Introduction To Data Science

Exercise 02 - Statistic and Data Visualization

Teacher: Le Ngoc Thanh - Tran Trung Kien - Nguyen Tien Huy
Student: Phan Vi Giai - 19127639

Task Table

STT	Task	Separated task	% Completed
1	Crawling data		100%
2	Preprocessing data		100%
3	Visualizing data	Statistic and Conclusion	100%
		Visualize data with charts	100%

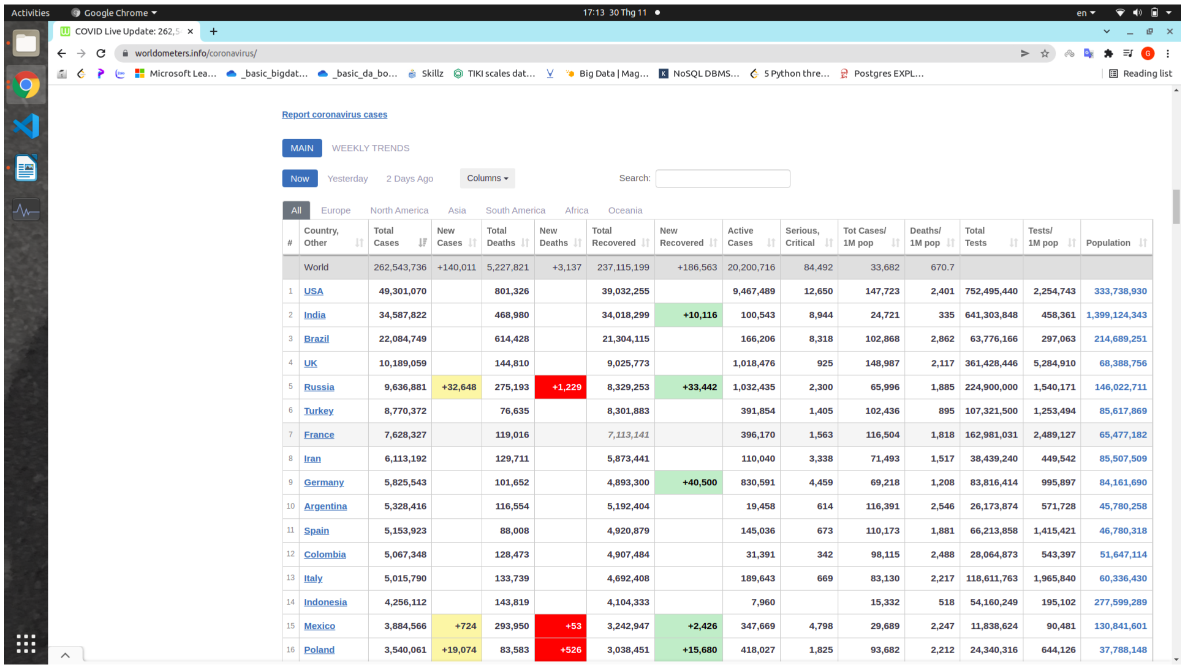
Table of Contents

Introduction To Data Science	Date: 30/11/2021
Exercise 02 - Data visualization	Author: Phan Vi Giai - 19127639

Exercise 02 - Statistic and Data Visualization

1. Introduction

From about the end of 2019 until now, a plague has spread terribly around the world. Every day thousands of people are infected and tens to hundreds of people die. Organization Worldometer (www.worldometers.info) has collected statistical data from many sources and from many countries. Join daily reports to aggregate into a table in Figure .
In the website, the organization Worldmeter also makes graphs to show the visual change of the epidemic situation. However, we do not use it for the time being. We are mandated by the country to understand this data as between data fields that are related, what is the relationship, is there an anomaly in the data as the country report differs from the data synthesis, anomalies in data jumping, ... Apply knowledge of statistics and data visualization to make sense of existing data.



Report coronavirus cases

#	Country, Other	Total Cases	New Cases	Total Deaths	New Deaths	Total Recovered	New Recovered	Active Cases	Serious, Critical	Tot Cases/ 1M pop	Deaths/ 1M pop	Total Tests	Tests/ 1M pop	Population
	World	262,543,736	+140,011	5,227,821	+3,137	237,115,199	+186,563	20,200,716	84,492	33,682	670.7			
1	USA	49,301,070		801,326		39,032,255		9,467,489	12,650	147,723	2,401	752,495,440	2,254,743	333,738,930
2	India	34,587,822		468,980		34,018,299	+10,116	100,543	8,944	24,721	335	641,303,848	458,361	1,399,124,343
3	Brazil	22,084,749		614,428		21,304,115		166,206	8,318	102,868	2,862	63,776,166	297,063	214,689,251
4	UK	10,189,059		144,810		9,025,773		1,018,476	925	148,987	2,117	361,428,446	5,284,910	68,388,756
5	Russia	9,636,881	+32,648	275,193	+1,229	8,329,253	+33,442	1,032,435	2,300	65,996	1,885	224,900,000	1,540,171	146,022,711
6	Turkey	8,770,372		76,635		8,301,883		391,854	1,405	102,436	895	107,321,500	1,253,494	85,617,869
7	France	7,628,327		119,016		*7,113,141*		396,170	1,563	116,504	1,818	162,981,031	2,489,127	65,477,182
8	Iran	6,113,192		129,711		5,873,441		110,040	3,338	71,493	1,517	38,439,240	449,542	85,507,509
9	Germany	5,825,543		101,652		4,893,300	+40,500	830,591	4,459	69,218	1,208	83,816,414	995,897	84,161,690
10	Argentina	5,328,416		116,554		5,192,404		19,458	614	116,391	2,546	26,173,874	571,728	45,780,258
11	Spain	5,153,923		88,008		4,920,879		145,036	673	110,173	1,881	66,213,858	1,415,421	46,780,318
12	Colombia	5,067,348		128,473		4,907,484		31,391	342	98,115	2,488	28,064,873	543,397	51,647,114
13	Italy	5,015,790		133,739		4,692,408		189,643	669	83,130	2,217	118,611,763	1,965,840	60,336,430
14	Indonesia	4,256,112		143,819		4,104,333		7,960		15,332	518	54,160,249	195,102	277,599,289
15	Mexico	3,884,566	+724	293,950	+53	3,242,947	+2,426	347,669	4,798	29,689	2,247	11,838,624	90,481	130,841,601
16	Poland	3,540,061	+19,074	83,583	+526	3,038,451	+15,680	418,027	1,825	93,682	2,212	24,340,316	644,126	37,788,148

2. Data preparation

2.1 Crawling Data

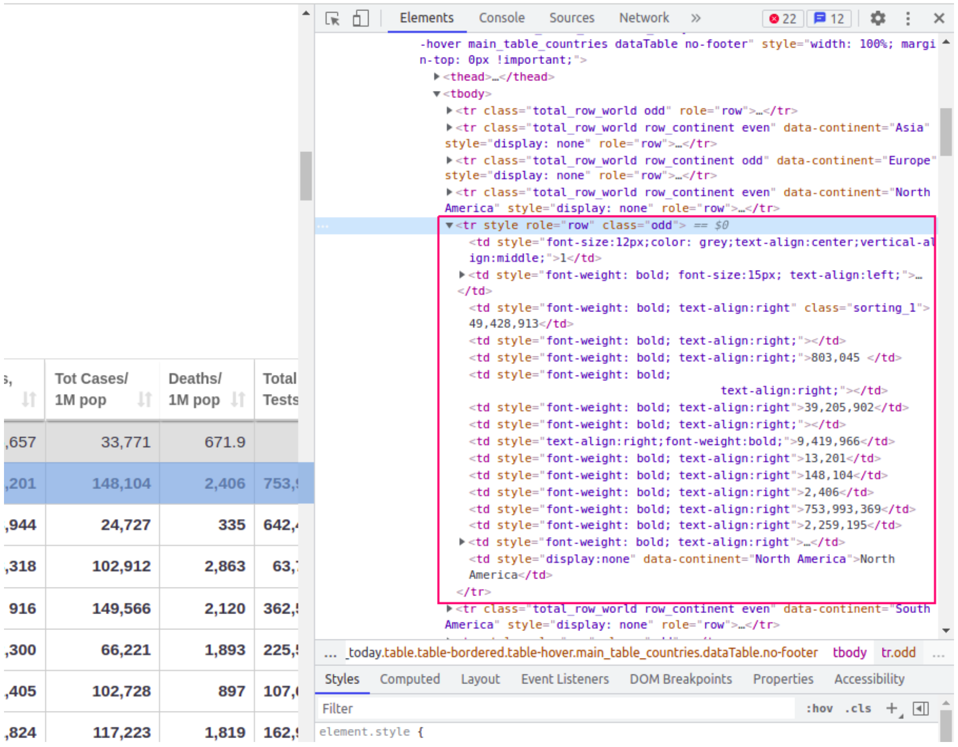
a. Approach

First, we need to get data for our statistics and visualization. But how do we get it ?
First of all, we access the [***www.worldometers.info***](www.worldometers.info) and view the base HTML.
The strategy is we use Python and crawl data from this website.
Because the data we need is included in the general table tag in HTML structure.
It is so easy to get each data in the table cells.

We must crawl data at midnight for having almost full data of that day.

b. Algorithm

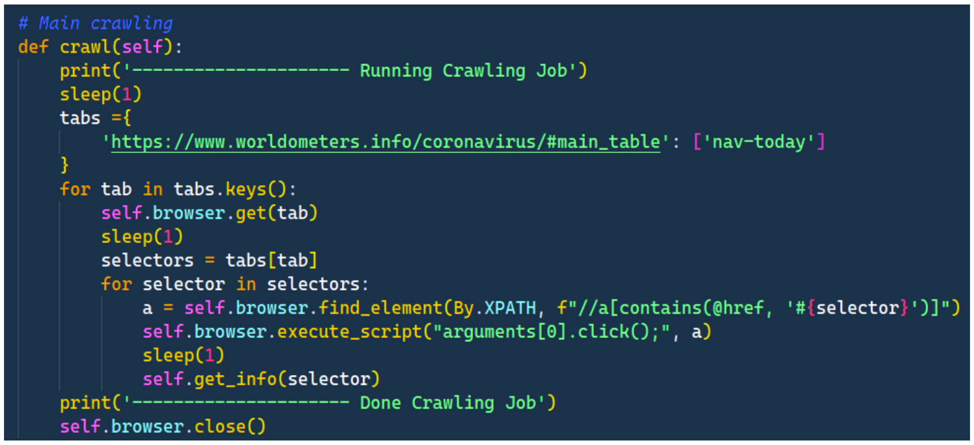
The idea is looping through every <tr> tag, with each <tr> tag we crawl each <td> tag to the end of table cells.



s,	Tot Cases/ 1M pop	Deaths/ 1M pop	Total Tests
,657	33,771	671.9	
,201	148,104	2,406	753,!
,944	24,727	335	642,‹
,318	102,912	2,863	63,'
916	149,566	2,120	362,!
,300	66,221	1,893	225,!
,405	102,728	897	107,(
,824	117,223	1,819	162,!

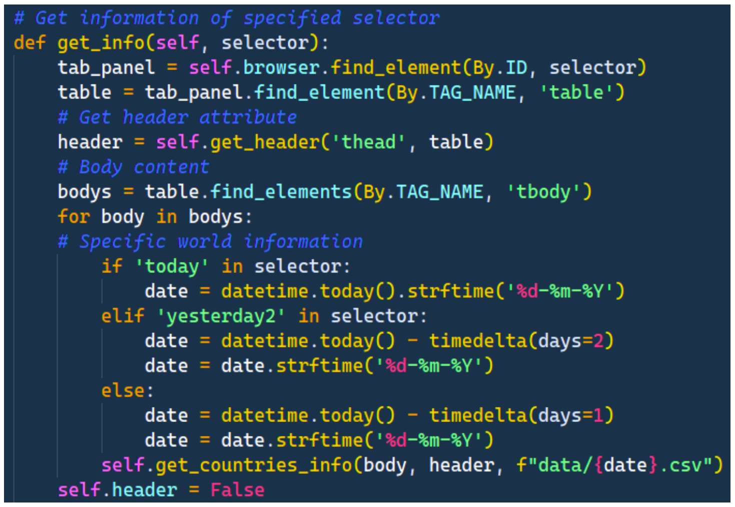
```
-hover main_table_countries dataTable no-footer" style="width: 100%; margi
n-top: 0px !important;">
  ▶<thead>…</thead>
  ▼<tbody>
    ▶<tr class="total_row_world odd" role="row">…</tr>
    ▶<tr class="total_row_world row_continent even" data-continent="Asia"
    style="display: none" role="row">…</tr>
    ▶<tr class="total_row_world row_continent odd" data-continent="Europe"
    style="display: none" role="row">…</tr>
    ▶<tr class="total_row_world row_continent even" data-continent="North
    America" style="display: none" role="row">…</tr>
    ▼<tr style role="row" class="odd"> == $0
       <td style="font-size:12px;color: grey;text-align:center;vertical-al
       ign:middle;">1</td>
      ▶<td style="font-weight: bold; font-size:15px; text-align:left;">…
      </td>
       <td style="font-weight: bold; text-align:right" class="sorting_1">
       49,428,913</td>
       <td style="font-weight: bold; text-align:right;"></td>
       <td style="font-weight: bold; text-align:right;">803,045 </td>
       <td style="font-weight: bold;
                                     text-align:right;"></td>
       <td style="font-weight: bold; text-align:right">39,205,902</td>
       <td style="font-weight: bold; text-align:right;"></td>
       <td style="text-align:right;font-weight:bold;">9,419,966</td>
       <td style="font-weight: bold; text-align:right">13,201</td>
       <td style="font-weight: bold; text-align:right">148,104</td>
       <td style="font-weight: bold; text-align:right">2,406</td>
       <td style="font-weight: bold; text-align:right">753,993,369</td>
       <td style="font-weight: bold; text-align:right">2,259,195</td>
      ▶<td style="font-weight: bold; text-align:right">…</td>
       <td style="display:none" data-continent="North America">North
       America</td>
      </tr>
    ▶<tr class="total_row_world row_continent even" data-continent="South
    America" style="display: none" role="row">…</tr>
```

… _today.table.table-bordered.table-hover.main_table_countries.dataTable.no-footer tbody tr.odd …



```python
# Main crawling
def crawl(self):
    print('--------------------- Running Crawling Job')
    sleep(1)
    tabs ={
        'https://www.worldometers.info/coronavirus/#main_table': ['nav-today']
    }
    for tab in tabs.keys():
        self.browser.get(tab)
        sleep(1)
        selectors = tabs[tab]
        for selector in selectors:
            a = self.browser.find_element(By.XPATH, f"//a[contains(@href, '#{selector}')]")
            self.browser.execute_script("arguments[0].click();", a)
            sleep(1)
            self.get_info(selector)
    print('--------------------- Done Crawling Job')
    self.browser.close()
```

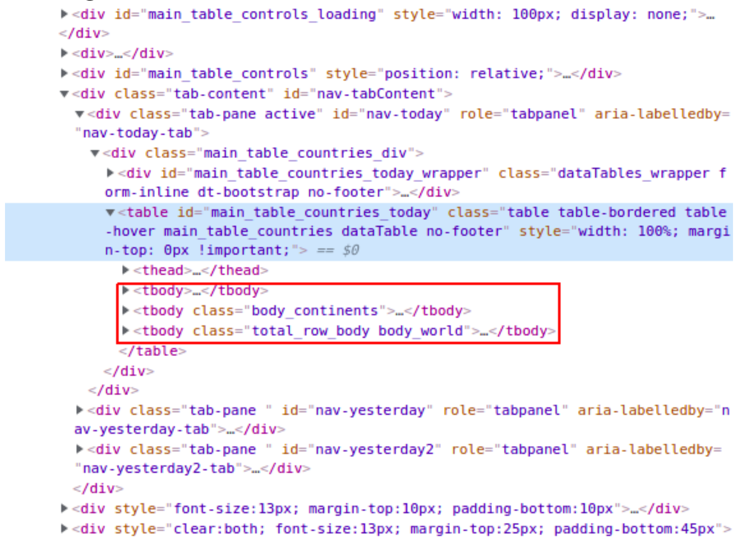
Our main function has a tab object which contains a key for link website and value is a table selector. The function which is crawling data is **get_info()**



```python
# Get information of specified selector
def get_info(self, selector):
    tab_panel = self.browser.find_element(By.ID, selector)
    table = tab_panel.find_element(By.TAG_NAME, 'table')
    # Get header attribute
    header = self.get_header('thead', table)
    # Body content
    bodys = table.find_elements(By.TAG_NAME, 'tbody')
    for body in bodys:
    # Specific world information
        if 'today' in selector:
            date = datetime.today().strftime('%d-%m-%Y')
        elif 'yesterday2' in selector:
            date = datetime.today() - timedelta(days=2)
            date = date.strftime('%d-%m-%Y')
        else:
            date = datetime.today() - timedelta(days=1)
            date = date.strftime('%d-%m-%Y')
        self.get_countries_info(body, header, f"data/{date}.csv")
    self.header = False
```

