EASC2410 Special Topic

Special Topic: Statistical Analysis and Modeling of the COViD-19 Pandemic

Dr. Binzheng Zhang
Department of Earth Sciences

Review of Lecture 15:

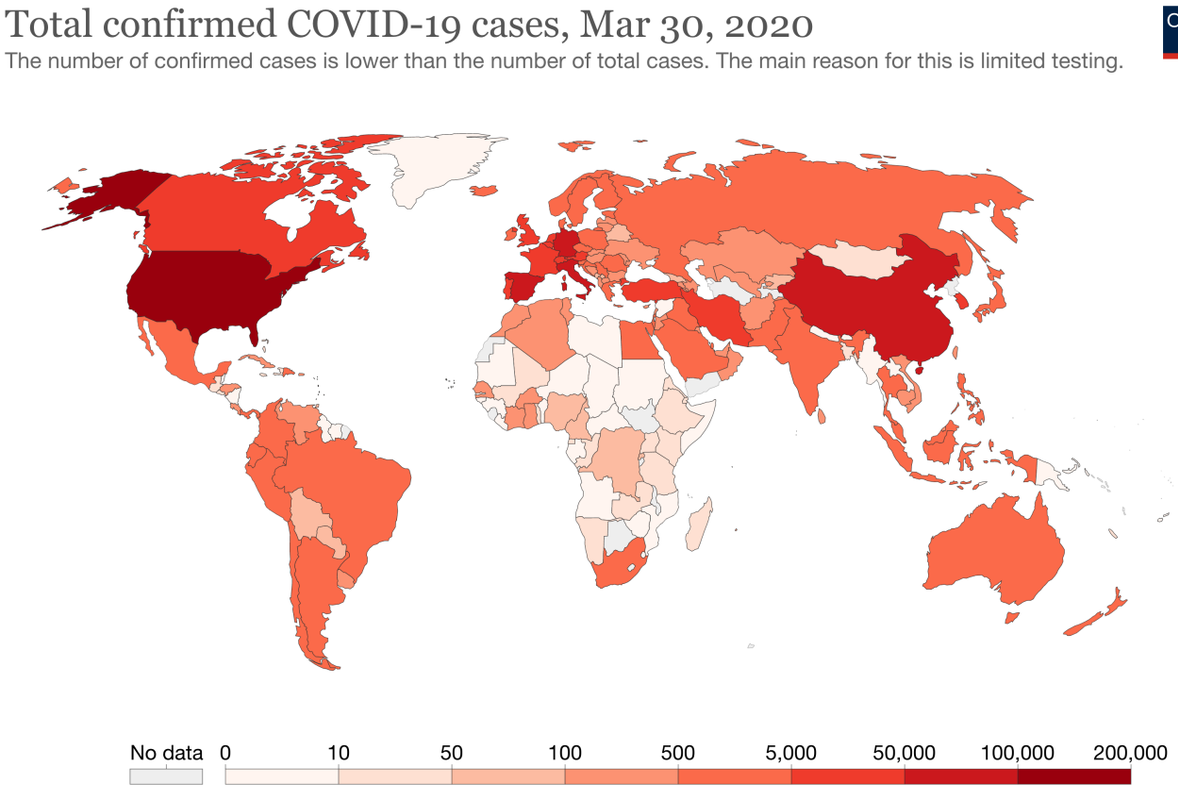
- basic concepts of correlations, covariance, regression
- Python to fit linear models for data sets
- Python to fit non-linear models for data sets
- understanding the quality of the fitting (pcov)

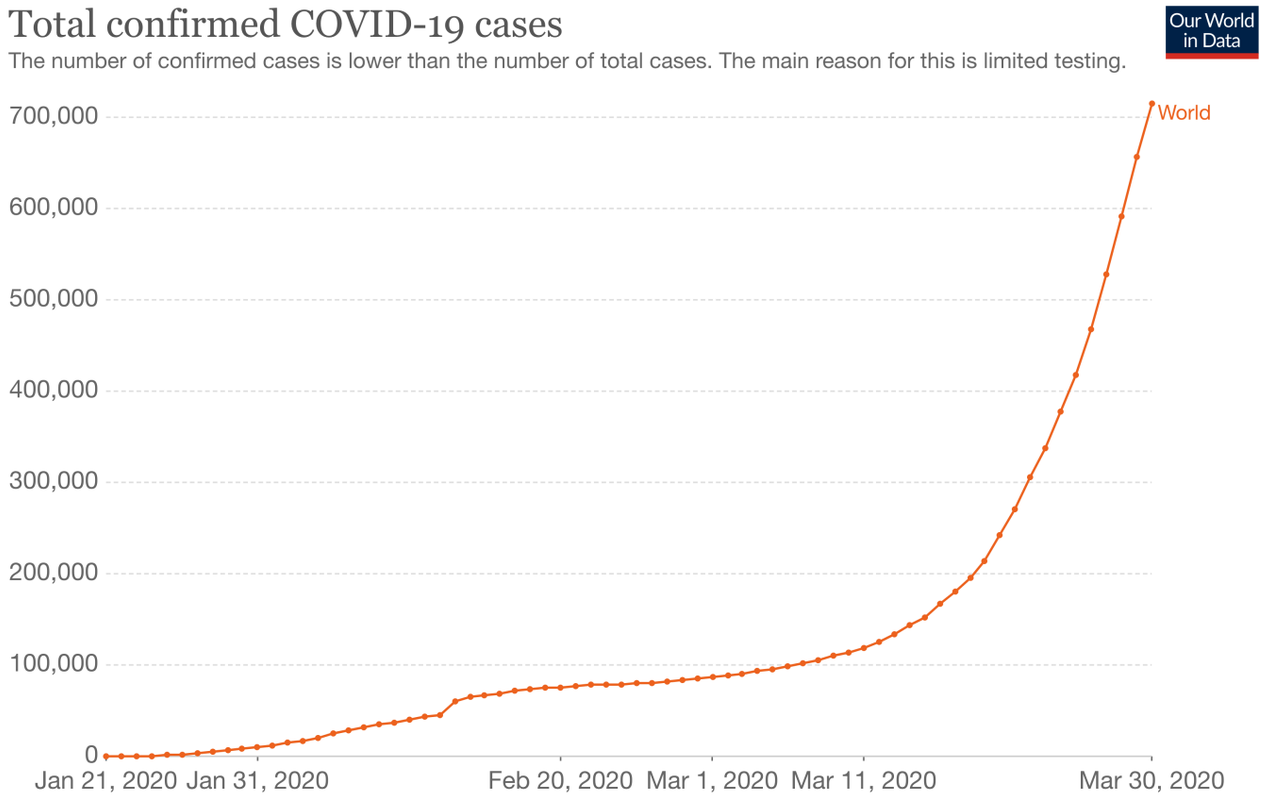
In Lecture 16, we will apply the Python knowledge to analyze the Covid-19 Pandemic, including:

- Program control - if, for loops
- Numpy Arrays
- Pandas data wrangling
- Regression (curve fitting)

The COVID-19 Pandemic

The COVID-19 outbreak is an unprecedented global public health challenge. In order for governments, organizations and individuals to respond to it effectively, it will be vital that they have easy access to good, clear data and a good understanding of what can and can not be said based on the available data.





Data is the key

The COVID-19 Pandemic - Data sources

The Johns Hopkins University data project in Github: https://github.com/CSSEGISandData/COVID-19

List of Available Data Sources:

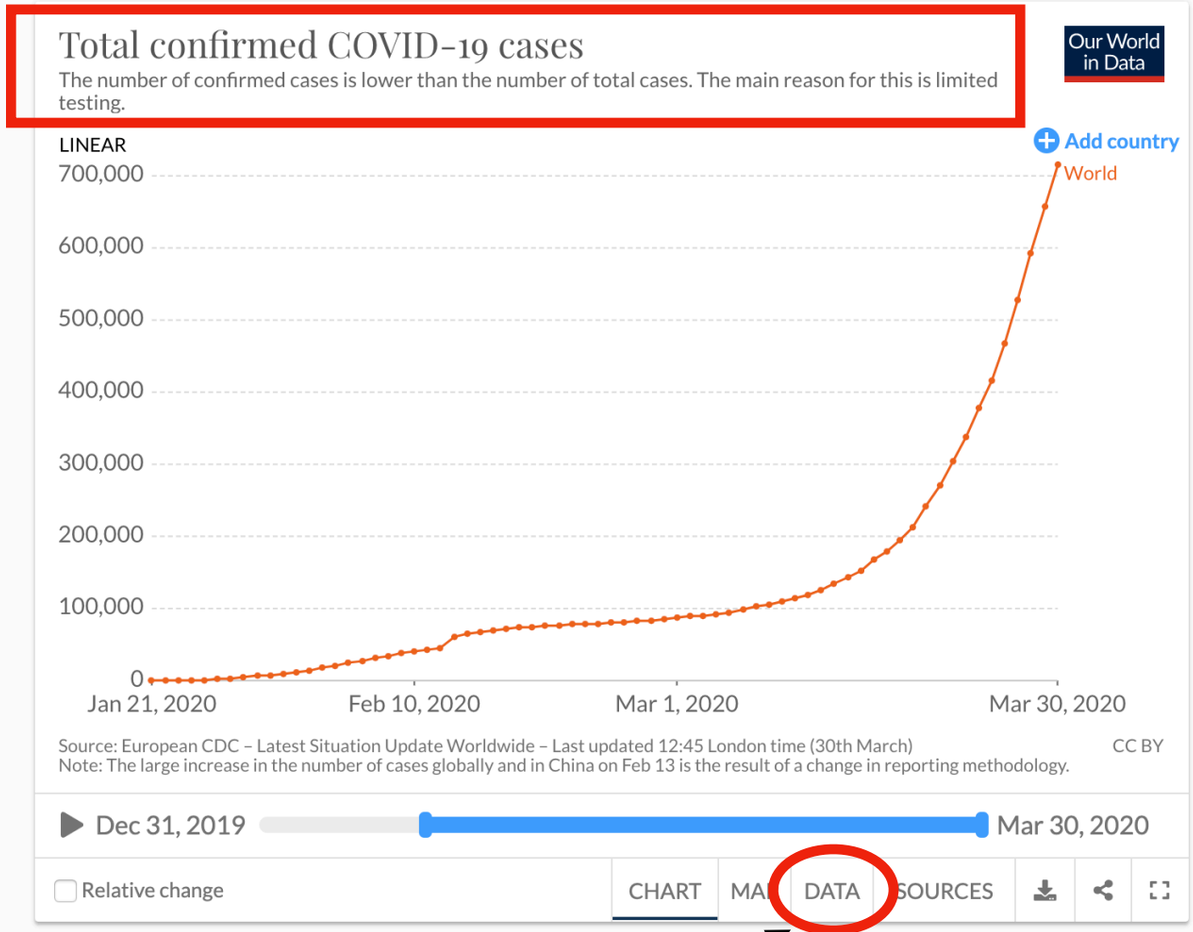
- World Health Organization (WHO): https://www.who.int/
- DXY.cn. Pneumonia. 2020. http://3g.dxy.cn/newh5/view/pneumonia.
- BNO News: https://bnonews.com/index.php/2020/02/the-latest-coronavirus-cases/
- National Health Commission of the People's Republic of China (NHC): http://www.nhc.gov.cn/xcs/yqtb/list_gzbd.shtml
- China CDC (CCDC): http://weekly.chinacdc.cn/news/TrackingtheEpidemic.htm
- Hong Kong Department of Health: https://www.chp.gov.hk/en/features/102465.html
- Macau Government: https://www.ssm.gov.mo/portal/
- Taiwan CDC: https://sites.google.com/cdc.gov.tw/2019ncov/taiwan?authuser=0
- US CDC: https://www.cdc.gov/coronavirus/2019-ncov/index.html
- Government of Canada: https://www.canada.ca/en/public-health/services/diseases/coronavirus.html
- Australia Government Department of Health: https://www.health.gov.au/news/coronavirus-update-at-a-glance
- European Centre for Disease Prevention and Control (ECDC): https://www.ecdc.europa.eu/en/geographical-distribution-2019-ncov-cases
- Ministry of Health Singapore (MOH): https://www.moh.gov.sg/covid-19
- Italy Ministry of Health: http://www.salute.gov.it/nuovocoronavirus
- 1Point3Arces: https://coronavirus.1point3acres.com/en
- WorldoMeters: https://www.worldometers.info/coronavirus/
- Berliner Morgenpost: https://interaktiv.morgenpost.de/corona-virus-karte-infektionen-deutschland-weltweit/

