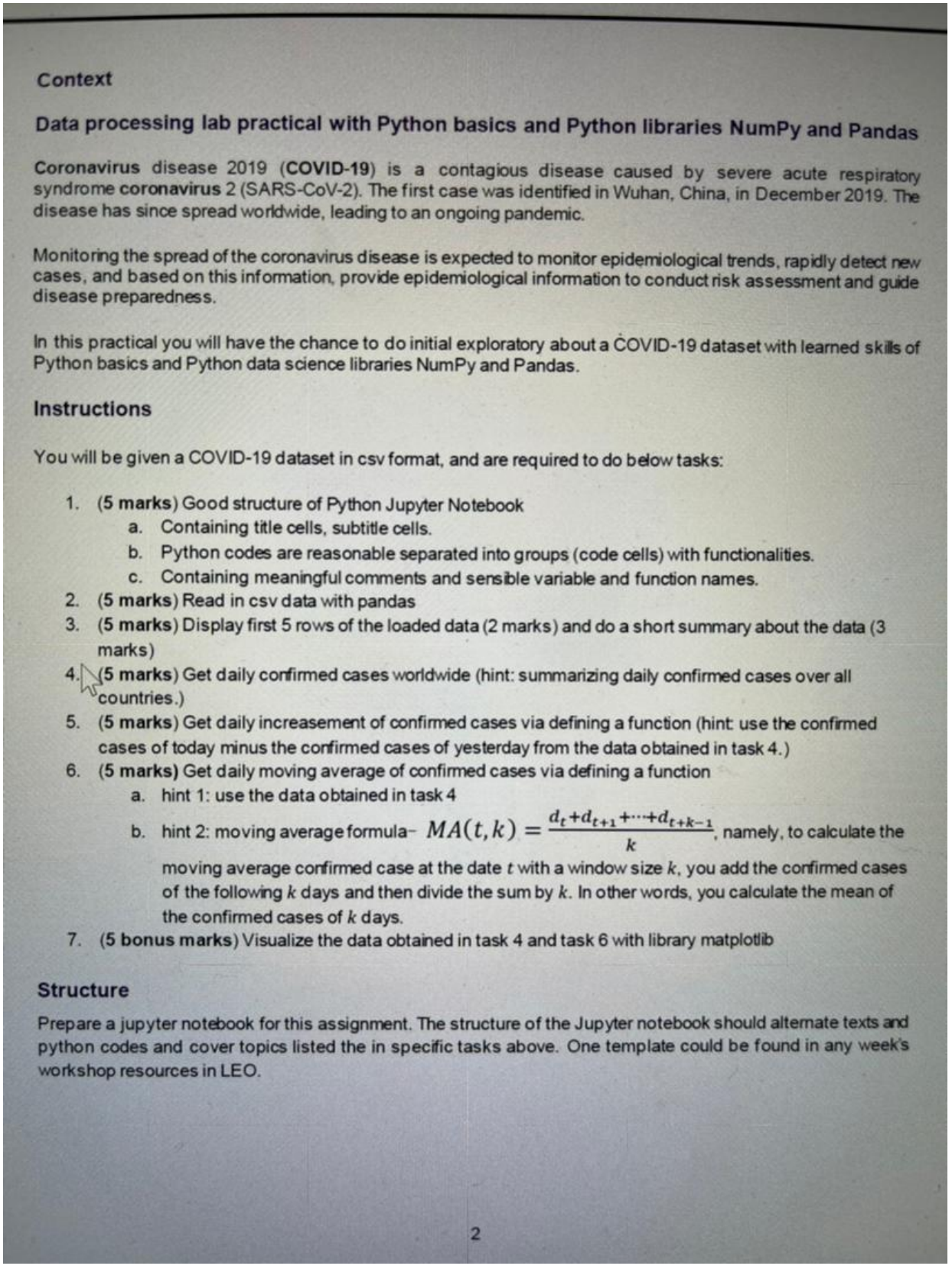
COVID-19 Data Processing with Python

Import Task

In [1]:
```
from IPython.display import Image
Image('Task.jpg')
```

Out[1]:



Context

Data processing lab practical with Python basics and Python libraries NumPy and Pandas

Coronavirus disease 2019 (**COVID-19**) is a contagious disease caused by severe acute respiratory syndrome **coronavirus** 2 (SARS-CoV-2). The first case was identified in Wuhan, China, in December 2019. The disease has since spread worldwide, leading to an ongoing pandemic.

Monitoring the spread of the coronavirus disease is expected to monitor epidemiological trends, rapidly detect new cases, and based on this information, provide epidemiological information to conduct risk assessment and guide disease preparedness.

In this practical you will have the chance to do initial exploratory about a COVID-19 dataset with learned skills of Python basics and Python data science libraries NumPy and Pandas.

