49:00	湖北省	Hubei	武汉	Wuhan	6
2020-03-12 09:21:00	湖北省	Hubei	武汉	Wuhan	6
2020-03-12 14:27:00	湖北省	Hubei	武汉	Wuhan	6
2020-03-12 17:33:00	湖北省	Hubei	武汉	Wuhan	6
2020-03-13 11:08:00	湖北省	Hubei	武汉	Wuhan	6

359 rows × 10 columns

In [33]:
```
plt.plot(covid19_wuhan4.city_confirmedCount)
plt.plot(covid19_wuhan4.city_curedCount)
plt.plot(covid19_wuhan4.city_deadCount)

plt.title('Covid19 in Wuhan, diagnosed/cured/fatalities')
plt.ylabel('diagnosed/cured/fatalities')
plt.xlabel('Date')
plt.grid(True)
```

```
C:\Users\Dino\AppData\Roaming\Python\Python36\site-packages\pandas\plotting\_matplotlib\converter.py:103: FutureWarning: Using an implicitly registered datetime converter for a matplotlib plotting method. The converter was registered by pandas on import. Future versions of pandas will require you to explicitly register matplotlib converters.

To register the converters:
        >>> from pandas.plotting import register_matplotlib_converters
        >>> register_matplotlib_converters()
  warnings.warn(msg, FutureWarning)
```



Let's drop some more columns:

In [34]:
```
drop_cols = ['provinceName', 'provinceEnglishName', 'cityName', 'cityEnglish

covid19_wuhan5 = covid19_wuhan4.drop(drop_cols, axis=1)
covid19_wuhan5
```

Out[34]:

updateTime	city_confirmedCount	city_curedCount	city_deadCount
2020-01-24 09:47:00	495	0	0
2020-01-24 09:48:00	495	0	0
2020-01-24 09:49:00	495	0	0
2020-01-24 09:50:00	495	0	0
2020-01-24 11:49:00	495	0	0
...	...	...	...
2020-03-11 18:49:00	49978	33117	2423
2020-03-12 09:21:00	49986	34094	2430
2020-03-12 14:27:00	49986	34094	2430
2020-03-12 17:33:00	49986	34096	2430
2020-03-13 11:08:00	49991	35197	2436

359 rows × 3 columns

Let's tally up the deceased amongst the recovered, to keep with the SIR model.

In [35]:
```
covid19_wuhan5['city_recoveredCount'] = covid19_wuhan5.city_curedCount + cov
covid19_wuhan5
```

Out[35]:

updateTime	city_confirmedCount	city_curedCount	city_deadCount	city_recoveredCount
2020-01-24 09:47:00	495	0	0	0
2020-01-24 09:48:00	495	0	0	0
2020-01-24 09:49:00	495	0	0	0
2020-01-24 09:50:00	495	0	0	0
2020-01-24 11:49:00	495	0	0	0
...	...	...	...	...
2020-03-11 18:49:00	49978	33117	2423	35540

In [36]:
```
drop_cols = ['city_curedCount', 'city_deadCount']
covid19_wuhan6 = covid19_wuhan5.drop(drop_cols, axis=1)
covid19_wuhan6
```

Out[36]:

updateTime	city_confirmedCount	city_recoveredCount
2020-01-24 09:47:00	495	0
2020-01-24 09:48:00	495	0
2020-01-24 09:49:00	495	0
2020-01-24 09:50:00	495	0
2020-01-24 11:49:00	495	0
...	...	...
2020-03-11 18:49:00	49978	35540
2020-03-12 09:21:00	49986	36524
2020-03-12 14:27:00	49986	36524
2020-03-12 17:33:00	49986	36526
2020-03-13 11:08:00	49991	37633

359 rows × 2 columns

Now let's extend an additional 3 months, where the epidemic (hopefully) tapers out:

In [37]:
```
from datetime import datetime
from dateutil.relativedelta import relativedelta

date_after_month = covid19_wuhan5.index[0] + relativedelta(months=2, days=1)
date_after_month
```

Out[37]:
```
Timestamp('2020-03-25 09:47:00')
```

In [38]:
```
extend_3_months = [x + relativedelta(months=1, days=19) for x in covid19_wuh
extend_3_months
```

Out[38]:
```
[Timestamp('2020-03-14 09:47:00'),
 Timestamp('2020-03-14 09:48:00'),
 Timestamp('2020-03-14 09:49:00'),
 Timestamp('2020-03-14 09:50:00'),
 Timestamp('2020-03-14 11:49:00'),
 Timestamp('2020-03-14 12:23:00'),
 Timestamp('2020-03-14 12:24:00'),
 Timestamp('2020-03-14 12:51:00'),
 Timestamp('2020-03-14 13:15:00'),
 Timestamp('2020-03-14 13:16:00'),
 Timestamp('2020-03-14 13:22:00'),
 Timestamp('2020-03-14 16:11:00'),
 Timestamp('2020-03-14 16:48:00'),
 Timestamp('2020-03-14 17:30:00'),
 Timestamp('2020-03-15 07:48:00'),
 Timestamp('2020-03-15 08:03:00'),
 Timestamp('2020-03-15 08:08:00'),
 Timestamp('2020-03-15 08:11:00'),
 Timestamp('2020-03-15 08:12:00'),
```

Let;s do the simplest possible extrapolation: We will just mirror the infected curve, and linearly extrapolate the recovered curve.

In [39]:
```
city_confirmedCount_mirror = covid19_wuhan5.city_confirmedCount.tolist()
city_confirmedCount_mirror.reverse()
city_confirmedCount_mirror
```

Out[39]:
```
[49991,
 49986,
 49986,
 49986,
 49978,
 49978,
 49965,
 49965,
 49965,
 49965,
 49965,
 49965,
 49965,
 49965,
 49948,
 49948,
 49948,
 49948,
 49948,
```

In [40]:
```
covid19_wuhan6.city_recoveredCount.tail()
```

Out[40]:
```
updateTime
2020-03-11 18:49:00    35540
2020-03-12 09:21:00    36524
2020-03-12 14:27:00    36524
2020-03-12 17:33:00    36526
2020-03-13 11:08:00    37633
Name: city_recoveredCount, dtype: int64
```

In [41]:
```
delta_extrapolant = 37633 - 36524
delta_extrapolant
```

Out[41]:
```
1109
```

In [42]:
```
delta_extrapolant = 300
```

In [43]:
```
len(extend_3_months)
```

Out[43]:
```
359
```

In [44]:
```
city_recoveredCount_extrapolate = [37633 + delta_extrapolant + i*delta_extra
city_recoveredCount_extrapolate
```

Out[44]:
```
[37933,
 38233,
 38533,
 38833,
 39133,
 39433,
 39733,
 40033,
 40333,
 40633,
 40933,
 41233,
 41533,
 41833,
 42133,
 42433,
 42733,
 43033,
 43333,
```

In [45]:
```
covid19_wuhan_extension = pd.DataFrame(list(zip(city_confirmedCount_mirror,
                                    index = extend_3_months, columns =['ci
covid19_wuhan_extension
```

Out[45]:

	city_confirmedCount	city_recoveredCount
2020-03-14 09:47:00	49991	37933
2020-03-14 09:48:00	49986	38233
2020-03-14 09:49:00	49986	38533
2020-03-14 09:50:00	49986	38833
2020-03-14 11:49:00	49978	39133
...	...	...
2020-04-30 18:49:00	495	144133
2020-05-01 09:21:00	495	144433
2020-05-01 14:27:00	495	144733
2020-05-01 17:33:00	495	145033
2020-05-02 11:08:00	495	145333

359 rows × 2 columns

In [46]:
```
covid19_wuhan_extension = pd.concat([covid19_wuhan6, covid19_wuhan_extension
covid19_wuhan_extension
```

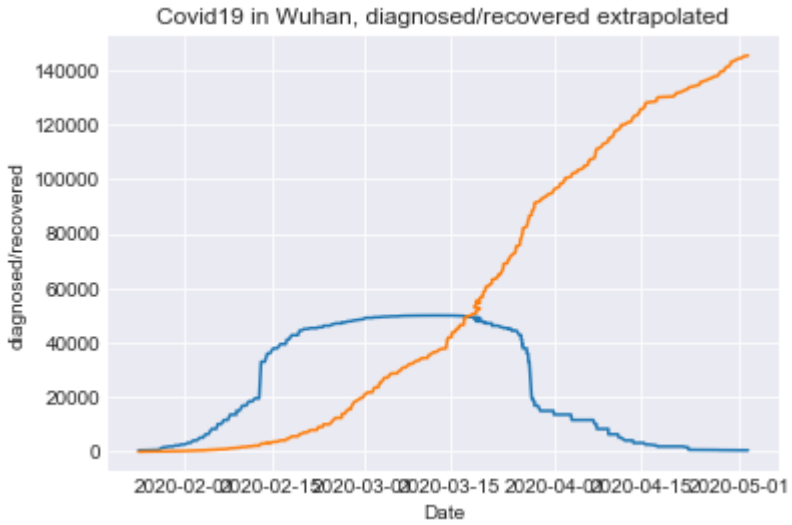
Out[46]:

	city_confirmedCount	city_recoveredCount
2020-01-24 09:47:00	495	0
2020-01-24 09:48:00	495	0
2020-01-24 09:49:00	495	0
2020-01-24 09:50:00	495	0
2020-01-24 11:49:00	495	0
...	...	...
2020-04-30 18:49:00	495	144133
2020-05-01 09:21:00	495	144433
2020-05-01 14:27:00	495	144733
2020-05-01 17:33:00	495	145033
2020-05-02 11:08:00	495	145333

718 rows × 2 columns

This yields the following curves:

In [47]:
```
plt.plot(covid19_wuhan_extension.city_confirmedCount)
plt.plot(covid19_wuhan_extension.city_recoveredCount)

plt.title('Covid19 in Wuhan, diagnosed/recovered extrapolated')
plt.ylabel('diagnosed/recovered')
plt.xlabel('Date')
plt.grid(True)
```



In [48]:
```
1 y_wuhan = covid19_wuhan_extension.to_numpy()
2 y_wuhan
```

Out[48]:
```
array([[   495,      0],
       [   495,      0],
       [   495,      0],
       ...,
       [   495, 144733],
       [   495, 145033],
       [   495, 145333]], dtype=int64)
```

In [49]:
```
1 y_wuhan.shape
```

Out[49]:
```
(718, 2)
```

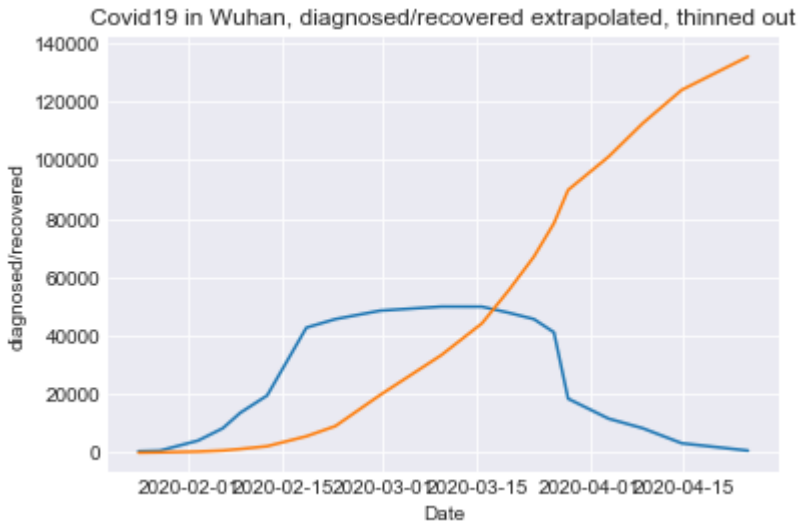
Let's reduce the size of our data by an order of magnitude:

In [50]:
```
1 covid19_wuhan_extension3 = covid19_wuhan_extension[::38]  # Selects every 38
```

In [51]:
```
1 len(covid19_wuhan_extension3)
```

Out[51]:
```
19
```

In [52]:
```
1 plt.plot(covid19_wuhan_extension3.city_confirmedCount)
2 plt.plot(covid19_wuhan_extension3.city_recoveredCount)
3
4 plt.title('Covid19 in Wuhan, diagnosed/recovered extrapolated, thinned out')
5 plt.ylabel('diagnosed/recovered')
6 plt.xlabel('Date')
7 plt.grid(True)
```



In [53]:
```
y_wuhan3 = covid19_wuhan_extension3.to_numpy()
y_wuhan3
```

Out[53]:
```
array([[   495,      0],
       [   698,    105],
       [  4109,    362],
       [  8351,    736],
       [ 13603,   1243],
       [ 19558,   2199],
       [ 42752,   5602],
       [ 45660,   9066],
       [ 48557,  19737],
       [ 49948,  33374],
       [ 49912,  44233],
       [ 47824,  55633],
       [ 45660,  67033],
       [ 41152,  78433],
       [ 18454,  89833],
       [ 11618, 101233],
       [  8351, 112633],
       [  3215, 124033],
       [   698, 135433]], dtype=int64)
```

In [54]:
```
y_wuhan3.shape
```

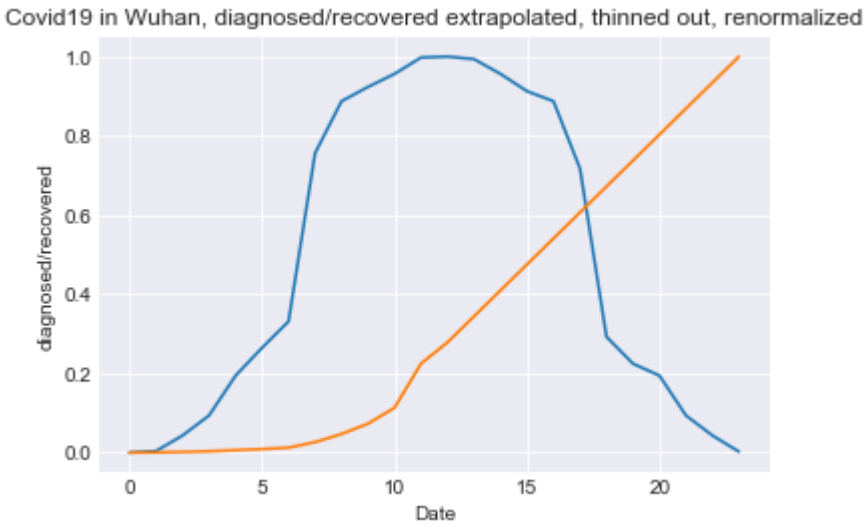
Out[54]:
```
(19, 2)
```

Now let's renormalize.

We might be tempted to do the following, but this does not respect the relative sizes of the infected and recovered curves:

In [188]:
```
from sklearn import preprocessing

x = covid19_wuhan_extension3.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
covid19_wuhan_extension4 = pd.DataFrame(x_scaled)
```

In [189]:
```
plt.plot(covid19_wuhan_extension4[0])
plt.plot(covid19_wuhan_extension4[1])

plt.title('Covid19 in Wuhan, diagnosed/recovered extrapolated, thinned out,
plt.ylabel('diagnosed/recovered')
plt.xlabel('Date')
plt.grid(True)
```



How to extrapolate the recovered curve?

