FIT1043 Introduction to Data Science

Assignment 1

Juliet Chow, 32112602

12th August 2021

Introduction

This assignment is to analyse the selected data of Country-Vaccinations, 2020-GDP, and 2020-Population. After importing and reading the data, it is needed to wrangle each of the data. Then select the countries that is needed. Furthermore, merge all of the data into one dataframe that will be called final_data. Besides that, visualisation of the data will be covered in this assignment too along with the explanations of each graph.

Import Libraries

The first step is to import library **pandas**, which is an open source data analysis tool for python programming language. The purpose of the importing this library is that we would like to use the data structure such as DataFrame and it's associated functions such as reading from CSV files and so on.

In [1]:
```
import pandas as pd
```

After importing pandas library, we create an alias of pandas that is called pd for the purpose of easier programming later on

In [2]:
```
import numpy as np
```

After importing numpy library, we create an alias of numpy that is called np for the purpose of easier programming later on

In [3]:
```
pwd
```

Out[3]:
```
'/Users/Juliet/Documents/Monash Uni/FIT1043/Assignment/data'
```

Checking the location of the intended files, they should be in a folder called "data"

In [4]:
```
country_vaccinations = pd.read_csv("Country-Vaccinations.csv")
```

Store the data of Country-Vaccinations.csv into country_vaccinations variable

In [5]:
```
country_vaccinations
```

Out[5]:

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_va
0	Afghanistan	AFG	2021-02-22	0.0	0.0	
1	Afghanistan	AFG	2021-02-23	NaN	NaN	
2	Afghanistan	AFG	2021-02-24	NaN	NaN	
3	Afghanistan	AFG	2021-02-25	NaN	NaN	
4	Afghanistan	AFG	2021-02-26	NaN	NaN	
...	...	...	...	...	...	
36731	Zimbabwe	ZWE	2021-08-04	2604265.0	1740598.0	8
36732	Zimbabwe	ZWE	2021-08-05	2701095.0	1780541.0	9
36733	Zimbabwe	ZWE	2021-08-06	2784270.0	1817598.0	9
36734	Zimbabwe	ZWE	2021-08-07	2853668.0	1851407.0	1(
36735	Zimbabwe	ZWE	2021-08-08	2886822.0	1864204.0	1(

36736 rows × 15 columns

Read data of country_vaccinations

In [6]:
```
country_vaccinations.head()
```

Out[6]:

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccina
0	Afghanistan	AFG	2021-02-22	0.0	0.0	N
1	Afghanistan	AFG	2021-02-23	NaN	NaN	N
2	Afghanistan	AFG	2021-02-24	NaN	NaN	N
3	Afghanistan	AFG	2021-02-25	NaN	NaN	N
4	Afghanistan	AFG	2021-02-26	NaN	NaN	N

As we can see from country_vaccinations.head() and compare it to country_vaccinations output from the code above, we have outputted the correct file from index 0 - 4

In [7]:
```
country_vaccinations.tail()
```

Out[7]:

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vacc
36731	Zimbabwe	ZWE	2021-08-04	2604265.0	1740598.0	86
36732	Zimbabwe	ZWE	2021-08-05	2701095.0	1780541.0	92
36733	Zimbabwe	ZWE	2021-08-06	2784270.0	1817598.0	96
36734	Zimbabwe	ZWE	2021-08-07	2853668.0	1851407.0	10
36735	Zimbabwe	ZWE	2021-08-08	2886822.0	1864204.0	10

As we can see from country_vaccinations.tail() and compare it to country_vaccinations output from the code above, we have outputted the correct file from index 36731 - 36735

In [8]:
```
country_vaccinations["country"]
```

Out[8]:
```
0         Afghanistan
1         Afghanistan
2         Afghanistan
3         Afghanistan
4         Afghanistan
             ...
36731        Zimbabwe
36732        Zimbabwe
36733        Zimbabwe
36734        Zimbabwe
36735        Zimbabwe
Name: country, Length: 36736, dtype: object
```

From the last two code (country_vaccinations.head() and country*vaccinations.tail()), we can see that they have the same content of country from index 0 to 4 and from index 36731 to 36735 when we displayed the content of* "country"_ column in country_vaccinations file

In [9]:
```
country_vaccinations.describe()
```

Out[9]:

	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_ra
count	2.025000e+04	1.933000e+04	1.646700e+04	1.667100e+0
mean	1.508521e+07	6.729697e+06	4.007112e+06	2.451619e+0
std	8.573533e+07	2.432046e+07	1.390578e+07	1.315092e+0
min	0.000000e+00	0.000000e+00	1.000000e+00	0.000000e+0
25%	1.544285e+05	1.267028e+05	5.800000e+04	4.691000e+0
50%	1.078089e+06	7.674365e+05	4.475780e+05	2.271200e+0
75%	5.463368e+06	3.605988e+06	2.139242e+06	1.032145e+0
max	1.782525e+09	6.220000e+08	2.232990e+08	2.474100e+0

We can see basic statistics of country_vaccinations by using .describe() function

In [10]:
```
population = pd.read_csv("2020-Population.csv")
```

Store the data of 2020-Population.csv into population variable

In [11]:
```
population
```

Out[11]:

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unn
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	United Nations	NaN	NaN	NaN	NaN	NaN	NaN	
4	Population Division	NaN	NaN	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	
300	285	Estimates	Bermuda	14	60	Country/Area	918	
301	286	Estimates	Canada	NaN	124	Country/Area	918	
302	287	Estimates	Greenland	26	304	Country/Area	918	
303	288	Estimates	Saint Pierre and Miquelon	2	666	Country/Area	918	
304	289	Estimates	United States of America	35	840	Country/Area	918	15

305 rows × 78 columns

Read the data of population

In [12]:
```
population.head()
```

Out[12]:

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unname
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	United Nations	NaN	NaN	NaN	NaN	NaN	NaN	Na
4	Population Division	NaN	NaN	NaN	NaN	NaN	NaN	Na

5 rows × 78 columns

As we can see from population.head() and compare it to population output from the code above, we have outputted the correct file from index 0 - 4

In [13]:
```
population.tail()
```

Out[13]:

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unn
300	285	Estimates	Bermuda	14	60	Country/Area	918	
301	286	Estimates	Canada	NaN	124	Country/Area	918	
302	287	Estimates	Greenland	26	304	Country/Area	918	
303	288	Estimates	Saint Pierre and Miquelon	2	666	Country/Area	918	
304	289	Estimates	United States of America	35	840	Country/Area	918	15

5 rows × 78 columns

As we can see from population.tail() and compare it to population output from the code above, we have outputted the correct file from index 300 to 304

In [14]:
```
population.loc[300]
```

Out[14]:
```
Unnamed: 0           285
Unnamed: 1     Estimates
Unnamed: 2       Bermuda
Unnamed: 3            14
Unnamed: 4            60
                   ...
Unnamed: 73           63
Unnamed: 74           63
Unnamed: 75           63
Unnamed: 76           63
Unnamed: 77           62
Name: 300, Length: 78, dtype: object
```

When we select the row of index 300, we outputted the result as above. We can also see from population.tail() the content in row of index 300 they have the same result outputted

In [15]:
```
population.describe()
```

Out[15]:

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	U
count	299	290	290	83	290	290	290	
unique	299	2	290	52	290	11	39	
top	193	Estimates	Africa	2	660	Country/Area	915	
freq	1	289	1	11	1	235	28	

4 rows × 78 columns

We can see basic statistics of population by using .describe() function

In [16]:
```
gross_dp = pd.read_csv("2020-GDP.csv")
```

Store the data of 2020-GDP.csv into gross_dp variable

In [17]:
```
gross_dp
```

Out[17]:

	Unnamed: 0	Gross domestic product 2020	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5
0	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	(millions of	NaN
2	NaN	Ranking	NaN	Economy	US dollars)	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN
4	USA	1	NaN	United States	20,936,600	NaN
...	...	...	...	...	...	...
324	NaN	NaN	NaN	NaN	NaN	NaN
325	NaN	NaN	NaN	NaN	NaN	NaN
326	NaN	NaN	NaN	NaN	NaN	NaN
327	NaN	NaN	NaN	NaN	NaN	NaN
328	NaN	NaN	NaN	NaN	NaN	NaN

329 rows × 6 columns

In [18]:
```
gross_dp.head()
```

Out[18]:

	Unnamed: 0	Gross domestic product 2020	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5
0	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	(millions of	NaN
2	NaN	Ranking	NaN	Economy	US dollars)	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN
4	USA	1	NaN	United States	20,936,600	NaN

As we can see from gross_dp.head() and compare it to gross_dp output from the code above, we have outputted the correct file from index 0 - 4

In [19]:
```
gross_dp.tail()
```

Out[19]:

	Unnamed: 0	Gross domestic product 2020	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5
324	NaN	NaN	NaN	NaN	NaN	NaN
325	NaN	NaN	NaN	NaN	NaN	NaN
326	NaN	NaN	NaN	NaN	NaN	NaN
327	NaN	NaN	NaN	NaN	NaN	NaN
328	NaN	NaN	NaN	NaN	NaN	NaN

As we can see from gross_dp.tail() and compare it to gross_dp output from the code above, we have outputted the correct file from index 324 to 328

In [20]:
```
gross_dp.loc[4]
```

Out[20]:
```
Unnamed: 0                          USA
Gross domestic product 2020           1
Unnamed: 2                          NaN
Unnamed: 3                United States
Unnamed: 4                  20,936,600
Unnamed: 5                          NaN
Name: 4, dtype: object
```

In row 4 of gross_dp data we can see the content from the code above. Both gross_dp.loc[4] and gross_dp in index 4, we can see we have outputted the corect data of 2020-GDP.csv

In [21]:
```
gross_dp.describe()
```

Out[21]:

	Unnamed: 2
count	0.0
mean	NaN
std	NaN
min	NaN
25%	NaN
50%	NaN
75%	NaN
max	NaN

We can see basic statistics of gross_dp by using .describe() function

Subsetting country_vaccinations data

In [22]:
```
fun = {'vaccines':'min','people_fully_vaccinated':'sum','daily_vaccinations
```

Use fun variable for the purpose of aggregation. Include the data that is needed for subsetting the file of country_vaccinations

In [23]:
```
country_vaccinations_subset = country_vaccinations.groupby("country").agg(f
```

Store the aggregation of fun by grouping it based on the country in country_vaccinations into country_vaccinations_subset

In [24]:
```
country_vaccinations_subset
```

Out[24]:

country	vaccines	people_fully_vaccinated	daily_vaccinations
Afghanistan	Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi...	2368962.0	1649463.0
Albania	Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac, ...	25325342.0	1237306.0
Algeria	Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac...	724812.0	4075025.0
Andorra	Oxford/AstraZeneca, Pfizer/BioNTech	179434.0	76689.0
Angola	Oxford/AstraZeneca	6795439.0	1690469.0
...	...	...	...
Wales	Moderna, Oxford/AstraZeneca, Pfizer/BioNTech	174473615.0	4396339.0
Wallis and Futuna	Oxford/AstraZeneca	55937.0	8937.0
Yemen	Oxford/AstraZeneca	21899.0	302943.0
Zambia	Oxford/AstraZeneca, Sinopharm/Beijing	3197289.0	475566.0
Zimbabwe	Sinopharm/Beijing, Sinovac, Sputnik V	47164502.0	2693298.0

222 rows × 3 columns

Display country_vaccinations_subset

In [25]:
```
country_vaccinations_subset.reset_index(inplace = True)
```

Reset the index of country_vaccinations_subset

In [26]:
```
country_vaccinations_subset
```

Out[26]:

	country	vaccines	people_fully_vaccinated	daily_vaccinations
0	Afghanistan	Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi...	2368962.0	1649463.0
1	Albania	Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac, ...	25325342.0	1237306.0
2	Algeria	Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac...	724812.0	4075025.0
3	Andorra	Oxford/AstraZeneca, Pfizer/BioNTech	179434.0	76689.0
4	Angola	Oxford/AstraZeneca	6795439.0	1690469.0
...	...	...	...	...
217	Wales	Moderna, Oxford/AstraZeneca, Pfizer/BioNTech	174473615.0	4396339.0
218	Wallis and Futuna	Oxford/AstraZeneca	55937.0	8937.0
219	Yemen	Oxford/AstraZeneca	21899.0	302943.0
220	Zambia	Oxford/AstraZeneca, Sinopharm/Beijing	3197289.0	475566.0
221	Zimbabwe	Sinopharm/Beijing, Sinovac, Sputnik V	47164502.0	2693298.0

222 rows × 4 columns

Display country_vaccinations_subset after resetting the index

In [27]:
```
country_vaccinations_subset.rename(columns = {"daily_vaccinations":"total_v
```

Rename daily_vaccinations into total_vaccinations because daily_vaccinations values shows total_vaccinations values

In [28]:
```
country_vaccinations_subset
```

Out[28]:

	country	vaccines	people_fully_vaccinated	total_vaccinations
0	Afghanistan	Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi...	2368962.0	1649463.0
1	Albania	Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac, ...	25325342.0	1237306.0
2	Algeria	Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac...	724812.0	4075025.0
3	Andorra	Oxford/AstraZeneca, Pfizer/BioNTech	179434.0	76689.0
4	Angola	Oxford/AstraZeneca	6795439.0	1690469.0
...	...	...	...	...
217	Wales	Moderna, Oxford/AstraZeneca, Pfizer/BioNTech	174473615.0	4396339.0
218	Wallis and Futuna	Oxford/AstraZeneca	55937.0	8937.0
219	Yemen	Oxford/AstraZeneca	21899.0	302943.0
220	Zambia	Oxford/AstraZeneca, Sinopharm/Beijing	3197289.0	475566.0
221	Zimbabwe	Sinopharm/Beijing, Sinovac, Sputnik V	47164502.0	2693298.0

222 rows × 4 columns

country_vaccinations have been successfully subsetted into country_vaccinations_subset

Subsetting 2020-Population data

In [29]:

```
population
```

Out[29]:

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unn
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	United Nations	NaN	NaN	NaN	NaN	NaN	NaN	
4	Population Division	NaN	NaN	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	
300	285	Estimates	Bermuda	14	60	Country/Area	918	
301	286	Estimates	Canada	NaN	124	Country/Area	918	
302	287	Estimates	Greenland	26	304	Country/Area	918	
303	288	Estimates	Saint Pierre and Miquelon	2	666	Country/Area	918	
304	289	Estimates	United States of America	35	840	Country/Area	918	15

305 rows × 78 columns

Display population

In [30]:
```
temp_population = population
```

Store population into temp_population so that population will not be changed in any form

In [31]:
```
temp_population.dropna(thresh=5, inplace= True)
```

Because there is an obvious NaN values in the first 5 rows, it needs to be deleted. I assumed rows that have more than 5 NaN values will be deleted from the dataframe. Since, logically if there is more than 5 NaN values then it is a unnecessary row present in the data

In [32]:
```
temp_population
```

Out[32]:

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6
15	Index	Variant	Region, subregion, country or area *	Notes	Country code	Type	Parent code
16	1	Estimates	WORLD	NaN	900	World	0
17	2	Estimates	UN development groups	a	1803	Label/Separator	900
18	3	Estimates	More developed regions	b	901	Development Group	1803
19	4	Estimates	Less developed regions	c	902	Development Group	1803
...	...	...	...	...	...	...	...
300	285	Estimates	Bermuda	14	60	Country/Area	918
301	286	Estimates	Canada	NaN	124	Country/Area	918
302	287	Estimates	Greenland	26	304	Country/Area	918
303	288	Estimates	Saint Pierre and Miquelon	2	666	Country/Area	918
304	289	Estimates	United States of America	35	840	Country/Area	918