Instructions

You will be given a COVID-19 dataset in csv format, and are required to do below tasks:

1. (**5 marks**) Good structure of Python Jupyter Notebook
   a. Containing title cells, subtitle cells.
   b. Python codes are reasonable separated into groups (code cells) with functionalities.
   c. Containing meaningful comments and sensible variable and function names.
2. (**5 marks**) Read in csv data with pandas
3. (**5 marks**) Display first 5 rows of the loaded data (2 marks) and do a short summary about the data (3 marks)
4. (**5 marks**) Get daily confirmed cases worldwide (hint: summarizing daily confirmed cases over all countries.)
5. (**5 marks**) Get daily increasement of confirmed cases via defining a function (hint: use the confirmed cases of today minus the confirmed cases of yesterday from the data obtained in task 4.)
6. (**5 marks**) Get daily moving average of confirmed cases via defining a function
   a. hint 1: use the data obtained in task 4
   b. hint 2: moving average formula– $MA(t,k) = \frac{d_t + d_{t+1} + \cdots + d_{t+k-1}}{k}$, namely, to calculate the moving average confirmed case at the date t with a window size k, you add the confirmed cases of the following k days and then divide the sum by k. In other words, you calculate the mean of the confirmed cases of k days.
7. (**5 bonus marks**) Visualize the data obtained in task 4 and task 6 with library matplotlib

Structure

Prepare a jupyter notebook for this assignment. The structure of the Jupyter notebook should alternate texts and python codes and cover topics listed the in specific tasks above. One template could be found in any week's workshop resources in LEO.

2

Checking the Versions of Libraries

In [2]:
```
import sys
# Version of Python IDLE
print('Python : {}'.format(sys.version))

import scipy
# Version of scipy
print('scipy : {}'.format(scipy.__version__))

import numpy
# Version of numpy
print('numpy : {}'.format(numpy.__version__))

import pandas
# Version of pandas
print('pandas : {}'.format(pandas.__version__))

import matplotlib
# Version of matplotlib
print('matplotlib : {}'.format(matplotlib.__version__))
```

```
Python : 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
scipy : 1.4.1
numpy : 1.18.1
pandas : 1.1.2
matplotlib : 3.1.3
```

Question 2 (Task 2)

Loading Libraries

All the Python libraries that we might require are not loaded to our working environment (Jupyter Notebook/ Colab Notebook) by default, even if they are already installed in our system. That is why we need to import each and every library that we want to use.

In data science, numpy and pandas are most commonly used libraries.

- Numpy is required for calculations like mean, median, square root, etc.
- Pandas is used for data processing and working with CSV files with DataFrames.

We chose alias names for our libraries for the sake of our convenience (numpy --> np and pandas --> pd).

