FIT1043 Introduction to Data Science

Assignment 1

Vanessa Khoo
31417493
1st April 2021

Introduction

This assignment (FIT1043) is to test the students' knowledge from the first 4 weeks of learning.

In lieu of the ongoing COVID-19 pandemic, more than a few organisations have recorded and shared important data relevant to the pandemic situation from all around the world, and made them available for access. In this assignment, we are given 3 sets of data, "Vaccinations.csv", "2019-GDP.csv" and "2020-Population.csv", sourced from Kaggle (link (https://www.kaggle.com/gpreda/covid-world-vaccination-progress)), The World Bank (link (https://datacatalog.worldbank.org/dataset/gdp-ranking)), and United Nations (link (https://population.un.org/wpp/Download/Standard/CSV/)) respectively.

This assignment and notebook is an attempt to show on the one hand, how the different datasets are to be read, wrangled, analysed and, on the other, to offer deeper statiscal analysis and insight regarding the data via graphical and non-graphical visualisation.

Here are the expected sequence of tasks to complete for marks:

1. Importing the libraries
2. Reading files
3. Wrangling the data
4. Merging files
5. Manage data issues
6. Feature engineer a new column "perCapitaGDP" & a final DataFrame with 11 rows & 7 columns
7. Provide some statistical description of final data (using basic statistics)
8. Plot appropriate graphs & provide basic insights to the 3 questions listed in the assignment sheet.

I will approach this assignment and problem by splitting it into **3 main phases** for me to tackle:

1. Understanding & Wrangling,
2. Feature Engineering & Insights,
3. Graph plotting & Insights.

Understanding & Wrangling: We begin this phase by reading and understanding the raw data. This is where the neccessary importing of the libraries and cleaning (wrangling) of the data happens. An additional approach taken here is to utilise .head() / .sample() whenever needed to observe data before or after making changes in order to find out the best next step to be done. With this, I am able to explicitly show readers my thought process on every step made alongside the comments and explanation.

Feature Engineering & Analysis: This is where new columns to be shown in the final DataFrame and final DataFrame is finalised. In this phase we will also do some analysis and provide some insight on this data in the final DataFrame.

Graph Plotting & Insights: In this last phase, we will plot graphs for visualiation to better understand and draw clearer insight from the data.

Phase i: Understanding & Wrangling

Task 1: Importing Libraries (for phase i & phase ii)

The 1st step is to import the library **pandas**, which is an open source data analysis tool for the python programming language. The purpose of importing this library is to use the data structure such as *DataFrame* and it's associated functions such as reading from CSV files and so on.

In [3]:

```
import pandas as pd
```

Task 2: Reading the files & showing that the data has been read correctly

First, we read the files using pd.read_csv("filepath") to read comma-separated values (csv) files into DataFrames.

In [4]:

```
# Reading the files with pd.read_csv("filepath")
gdp = pd.read_csv("C:\\Users\\Aaron Khoo\\Documents\\Vanessa\\1_Monash\\Y1S2\\FIT1043\\Assi
population = pd.read_csv("C:\\Users\\Aaron Khoo\\Documents\\Vanessa\\1_Monash\\Y1S2\\FIT104
vaccinations = pd.read_csv("C:\\Users\\Aaron Khoo\\Documents\\Vanessa\\1_Monash\\Y1S2\\FIT1
```

To understand the type of data and fields in each dataset, let us see the first 30 rows of each with the .head(30) function. The following 3 cells uses the .head() function, which is used to get the first n rows of DataFrame. If n is not specified, it will give the first 5 rows.

In [5]:

```
gdp.head(30)
```

Out[5]:

	Unnamed: 0	Gross domestic product 2019	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	21,427,700	NaN
5	CHN	2	NaN	China	14,342,903	NaN
6	JPN	3	NaN	Japan	5,081,770	NaN
7	DEU	4	NaN	Germany	3,845,630	NaN
8	IND	5	NaN	India	2,875,142	NaN
9	GBR	6	NaN	United Kingdom	2,827,113	NaN
10	FRA	7	NaN	France	2,715,518	NaN
11	ITA	8	NaN	Italy	2,001,244	NaN
12	BRA	9	NaN	Brazil	1,839,758	NaN
13	CAN	10	NaN	Canada	1,736,426	NaN
14	RUS	11	NaN	Russian Federation	1,699,877	a
15	KOR	12	NaN	Korea, Rep.	1,642,383	NaN
16	ESP	13	NaN	Spain	1,394,116	NaN
17	AUS	14	NaN	Australia	1,392,681	NaN
18	MEX	15	NaN	Mexico	1,258,287	NaN
19	IDN	16	NaN	Indonesia	1,119,191	NaN
20	NLD	17	NaN	Netherlands	909,070	NaN
21	SAU	18	NaN	Saudi Arabia	792,967	NaN
22	TUR	19	NaN	Turkey	754,412	NaN
23	CHE	20	NaN	Switzerland	703,082	NaN
24	POL	21	NaN	Poland	592,164	NaN
25	THA	22	NaN	Thailand	543,650	NaN
26	SWE	23	NaN	Sweden	530,833	NaN
27	BEL	24	NaN	Belgium	529,607	NaN
28	ARG	25	NaN	Argentina	449,663	b
29	NGA	26	NaN	Nigeria	448,120	NaN

In [6]:

```
population.head(30)
```

Out[6]:

	United Nations	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unna
0	Population Division	NaN	NaN	NaN	NaN	
1	Department of Economic and Social Affairs	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	
3	World Population Prospects 2019	NaN	NaN	NaN	NaN	
4	File POP/1-1: Total population (both sexes com...	NaN	NaN	NaN	NaN	
5	Estimates, 1950 - 2020	NaN	NaN	NaN	NaN	
6	POP/DB/WPP/Rev.2019/POP/F01-1	NaN	NaN	NaN	NaN	
7	© August 2019 by United Nations, made availabl...	NaN	NaN	NaN	NaN	
8	Suggested citation: United Nations, Department...	NaN	NaN	NaN	NaN	
9	NaN	NaN	NaN	NaN	NaN	
10	NaN	NaN	NaN	NaN	NaN	
11	Index	Variant	Region, subregion, country or area *	Notes	Country code	
12	1	Estimates	WORLD	NaN	900	
13	2	Estimates	UN development groups	a	1803	Label/Se
14	3	Estimates	More developed regions	b	901	Devel
15	4	Estimates	Less developed regions	c	902	Devel
16	5	Estimates	Least developed countries	d	941	Devel
17	6	Estimates	Less developed regions, excluding least develo...	e	934	Devel