290 rows × 78 columns

The unnecessary rows have been deleted

In [33]:
```
temp_population.rename(columns=temp_population.iloc[0], inplace = True)
```

Make the first row into the heading title of each column

In [34]:
```
temp_population
```

Out[34]:

	Index	Variant	Region, subregion, country or area *	Notes	Country code	Type	Parent code	1950	1951	19
15	Index	Variant	Region, subregion, country or area *	Notes	Country code	Type	Parent code	1950	1951	19
16	1	Estimates	WORLD	NaN	900	World	0	2 536 431	2 584 034	6 8
17	2	Estimates	UN development groups	a	1803	Label/Separator	900	...	...	
18	3	Estimates	More developed regions	b	901	Development Group	1803	814 819	824 004	8 7
19	4	Estimates	Less developed regions	c	902	Development Group	1803	1 721 612	1 760 031	1 7
...	...	...	...	...	...	...	...	...	...	
300	285	Estimates	Bermuda	14	60	Country/Area	918	37	38	
301	286	Estimates	Canada	NaN	124	Country/Area	918	13 733	14 078	4
302	287	Estimates	Greenland	26	304	Country/Area	918	23	23	
303	288	Estimates	Saint Pierre and Miquelon	2	666	Country/Area	918	5	5	
304	289	Estimates	United States of America	35	840	Country/Area	918	158 804	160 872	1 2

290 rows × 78 columns

Display the result after the heading title have been created

In [35]:
```
temp_population.drop(15, inplace = True)
```

Delete the row with the number of 15 because we have already used it as the heading title

In [36]:
```
temp_population
```

Out[36]:

	Index	Variant	Region, subregion, country or area *	Notes	Country code	Type	Parent code	1950	1951	19
16	1	Estimates	WORLD	NaN	900	World	0	2 536 431	2 584 034	6 8
17	2	Estimates	UN development groups	a	1803	Label/Separator	900	...	...	
18	3	Estimates	More developed regions	b	901	Development Group	1803	814 819	824 004	8 7
19	4	Estimates	Less developed regions	c	902	Development Group	1803	1 721 612	1 760 031	1 7
20	5	Estimates	Least developed countries	d	941	Development Group	902	195 428	199 180	2
...	...	...	...	...	...	...	...	...	...	
300	285	Estimates	Bermuda	14	60	Country/Area	918	37	38	
301	286	Estimates	Canada	NaN	124	Country/Area	918	13 733	14 078	4
302	287	Estimates	Greenland	26	304	Country/Area	918	23	23	
303	288	Estimates	Saint Pierre and Miquelon	2	666	Country/Area	918	5	5	
304	289	Estimates	United States of America	35	840	Country/Area	918	158 804	160 872	1 2

289 rows × 78 columns

Display the temp_population

In [37]:

```
for i in range (1950,2021):
    temp_population[str(i)] = temp_population[str(i)].str.replace(' ','')
    temp_population[str(i)] = pd.to_numeric(temp_population[str(i)], errors
    temp_population[str(i)] = temp_population[str(i)].astype("Int64")
```

Because the type of the values in row 1950 - 2020 are all strings. On the other hand, I need those values to calculate the population. I need to change the type of those values into integers

In [38]:

```
temp_population["population (in thousands)"] = temp_population.iloc[:,8:].s
```

Then calculate the populations by adding the values from the row of index 8 until the end of the row

In [39]:
```
temp_population
```

Out[39]:

	Index	Variant	Region, subregion, country or area *	Notes	Country code	Type	Parent code	1950	19
16	1	Estimates	WORLD	NaN	900	World	0	2536431	2584(
17	2	Estimates	UN development groups	a	1803	Label/Separator	900	<NA>	<N
18	3	Estimates	More developed regions	b	901	Development Group	1803	814819	824(
19	4	Estimates	Less developed regions	c	902	Development Group	1803	1721612	1760
20	5	Estimates	Least developed countries	d	941	Development Group	902	195428	199
...	...	...	...	...	...	...	...	...	
300	285	Estimates	Bermuda	14	60	Country/Area	918	37	
301	286	Estimates	Canada	NaN	124	Country/Area	918	13733	14(
302	287	Estimates	Greenland	26	304	Country/Area	918	23	
303	288	Estimates	Saint Pierre and Miquelon	2	666	Country/Area	918	5	
304	289	Estimates	United States of America	35	840	Country/Area	918	158804	160

289 rows × 79 columns

Display the result after calculating population (in thousands)

In [40]:
```
population_subset = temp_population.loc[:,["Region, subregion, country or a
```

We choose Region, subregion, country or area * and population (in thousands) columns and store it into population_subset

In [41]:
```
population_subset.rename(columns = {"Region, subregion, country or area *":
```

Rename Region, subregion, country or area * into country

In [42]:
```
population_subset
```

Out[42]:

	country	population (in thousands)
16	WORLD	350140691.0
17	UN development groups	0.0
18	More developed regions	76493086.0
19	Less developed regions	273647600.0
20	Least developed countries	35912614.0
...	...	...
300	Bermuda	3990.0
301	Canada	1827583.0
302	Greenland	3377.0
303	Saint Pierre and Miquelon	401.0
304	United States of America	17213544.0

289 rows × 2 columns

Display population_subset

In [43]:
```
population_subset.reset_index(inplace = True)
```

Reset the index of population_subset

In [44]:
```
population_subset
```

Out[44]:

	index	country	population (in thousands)
0	16	WORLD	350140691.0
1	17	UN development groups	0.0
2	18	More developed regions	76493086.0
3	19	Less developed regions	273647600.0
4	20	Least developed countries	35912614.0
...	...	...	...
284	300	Bermuda	3990.0
285	301	Canada	1827583.0
286	302	Greenland	3377.0
287	303	Saint Pierre and Miquelon	401.0
288	304	United States of America	17213544.0

289 rows × 3 columns

Display population_subset once more

In [45]:
```
population_subset.drop("index", axis = 1, inplace=True)
```

We do not need the index column. Thus, we need to delete it

In [46]:
```
population_subset
```

Out[46]:

	country	population (in thousands)
0	WORLD	350140691.0
1	UN development groups	0.0
2	More developed regions	76493086.0
3	Less developed regions	273647600.0
4	Least developed countries	35912614.0
...	...	...
284	Bermuda	3990.0
285	Canada	1827583.0
286	Greenland	3377.0
287	Saint Pierre and Miquelon	401.0
288	United States of America	17213544.0

289 rows × 2 columns

population have been successfully subsetted into population_subset

Subsetting 2020-GDP data

In [47]:
```
gross_dp
```

Out[47]:

	Unnamed: 0	Gross domestic product 2020	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5
0	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	(millions of	NaN
2	NaN	Ranking	NaN	Economy	US dollars)	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN
4	USA	1	NaN	United States	20,936,600	NaN
...	...	...	...	...	...	...
324	NaN	NaN	NaN	NaN	NaN	NaN
325	NaN	NaN	NaN	NaN	NaN	NaN
326	NaN	NaN	NaN	NaN	NaN	NaN
327	NaN	NaN	NaN	NaN	NaN	NaN
328	NaN	NaN	NaN	NaN	NaN	NaN

329 rows × 6 columns

Display gross_dp

In [48]:
```
temp_gross_dp = gross_dp
```

Store gross_dp into temp_gross_dp so that gross_dp will not be changed in any form