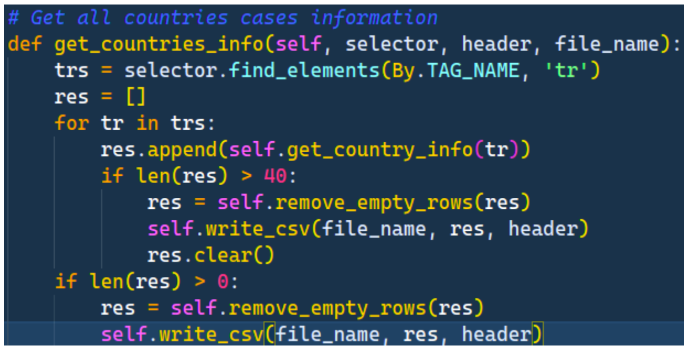
The idea of this function is:

- First, find the tab panel of argument selector which is passed
- Find the main table which contains our table cells
- Then, get the header from table element above



```html
<div id="main_table_controls_loading" style="width: 100px; display: none;">…
</div>
<div>…</div>
<div id="main_table_controls" style="position: relative;">…</div>
<div class="tab-content" id="nav-tabContent">
  <div class="tab-pane active" id="nav-today" role="tabpanel" aria-labelledby=
  "nav-today-tab">
    <div class="main_table_countries_div">
      <div id="main_table_countries_today_wrapper" class="dataTables_wrapper f
      orm-inline dt-bootstrap no-footer">…</div>
      <table id="main_table_countries_today" class="table table-bordered table
      -hover main_table_countries dataTable no-footer" style="width: 100%; margi
      n-top: 0px !important;"> == $0
        <thead>…</thead>
        <tbody>…</tbody>
        <tbody class="body_continents">…</tbody>
        <tbody class="total_row_body body_world">…</tbody>
      </table>
    </div>
  </div>
  <div class="tab-pane " id="nav-yesterday" role="tabpanel" aria-labelledby="n
  av-yesterday-tab">…</div>
  <div class="tab-pane " id="nav-yesterday2" role="tabpanel" aria-labelledby=
  "nav-yesterday2-tab">…</div>
</div>
<div style="font-size:13px; margin-top:10px; padding-bottom:10px">…</div>
<div style="clear:both; font-size:13px; margin-top:25px; padding-bottom:45px">
```

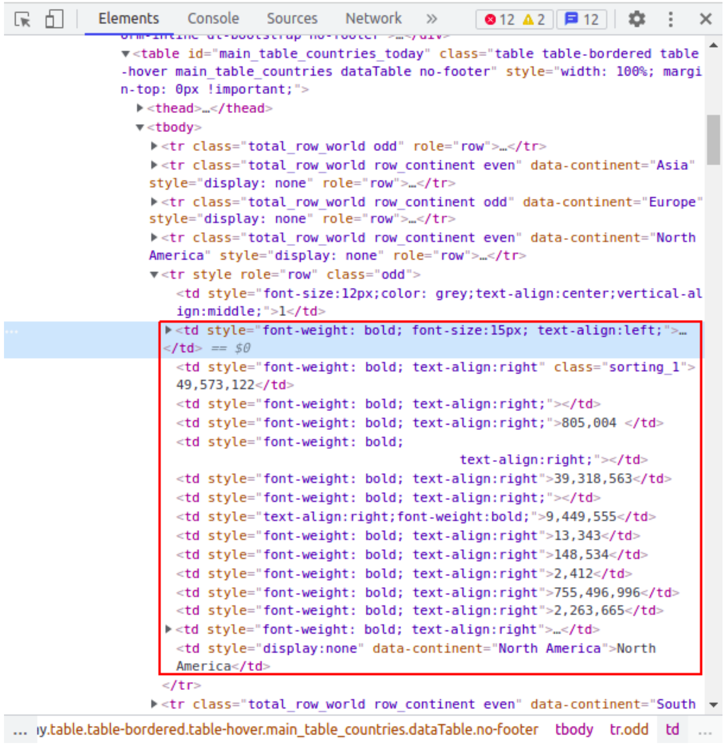
- Because the table contain some <tbody> tags, so that we must find all <tbody> tags and loop through for collecting each table cells
- Then calling the **get_countries_info()** for collection each data cell in the table



```python
# Get all countries cases information
def get_countries_info(self, selector, header, file_name):
    trs = selector.find_elements(By.TAG_NAME, 'tr')
    res = []
    for tr in trs:
        res.append(self.get_country_info(tr))
        if len(res) > 40:
            res = self.remove_empty_rows(res)
            self.write_csv(file_name, res, header)
            res.clear()
    if len(res) > 0:
        res = self.remove_empty_rows(res)
        self.write_csv(file_name, res, header)
```

The idea of this function is:

- First, find all <tr> tags and store in trs variable

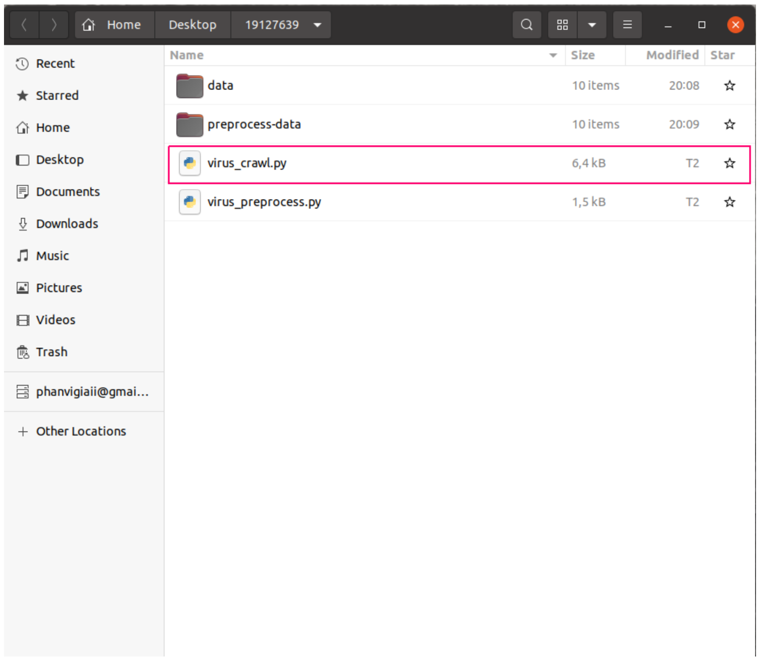


```html
▼<table id="main_table_countries_today" class="table table-bordered table
-hover main_table_countries dataTable no-footer" style="width: 100%; margi
n-top: 0px !important;">
  ▶<thead>…</thead>
  ▼<tbody>
    ▶<tr class="total_row_world odd" role="row">…</tr>
    ▶<tr class="total_row_world row_continent even" data-continent="Asia"
    style="display: none" role="row">…</tr>
    ▶<tr class="total_row_world row_continent odd" data-continent="Europe"
    style="display: none" role="row">…</tr>
    ▶<tr class="total_row_world row_continent even" data-continent="North
    America" style="display: none" role="row">…</tr>
    ▼<tr style role="row" class="odd">
        <td style="font-size:12px;color: grey;text-align:center;vertical-al
        ign:middle;">1</td>
      ▶<td style="font-weight: bold; font-size:15px; text-align:left;">…
      </td> == $0
        <td style="font-weight: bold; text-align:right" class="sorting_1">
        49,573,122</td>
        <td style="font-weight: bold; text-align:right;"></td>
        <td style="font-weight: bold; text-align:right;">805,004 </td>
        <td style="font-weight: bold;
                                   text-align:right;"></td>
        <td style="font-weight: bold; text-align:right">39,318,563</td>
        <td style="font-weight: bold; text-align:right;"></td>
        <td style="text-align:right;font-weight:bold;">9,449,555</td>
        <td style="font-weight: bold; text-align:right">13,343</td>
        <td style="font-weight: bold; text-align:right">148,534</td>
        <td style="font-weight: bold; text-align:right">2,412</td>
        <td style="font-weight: bold; text-align:right">755,496,996</td>
        <td style="font-weight: bold; text-align:right">2,263,665</td>
      ▶<td style="font-weight: bold; text-align:right">…</td>
        <td style="display:none" data-continent="North America">North
        America</td>
      </tr>
    ▶<tr class="total_row_world row_continent even" data-continent="South
… ıy.table.table-bordered.table-hover.main_table_countries.dataTable.no-footer  tbody  tr.odd  td  …
```

- Loop each <tr> tag and append data to res variable
- Final, we write result to target file with strategy is saving batch size

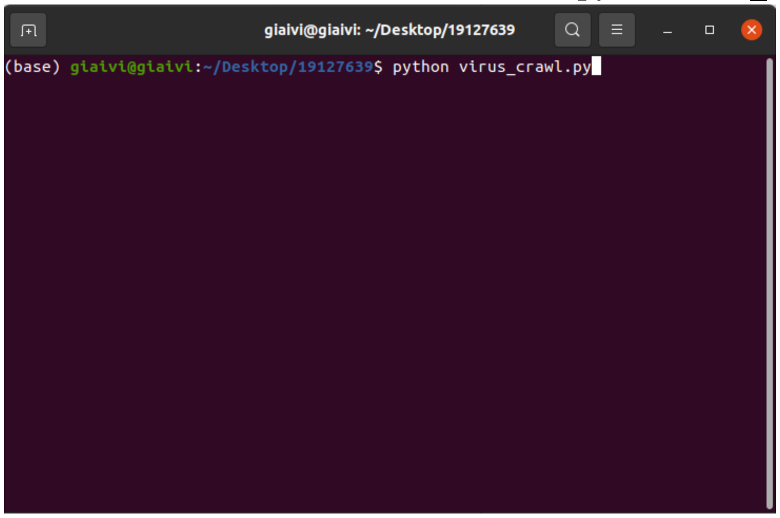
c. *How to run*

First, we open the folder which contain our source code



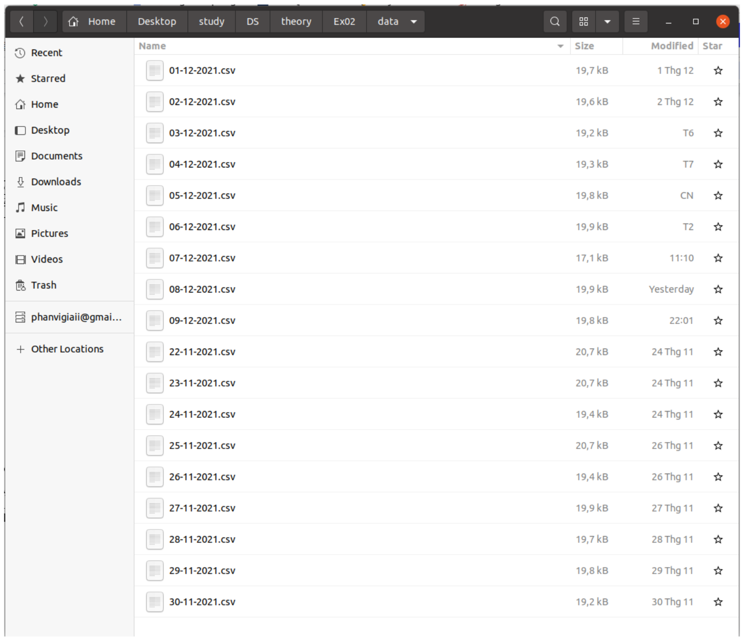
Name	Size	Modified	Star
data	10 items	20:08	☆
preprocess-data	10 items	20:09	☆
virus_crawl.py	6,4 kB	T2	☆
virus_preprocess.py	1,5 kB	T2	☆

File **virus_crawl.py** is the main file for crawling data from the website.
Then open the terminal in this folder and run the command **python virus_crawl.py**



```
(base) giaivi@giaivi:~/Desktop/19127639$ python virus_crawl.py
```

Finally, we check the result in the **data folder** which is raw data from the website we parsed.

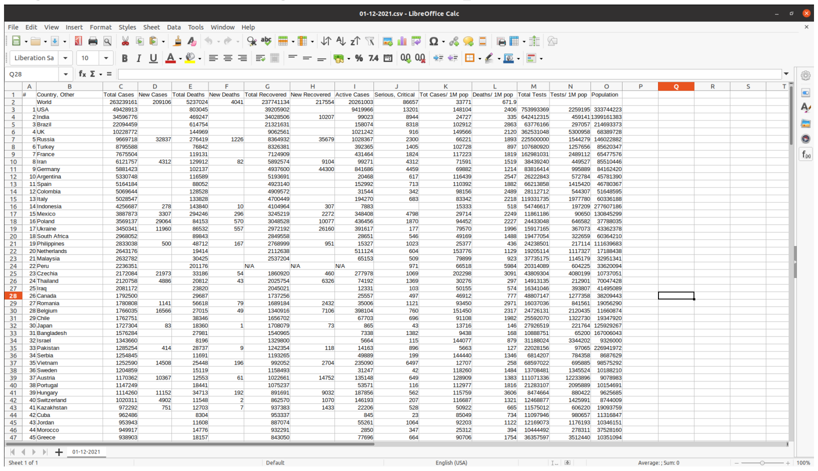


Name	Size	Modified	Star
01-12-2021.csv	19,7 kB	1 Thg 12	☆
02-12-2021.csv	19,6 kB	2 Thg 12	☆
03-12-2021.csv	19,2 kB	T6	☆
04-12-2021.csv	19,3 kB	T7	☆
05-12-2021.csv	19,8 kB	CN	☆
06-12-2021.csv	19,9 kB	T2	☆
07-12-2021.csv	17,1 kB	11:10	☆
08-12-2021.csv	19,9 kB	Yesterday	☆
09-12-2021.csv	19,8 kB	22:01	☆
22-11-2021.csv	20,7 kB	24 Thg 11	☆
23-11-2021.csv	20,7 kB	24 Thg 11	☆
24-11-2021.csv	19,4 kB	24 Thg 11	☆
25-11-2021.csv	20,7 kB	26 Thg 11	☆
26-11-2021.csv	19,4 kB	26 Thg 11	☆
27-11-2021.csv	19,9 kB	27 Thg 11	☆
28-11-2021.csv	19,7 kB	28 Thg 11	☆
29-11-2021.csv	19,8 kB	29 Thg 11	☆
30-11-2021.csv	19,2 kB	30 Thg 11	☆

2.2 Preprocessing Data

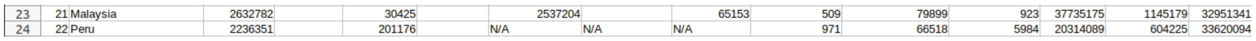
a. Approach

After crawling data from the above step, we then check our raw data to find whether this has a fault type, missing data, or something else… Just open our csv file.