In [3]:
```
%matplotlib inline
import numpy as np
# NumPy is a basic package for scientific computing with Python and especially for data analysis
# Fundamental package for linear algebra and multidimensional arrays

import pandas as pd
# Pandas is an open source Python library for highly specialized data analysis.
# Data analysis and manipulation tool

import matplotlib.pyplot as plt
# Matplotlib is a Python library specializing in the development of two-dimensional charts (incl

# To ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

Loading Dataset

Pandas module is used for reading files. We have our data in '.xlsx' format. We will use 'read_excel()' and '.csv' format, we will use 'read_csv' function for loading the data.

In [4]:
```
# Data given is in an excel format
Data = pd.read_excel('assessment2_data copy.xlsx')
```

Question 3a (Task 3a)

Displaying the first 5 rows of the dataset

Head()

To take a closer look at the data, we take help of " .head()" function of pandas library which returns first five observations of the data set. Similarly ".tail()" returns last five observations of the data set.

You'll see that this is a great way to get an initial feeling of your data and maybe understand it a bit better already!

In [5]:
```
# Displaying the first 5 row of the data given
Data.head()
```

Out[5]:

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	...	3/20/21	3/21
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	0	0	...	56093	561
1	NaN	Albania	41.15330	20.168300	0	0	0	0	0	0	...	120541	1212
2	NaN	Algeria	28.03390	1.659600	0	0	0	0	0	0	...	116066	1161
3	NaN	Andorra	42.50630	1.521800	0	0	0	0	0	0	...	11481	115
4	NaN	Angola	-11.20270	17.873900	0	0	0	0	0	0	...	21696	217

5 rows × 437 columns

Question 3b (Task 3b)

Short Summarry about the Data

Shape

Find out the total number of rows and columns in the dataset using ".shape".

In [6]:
```
Data.shape
```

Out[6]: (274, 437)

Observation:

- Dataset comprises of 274 observations(rows) and 437 features (columns).

Get a Statistical Overview using Describe()

The describe() function in pandas is very handy in getting various summary statistics.This function returns the count, mean, standard deviation, minimum and maximum values and the quartiles of the data.

It is always interesting to know the basic statistical characteristics of each numerical variables.

In [7]:
```
Data.describe()
```

Out[7]:

	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	1/28/20	
count	273.000000	273.000000	274.000000	274.000000	274.000000	274.000000	274.000000	274.000000	274.000000	27
mean	20.534804	23.028143	2.032847	2.390511	3.434307	5.229927	7.729927	10.682482	20.357664	2
std	25.194592	73.596166	26.879101	26.977077	33.585238	46.743494	65.324493	88.014971	215.981285	21
min	-51.796300	-178.116500	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	5.152149	-19.020800	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	21.694000	20.939400	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
75%	41.112900	84.250000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
max	71.706900	178.065000	444.000000	444.000000	549.000000	761.000000	1058.000000	1423.000000	3554.000000	355

8 rows × 435 columns

Let's explore different statistical measures that we have got from describe().

- **count:** total count of non-null values in the column
- **mean**: the average of all the values in that column
- **min:** the minimum value in the column
- **max:** the maximum value in the column
- **25%:** first quartile in the column after we arrange those values in ascending order
- **50%:** this is the median or the second quartile
- **75%:** the third quartile
- **std:** this is the standard deviation (i.e. measure of depreciation, you must have read in the basics of statistics study material)

Note: 25%, 50%, and 75% are nothing but corresponding percentile values

Exploring the Features/ Variables

Variables and features both are the same, they are often used interchangeably. All the column names in a dataset are variables.

Let's explore the features / columns of the datasets.

In [8]:
```
1 Data.columns
```

Out[8]:
```
Index(['Province/State', 'Country/Region',           'Lat',          'Long',
               '1/22/20',        '1/23/20',        '1/24/20',        '1/25/20',
               '1/26/20',        '1/27/20',
       ...
               '3/20/21',        '3/21/21',        '3/22/21',        '3/23/21',
               '3/24/21',        '3/25/21',        '3/26/21',        '3/27/21',
               '3/28/21',        '3/29/21'],
      dtype='object', length=437)
```

We can infer that only two columns/features are Categorical while other features are numerical

info()

df.info return information about the data frame including the data types of each column, number of null values in each column and memory usage of the entire data.

In [9]:
```
1 Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 274 entries, 0 to 273
Columns: 437 entries, Province/State to 3/29/21
dtypes: float64(2), int64(433), object(2)
memory usage: 935.6+ KB
```

Check for Missing Values

Handling missing values is an essential part of the data cleaning and preparation process because almost all data in real life comes with some missing values.

Pandas provides `isnull()`, `isna()` functions to detect missing values. Both of them do the same thing.

- `df.isna()` returns the dataframe with boolean values indicating missing values.
- You can also choose to use `notna()` which is just the opposite of isna().
- `df.isna().any()` returns a boolean value for each column. If there is at least one missing value in that column, the result is True.
- `df.isna().sum()` returns the number of missing values in each column.

isna() or isnull() Function

In [10]:
```
1 Data.isna().sum()
```

Out[10]:
```
Province/State    189
Country/Region      0
Lat                 1
Long                1
1/22/20             0
                 ...
3/25/21             0
3/26/21             0
3/27/21             0
3/28/21             0
3/29/21             0
Length: 437, dtype: int64
```

Question 4 (Task 4)

Get daily confirmed cases worldwide (hint: sumarizing daily confirmed cases over all countries)

In [11]:
```
1 Data.head(10)
```

Out[11]:

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	...	3/20/21	3
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	0	0	...	56093	
1	NaN	Albania	41.15330	20.168300	0	0	0	0	0	0	...	120541	1
2	NaN	Algeria	28.03390	1.659600	0	0	0	0	0	0	...	116066	1
3	NaN	Andorra	42.50630	1.521800	0	0	0	0	0	0	...	11481	
4	NaN	Angola	-11.20270	17.873900	0	0	0	0	0	0	...	21696	
5	NaN	Antigua and Barbuda	17.06080	-61.796400	0	0	0	0	0	0	...	1033	
6	NaN	Argentina	-38.41610	-63.616700	0	0	0	0	0	0	...	2241739	22
7	NaN	Armenia	40.06910	45.038200	0	0	0	0	0	0	...	183127	1
8	Australian Capital Territory	Australia	-35.47350	149.012400	0	0	0	0	0	0	...	123	
9	New South Wales	Australia	-33.86880	151.209300	0	0	0	0	3	4	...	5261	

10 rows × 437 columns

Note that daily confirmed cases worldwide is equal to the sum of each datapoints in respective date columns starting from (1/22/20) to (3/29/21)

In [12]:
```
1 # Slicing the data set to get the date part in order to summarize daily confirmed cases
2 Date_data = Data.iloc[:,4:] # Date_date is simply a variable that contain all Date column and its
```

In [13]:
```
1 Date_data
```

Out[13]:

	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	1/28/20	1/29/20	1/30/20	1/31/20	...	3/20/21	3/21/21	3/22/21	3/23/
0	0	0	0	0	0	0	0	0	0	0	...	56093	56103	56153	561
1	0	0	0	0	0	0	0	0	0	0	...	120541	121200	121544	1218
2	0	0	0	0	0	0	0	0	0	0	...	116066	116157	116255	1163
3	0	0	0	0	0	0	0	0	0	0	...	11481	11517	11545	115
4	0	0	0	0	0	0	0	0	0	0	...	21696	21733	21757	217
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
269	0	2	2	2	2	2	2	2	2	2	...	2572	2572	2575	25
270	0	0	0	0	0	0	0	0	0	0	...	221391	223638	225976	2280
271	0	0	0	0	0	0	0	0	0	0	...	3278	3418	3516	36
272	0	0	0	0	0	0	0	0	0	0	...	86273	86449	86535	867
273	0	0	0	0	0	0	0	0	0	0	...	36662	36665	36684	367

274 rows × 433 columns

sum of each column gives the daily confirmed cases worldwide.

In [14]:
```
1 daily_confirmed_cases_worldwide = Date_data.sum(axis = 0) # axis = 0 means we are summing along
```

In [15]:
```
1 daily_confirmed_cases_worldwide
```

Out[15]:
```
1/22/20          557
1/23/20          655
1/24/20          941
1/25/20         1433
1/26/20         2118
               ...
3/25/21    125491735
3/26/21    126130915
3/27/21    126714740
3/28/21    127185164
3/29/21    127644090
Length: 433, dtype: int64
```

Transfering/Formating out daily confirmed cases into a csv file.

In [16]:
```
1 # Slicing out the date of each day in which the virus exist
2 daily_confirmed_cases_worldwide.index
```

Out[16]:
```
Index(['1/22/20', '1/23/20', '1/24/20', '1/25/20', '1/26/20', '1/27/20',
       '1/28/20', '1/29/20', '1/30/20', '1/31/20',
       ...
       '3/20/21', '3/21/21', '3/22/21', '3/23/21', '3/24/21', '3/25/21',
       '3/26/21', '3/27/21', '3/28/21', '3/29/21'],
      dtype='object', length=433)
```

In [17]:
```
# Slicing out the total number of daily case worldwide
daily_confirmed_cases_worldwide.iloc[::]
```

Out[17]:
```
1/22/20          557
1/23/20          655
1/24/20          941
1/25/20         1433
1/26/20         2118
               ...
3/25/21    125491735
3/26/21    126130915
3/27/21    126714740
3/28/21    127185164
3/29/21    127644090
Length: 433, dtype: int64
```

In [18]:
```
### Transfering results from numpy array into a Data Frame
```

In [19]:
```
result = {'Dates': daily_confirmed_cases_worldwide.index, 'daily_confirmed_cases_worldwide': dai
results = pd.DataFrame(result) # Converting Dictionary Into a data frame.
results.to_csv('daily_confirmed_cases_worldwide.csv', index = False)
```

Data obtained in task 4 is named 'daily_confirmed_cases_worldwide.csv'

Solution to daily confirmed cases worldwide of respective date columns starting from (1/22/20) to (3/29/21) in the csv file named 'daily_confirmed_cases_worldwide.csv'

In [20]:
```
daily_confirmed_cases_worldwide = pd.read_csv('daily_confirmed_cases_worldwide.csv')
```

In [21]:
```
daily_confirmed_cases_worldwide.head()
```

Out[21]:

	Dates	daily_confirmed_cases_worldwide
0	1/22/20	557
1	1/23/20	655
2	1/24/20	941
3	1/25/20	1433
4	1/26/20	2118

In [22]:
```
daily_confirmed_cases_worldwide.shape
```

Out[22]:
```
(433, 2)
```

Question 5 (Task 5)

Get daily increment of confirmed cases via a function

In [23]:
```
# It is noteworthy that by just extracting the loop, this can be done also. But using a function
def Increment_func(index):