In [49]:
```
temp_gross_dp.dropna(thresh = 2, inplace = True)
```

Rows that have more than 2 NaN values will be deleted

In [50]:
```
temp_gross_dp
```

Out[50]:

	Unnamed: 0	Gross domestic product 2020	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5
2	NaN	Ranking	NaN	Economy	US dollars)	NaN
4	USA	1	NaN	United States	20,936,600	NaN
5	CHN	2	NaN	China	14,722,731	NaN
6	JPN	3	NaN	Japan	5,064,873	NaN
7	DEU	4	NaN	Germany	3,806,060	NaN
...	...	...	...	...	...	...
233	SSF	NaN	NaN	Sub-Saharan Africa	1,685,632	NaN
234	LIC	NaN	NaN	Low income	546,661	NaN
235	LMC	NaN	NaN	Lower middle income	7,328,774	NaN
236	UMC	NaN	NaN	Upper middle income	23,630,843	NaN
237	HIC	NaN	NaN	High income	53,255,167	NaN

230 rows × 6 columns

Display temp_gross_dp after the unnecessary rows have been deleted

In [51]:
```
temp_gross_dp.rename(columns=temp_gross_dp.iloc[0], inplace = True)
```

Make the first row into the heading title of each column

In [52]:
```
temp_gross_dp
```

Out[52]:

	NaN	Ranking	NaN	Economy	US dollars)	NaN
2	NaN	Ranking	NaN	Economy	US dollars)	NaN
4	USA	1	NaN	United States	20,936,600	NaN
5	CHN	2	NaN	China	14,722,731	NaN
6	JPN	3	NaN	Japan	5,064,873	NaN
7	DEU	4	NaN	Germany	3,806,060	NaN
...	...	...	...	...	...	...
233	SSF	NaN	NaN	Sub-Saharan Africa	1,685,632	NaN
234	LIC	NaN	NaN	Low income	546,661	NaN
235	LMC	NaN	NaN	Lower middle income	7,328,774	NaN
236	UMC	NaN	NaN	Upper middle income	23,630,843	NaN
237	HIC	NaN	NaN	High income	53,255,167	NaN

230 rows × 6 columns

Display the temp_gross_dp after creating the heading title

In [53]:
```
temp_gross_dp.drop(2, inplace = True)
```

Delete the row 2 because we do not need it anymore

In [54]:
```
temp_gross_dp
```

Out[54]:

	NaN	Ranking	NaN	Economy	US dollars)	NaN
4	USA	1	NaN	United States	20,936,600	NaN
5	CHN	2	NaN	China	14,722,731	NaN
6	JPN	3	NaN	Japan	5,064,873	NaN
7	DEU	4	NaN	Germany	3,806,060	NaN
8	GBR	5	NaN	United Kingdom	2,707,744	NaN
...	...	...	...	...	...	...
233	SSF	NaN	NaN	Sub-Saharan Africa	1,685,632	NaN
234	LIC	NaN	NaN	Low income	546,661	NaN
235	LMC	NaN	NaN	Lower middle income	7,328,774	NaN
236	UMC	NaN	NaN	Upper middle income	23,630,843	NaN
237	HIC	NaN	NaN	High income	53,255,167	NaN

229 rows × 6 columns

In [55]:
```
temp_gross_dp.dropna(axis='columns', thresh=9, inplace = True)
```

Delete the columns with more than 9 NaN values

In [56]:
```
temp_gross_dp
```

Out[56]:

	NaN	Ranking	Economy	US dollars)
4	USA	1	United States	20,936,600
5	CHN	2	China	14,722,731
6	JPN	3	Japan	5,064,873
7	DEU	4	Germany	3,806,060
8	GBR	5	United Kingdom	2,707,744
...	...	...	...	...
233	SSF	NaN	Sub-Saharan Africa	1,685,632
234	LIC	NaN	Low income	546,661
235	LMC	NaN	Lower middle income	7,328,774
236	UMC	NaN	Upper middle income	23,630,843
237	HIC	NaN	High income	53,255,167

229 rows × 4 columns

In [57]:
```
temp_gross_dp.set_axis(['Country Code', 'Ranking','country','GDP (in millic
```

Rename the title of each column

In [58]:
```
temp_gross_dp
```

Out[58]:

	Country Code	Ranking	country	GDP (in millions)
4	USA	1	United States	20,936,600
5	CHN	2	China	14,722,731
6	JPN	3	Japan	5,064,873
7	DEU	4	Germany	3,806,060
8	GBR	5	United Kingdom	2,707,744
...	...	...	...	...
233	SSF	NaN	Sub-Saharan Africa	1,685,632
234	LIC	NaN	Low income	546,661
235	LMC	NaN	Lower middle income	7,328,774
236	UMC	NaN	Upper middle income	23,630,843
237	HIC	NaN	High income	53,255,167

229 rows × 4 columns

In [59]:
```
gdp_subset = temp_gross_dp.loc[:,["country","GDP (in millions)"]]
```

We only need country and GDP (in millions) for the data subset of GDP

In [60]:
```
gdp_subset
```

Out[60]:

	country	GDP (in millions)
4	United States	20,936,600
5	China	14,722,731
6	Japan	5,064,873
7	Germany	3,806,060
8	United Kingdom	2,707,744
...	...	...
233	Sub-Saharan Africa	1,685,632
234	Low income	546,661
235	Lower middle income	7,328,774
236	Upper middle income	23,630,843
237	High income	53,255,167

229 rows × 2 columns

In [61]:
```
gdp_subset.reset_index(inplace = True)
```

Reset the index of gdp_subset

In [62]:
```
gdp_subset
```

Out[62]:

	index	country	GDP (in millions)
0	4	United States	20,936,600
1	5	China	14,722,731
2	6	Japan	5,064,873
3	7	Germany	3,806,060
4	8	United Kingdom	2,707,744
...	...	...	...
224	233	Sub-Saharan Africa	1,685,632
225	234	Low income	546,661
226	235	Lower middle income	7,328,774
227	236	Upper middle income	23,630,843
228	237	High income	53,255,167

229 rows × 3 columns

Because GDP (in millions) is still in string we need to change it into integer

In [63]:
```
gdp_subset.drop("index", axis = 1, inplace=True)
```

Delete index column because we do not need it anymore

In [64]:
```
gdp_subset
```

Out[64]:

	country	GDP (in millions)
0	United States	20,936,600
1	China	14,722,731
2	Japan	5,064,873
3	Germany	3,806,060
4	United Kingdom	2,707,744
...	...	...
224	Sub-Saharan Africa	1,685,632
225	Low income	546,661
226	Lower middle income	7,328,774
227	Upper middle income	23,630,843
228	High income	53,255,167

229 rows × 2 columns

gross_dp have been successfully subsetted into gdp_subset

Choosing the data of the selected countries

In [65]:
```
selected_countries = ['Indonesia', 'Malaysia', 'Singapore', 'Thailand', 'Ph
```

Store the selected_countries into list

In [66]:
```
country_vaccinations_subset = country_vaccinations_subset[country_vaccinati
```

Choose the data of the selected countries by using .isin function and store it into country_vaccinations_subset

In [67]:
```
country_vaccinations_subset
```

Out[67]:

	country	vaccines	people_fully_vaccinated	total_vaccinations
10	Australia	Oxford/AstraZeneca, Pfizer/BioNTech	1.424238e+08	13222783.0
91	Indonesia	Moderna, Oxford/AstraZeneca, Sinopharm/Beijing...	1.416955e+09	72386296.0
120	Malaysia	Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac	2.924130e+08	23687251.0
156	Philippines	Johnson&Johnson, Moderna, Oxford/AstraZeneca, ...	1.749103e+08	23230492.0
177	Singapore	Moderna, Pfizer/BioNTech	1.047836e+08	7911869.0
196	Thailand	Oxford/AstraZeneca, Sinovac	2.075185e+08	18349011.0