Since we're looking for an upper bound for R_0, let's make the ***naive*** assumption that 140,000 is the population of Wuhan that is susceptible to the virus (137233 is the max of the recovered column). This is ***most*** probably ***not true***, since the population of the city of Wuhan (https://en.wikipedia.org/wiki/Wuhan), the 9th most populous Chinese city, is closer to 11 million. But probably not the entirety of that number is as exposed as the city center. I think the urban city center (9 million) may be a more appropriate statistic.

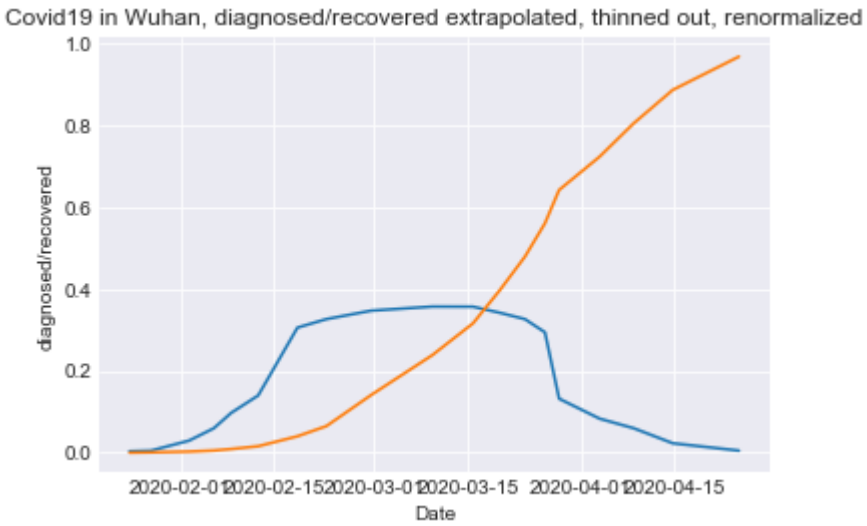
Let's assume this number for now and proceed. We'll should do another analysis with a higher number.

In [55]:
```
1 covid19_wuhan_extension4 = covid19_wuhan_extension3.div(140000)
2 covid19_wuhan_extension4
```

Out[55]:

	city_confirmedCount	city_recoveredCount
2020-01-24 09:47:00	0.003536	0.000000
2020-01-27 15:16:00	0.004986	0.000750
2020-02-02 07:28:00	0.029350	0.002586
2020-02-06 00:43:00	0.059650	0.005257
2020-02-08 15:01:00	0.097164	0.008879
2020-02-12 16:00:00	0.139700	0.015707
2020-02-18 13:05:00	0.305371	0.040014
2020-02-22 20:55:00	0.326143	0.064757
2020-02-29 14:14:00	0.346836	0.140979
2020-03-09 18:07:00	0.356771	0.238386
2020-03-15 20:07:00	0.356514	0.315950
2020-03-19 21:03:00	0.341600	0.397379
2020-03-23 14:43:00	0.326143	0.478807
2020-03-26 14:31:00	0.293943	0.560236
2020-03-28 17:30:00	0.131814	0.641664
2020-04-03 19:14:00	0.082986	0.723093
2020-04-08 21:36:00	0.059650	0.804521
2020-04-14 17:50:00	0.022964	0.885950
2020-04-24 15:47:00	0.004986	0.967379

In [56]:
```
plt.plot(covid19_wuhan_extension4.city_confirmedCount)
plt.plot(covid19_wuhan_extension4.city_recoveredCount)

plt.title('Covid19 in Wuhan, diagnosed/recovered extrapolated, thinned out,
plt.ylabel('diagnosed/recovered')
plt.xlabel('Date')
plt.grid(True)
```



In [57]:
```
y_wuhan4 = covid19_wuhan_extension4.to_numpy()
y_wuhan4
```

Out[57]:
```
array([[3.53571429e-03, 0.00000000e+00],
       [4.98571429e-03, 7.50000000e-04],
       [2.93500000e-02, 2.58571429e-03],
       [5.96500000e-02, 5.25714286e-03],
       [9.71642857e-02, 8.87857143e-03],
       [1.39700000e-01, 1.57071429e-02],
       [3.05371429e-01, 4.00142857e-02],
       [3.26142857e-01, 6.47571429e-02],
       [3.46835714e-01, 1.40978571e-01],
       [3.56771429e-01, 2.38385714e-01],
       [3.56514286e-01, 3.15950000e-01],
       [3.41600000e-01, 3.97378571e-01],
       [3.26142857e-01, 4.78807143e-01],
       [2.93942857e-01, 5.60235714e-01],
       [1.31814286e-01, 6.41664286e-01],
       [8.29857143e-02, 7.23092857e-01],
       [5.96500000e-02, 8.04521429e-01],
       [2.29642857e-02, 8.85950000e-01],
       [4.98571429e-03, 9.67378571e-01]])
```

Let's do a small fudge on the `e-04` number to get rid of pesky exponential notation:

In [58]:
```
# small fudge to get rid of exponential notation!
y_wuhan4[1,1] = 0.001
y_wuhan4
```

Out[58]:
```
array([[0.00353571, 0.        ],
       [0.00498571, 0.001     ],
       [0.02935   , 0.00258571],
       [0.05965   , 0.00525714],
       [0.09716429, 0.00887857],
       [0.1397    , 0.01570714],
       [0.30537143, 0.04001429],
       [0.32614286, 0.06475714],
       [0.34683571, 0.14097857],
       [0.35677143, 0.23838571],
       [0.35651429, 0.31595   ],
       [0.3416    , 0.39737857],
       [0.32614286, 0.47880714],
       [0.29394286, 0.56023571],
       [0.13181429, 0.64166429],
       [0.08298571, 0.72309286],
       [0.05965   , 0.80452143],
       [0.02296429, 0.88595   ],
       [0.00498571, 0.96737857]])
```

Let's do another fudge on the first row's 0.0 cell, since without it the NUTS sim crashes out with a `Bad Initial Energy` error!

In [59]:
```
# to get rid of bad initial energy
y_wuhan4[0,1] = 0.001
y_wuhan4
```

Out[59]:
```
array([[0.00353571, 0.001     ],
       [0.00498571, 0.001     ],
       [0.02935   , 0.00258571],
       [0.05965   , 0.00525714],
       [0.09716429, 0.00887857],
       [0.1397    , 0.01570714],
       [0.30537143, 0.04001429],
       [0.32614286, 0.06475714],
       [0.34683571, 0.14097857],
       [0.35677143, 0.23838571],
       [0.35651429, 0.31595   ],
       [0.3416    , 0.39737857],
       [0.32614286, 0.47880714],
       [0.29394286, 0.56023571],
       [0.13181429, 0.64166429],
       [0.08298571, 0.72309286],
       [0.05965   , 0.80452143],
       [0.02296429, 0.88595   ],
       [0.00498571, 0.96737857]])
```

In [60]:
```
1 yobs
```

Out[60]:
```
array([[0.02152122, 0.00371614],
       [0.03747861, 0.01306067],
       [0.08530631, 0.02434567],
       [0.1248505 , 0.06808261],
       [0.24997992, 0.09022543],
       [0.34884741, 0.17233984],
       [0.38146798, 0.27370836],
       [0.40200712, 0.37542981],
       [0.34821233, 0.46835361],
       [0.33305082, 0.5562813 ],
       [0.34586895, 0.69002402],
       [0.26367482, 0.69526107],
       [0.1971184 , 0.77343443],
       [0.15749306, 0.96260052],
       [0.11611039, 0.82150439],
       [0.1019318 , 0.81480331],
       [0.08411772, 0.97402679],
       [0.06459255, 0.84040777],
       [0.05251203, 0.88166701]])
```

In [67]:
```
1 y_wuhan4.shape
```

Out[67]:
```
(19, 2)
```

In [62]:
```
1 yobs.shape
```

Out[62]:
```
(19, 2)
```

Now we have our data, we are ready for our Bayesian modeling!

Bayesian modeling Chinese stats

We import `DifferentialEquation` from `PyMC3` :

In [63]:
```
1 import pymc3 as pm
2 from pymc3.ode import DifferentialEquation
3 import theano
```

Matching shapes for X (time) and Y:

In [ ]:
```
1 np.arange(0.25, 5, 0.2t).shape
```

Our epidemic model dynamics are captured thusly:

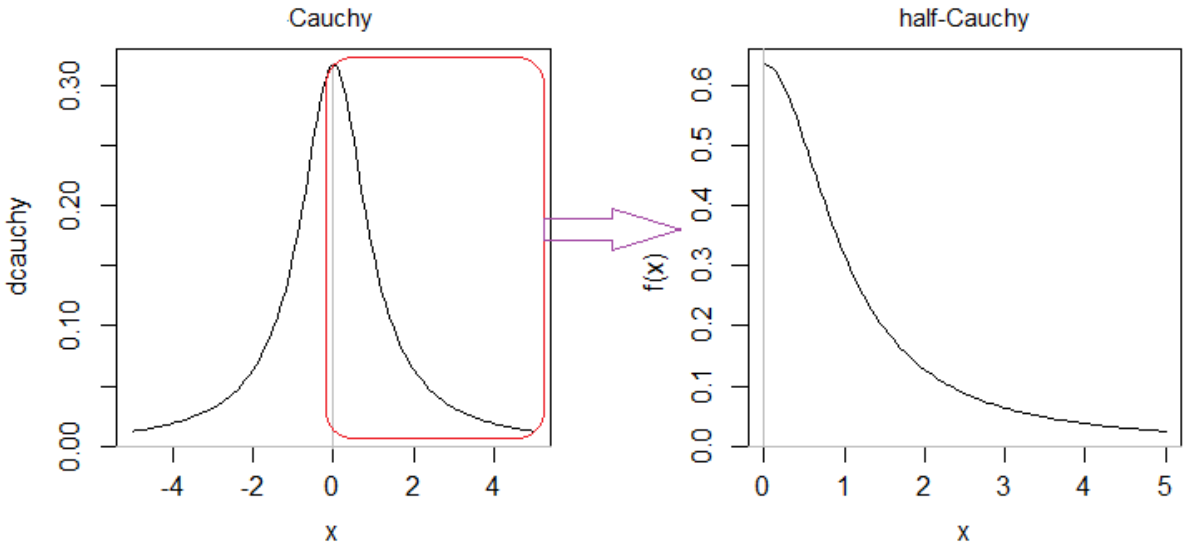
In [66]:
```
sir_model = DifferentialEquation(
    func = SIR,
    times = np.arange(0.25, 5, 0.25),
    n_states = 2,
    n_theta = 2,
    t0 = 0,
)
```

We will model observations with a log-normal distribution (https://en.wikipedia.org/wiki/Log-normal_distribution), a continuous probability density function (pdf) whose logarithm is normally distributed. Thus, if the random variable X is log-normally distributed, then Y = ln(X) has a normal distribution.

Taking the logarithm of a dataset is a prettyfier of data and is often a first target model. So we will assume that our Chinese I (infected) and R (recovered) data are log-normally distributed.

We will assume that there is noise/error in the Chinese data, so we will tack on a standard deviation. Since that quantity is never negative, it is often modelled as a Half Cauchy (https://en.wikipedia.org/wiki/Cauchy_distribution) distribution. It has an interesting history.

> **NOTE**: The half-Cauchy is heavy tailed and is regarded as fairly weakly informative. Gelman advocates for half-t priors (including the half-Cauchy) because they have better behavior for small parameter values but only regards it as weakly informative when a large scale parameter is used. See A. Gelman (2006), "Prior distributions for variance parameters in hierarchical models" Bayesian Analysis, Vol. 1, N. 3, pp. 515–533 (http://www.stat.columbia.edu/~gelman/research/published/taumain.pdf).



To model the parameters themselves, we will assume a very uninformative R_0 that is just > 1 so that we have an epidemic, a γ that also follows a log-normal distribution (so we only use one pdf type for our model, but hey, *you're free to try another one*!), and a β that is deterministically given by $\gamma * R_0$.

The data whose likelihood we're trying to model is given by the `sir_model` above: our function `SIR` given even further above, a time discretization from 0.25 to 5, starting from time 0, with 2D data (I, R) for Wuhan, Hubei, and 2D parameters (`R0` and `gama`). We pick Wuhan because it gives us the worst possible case since it's the epicenter of the disease.

Let's start our simulations with results we already know, to double-check our code. We will estimate the parameters for our `yobs` data, which was artifically created by picking some parameter values from the hat.

In [214]:
```
import arviz as az

with pm.Model() as model5:
    sigma = pm.HalfCauchy('sigma', 1, shape=2)

    # R0 is bounded below by 1 because we see an epidemic has occured
    R0 = pm.Bound(pm.Normal, lower=1)('R0', 2, 3)
    gama = pm.Lognormal('gama', pm.math.log(2), 2)
    beta = pm.Deterministic('beta', gama*R0)

    sir_curves = sir_model(y0=[0.01, 0.0], theta=[beta, gama])

    # data likelihood
    #Y = pm.Lognormal('Y', mu=pm.math.log(sir_curves), sd=sigma, observed=y_
    Y = pm.Lognormal('Y', mu=pm.math.log(sir_curves), sd=sigma, observed=yob

    prior = pm.sample_prior_predictive()
    trace = pm.sample(2000, tune=1000, target_accept=0.9, cores=1)
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Sequential sampling (2 chains in 1 job)
NUTS: [gama, R0, sigma]
Sampling chain 0, 0 divergences: 100%|██████████| 3000/3000 [44:43<00:00,  1.12
it/s]
Sampling chain 1, 0 divergences: 100%|██████████| 3000/3000 [39:40<00:00,  1.26
it/s]
```

Let's generate data from the model using parameters sampled from the posterior distribution:

In [218]:
```
with model5:
    posterior_predictive = pm.sample_posterior_predictive(trace)
```

```
100%|██████████| 4000/4000 [04:11<00:00, 15.92it/s]
```

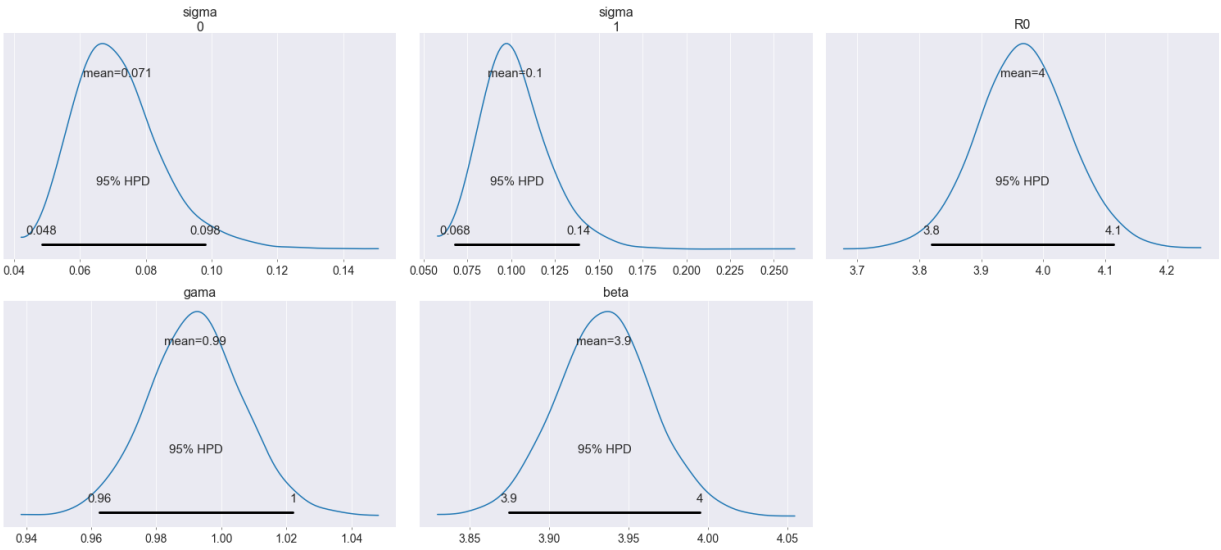
Let's create some data:

In [219]:
```
with model5:
    data = az.from_pymc3(trace=trace, prior=prior, posterior_predictive=post
```

Let's look at results:

In [220]:
```
1 az.plot_posterior(data, round_to=2, credible_interval=0.95)
```

Out[220]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x0000025A080C2F28>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000025A0F2D0D30>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000025A0EC86128>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000025A12F6E748>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000025A0200AEF0>],
      dtype=object)
```



As can be seen from the posterior plots, β is well estimated (4) by leveraging estimation of the non-dimensional parameter R_0 (3.9), and of γ (1).

> **NOTE**: In general, the nonlinearity at the heart of epidemic models (e.g., the mass-action βSI term) can make the numerical work of model fitting very sensitive: Any given realization of a model can move away from the most likely model in a hurry. There may be only a very *narrow* band of possible parameter combinations that give reasonable estimates for your model and it is difficult to figure out a priori where that band might lie. In other words, don't expect MCMC-estimation to easily converge.

It is always a good strategy to use a *different model* to see if it yields similar results (from [here (https://en.wikipedia.org/wiki/List_of_probability_distributions)](https://en.wikipedia.org/wiki/List_of_probability_distributions)) (well, not really in this case since we *already* verified we obtain the right parameters, but don't forget that we are just getting *ready* for CoVid19 simulations). I immediately thought of the Poisson distribution, a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time or space if these events occur with a known constant mean rate and independently of the time since the last event. After all, we are talking about discrete events: infections, recoveries. But the Poisson distribution has a *single* parameter, and this does not allow me to inject any *noise* in the observations.

Now that we're confident of our model, let's repeat the computation, with `wuhan4` data!

In [68]:
```
with pm.Model() as model7:
    sigma = pm.HalfCauchy('sigma', 1, shape=2)

    # R0 is bounded below by 1 because we see an epidemic has occured
    R0 = pm.Bound(pm.Normal, lower=1)('R0', 2, 3)
    gama = pm.Lognormal('gama', pm.math.log(2), 2)
    beta = pm.Deterministic('beta', gama*R0)

    sir_curves = sir_model(y0=[0.01, 0.0], theta=[beta, gama])

    # data likelihood
    Y = pm.Lognormal('Y', mu=pm.math.log(sir_curves), sd=sigma, observed=y_w

    prior = pm.sample_prior_predictive()
    trace2 = pm.sample(2000, tune=1000, target_accept=0.9, cores=1)
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Sequential sampling (2 chains in 1 job)
NUTS: [gama, R0, sigma]
Sampling chain 0, 0 divergences: 100%|██████████| 3000/3000 [50:01<00:00,  1.00
s/it]
Sampling chain 1, 0 divergences: 100%|██████████| 3000/3000 [45:43<00:00,  1.09
it/s]
The number of effective samples is smaller than 25% for some parameters.
```

Let's generate data from the model using parameters sampled from the posterior distribution:

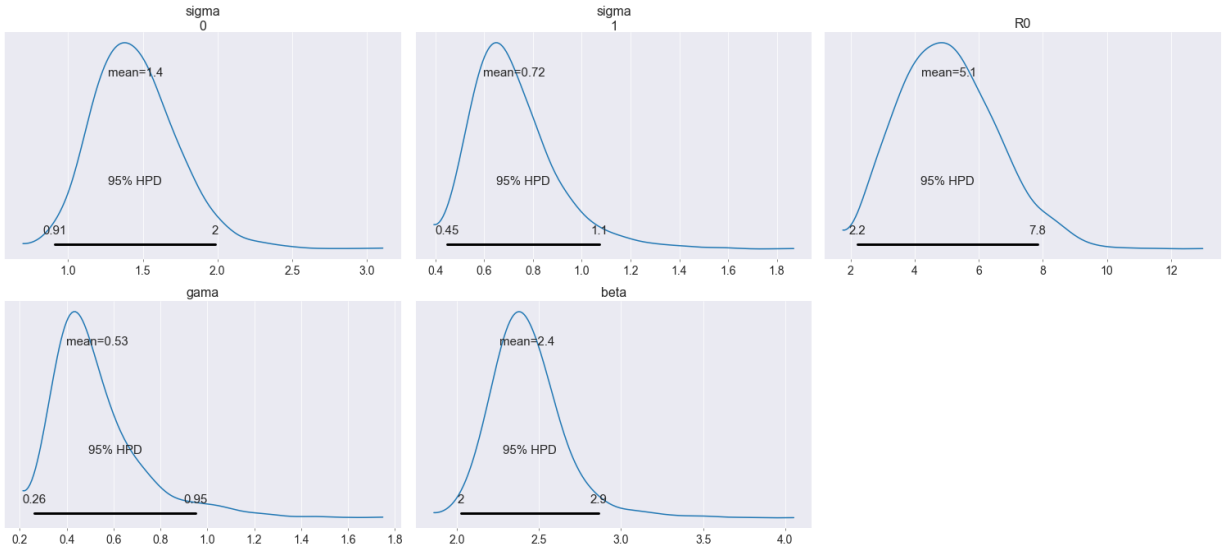
In [69]:
```
with model7:
    posterior_predictive2 = pm.sample_posterior_predictive(trace2)
```

```
100%|██████████| 4000/4000 [03:10<00:00, 20.96it/s]
```

In [71]:
```
import arviz as az

with model7:
    data2 = az.from_pymc3(trace=trace2, prior=prior, posterior_predictive=po
```

In [72]:
```
1 az.plot_posterior(data2, round_to=2, credible_interval=0.95)
```

Out[72]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x000001EB13573E48>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001EB135A1B70>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001EB4004FF60>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001EB123227B8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001EB135CEAC8>],
      dtype=object)
```



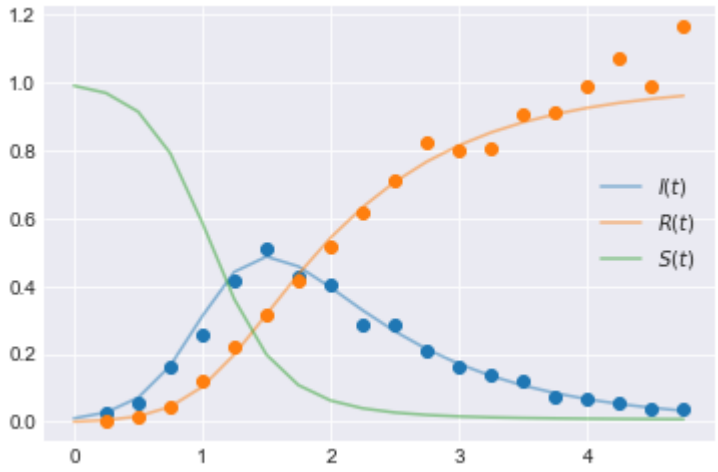
Results

We can now estimate the upper limit value of R_0 for the Coronavirus in Wuhan: $R_0 = 5.1$, for a suscpetible Wuhanese population of 150,000. That means one person infects about 5 people!

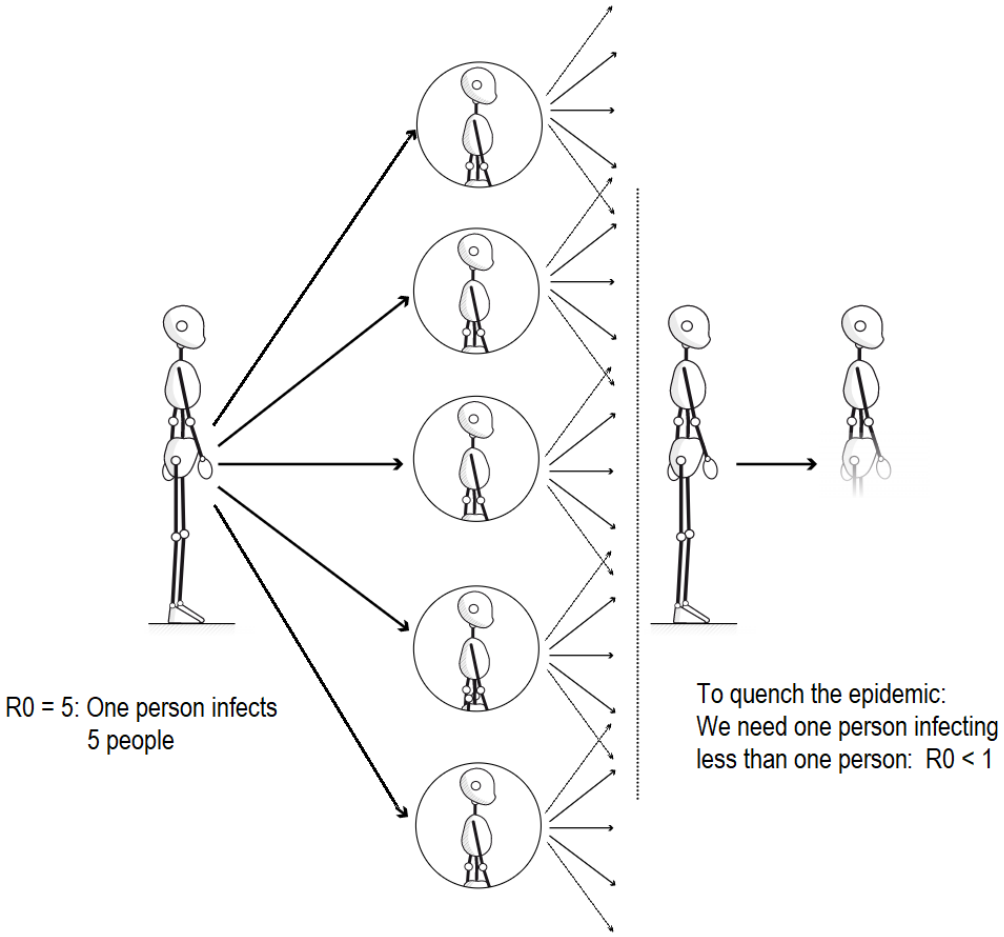
We can now rerun our ODE sim to see what percentage of the population ultimately gets infected:

In [73]:
```
times = np.arange(0, 5, 0.25)
beta2, gamma = 5.1, 1.0
y2 = odeint(SIR, t=times, y0=[0.01, 0.0], args=((beta2, gamma),), rtol=1e-8)
yobs2 = np.random.lognormal(mean=np.log(y2[1::]), sigma=[0.1, 0.1])

plt.plot(times[1::],yobs2, marker='o', linestyle='none')
plt.plot(times, y2[:,0], color='C0', alpha=0.5, label=f'$I(t)$')
plt.plot(times, y2[:,1], color ='C1', alpha=0.5, label=f'$R(t)$')
plt.plot(times, susceptible(y2[:,0], y2[:,1]), color ='C2', alpha=0.5, label
plt.legend()
```

Out[73]: <matplotlib.legend.Legend at 0x1eb13658048>



Half of the population gets infected by covid19 before there is enough immunity for the disease to fizzle. And how many people die because of preexisting comorbid conditions or because they ingest too high of a virus load? Probably a *lot*.



Interim conclusion

This estimation is based on the density of the population in Wuhan (very dense) and an *unrealistically low* total number of suscpetible people (140,000). So an R_0 of 5 is probably an *overestimation* and an upper bound of the R_0 for Covid19 in other parts of the world. Nevertheless, it underscores the infectivity of Covid19, which is much higher than the common flu. To think that I still see one-month old posters on the wall that clamor that the flu kills more people than Covid19. For reference, Ebola (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4347917/) has an R_0 of *two*, so on average, a person who has Ebola will pass it on to *two* other people (of course Ebola, *once infected*, is a much deadlier virus).

We will now move on to a more refined extrapolation, with different numbers for the suscpetible population, for future data.

Middle ground for R_0

In [99]:
```
1 covid19_wuhan = covid19_china[covid19_china['cityEnglishName']=='Wuhan']
```

In [100]:
```
1 covid19_wuhan.drop_duplicates('updateTime', inplace = True)
```

```
d:\Anaconda3.5.1\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy (http://pandas.pydat
a.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  """Entry point for launching an IPython kernel.
```

In [101]:
```
1 covid19_wuhan2 = covid19_wuhan.sort_values(by=['updateTime'])
```

In [102]:
```
1 covid19_wuhan3 = covid19_wuhan2.set_index('updateTime')
```

In [103]:
```
1 covid19_wuhan4 = covid19_wuhan3.copy()
```

In [104]:
```
1 covid19_wuhan4
```

Out[104]:

updateTime	provinceName	provinceEnglishName	province_zipCode	cityName	cityEnglishName	c
2020-01-24 09:47:38.698	湖北省	Hubei	420000	武汉	Wuhan	
2020-01-24 09:48:39.253	湖北省	Hubei	420000	武汉	Wuhan	
2020-01-24 09:49:39.772	湖北省	Hubei	420000	武汉	Wuhan	
2020-01-24 09:50:40.357	湖北省	Hubei	420000	武汉	Wuhan	
2020-01-24 11:49:48.584	湖北省	Hubei	420000	武汉	Wuhan	
...	...	...	...	...	...	...
2020-03-11 18:49:50.822	湖北省	Hubei	420000	武汉	Wuhan	
2020-03-12 09:21:37.890	湖北省	Hubei	420000	武汉	Wuhan	
2020-03-12 14:27:43.370	湖北省	Hubei	420000	武汉	Wuhan	
2020-03-12 17:33:24.482	湖北省	Hubei	420000	武汉	Wuhan	
2020-03-13 11:08:59.974	湖北省	Hubei	420000	武汉	Wuhan	

359 rows × 14 columns

In [105]:
```
1 covid19_wuhan4.index
```

Out[105]:
```
Index(['2020-01-24 09:47:38.698', '2020-01-24 09:48:39.253',
       '2020-01-24 09:49:39.772', '2020-01-24 09:50:40.357',
       '2020-01-24 11:49:48.584', '2020-01-24 12:23:06.852',
       '2020-01-24 12:24:07.515', '2020-01-24 12:51:13.280',
       '2020-01-24 13:15:28.296', '2020-01-24 13:16:28.914',
       ...
       '2020-03-10 16:30:50.904', '2020-03-10 16:43:54.036',
       '2020-03-10 23:01:19.213', '2020-03-11 09:12:44.046',
       '2020-03-11 13:56:46.112', '2020-03-11 18:49:50.822',
       '2020-03-12 09:21:37.890', '2020-03-12 14:27:43.370',
       '2020-03-12 17:33:24.482', '2020-03-13 11:08:59.974'],
      dtype='object', name='updateTime', length=359)
```

In [106]:
```
1 from datetime import datetime
2 datetime.strptime('2020-03-12 17:33:24.482', '%Y-%m-%d %H:%M:%S.%f')
```

Out[106]:
```
datetime.datetime(2020, 3, 12, 17, 33, 24, 482000)
```

In [107]:
```
covid19_wuhan4.index = covid19_wuhan4.index.map(
    lambda x: datetime.strptime(x, '%Y-%m-%d %H:%M:%S.%f').replace(second=0))
covid19_wuhan4
```