#	Country, Other	Total Cases	New Cases	Total Deaths	New Deaths	Total Recovered	New Recovered	Active Cases	Serious, Critical	Tot Cases/ 1M pop	Deaths/ 1M pop	Total Tests	Tests/ 1M pop	Population
	World	263239161	209106	5237024	4041	237741134	217554	20261003	86657	33771	671.9			
1	USA	49428913		803045		39205902		9419966	13201	148104	2406	753993369	2259195	333744223
2	India	34596776		469247		34028506	10207	99023	8944	24727	335	642412315	459141	1399161383
3	Brazil	22094459		614754		21321631		158074	8318	102912	2863	63776166	297057	214693373
4	UK	10228772		144969		9062561		1021242	916	149566	2120	362531048	5300958	68389728
5	Russia	9669718	32837	276419	1226	8364932	35679	1028367	2300	66221	1893	225500000	1544279	146022882
6	Turkey	8795588		76842		8326381		392365	1405	102728	897	107680920	1257656	85620347
7	France	7675504		119131		7124909		431464	1824	117223	1819	162981031	2489112	65477576
8	Iran	6121757	4312	129912	82	5892574	9104	99271	4312	71591	1519	38439240	449527	85510446
9	Germany	5881423		102137		4937600	44300	841686	4459	69882	1214	83816414	995889	84162420
10	Argentina	5330748		116589		5193691		20468	617	116439	2547	26222843	572784	45781390
11	Spain	5164184		88052		4923140		152992	713	110392	1882	66213858	1415420	46780367
12	Colombia	5069644		128528		4909572		31544	342	98156	2489	28112712	544307	51648595
13	Italy	5028547		133828		4700449		194270	683	83342	2218	119331735	1977780	60336188
14	Indonesia	4256687	278	143840	10	4104964	307	7883		15333	518	54746617	197209	277607186
15	Mexico	3887873	3307	294246	296	3245219	2272	348408	4798	29714	2249	11861186	90650	130845299
16	Poland	3569137	29064	84153	570	3048528	10077	436456	1870	94452	2227	24433048	646582	37788035
17	Ukraine	3450341	11960	86532	557	2972192	26160	391617	177	79570	1996	15917165	367073	43362378
18	South Africa	2968052		89843		2849558		28651	546	49169	1488	19477054	322659	60364210
19	Philippines	2833038	500	48712	167	2768999	951	15327	1023	25377	436	24238501	217114	111639683
20	Netherlands	2643176		19414		2112638		511124	604	153776	1129	19205114	1117327	17188438
21	Malaysia	2632782		30425		2537204		65153	509	79899	923	37735175	1145179	32951341
22	Peru	2236351		201176		N/A	N/A	N/A	971	66518	5984	20314089	604225	33620094
23	Czechia	2172084	21973	33186	54	1860920	460	277978	1069	202298	3091	43809304	4080199	10737051
24	Thailand	2120758	4886	20812	43	2025754	6326	74192	1369	30276	297	14913135	212901	70047428
25	Iraq	2081172		23820		2045021		12331	103	50155	574	16341046	393807	41495089
26	Canada	1792500		29687		1737256		25557	497	46912	777	48807147	1277358	38209443
27	Romania	1780808	1141	56618	79	1689184	2432	35006	1121	93450	2971	16037036	841561	19056290
28	Belgium	1766035	16566	27015	49	1340916	7106	398104	760	151450	2317	24726131	2120435	11660874
29	Chile	1762751		38346		1656702		67703	696	91108	1982	25592070	1322730	19347920
30	Japan	1727304	83	18360	1	1708079	73	865	43	13716	146	27926519	221764	125929267
31	Bangladesh	1576284		27981		1540965		7338	1382	9438	168	10888751	65200	167006043
32	Israel	1343660		8196		1329800		5664	115	144077	879	31188024	3344202	9326000
33	Pakistan	1285254	414	28737	9	1242354	118	14163	896	5663	127	22028156	97065	226941972
34	Serbia	1254845		11691		1193265		49889	199	144440	1346	6814207	784358	8687629
35	Vietnam	1252590	14508	25448	196	992052	2704	235090	6497	12707	258	68597022	695885	98575292
36	Sweden	1204859		15119		1158493		31247	42	118260	1484	13708481	1345524	10188210
37	Austria	1170362	10367	12553	61	1022661	14752	135148	649	128909	1383	111071336	12233896	9078983
38	Portugal	1147249		18441		1075237		53571	116	112977	1816	21283107	2095889	10154691
39	Hungary	1114260	11152	34713	192	891691	9032	187856	562	115759	3606	8474664	880422	9625685
40	Switzerland	1020311	4902	11548	2	862570	1070	146193	207	116687	1321	12468877	1425991	8744009
41	Kazakhstan	972292	751	12703	7	937383	1433	22206	528	50922	665	11575012	606220	19093759
42	Cuba	962486		8304		953337		845	23	85049	734	11097946	980657	11316847
43	Jordan	953943		11608		887074		55261	1064	92203	1122	12169073	1176193	10346151
44	Morocco	949917		14776		932291		2850	347	25312	394	10444492	278311	37528160
45	Greece	938903		18157		843050		77696	664	90706	1754	36357597	3512440	10351094

Glance at our raw data. We just find some cells are missing and some cells have N/A value.



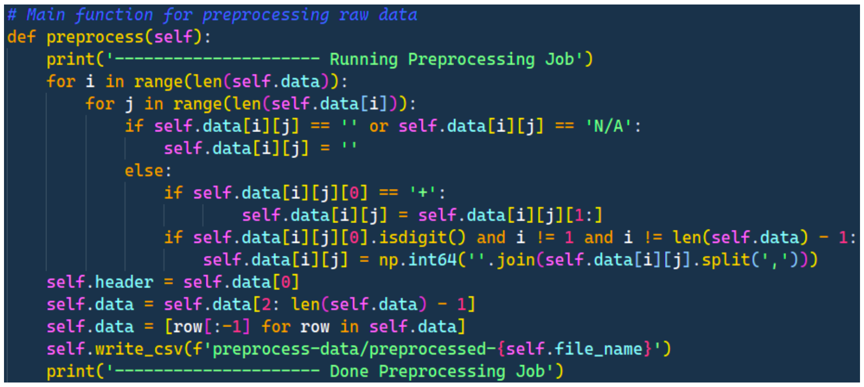
23	21 Malaysia	2632782		30425		2537204		65153	509	79899	923	37735175	1145179	32951341
24	22 Peru	2236351		201176		N/A	N/A	N/A	971	66518	5984	20314089	604225	33620094

Look at the row 23 and 24 which have problems.

b. Algorithm

The idea for preprocessing data is to replace all missing values and N/A value with empty

value.
Our algorithm is to loop through each row and replace missing values or N/A values with empty values for easily counting missing values in the later step. Then save the preprocessed data to a new folder.

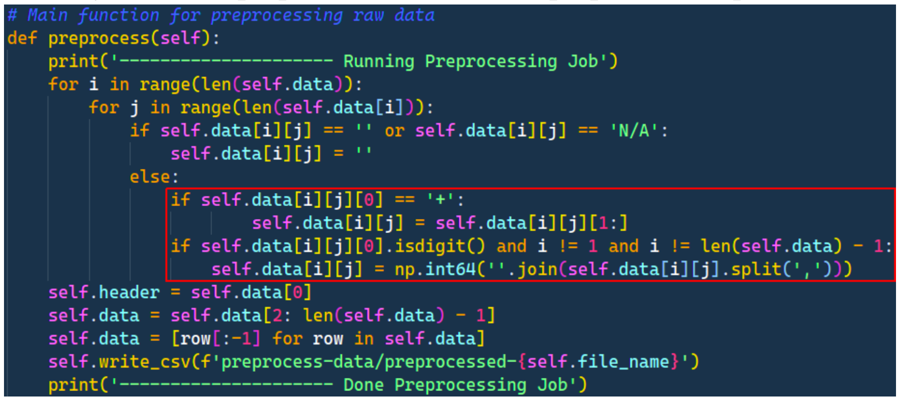


```python
# Main function for preprocessing raw data
def preprocess(self):
    print('--------------------- Running Preprocessing Job')
    for i in range(len(self.data)):
        for j in range(len(self.data[i])):
            if self.data[i][j] == '' or self.data[i][j] == 'N/A':
                self.data[i][j] = ''
            else:
                if self.data[i][j][0] == '+':
                        self.data[i][j] = self.data[i][j][1:]
                if self.data[i][j][0].isdigit() and i != 1 and i != len(self.data) - 1:
                    self.data[i][j] = np.int64(''.join(self.data[i][j].split(',')))
    self.header = self.data[0]
    self.data = self.data[2: len(self.data) - 1]
    self.data = [row[:-1] for row in self.data]
    self.write_csv(f'preprocess-data/preprocessed-{self.file_name}')
    print('--------------------- Done Preprocessing Job')
```

To run this preprocess task, the constraint is that we must has raw data first by running a crawling data task.

The **preprocess function** is the main function, which do some specific task:

- Running out each row and replace each cell has empty or N/A value with empty value
- Parse header field with the first row of raw data
- Finally, save our preprocessed data with 'preprocessed' prefix of file name



```python
# Main function for preprocessing raw data
def preprocess(self):
    print('--------------------- Running Preprocessing Job')
    for i in range(len(self.data)):
        for j in range(len(self.data[i])):
            if self.data[i][j] == '' or self.data[i][j] == 'N/A':
                self.data[i][j] = ''
            else:
                if self.data[i][j][0] == '+':
                        self.data[i][j] = self.data[i][j][1:]
                if self.data[i][j][0].isdigit() and i != 1 and i != len(self.data) - 1:
                    self.data[i][j] = np.int64(''.join(self.data[i][j].split(',')))
    self.header = self.data[0]
    self.data = self.data[2: len(self.data) - 1]
    self.data = [row[:-1] for row in self.data]
    self.write_csv(f'preprocess-data/preprocessed-{self.file_name}')
    print('--------------------- Done Preprocessing Job')
```

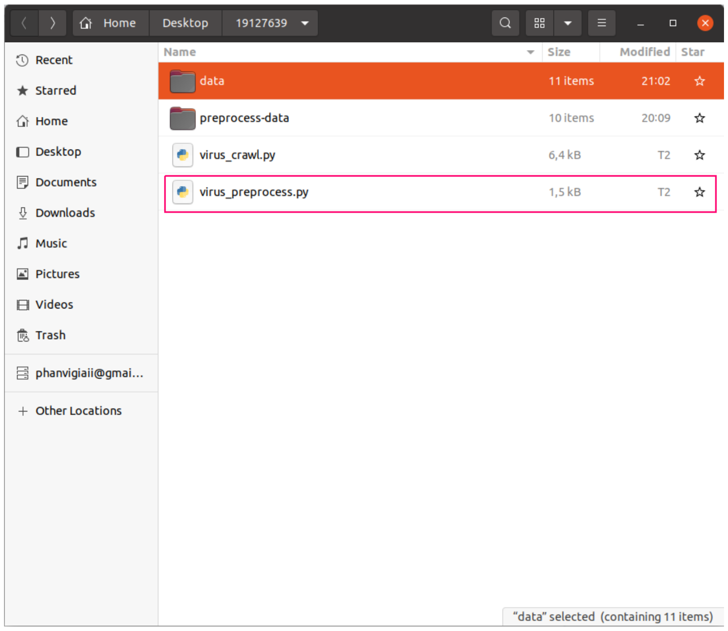
- In the field 'New Cases' and 'New Deaths' and 'New Recovered', data has '+' in the prefix indicating some cases added. So that we must remove a '+' in the prefix
- All the number has format ',' so we must remove the ',' and cast the type to int

Advantage of this approach is:

- Easy
- Fast
- No performance for calculating

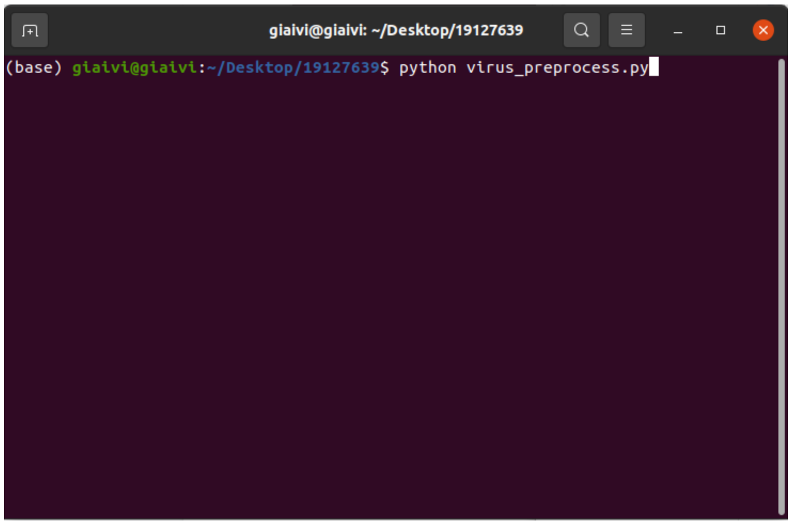
c. *How to run*

To run this preprocess data job, we need to run crawl data first.
First, run crawl data to get our raw data.
Then open a folder which contains our source code.



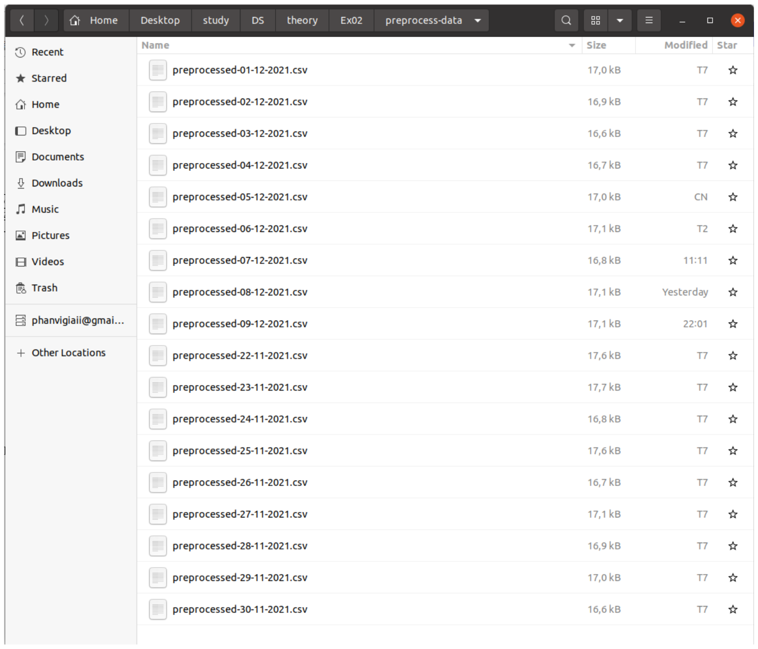
Name	Size	Modified	Star
data	11 items	21:02	☆
preprocess-data	10 items	20:09	☆
virus_crawl.py	6,4 kB	T2	☆
virus_preprocess.py	1,5 kB	T2	☆

"data" selected (containing 11 items)



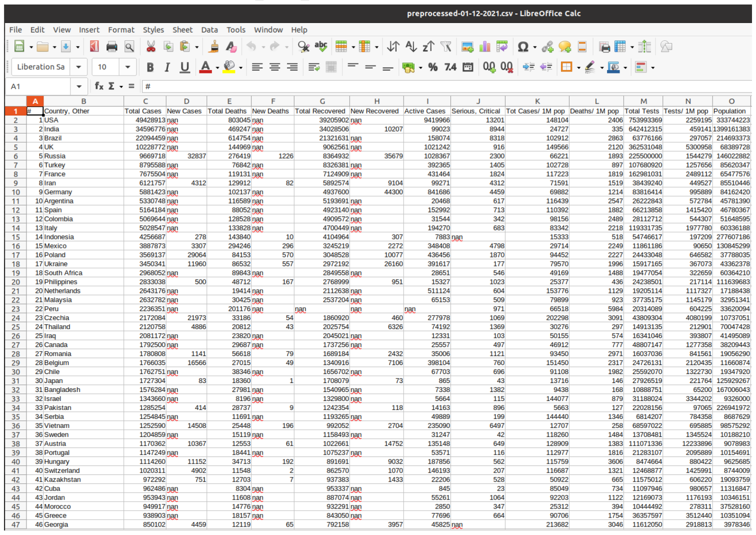
```
(base) giaivi@giaivi:~/Desktop/19127639$ python virus_preprocess.py
```

Run command line **python virus_preprocess.py**



Name	Size	Modified	Star
preprocessed-01-12-2021.csv	17,0 kB	T7	☆
preprocessed-02-12-2021.csv	16,9 kB	T7	☆
preprocessed-03-12-2021.csv	16,6 kB	T7	☆
preprocessed-04-12-2021.csv	16,7 kB	T7	☆
preprocessed-05-12-2021.csv	17,0 kB	CN	☆
preprocessed-06-12-2021.csv	17,1 kB	T2	☆
preprocessed-07-12-2021.csv	16,8 kB	11:11	☆
preprocessed-08-12-2021.csv	17,1 kB	Yesterday	☆
preprocessed-09-12-2021.csv	17,1 kB	22:01	☆
preprocessed-22-11-2021.csv	17,6 kB	T7	☆
preprocessed-23-11-2021.csv	17,7 kB	T7	☆
preprocessed-24-11-2021.csv	16,8 kB	T7	☆
preprocessed-25-11-2021.csv	17,6 kB	T7	☆
preprocessed-26-11-2021.csv	16,7 kB	T7	☆
preprocessed-27-11-2021.csv	17,1 kB	T7	☆
preprocessed-28-11-2021.csv	16,9 kB	T7	☆
preprocessed-29-11-2021.csv	17,0 kB	T7	☆
preprocessed-30-11-2021.csv	16,6 kB	T7	☆