In [68]:

```
population_subset = population_subset[population_subset['country'].isin(sel
```

Choose the data of the selected countries by using .isin function and store it into population_subset

In [69]:

```
population_subset
```

Out[69]:

	country	population (in thousands)
138	Indonesia	11654222.0
140	Malaysia	1223634.0
142	Philippines	4115063.0
143	Singapore	218702.0
144	Thailand	3460815.0
202	Australia	1131477.0

In [70]:

```
gdp_subset = gdp_subset[gdp_subset['country'].isin(selected_countries)]
```

Choose the data of the selected countries by using .isin function and store it into gdp_subset

In [71]:

```
gdp_subset
```

Out[71]:

	country	GDP (in millions)
12	Australia	1,330,901
15	Indonesia	1,058,424
23	Thailand	501,795
32	Philippines	361,489
35	Singapore	339,998
36	Malaysia	336,664

I choose to use list as the data structure to store the selected countries because the function that I used which is .isin() only accepts list as the argument. Furthermore, .isin() function is the fastest way that I knew in order to immediately choose the selected countries

Merge all the subsets into one dataframe

In [72]:
```
final_data = pd.merge((pd.merge(country_vaccinations_subset, population_sub
```

Merge all data subsets into final_data

In [73]:
```
final_data
```

Out[73]:

	country	vaccines	people_fully_vaccinated	total_vaccinations	population (in thousands)	
0	Australia	Oxford/AstraZeneca, Pfizer/BioNTech	1.424238e+08	13222783.0	1131477.0	1
1	Indonesia	Moderna, Oxford/AstraZeneca, Sinopharm/Beijing...	1.416955e+09	72386296.0	11654222.0	1
2	Malaysia	Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac	2.924130e+08	23687251.0	1223634.0	
3	Philippines	Johnson&Johnson, Moderna, Oxford/AstraZeneca, ...	1.749103e+08	23230492.0	4115063.0	
4	Singapore	Moderna, Pfizer/BioNTech	1.047836e+08	7911869.0	218702.0	
5	Thailand	Oxford/AstraZeneca, Sinovac	2.075185e+08	18349011.0	3460815.0	

In [74]:
```
final_data.dtypes
```

Out[74]:
```
country                          object
vaccines                         object
people_fully_vaccinated         float64
total_vaccinations              float64
population (in thousands)       float64
GDP (in millions)                object
dtype: object
```

Because GDP (in millions) is in object type we need to change it into int

In [75]:
```
final_data['GDP (in millions)'] = final_data['GDP (in millions)'].str.repla
```

Code to change GDP (in millions) into integers

In [76]:
```
final_data
```

Out[76]:

	country	vaccines	people_fully_vaccinated	total_vaccinations	population (in thousands)	r
0	Australia	Oxford/AstraZeneca, Pfizer/BioNTech	1.424238e+08	13222783.0	1131477.0	
1	Indonesia	Moderna, Oxford/AstraZeneca, Sinopharm/Beijing...	1.416955e+09	72386296.0	11654222.0	
2	Malaysia	Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac	2.924130e+08	23687251.0	1223634.0	
3	Philippines	Johnson&Johnson, Moderna, Oxford/AstraZeneca, ...	1.749103e+08	23230492.0	4115063.0	
4	Singapore	Moderna, Pfizer/BioNTech	1.047836e+08	7911869.0	218702.0	
5	Thailand	Oxford/AstraZeneca, Sinovac	2.075185e+08	18349011.0	3460815.0	

In [77]:
```
final_data['perCapitaGDP (in thousands)'] = final_data['GDP (in millions)']
```

calculate perCapitaGDP (in thousands) by dividing GDP (in millions) by population (in thousands)

In [78]:
```
final_data
```

Out[78]:

	country	vaccines	people_fully_vaccinated	total_vaccinations	population (in thousands)	r
0	Australia	Oxford/AstraZeneca, Pfizer/BioNTech	1.424238e+08	13222783.0	1131477.0	
1	Indonesia	Moderna, Oxford/AstraZeneca, Sinopharm/Beijing...	1.416955e+09	72386296.0	11654222.0	
2	Malaysia	Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac	2.924130e+08	23687251.0	1223634.0	
3	Philippines	Johnson&Johnson, Moderna, Oxford/AstraZeneca, ...	1.749103e+08	23230492.0	4115063.0	
4	Singapore	Moderna, Pfizer/BioNTech	1.047836e+08	7911869.0	218702.0	
5	Thailand	Oxford/AstraZeneca, Sinovac	2.075185e+08	18349011.0	3460815.0	

In [79]:
```
final_data.dtypes
```

Out[79]:
```
country                          object
vaccines                         object
people_fully_vaccinated         float64
total_vaccinations              float64
population (in thousands)       float64
GDP (in millions)                 int64
perCapitaGDP (in thousands)     float64
dtype: object
```

All the float data type will need to be change into integer for better visualization

In [80]:
```
final_data = final_data.astype({'people_fully_vaccinated': 'int', 'total_va
```

Code to change float data types into integer

In [81]:
```
final_data
```

Out[81]:

	country	vaccines	people_fully_vaccinated	total_vaccinations	population (in thousands)	r
0	Australia	Oxford/AstraZeneca, Pfizer/BioNTech	142423826	13222783	1131477	
1	Indonesia	Moderna, Oxford/AstraZeneca, Sinopharm/Beijing...	1416955068	72386296	11654222	
2	Malaysia	Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac	292412958	23687251	1223634	
3	Philippines	Johnson&Johnson, Moderna, Oxford/AstraZeneca, ...	174910302	23230492	4115063	
4	Singapore	Moderna, Pfizer/BioNTech	104783574	7911869	218702	
5	Thailand	Oxford/AstraZeneca, Sinovac	207518483	18349011	3460815	

After changing the data types, display final_data

In [82]:
```
final_data['percentage_of_people_fully_vaccinated (%)'] = final_data['peopl
```

Caculate percentage of people fully vaccinated by using above formula

In [83]:
```
final_data
```

Out[83]:

	country	vaccines	people_fully_vaccinated	total_vaccinations	population (in thousands)	r
0	Australia	Oxford/AstraZeneca, Pfizer/BioNTech	142423826	13222783	1131477	
1	Indonesia	Moderna, Oxford/AstraZeneca, Sinopharm/Beijing...	1416955068	72386296	11654222	
2	Malaysia	Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac	292412958	23687251	1223634	
3	Philippines	Johnson&Johnson, Moderna, Oxford/AstraZeneca, ...	174910302	23230492	4115063	
4	Singapore	Moderna, Pfizer/BioNTech	104783574	7911869	218702	
5	Thailand	Oxford/AstraZeneca, Sinovac	207518483	18349011	3460815	

In [84]:
```
final_data.describe().astype({'people_fully_vaccinated': 'int', 'total_vacc
```

Out[84]:

	people_fully_vaccinated	total_vaccinations	population (in thousands)	GDP (in millions)	perCapitaGDP (in thousands)	pe
count	6	6	6	6	6.000000	
mean	389834035	26464617	3633985	654878	0.554943	
std	507220904	23291146	4202906	431225	0.642700	
min	104783574	7911869	218702	336664	0.087845	
25%	150545445	14504340	1154516	345370	0.104363	
50%	191214392	20789751	2342224	431642	0.210064	
75%	271189339	23573061	3951501	919266	0.950972	
max	1416955068	72386296	11654222	1330901	1.554618	

From the data above, it can be seen that only Singapore that has almost half of its population vaccinated. While the other countries only managed to do 4% - 23% of its population with Philippines that has the lowest percentage of people fully vacinnated in its country.