	United Nations	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unna
18	7	Estimates	Less developed regions, excluding China	NaN	948	Devel
19	8	Estimates	Land-locked Developing Countries (LLDC)	f	1636	Speci
20	9	Estimates	Small Island Developing States (SIDS)	g	1637	Speci
21	10	Estimates	World Bank income groups	NaN	1802	Label/Se
22	11	Estimates	High-income countries	h	1503	Income
23	12	Estimates	Middle-income countries	h	1517	Income
24	13	Estimates	Upper-middle-income countries	h	1502	Income
25	14	Estimates	Lower-middle-income countries	h	1501	Income
26	15	Estimates	Low-income countries	h	1500	Income
27	16	Estimates	No income group available	NaN	1518	Income
28	17	Estimates	Geographic regions	i	1840	Label/Se
29	18	Estimates	Africa	j	903	

30 rows × 78 columns

In [7]:

```python
vaccinations.head(30)
```

Out[7]:

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	da
0	Albania	ALB	2021-01-10	0.0	0.0	NaN	
1	Albania	ALB	2021-01-11	NaN	NaN	NaN	
2	Albania	ALB	2021-01-12	128.0	128.0	NaN	
3	Albania	ALB	2021-01-13	188.0	188.0	NaN	
4	Albania	ALB	2021-01-14	266.0	266.0	NaN	
5	Albania	ALB	2021-01-15	308.0	308.0	NaN	
6	Albania	ALB	2021-01-16	369.0	369.0	NaN	
7	Albania	ALB	2021-01-17	405.0	405.0	NaN	
8	Albania	ALB	2021-01-18	447.0	447.0	NaN	
9	Albania	ALB	2021-01-19	483.0	483.0	NaN	
10	Albania	ALB	2021-01-20	519.0	519.0	NaN	
11	Albania	ALB	2021-01-21	549.0	549.0	NaN	
12	Albania	ALB	2021-01-22	NaN	NaN	NaN	
13	Albania	ALB	2021-01-23	NaN	NaN	NaN	
14	Albania	ALB	2021-01-24	NaN	NaN	NaN	
15	Albania	ALB	2021-01-25	NaN	NaN	NaN	
16	Albania	ALB	2021-01-26	NaN	NaN	NaN	
17	Albania	ALB	2021-01-27	NaN	NaN	NaN	
18	Albania	ALB	2021-01-28	NaN	NaN	NaN	
19	Albania	ALB	2021-01-29	NaN	NaN	NaN	
20	Albania	ALB	2021-01-30	NaN	NaN	NaN	
21	Albania	ALB	2021-01-31	NaN	NaN	NaN	

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	da
22	Albania	ALB	2021-02-01	NaN	NaN	NaN	
23	Albania	ALB	2021-02-02	550.0	549.0	1.0	
24	Albania	ALB	2021-02-03	NaN	NaN	NaN	
25	Albania	ALB	2021-02-04	NaN	NaN	NaN	
26	Albania	ALB	2021-02-05	NaN	NaN	NaN	
27	Albania	ALB	2021-02-06	NaN	NaN	NaN	
28	Albania	ALB	2021-02-07	NaN	NaN	NaN	
29	Albania	ALB	2021-02-08	NaN	NaN	NaN	

Task 3: Wrangling the data

Now, the data will be wrangled and cleaned in order to prepare the FinalDataframe in phase ii (Feature Engineering & Insights).

Firstly, since this assignment only needs the South East Asian countries, including East Timor, I have created a **list** containing all the 11 South East Asian countries, as well as any alternate names of the countries found in the datasets. The reason why a list is the chosen data structure to store these names is because:

1. This makes it easier during the wrangling of the datasets when we want to remove rows not related to any South East Asian country.

The cell below does exactly as described.

In [8]:

```
# A list is created to store names of the countries required
southeastasian = ['Indonesia', 'Thailand', 'Philippines', 'Vietnam', 'Singapore', 'Malaysia
```

Let us now clean each DataFrame one by one as each dataset needs to have some data excluded from it for the final DataFrame.

VACCINATIONS WRANGLING

We start first with the vaccinations dataset. To better understand the data, we use .head() to see the first 5 rows of it just before we start to wrangle the data. As well as .shape to see the shape of the dataFrame.

In [9]:

```
vaccinations.head()
```

Out[9]:

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	dail
0	Albania	ALB	2021-01-10	0.0	0.0	NaN	
1	Albania	ALB	2021-01-11	NaN	NaN	NaN	
2	Albania	ALB	2021-01-12	128.0	128.0	NaN	
3	Albania	ALB	2021-01-13	188.0	188.0	NaN	
4	Albania	ALB	2021-01-14	266.0	266.0	NaN	

In [10]:

```
vaccinations.shape
```

Out[10]:

```
(4146, 15)
```

By observation of the above, we can tell that there are many countries & rows within the dataset and that there are multiple rows per country. Therefore, we want to find a way to:

1. Remove certain columns that aren't needed.
2. Remove countries not in southeast asian.
3. Group the southeast asian countries' needed values.

The cell below first **removes** the unwanted columns and keeps 'country', 'people_fully_vaccinated', 'daily_vaccinations', 'vaccines', as specified in the assignment specfications. To see how it looks like, we use the .sample(5) function to see 5 random rows in the dataframe.

In [11]:

```
vaccinations = vaccinations[['country', 'people_fully_vaccinated', 'daily_vaccinations', 'v
vaccinations.sample(5)
```

Out[11]:

	country	people_fully_vaccinated	daily_vaccinations	vaccines
1483	Germany	NaN	39448.0	Moderna, Oxford/AstraZeneca, Pfizer/BioNTech
234	Azerbaijan	NaN	3250.0	Oxford/AstraZeneca, Sputnik V
891	Costa Rica	2421.0	2677.0	Pfizer/BioNTech
2895	Pakistan	NaN	4043.0	Oxford/AstraZeneca, Sinopharm/Beijing, Sputnik V
13	Albania	NaN	26.0	Pfizer/BioNTech

Now, let us remove the unwanted countries. (those not south east asian). We do this by creating a boolean series to find out which rows have a value in "country" column that is in the southeastasian list. This boolean is then used to create a vaccinations_southeast dataframe that only includes rows that returned true in the boolean series.

In [12]:

```
# create bool series from isin() for southeastasian countries
southeast = vaccinations["country"].isin(southeastasian)
# update DataFrame to include country == those in southeastasian ONLY
vaccinations_southeast = vaccinations[southeast].reset_index()
vaccinations_southeast.drop(vaccinations_southeast.columns[0], axis = 1, inplace=True)
vaccinations_southeast.head()
```

Out[12]:

	country	people_fully_vaccinated	daily_vaccinations	vaccines
0	Cambodia	NaN	NaN	Sinopharm/Beijing
1	Cambodia	NaN	1492.0	Sinopharm/Beijing
2	Cambodia	NaN	871.0	Sinopharm/Beijing
3	Cambodia	NaN	663.0	Sinopharm/Beijing
4	Cambodia	NaN	560.0	Sinopharm/Beijing

We are then required to group the vaccination rows by country and have the values summed together. To tackle this, I created an aggregate condition, where we want to **group the vaccinations by country**, **get the latest value for people_fully_vaccinated, sum for daily_vaccinations values by country**, and **get the vaccine values by only keeping the last value of vaccines column for each group (since they are duplicates).** We use a combination of .groupby() & .agg() to do this, where .groupby() groups rows by the same values in the specified column, and .agg() is used to pass a function or list of function to be applied on a series or even each element of series separately. After the groupby & agg functions, we use .columns = [] to reset the column names to what was mentioned in the assignment.