Then check the result in **folder preprocess-data**



#	Country, Other	Total Cases	New Cases	Total Deaths	New Deaths	Total Recovered	New Recovered	Active Cases	Serious, Critical	Tot Cases/ 1M pop	Deaths/ 1M pop	Total Tests	Tests/ 1M pop	Population
1	USA	49428913	nan	803045	nan	39205902	nan	9419966	13201	148104	2406	753993369	2259195	333744223
2	India	34596776	nan	469247	nan	34028506	10207	99023	8944	24727	335	642412315	459141	1399161383
3	Brazil	22094459	nan	614754	nan	21321631	nan	158074	8318	102912	2863	63776166	297057	214693373
4	UK	10228772	nan	144969	nan	9062561	nan	1021242	916	149566	2120	362531048	5300958	68389728
5	Russia	9669718	32837	276419	1226	8364932	35679	1028367	2300	66221	1893	225500000	1544279	146022882
6	Turkey	8795588	nan	76842	nan	8326381	nan	392365	1405	102728	897	107680920	1257656	85620347
7	France	7675504	nan	119131	nan	7124909	nan	431464	1824	117223	1819	162981031	2489112	65477576
8	Iran	6121757	4312	129912	82	5892574	9104	99271	4312	71591	1519	38439240	449527	85510446
9	Germany	5881423	nan	102137	nan	4937600	44300	841686	4459	69882	1214	83816414	995889	84162420
10	Argentina	5330748	nan	116589	nan	5193691	nan	20468	617	116439	2547	26222843	572784	45781390
11	Spain	5164184	nan	88052	nan	4923140	nan	152992	713	110392	1882	66213858	1415420	46780367
12	Colombia	5069644	nan	128528	nan	4909572	nan	31544	342	98156	2489	28112712	544307	51648595
13	Italy	5028547	nan	133828	nan	4700449	nan	194270	683	83342	2218	119331735	1977780	60336188
14	Indonesia	4256687	278	143840	10	4104964	307	7883	nan	15333	518	54746617	197209	277607186
15	Mexico	3887873	3307	294246	296	3245219	2272	348408	4798	29714	2249	11861186	90650	130845299
16	Poland	3569137	29064	84153	570	3048528	10077	436456	1870	94452	2227	24433048	646582	37788035
17	Ukraine	3450341	11960	86532	557	2972192	26160	391617	177	79570	1996	15917165	367073	43362378
18	South Africa	2968052	nan	89843	nan	2849558	nan	28651	546	49169	1488	19477054	322659	60364210
19	Philippines	2833038	500	48712	167	2768999	951	15327	1023	25377	436	24238501	217114	111639683
20	Netherlands	2643176	nan	19414	nan	2112638	nan	511124	604	153776	1129	19205114	1117327	17188438
21	Malaysia	2632782	nan	30425	nan	2537204	nan	65153	509	79899	923	37735175	1145179	32951341
22	Peru	2236351	nan	201176	nan	nan	nan	nan	971	66518	5984	20314089	604225	33620094
23	Czechia	2172084	21973	33186	54	1860920	460	277978	1069	202298	3091	43809304	4080199	10737051
24	Thailand	2120758	4886	20812	43	2025754	6326	74192	1369	30276	297	14913135	212901	70047428
25	Iraq	2081172	nan	23820	nan	2045021	nan	12331	103	50155	574	16341046	393807	41495089
26	Canada	1792500	nan	29687	nan	1737256	nan	25557	497	46912	777	48807147	1277358	38209443
27	Romania	1780808	1141	56618	79	1689184	2432	35006	1121	93450	2971	16037036	841561	19056290
28	Belgium	1766035	16566	27015	49	1340916	7106	398104	760	151450	2317	24726131	2120435	11660874
29	Chile	1762751	nan	38346	nan	1656702	nan	67703	696	91108	1982	25592070	1322730	19347920
30	Japan	1727304	83	18360	1	1708079	73	865	43	13716	146	27926519	221764	125929267
31	Bangladesh	1576284	nan	27981	nan	1540965	nan	7338	1382	9438	168	10888751	65200	167006043
32	Israel	1343660	nan	8196	nan	1329800	nan	5664	115	144077	879	31188024	3344202	9326000
33	Pakistan	1285254	414	28737	9	1242354	118	14163	896	5663	127	22028156	97065	226941972
34	Serbia	1254845	nan	11691	nan	1193265	nan	49889	199	144440	1346	6814207	784358	8687629
35	Vietnam	1252590	14508	25448	196	992052	2704	235090	6497	12707	258	68597022	695885	98575292
36	Sweden	1204859	nan	15119	nan	1158493	nan	31247	42	118260	1484	13708481	1345524	10188210
37	Austria	1170362	10367	12553	61	1022661	14752	135148	649	128909	1383	111071336	12233896	9078983
38	Portugal	1147249	nan	18441	nan	1075237	nan	53571	116	112977	1816	21283107	2095889	10154691
39	Hungary	1114260	11152	34713	192	891691	9032	187856	562	115759	3606	8474664	880422	9625685
40	Switzerland	1020311	4902	11548	2	862570	1070	146193	207	116687	1321	12468877	1425991	8744009
41	Kazakhstan	972292	751	12703	7	937383	1433	22206	528	50922	665	11575012	606220	19093759
42	Cuba	962486	nan	8304	nan	953337	nan	845	23	85049	734	11097946	980657	11316847
43	Jordan	953943	nan	11608	nan	887074	nan	55261	1064	92203	1122	12169073	1176193	10346151
44	Morocco	949917	nan	14776	nan	932291	nan	2850	347	25312	394	10444492	278311	37528160
45	Greece	938903	nan	18157	nan	843050	nan	77696	664	90706	1754	36357597	3512440	10351094
46	Georgia	850102	4459	12119	65	792158	3957	45825	nan	213682	3046	11612050	2918813	3978346

missing data or N/A value is replaced with nan value

2.3 Automatic crawl and preprocess data with Apache AirFlow

a. Approach

In general, we manually run two command line for crawling and preprocessing data:

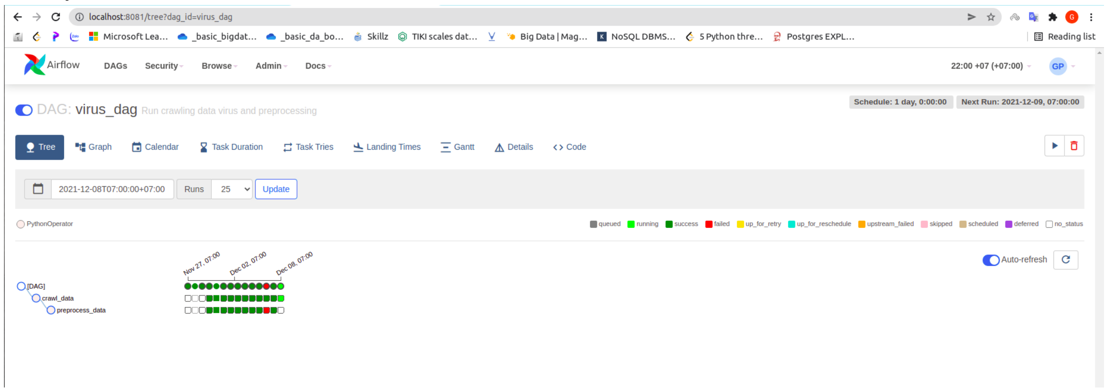
- python virus_crawl.py
- python virus_preprocessing.py

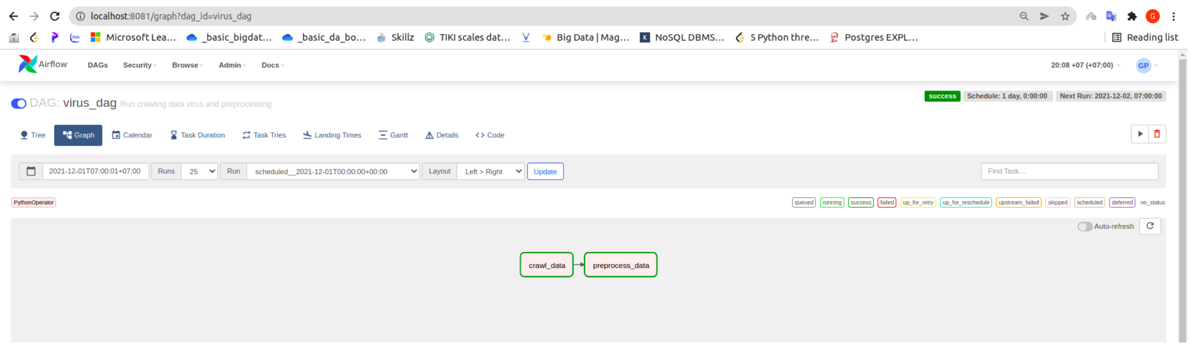
In this project, we must run two command line above ***daily*** for collecting data from the website: https://www.worldometers.info/coronavirus/

But, can we do those tasks in an automatic way ? Yes, we can do it.

What is the AirFlow ? Airflow is a platform to programmatically author, schedule and monitor workflows or data pipelines.

Using the AirFlow, we can automatically schedule our tasks. In this project, we build one data pipeline for crawling and then preprocess our data **daily at midnight** to collect almost full data of that day.


localhost:8081/tree?dag_id=virus_dag
Airflow
DAGs
Security
Browse
Admin
Docs
22:00 +07 (+07:00)
DAG: virus_dag Run crawling data virus and preprocessing
Schedule: 1 day, 0:00:00
Next Run: 2021-12-09, 07:00:00
Tree
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
2021-12-08T07:00:00+07:00
Runs 25
Update
PythonOperator
queued running success failed up_for_retry up_for_reschedule upstream_failed skipped scheduled deferred no_status
Auto-refresh
Nov 27, 07:00
Dec 02, 07:00
Dec 08, 07:00
[DAG]
crawl_data
preprocess_data


localhost:8081/graph?dag_id=virus_dag
Airflow
DAGs
Security
Browse
Admin
Docs
20:08 +07 (+07:00)
DAG: virus_dag Run crawling data virus and preprocessing
success
Schedule: 1 day, 0:00:00
Next Run: 2021-12-02, 07:00:00
Tree
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
2021-12-01T07:00:01+07:00
Runs 25
Run scheduled__2021-12-01T00:00:00+00:00
Layout Left > Right
Update
Find Task...
PythonOperator
queued running success failed up_for_retry up_for_reschedule upstream_failed skipped scheduled deferred no_status
Auto-refresh
crawl_data
preprocess_data

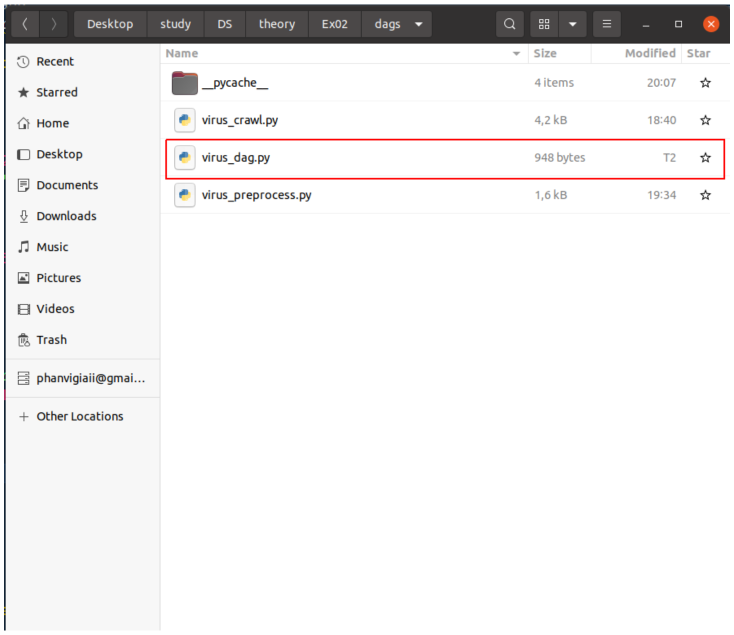
AirFlow dashboard

b. Algorithm

To build the data pipeline, we need the **dag.**

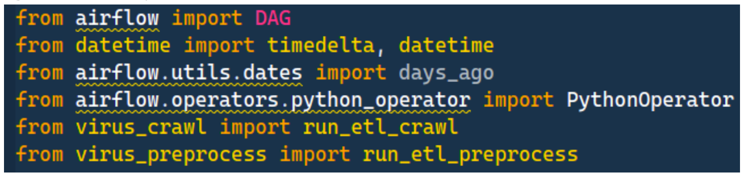
What is **dag** ? **Dag** contains these attributes:

- Workflow as a Directed Acyclic Graph (DAG) with multiple tasks which can be executed independently.
- Airflow DAGs are composed of Tasks.



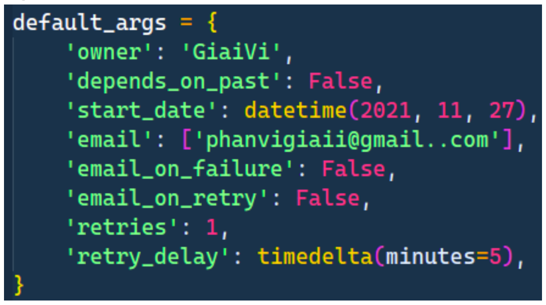
```
Name                 Size        Modified  Star
__pycache__          4 items     20:07
virus_crawl.py       4,2 kB      18:40
virus_dag.py         948 bytes   T2
virus_preprocess.py  1,6 kB      19:34
```

First, we create a **dag file** named **virus_dag.py**
Following these step in **Apache AirFlow** website
Step 1: Importing necessary modules



```python
from airflow import DAG
from datetime import timedelta, datetime
from airflow.utils.dates import days_ago
from airflow.operators.python_operator import PythonOperator
from virus_crawl import run_etl_crawl
from virus_preprocess import run_etl_preprocess
```

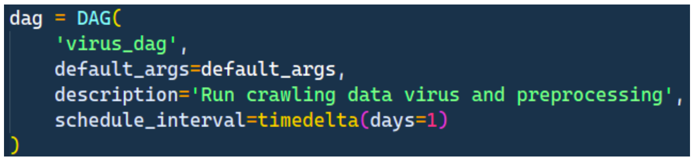
Step 2: Define default argument



```python
default_args = {
    'owner': 'GiaiVi',
    'depends_on_past': False,
    'start_date': datetime(2021, 11, 27),
    'email': ['phanvigiaii@gmail..com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}
```

- Owner is me
- Start date: from 27-11-2021
- Retries: 1 time
- Retry delay: 5 minutes after failure

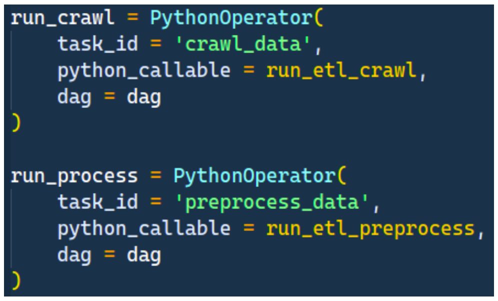
Step 3: Instantiate a DAG



```python
dag = DAG(
    'virus_dag',
    default_args=default_args,
    description='Run crawling data virus and preprocessing',
    schedule_interval=timedelta(days=1)
)
```

- dag id: virus_dag
- Describe the dag with description field
- schedule_interval: 1 day (mean daily)

Step 4: Specify our tasks

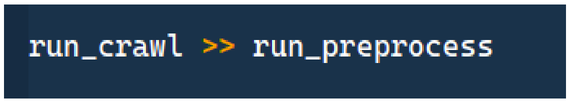


```python
run_crawl = PythonOperator(
    task_id = 'crawl_data',
    python_callable = run_etl_crawl,
    dag = dag
)

run_process = PythonOperator(
    task_id = 'preprocess_data',
    python_callable = run_etl_preprocess,
    dag = dag
)
```

- We have two task: crawling data and preprocessing data
- Use PythonOperator for calling python function
- task_id is our task name: crawl_data and preprocess_data
- python_callable is passed with our imported function

Step 5: Setting up dependencies

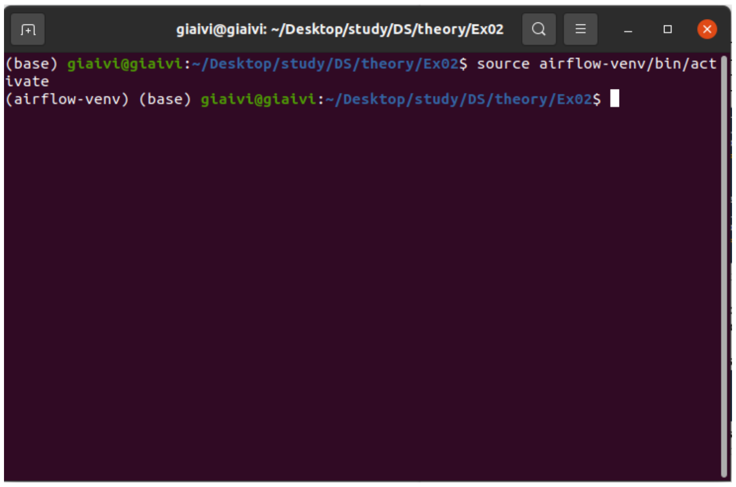


```python
run_crawl >> run_preprocess
```

- We specify **run_process task** depend on **run_crawl**
- Because we need raw data for preprocessing