Moreover, from 75th percentile values of people_fully_vaccinated. It is shown that Indonesia and Malaysia fall into that category. This means that both of the countries have the most people that have been fully vaccinated than the other countries. Indonesia has the most people that have been fully vaccinated, this does make sense since its population is the highest among the others.

Furthermore, it can be seen from the mean of perCapitaGDP, Australia and Singapore are more prosperous than other countries as it surpassed the mean of perCapitaGDP. Although from the mean of GDP(in millions), Australia and Indonesia are way better off than the other countries. It does not guarantee that the nation is prosperous enough with the large amount of population it has, for instance, Indonesia.

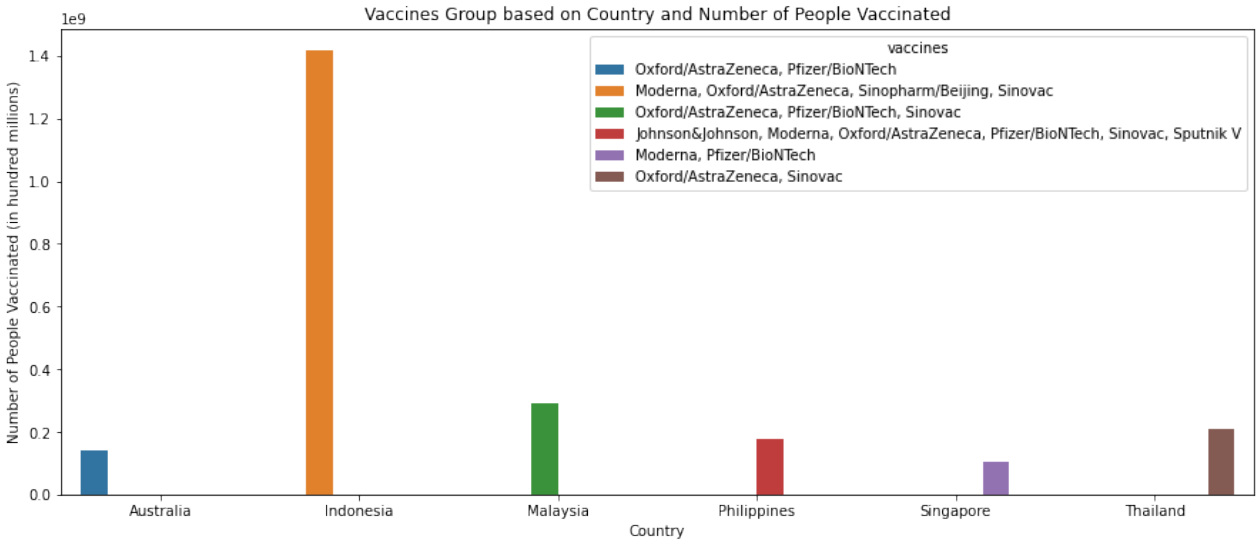
Question 1

In [85]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

Import the necessary libraries and make an alias for each of the library

In [86]:
```
plt.figure(figsize = (15,6))
sns.barplot(x = final_data['country'], y = final_data['people_fully_vaccina
plt.title('Vaccines Group based on Country and Number of People Vaccinated'
plt.xlabel('Country')
plt.ylabel('Number of People Vaccinated (in hundred millions)')
```

Out[86]:
```
Text(0, 0.5, 'Number of People Vaccinated (in hundred millions)')
```



I choose to plot the graph by using bar graph because displaying the spread of subjects across the different categories of a variable (categorical data) is most easily done by a bar chart.

It can be seen that even though each type of vaccines are equally distributed, there will be some countries that still get the most vaccine out of all countries because of its population. For instance, Indonesia with the highest population got the most vaccines than the other countries followed by Malaysia and Thailand. Therefore, it is quite impossible that all countries will get the same amount of vaccines even though it is equally distributed

Question 2

In [87]:
```
fun = {'population (in thousands)':'min', 'total_vaccinations':'min','peopl
groupbyCountry = final_data.groupby('country').agg(fun)
groupbyCountry
```

Out[87]:

country	population (in thousands)	total_vaccinations	people_fully_vaccinated
Australia	1131477	13222783	142423826
Indonesia	11654222	72386296	1416955068
Malaysia	1223634	23687251	292412958
Philippines	4115063	23230492	174910302
Singapore	218702	7911869	104783574
Thailand	3460815	18349011	207518483

We need to aggragate the necessarry data based on final_data in country column

In [88]:
```
groupbyCountry = groupbyCountry.reset_index()
```

Reset the index

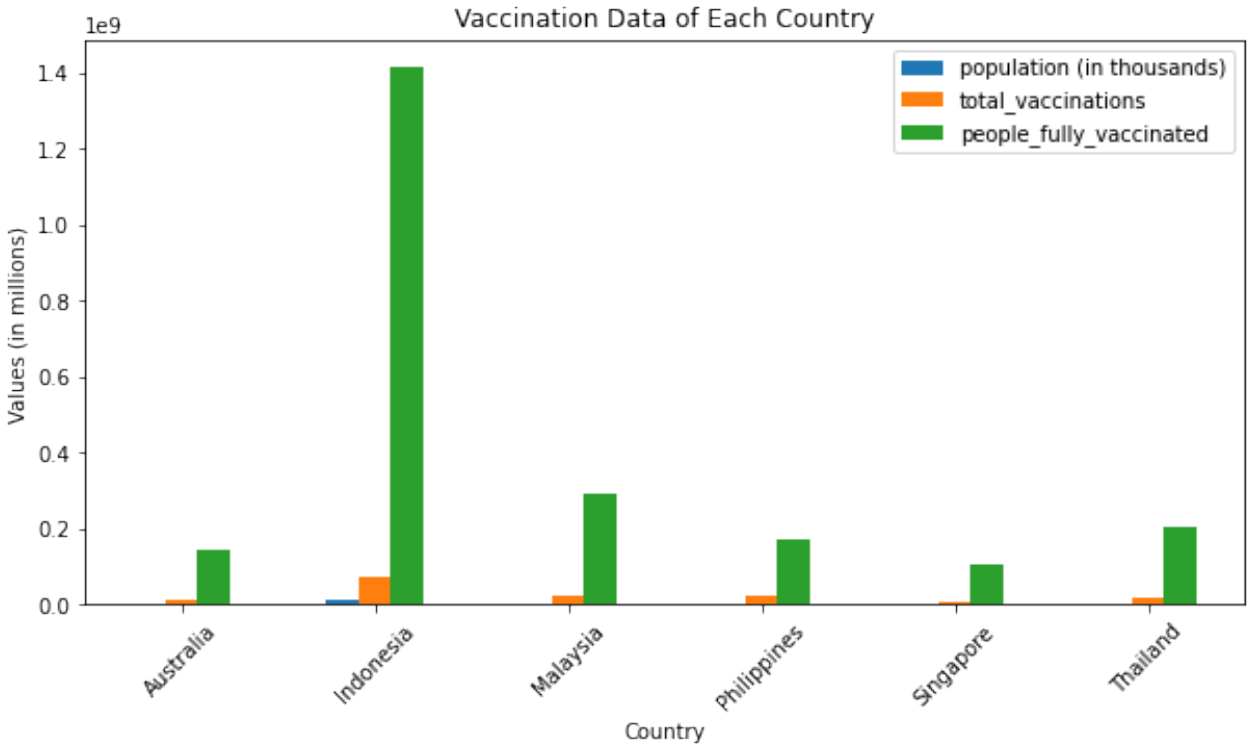
In [89]:
```
groupbyCountry
```

Out[89]:

	country	population (in thousands)	total_vaccinations	people_fully_vaccinated
0	Australia	1131477	13222783	142423826
1	Indonesia	11654222	72386296	1416955068
2	Malaysia	1223634	23687251	292412958
3	Philippines	4115063	23230492	174910302
4	Singapore	218702	7911869	104783574
5	Thailand	3460815	18349011	207518483