    # Create a list where all increment will be appended into for conversion into a dataframe
    Increment =[0,]

    # For loop statement that will iterate through the daily cases based on the shape of the data
    for i in range(0,432):

        #Hint: From data obtained in task 4- using confirmed cases yesterday minus confirmed case
        increment = daily_confirmed_cases_worldwide['daily_confirmed_cases_worldwide'][i + 1] - (
        print(increment)

        #Appending the increment into the list (Increment) created above.
        Increment.append(increment)
```

In [24]:
```
Increment_func(0)
```

```
98
286
492
685
809
2651
589
2068
1692
2111
4749
3100
4011
3745
3160
3593
2734
3030
2609
```

Alternatively, this can be done by using a for_loop to make it easy to append into a dataframe

In [25]:
```
Increment =[0,]

# For loop statement that will iterate through the daily cases based on the shape of the data abo
for i in range(0,432):

    #Hint: From data obtained in task 4- using confirmed cases yesterday minus confirmed cases to
    increment = daily_confirmed_cases_worldwide['daily_confirmed_cases_worldwide'][i + 1] - daily
    print(increment)

    #Appending the increment into the list (Increment) created above.
    Increment.append(increment)
```

```
98
286
492
685
809
2651
589
2068
1692
2111
4749
3100
4011
3745
3160
3593
2734
3030
2609
```

In [26]:
```
# Naming the increment column and displaying it.
daily_confirmed_cases_worldwide['Increment_daily_confirmed_cases'] = Increment

print(daily_confirmed_cases_worldwide)
```

```
       Dates  daily_confirmed_cases_worldwide  Increment_daily_confirmed_cases
0    1/22/20                              557                                0
1    1/23/20                              655                               98
2    1/24/20                              941                              286
3    1/25/20                             1433                              492
4    1/26/20                             2118                              685
..       ...                              ...                              ...
428  3/25/21                        125491735                           650380
429  3/26/21                        126130915                           639180
430  3/27/21                        126714740                           583825
431  3/28/21                        127185164                           470424
432  3/29/21                        127644090                           458926