Note: total_vaccinations = sum of the daily_vaccinations column *instead* of the total_vaccinations columns, because the values for total_vaccinations (in the dataset) are inconsistent.

In [13]:

```
agg1 = {'people_fully_vaccinated':'last', 'daily_vaccinations':'sum', 'vaccines':'last'}
vaccinations_final = vaccinations_southeast.groupby('country').agg(agg1).reset_index()
vaccinations_final.columns = ['country', 'people_fully_vaccinated', 'total_vaccinations', '
vaccinations_final
```

Out[13]:

	country	people_fully_vaccinated	total_vaccinations	vaccines
0	Cambodia	NaN	8171.0	Sinopharm/Beijing
1	Indonesia	825650.0	2022788.0	Sinovac
2	Myanmar	NaN	82823.0	Oxford/AstraZeneca
3	Singapore	110000.0	329630.0	Pfizer/BioNTech

Lastly, in order to improve future manipulation & usage of the data in this dataframe, let us see if there is a need to alter the data types of the column values. We use .dtypes to do this in the cell below.

In [14]:

```
vaccinations_final.dtypes
```

Out[14]:

```
country                     object
people_fully_vaccinated    float64
total_vaccinations         float64
vaccines                    object
dtype: object
```

From the above cell output, intuitively we can see that the columns that hold numeric values, are indeed of numeric type, and that columns that hold object/string values, are indeed of object type. Thus, there is no need to do any datatype changes.

Therefore with this final step, we observe that the above vaccinations_final DataFrame has been cleaned according to the requirements and ready to be used for the merging to get the final DataFrame later.

GDP WRANGLING

Next, we move onto the GDP dataset. To understand the the data, let us use .head(10) to see the first 10 rows of the dataset. As well as .shape to see the shape of the dataFrame. Note that we only need southeast asian countries and their GDP values for the final dataframe later.

In [15]:

```
gdp.head(10)
```

Out[15]:

	Unnamed: 0	Gross domestic product 2019	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	21,427,700	NaN
5	CHN	2	NaN	China	14,342,903	NaN
6	JPN	3	NaN	Japan	5,081,770	NaN
7	DEU	4	NaN	Germany	3,845,630	NaN
8	IND	5	NaN	India	2,875,142	NaN
9	GBR	6	NaN	United Kingdom	2,827,113	NaN

In [16]:

```
gdp.shape
```

Out[16]:

```
(244, 6)
```

From the above data, we can observe that:

```
1. The top 4 rows are rows that do not contain the data we want, and
2. Columns 0, 1, 2, 5 are columns we do not need.
```

Thus, the cell below first **removes** the first 4 rows, and then **removes** the unwanted columns and keeps only 'country' and 'GDP'. The new dataframe with the columns removed is called 'gdp_dropped'.

To do this, we use the function .drop(), which removes rows or columns by specifying label names and corresponding axis, or by specifying directly index or column names.

- To remove the first 4 rows, the parameters for .drop() is .index([0:4]), which selects the rows according to the indexes, there is no need to mention the axis for this first use of .drop() as .drop() automatically sets the axis for rows if it is not mentioned.
- To remove columns 0, 1, 2, 5, we again use .drop(), and in the parameters enter the list of column labels to drop. Here we specify the axis=1 as we want to remove columns, not rows. Also, we set inplace=True to make immediate change to gdp_dropped.

After the removal of the rows & columns, we rename the columns with .columns = [].

In [17]:

```
# Removes the first 4 rows of the dataframe, and columns 0, 1, 2, 5.
# After the removal, the dataframe's headers are renamed appropriately.
gdp_dropped = gdp.drop(gdp.index[0:4])
gdp_dropped.drop(gdp_dropped.columns[[0, 1, 2, 5]], axis=1, inplace=True)
gdp_dropped.columns=['country', 'GDP']
```

To see how it looks like, we use the .head(10) function to see the first 10 rows in the new gdp_dropped dataframe.

In [18]:

```
gdp_dropped.head(10)
```

Out[18]:

	country	GDP
4	United States	21,427,700
5	China	14,342,903
6	Japan	5,081,770
7	Germany	3,845,630
8	India	2,875,142
9	United Kingdom	2,827,113
10	France	2,715,518
11	Italy	2,001,244
12	Brazil	1,839,758
13	Canada	1,736,426

From the above, we observe that

1. It still includes countries **not** in southeast asia
2. index column values do not start at 0

Thus in the following cell, **a boolean series is created using the .isin() function to check if the country values are in the southeastasian list**. This boolean series is then used to create a **'gdp_final' dataframe with only the southeast asian countries' rows**. The .reset_index() is then used to reset the index back to start at 0. To remove the additional 'index' column made after resetting the index, we use .drop() to remove column 0.

Then, we view the final gdp_final dataframe, we will expect to see 11 rows.

In [19]:

```
# Create boolean series from isin() for southeastasian countries.
southeast = gdp_dropped["country"].isin(southeastasian)

# update DataFrame to include southeastasian countries only & reset the index.
gdp_final = gdp_dropped[southeast].reset_index()

# Removing the additional 'index' column
gdp_final.drop(gdp_final.columns[0], axis = 1, inplace=True)
gdp_final
```

Out[19]:

	country	GDP
0	Indonesia	1,119,191
1	Thailand	543,650
2	Philippines	376,796
3	Singapore	372,063
4	Malaysia	364,702
5	Vietnam	261,921
6	Myanmar	76,086
7	Cambodia	27,089
8	Lao PDR	18,174
9	Brunei Darussalam	13,469
10	Timor-Leste	1,674

Lastly, in order to improve future manipulation & usage of the data in this dataframe, let us see if there is a need to alter the data types of the column values. We use .dtypes to do this in the cell below.

In [20]:

```
gdp_final.dtypes
```

Out[20]:

```
country    object
GDP        object
dtype: object
```

As observed above, we can see that the GDP columns, which intuitively we know should be an integer or float, is an object type and not a numeric type. Thus, the cell below helps to replace unwanted characters (',' for GDP) for the GDP column. After removing these unwanted characters, we then convert the column values to be numeric values.

We do this by:

- Using the .apply() function to apply the passed in function to each column of 'cols'

Then, to check whether the datatype has changed, the .dtypes attribute is used.