Out[107]:

updateTime	provinceName	provinceEnglishName	province_zipCode	cityName	cityEnglishName	c
2020-01-24 09:47:00.698	湖北省	Hubei	420000	武汉	Wuhan	
2020-01-24 09:48:00.253	湖北省	Hubei	420000	武汉	Wuhan	
2020-01-24 09:49:00.772	湖北省	Hubei	420000	武汉	Wuhan	
2020-01-24 09:50:00.357	湖北省	Hubei	420000	武汉	Wuhan	
2020-01-24 11:49:00.584	湖北省	Hubei	420000	武汉	Wuhan	
...	...	...	...	...	...	
2020-03-11 18:49:00.822	湖北省	Hubei	420000	武汉	Wuhan	
2020-03-12 09:21:00.890	湖北省	Hubei	420000	武汉	Wuhan	
2020-03-12 14:27:00.370	湖北省	Hubei	420000	武汉	Wuhan	
2020-03-12 17:33:00.482	湖北省	Hubei	420000	武汉	Wuhan	
2020-03-13 11:08:00.974	湖北省	Hubei	420000	武汉	Wuhan	

359 rows × 14 columns

In [108]:
```
covid19_wuhan4.index = covid19_wuhan4.index.map(
    lambda x: x.replace(microsecond=0))
covid19_wuhan4
```

Out[108]:

updateTime	provinceName	provinceEnglishName	province_zipCode	cityName	cityEnglishName	c
2020-01-24 09:47:00	湖北省	Hubei	420000	武汉	Wuhan	
2020-01-24 09:48:00	湖北省	Hubei	420000	武汉	Wuhan	
2020-01-24 09:49:00	湖北省	Hubei	420000	武汉	Wuhan	
2020-01-24 09:50:00	湖北省	Hubei	420000	武汉	Wuhan	
2020-01-24 11:49:00	湖北省	Hubei	420000	武汉	Wuhan	
...	...	...	...	...	...	...
2020-03-11 18:49:00	湖北省	Hubei	420000	武汉	Wuhan	
2020-03-12 09:21:00	湖北省	Hubei	420000	武汉	Wuhan	
2020-03-12 14:27:00	湖北省	Hubei	420000	武汉	Wuhan	
2020-03-12 17:33:00	湖北省	Hubei	420000	武汉	Wuhan	
2020-03-13 11:08:00	湖北省	Hubei	420000	武汉	Wuhan	

359 rows × 14 columns

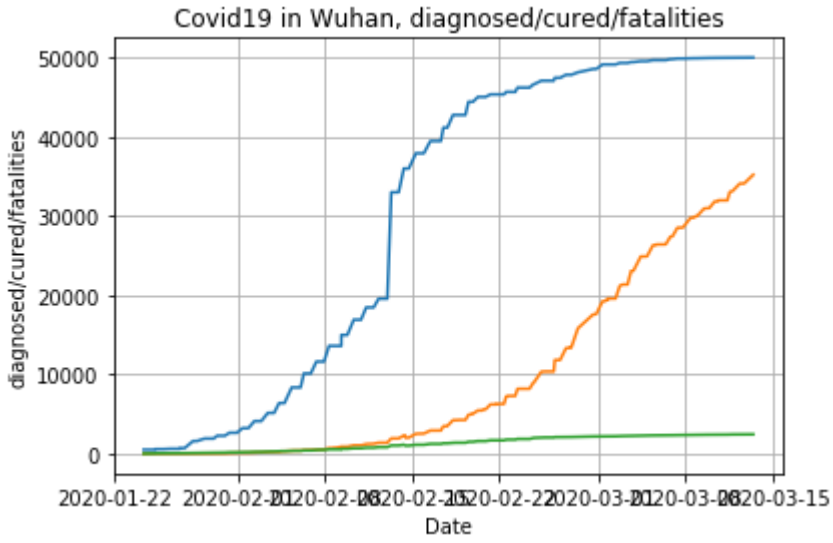
In [109]:
```
1 covid19_wuhan4
```

Out[109]:

updateTime	provinceName	provinceEnglishName	province_zipCode	cityName	cityEnglishName	c
2020-01-24 09:47:00	湖北省	Hubei	420000	武汉	Wuhan	
2020-01-24 09:48:00	湖北省	Hubei	420000	武汉	Wuhan	
2020-01-24 09:49:00	湖北省	Hubei	420000	武汉	Wuhan	
2020-01-24 09:50:00	湖北省	Hubei	420000	武汉	Wuhan	
2020-01-24 11:49:00	湖北省	Hubei	420000	武汉	Wuhan	
...	...	...	...	...	...	...
2020-03-11 18:49:00	湖北省	Hubei	420000	武汉	Wuhan	
2020-03-12 09:21:00	湖北省	Hubei	420000	武汉	Wuhan	
2020-03-12 14:27:00	湖北省	Hubei	420000	武汉	Wuhan	
2020-03-12 17:33:00	湖北省	Hubei	420000	武汉	Wuhan	
2020-03-13 11:08:00	湖北省	Hubei	420000	武汉	Wuhan	

359 rows × 14 columns

In [110]:
```
plt.plot(covid19_wuhan4.city_confirmedCount)
plt.plot(covid19_wuhan4.city_curedCount)
plt.plot(covid19_wuhan4.city_deadCount)

plt.title('Covid19 in Wuhan, diagnosed/cured/fatalities')
plt.ylabel('diagnosed/cured/fatalities')
plt.xlabel('Date')
plt.grid(True)
```



In [111]:
```
drop_cols = ['provinceName', 'provinceEnglishName', 'cityName', 'cityEnglish
             'province_curedCount', 'province_deadCount', 'province_suspecte
             'province_zipCode', 'city_zipCode']

covid19_wuhan5 = covid19_wuhan4.drop(drop_cols, axis=1)
covid19_wuhan5
```

Out[111]:

updateTime	city_confirmedCount	city_curedCount	city_deadCount
2020-01-24 09:47:00	495	0	0
2020-01-24 09:48:00	495	0	0
2020-01-24 09:49:00	495	0	0
2020-01-24 09:50:00	495	0	0
2020-01-24 11:49:00	495	0	0
...	...	...	...
2020-03-11 18:49:00	49978	33117	2423
2020-03-12 09:21:00	49986	34094	2430
2020-03-12 14:27:00	49986	34094	2430
2020-03-12 17:33:00	49986	34096	2430
2020-03-13 11:08:00	49991	35197	2436

359 rows × 3 columns

In [112]:
```
drop_cols = ['province_suspectedCount', 'city_suspectedCount', 'province_zip

covid19_china.drop(drop_cols, axis=1, inplace=True)
covid19_china.head()
```

Out[112]:

	provinceName	provinceEnglishName	cityName	cityEnglishName	province_confirmedCount	prov
0	辽宁省	Liaoning	丹东	Dandong	125	
1	辽宁省	Liaoning	沈阳	Shenyang	125	
2	辽宁省	Liaoning	大连	Dalian	125	
3	辽宁省	Liaoning	葫芦岛	Huludao	125	
4	辽宁省	Liaoning	朝阳	Chaoyang	125	

In [113]:
```
covid19_wuhan5['city_recoveredCount'] = covid19_wuhan5.city_curedCount + cov
covid19_wuhan5
```

Out[113]:

updateTime	city_confirmedCount	city_curedCount	city_deadCount	city_recoveredCount
2020-01-24 09:47:00	495	0	0	0
2020-01-24 09:48:00	495	0	0	0
2020-01-24 09:49:00	495	0	0	0
2020-01-24 09:50:00	495	0	0	0
2020-01-24 11:49:00	495	0	0	0
...	...	...	...	...
2020-03-11 18:49:00	49978	33117	2423	35540
2020-03-12 09:21:00	49986	34094	2430	36524
2020-03-12 14:27:00	49986	34094	2430	36524
2020-03-12 17:33:00	49986	34096	2430	36526
2020-03-13 11:08:00	49991	35197	2436	37633

359 rows × 4 columns

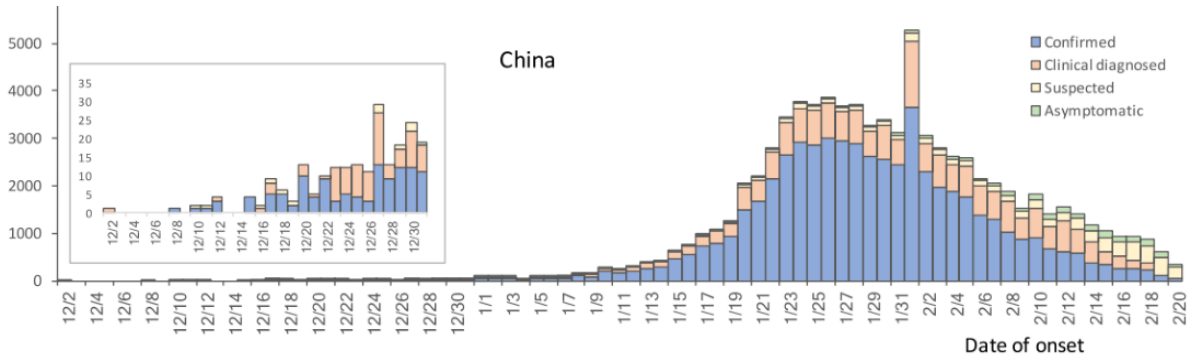
In [114]:
```
drop_cols = ['city_curedCount', 'city_deadCount']
covid19_wuhan6 = covid19_wuhan5.drop(drop_cols, axis=1)
covid19_wuhan6
```

Out[114]:

updateTime	city_confirmedCount	city_recoveredCount
2020-01-24 09:47:00	495	0
2020-01-24 09:48:00	495	0
2020-01-24 09:49:00	495	0
2020-01-24 09:50:00	495	0
2020-01-24 11:49:00	495	0
...	...	...
2020-03-11 18:49:00	49978	35540
2020-03-12 09:21:00	49986	36524
2020-03-12 14:27:00	49986	36524
2020-03-12 17:33:00	49986	36526
2020-03-13 11:08:00	49991	37633

359 rows × 2 columns

We now look at an estimate plot we found:



Based on this plot, we see that we probably need to extend for a longer time period, to give time for the epidemic to taper out.

In [115]:
```
from datetime import datetime
from dateutil.relativedelta import relativedelta

date_after_month = covid19_wuhan5.index[0] + relativedelta(months=2, days=1)
date_after_month
```

Out[115]: Timestamp('2020-03-25 09:47:00')

In [116]:
```
1 extend_1month_19days = [x + relativedelta(months=1, days=19) for x in covid1
2 extend_1month_19days
```

Out[116]:
```
[Timestamp('2020-03-14 09:47:00'),
 Timestamp('2020-03-14 09:48:00'),
 Timestamp('2020-03-14 09:49:00'),
 Timestamp('2020-03-14 09:50:00'),
 Timestamp('2020-03-14 11:49:00'),
 Timestamp('2020-03-14 12:23:00'),
 Timestamp('2020-03-14 12:24:00'),
 Timestamp('2020-03-14 12:51:00'),
 Timestamp('2020-03-14 13:15:00'),
 Timestamp('2020-03-14 13:16:00'),
 Timestamp('2020-03-14 13:22:00'),
 Timestamp('2020-03-14 16:11:00'),
 Timestamp('2020-03-14 16:48:00'),
 Timestamp('2020-03-14 17:30:00'),
 Timestamp('2020-03-15 07:48:00'),
 Timestamp('2020-03-15 08:03:00'),
 Timestamp('2020-03-15 08:08:00'),
 Timestamp('2020-03-15 08:11:00'),
 Timestamp('2020-03-15 08:12:00'),
```

In [117]:
```
1 len(extend_1month_19days)
```

Out[117]:
```
359
```

In [118]:
```
1 extend_1month_19days[358]
```

Out[118]:
```
Timestamp('2020-05-02 11:08:00')
```

In [119]:
```
extend_2more_months = [extend_1month_19days[358] + relativedelta(days=x) for
extend_2more_months
```

Out[119]:
```
[Timestamp('2020-05-03 11:08:00'),
 Timestamp('2020-05-04 11:08:00'),
 Timestamp('2020-05-05 11:08:00'),
 Timestamp('2020-05-06 11:08:00'),
 Timestamp('2020-05-07 11:08:00'),
 Timestamp('2020-05-08 11:08:00'),
 Timestamp('2020-05-09 11:08:00'),
 Timestamp('2020-05-10 11:08:00'),
 Timestamp('2020-05-11 11:08:00'),
 Timestamp('2020-05-12 11:08:00'),
 Timestamp('2020-05-13 11:08:00'),
 Timestamp('2020-05-14 11:08:00'),
 Timestamp('2020-05-15 11:08:00'),
 Timestamp('2020-05-16 11:08:00'),
 Timestamp('2020-05-17 11:08:00'),
 Timestamp('2020-05-18 11:08:00'),
 Timestamp('2020-05-19 11:08:00'),
 Timestamp('2020-05-20 11:08:00'),
 Timestamp('2020-05-21 11:08:00'),
 Timestamp('2020-05-22 11:08:00'),
 Timestamp('2020-05-23 11:08:00'),
 Timestamp('2020-05-24 11:08:00'),
 Timestamp('2020-05-25 11:08:00'),
 Timestamp('2020-05-26 11:08:00'),
 Timestamp('2020-05-27 11:08:00'),
 Timestamp('2020-05-28 11:08:00'),
 Timestamp('2020-05-29 11:08:00'),
 Timestamp('2020-05-30 11:08:00'),
 Timestamp('2020-05-31 11:08:00'),
 Timestamp('2020-06-01 11:08:00'),
 Timestamp('2020-06-02 11:08:00'),
 Timestamp('2020-06-03 11:08:00'),
 Timestamp('2020-06-04 11:08:00'),
 Timestamp('2020-06-05 11:08:00'),
 Timestamp('2020-06-06 11:08:00'),
 Timestamp('2020-06-07 11:08:00'),
 Timestamp('2020-06-08 11:08:00'),
 Timestamp('2020-06-09 11:08:00'),
 Timestamp('2020-06-10 11:08:00'),
 Timestamp('2020-06-11 11:08:00'),
 Timestamp('2020-06-12 11:08:00'),
 Timestamp('2020-06-13 11:08:00'),
 Timestamp('2020-06-14 11:08:00'),
 Timestamp('2020-06-15 11:08:00'),
 Timestamp('2020-06-16 11:08:00'),
 Timestamp('2020-06-17 11:08:00'),
 Timestamp('2020-06-18 11:08:00'),
 Timestamp('2020-06-19 11:08:00'),
 Timestamp('2020-06-20 11:08:00'),
 Timestamp('2020-06-21 11:08:00'),
 Timestamp('2020-06-22 11:08:00'),
 Timestamp('2020-06-23 11:08:00'),
 Timestamp('2020-06-24 11:08:00'),
```

```
 Timestamp('2020-06-25 11:08:00'),
 Timestamp('2020-06-26 11:08:00'),
 Timestamp('2020-06-27 11:08:00'),
 Timestamp('2020-06-28 11:08:00'),
 Timestamp('2020-06-29 11:08:00'),
 Timestamp('2020-06-30 11:08:00')]
```

In [120]:
```
1 extend_3_months = extend_1month_19days + extend_2more_months
2 extend_3_months
```