[433 rows x 3 columns]
```

Creating a file for task 5 that shows the increment of daily confirmed cased

In [27]:
```
#Displaying the Dataframe that shows the respective days (dates) and increasement
result = {'Dates': daily_confirmed_cases_worldwide['Dates'], 'Increment_daily_confirmed_cases' :
results = pd.DataFrame(result)
```

In [28]:
```
1 results
```

Out[28]:

	Dates	Increment_daily_confirmed_cases
0	1/22/20	0
1	1/23/20	98
2	1/24/20	286
3	1/25/20	492
4	1/26/20	685
...	...	...
428	3/25/21	650380
429	3/26/21	639180
430	3/27/21	583825
431	3/28/21	470424
432	3/29/21	458926

433 rows × 2 columns

In [29]:
```
1 daily_confirmed_cases_worldwide.to_csv('Increment_daily_confirmed_cases.csv', index = False)
```

In [30]:
```
1 Increment = pd.read_csv('Increment_daily_confirmed_cases.csv')
2 Increment
```

Out[30]:

	Dates	daily_confirmed_cases_worldwide	Increment_daily_confirmed_cases
0	1/22/20	557	0
1	1/23/20	655	98
2	1/24/20	941	286
3	1/25/20	1433	492
4	1/26/20	2118	685
...	...	...	...
428	3/25/21	125491735	650380
429	3/26/21	126130915	639180
430	3/27/21	126714740	583825
431	3/28/21	127185164	470424
432	3/29/21	127644090	458926

433 rows × 3 columns

Data obtained in task 5 is named 'Increment_daily_confirmed_cases.csv'

Question 6 (Task 6)

Average Daily moving average of confirmed cases

Using Hint 1: From the data gotten from task 4

In [31]:
```
1 # Displaying data gotten from task 4
2
3 task_4_data = pd.read_csv('daily_confirmed_cases_worldwide.csv')
```

In [32]:
```
task_4_data.head()
```

Out[32]:

	Dates	daily_confirmed_cases_worldwide
0	1/22/20	557
1	1/23/20	655
2	1/24/20	941
3	1/25/20	1433
4	1/26/20	2118

Simple Moving Average (SMA): Simple Moving Average (SMA) uses a sliding window to take the average over a set number of time periods. It is an equally weighted mean of the previous n data.

For the question, we will be using Simple Moving Average.

Let's calculate SMA for a window size of 3, which means you will consider three values each time (3 days) to calculate the moving average, and for every new value, the oldest value will be ignored.

To implement this, you will use pandas .iloc function, since the demand column is what you need, you will fix the position of that in the .iloc function while the row will be a variable i which you will keep iterating until you reach the end of the dataframe.

In [33]:
```
# Simple Moving Average(SMA) for a window size of 3, which means you will consider three values e
# And for every new value, the oldest value will be ignored.
# Our window size is denoted with k
for i in range(0,task_4_data.shape[0]-2):
    task_4_data.loc[task_4_data.index[i + 2],'SMA_(k = 3)'] = np.round(((task_4_data.iloc[i,1]+ t
```

In [34]:
```
task_4_data.head()
```

Out[34]:

	Dates	daily_confirmed_cases_worldwide	SMA_(k = 3)
0	1/22/20	557	NaN
1	1/23/20	655	NaN
2	1/24/20	941	717.7
3	1/25/20	1433	1009.7
4	1/26/20	2118	1497.3

Let's calculate SMA for a window size of 4, which means you will consider four values each time (4 days) to calculate the moving average, and for every new value, the oldest value will be ignored.

In [35]:
```
for i in range(0,task_4_data.shape[0]-3):
    task_4_data.loc[task_4_data.index[i+3],'SMA_(k = 4)'] = np.round(((task_4_data.iloc[i,1]+ ta
```

In [36]:
```
task_4_data.head(20)
```

Out[36]:

	Dates	daily_confirmed_cases_worldwide	SMA_(k = 3)	SMA_(k = 4)
0	1/22/20	557	NaN	NaN
1	1/23/20	655	NaN	NaN
2	1/24/20	941	717.7	NaN
3	1/25/20	1433	1009.7	896.5
4	1/26/20	2118	1497.3	1286.8
5	1/27/20	2927	2159.3	1854.8
6	1/28/20	5578	3541.0	3014.0
7	1/29/20	6167	4890.7	4197.5
8	1/30/20	8235	6660.0	5726.8
9	1/31/20	9927	8109.7	7476.8
10	2020-01-02 00:00:00	12038	10066.7	9091.8
11	2020-02-02 00:00:00	16787	12917.3	11746.8
12	2020-03-02 00:00:00	19887	16237.3	14659.8
13	2020-04-02 00:00:00	23898	20190.7	18152.5
14	2020-05-02 00:00:00	27643	23809.3	22053.8
15	2020-06-02 00:00:00	30803	27448.0	25557.8
16	2020-07-02 00:00:00	34396	30947.3	29185.0