In [21]:

```
# Removing the unwanted characters.
gdp_final['GDP'] = gdp_final['GDP'].str.replace(',', '')

# Using .apply() to convert the object type values to numeric type
gdp_final['GDP'] = gdp_final['GDP'].apply(pd.to_numeric, errors='coerce')

# Checking if the datatype has indeed changed
gdp_final.dtypes
```

Out[21]:

```
country     object
GDP          int64
dtype: object
```

From the above cell output, we see that the 'GDP' column is of 'int64' type, which would be faithful to the values the column holds. This would also be useful in future usage of this data.

Therefore with this final step, we observe that the above gdp_final DataFrame has been cleaned according to the requirements and ready to be used for the merging to get the final DataFrame later.

POPULATIONS WRANGLING

Next, we move onto the population dataframe. To understand the the data, let us use .head(30) to see the first 30 rows of the dataset. As well as .shape to see the shape of the dataFrame.

Note that we only need **southeast asian countries** and their **2019 population values** for the final dataframe later.

In [22]:

```
population.head(30)
```

Out[22]:

	United Nations	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed
0	Population Division	NaN	NaN	NaN	NaN	Na
1	Department of Economic and Social Affairs	NaN	NaN	NaN	NaN	Na
2	NaN	NaN	NaN	NaN	NaN	Na
3	World Population Prospects 2019	NaN	NaN	NaN	NaN	Na
4	File POP/1-1: Total population (both sexes com...	NaN	NaN	NaN	NaN	Na
5	Estimates, 1950 - 2020	NaN	NaN	NaN	NaN	Na
6	POP/DB/WPP/Rev.2019/POP/F01-1	NaN	NaN	NaN	NaN	Na
7	© August 2019 by United Nations, made availabl...	NaN	NaN	NaN	NaN	Na
8	Suggested citation: United Nations, Department...	NaN	NaN	NaN	NaN	Na
9	NaN	NaN	NaN	NaN	NaN	Na
10	NaN	NaN	NaN	NaN	NaN	Na
11	Index	Variant	Region, subregion, country or area *	Notes	Country code	Ty
12	1	Estimates	WORLD	NaN	900	Wo
13	2	Estimates	UN development groups	a	1803	Label/Separa
14	3	Estimates	More developed regions	b	901	Developme Gro
15	4	Estimates	Less developed regions	c	902	Developme Gro
16	5	Estimates	Least developed countries	d	941	Developme Gro
17	6	Estimates	Less developed regions, excluding least develo...	e	934	Developme Gro

	United Nations	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed
18	7	Estimates	Less developed regions, excluding China	NaN	948	Developme Gro
19	8	Estimates	Land-locked Developing Countries (LLDC)	f	1636	Special oth
20	9	Estimates	Small Island Developing States (SIDS)	g	1637	Special oth
21	10	Estimates	World Bank income groups	NaN	1802	Label/Separa
22	11	Estimates	High-income countries	h	1503	Income Gro
23	12	Estimates	Middle-income countries	h	1517	Income Gro
24	13	Estimates	Upper-middle-income countries	h	1502	Income Gro
25	14	Estimates	Lower-middle-income countries	h	1501	Income Gro
26	15	Estimates	Low-income countries	h	1500	Income Gro
27	16	Estimates	No income group available	NaN	1518	Income Gro
28	17	Estimates	Geographic regions	i	1840	Label/Separa
29	18	Estimates	Africa	j	903	Regi

30 rows × 78 columns

In [23]:

```
population.shape
```

Out[23]:

```
(301, 78)
```

From the above data, we can observe that:

1. The top 11 rows are rows that do not contain the data we want, and
2. Columns 0-1, 3-75, 77 are columns we do not need.

Thus, the cell below first **removes** rows 0 to 10 (first 11 rows) using .drop() to drop the selected rows & .index() to select the rows. Then columns 0-1, 3-75, 77 are removed by:

- first dropping the middle 3-75 columns, then dropping column 0, 1 & 77 (note that 77 would be column with index 4 after the first drop.

We will then view the top 5 rows of the dataFrame.

In [24]:

```
population_dropped = population.drop(population.index[0:11])
population_dropped.drop(population_dropped.columns[3:76], axis=1, inplace=True)
population_dropped.drop(population_dropped.columns[[0, 1, 4]], axis=1, inplace=True)
population_dropped.head()
```

Out[24]:

	Unnamed: 2	Unnamed: 76
11	Region, subregion, country or area *	2019
12	WORLD	7 713 468
13	UN development groups	...
14	More developed regions	1 270 630
15	Less developed regions	6 442 838

From the above cell output, we can see that:

```
1. The column headers are in the first row
2. It includes countries not in southeast asia.
```

Thus, to solve 1., the following cell first renames the column headers with the first row, then removes this first row from the dataframe.

In [25]:

```
population_dropped.rename(columns=population_dropped.iloc[0], inplace=True)
population_dropped = population_dropped[1:]
population_dropped.head()
```

Out[25]:

	Region, subregion, country or area *	2019
12	WORLD	7 713 468
13	UN development groups	...
14	More developed regions	1 270 630
15	Less developed regions	6 442 838
16	Least developed countries	1 033 389

Subsequently to solve 2., we will remove non-southeast asian countries by:

- Creating a boolean series with .isin() referring to the southeastasian list to see which values in the "Region, subregion, country or area *" column are in southeast asia. Then, with this boolean series we create a new 'population_final' dataframe that includes all rows with countries in southeast asia.

To reset the index, we again use .reset_index() to reset the index after dropping some rows, and use .drop() to remove the additional index column created.

We expect 11 rows for 11 southeast asian countried after viewing population_final at the end.

In [26]:

```
# Create bool series from isin() for southeastasian countries
southeast = population_dropped["Region, subregion, country or area *"].isin(southeastasian)

# 'population_final' to include only countries in southeastasia
population_final = population_dropped[southeast].reset_index()
population_final.drop(population_final.columns[0], axis=1, inplace=True)
population_final
```

Out[26]:

	Region, subregion, country or area *	2019
0	Brunei Darussalam	433
1	Cambodia	16 487
2	Indonesia	270 626
3	Lao People's Democratic Republic	7 169
4	Malaysia	31 950
5	Myanmar	54 045
6	Philippines	108 117
7	Singapore	5 804
8	Thailand	69 626
9	Timor-Leste	1 293
10	Viet Nam	96 462

Now, we will rename the column header names with .column = ['country', '2019_population'], to make this final population dataframe's column headers be in line for the final dataframe later.

In [27]:

```
# Renaming the column headers by assignment with .column
population_final.columns = ['country', '2019_population']
population_final
```

Out[27]:

	country	2019_population
0	Brunei Darussalam	433
1	Cambodia	16 487
2	Indonesia	270 626
3	Lao People's Democratic Republic	7 169
4	Malaysia	31 950
5	Myanmar	54 045
6	Philippines	108 117
7	Singapore	5 804
8	Thailand	69 626
9	Timor-Leste	1 293
10	Viet Nam	96 462

As mentioned at the start of phase i, there are many ways that each countries's names are represented in different datasets. Therefore, to make sure we keep it to a standard name, the following cell changes the specfic country column cell values to their standard names.