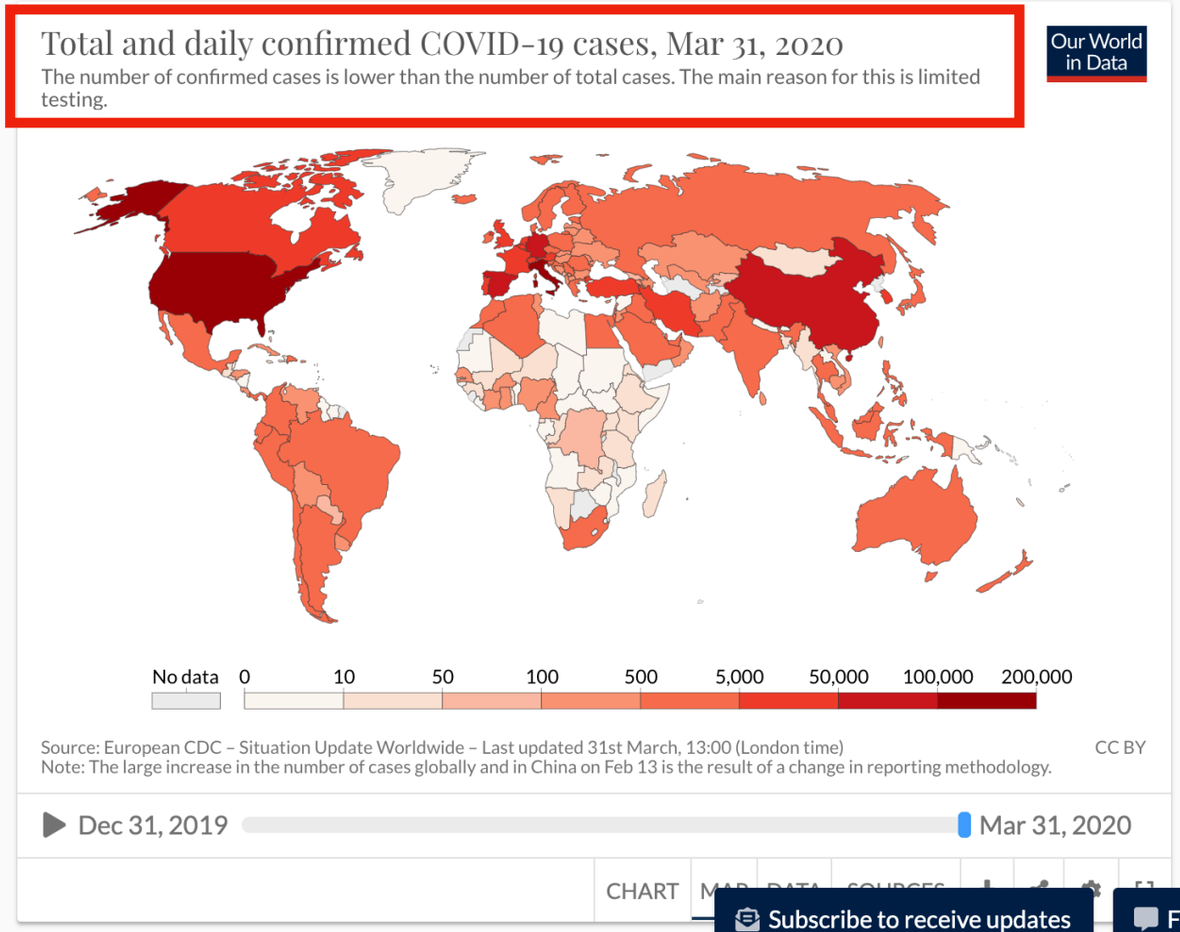
Steps for the data analysis

1. Get the data from one of the sources https://ourworldindata.org/coronavirus

Data: The data shown here is published by the *European Center for Disease Prevention and Control* (ECDC). Here is the documentation of the data and an option to download all data.

The date here reflects to the date of *reporting*, not necessarily the confirmed case figures on that given day. For example, data shown for 25th March presents the latest figures as of 10am CET on 25th March.





Click here to download the csv file

The data shown here is published by the *European Center for Disease Prevention and Control* (ECDC).

Task 1: Load into a Data Frame

Load the file

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

import warnings
warnings.filterwarnings("ignore")

fn = '../Datasets/total-and-daily-cases-covid-19.csv' # file name
df = pd.read_csv(fn) # load file into data frame
df.head() # show the first 10 rows
```

	Entity	Code	Date	Total confirmed cases (cases)	Daily new confirmed cases (cases)
0	Afghanistan	AFG	Dec 31, 2019	0	0
1	Afghanistan	AFG	Jan 1, 2020	0	0
2	Afghanistan	AFG	Jan 2, 2020	0	0
3	Afghanistan	AFG	Jan 3, 2020	0	0
4	Afghanistan	AFG	Jan 4, 2020	0	0

Need a couple of things:

1. **The column names are too long**
2. **The Date column here is string**

Change the column names

```
# first change the column names slightly
df.rename(columns={'Total confirmed cases (cases)':'TotalCases',\
                   'Daily new confirmed cases (cases)':'DailyCases'},\
                    inplace=True)
df.head()
```

	Entity	Code	Date	TotalCases	DailyCases
0	Afghanistan	AFG	Dec 31, 2019	0	0
1	Afghanistan	AFG	Jan 1, 2020	0	0
2	Afghanistan	AFG	Jan 2, 2020	0	0
3	Afghanistan	AFG	Jan 3, 2020	0	0
4	Afghanistan	AFG	Jan 4, 2020	0	0

Task 2: Adding Date information in the data frame

```
1  import time
2  from datetime import datetime,timedelta
3
4  # now let's convert the date strings to datetime object by adding a new column 'DT'
5  df['DT']=pd.to_datetime(df.Date)
6  df.head()
```

The “datetime” module, dealing with time series

	Entity	Code	Date	TotalCases	DailyCases	DT
0	Afghanistan	AFG	Dec 31, 2019	0	0	2019-12-31
1	Afghanistan	AFG	Jan 1, 2020	0	0	2020-01-01
2	Afghanistan	AFG	Jan 2, 2020	0	0	2020-01-02
3	Afghanistan	AFG	Jan 3, 2020	0	0	2020-01-03
4	Afghanistan	AFG	Jan 4, 2020	0	0	2020-01-04

Now we have a new data column named “DT”

You can now do data/time manipulations with this column data, full documentation is here: https://docs.python.org/3/library/datetime.html

For example let’s calculate the number of days since 2019-12-31, which will be used in line plots later

```
1  # now let's compute the number of days since 2019-12-31
2
3  date = df['DT'] # push the datetime column 'DT' into a series
4
5  date_zero = datetime.strptime('2019-12-31','%Y-%m-%d') # starting time
6
7  df['days'] = date.map(lambda x : (x - date_zero).days) # compute the delta time
8
9  df.head()
```

Create a start time at 2019-12-31

Map every element in ‘date’ into days using the lambda function by calculating the difference between the current date x and the date_zero; Then put it back to the data frame and name it as ‘days’

	Entity	Code	Date	TotalCases	DailyCases	DT	days
0	Afghanistan	AFG	Dec 31, 2019	0	0	2019-12-31	0
1	Afghanistan	AFG	Jan 1, 2020	0	0	2020-01-01	1
2	Afghanistan	AFG	Jan 2, 2020	0	0	2020-01-02	2
3	Afghanistan	AFG	Jan 3, 2020	0	0	2020-01-03	3

Task 3: Generate a time series of the total cases

Let's try the .plot() function of data frames:

```
DataFrame.plot(x=None, y=None, kind='line', ax=None, subplots=False,
               sharex=None, sharey=False, layout=None,figsize=None,
               use_index=True, title=None, grid=None, legend=True,
               style=None, logx=False, logy=False, loglog=False,
               xticks=None, yticks=None, xlim=None, ylim=None, rot=None,
               xerr=None,secondary_y=False, sort_columns=False, **kwds)
```

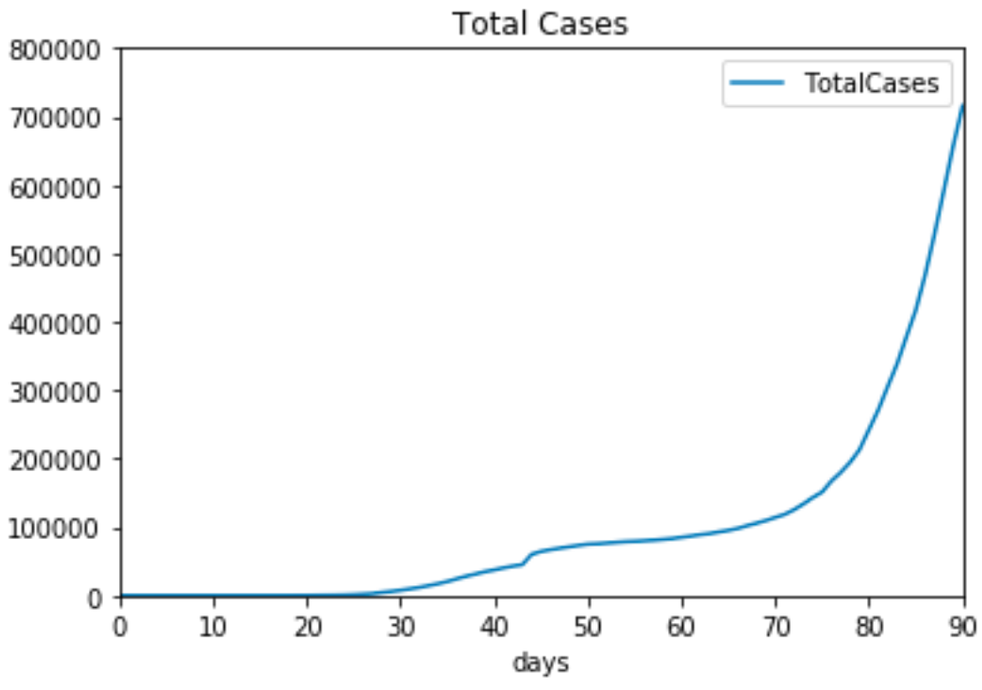
We only want the total (filtering)

x axis, using the 'days' column

y axis, using the 'TotalCases' column

```
df[df.Entity=='World'].plot('days','TotalCases',\
kind='line',title='Total Cases',\
ylim=[0,800000],xlim=[0,df.days.max()] )

plt.show()
```

Specifying the properties of the plot



You can also use the plt.plot() function doing the same thing:

```
plt.plot(df[df.Entity=='World'].days, df[df.Entity=='World'].TotalCases)
plt.xlabel('Days since '+str(date_zero))
plt.ylabel('Cases')
plt.title('World cases in total')
plt.xlim([0,90])
plt.ylim([0,800000])
plt.show()
```

Task 3: Generate a time series of the total cases

We can also plot the total cases as a function of the date object column “DT” (not the “Date” column!):

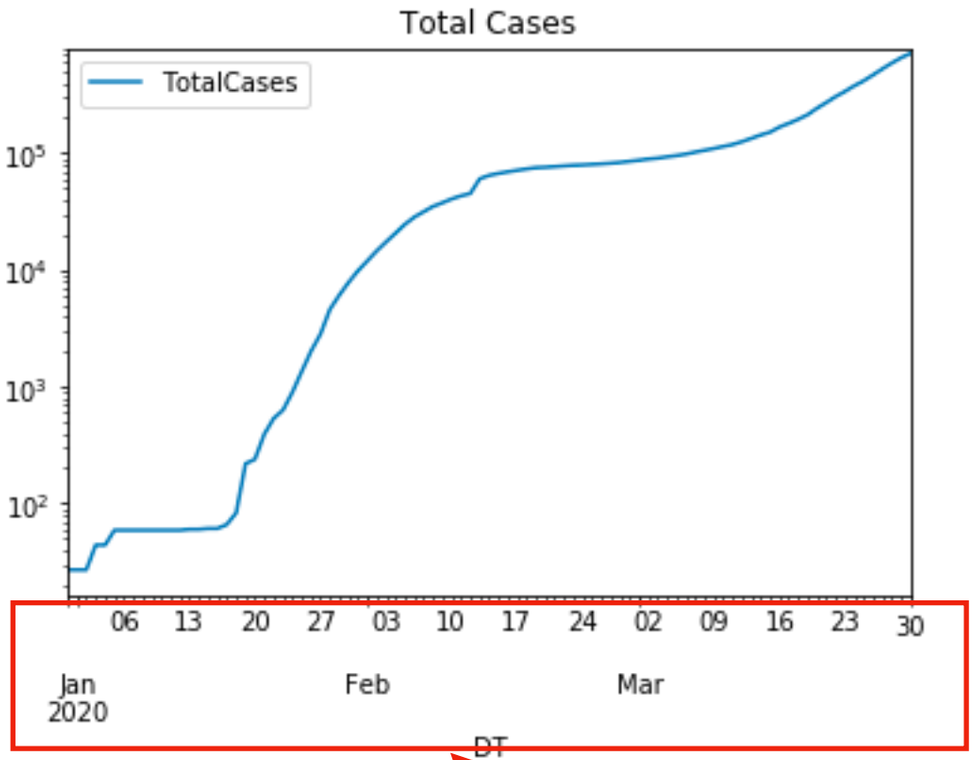
We only want the total (filtering)

x axis, using the ‘DT’ column

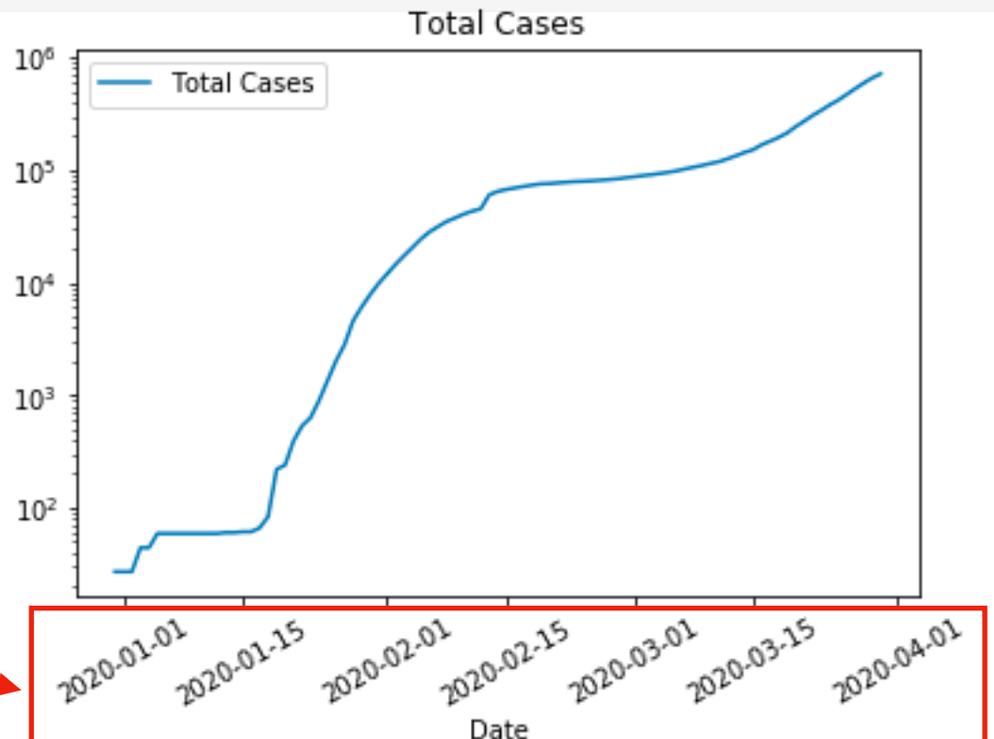
y axis, using the ‘TotalCases’ column

```
df[df.Entity=='World'].plot('DT','TotalCases',\
                            kind='line',title='Total Cases',\
                            ylim=[0,800000],logy=True )
plt.legend()
plt.show()
```

Changing the y scale to be logarithm




```
plt.plot(df[df.Entity=='World'].DT,df[df.Entity=='World'].TotalCases,\
         label='Total Cases')
plt.xticks(rotation=30)
plt.xlabel('Date')
plt.title('Total Cases')
plt.yscale('log')
plt.legend()
plt.show()
```

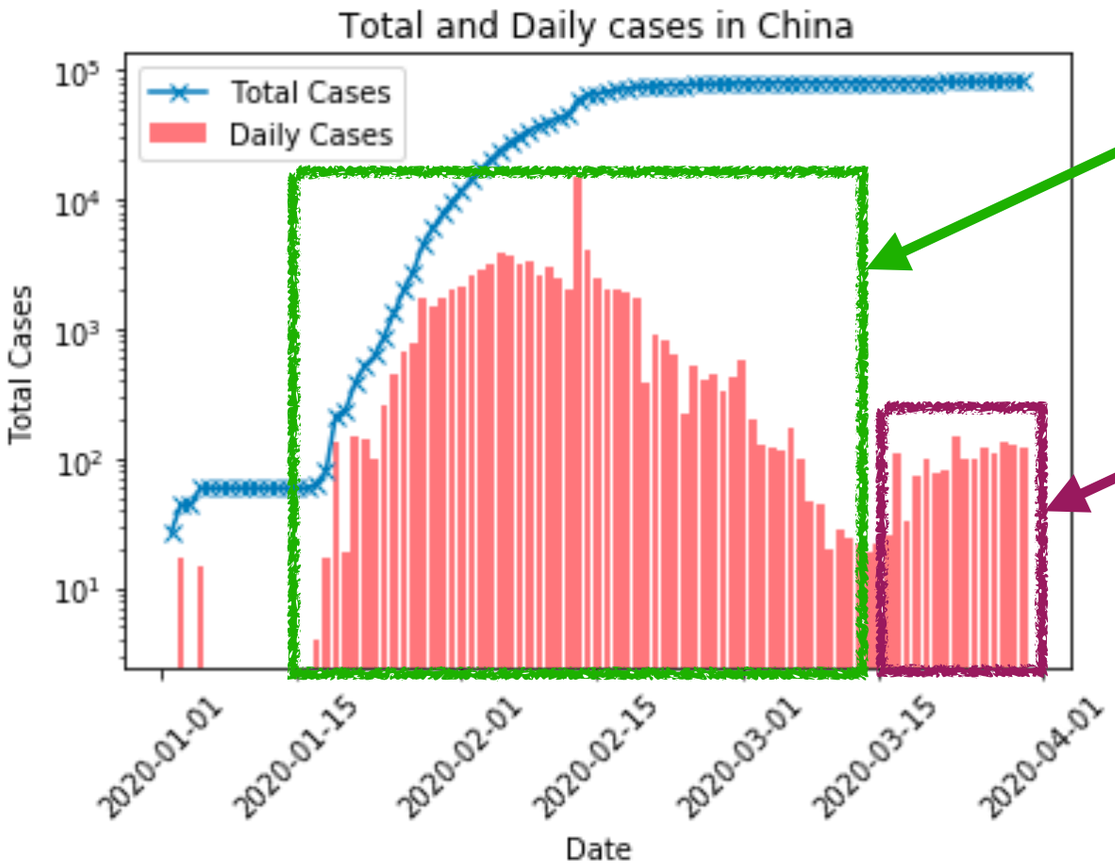


Pay attention to the differences in the xlabel

Task 4: Generate a time series with more data together

We can also plot the total cases together with the total daily new cases (using the plt.plot() function):