17	2020-08-02 00:00:00	37130	34109.7	32493.0
18	2020-09-02 00:00:00	40160	37228.7	35622.2
19	2020-10-02 00:00:00	42769	40019.7	38613.8

For a sanity check, let's also use the pandas in-built rolling function and see if it matches with our custom python based simple moving average.

In [37]:
```
task_4_data['pandas_SMA_3'] = task_4_data.iloc[:,1].rolling(window=3).mean()
task_4_data.head()
```

Out[37]:

	Dates	daily_confirmed_cases_worldwide	SMA_(k = 3)	SMA_(k = 4)	pandas_SMA_3
0	1/22/20	557	NaN	NaN	NaN
1	1/23/20	655	NaN	NaN	NaN
2	1/24/20	941	717.7	NaN	717.666667
3	1/25/20	1433	1009.7	896.5	1009.666667
4	1/26/20	2118	1497.3	1286.8	1497.333333

In [38]:
```
task_4_data['pandas_SMA_4'] = task_4_data.iloc[:,1].rolling(window=4).mean()
task_4_data.head()
```

Out[38]:

	Dates	daily_confirmed_cases_worldwide	SMA_(k = 3)	SMA_(k = 4)	pandas_SMA_3	pandas_SMA_4
0	1/22/20	557	NaN	NaN	NaN	NaN
1	1/23/20	655	NaN	NaN	NaN	NaN
2	1/24/20	941	717.7	NaN	717.666667	NaN
3	1/25/20	1433	1009.7	896.5	1009.666667	896.50
4	1/26/20	2118	1497.3	1286.8	1497.333333	1286.75

In [39]:
```
task_4_data.head(20)
```

Out[39]:

	Dates	daily_confirmed_cases_worldwide	SMA_(k = 3)	SMA_(k = 4)	pandas_SMA_3	pandas_SMA_4
0	1/22/20	557	NaN	NaN	NaN	NaN
1	1/23/20	655	NaN	NaN	NaN	NaN
2	1/24/20	941	717.7	NaN	717.666667	NaN
3	1/25/20	1433	1009.7	896.5	1009.666667	896.50
4	1/26/20	2118	1497.3	1286.8	1497.333333	1286.75
5	1/27/20	2927	2159.3	1854.8	2159.333333	1854.75
6	1/28/20	5578	3541.0	3014.0	3541.000000	3014.00
7	1/29/20	6167	4890.7	4197.5	4890.666667	4197.50
8	1/30/20	8235	6660.0	5726.8	6660.000000	5726.75
9	1/31/20	9927	8109.7	7476.8	8109.666667	7476.75
10	2020-01-02 00:00:00	12038	10066.7	9091.8	10066.666667	9091.75
11	2020-02-02 00:00:00	16787	12917.3	11746.8	12917.333333	11746.75
12	2020-03-02 00:00:00	19887	16237.3	14659.8	16237.333333	14659.75
13	2020-04-02 00:00:00	23898	20190.7	18152.5	20190.666667	18152.50
14	2020-05-02 00:00:00	27643	23809.3	22053.8	23809.333333	22053.75
15	2020-06-02 00:00:00	30803	27448.0	25557.8	27448.000000	25557.75
16	2020-07-02 00:00:00	34396	30947.3	29185.0	30947.333333	29185.00
17	2020-08-02 00:00:00	37130	34109.7	32493.0	34109.666667	32493.00
18	2020-09-02 00:00:00	40160	37228.7	35622.2	37228.666667	35622.25
19	2020-10-02 00:00:00	42769	40019.7	38613.8	40019.666667	38613.75

In [40]:
```
task_4_data.to_csv('task_4_data_new.csv', index = False)
```

In [41]:
```
new_data = pd.read_csv('task_4_data_new.csv')
```

In [42]:
```
1 new_data.head(20)
```

Out[42]:

	Dates	daily_confirmed_cases_worldwide	SMA_(k = 3)	SMA_(k = 4)	pandas_SMA_3	pandas_SMA_4
0	1/22/20	557	NaN	NaN	NaN	NaN
1	1/23/20	655	NaN	NaN	NaN	NaN
2	1/24/20	941	717.7	NaN	717.666667	NaN
3	1/25/20	1433	1009.7	896.5	1009.666667	896.50
4	1/26/20	2118	1497.3	1286.8	1497.333333	1286.75
5	1/27/20	2927	2159.3	1854.8	2159.333333	1854.75
6	1/28/20	5578	3541.0	3014.0	3541.000000	3014.00
7	1/29/20	6167	4890.7	4197.5	4890.666667	4197.50
8	1/30/20	8235	6660.0	5726.8	6660.000000	5726.75
9	1/31/20	9927	8109.7	7476.8	8109.666667	7476.75
10	2020-01-02 00:00:00	12038	10066.7	9091.8	10066.666667	9091.75
11	2020-02-02 00:00:00	16787	12917.3	11746.8	12917.333333	11746.75
12	2020-03-02 00:00:00	19887	16237.3	14659.8	16237.333333	14659.75
13	2020-04-02 00:00:00	23898	20190.7	18152.5	20190.666667	18152.50
14	2020-05-02 00:00:00	27643	23809.3	22053.8	23809.333333	22053.75
15	2020-06-02 00:00:00	30803	27448.0	25557.8	27448.000000	25557.75
16	2020-07-02 00:00:00	34396	30947.3	29185.0	30947.333333	29185.00
17	2020-08-02 00:00:00	37130	34109.7	32493.0	34109.666667	32493.00
18	2020-09-02 00:00:00	40160	37228.7	35622.2	37228.666667	35622.25
19	2020-10-02 00:00:00	42769	40019.7	38613.8	40019.666667	38613.75

Question 7 (Task 7)

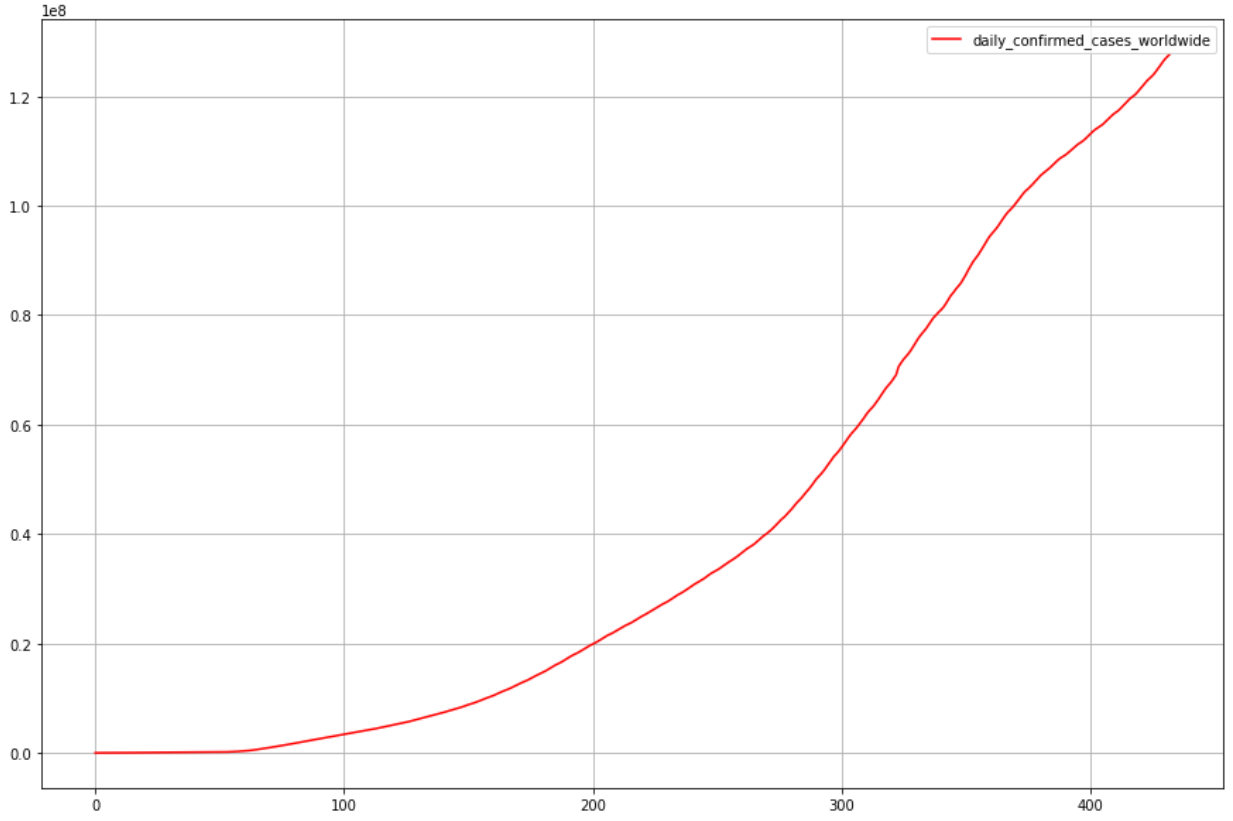
Visualizing results from Task 6 and Task 4

Now, I will plot the data of the moving averages that I calculated using the Matplotlib Library

In [43]:
```
%matplotlib inline
import matplotlib.pyplot as plt