In [28]:

```
# Change unconventional country names into normal country names
population_final['country'][3] = 'Lao PDR'
population_final['country'][10] = 'Vietnam'
population_final
```

Out[28]:

	country	2019_population
0	Brunei Darussalam	433
1	Cambodia	16 487
2	Indonesia	270 626
3	Lao PDR	7 169
4	Malaysia	31 950
5	Myanmar	54 045
6	Philippines	108 117
7	Singapore	5 804
8	Thailand	69 626
9	Timor-Leste	1 293
10	Vietnam	96 462

Lastly, in order to improve future manipulation & usage of the data in this dataframe, let us see if there is a need to alter the data types of the column values. We use .dtypes to do this in the cell below.

In [29]:

```
population_final.dtypes
```

Out[29]:

```
country            object
2019_population    object
dtype: object
```

As observed above, we can see that the 2019_population, which intuitively we know should be an integer or float, is of object type instead of a numeric type. Thus, the cell below helps to replace unwanted characters (space ' ' for 2019_population) for the column's values. After removing these unwanted characters, we then convert the column values to be numeric values.

We do this by:

- Using the .apply() function to apply the passed in function for '2019_population'

Then, to check whether the datatype has changed, the .dtypes attribute is used.

In [30]:

```
# Remove the spaces & commas between the numbers (in string type)
population_final['2019_population'] = population_final['2019_population'].str.replace(' ',

# Convert string to numeric for the 2019_population
population_final['2019_population'] = population_final['2019_population'].apply(pd.to_numer

# Checking if the datatype has changed
population_final.dtypes
```

Out[30]:

```
country            object
2019_population     int64
dtype: object
```

From the above cell output, we see that the '2019_population' column is of 'int64' type, which would be faithful to the values the column holds. This would also be useful in future usage of this data.

Therefore with this final step, we observe that the above population_final DataFrame has been cleaned according to the requirements and ready to be used for the merging to get the final DataFrame later.

Phase ii: Feature Engineering & Insights

In phase ii, we will be merging all the cleaned dataframes into one single dataframe to feature engineer a new column and result with the final dataframe required by the assignment specifications.

In this phase we will complete:

- Task 4: Merging the files correctly
- Task 5: Manage any data type issues or data issues
- Task 6: Feature Engineer 'perCapitaGDP'
- Task 7: Provide Statistical Description of final data

Task 4: Merging the files & Creating the Final DataFrame

My approach to utilise a function called 'reduce()' from the functools library, this function helps us with merging > 2 dataframes because:

The reduce(func, seq) function is used to apply a particular function passed in its argument to all of the list elements mentioned in the sequence.

How it works:

1. In the first step, the first two elements of the sequence are picked and the result is obtained after applying the function.
2. The next step is to apply the same function to the previously attained result and the next element just succeeding the second element and the result is stored.
3. This process continues until no more elements are left to return the final result.

Merging 3 dataframes

Merging 3 dataframes

The cell below first imports the specific function 'reduce()' from the 'functools' library for reasons explained above.

In [31]:

```
from functools import reduce
```

The next cell then uses this reduce() function, along with an anonymous function utilising pd.merge() to merge the dataframes in the dfs list.

How it goes:

- A 'dfs' list is created to list down the dataframes we want to merge.
- We use the reduce() function, and pass in the lambda function with pd.merge() as the function to be applied for each subsequent element in 'dfs'.
- pd.merge() will take in the left & right dataframe, the 'on' parameter as 'country' (since all 3 dfs have the 'country' column), and the 'how' parameter as 'outer'. Putting how="outer" gets all the rows from the left dataframe, all the rows from the right dataframe, and match up rows where possible, with NaNs elsewhere. This is so that no data is leftout from both dataframes.

In [32]:

```
# Create dfs list for the dataframes we want to merge
dfs = [vaccinations_final, population_final, gdp_final]

# Create final DataFrame by  using the reduce() and pd.merge function
# to iteratively merge the dataframes together into one final dataframe.
df_final = reduce(lambda left,right: pd.merge(left,right,on='country', how="outer"), dfs)

# Note: We don't fill up NaN values as 0, as these might be some values that are missing fr
df_final
```

Out[32]:

	country	people_fully_vaccinated	total_vaccinations	vaccines	2019_population
0	Cambodia	NaN	8171.0	Sinopharm/Beijing	16487
1	Indonesia	825650.0	2022788.0	Sinovac	270626
2	Myanmar	NaN	82823.0	Oxford/AstraZeneca	54045
3	Singapore	110000.0	329630.0	Pfizer/BioNTech	5804
4	Brunei Darussalam	NaN	NaN	NaN	433
5	Lao PDR	NaN	NaN	NaN	7169
6	Malaysia	NaN	NaN	NaN	31950
7	Philippines	NaN	NaN	NaN	108117
8	Thailand	NaN	NaN	NaN	69626
9	Timor-Leste	NaN	NaN	NaN	1293
10	Vietnam	NaN	NaN	NaN	96462

Task 5, 6: Feature engineer 'perCapitaGDP' column

To feature engineer this column, it is expected that we will divide the GDP column's values by the 2019_population's values to arrive at perCapitaGDP's column values.

To make sure that all related rows have numeric datatypes to perform arithmetic calculations, the cell below uses the dataframe attribute .dtypes to see each column's data types.

In [33]:

```
df_final.dtypes  # to see the data types
```

Out[33]:

```
country                     object
people_fully_vaccinated    float64
total_vaccinations         float64
vaccines                    object
2019_population              int64
GDP                          int64
dtype: object
```

As observed above, we can make an intuitive observation by realizing:

- Columns that should be numeric, namely 'people_fully_vaccinated', 'total_vaccinations', '2019_population' & 'GDP', are numeric.
- Columns that should be object(string) types, namely 'country' & 'GDP' are rightfully object types.

Therefore, there is no need for us to manipulate and change the datatype for any column values, since most data type manipulation was done beforehand.

Thus, we will now **feature engineer the new perCapitaGDP column using the 2019_population & GDP columns**.

Note:

1. The 2019_population values are in *thousands*.
2. We need to rename the neccessary column headers as described in the assignment specifications.