```
C = 'China' # let's filter the data to only look at the results from China

date_zero = datetime.strptime('2020-1-1','%Y-%m-%d') # starting time, datetime object

df_china = df[(df.Entity==C)&(df.DT>date_zero)] # filter the data to only have China after 2020-1-1

plt.plot(df_china['DT'],df_china['TotalCases'],'-x',label='Total Cases') # you can use the datetime directly in a
plt.bar(df_china['DT'],df_china['DailyCases'],color='r',label='Daily Cases',alpha=0.5) # you can use the datetime

plt.yscale('log')
plt.xticks(rotation=45) # the rotation option basically tilts the lables 45 degrees so they don't get clustered
plt.xlabel('Date')
plt.ylabel('Total Cases')
plt.title('Total and Daily cases in China')
plt.legend()
plt.show()
```



Mostly Domestic cases (before Mar 10, 2020)

Mostly Imported cases (after Mar 10, 2020)

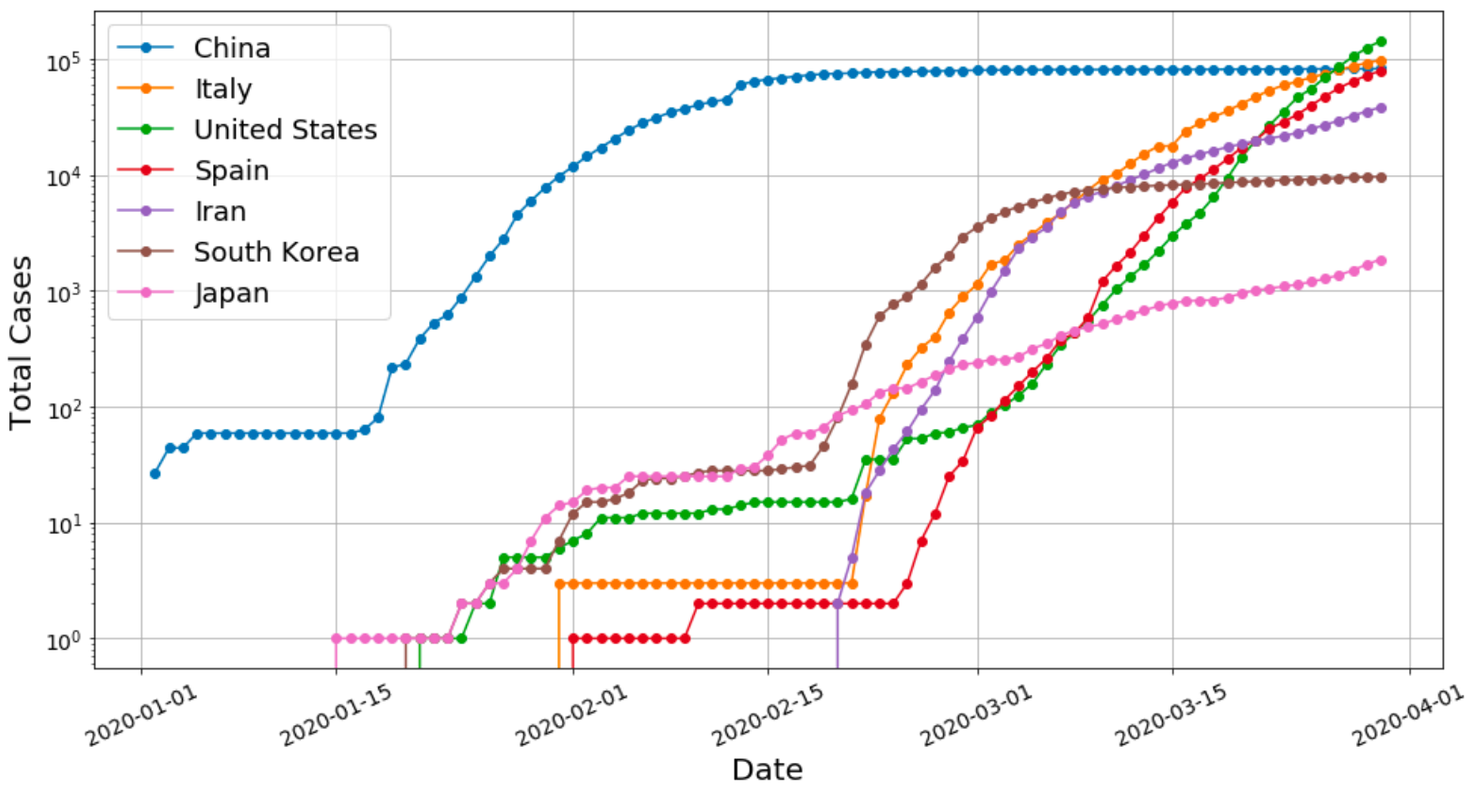
Task 4: Generate time series for multiple nations

Now let's put the time series of total confirmed cases in the same figure using a iterative method

```
countries = ['China','Italy','United States','Spain','Iran','South Korea','Japan'] # list of interested countries
date_zero = datetime.strptime('2020-1-1','%Y-%m-%d') # starting time, datetime object

plt.figure(figsize=(16,8)) # specify figure size

for C in countries: # step through the list of countries

    df_c = df[(df.Entity==C)&(df.DT>date_zero)] # filter the data to only have country C after date_zero

    plt.plot(df_c['DT'],df_c['TotalCases'],'-o',label=C) # plot the data

plt.yscale('log') # change the y scale to be logarithm
plt.xticks(rotation=25,fontsize=14)
plt.yticks(fontsize=14)
plt.xlabel('Date',fontsize=20)
plt.ylabel('Total Cases',fontsize=20)
plt.legend(fontsize=18)
plt.grid()
```

Looping over the list of "countries" to do two things:

1) filter the data frame using the country name C and the date (after date_zero)
2) Plot the TotalCases as a function of DT



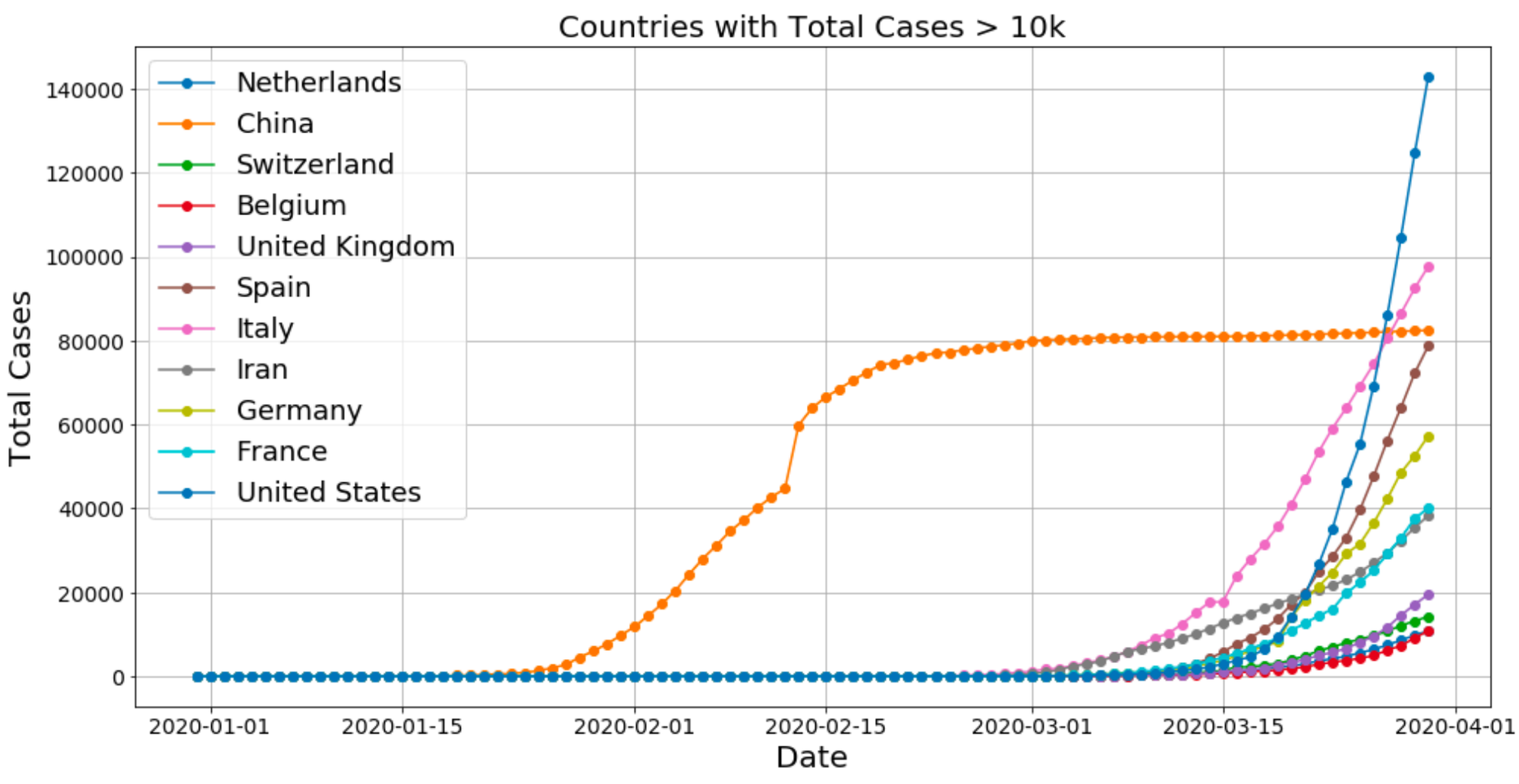
	Entity	Code	Date	TotalCases	DailyCases	DT	days
0	Afghanistan	AFG	Dec 31, 2019	0	0	2019-12-31	0
1	Afghanistan	AFG	Jan 1, 2020	0	0	2020-01-01	1
2	Afghanistan	AFG	Jan 2, 2020	0	0	2020-01-02	2
3	Afghanistan	AFG	Jan 3, 2020	0	0	2020-01-03	3

Now we can clearly see the evolution of total cases in different countries

Task 5: Generate time series with selection criteria

Now let's put the time series of total confirmed cases in the same based on total cases > 10000