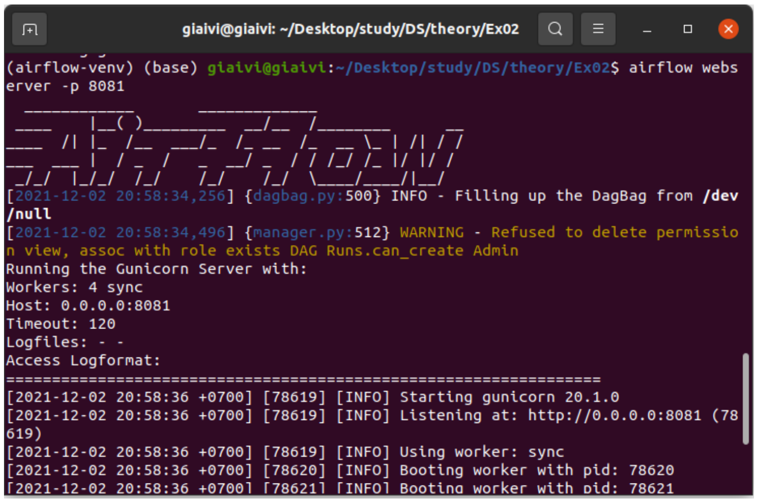
c. *How to run*

First, install AirFlow from official website



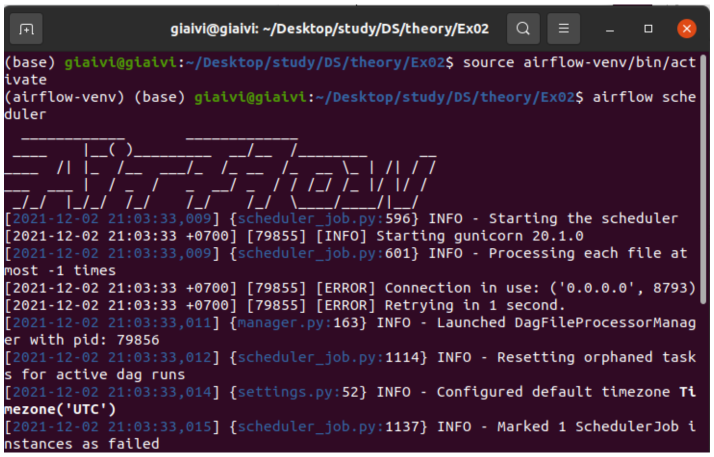
```
(base) giaivi@giaivi:~/Desktop/study/DS/theory/Ex02$ source airflow-venv/bin/act
ivate
(airflow-venv) (base) giaivi@giaivi:~/Desktop/study/DS/theory/Ex02$ 
```

Activate AirFlow environment with command: **source airflow-venv/bin/activate**



```
(airflow-venv) (base) giaivi@giaivi:~/Desktop/study/DS/theory/Ex02$ airflow webs
erver -p 8081
  ____________       _____________
 ____    |__( )_________  __/__  /________      __
____  /| |_  /__  ___/_  /_ __  /_  __ \_ | /| / /
___  ___ |  / _  /   _  __/ _  / / /_/ /_ |/ |/ /
 _/_/  |_/_/  /_/    /_/    /_/  \____/____/|__/
[2021-12-02 20:58:34,256] {dagbag.py:500} INFO - Filling up the DagBag from /dev
/null
[2021-12-02 20:58:34,496] {manager.py:512} WARNING - Refused to delete permissio
n view, assoc with role exists DAG Runs.can_create Admin
Running the Gunicorn Server with:
Workers: 4 sync
Host: 0.0.0.0:8081
Timeout: 120
Logfiles: - -
Access Logformat:
=================================================================
[2021-12-02 20:58:36 +0700] [78619] [INFO] Starting gunicorn 20.1.0
[2021-12-02 20:58:36 +0700] [78619] [INFO] Listening at: http://0.0.0.0:8081 (78
619)
[2021-12-02 20:58:36 +0700] [78619] [INFO] Using worker: sync
[2021-12-02 20:58:36 +0700] [78620] [INFO] Booting worker with pid: 78620
[2021-12-02 20:58:36 +0700] [78621] [INFO] Booting worker with pid: 78621
```

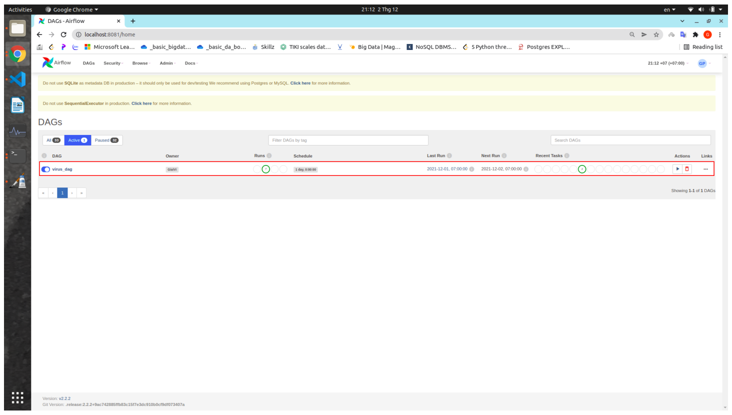
Running airflow server with command: **airflow webserver -p 8081**
Run server with port 8081



```
giaivi@giaivi: ~/Desktop/study/DS/theory/Ex02
(base) giaivi@giaivi:~/Desktop/study/DS/theory/Ex02$ source airflow-venv/bin/act
ivate
(airflow-venv) (base) giaivi@giaivi:~/Desktop/study/DS/theory/Ex02$ airflow sche
duler
  ____________       _____________
 ____    |__( )_________  __/__  /________      __
____  /| |_  /__  ___/_  /_ __  /_  __ \_ | /| / /
___  ___ |  / _  /   _  __/ _  / / /_/ /_ |/ |/ /
 _/_/  |_/_/  /_/    /_/    /_/  \____/____/|__/
[2021-12-02 21:03:33,009] {scheduler_job.py:596} INFO - Starting the scheduler
[2021-12-02 21:03:33 +0700] [79855] [INFO] Starting gunicorn 20.1.0
[2021-12-02 21:03:33,009] {scheduler_job.py:601} INFO - Processing each file at
most -1 times
[2021-12-02 21:03:33 +0700] [79855] [ERROR] Connection in use: ('0.0.0.0', 8793)
[2021-12-02 21:03:33 +0700] [79855] [ERROR] Retrying in 1 second.
[2021-12-02 21:03:33,011] {manager.py:163} INFO - Launched DagFileProcessorManag
er with pid: 79856
[2021-12-02 21:03:33,012] {scheduler_job.py:1114} INFO - Resetting orphaned task
s for active dag runs
[2021-12-02 21:03:33,014] {settings.py:52} INFO - Configured default timezone Ti
mezone('UTC')
[2021-12-02 21:03:33,015] {scheduler_job.py:1137} INFO - Marked 1 SchedulerJob i
nstances as failed
```

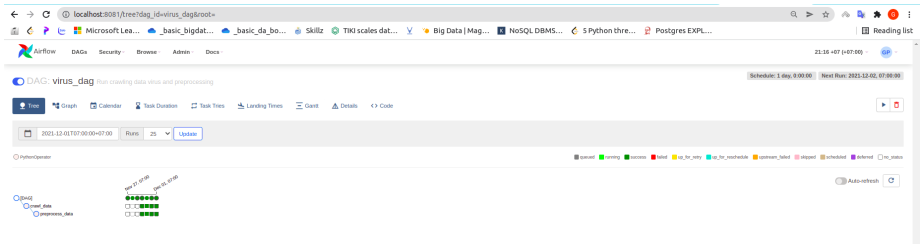
Open another terminal, activate airflow environment like above step
Then, running airflow scheduler server with command: **airflow scheduler**

Finally, open web browser and access to http://localhost:8081/

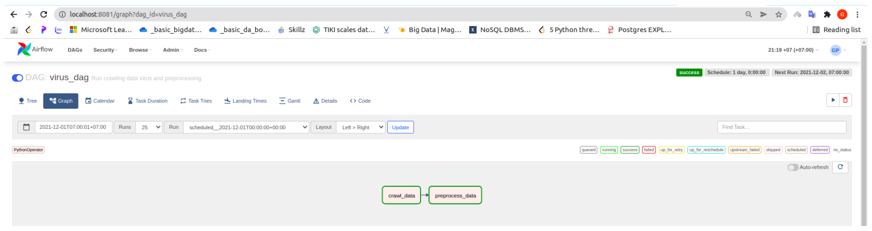


AirFlow dashboard with **virus_dag** created above

Click on our dag and we have a control panel for manipulating tasks and so on...



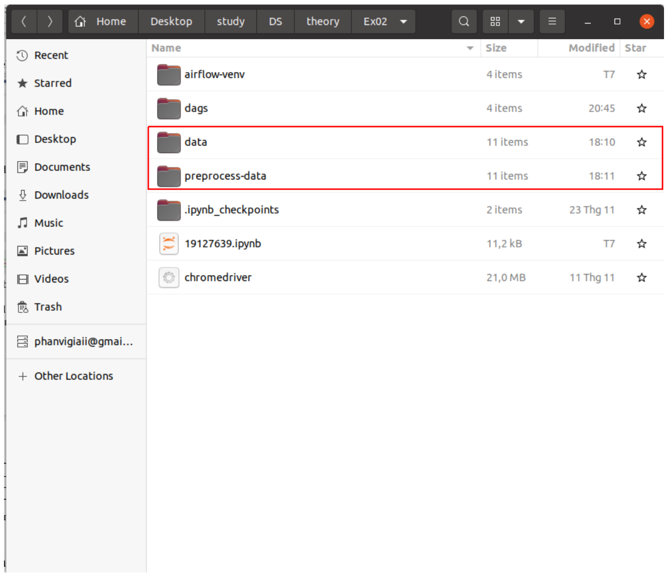
Our scheduler



View dependable tasks

That is all, with AirFlow we can automatically run our job. But in this level, we don't have a server for deployment to running AirFlow server and scheduler 24/7.

Every day, the only thing we just do is run one command: **airflow scheduler**. The scheduler starts and automatically runs our job.



Name	Size	Modified	Star
airflow-venv	4 items	T7	☆
dags	4 items	20:45	☆
data	11 items	18:10	☆
preprocess-data	11 items	18:11	☆
.ipynb_checkpoints	2 items	23 Thg 11	☆
19127639.ipynb	11,2 kB	T7	☆
chromedriver	21,0 MB	11 Thg 11	☆

View a result in **folder data** which contains raw data and **folder preprocess-data** which contains preprocessed data.

3. Statistic and Data Visualization

3.1 Expectations

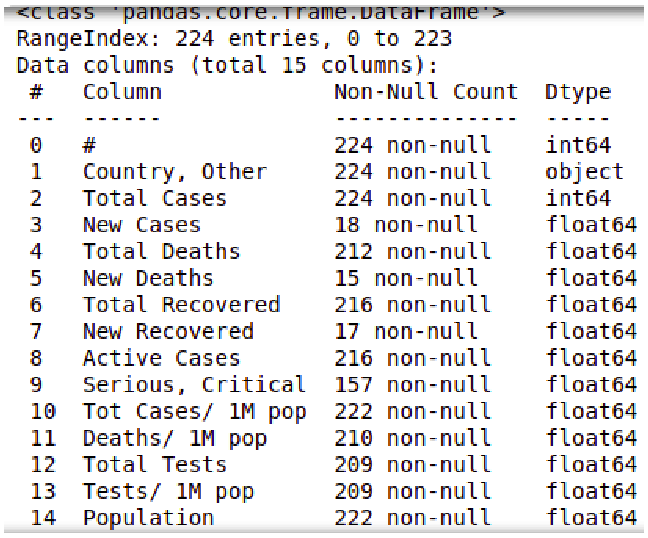
After finishing to collect data from the website, what do we expect from our data ? In my expectations:

- We will get some columns depending on others like covariate, inverse
- We expect main column will be in standard deviation
- Percentage losing of data very small
- Type of columns are correct and reasonable

3.2 Statistic and Conclusion

a. Statistic

First, we read the **"preprocessed-04-12-2021.csv"** file in the preprocess-data folder. We have the result below:

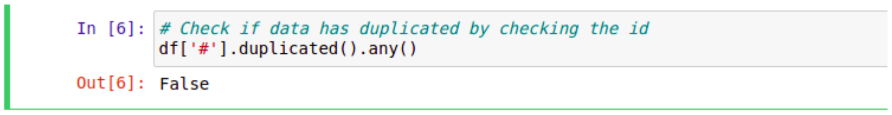


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 224 entries, 0 to 223
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   #                  224 non-null    int64
 1   Country, Other     224 non-null    object
 2   Total Cases        224 non-null    int64
 3   New Cases          18 non-null     float64
 4   Total Deaths       212 non-null    float64
 5   New Deaths         15 non-null     float64
 6   Total Recovered    216 non-null    float64
 7   New Recovered      17 non-null     float64
 8   Active Cases       216 non-null    float64
 9   Serious, Critical  157 non-null    float64
 10  Tot Cases/ 1M pop  222 non-null    float64
 11  Deaths/ 1M pop     210 non-null    float64
 12  Total Tests        209 non-null    float64
 13  Tests/ 1M pop      209 non-null    float64
 14  Population         222 non-null    float64
```

In total, data has 15 columns and 224 rows.

Each column has the meaning:

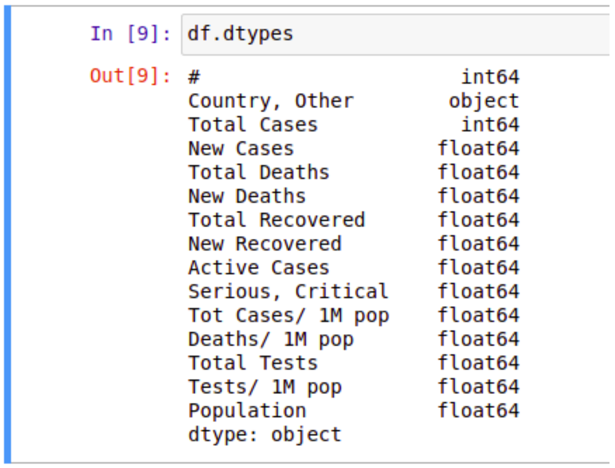
- "#" mean id of the row
- "Country, Other" mean the country which is reported about virus
- "Total Cases" mean total cases are calculated from the way it's always been.
- "New Cases" mean cases people are inflected in that day
- "Total Deaths" mean total deaths from that country until that day
- "New Deaths" mean new deaths happen in that day
- "Total Recovered" mean total cases recovered until that day
- "New Recovered" mean new cases recovered in that day
- "Active Cases" mean cases is inflected in that day which is calculated by "Total Cases" minus "Total Deaths"
- "Serious, Critical" mean some cases in danger, severe illness
- "Tot Cases/ 1M pop" mean total cases but is calculated by 1 million population
- "Deaths/ 1M pop" mean total deaths but is calculated by 1 million population
- "Total Tests" mean total tests which country tested virus for people until that day
- "Tests/ 1M pop" mean total tests but is calculated by 1 million population
- "Population" mean total number of people in the country



```
In [6]: # Check if data has duplicated by checking the id
        df['#'].duplicated().any()

Out[6]: False
```

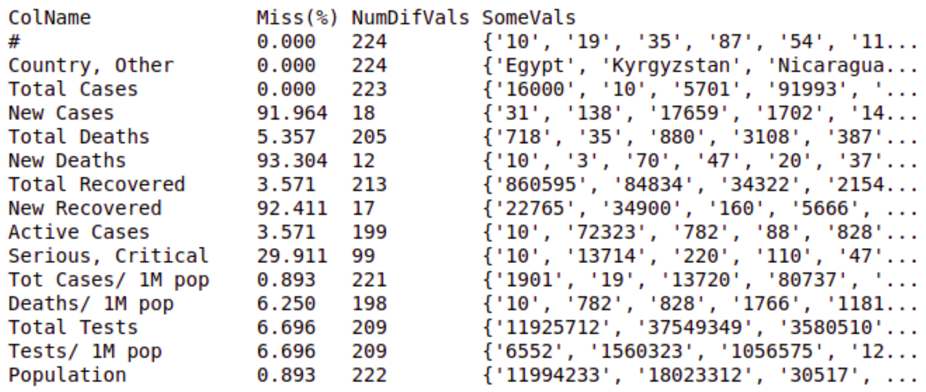
In data, there is no duplicated row by using built-in function of DataFrame to check it and return False



```
In [9]: df.dtypes

Out[9]: #                       int64
        Country, Other         object
        Total Cases             int64
        New Cases             float64
        Total Deaths          float64
        New Deaths            float64
        Total Recovered       float64
        New Recovered         float64
        Active Cases          float64
        Serious, Critical     float64
        Tot Cases/ 1M pop     float64
        Deaths/ 1M pop        float64
        Total Tests           float64
        Tests/ 1M pop         float64
        Population            float64
        dtype: object
```

Column "#", "Total Cases" has type int64 for numerical value
Column "Country, Other" has type object for string value
The rest have type float64 for numerical value