Out[120]:
```
[Timestamp('2020-03-14 09:47:00'),
 Timestamp('2020-03-14 09:48:00'),
 Timestamp('2020-03-14 09:49:00'),
 Timestamp('2020-03-14 09:50:00'),
 Timestamp('2020-03-14 11:49:00'),
 Timestamp('2020-03-14 12:23:00'),
 Timestamp('2020-03-14 12:24:00'),
 Timestamp('2020-03-14 12:51:00'),
 Timestamp('2020-03-14 13:15:00'),
 Timestamp('2020-03-14 13:16:00'),
 Timestamp('2020-03-14 13:22:00'),
 Timestamp('2020-03-14 16:11:00'),
 Timestamp('2020-03-14 16:48:00'),
 Timestamp('2020-03-14 17:30:00'),
 Timestamp('2020-03-15 07:48:00'),
 Timestamp('2020-03-15 08:03:00'),
 Timestamp('2020-03-15 08:08:00'),
 Timestamp('2020-03-15 08:11:00'),
 Timestamp('2020-03-15 08:12:00'),
```

In [121]:
```
1 df_extend_3_months = pd.DataFrame(extend_3_months, columns=['updateTime'])
```

In [122]:
```
1 df_extend_3_months
```

Out[122]:

	updateTime
0	2020-03-14 09:47:00
1	2020-03-14 09:48:00
2	2020-03-14 09:49:00
3	2020-03-14 09:50:00
4	2020-03-14 11:49:00
...	...
413	2020-06-26 11:08:00
414	2020-06-27 11:08:00
415	2020-06-28 11:08:00
416	2020-06-29 11:08:00
417	2020-06-30 11:08:00

418 rows × 1 columns

In [123]:
```
1 df_extend_3_months.index = df_extend_3_months['updateTime']
```

In [124]:
```
1 df_extend_3_months
```

Out[124]:

	updateTime
updateTime	
2020-03-14 09:47:00	2020-03-14 09:47:00
2020-03-14 09:48:00	2020-03-14 09:48:00
2020-03-14 09:49:00	2020-03-14 09:49:00
2020-03-14 09:50:00	2020-03-14 09:50:00
2020-03-14 11:49:00	2020-03-14 11:49:00
...	...
2020-06-26 11:08:00	2020-06-26 11:08:00
2020-06-27 11:08:00	2020-06-27 11:08:00
2020-06-28 11:08:00	2020-06-28 11:08:00
2020-06-29 11:08:00	2020-06-29 11:08:00
2020-06-30 11:08:00	2020-06-30 11:08:00

418 rows × 1 columns

In [125]:
```
1 df_extend_3_months.drop(['updateTime'], axis=1, inplace=True)
```

In [126]:
```
1 df_extend_3_months
```

Out[126]:

updateTime
2020-03-14 09:47:00
2020-03-14 09:48:00
2020-03-14 09:49:00
2020-03-14 09:50:00
2020-03-14 11:49:00
...
2020-06-26 11:08:00
2020-06-27 11:08:00
2020-06-28 11:08:00
2020-06-29 11:08:00
2020-06-30 11:08:00

418 rows × 0 columns

In [127]:
```
1 df_extend_3_months['city_confirmedCount'] = float("NaN")
2 df_extend_3_months['city_recoveredCount'] = float("NaN")
```

In [128]:
```
1 df_extend_3_months
```

Out[128]:

updateTime	city_confirmedCount	city_recoveredCount
2020-03-14 09:47:00	NaN	NaN
2020-03-14 09:48:00	NaN	NaN
2020-03-14 09:49:00	NaN	NaN
2020-03-14 09:50:00	NaN	NaN
2020-03-14 11:49:00	NaN	NaN
...	...	...
2020-06-26 11:08:00	NaN	NaN
2020-06-27 11:08:00	NaN	NaN
2020-06-28 11:08:00	NaN	NaN
2020-06-29 11:08:00	NaN	NaN
2020-06-30 11:08:00	NaN	NaN

418 rows × 2 columns

In [129]:
```
1 len(df_extend_3_months)
```

Out[129]:
```
418
```

In [130]:
```
1 df_extend_3_months.index
```

Out[130]:
```
DatetimeIndex(['2020-03-14 09:47:00', '2020-03-14 09:48:00',
               '2020-03-14 09:49:00', '2020-03-14 09:50:00',
               '2020-03-14 11:49:00', '2020-03-14 12:23:00',
               '2020-03-14 12:24:00', '2020-03-14 12:51:00',
               '2020-03-14 13:15:00', '2020-03-14 13:16:00',
               ...
               '2020-06-21 11:08:00', '2020-06-22 11:08:00',
               '2020-06-23 11:08:00', '2020-06-24 11:08:00',
               '2020-06-25 11:08:00', '2020-06-26 11:08:00',
               '2020-06-27 11:08:00', '2020-06-28 11:08:00',
               '2020-06-29 11:08:00', '2020-06-30 11:08:00'],
              dtype='datetime64[ns]', name='updateTime', length=418, freq=None)
```

In [131]:
```
1 covid19_wuhan7 = pd.concat([covid19_wuhan6, df_extend_3_months], sort=False)
```

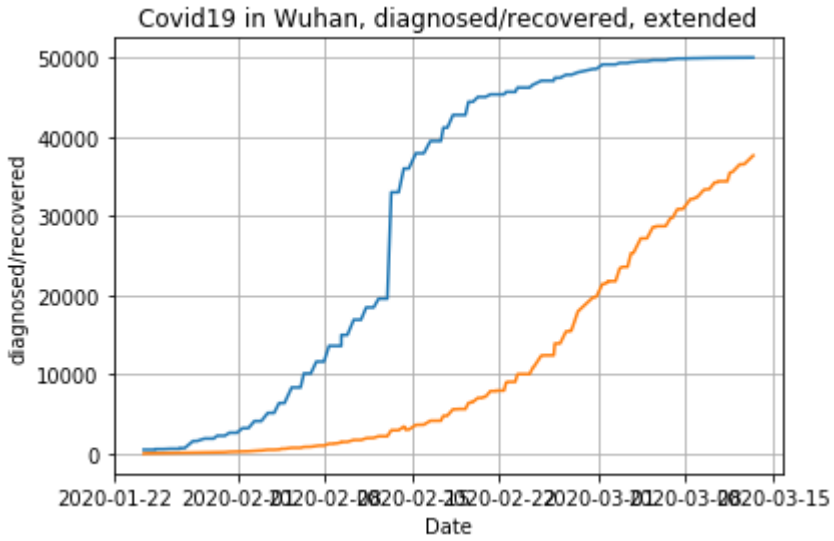
In [132]:
```
1 covid19_wuhan7
```

Out[132]:

updateTime	city_confirmedCount	city_recoveredCount
2020-01-24 09:47:00	495.0	0.0
2020-01-24 09:48:00	495.0	0.0
2020-01-24 09:49:00	495.0	0.0
2020-01-24 09:50:00	495.0	0.0
2020-01-24 11:49:00	495.0	0.0
...	...	...
2020-06-26 11:08:00	NaN	NaN
2020-06-27 11:08:00	NaN	NaN
2020-06-28 11:08:00	NaN	NaN
2020-06-29 11:08:00	NaN	NaN
2020-06-30 11:08:00	NaN	NaN

777 rows × 2 columns

In [133]:
```
plt.plot(covid19_wuhan7.city_confirmedCount)
plt.plot(covid19_wuhan7.city_recoveredCount)

plt.title('Covid19 in Wuhan, diagnosed/recovered, extended')
plt.ylabel('diagnosed/recovered')
plt.xlabel('Date')
plt.grid(True)
```

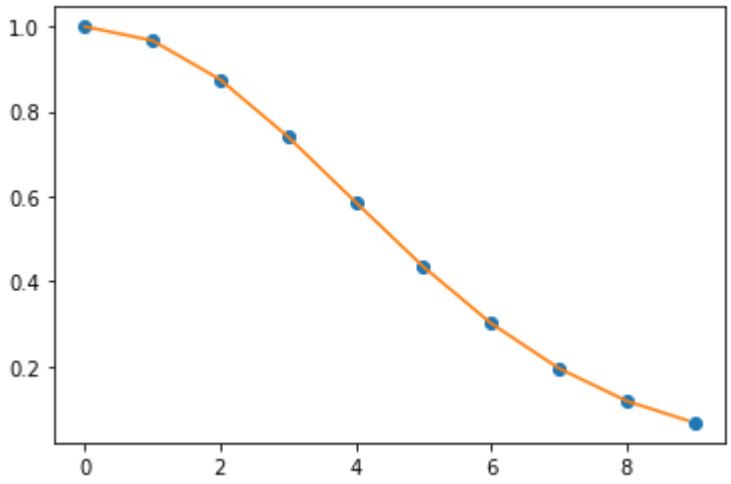


Now we have to extrapolate the I and R curves.

Let's start with the I curve.

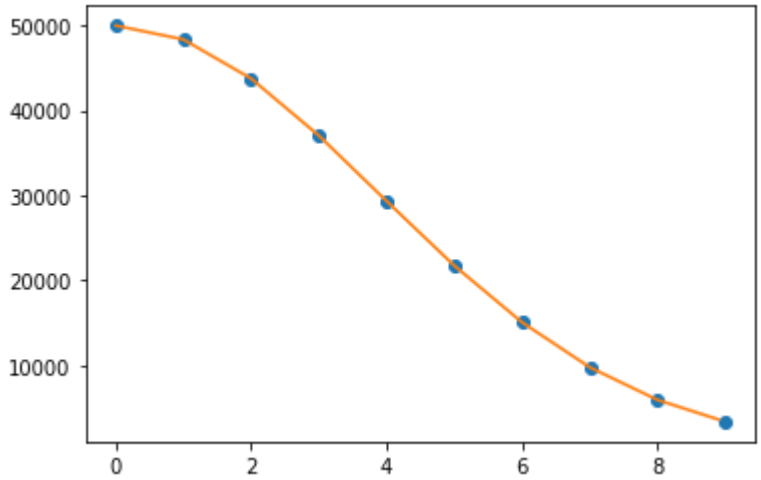
In [134]:
```
from scipy import interpolate
x = np.arange(0, 10)
y = np.exp(-x**2/30.0)
f = interpolate.interp1d(x, y)
xnew = np.arange(0, 9, 0.1)
ynew = f(xnew)   # use interpolation function returned by `interp1d`
plt.plot(x, y, 'o', xnew, ynew, '-')
```

Out[134]:
```
[<matplotlib.lines.Line2D at 0x21b2464ecc0>,
 <matplotlib.lines.Line2D at 0x21b2464edd8>]
```



In [135]:
```
x = np.arange(0, 10)
peak = 50000
y = peak * np.exp(-x**2/30.0)
f = interpolate.interp1d(x, y)
xnew = np.arange(0, 9, 0.1)
ynew = f(xnew)   # use interpolation function returned by `interp1d`
plt.plot(x, y, 'o', xnew, ynew, '-')
```

Out[135]:
```
[<matplotlib.lines.Line2D at 0x21b246aacf8>,
 <matplotlib.lines.Line2D at 0x21b246aae10>]
```



In [136]:
```
1 covid19_wuhan7.city_confirmedCount[358], covid19_wuhan7.city_confirmedCount[
```

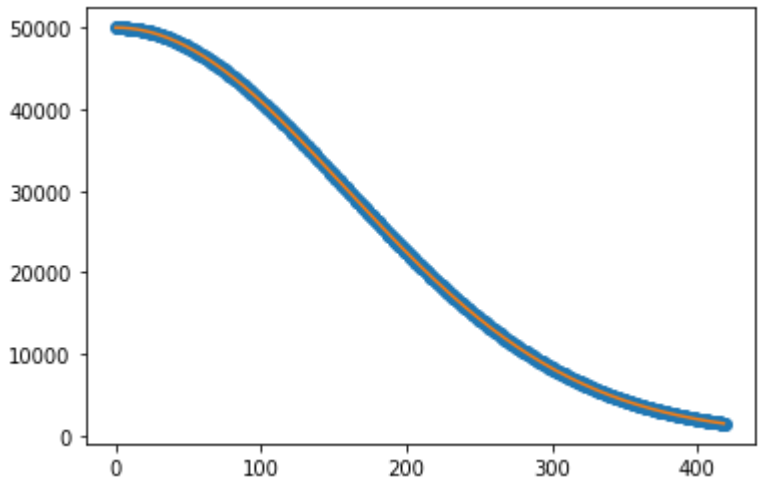
Out[136]: (49991.0, nan, 777)

In [137]:
```
1 777 - 358
```

Out[137]: 419

In [138]:
```
1 x = np.arange(0, 420)
2 peak = 50000
3 y = peak * np.exp(-x**2/peak)
4 f = interpolate.interp1d(x, y)
5 xnew = np.arange(0, 419, 1)
6 ynew = f(xnew)   # use interpolation function returned by `interp1d`
7 plt.plot(x, y, 'o', xnew, ynew, '-')
```

Out[138]:
```
[<matplotlib.lines.Line2D at 0x21b24700d68>,
 <matplotlib.lines.Line2D at 0x21b24700e80>]
```

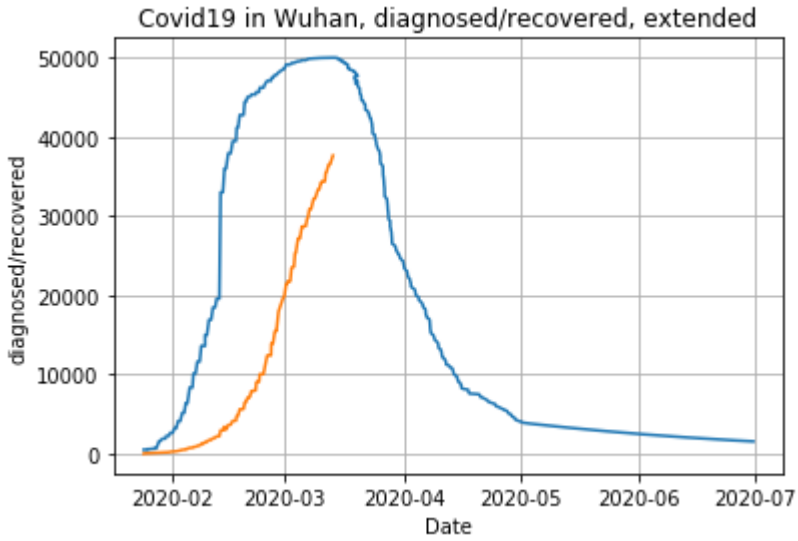


In [139]:
```
1 len(ynew), len(covid19_wuhan7.city_confirmedCount[359:777])
```

Out[139]: (419, 418)

In [140]:
```
1 covid19_wuhan7.city_confirmedCount[359:777] = ynew[:-1]
```

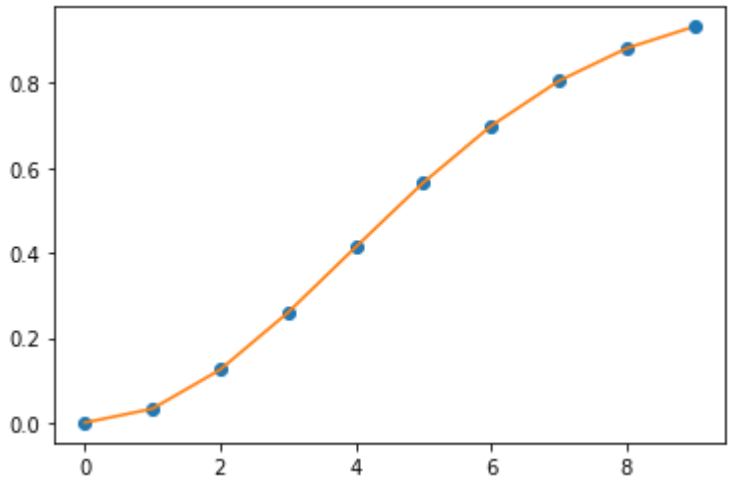
In [141]:
```
plt.plot(covid19_wuhan7.city_confirmedCount)
plt.plot(covid19_wuhan7.city_recoveredCount)

plt.title('Covid19 in Wuhan, diagnosed/recovered, extended')
plt.ylabel('diagnosed/recovered')
plt.xlabel('Date')
plt.grid(True)
```



Now for the R curve.