In [90]:
```
ax=groupbyCountry.plot.bar(figsize=(10,5))
ax.set_xticklabels(groupbyCountry['country'], rotation = 45)
ax.set(title = 'Vaccination Data of Each Country', xlabel = 'Country', ylab
```

Out[90]:
```
[Text(0.5, 1.0, 'Vaccination Data of Each Country'),
 Text(0.5, 0, 'Country'),
 Text(0, 0.5, 'Values (in millions)')]
```



The problem with this graph is it is difficult to visualise due to large differences in the numbers between people_fully_vaccinated and the other attributes making it impossible to see the bar graph of of total_vaccinations and people_fully_vaccinated. Thus, it is needed to create a graph for a better visualization

In [91]:
```
groupbyCountry['people_fully_vaccinated'] = groupbyCountry['people_fully_va
groupbyCountry['total_vaccinations'] = groupbyCountry['total_vaccinations']
```

I change the scaling of both people_fully_vaccinated (1:100) and total_vaccinations (1:10) so that all of the values can be seen in the bar graph

In [92]:
```
groupbyCountry.rename(columns = {'people_fully_vaccinated':'people_fully_va
```

Rename the columns

In [93]:
```
groupbyCountry
```

Out[93]:

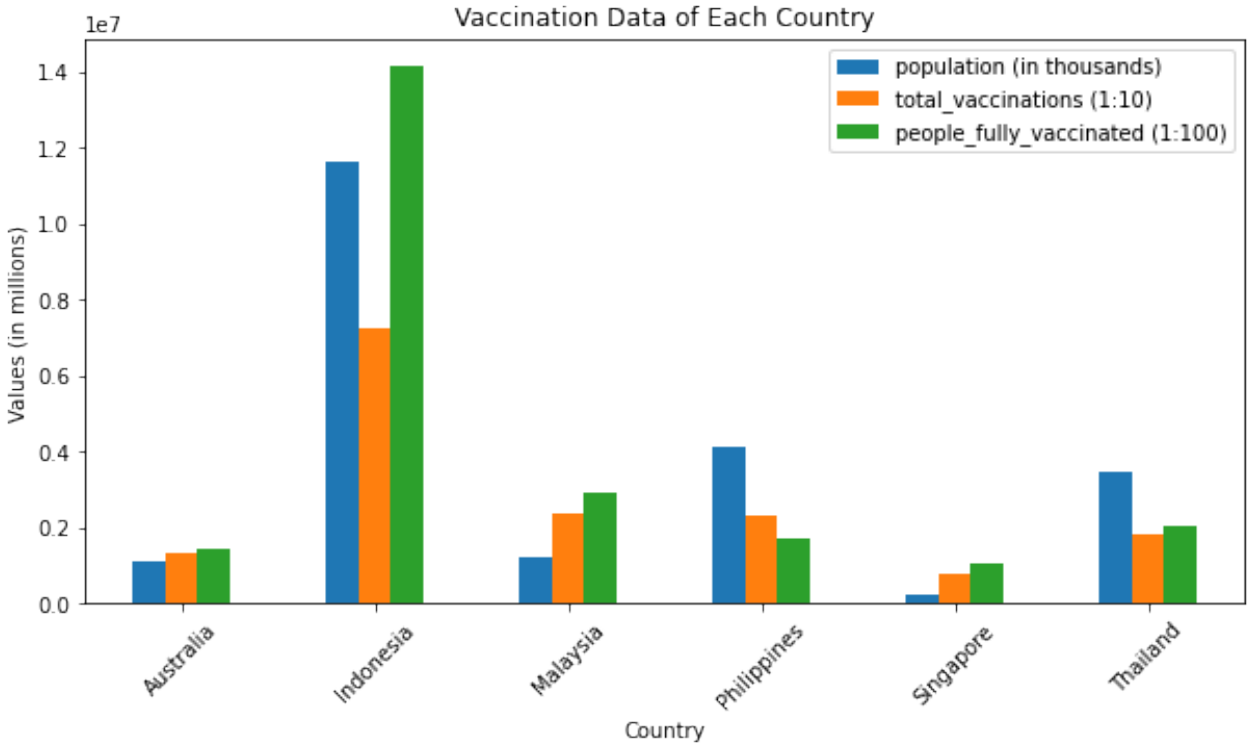
	country	population (in thousands)	total_vaccinations (1:10)	people_fully_vaccinated (1:100)
0	Australia	1131477	1322278.3	1424238.26
1	Indonesia	11654222	7238629.6	14169550.68
2	Malaysia	1223634	2368725.1	2924129.58
3	Philippines	4115063	2323049.2	1749103.02
4	Singapore	218702	791186.9	1047835.74
5	Thailand	3460815	1834901.1	2075184.83

In [94]:

```
ax=groupbyCountry.plot.bar(figsize=(10,5))
ax.set_xticklabels(groupbyCountry['country'], rotation = 45)
ax.set(title = 'Vaccination Data of Each Country', xlabel = 'Country', ylab
```

Out[94]:

```
[Text(0.5, 1.0, 'Vaccination Data of Each Country'),
 Text(0.5, 0, 'Country'),
 Text(0, 0.5, 'Values (in millions)')]
```



I choose to scale the total_vaccinations and people_fully_vaccinated into both millions, so that all of these values can be seen in the bar graph. Using those data may not be appropriate, because of the scaling we cannot see purely based on this graph. We still need to visualize the bar graph by ourselves with the scaling given. For example, if people_fully_vaccinated in Philippines is shown lower than population and total_vaccinations. In fact, people_fully_vaccinated bar must be higher than population and total_vaccinations.

Question 3

In [95]:
```
filt = (country_vaccinations.country == "Australia")
australia_vaccinations = country_vaccinations[filt].loc[:,['country','date'
```

Filter only the necessarry data and choose only country, date, and daily_vaccinations column

In [96]:
```
australia_vaccinations
```

Out[96]:

	country	date	daily_vaccinations
1746	Australia	2021-02-15	NaN
1747	Australia	2021-02-16	0.0
1748	Australia	2021-02-17	0.0
1749	Australia	2021-02-18	0.0
1750	Australia	2021-02-19	0.0
...	...	...	...
1917	Australia	2021-08-05	176432.0
1918	Australia	2021-08-06	180617.0
1919	Australia	2021-08-07	184239.0
1920	Australia	2021-08-08	188408.0
1921	Australia	2021-08-09	189893.0

176 rows × 3 columns

In [97]:
```
australia_vaccinations['cum_sum_of_daily_vaccinations'] = australia_vaccina
```