```
ColName             Miss(%) NumDifVals SomeVals
#                   0.000   224        {'10', '19', '35', '87', '54', '11...
Country, Other      0.000   224        {'Egypt', 'Kyrgyzstan', 'Nicaragua...
Total Cases         0.000   223        {'16000', '10', '5701', '91993', '...
New Cases           91.964  18         {'31', '138', '17659', '1702', '14...
Total Deaths        5.357   205        {'718', '35', '880', '3108', '387'...
New Deaths          93.304  12         {'10', '3', '70', '47', '20', '37'...
Total Recovered     3.571   213        {'860595', '84834', '34322', '2154...
New Recovered       92.411  17         {'22765', '34900', '160', '5666', ...
Active Cases        3.571   199        {'10', '72323', '782', '88', '828'...
Serious, Critical   29.911  99         {'10', '13714', '220', '110', '47'...
Tot Cases/ 1M pop   0.893   221        {'1901', '19', '13720', '80737', '...
Deaths/ 1M pop      6.250   198        {'10', '782', '828', '1766', '1181...
Total Tests         6.696   209        {'11925712', '37549349', '3580510'...
Tests/ 1M pop       6.696   209        {'6552', '1560323', '1056575', '12...
Population          0.893   222        {'11994233', '18023312', '30517', ...
```

The image above summary missing and different values which is calculated from our data
Base on the result:

- The columns "#", "Country, Other" and "Total Cases" have 0% missing mean that those attributes has no missing value and two divide three has unique values but not the column "Total Cases" has duplicated in two last row
- The columns "New Cases" and "New Deaths" are the largest missing percentages. The Ministry of Health can not record new cases and new deaths everyday because of the exploding infection rate in the country
- The remain has normal value

b. Conclusion

Through some statistical information, we can recognize that the inflection of CoronaVirus has exploded all over the world.
The new types of CoronaVirus has been raise, quote from one paper:
"A new COVID-19 variant detected in South Africa with a high number of mutations has raised concerns among scientists and triggered travel restrictions by a number of countries amid fears of coronavirus transmissions. The National Institute for Communicable Diseases (NICD) said 22 positive cases of the new variant have been recorded in the country following genomic sequencing. News of the announcement broke on Thursday…"
That is the alarm for all people in the world of the dangerous dead if the government has no countermeasures.
The survival of people depends on their consciousness and the sacrifices of doctors around the world.

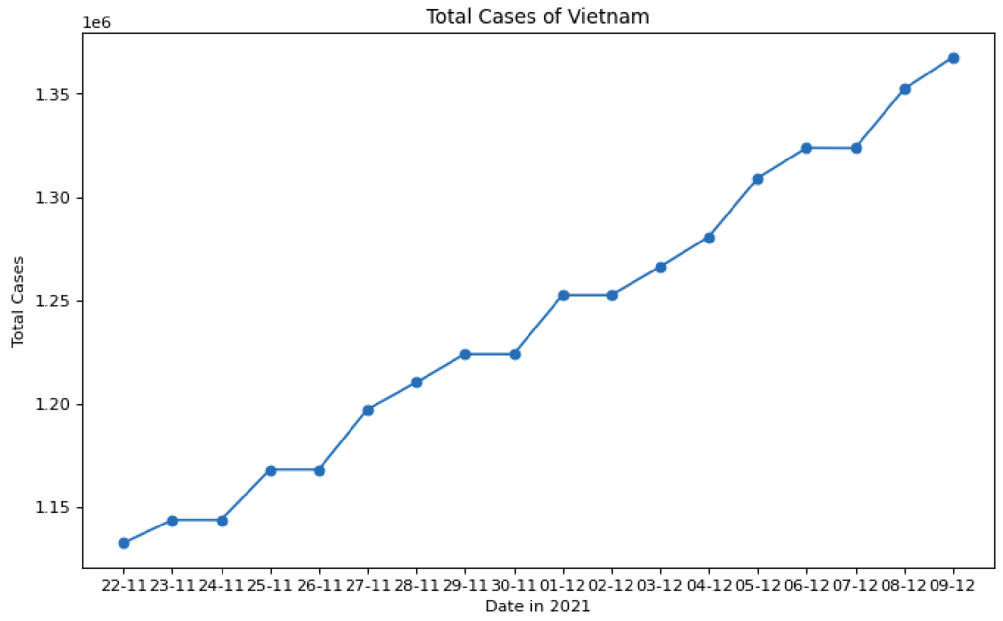
3.3 Visualization

a. Choose column for visualizing

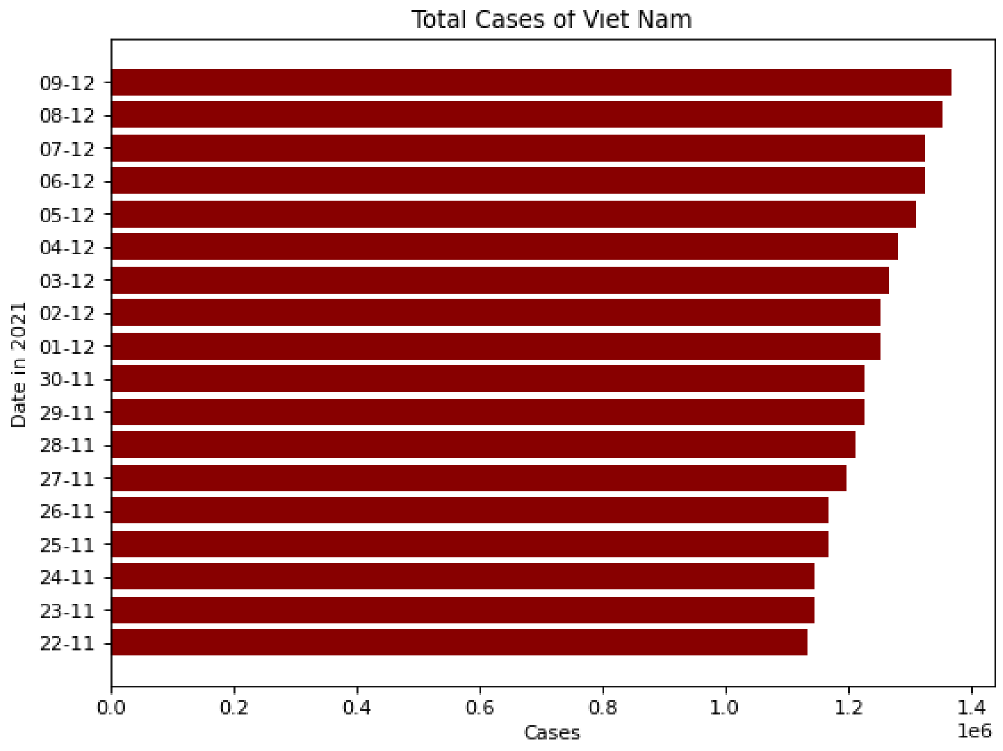
Because "#" is a column specifying Id for each row and "Country, Other" is a column describing data of where. So that we choose the remaining column for the variable of the visualization chart.

b. Comparison Plots

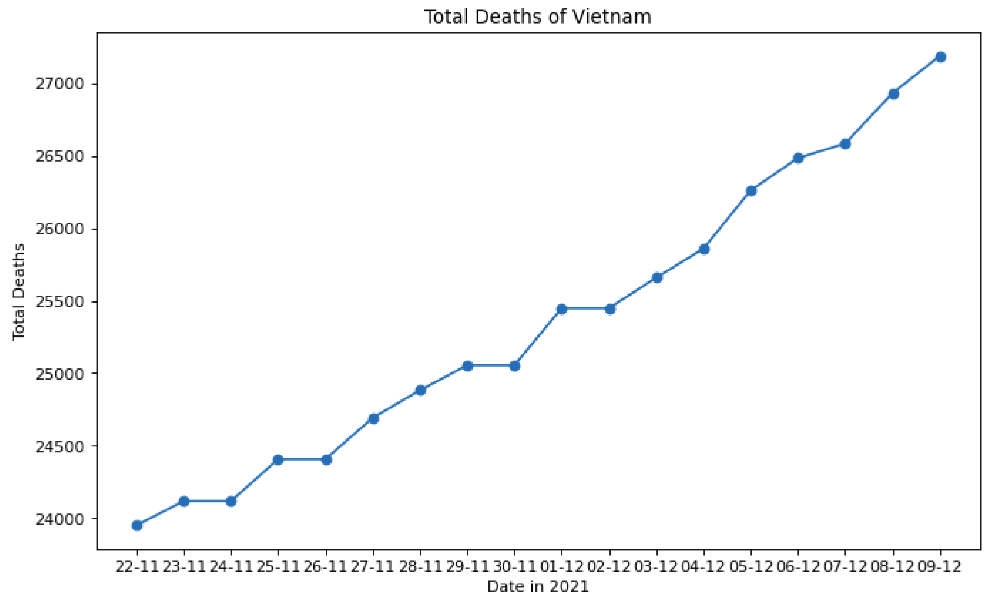
First, we visualize each column of data with a relation date. To know how the trend of cases increases or decreases until now ? In this part, we use **line charts** and **bar charts** for visualizing data because three columns "Total Cases", "Total Recovered" and "Total Deaths" are numerical types and we want to see how the data change overtime.