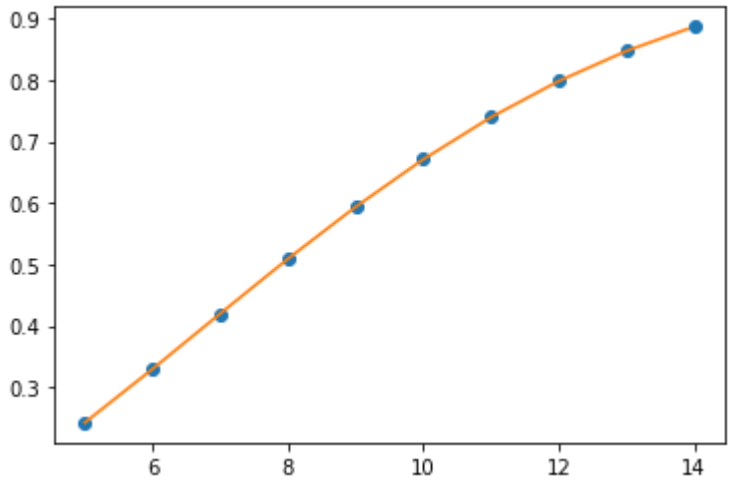
In [142]:
```
x = np.arange(0, 10)
y = 1 - np.exp(-x**2/30.0)
f = interpolate.interp1d(x, y)
xnew = np.arange(0, 9, 0.1)
ynew = f(xnew)   # use interpolation function returned by `interp1d`
plt.plot(x, y, 'o', xnew, ynew, '-')
```

Out[142]:
```
[<matplotlib.lines.Line2D at 0x21b247bbbe0>,
 <matplotlib.lines.Line2D at 0x21b23f7b160>]
```



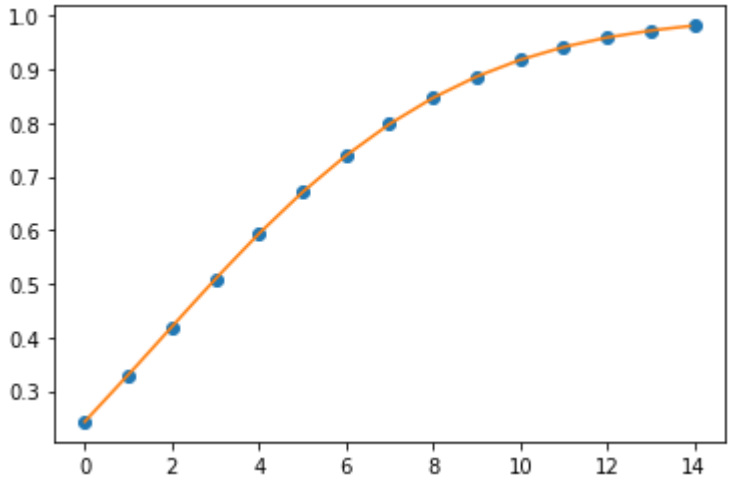
In [143]:
```
x = np.arange(5, 15)
y = 1 - np.exp(-x**2/90.0)
f = interpolate.interp1d(x, y)
xnew = np.arange(5, 14, 0.1)
ynew = f(xnew)   # use interpolation function returned by `interp1d`
plt.plot(x, y, 'o', xnew, ynew, '-')
```

Out[143]:
```
[<matplotlib.lines.Line2D at 0x21b24817b38>,
 <matplotlib.lines.Line2D at 0x21b24817c50>]
```



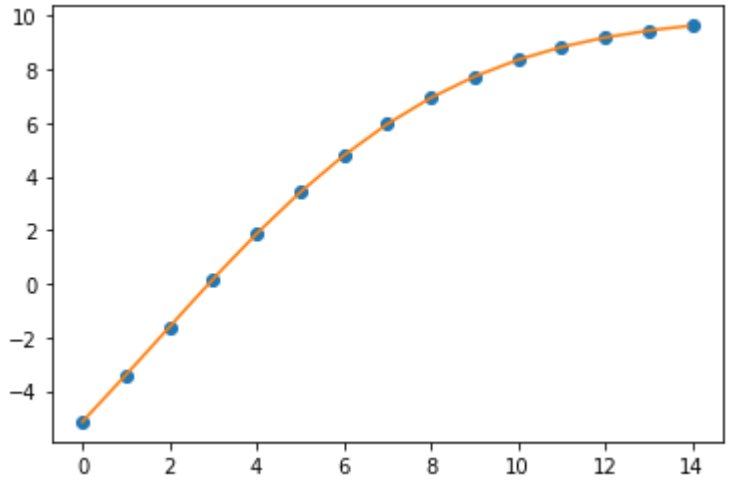
In [144]:
```
x = np.arange(0, 15)
y = 1 - np.exp(-(x+5)**2/90.0)
f = interpolate.interp1d(x, y)
xnew = np.arange(0, 14, 0.1)
ynew = f(xnew)   # use interpolation function returned by `interp1d`
plt.plot(x, y, 'o', xnew, ynew, '-')
```

Out[144]:
```
[<matplotlib.lines.Line2D at 0x21b24879550>,
 <matplotlib.lines.Line2D at 0x21b24879668>]
```



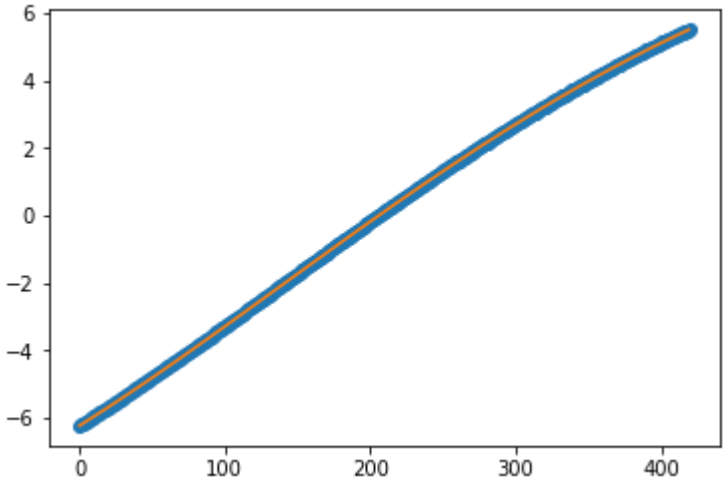
In [145]:
```
x = np.arange(0, 15)
y = 10 - 20*np.exp(-(x+5)**2/90.0)
f = interpolate.interp1d(x, y)
xnew = np.arange(0, 14, 0.1)
ynew = f(xnew)   # use interpolation function returned by `interp1d`
plt.plot(x, y, 'o', xnew, ynew, '-')
```

Out[145]:
```
[<matplotlib.lines.Line2D at 0x21b236d1a90>,
 <matplotlib.lines.Line2D at 0x21b236d1d68>]
```



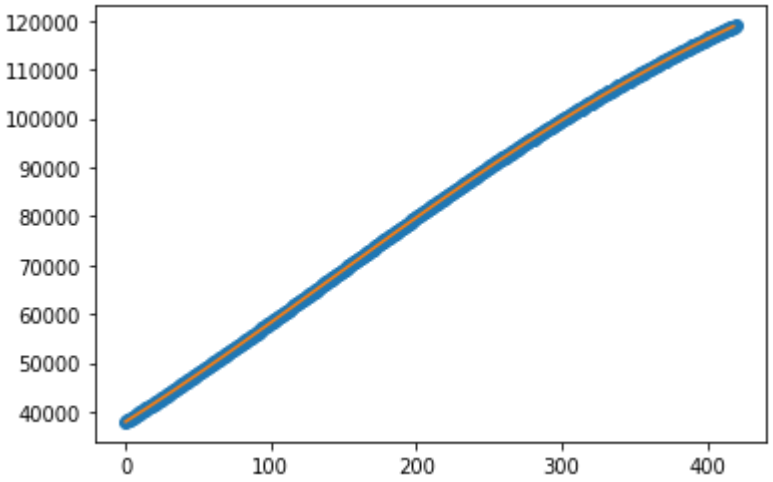
In [146]:
```
x = np.arange(0, 420)
y = 10 - 20*np.exp(-(x+250)**2/300000)
f = interpolate.interp1d(x, y)
xnew = np.arange(0, 419, 0.1)
ynew = f(xnew)   # use interpolation function returned by `interp1d`
plt.plot(x, y, 'o', xnew, ynew, '-')
```

Out[146]:
```
[<matplotlib.lines.Line2D at 0x21b2473b668>,
 <matplotlib.lines.Line2D at 0x21b2473ba90>]
```



In [147]:
```
###### x = np.arange(0, 420)
y = 150000 - 138000*np.exp(-(x+250)**2/300000)
f = interpolate.interp1d(x, y)
xnew = np.arange(0, 419, 1)
ynew = f(xnew)   # use interpolation function returned by `interp1d`
plt.plot(x, y, 'o', xnew, ynew, '-')
```

Out[147]:
```
[<matplotlib.lines.Line2D at 0x21b23f7b6a0>,
 <matplotlib.lines.Line2D at 0x21b23f7b630>]
```



In [148]:
```
1 ynew[0]
```

Out[148]: 37952.78423121238

In [149]:
```
1 covid19_wuhan7.city_recoveredCount[358], covid19_wuhan7.city_recoveredCount[
```

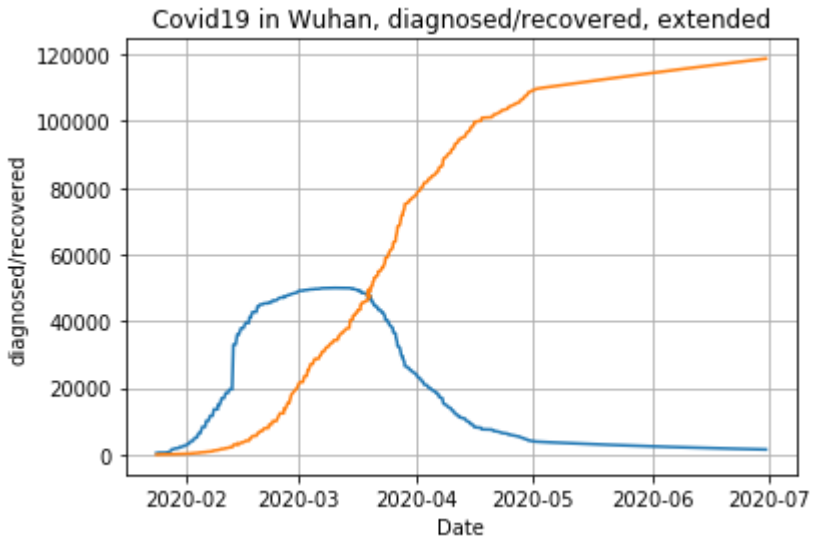
Out[149]: (37633.0, nan, 777)

In [150]:
```
1 len(ynew), len(covid19_wuhan7.city_recoveredCount[359:777])
```

Out[150]: (419, 418)

In [151]:
```
1 covid19_wuhan7.city_recoveredCount[359:777] = ynew[:-1]
```

In [152]:
```
1 plt.plot(covid19_wuhan7.city_confirmedCount)
2 plt.plot(covid19_wuhan7.city_recoveredCount)
3
4 plt.title('Covid19 in Wuhan, diagnosed/recovered, extended')
5 plt.ylabel('diagnosed/recovered')
6 plt.xlabel('Date')
7 plt.grid(True)
```



In [153]:
```
1 covid19_wuhan7
```

Out[153]:

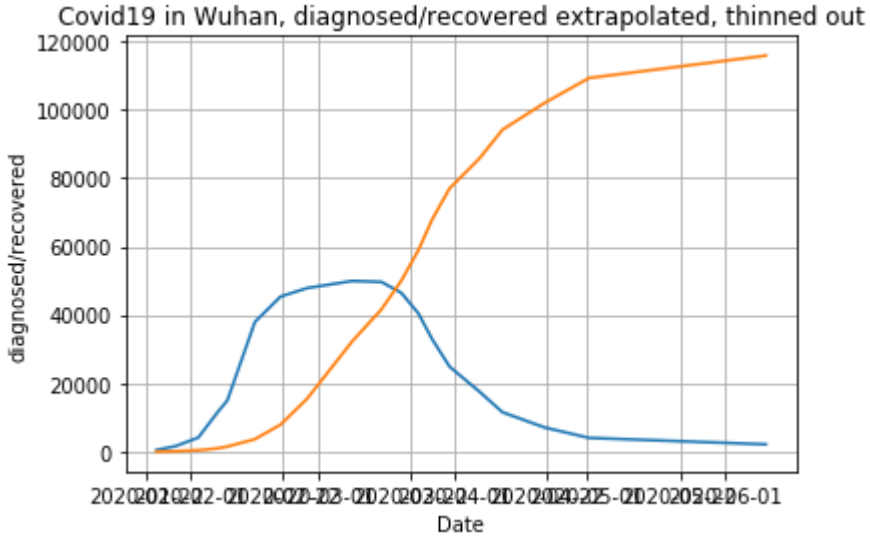
updateTime	city_confirmedCount	city_recoveredCount
2020-01-24 09:47:00	495.000000	0.000000
2020-01-24 09:48:00	495.000000	0.000000
2020-01-24 09:49:00	495.000000	0.000000
2020-01-24 09:50:00	495.000000	0.000000
2020-01-24 11:49:00	495.000000	0.000000
...	...	...
2020-06-26 11:08:00	1649.781748	118118.570748
2020-06-27 11:08:00	1622.718786	118259.281502
2020-06-28 11:08:00	1596.035921	118399.581890
2020-06-29 11:08:00	1569.729020	118539.471860
2020-06-30 11:08:00	1543.793973	118678.951367

777 rows × 2 columns

In [154]:
```
1 covid19_wuhan8 = covid19_wuhan7[::42]  # Selects every 42nd raw starting fro
2 len(covid19_wuhan8)
```

Out[154]: 19

In [155]:
```
plt.plot(covid19_wuhan8.city_confirmedCount)
plt.plot(covid19_wuhan8.city_recoveredCount)

plt.title('Covid19 in Wuhan, diagnosed/recovered extrapolated, thinned out')
plt.ylabel('diagnosed/recovered')
plt.xlabel('Date')
plt.grid(True)
```