Calculate the cumulative sum of daily vaccinations by using .cumsum() function

In [98]:
```
australia_vaccinations
```

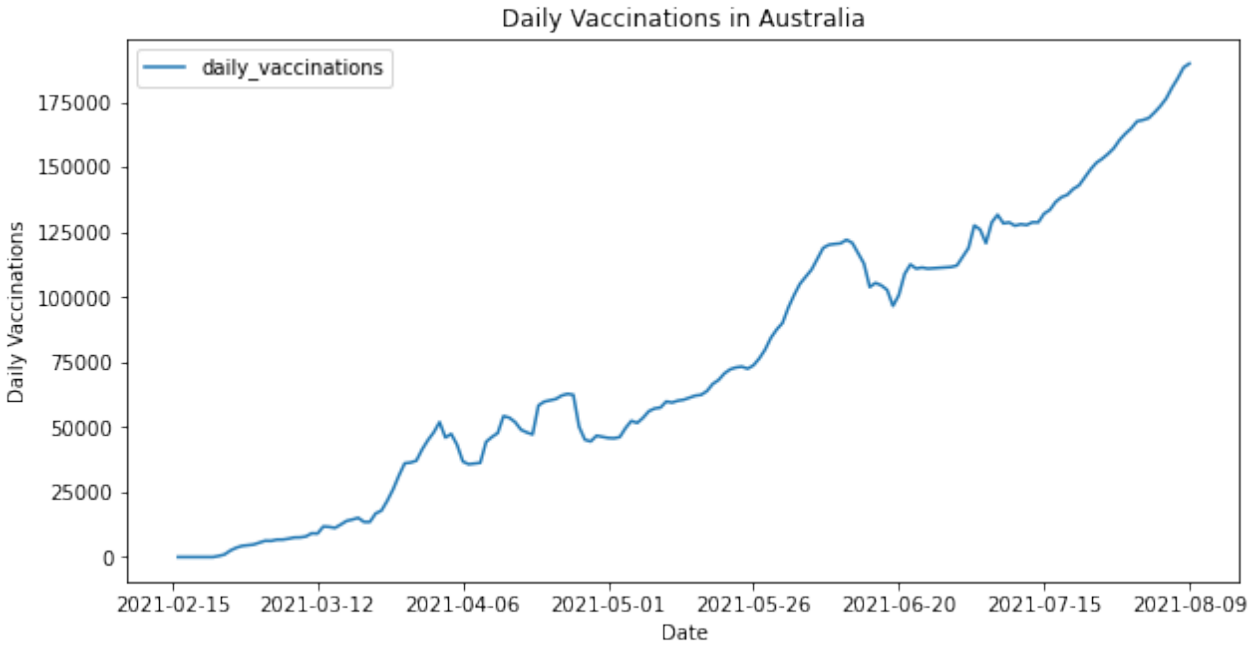
Out[98]:

	country	date	daily_vaccinations	cum_sum_of_daily_vaccinations
1746	Australia	2021-02-15	NaN	NaN
1747	Australia	2021-02-16	0.0	0.0
1748	Australia	2021-02-17	0.0	0.0
1749	Australia	2021-02-18	0.0	0.0
1750	Australia	2021-02-19	0.0	0.0
...	...	...	...	...
1917	Australia	2021-08-05	176432.0	12479626.0
1918	Australia	2021-08-06	180617.0	12660243.0
1919	Australia	2021-08-07	184239.0	12844482.0
1920	Australia	2021-08-08	188408.0	13032890.0
1921	Australia	2021-08-09	189893.0	13222783.0

176 rows × 4 columns

In [99]:
```
ax = australia_vaccinations.plot(x = 'date', y = 'daily_vaccinations', figs
ax.set(title = "Daily Vaccinations in Australia", xlabel = 'Date', ylabel =
```

Out[99]:
```
[Text(0.5, 1.0, 'Daily Vaccinations in Australia'),
 Text(0.5, 0, 'Date'),
 Text(0, 0.5, 'Daily Vaccinations')]
```



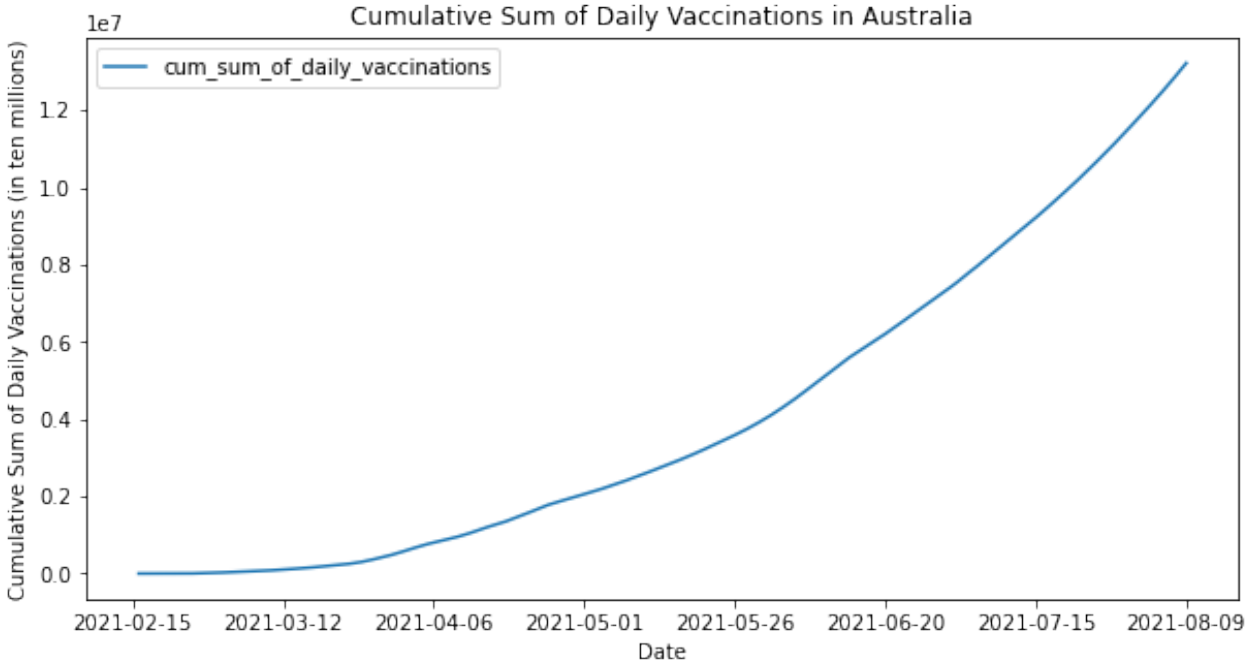
This graph is useful to know the trend of the daily vaccinations over time. As we can see daily vaccinations may have fluctuated in some period of time but on a whole it increases over time

In [100…

```
ax = australia_vaccinations.plot(x = 'date', y = 'cum_sum_of_daily_vaccinat
ax.set(title = "Cumulative Sum of Daily Vaccinations in Australia", xlabel
```

Out[100…

```
[Text(0.5, 1.0, 'Cumulative Sum of Daily Vaccinations in Australia'),
 Text(0.5, 0, 'Date'),
 Text(0, 0.5, 'Cumulative Sum of Daily Vaccinations (in ten millions)')]
```



This line graph shows the cummulative sum of daily vaccinations per day and it is shown that it increases over time

In [ ]:

```
ax = australia_vaccinations.plot(x = 'date', figsize = (10,5))
ax.set(title = "Daily & Cumulative Sum of Vaccinations in Australia", xlabe
```

Out[ ]:

```
[Text(0.5, 1.0, 'Daily & Cumulative Sum of Vaccinations in Australia'),
 Text(0.5, 0, 'Date'),
 Text(0, 0.5, 'Daily & Cumulative Sum of Vaccinations (in ten millions)')]
```

The first line which is the daily_vaccinations would be useful if we want to know how many vaccinations have been done in a day. In fact, the first line does not show much information other than that. On the other hand, the second line which is the cumulative sum of daily vaccinations would be useful in order to know the growth of daily vaccinations over time. The cumulative sum line is concaved up, it can predicted that daily vaccinations will continue on growing exponentially over time.

Conclusion

In conclusion, not all countries have managed to have half of its citizens fully vaccinated only Singapore that almost managed to do it. Futhermore, by using the value of GDP (in millions) we cannot immediately say that a country is prosperous. We need to see it from its perCapitaGDP that shows the prosperity of its country. In addition, it is quite impossible that all countries will get the same amount of vaccines even though it is equally distributed. Moreover, visualizing the data with appropriate way is very crucial for better understanding of the graph. Lastly, the cummulative sum of daily vaccinations in Australia is concaved up, it can predicted that daily vaccinations will continue on growing exponentially over time.

By completing this assignment, I also learned about how to wrangle and visualise the data, some basic statistics and some domain knowledge (e.g. GDP of countries and also about Covid-19 vaccination).