In [34]:

```
# Add new column for perCapitaGDP, perCapitaGDP = GDP/2019_population
df_final['perCapitaGDP'] = df_final['GDP'] / (df_final['2019_population']*1000)   # *1000 s

# Renaming the '2019_population' to 'population'
df_final.rename(columns={'2019_population':'population (thousands)'}, inplace=True)

# Viewing the final dataframe
df_final
```

Out[34]:

	country	people_fully_vaccinated	total_vaccinations	vaccines	population (thousands)	
0	Cambodia	NaN	8171.0	Sinopharm/Beijing	16487	2
1	Indonesia	825650.0	2022788.0	Sinovac	270626	111
2	Myanmar	NaN	82823.0	Oxford/AstraZeneca	54045	7
3	Singapore	110000.0	329630.0	Pfizer/BioNTech	5804	37
4	Brunei Darussalam	NaN	NaN	NaN	433	1
5	Lao PDR	NaN	NaN	NaN	7169	1
6	Malaysia	NaN	NaN	NaN	31950	36
7	Philippines	NaN	NaN	NaN	108117	37
8	Thailand	NaN	NaN	NaN	69626	54
9	Timor-Leste	NaN	NaN	NaN	1293	
10	Vietnam	NaN	NaN	NaN	96462	26

Task 8: Statistical Description of final data from DataFrame

Let us use basic statistics (the 5 number summary & additional 2 metrics) and intepret the data.

In [35]:

```
df_final.describe().apply(lambda s: s.apply('{0:.4f}'.format))
```

Out[35]:

	people_fully_vaccinated	total_vaccinations	population (thousands)	GDP	perCapitaGDP
count	2.0000	4.0000	11.0000	11.0000	11.0000
mean	467825.0000	610853.0000	60182.9091	288619.5455	0.0120
std	506040.9680	951260.2332	79711.3749	334796.0303	0.0193
min	110000.0000	8171.0000	433.0000	1674.0000	0.0013
25%	288912.5000	64160.0000	6486.5000	22631.5000	0.0021
50%	467825.0000	206226.5000	31950.0000	261921.0000	0.0035
75%	646737.5000	752919.5000	83044.0000	374429.5000	0.0096
max	825650.0000	2022788.0000	270626.0000	1119191.0000	0.0641

Analysis:

Measures of Centrality: Mean & Median Values

1) The mean people_fully_vaccinated over the 2 countries that have values for them is 467825. While the mean for total_vaccinations is 610853 over 4 countries that have values for them.

- From this analysis of the mean, note how the total_vaccination is higher than the people_fully_vaccinated 's mean. From my understanding and knowledge, this phenomenon can be explained as such:
- A person, depending on the immunization scheme, will receive one or more (typically 2) vaccines. Thus, at a certain moment, the number of vaccination might be larger than the number of people who have been vaccinated. Thus, we can see that the mean total_vaccinations is higher than the mean people_fully_vaccinated here.

2) The mean people_fully_vaccinated is equal to the median. Therefore, we can conclude that the distribution for people_fully_vaccinated with the current 2 countries' values is most likely symmetric and the distribution has zero skewness.

3) The mean total_vaccinations is greater than the median, thus we can conclude that the distribution for people_fully_vaccinated is most likely positively skewed.

2) The mean population is 60182 and the median is 31950.

- This mean helps us gauge the estimated population size of a southeast asian country. However, we can see that there is a huge difference of 270193 between the maximum population (indonesia - 270626) and the minimum population (Brunei - 433). Thus, we will have to analyse spread later to better analyse this phenonmenon.
- Also, the mean population is greater than the median, thus we can conclude that the distribution for population is most likely positively skewed.

3) The mean GDP is 288619 dollars, while median GDP = 261921 dollars. From this, we can see that the GDP mean higher than the median, can so we can expect a somewhat positive skew distribution for GDP.

4) The mean perCapitaGDP is 0.0120 dollars. The median is 0.0035. We can see that the perCapitaGDP's distrubution would also be a positive skew, as the mean > median.

Measure of Spread: Interquartile Range

Interquartile range tells us how far apart the first and third quartile are and so indicates how spread out the middle 50% of our set of data is. It can also be useful in identifying when a value is an outlier.

1) The IQR for people_fully_vaccinated & total_vaccinations is 357825 & 688759 respectively. This tells us that it's more likely that total_vaccinations will vary much as compared to people_fully_vaccinated.

- For total_vaccinations, the maximum (2022788-Indonesia) is much much higher than the 3rd quartile (752919). We can observe that this might be an outlier and let us see why that is so. One possible implication causing this is the fact that Indonesia has a very high population compared to other countries, thus would directly correlate into having more vaccinations and people being vaccinated.

2) The IQR for population is 76558.

- We can observe that the max population (270626-Indonesia) is much higher than the 75th percentile (or 3rd quartile), and thus this can be interpreted as an outlier - since an outlier is an observation that lies an abnormal distance from other values. However, this outlier should not be considered abnormal as it is expected that population values will vary alot.

3) The IQR for GDP is 351798 dollars & the IQR for perCapitaGDP is 0.0075 dollars.

- Thus, we can observe that max GDP is much higher than the 3rd quartile, and this is expected as this maximum is from Indonesia, which has the highest population size.
- For perCapitaGDP, the max perCapitaGDP is much higher than the 3rd quartile at 0.0641 dollars. This shows us that Singapore, although not as big a country as Indonesia, has been able to have decent GDP for their population size in order to have such as high perCapitaGDP.

Measure of Spread: Standard Deviation

Note: A low standard deviation indicates that the data points tend to be very close to the mean; a high standard deviation indicates that the data points are spread out over a large range of values.

1) For people_fully_vaccinated:

- The standard deviation is very high, much higher than total_vaccination's standard deviation. This implies that the data points are very much spread out over a very large range of values. To be fair, there were only 2 values accounted for for people_fully_vaccinated, thus this might not give an accurate representation of the true spread for this data.

2) For total_vaccinations

- The spread is also high. However, do note like for people_fully_vaccinated, there were only 4 values accounted for for total_vaccinations, thus this might not give an accurate representation of the true spread for this data.

3) For population

- The standard deviation is also high at 79711, meaning that the data points are spread out over a large range. This is expected as there are very hugely varying sizes for southeast asian countries.

4) For GDP and perCapitaGDP, the standard deviation is 334796 & 0.0193 repectively.

- For GDP & perCapitaGDP, the standard deviation is higher then the mean itself, so the spread is huge.

5) For all columns, the SD > mean thus this implies that the data is very widely distributed with a strong positive skewness and would possibly have outliers within the data.

- However, let us remember that these values come from countries of different and varying economical level and size, along with mssing values for more than a few countries. Therefore it is expected for the spread to be large.

Phase iii: Graph Plotting & Insights

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

Then, we will import **pyplot, matplotlib's plotting framework.** Matplotlib is a plotting library for the Python programming language. We import the module "matplotlib.pyplot" and bind that to the name "plt". This framework helps us to better show and visualise our data.

Also, we will import **ticker, matplotlib's ticker framework.** This ticker framework is designed in order to manipulate the ticks being shown (eg: Ticks being used as the values used to show specific points on the coordinate axis). A tick can be a number or a string.

We will also import **floor, a function from the math module**, to be used to round values down when needed.