In [156]:
```
covid19_wuhan8.to_numpy()
```

Out[156]:
```
array([[   495.        ,      0.        ],
       [  1590.        ,    132.        ],
       [  4109.        ,    399.        ],
       [ 11618.        ,    974.        ],
       [ 14982.        ,   1485.        ],
       [ 37914.        ,   3642.        ],
       [ 45346.        ,   7898.        ],
       [ 47824.        ,  15432.        ],
       [ 49912.        ,  32278.        ],
       [ 49640.30007926,  41575.90403277],
       [ 46414.08668136,  50031.97830227],
       [ 40440.95485127,  58906.14471023],
       [ 32835.91892612,  67963.00108155],
       [ 24844.66410716,  76983.14157437],
       [ 17517.54596888,  85771.28712071],
       [ 11509.84786982,  94162.23950932],
       [  7047.28901103, 102024.50537855],
       [  4020.9693501 , 109261.63196995],
       [  2137.94054909, 115811.46507259]])
```

Renormalizing:

In [157]:
```
1 covid19_wuhan9 = covid19_wuhan8.div(115812)
2 covid19_wuhan9
```

Out[157]:

updateTime	city_confirmedCount	city_recoveredCount
2020-01-24 09:47:00	0.004274	0.000000
2020-01-28 16:36:00	0.013729	0.001140
2020-02-02 20:50:00	0.035480	0.003445
2020-02-07 07:15:00	0.100318	0.008410
2020-02-09 11:01:00	0.129365	0.012823
2020-02-15 16:32:00	0.327375	0.031448
2020-02-21 12:12:00	0.391548	0.068197
2020-02-27 11:55:00	0.412945	0.133250
2020-03-08 18:05:00	0.430974	0.278710
2020-03-15 08:14:00	0.428628	0.358995
2020-03-19 21:16:00	0.400771	0.432010
2020-03-23 17:34:00	0.349195	0.508636
2020-03-26 21:05:00	0.283528	0.586839
2020-03-30 19:21:00	0.214526	0.664725
2020-04-06 14:14:00	0.151258	0.740608
2020-04-11 19:19:00	0.099384	0.813061
2020-04-21 09:14:00	0.060851	0.880949
2020-05-01 09:21:00	0.034720	0.943440
2020-06-10 11:08:00	0.018460	0.999995

In [158]:
```
y_wuhan9 = covid19_wuhan9.to_numpy()
y_wuhan9
```

Out[158]:
```
array([[0.00427417, 0.        ],
       [0.01372915, 0.00113978],
       [0.03547992, 0.00344524],
       [0.10031776, 0.00841018],
       [0.12936483, 0.01282251],
       [0.3273754 , 0.03144752],
       [0.39154837, 0.06819673],
       [0.41294512, 0.13325044],
       [0.43097434, 0.27871032],
       [0.42862829, 0.35899478],
       [0.40077096, 0.43201031],
       [0.34919486, 0.50863593],
       [0.28352778, 0.58683902],
       [0.21452582, 0.66472509],
       [0.15125847, 0.74060794],
       [0.0993839 , 0.81306116],
       [0.06085111, 0.88094934],
       [0.0347198 , 0.94343964],
       [0.01846044, 0.99999538]])
```

Without the fudge below, the NUTS sim will probably crash out with `Bad Initial Energy` error!

In [159]:
```
# to get rid of bad initial energy
y_wuhan9[0,1] = 0.0010
y_wuhan9
```

Out[159]:
```
array([[0.00427417, 0.001     ],
       [0.01372915, 0.00113978],
       [0.03547992, 0.00344524],
       [0.10031776, 0.00841018],
       [0.12936483, 0.01282251],
       [0.3273754 , 0.03144752],
       [0.39154837, 0.06819673],
       [0.41294512, 0.13325044],
       [0.43097434, 0.27871032],
       [0.42862829, 0.35899478],
       [0.40077096, 0.43201031],
       [0.34919486, 0.50863593],
       [0.28352778, 0.58683902],
       [0.21452582, 0.66472509],
       [0.15125847, 0.74060794],
       [0.0993839 , 0.81306116],
       [0.06085111, 0.88094934],
       [0.0347198 , 0.94343964],
       [0.01846044, 0.99999538]])
```

In [160]:
```
y_wuhan9.shape
```

Out[160]:
```
(19, 2)
```

Now we have our data, we are ready for our Bayesian modeling!

Bayesian modeling Chinese stats

We import `DifferentialEquation` from `PyMC3` :

In [161]:
```
import pymc3 as pm
from pymc3.ode import DifferentialEquation
import theano
```

Matching shapes for X (time) and Y:

In [162]:
```
np.arange(0.25, 5, 0.25).shape
```

Out[162]:
```
(19,)
```

Our epidemic model dynamics are captured thusly (repeated from above for peace of mind):

In [163]:
```
def SIR(y, t, p):
    di = p[0] * (1. - y[0] - y[1] - p[1]/p[0]) * y[0]
    dr = p[1] * y[0]
    return [di, dr]
```

In [164]:
```
def susceptible(i, r):
    return 1 - i - r
```

In [165]:
```
sir_model = DifferentialEquation(
    func = SIR,
    times = np.arange(0.25, 5, 0.25),
    n_states = 2,
    n_theta = 2,
    t0 = 0,
)
```

In [166]:
```
import arviz as az

with pm.Model() as model9:
    sigma = pm.HalfCauchy('sigma', 1, shape=2)

    # R0 is bounded below by 1 because we see an epidemic has occured
    R0 = pm.Bound(pm.Normal, lower=1)('R0', 2, 3)
    gama = pm.Lognormal('gama', pm.math.log(2), 2)
    beta = pm.Deterministic('beta', gama*R0)

    sir_curves = sir_model(y0=[0.01, 0.0], theta=[beta, gama])

    # data likelihood
    Y = pm.Lognormal('Y', mu=pm.math.log(sir_curves), sd=sigma, observed=y_w

    prior = pm.sample_prior_predictive()
    trace = pm.sample(2000, tune=1000, target_accept=0.9, cores=1)
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Sequential sampling (2 chains in 1 job)
NUTS: [gama, R0, sigma]
Sampling chain 0, 0 divergences: 100%|██████████| 3000/3000 [36:39<00:00,  1.36
it/s]
Sampling chain 1, 0 divergences: 100%|██████████| 3000/3000 [43:30<00:00,  1.15
it/s]
The number of effective samples is smaller than 25% for some parameters.
```

Let's generate data from the model using parameters sampled from the posterior distribution:

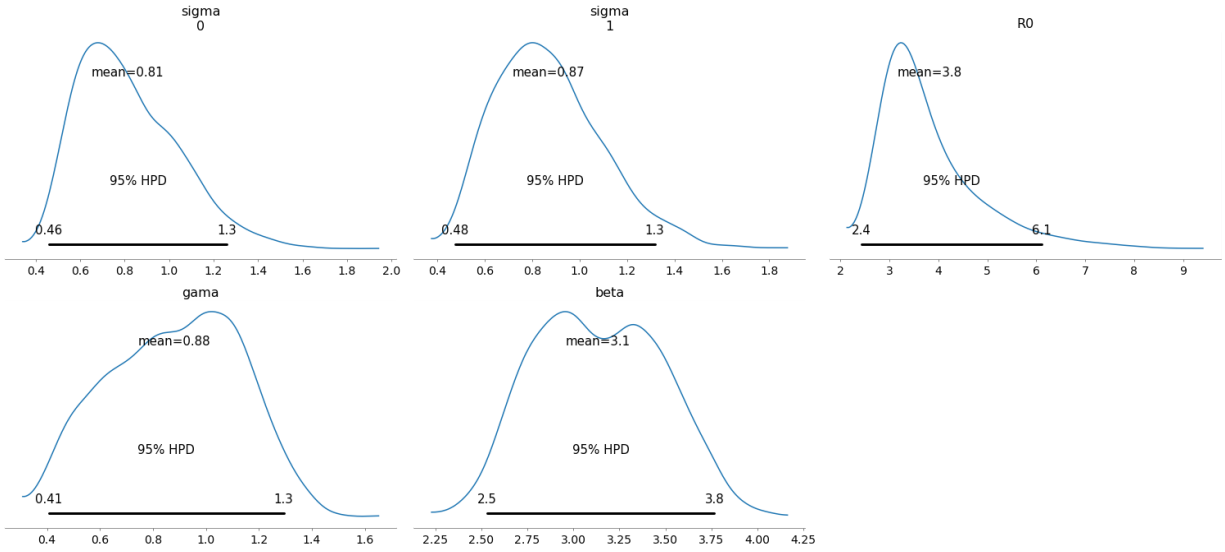
In [167]:
```
with model9:
    posterior_predictive = pm.sample_posterior_predictive(trace)
```

```
100%|██████████| 4000/4000 [04:19<00:00, 15.39it/s]
```

In [169]:
```
with model9:
    data = az.from_pymc3(trace=trace, prior=prior, posterior_predictive=post
```

In [170]:
```
1 az.plot_posterior(data, round_to=2, credible_interval=0.95)
```

Out[170]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x0000021D0EFA5AC8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000021D0EF8B240>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000021D0B46D710>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000021D0EE9A7B8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000021D3A5E1320>],
      dtype=object)
```



As can be seen from the posterior plots, R_0 is estimated at 3.8.

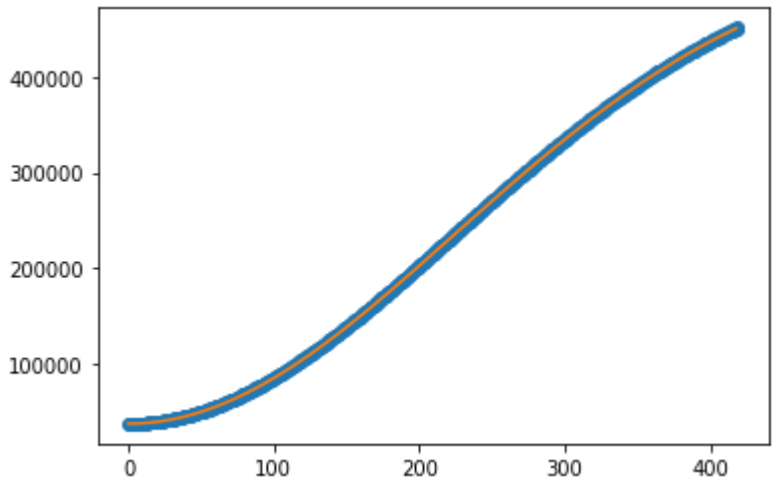
Lower bound for R_0 (highest susceptible population estimate)

Now we repeat the computation, with a ***higher susceptible population with a recovered profile commensurate with a tapering out of the epidemic in June***.

In [ ]:
```
1 10000000 - 37900
```

In [171]:
```
x = np.arange(0, 420)

#y = 10000000 - 12269500*np.exp(-(x+250)**2/300000)
#y = 10000000 - 11647000*np.exp(-(x+250)**2/400000)
#y = 10000000 - 10024000*np.exp(-(x+50)**2/400000)
#y = 10000000 - 9962100*np.exp(-(x)**2/400000)

#x0,y0 = 0,37900, x1,y1 = 420,max
#y = ax +b
#b = 37900
#a = (max - 37900) / 420
#y = (400000. - 37900.) / 420.*x + 37900

y = 537900 - 500000*np.exp(-(x)**2/100000)
f = interpolate.interp1d(x, y)
xnew = np.arange(0, 419, 1)
ynew = f(xnew)   # use interpolation function returned by `interp1d`
plt.plot(x, y, 'o', xnew, ynew, '-')
```

Out[171]:
```
[<matplotlib.lines.Line2D at 0x21d02ab1b38>,
 <matplotlib.lines.Line2D at 0x21d02ab1ac8>]
```

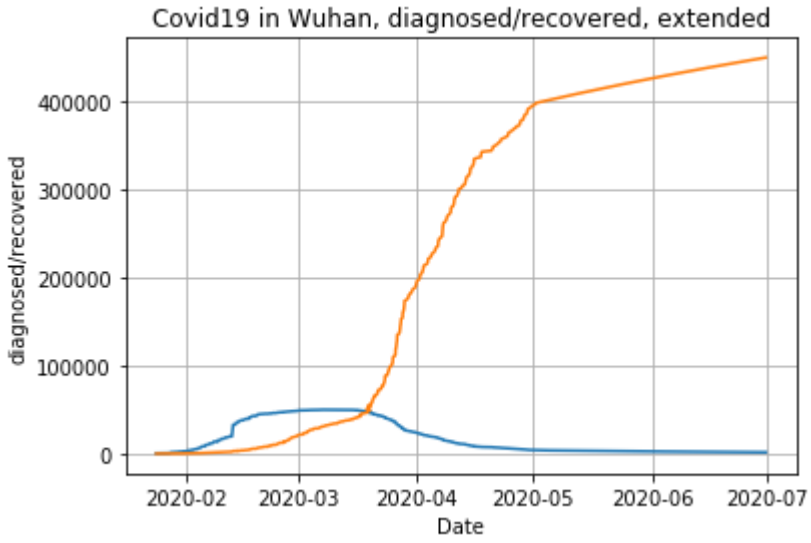


In [172]:
```
ynew[0]
```

Out[172]:
```
37900.0
```

In [173]:
```
covid19_wuhan7.city_recoveredCount[359:777] = ynew[:-1]
```

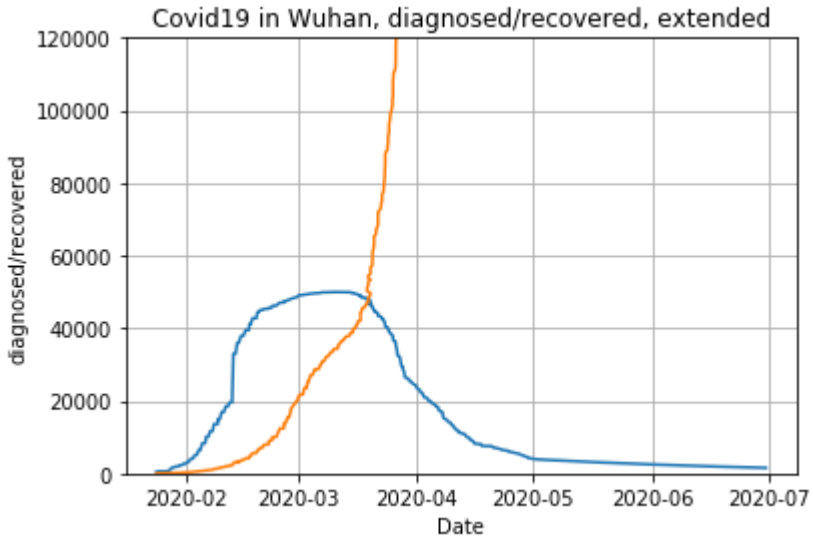
In [174]:
```
plt.plot(covid19_wuhan7.city_confirmedCount)
plt.plot(covid19_wuhan7.city_recoveredCount)

plt.title('Covid19 in Wuhan, diagnosed/recovered, extended')
plt.ylabel('diagnosed/recovered')
plt.xlabel('Date')
plt.grid(True)
```



This corresponds to a much "*flatter*" I curve. This is what it means to "*flatten the curve*" (of the disease), which you may have heard a lot of, lately.

Zooming in:

In [175]:
```
plt.plot(covid19_wuhan7.city_confirmedCount)
plt.plot(covid19_wuhan7.city_recoveredCount)