plt.figure(figsize=[15,10])
plt.grid(True)

plt.plot(new_data['daily_confirmed_cases_worldwide'],label='daily_confirmed_cases_worldwide', co
plt.legend()
plt.show()
```

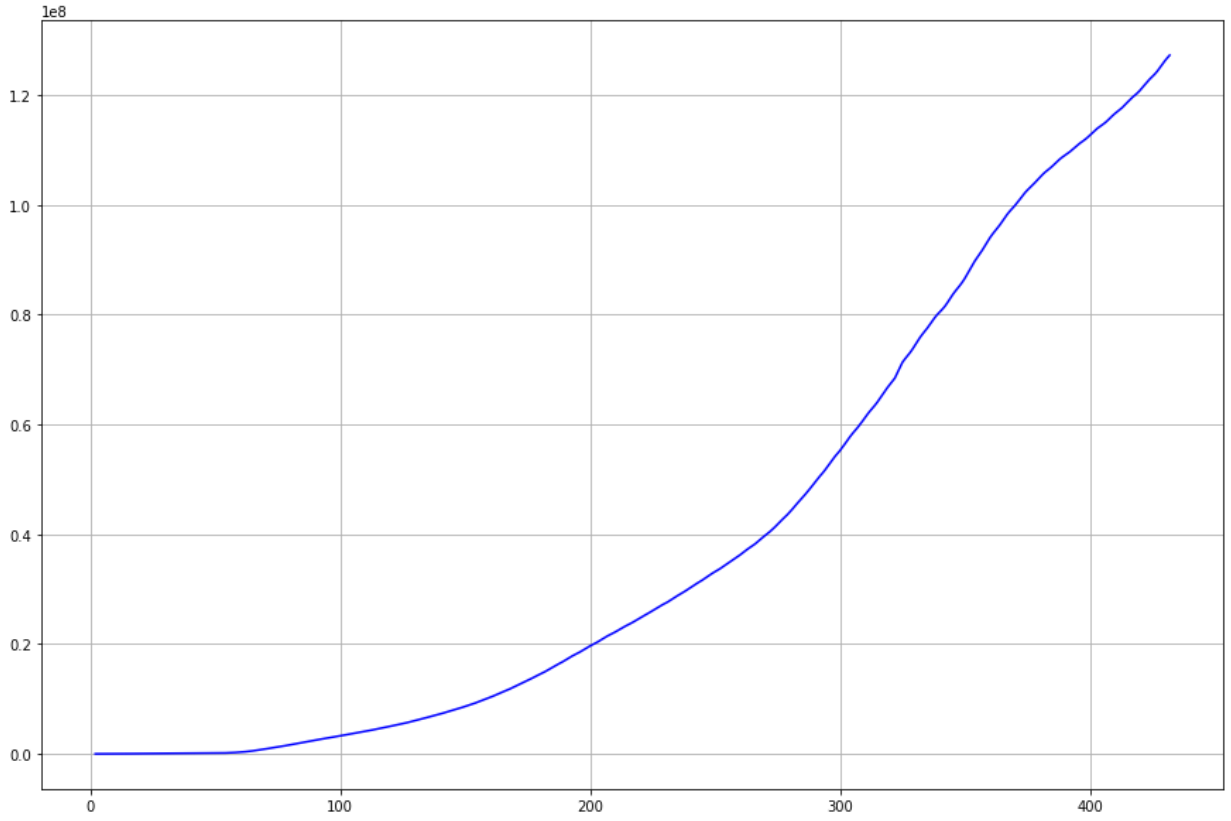


The graph shows daily increase in the number of cases

In [44]:
```
# Where SMA_(k = 3) is Simple moving average with windom size = 3 (days)