Next we have use **%matplotlib inline**, so that the output of plotting commands can be displayed inline within frontends like the Jupyter notebook, directly below the code cell that produced it.

In [36]:

```
# import libraries
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from math import floor
%matplotlib inline
```

Question 1

Each country currently may have only 1 vaccine being used but some will be more than 1 vaccine type. If there are more than 1 vaccine type, assume that it will be equally distributed to the country's population. With this in mind, how would you visualise the estimated number of people vaccinated for each vaccine type for the South East Asian population?

In my own interpretation, this question is telling us to find out for each vaccine type: 'how many people **would be** vaccinated for each vaccine if the vaccines are equally distributed to the country's population?'. Thus, for this reason via my own interpretation of the question, I will use **population** values to determine the estimated number of people.

To plot the number of south east asian vaccinated with a specific vaccine against the vaccine types, my approach is to:

1. **Create a new DataFrame from the df_final data (from phase ii) just for this question, to plot the required chart.**

Note: Our expected DataFrame for question 1 should have, **'Vaccine Type'** & **'Estimated number of people vaccinated'** with the specific vaccine.

Let us start by getting the data for the 'Vaccine Type' column:

To get the values for the Vaccine Type column's values, my approach is to:

- Iterate through the column values of the df_final 'vaccines' column and get the final list of vaccines for the Vaccine Type column for the chart's DataFrame.
- The cell below does this by looping through every row of df_final['vaccines'] and splitting the multiple vaccines by ',' (since some values will have multiple vaccines mentioned separated by ',') and adding the vaccines to the vaccines_list.
- We use a looping approach so as to ensure this would work for larger sets of data. (eg: if we want to get vaccines across **all** not just southeast asian countries)
- After the loop, we will remove duplicates of vaccines using the inbuilt set() function, then put this back into a list with list().

In [37]:

```
vaccines_list = []
for x in df_final['vaccines']:
    if type(x) != str: break
    else: vaccines_list += [x]
vaccines_list = list(set(vaccines_list))
vaccines_list
```

Out[37]:

```
['Sinopharm/Beijing', 'Pfizer/BioNTech', 'Sinovac', 'Oxford/AstraZeneca']
```

Now that we have the list for the Vaccine Type column, the cell below creates the frame that will be used for Q1's plotting. For the 'Number of people vaccinated' column, we will intialise the values as 0 first.

In [38]:

```
# create dataframe for bar chart
q1_df = pd.DataFrame({
    'Vaccine Type' : vaccines_list,
    'Estimated number of people vaccinated' : [0]*len(vaccines_list)
})
q1_df
```

Out[38]:

	Vaccine Type	Estimated number of people vaccinated
0	Sinopharm/Beijing	0
1	Pfizer/BioNTech	0
2	Sinovac	0
3	Oxford/AstraZeneca	0

Now, let us find the values for the 'Number of People Vaccinated' column. We will create a **helper** dataframe called vc_population to extract the 'population (thousands)' & 'vaccines' column from final_DF. With this helper dataframe, we can then use the values to accurately find out the number of people vaccinated with each vaccine.

In [39]:

```
vc_population = df_final.iloc[:, [3, 4]]
vc_population
```

Out[39]:

	vaccines	population (thousands)
0	Sinopharm/Beijing	16487
1	Sinovac	270626
2	Oxford/AstraZeneca	54045
3	Pfizer/BioNTech	5804
4	NaN	433
5	NaN	7169
6	NaN	31950
7	NaN	108117
8	NaN	69626
9	NaN	1293
10	NaN	96462

With this **helper** dataframe, we will iterate through each row with .iterrows(). For each row, the code below splits the vaccine values by '/', since each column vaccine value either has one vaccine or multiple vaccines split by '/'. And it accumulates the value in the total_vaccinations list for the vaccine while going through each row.

Note that if there are more than 1 vaccine type, we assume that it will be equally distributed to the country's population, thus we **divide the population of that country** by the **number of vaccines used in the country** to find out -> **the estimated number of people vaccinated with each vaccine for that country**. This number of people vaccinated with each vaccine for that country will then be accumulated onto the 'number of people vaccinated' value for that vaccine in the 'q1_df' dataframe.

In [40]:

```
for index, row in vc_population.iterrows():    # for each row in the DataFrame
    if type(row['vaccines']) != str:                # if the vaccine column for that row
        break
    vcs = row['vaccines'].split(',')            # vcs = is the list of vaccines that the cur
    pop = row['population (thousands)']*1000   # pop = is the value of the populations colu
    dist_num_ppl = pop/len(vcs)  # This is the distributed number of people per vaccine if
    for vc in vcs:                              # for each vaccine type found
        i = vaccines_list.index(vc)             # index of the vaccine type found within the
        q1_df.iloc[i,1] = q1_df.iloc[i,1]  + dist_num_ppl  # accumulate the current value f
q1_df
```

Out[40]:

	Vaccine Type	Estimated number of people vaccinated
0	Sinopharm/Beijing	16487000.0
1	Pfizer/BioNTech	5804000.0
2	Sinovac	270626000.0
3	Oxford/AstraZeneca	54045000.0

Now, we have our DataFrame ready to be plotted into the desired chart type: **a Bar Chart.** We use a bar chart as we are looking at the relationship between a categorical data type and a numeric type. This helps us to identify patterns and compare values between each category.

However, before we plot the data, let us include an additional metric: **Percentage of People vaccinated per vaccine** out of the **total number of people**. This is computed just so that it will be easier for us to understand the raw number of people values and the % difference between the different vaccines, and these values will be utilised in just the **labelling** of each bar.

In the cell below, we carry out the neccessary operations to compute the percentage as mentioned above. These operations include:

1. Finding the total number of people vaccinated with the sum() function.
2. Creating a new list 'percentages' filled with the number of people vaccinated/total*100 for each row. (rounded to 2 dec places & changed to String type -> explained in step 3.).
3. Creating a new column 'Percentage Out Of Total Vaccinated' to include String values of 'Number of people vaccinated', concatenated with String values in 'percentages' enclosed within round brackets.(Eg: '4000 (45%)') We want it in this format and data type so that it can be used for labelling later. To do this, we use functions such as .round(), floor() and astype(str), to round values and to change column value types to String types.

In [41]:

```
# total = total number of people vaccinated
total = sum(q1_df['Estimated number of people vaccinated'])

# Create a new 'percentages' list, round to 2 decimal places, and change to String type.
percentages = (q1_df['Estimated number of people vaccinated']/total*100).round(2).astype(st

# Utilise values from percentages and concatenate them within brackets with the 'Number of
q1_df['Percentage out of Total Vaccinated'] = q1_df['Estimated number of people vaccinated'
q1_df
```

Out[41]:

	Vaccine Type	Estimated number of people vaccinated	Percentage out of Total Vaccinated
0	Sinopharm/Beijing	16487000.0	16487000 (4.75%)
1	Pfizer/BioNTech	5804000.0	5804000 (1.67%)
2	Sinovac	270626000.0	270626000 (78.0%)
3	Oxford/AstraZeneca	54045000.0	54045000 (15.58%)

Thus, now we have **all** adequate information needed (x values, y values, label values) to plot the Q1 bar chart.