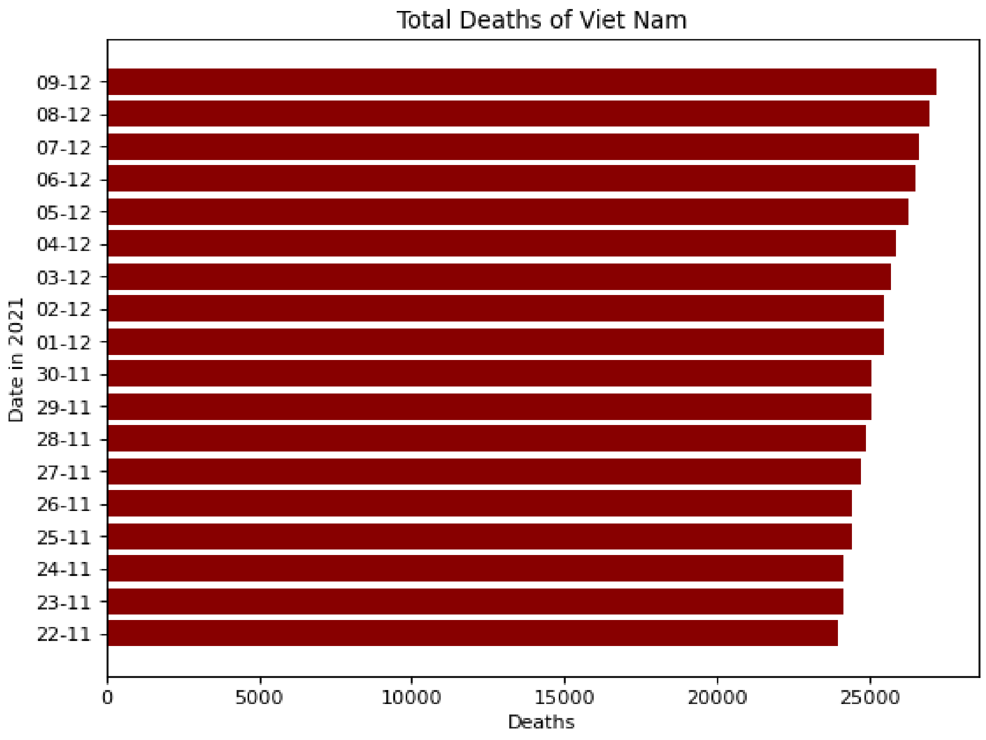


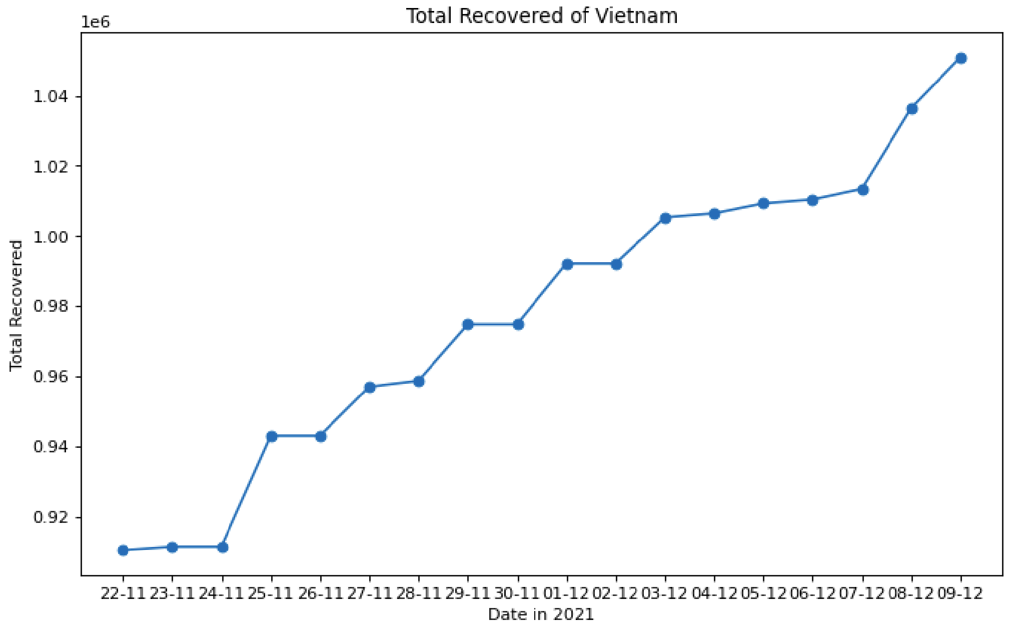
Total Cases of Vietnam
1e6
Total Cases
1.15
1.20
1.25
1.30
1.35
22-11 23-11 24-11 25-11 26-11 27-11 28-11 29-11 30-11 01-12 02-12 03-12 04-12 05-12 06-12 07-12 08-12 09-12
Date in 2021

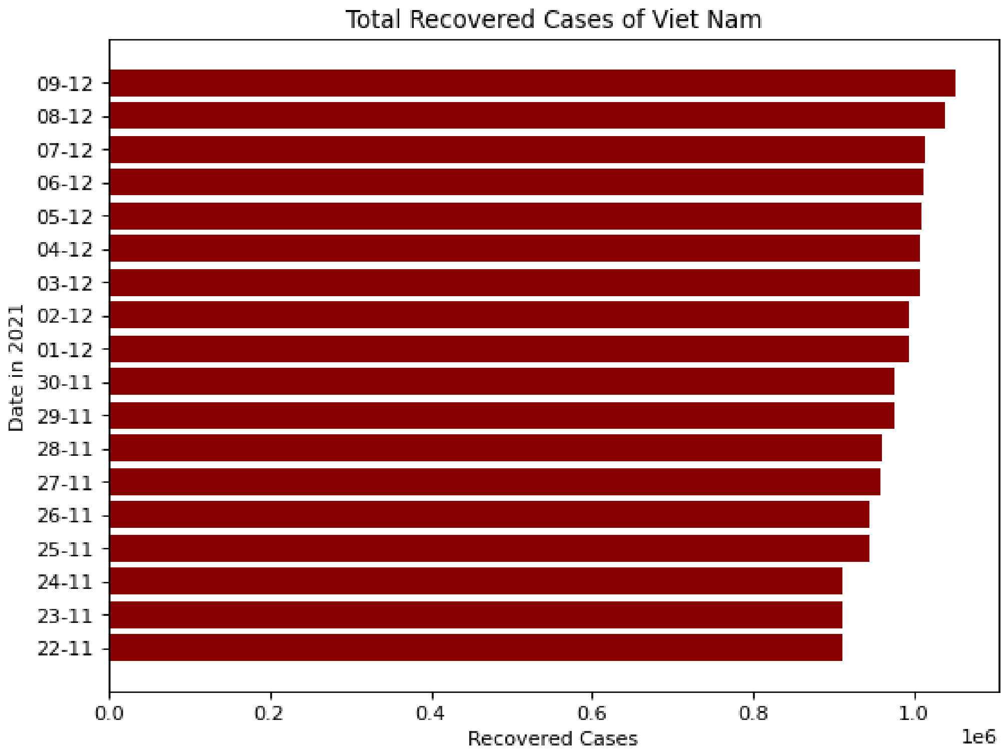


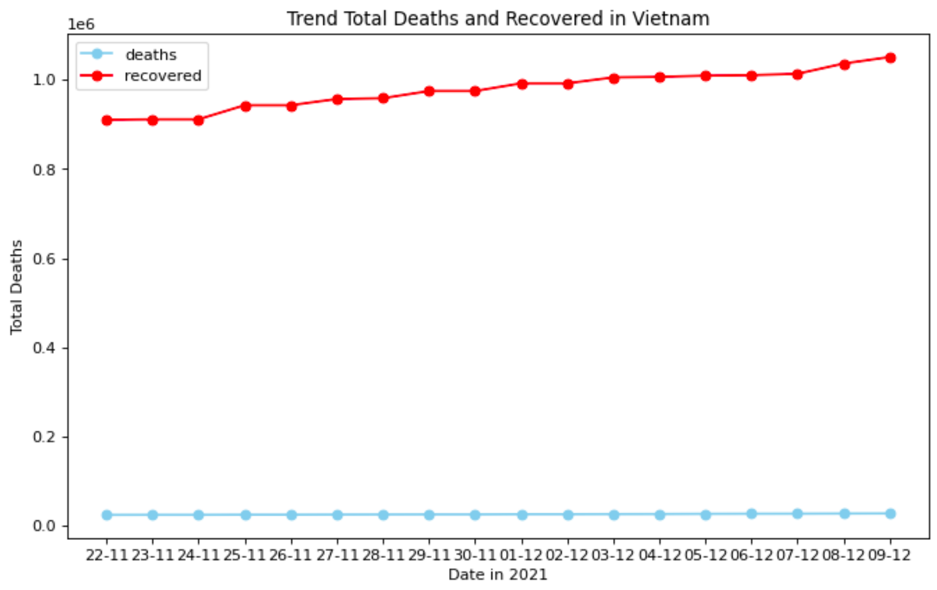
Total Cases of Viet Nam
09-12
08-12
07-12
06-12
05-12
04-12
03-12
02-12
01-12
30-11
29-11
28-11
27-11
26-11
25-11
24-11
23-11
22-11
Date in 2021
0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4
Cases
1e6


Total Deaths of Vietnam
Total Deaths
27000
26500
26000
25500
25000
24500
24000
22-11
23-11
24-11
25-11
26-11
27-11
28-11
29-11
30-11
01-12
02-12
03-12
04-12
05-12
06-12
07-12
08-12
09-12
Date in 2021


Total Deaths of Viet Nam
Date in 2021
09-12
08-12
07-12
06-12
05-12
04-12
03-12
02-12
01-12
30-11
29-11
28-11
27-11
26-11
25-11
24-11
23-11
22-11
0
5000
10000
15000
20000
25000
Deaths

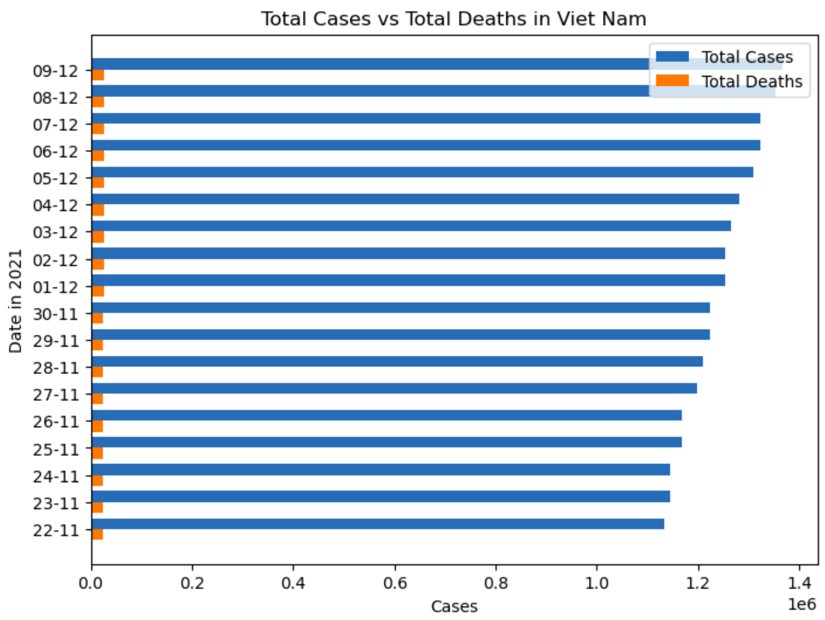

Total Recovered of Vietnam
1e6
Total Recovered
0.92
0.94
0.96
0.98
1.00
1.02
1.04
22-11 23-11 24-11 25-11 26-11 27-11 28-11 29-11 30-11 01-12 02-12 03-12 04-12 05-12 06-12 07-12 08-12 09-12
Date in 2021


Total Recovered Cases of Viet Nam
Date in 2021
09-12
08-12
07-12
06-12
05-12
04-12
03-12
02-12
01-12
30-11
29-11
28-11
27-11
26-11
25-11
24-11
23-11
22-11
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Recovered Cases

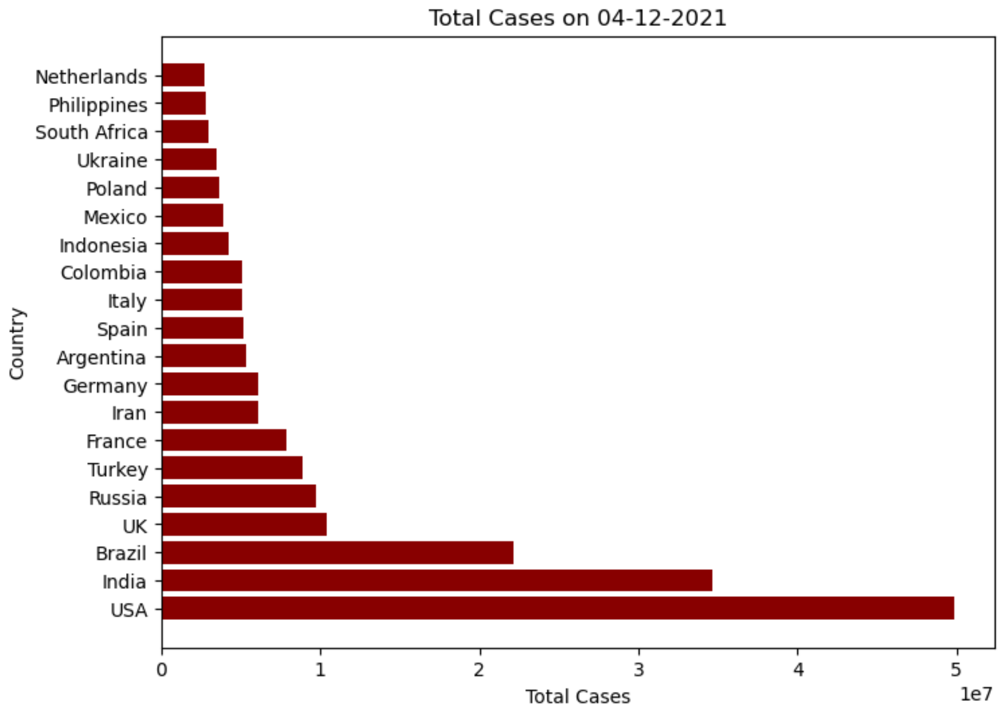


Above charts indicate how the trend of CoronaVirus happening in VietNam through 22-11-2021 until 09-12-2021.

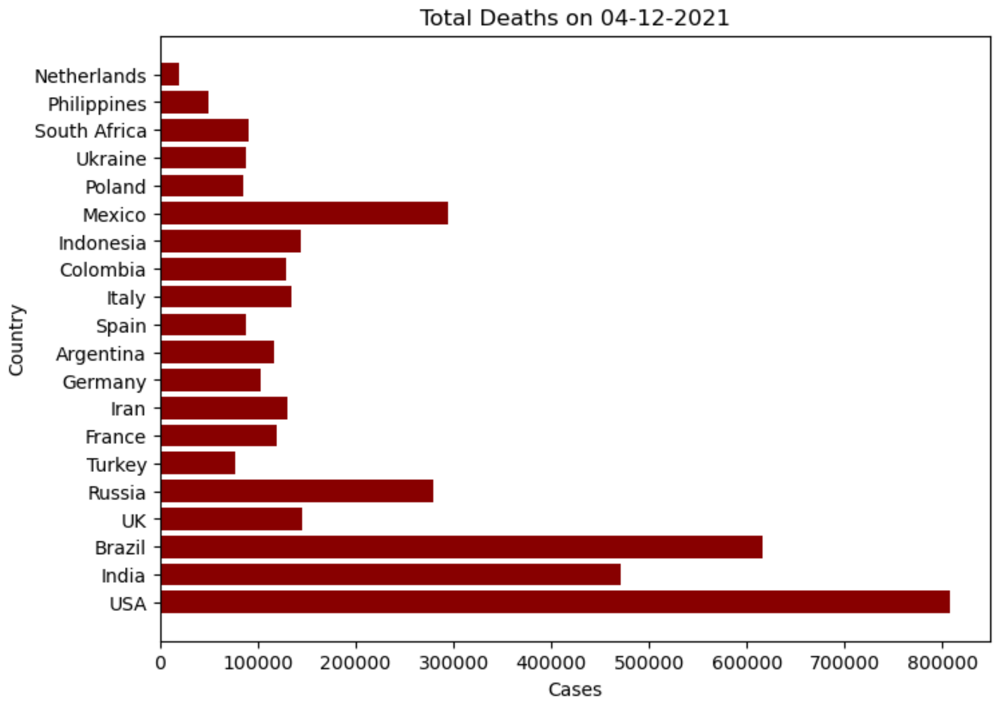
Total number of people infected by Virus is increasing cause total deaths also increase. But somehow, one positive thing is that the number of people recovering from the disease is increasing day by day.



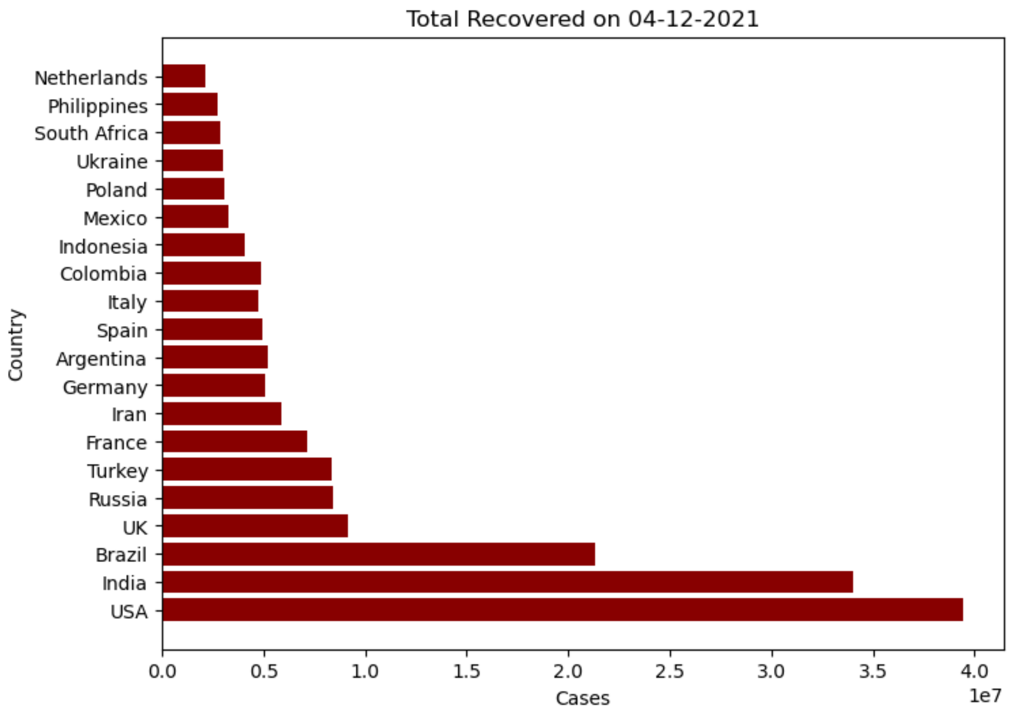
This chart shows a comparison between total inflected cases and total deaths in Viet Nam. “Total Deaths” occupies a small part of the whole. But let's not be subjective about this dangerous disease. An example is the terrible virus disease in India and the US.



Bar chart show top 20 country leading the number of Total Cases in the world



Bar chart show top 20 country leading the number of Total Deaths in the world



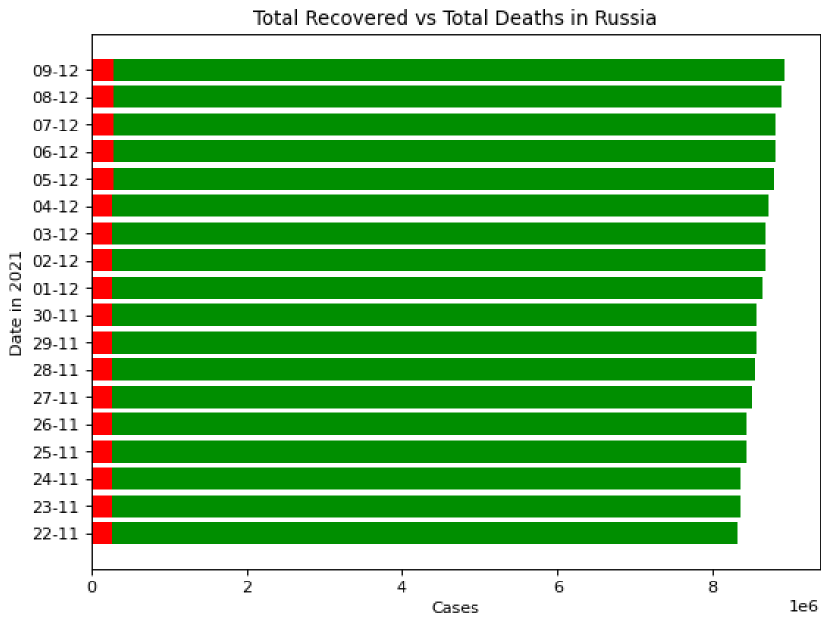
Bar chart show top 20 country leading the number of Total Recovered in the world

The top of the three charts above is the US. The charts show that the disease covers the whole US because of the subjectivity and contempt for the disease since it has not yet spreaded. The ranked 2nd and 3rd are poor and backward countries are Brazil and India. The disease also spread and took many lives.

c. Composition Plots

In this part, we use **Stacked Bar Chart** to show how each variable is divided into sub-categories and the proportion of the sub-category. We use “Total Recovered” and “Total Deaths” columns for visualizing because the two column meeting condition is a numeric attribute.

Data is collected in **Russia**.



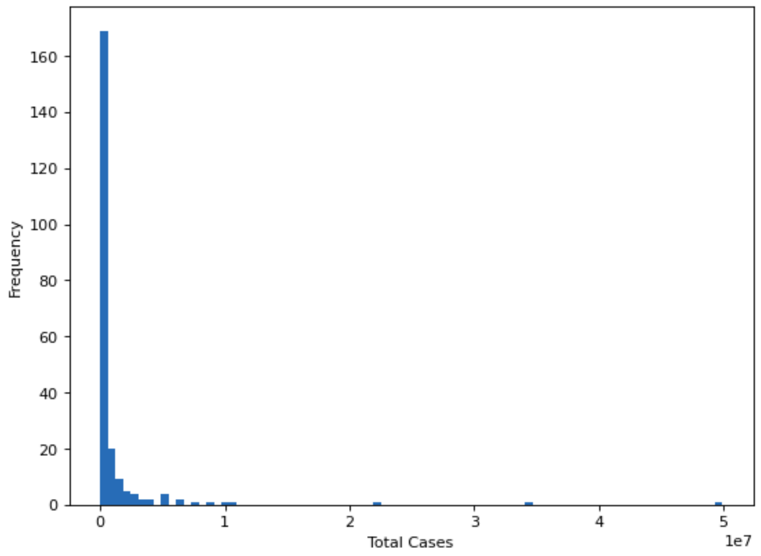
"Total Deaths" variable occupies less part than the "Total Recovered" variable. Russia is also a typical example like Vietnam.

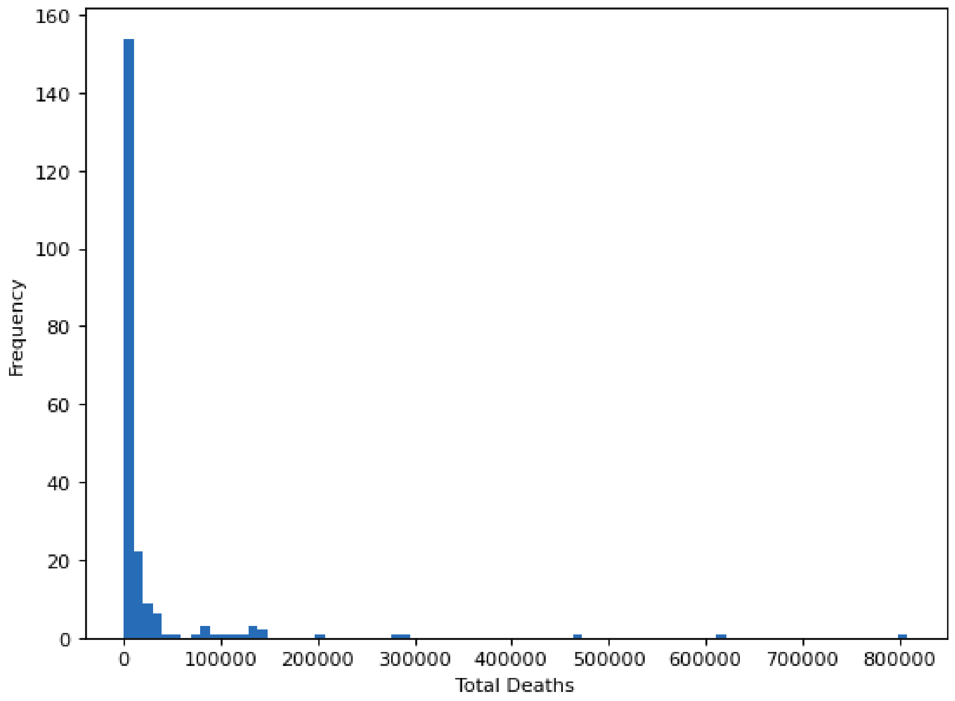
d. Distribution Plots