```
countries = set(df.Entity) # list of all countries - NOTE: set() converts the dt.Entity column to a set
                           # sets are unique, another way to get all the countries is to use np.unique()
date_zero = datetime.strptime('2020-3-30','%Y-%m-%d') # starting time, datetime object

plt.figure(figsize=(16,8))

for C in countries: # step through the list of countries

    if ( df[df.Entity==C].TotalCases.max()>10000 )& (C!='World'): # condition: 1. Country C has max cases> 10000
                                                                  #            2. Country C is not 'World'
        df_c = df[(df.Entity==C)] # filter the data
        plt.plot(df_c['DT'],df_c['TotalCases'],'-o',label=C) # plot the data

plt.yscale('linear')
```



Here I used a linear scale

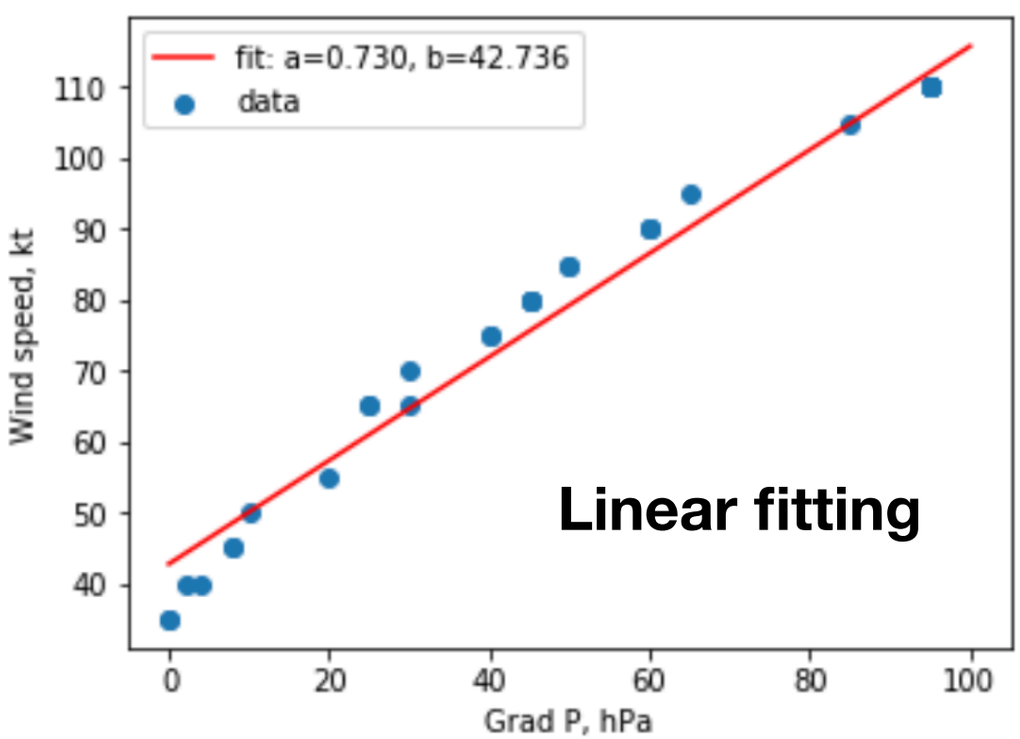
Fitting the Data to Mathematical Functions

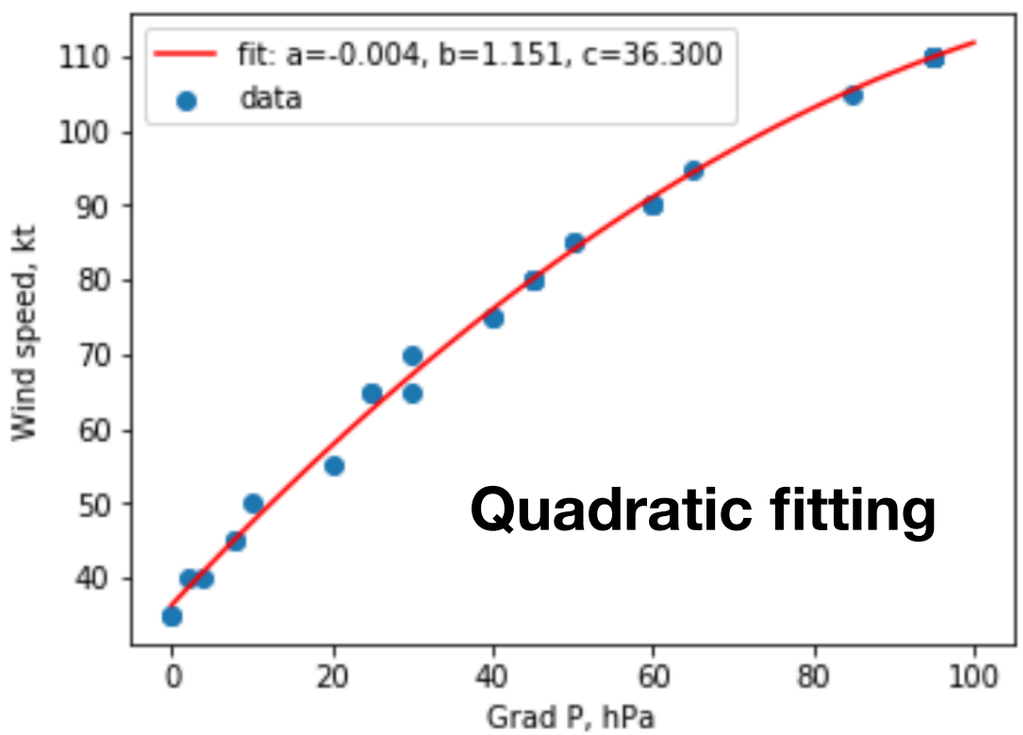
Recall:

```
def func_linear(x, a, b):
    return a * x + b
```

Linear function (y=ax+b)

```
plt.figure(figsize=(12,4))
# linear fit
popt, pcov = curve_fit(func_linear, mg['gradP'], mg['Wind'])

plt.subplot(1,2,1)
plt.scatter(mg.gradP, mg.Wind,label='data') # make a plot showing the relationship between gradP and Wind
plt.xlabel('Grad P, hPa')
plt.ylabel('Wind speed, kt')

x=np.linspace(0,100,50)

plt.plot(x, func_linear(x, *popt), 'r-',label='fit: a=%5.3f, b=%5.3f' % tuple(popt))
plt.legend()
```



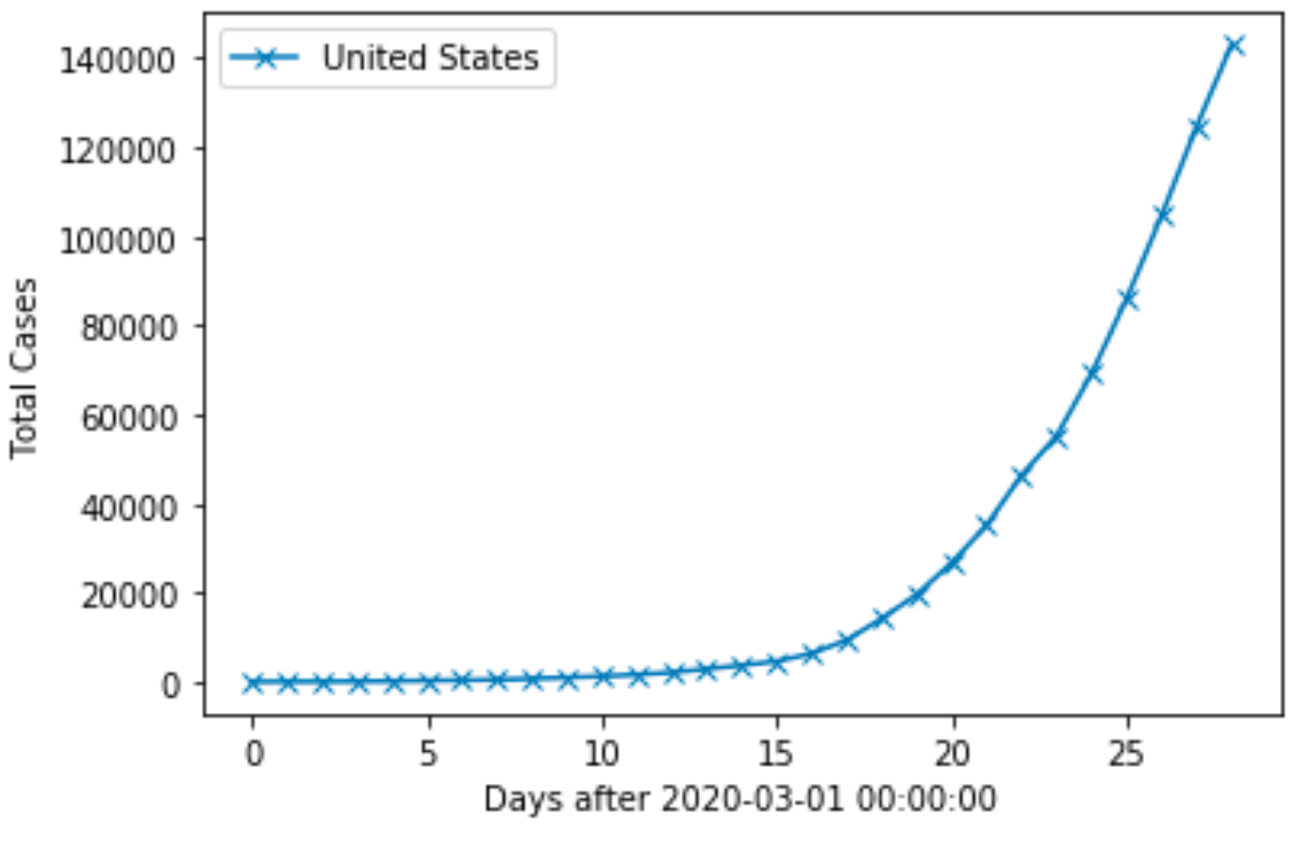


Fitting the Data to Mathematical Functions

Let's try fit the TotalCase in the US using a linear function (y=ax+b)

First let's plot the data from the US, starting from 2020-3-1:

```
C = 'United States' # let's filter the data to only look at the results from the US

date_zero = datetime.strptime('2020-3-1','%Y-%m-%d') # starting time, datetime object
df_us = df[(df.Entity==C)&(df.DT>date_zero)] # filter the data frame by Entity and DT

# now let's plot a time series, say the total cases in the US as a function of time
days = np.arange(df_us['DT'].size)
plt.plot(days,df_us['TotalCases'],'-x',label=C) # you can use the datetime directly in a line plot
plt.yscale('linear')
plt.xlabel('Days after '+ str(date_zero))
plt.ylabel('Total Cases')
plt.legend()
plt.show()
```



df_us.head( )

	Entity	Code	Date	TotalCases	DailyCases	DT	days
7877	United States	USA	Mar 2, 2020	89	20	2020-03-02	62
7878	United States	USA	Mar 3, 2020	103	14	2020-03-03	63
7879	United States	USA	Mar 4, 2020	125	22	2020-03-04	64
7880	United States	USA	Mar 5, 2020	159	34	2020-03-05	65
7881	United States	USA	Mar 6, 2020	233	74	2020-03-06	66

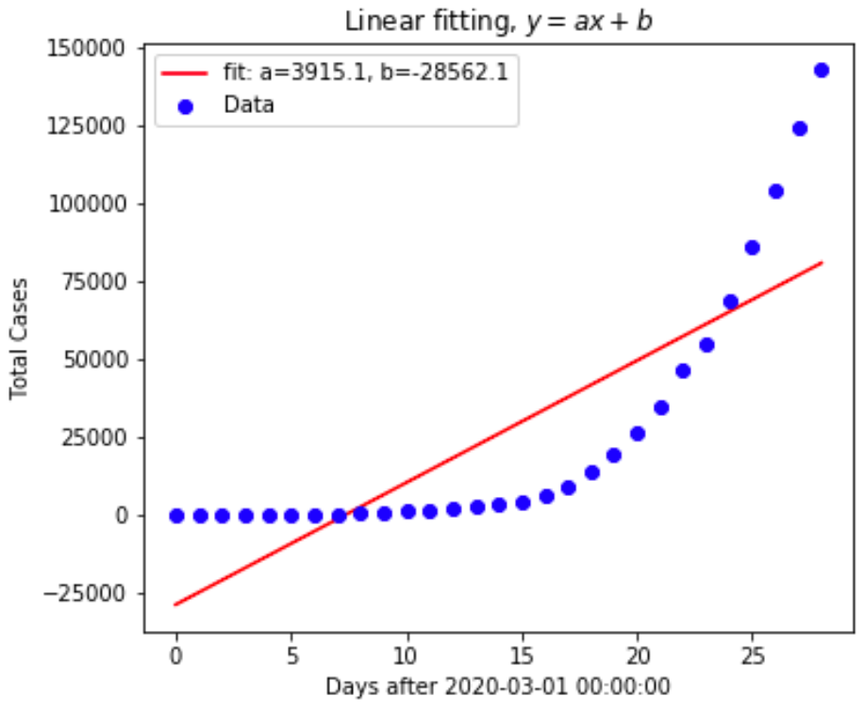
Here the array "days" is a numpy array, which has the same size as the column data 'DT' (or 'TotalCases'), so that it can be used to plot and fit a mathematical function easily.

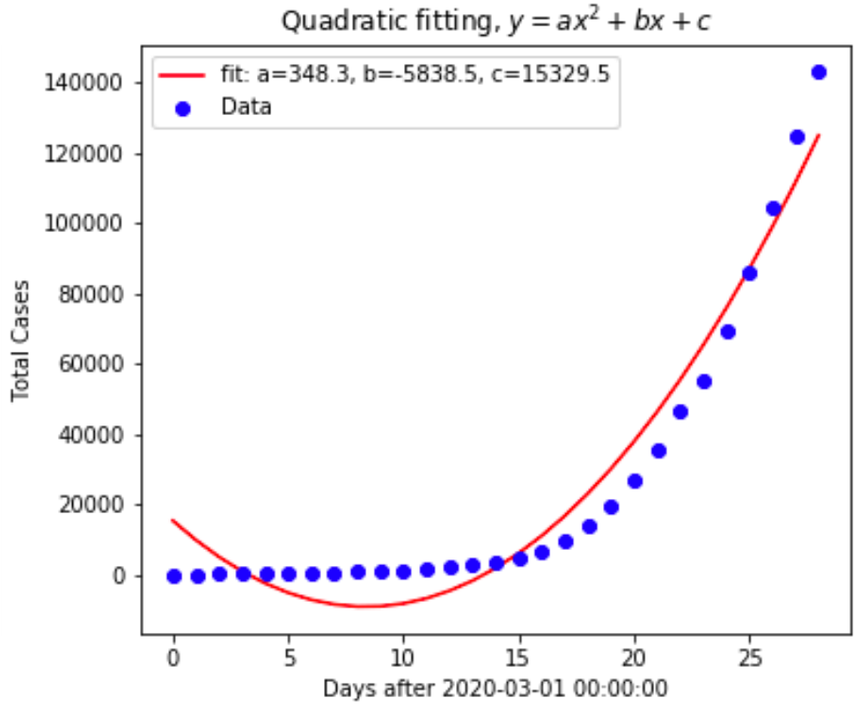
days[0] = 0, which is date 2020-3-2, and so on

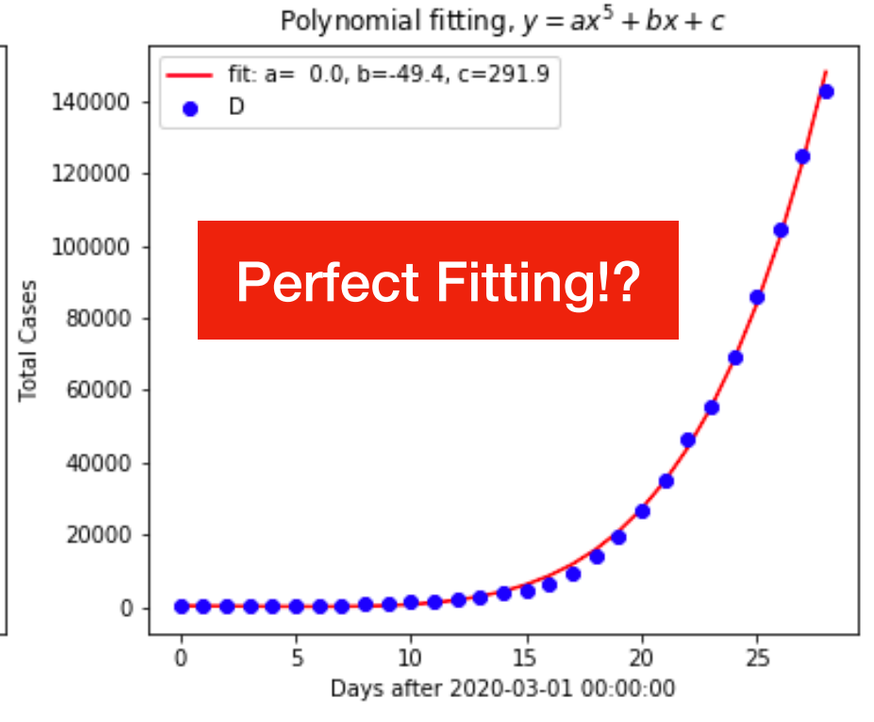
Fitting the Data to Mathematical Functions

Let's try fit the TotalCase in the US using a linear function (y=ax+b)

```
from scipy.optimize import curve_fit # import function curve_fit

# define a linear function
def func_linear(x, a, b):
    return a *x + b

plt.figure(figsize=(16,4))
plt.subplot(1,3,1)
popt, pcov = curve_fit(func_linear, days, df_us['TotalCases']) # curve fitting
plt.plot(days, func_linear(days, *popt), 'r-', label='fit: a=%5.1f, b=%5.1f' % tuple(popt)) # Plot the results
plt.plot(days, df_us['TotalCases'], 'bo', label='Data') # plot the original data
plt.xlabel('Days after '+ str(date_zero))
plt.ylabel('Total Cases')
plt.title('Linear fitting, $y = ax+b$')
plt.legend()
```





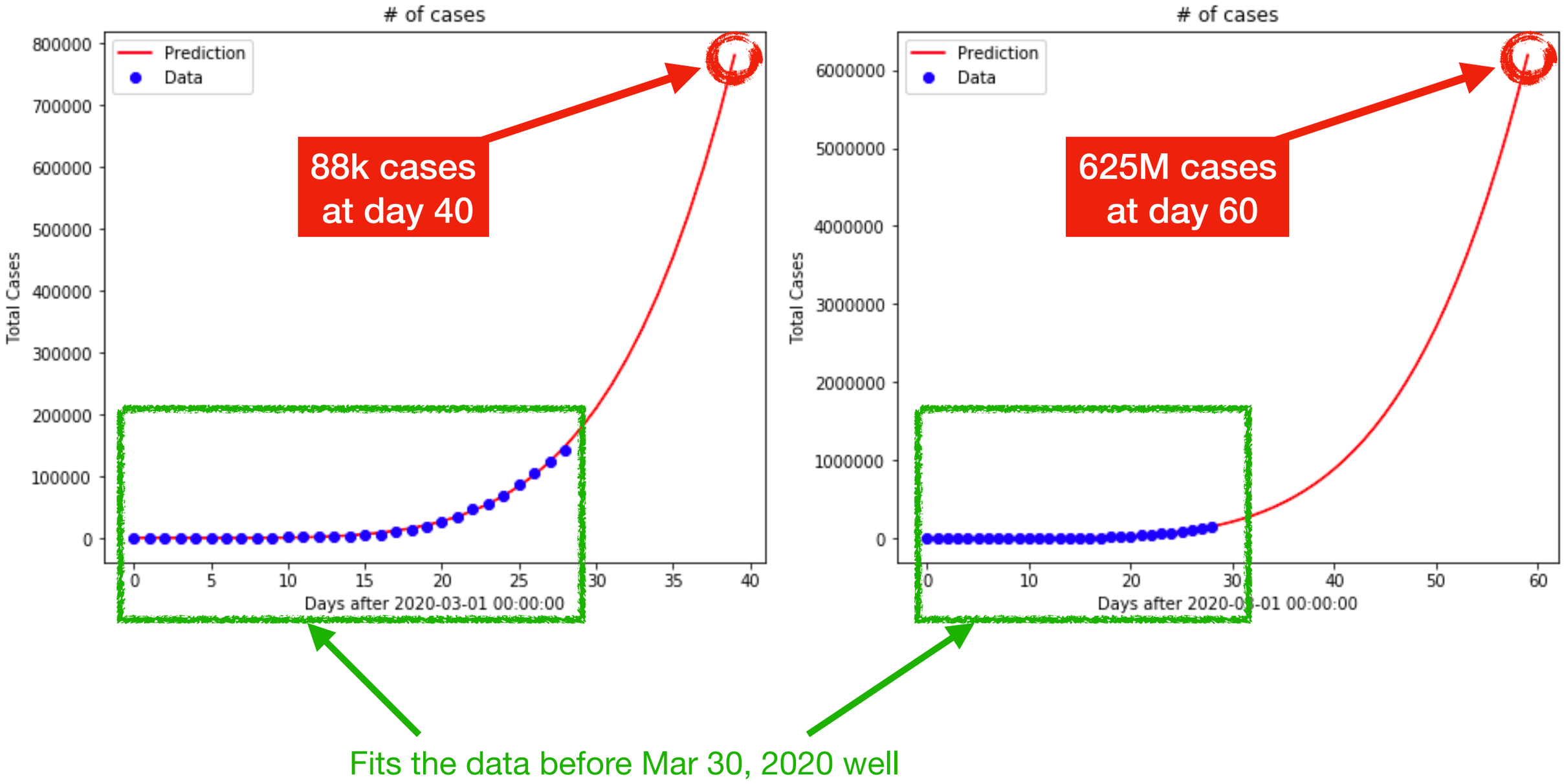