plt.ylim(0, 120000)
plt.title('Covid19 in Wuhan, diagnosed/recovered, extended')
plt.ylabel('diagnosed/recovered')
plt.xlabel('Date')
plt.grid(True)
```



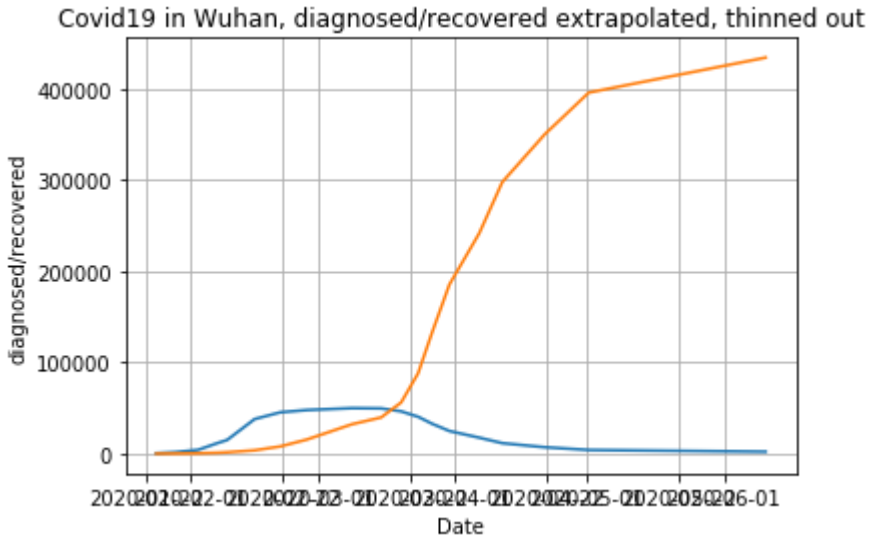
And so we see this corresponds to a lower bound of the R_0 of the disease, because the recovered count skyrockets unrealistically. Also, the amount of individuals recovered exceeds the amount of individuals infected, which is nonsensical. This will be addressed by renormalization further, which will have the effect of flattening the curve, but this just reflects our attempt to produce a safe lower bound for R_0.

Let's thin out the data:

In [176]:
```
covid19_wuhan8 = covid19_wuhan7[::42]  # Selects every 42nd raw starting fro
len(covid19_wuhan8)
```

Out[176]: 19

In [177]:
```
plt.plot(covid19_wuhan8.city_confirmedCount)
plt.plot(covid19_wuhan8.city_recoveredCount)

plt.title('Covid19 in Wuhan, diagnosed/recovered extrapolated, thinned out')
plt.ylabel('diagnosed/recovered')
plt.xlabel('Date')
plt.grid(True)
```



In [178]:
```
covid19_wuhan8.to_numpy()
```

Out[178]:
```
array([[4.95000000e+02, 0.00000000e+00],
       [1.59000000e+03, 1.32000000e+02],
       [4.10900000e+03, 3.99000000e+02],
       [1.16180000e+04, 9.74000000e+02],
       [1.49820000e+04, 1.48500000e+03],
       [3.79140000e+04, 3.64200000e+03],
       [4.53460000e+04, 7.89800000e+03],
       [4.78240000e+04, 1.54320000e+04],
       [4.99120000e+04, 3.22780000e+04],
       [4.96403001e+04, 3.97017459e+04],
       [4.64140867e+04, 5.61631077e+04],
       [4.04409549e+04, 8.82281482e+04],
       [3.28359189e+04, 1.32709187e+05],
       [2.48446641e+04, 1.85446712e+05],
       [1.75175460e+04, 2.41947757e+05],
       [1.15098479e+04, 2.98005775e+05],
       [7.04728901e+03, 3.50186268e+05],
       [4.02096935e+03, 3.96108439e+05],
       [2.13794055e+03, 4.34508981e+05]])
```

Let's normalize:

In [179]:
```
1 covid19_wuhan9 = covid19_wuhan8.div(4.34510000e+05)
2 covid19_wuhan9
```

Out[179]:

updateTime	city_confirmedCount	city_recoveredCount
2020-01-24 09:47:00	0.001139	0.000000
2020-01-28 16:36:00	0.003659	0.000304
2020-02-02 20:50:00	0.009457	0.000918
2020-02-07 07:15:00	0.026738	0.002242
2020-02-09 11:01:00	0.034480	0.003418
2020-02-15 16:32:00	0.087257	0.008382
2020-02-21 12:12:00	0.104361	0.018177
2020-02-27 11:55:00	0.110064	0.035516
2020-03-08 18:05:00	0.114870	0.074286
2020-03-15 08:14:00	0.114244	0.091371
2020-03-19 21:16:00	0.106819	0.129256
2020-03-23 17:34:00	0.093073	0.203052
2020-03-26 21:05:00	0.075570	0.305423
2020-03-30 19:21:00	0.057179	0.426795
2020-04-06 14:14:00	0.040316	0.556829
2020-04-11 19:19:00	0.026489	0.685843
2020-04-21 09:14:00	0.016219	0.805934
2020-05-01 09:21:00	0.009254	0.911621
2020-06-10 11:08:00	0.004920	0.999998

In [180]:
```
y_wuhan9 = covid19_wuhan9.to_numpy()
y_wuhan9
```

Out[180]:
```
array([[1.13921429e-03, 0.00000000e+00],
       [3.65929438e-03, 3.03790477e-04],
       [9.45662931e-03, 9.18275759e-04],
       [2.67381648e-02, 2.24160549e-03],
       [3.44802191e-02, 3.41764286e-03],
       [8.72569101e-02, 8.38185542e-03],
       [1.04361234e-01, 1.81767969e-02],
       [1.10064210e-01, 3.55158684e-02],
       [1.14869623e-01, 7.42859773e-02],
       [1.14244321e-01, 9.13713054e-02],
       [1.06819375e-01, 1.29256191e-01],
       [9.30725526e-02, 2.03052055e-01],
       [7.55699959e-02, 3.05422631e-01],
       [5.71785784e-02, 4.26795037e-01],
       [4.03156336e-02, 5.56828974e-01],
       [2.64892589e-02, 6.85843305e-01],
       [1.62189340e-02, 8.05933738e-01],
       [9.25403178e-03, 9.11620996e-01],
       [4.92034832e-03, 9.99997656e-01]])
```

In [181]:
```
# to get rid of potential bad initial energy
y_wuhan9[0,1] = 1.03790477e-04
y_wuhan9
```

Out[181]:
```
array([[1.13921429e-03, 1.03790477e-04],
       [3.65929438e-03, 3.03790477e-04],
       [9.45662931e-03, 9.18275759e-04],
       [2.67381648e-02, 2.24160549e-03],
       [3.44802191e-02, 3.41764286e-03],
       [8.72569101e-02, 8.38185542e-03],
       [1.04361234e-01, 1.81767969e-02],
       [1.10064210e-01, 3.55158684e-02],
       [1.14869623e-01, 7.42859773e-02],
       [1.14244321e-01, 9.13713054e-02],
       [1.06819375e-01, 1.29256191e-01],
       [9.30725526e-02, 2.03052055e-01],
       [7.55699959e-02, 3.05422631e-01],
       [5.71785784e-02, 4.26795037e-01],
       [4.03156336e-02, 5.56828974e-01],
       [2.64892589e-02, 6.85843305e-01],
       [1.62189340e-02, 8.05933738e-01],
       [9.25403178e-03, 9.11620996e-01],
       [4.92034832e-03, 9.99997656e-01]])
```

In [182]:
```
y_wuhan9.shape
```

Out[182]:
```
(19, 2)
```

In [183]:
```
with pm.Model() as model10:
    sigma = pm.HalfCauchy('sigma', 1, shape=2)

    # R0 is bounded below by 1 because we see an epidemic has occured
    R0 = pm.Bound(pm.Normal, lower=1)('R0', 2, 3)
    gama = pm.Lognormal('gama', pm.math.log(2), 2)
    beta = pm.Deterministic('beta', gama*R0)

    sir_curves = sir_model(y0=[0.01, 0.0], theta=[beta, gama])

    # data likelihood
    Y = pm.Lognormal('Y', mu=pm.math.log(sir_curves), sd=sigma, observed=y_w

    prior = pm.sample_prior_predictive()
    trace10 = pm.sample(2000, tune=1000, target_accept=0.9, cores=1)
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Sequential sampling (2 chains in 1 job)
NUTS: [gama, R0, sigma]
Sampling chain 0, 0 divergences: 100%|██████████| 3000/3000 [26:59<00:00,  1.85
it/s]
Sampling chain 1, 0 divergences: 100%|██████████| 3000/3000 [26:02<00:00,  1.92
it/s]
The number of effective samples is smaller than 10% for some parameters.
```

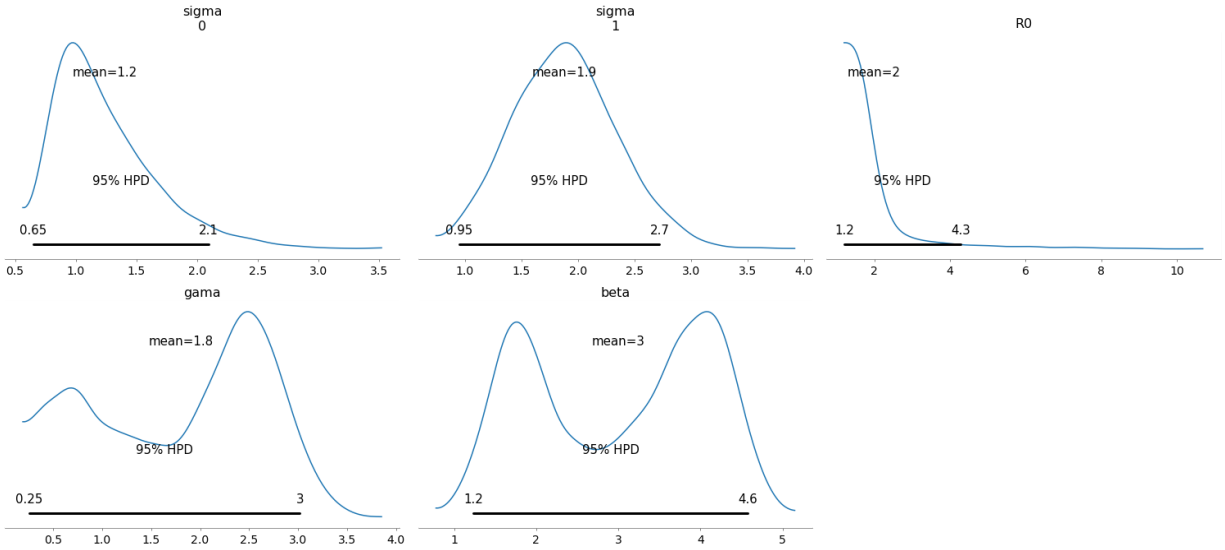
In [184]:
```
with model10:
    posterior_predictive10 = pm.sample_posterior_predictive(trace10)
```

```
100%|██████████| 4000/4000 [02:37<00:00, 25.41it/s]
```

In [185]:
```
import arviz as az

with model10:
    data10 = az.from_pymc3(trace=trace10, prior=prior, posterior_predictive=
```

In [186]:
```
1 az.plot_posterior(data10, round_to=2, credible_interval=0.95)
```

Out[186]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x0000021E56E1B438>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000021E50BC1C88>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000021E52262518>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000021EC5C60B38>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000021EC6DA1E80>],
      dtype=object)
```



So $R_0 = 2.0$ is a lower bound for the Coronavirus in Wuhan (= the R_0 of Ebola).

Conclusion

In this notebook we extrapolated data from Wuhan, China, in order to compute the R_0 of Covid19. Since R_0 depends on social concentrations of the population that an epidemic targets, it is challenging to apply it to different populations. Nevertheless, an upper and lower bound on the epicenter of the epidemic provides us with an estimate of the contagiousness of the virus, which is higher than the flu or SARS. This data was available early on in the epidemic and this extrapolation could have easily been carried out by government agencies throughout, so that countries could have been better prepared for what they're going through now.

Our estimate for R_0 would be in between the high and lower bounds, around 3.5 probably.

Now, countries are forced to enact tough social distancing measures in order to "*flatten the curve*". The need to drastically cut down R_0 is why China has, and Italy, Spain, and the US amonst other nations are undergoing such drastic social distancing measures. *Every person needs to cut down their social interactions*!

This science would not be possible without brave souls, like Li Wenliang, for whom saving his patients was more important than political propaganda. May he be a model for all. We owe him, and many other doctors, nurses, caretakers at the front line of the disease, so much.

Shi (https://www.scientificamerican.com/article/how-chinas-bat-woman-hunted-down-viruses-from-sars-to-the-new-coronavirus1/?utm_source=pocket-newtab) is quite a role model, too!

References

- The SIR Model for Spread of Disease - The Differential Equation Model, David Smith, Lang Moore, 2004 (https://www.maa.org/press/periodicals/loci/joma/the-sir-model-for-spread-of-disease-the-differential-equation-model)
- Appropriate Models for the Management of Infectious Diseases, Helen J Wearing, Pejman Rohani, Matt J Keeling, 2005 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1181873/)
- Power-Law Models for Infectious Disease Spread, Sebastian Meyer and Leonhard Held, 2014 (https://arxiv.org/pdf/1308.5115.pdf)
- The Effect of Disease-Induced Mortality on Structural Network Properties, Lazaros K. Gallos, Nina H. Fefferman, 2015 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4552173/)
- Fitting Epidemic Models to Data, James Holland Jones, 2018 (http://web.stanford.edu/class/earthsys214/notes/fit.html)
- Direct likelihood-based inference for discretely observed stochastic compartmental models of infectious disease, Lam Si Tung Ho, Forrest W. Crawford, Marc A. Suchard, 2018 (https://arxiv.org/pdf/1608.06769.pdf)
- Modeling and inference for infectious disease dynamics: a likelihood-based approach, Carles Bretó, 2018 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5939946/)
- Profile likelihood-based analyses of infectious disease models, Christian Tönsing, Jens Timmer, Clemens Kreutz, 2018 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5939946/), also here (http://sysbio.uni-freiburg.de/ctoensing/paper/Toensing2017_ProfileLikelihoodInfectiousDiseaseModels.pdf)
- Model selection and parameter estimation for dynamic epidemic models via iterated filtering: application to rotavirus in Germany, Theresa Stocks, Tom Britton, Michael Höhle, 2018 (https://academic.oup.com/biostatistics/advance-article/doi/10.1093/biostatistics/kxy057/5108499)
- Parameterizing Spatial Models of Infectious Disease Transmission that Incorporate Infection Time Uncertainty Using Sampling-Based Likelihood Approximations, Rajat Malik, Rob Deardon, Grace P. S. Kwong, 2018 (https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0146253)
- A Survey of the Individual-Based Model applied in Biomedical and Epidemiology Research, Erivelton G Nepomuceno1, Denise F Resende1, Márcio J Lacerda1, 2018 (https://arxiv.org/ftp/arxiv/papers/1902/1902.02784.pdf)
- Dimitri Pananos blog post on novel PyMC3 functionality (https://dpananos.github.io/posts/2019/08/blog-post-21)
- Covid-19 — Navigating the Uncharted, Anthony S. Fauci, M.D., H. Clifford Lane, M.D., and Robert R. Redfield, M.D, 2020 (https://www.nejm.org/doi/full/10.1056/NEJMe2002387)

and remember..

Covid19 is not a *foreign* virus, as some.."*leaders*" have called it. It is *our* virus, and we will beat it *together*, one species, *one* human.