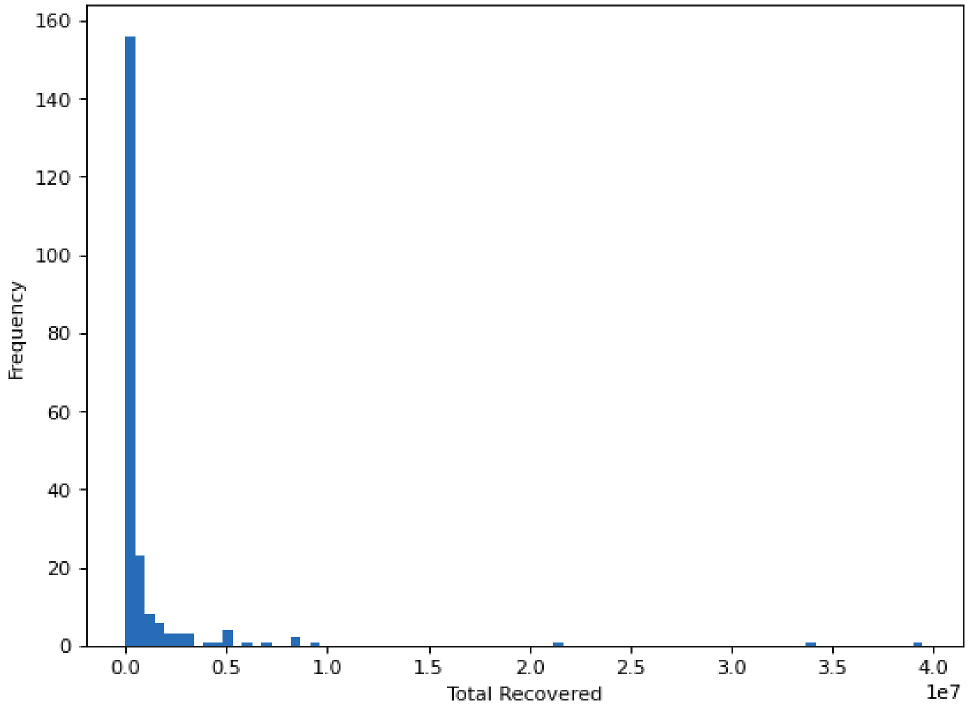
We use two types of Distribution Plots: **Histogram** and **Boxplot**.

We want to know more about the distribution of "Total Cases", "Total Deaths", "Total Recovered" and its frequency. Also, see some statistical measurements like: Median, Mean, Upper Quartile, Lower Quartile,...

Data we use in this example is on **04-12-2021** for **Histogram** and data we use for **Boxplot** is of **Russia.**

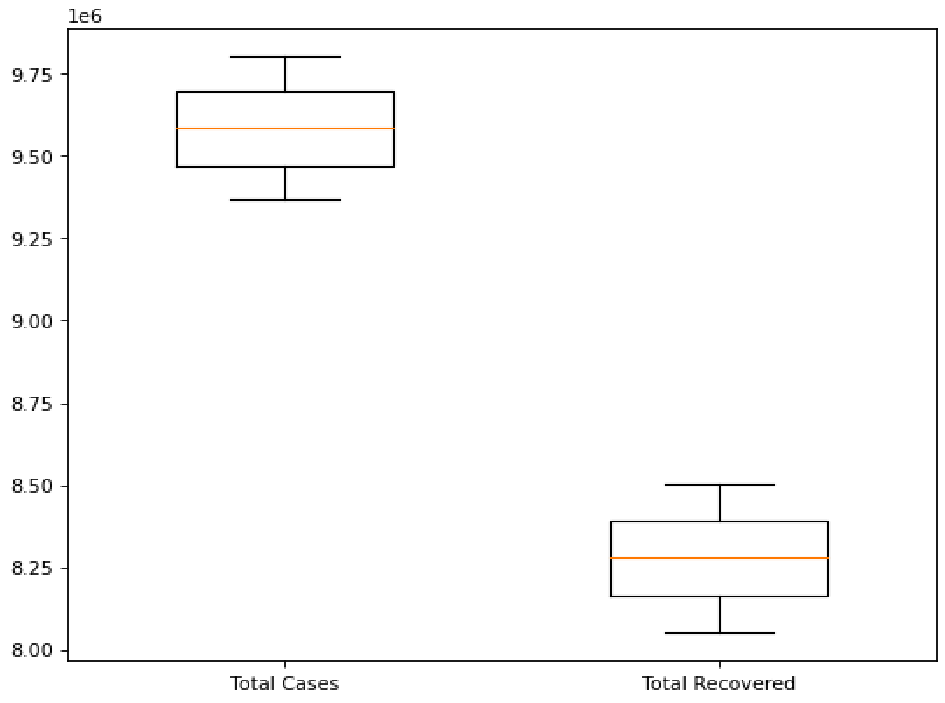


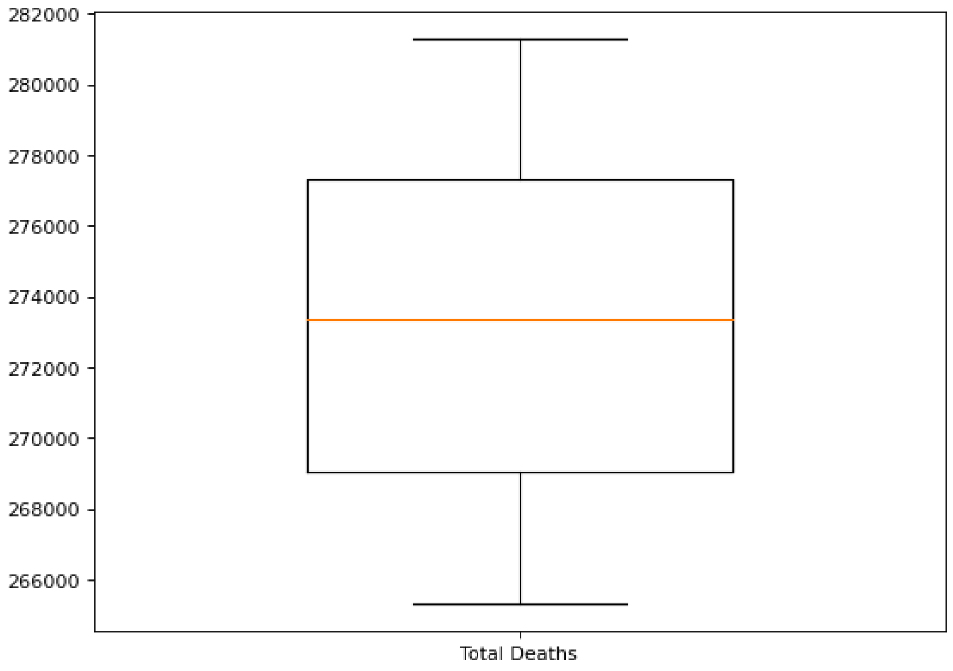




From the charts above, we can draw the conclusion that the distribution of the three attributes all revolve around the interval 0 and 500.000 cases. Little frequency in the middle, focused mainly on the left side of the chart.

There are more than 140 countries with at least 500,000 cases.





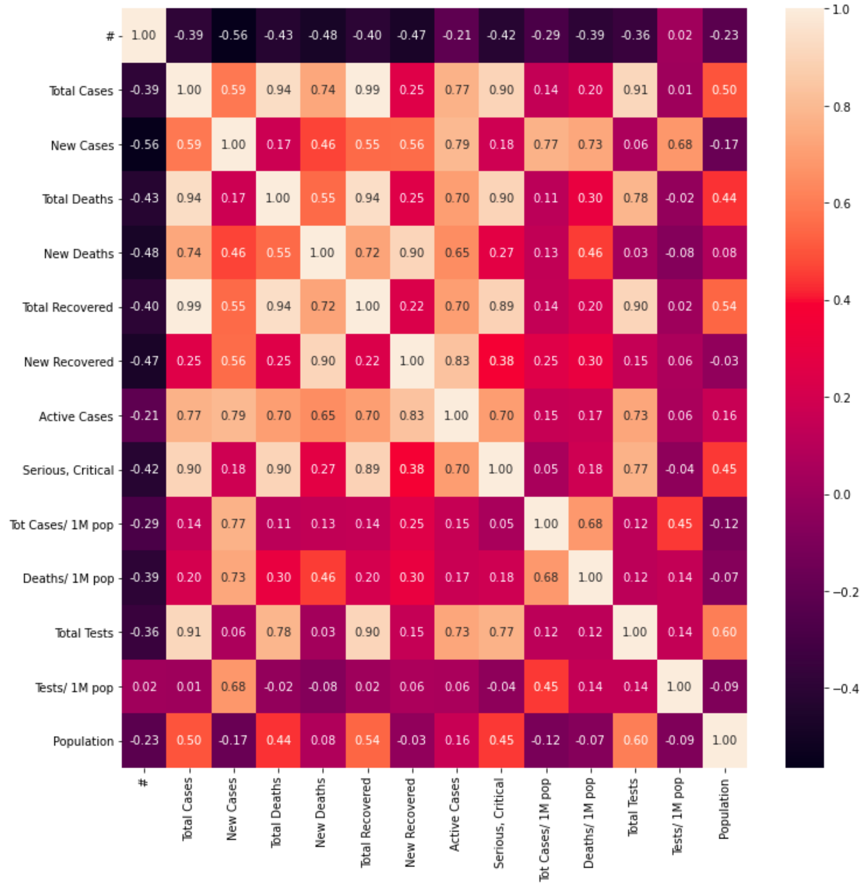
Three boxplots show some statistical measurements:

- "Total Cases" has a median around 950.000 and 975.000 cases mean that fifty-percentages of countries in that range. It shows that the whole world has a very high number of infections.

- "Total Recovered" has a median around 825.000 and 850.000 cases mean that fifty-percentages of countries in that range.
- "Total Deaths" has a median around 272.000 and 274.000 cases mean that fifty-percentages of countries in that range.

e. Relation Plots

First, we use the **Correlation Matrix** to find correlational relationships of whole attributes.



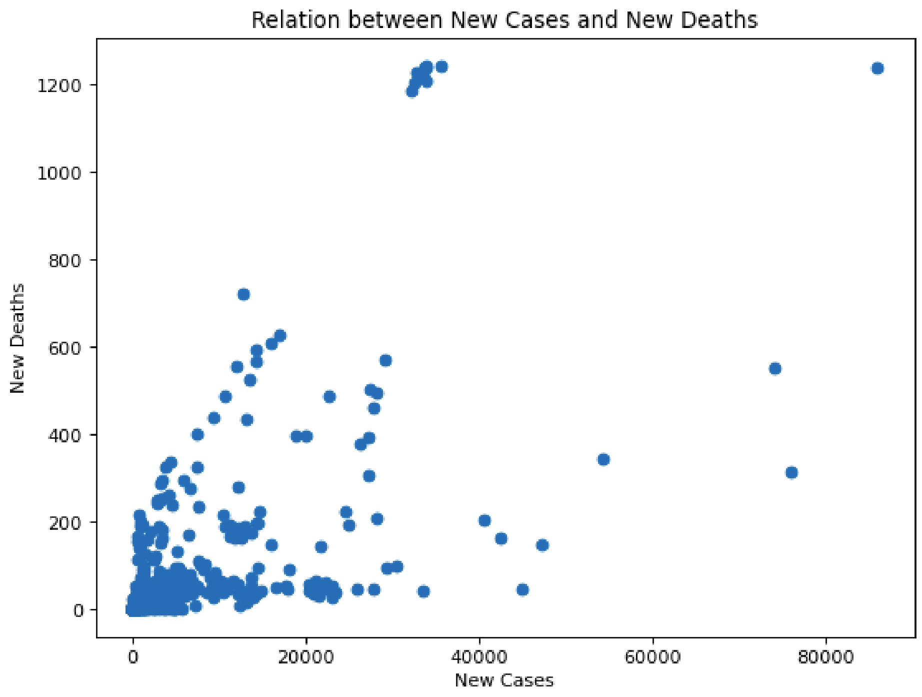
We must ignore all "Total" prefix columns because "Total" prefix columns only increment if we consider it, it makes no sense.

We can see that, "New Deaths" and "New Recovered" pair have the greatest correlational value of all. That means two columns have a relationship with each other.

In Relation Plots, we use **Scatter Charts** to find the relationship between two attributes of our data in order to find if the attributes covariate or inverse ?

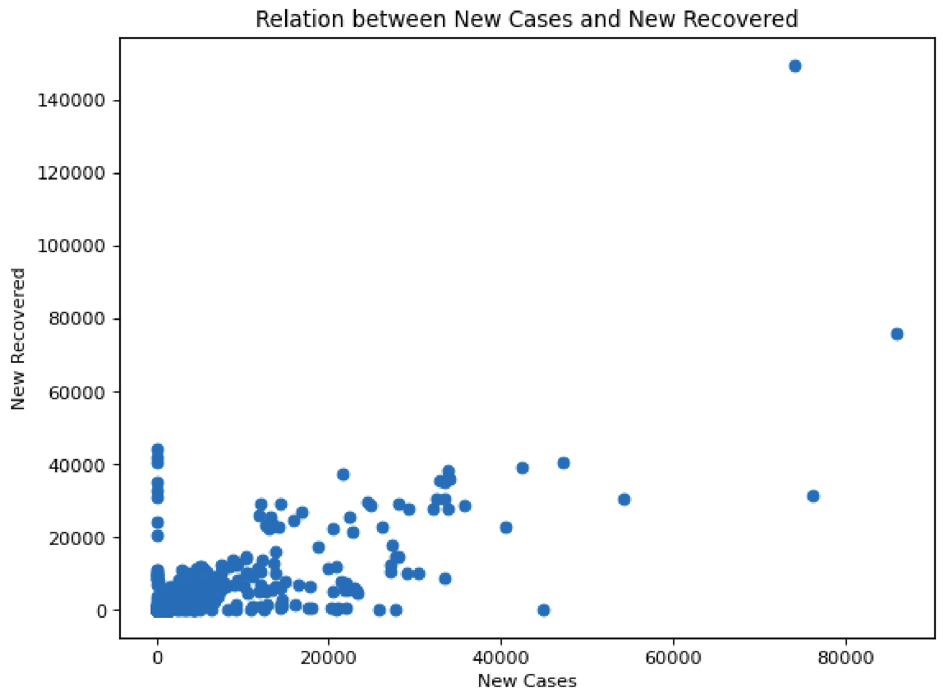
In this example, we find the relationship of some pair attributes: "New Cases" and "New

Deaths", "New Deaths" and "New Recovered", "New Cases" and "New Recovered". Data is collected from all countries in the world from 22-11-2021 until 09-12-2021.

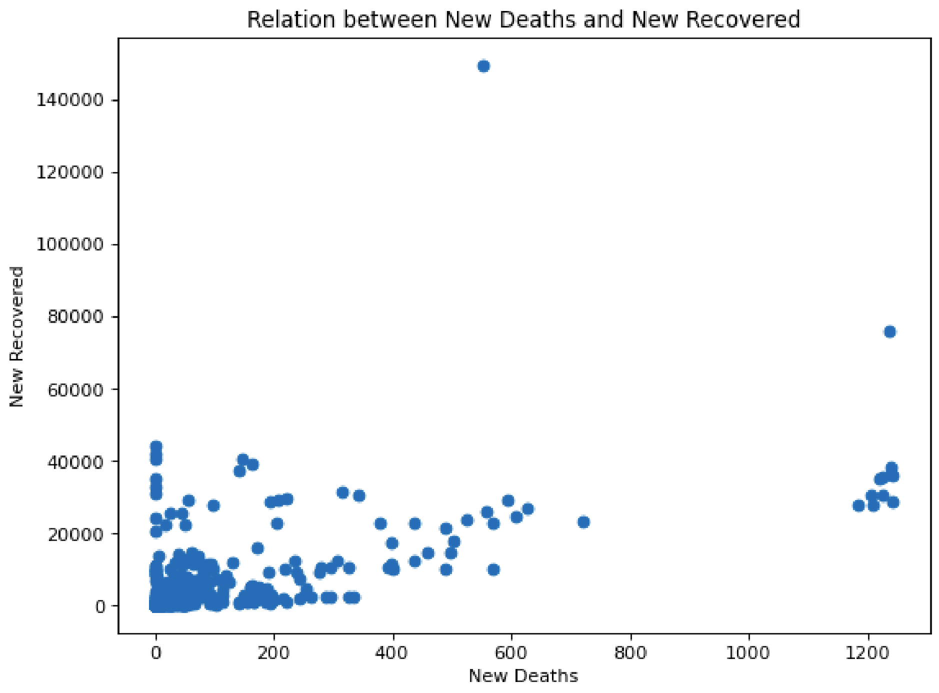


"New Cases" and "New Deaths" are not covariate. Some points of "New Deaths" increase but not "New Cases" and vice versa.

The number of points is most concentrated in the left region, showing countries with the number of deaths per day about 200 cases and the number of new cases per day about 20000 cases.



Looking at the above chart, we can easily see that "New Cases" and "New Recovered" may be covariates. Because when "New Recovered" increases, so do "New Cases". Data is plotted, maybe lying on a straight line.



And the same as "New Deaths" and "New Recovered".

The number of points is most concentrated in the left region, showing countries with the number of recovered cases per day about 20000 cases and the number of death cases per day about 200 cases. And there are some countries with very high death cases around 1200 a day.

4. References

https://airflow.apache.org/
https://www.applydatascience.com/airflow/airflow-tutorial-introduction/
https://devdocs.io/matplotlib~3.1/
Topic 04 - Visualization - Lecture document