```
# define a quadratic function
def func_quad(x, a, b, c):
    return a *x**2 + b*x + c
```

```
# define a polynomial (say, n=5) func
def func_poly(x, a, b, c):
    return a *x**5 + b*x + c
```

Fitting the Data to Mathematical Functions

Let's try extrapolate the polynomial fitting to "predict" cases after 10 and 30 days:



Fits the data before Mar 30, 2020 well

Although we don't know the answer yet, but the numbers "predicted" are likely over estimations because with measures the growth should slow down. Again, a good fit doesn't mean good science!

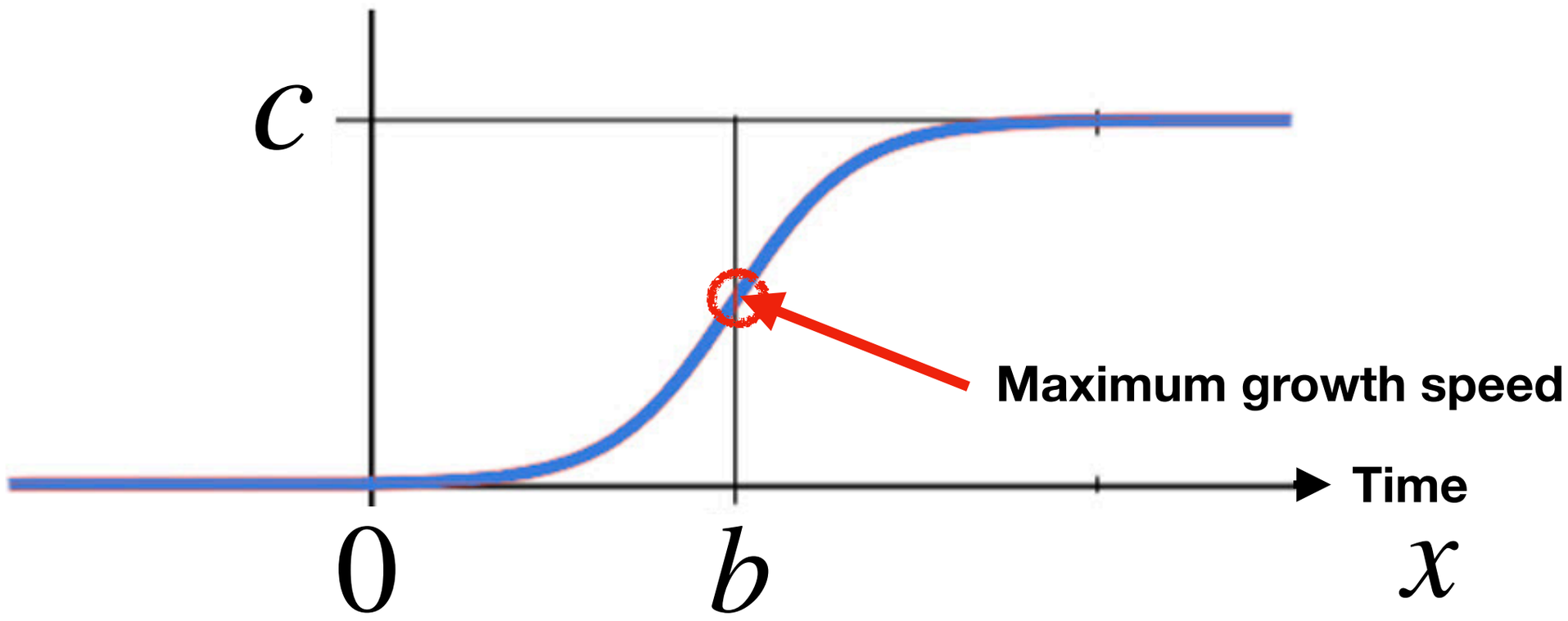
Fitting the Data to Mathematical Functions

Let's try fit the TotalCase using a logistic function

$$f(x) = \frac{c}{1 + e^{-(x-b)/a}}$$

In this formula, we have the variable x that is the time and three parameters: a,b,c.

- a refers to the infection speed (aka, the R_0 value, for SARS-CoV it was ~3)
- b is the day with the maximum infection growth occurred
- c is the total number of recorded infected people at the infection's end



Fitting the Data to Mathematical Functions

Let's try fit the TotalCase from China using a logistic function (it's almost over)

Step1: filter the data frame to select *Entity*=='*China*' and DT>= '2020-1-1'

```
C = 'China'
date_zero = datetime.strptime('2020-1-1','%Y-%m-%d') # starting time, datetime object
df_c = df[(df.Entity==C)&(df.DT>=date_zero)] # filter the China data, after 1-1-2020
```

Step2: define a logistic function f(x,a,b,c) for fitting

```
# define a logistic function for curve fitting
def logistic_model(x,a,b,c):
    return c/(1+np.exp(-(x-b)/a))
```

$$f(x) = \frac{c}{1 + e^{-(x-b)/a}}$$

Step3: curve fitting use the days and 'TotaCases'