plt.figure(figsize=[15,10])
plt.grid(True)
plt.plot(new_data['SMA_(k = 3)'],label='SMA 3 days', color = 'blue')
```

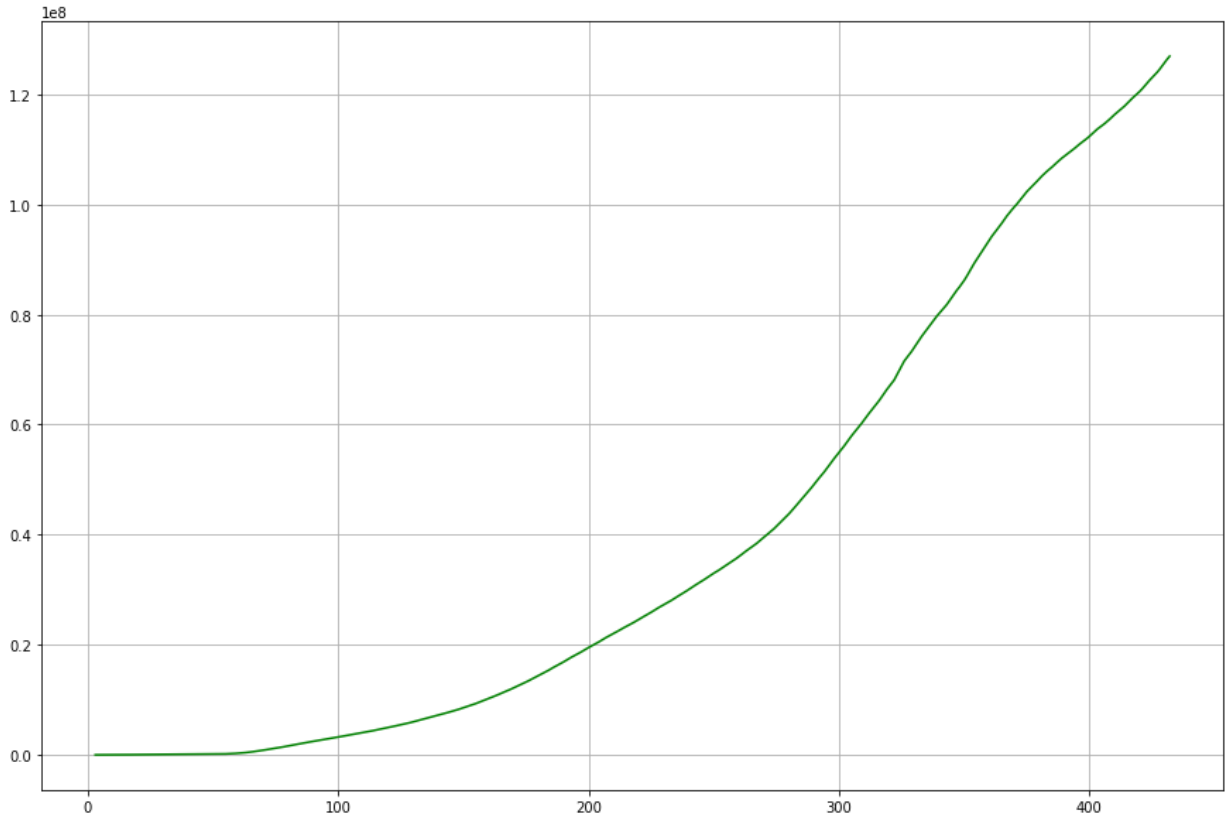
Out[44]: [<matplotlib.lines.Line2D at 0x2283db48e88>]



In [45]:
```
# Where SMA_(k = 4) is Simple moving average with windom size = 4 (days)

plt.figure(figsize=[15,10])
plt.grid(True)
plt.plot(new_data['SMA_(k = 4)'],label='SMA 4 days', color = 'green')
```

Out[45]: [<matplotlib.lines.Line2D at 0x2283d7e1148>]



It is noteworthy that window size (k) can be changed based on the moving average that you want to visualize.

In [ ]:
```
1
```

In [ ]:
```
1
```