```
days = np.arange(df_c['DT'].size) # days is a new array from starting from 0, better for fitting
popt,pcov = curve_fit(logistic_model,days, df_c['TotalCases'],p0=[3.8,40,80000])
```

Note: 1) **days** here is a new numpy array that has the same size as the df_c['DT'] column;
2) the **days** array starts from 0
3) p0 here is the initial guess of the (a,b,c) values (optional), default None
4) sometimes if you don't give a p0 value, the curve fitting fails and you get 0

How to estimate p0? Use the meanings of a, b and c!

Estimate the p0 values (initial guess)

Recall the mathematical form of the logistic function

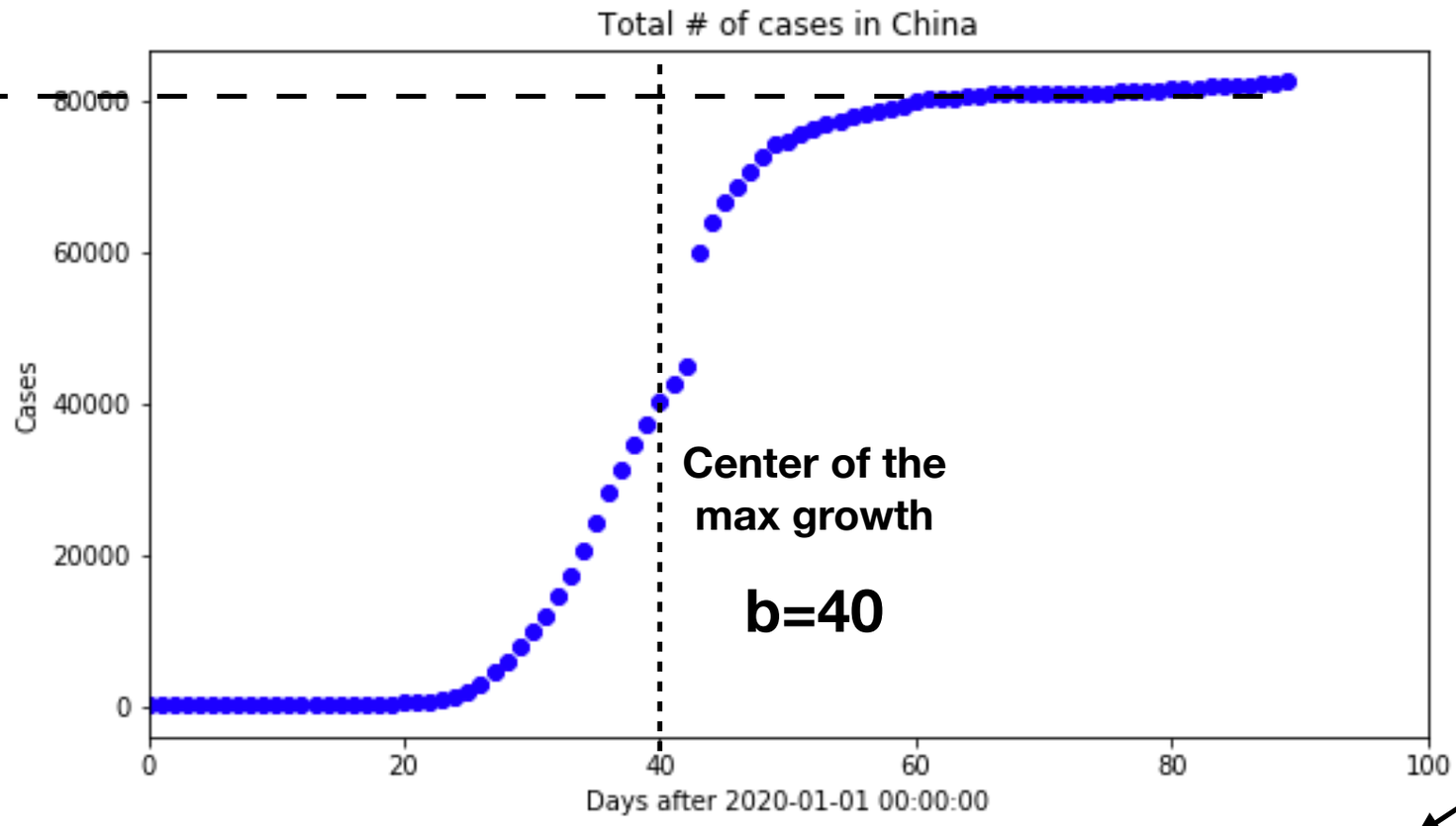
$$f(x) = \frac{c}{1 + e^{-(x-b)/a}}$$

In this formula, we have the variable x that is the time and three parameters: a,b,c.

- a refers to the infection speed (aka, the R_0 value, for SARS-CoV it was ~3)
- b is the day with the maximum infection growth occurred
- c is the total number of recorded infected people at the infection's end

C = 80000

Saturate value



Center of the max growth

b=40

a=3.5

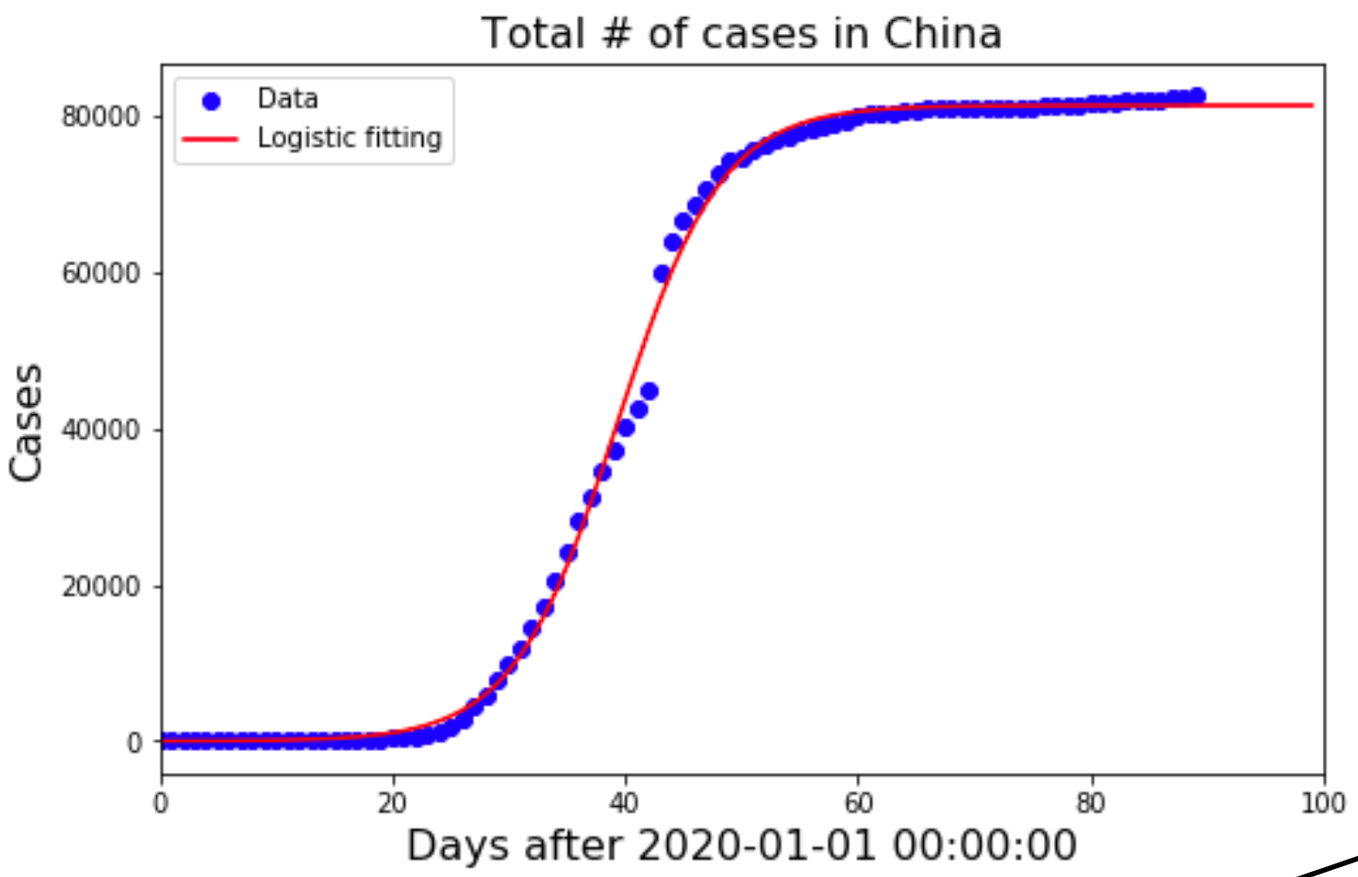
R0 value of SARS (science)

p0 = [a, b, c]

```
popt,pcov = curve_fit(logistic_model,days, df_c['TotalCases'],p0=[3.8,40,80000])
```

Logistic Fitting Results

Plot the fitting with data



Quality of the fitting

```
popt =  [4.47579086e+00 3.93352916e+01 8.12437920e+04]
-------------------------------
pcov=
[[8.07195007e-03 1.38992331e-03 8.41011151e+00]
 [1.38992331e-03 1.08155664e-02 1.08511870e+01]
 [8.41011151e+00 1.08511870e+01 6.56578998e+04]]
-------------------------------
According to the fitting results:
a = 4.48 +/- 0.09 (R0 Value)
b = 39.34 +/- 0.1 (Peak Growth Date)
c = 81243.79 +/- 256.24 (Capacity)
```

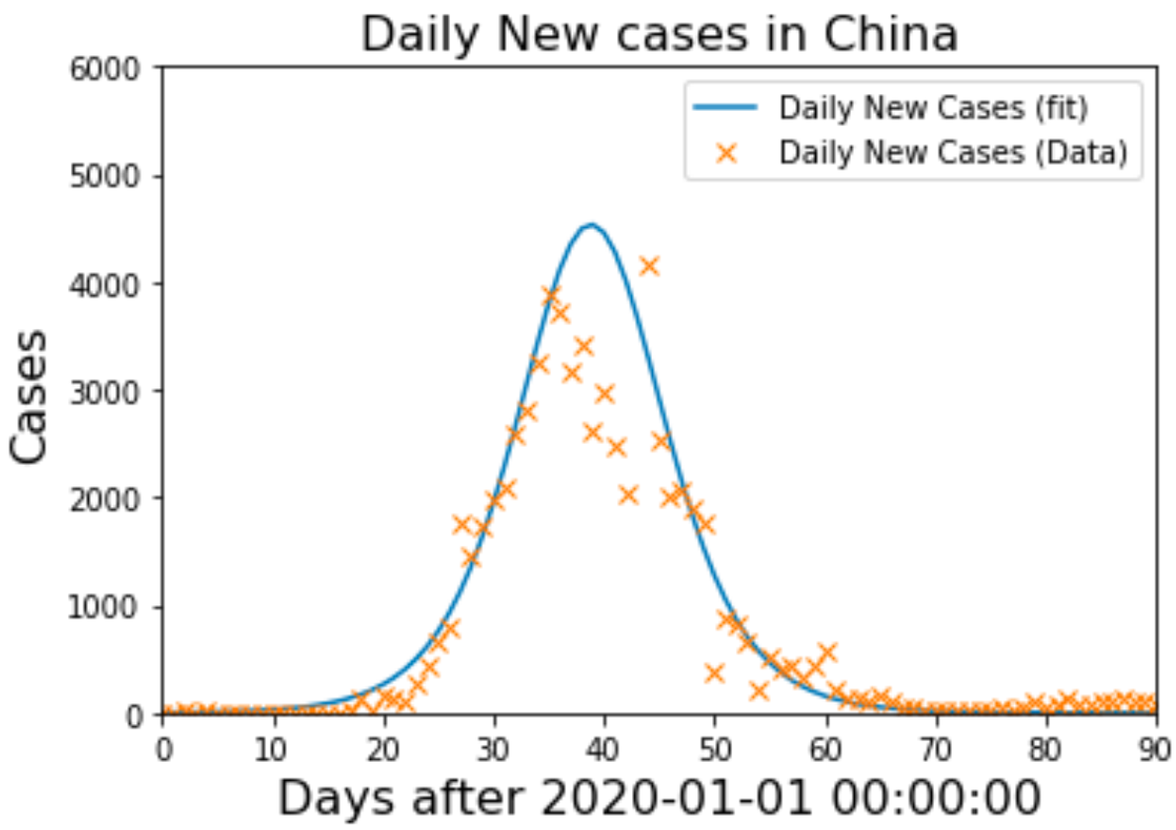
$$pcov = \begin{bmatrix} cov(a,a) & cov(a,b) & cov(a,c) \\ cov(b,a) & cov(b,b) & cov(b,c) \\ cov(c,a) & cov(c,b) & cov(c,c) \end{bmatrix}$$

```
1  print('popt = ',popt)
2  print('-------------------------------')
3  print('pcov=')
4  print(pcov)
5  print('-------------------------------')
6  print('According to the fitting results:')
7  print("a =", np.round(popt[0],2), "+/-", np.round(pcov[0,0]**0.5,2),'(R0 Value)')
8  print("b =", np.round(popt[1],2), "+/-", np.round(pcov[1,1]**0.5,2),'(Peak Growth Date)')
9  print("c =", np.round(popt[2],2), "+/-", np.round(pcov[2,2]**0.5,2),'(Capacity)')
```

Logistic Fitting Results (Daily New Cases)

```
y=logistic_model(days,*popt)
plt.plot(np.diff(y),label='Daily New Cases (fit)')
days = np.arange(df_c['DT'].size)
plt.plot(days,df_c['DailyCases'],'x',label='Daily New Cases (Data)')
plt.legend()
plt.xlabel('Days after '+ str(date_zero),fontsize=16)
plt.ylabel('Cases',fontsize=16)
plt.title('Daily New cases in '+ C,fontsize=16)
plt.xlim([0,90])
plt.ylim([0,6000])
plt.show()
```

Here y is the array of the fit curve, so numpy provides a handy function .diff( ) computing the delta between to data points, which is basically the ‘daily new cases’ column in the data frame:



Seems like the logistic fitting works quite well.

Now the question is: are these fitting curves useful in reality in terms of predicting cases in the future?

Time to try analyze the data on your own!