Plotting Question 1's graph

My approach here is to:

1) Use pd.plot to plot the axes. This helps create the axes of the plot, we can specify the type of the plot like so: pd.plot.bar().

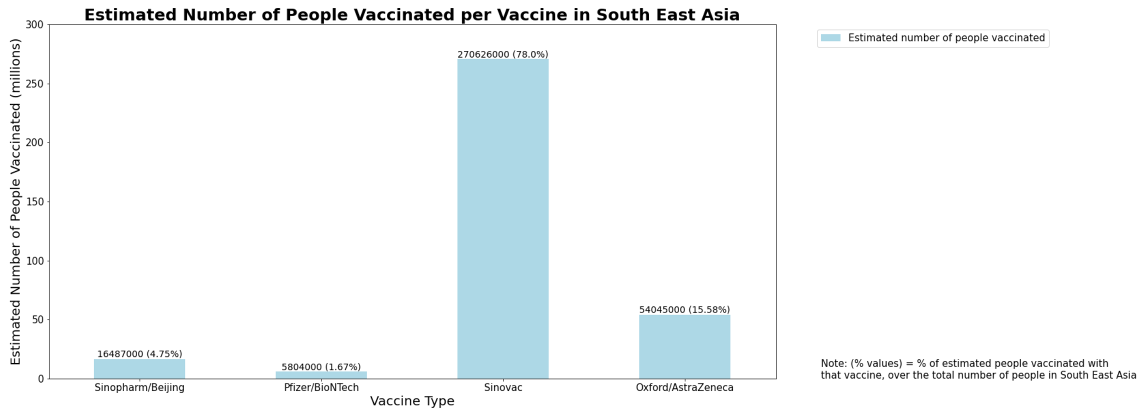
2) Use plt.title, plt.xlabel, plt.ylabel and other functions from the pyplot framework in order to label and add headers to our plot.

3) Use the ticker framework as well to format the tickers (for y axis).

4) To set the labels, I use a loop to loop though each bar in the plot and use ax.text() to add a text label above each bar. (Mckinney, 2020)

In [42]:

```python
# Make a plot of the q1_df DataFrame, name this plot as ax.
# - An ndarray is returned by q1_df.plot.bar with one matplotlib.axes.Axes per column
# - figsize sets size of the plot
ax = q1_df.plot.bar(figsize=(20, 10), color="lightblue")  # set bar colour to light blue.

# Modify the plot title
plt.title('Estimated Number of People Vaccinated per Vaccine in South East Asia', fontsize=

# Modify the x-label title
plt.xlabel("Vaccine Type", fontsize=20)

# Modify the x-ticklabels for the plot with .set_xticklabels(), since ax is already a matpl
ax.set_xticklabels(q1_df['Vaccine Type'], rotation = 0, fontsize=15) # We will use vaccine

# Modify the y-label title
plt.ylabel('Estimated Number of People Vaccinated (millions)', fontsize=20)

# Modify the y-axis limit (to reduce unwanted space above bars)
plt.ylim([0, 300*1e6])

# Modify yticklabels:
# # Use FuncFormatter from the matplotlib ticker framework. This is so that we can format t
# # it shows labels in Millions, rather than in such large numbers that might be hard for t
# # The scale_y here represents by which scale I want to divide the column values by, which
scale_y = 1e6   # scale = millions
formatter = ticker.FuncFormatter(lambda x, pos: '{0:g}'.format(x/scale_y)) # define my own
# Format the axis's y-axis major ticks with .set_major_formatter() & the formatter.
ax.yaxis.set_major_formatter(formatter)
ax.tick_params(labelsize=15)

# Create list for each bar's Value Labels
labels = q1_df['Percentage out of Total Vaccinated'] # to show in percentages
rects = ax.patches  # A patch is a 2D artist with a face color and an edge color, thus each

# Make some labels.
for rect, label in zip(rects, labels):  # for each patch (bar) and each label, since they c
    height = rect.get_height()   # gets the height of the patch (bar)
    ax.text(rect.get_x() + rect.get_width() / 2, height + 5, label, ha='center', va='bottom

# Put legend outside of the plot.
ax.legend(bbox_to_anchor=(1.05, 1), loc='upper left', fontsize=15)

# Set background colour
ax.set_facecolor('white')

# Annotate: to explain the % used for labelling.
ax.annotate('Note: (% values) = % of estimated people vaccinated with \nthat vaccine, over

# Using plt.show() to show the figure & plot.
plt.show()
```



Insight and Analysis. (although it may be straight forward and easily understood from the visualisation)

From the bar chart above, there are a few key points to take note on and analyze:

1. Sinovac has the highest amount of people that would be vaccinated with it, at a whopping 78% of the total south east asian population. This main reason as to why this is so is because:
    - Sinovac is only used in Indonesia, and Indonesia's population is the highest out of all other south east asian countries. Thus, this explains the high estimated percentage of people vaccinated with Sinovac in the bar chart.
2. Oxford/AstraZeneca is next highest, estimated to be vaccinated by 15.58% of the total south east asian population.
    - This vaccine was used only by Myanmar. This vaccine has the highest estimated number of people vaccinated after Sinovac because Myanmar's population is lower than Indonesia's but higher than the rest of the countries using vaccines.
3. Folllowing AstraZeneca, Sinopharm/Beijing is the next highest, estimated to be vaccinated by 4.75% of the total south east asian population.
    - This vaccine was used only by Cambodia. The reason why this vaccine has a lower number of estimated people is because Cambodia's population is smaller than Myanmar & Indonesia.
4. Lastly, Pfizer/BioNTech is the lowest, estimated to be vaccinated by 1.67% of the total south east asian population.
    - This is because Pfizer/BioNTech was only used Singapore, and thus the reason why these vaccines have the lowest number of people is because Singapore's population is much much smaller than the other countries who use vaccines.

Therefore, from these points, we can clearly observe that there is a correlation between the country (where the vaccine is being used)'s population and the estimated number of people vaccinated with the vaccine.

The higher the population in a country where the vaccine is being used, the higher the estimated number of people vaccinated with that vaccine.

QUESTION 2

For each of the country, plot a bar graph, with side-by-side bars for population, total vaccinations, and

people_fully_vaccinated. There are two challenges here, firstly, the default graph will be difficult to visualise due to large differences in the numbers, and secondly, this information may not give a good visualsation. These 2 challenges are for you to figure out, make the appropriate code changes for the visualisation, and be able to explain why is the data used for the graph may be misleading (some general knowledge / domain needed).

To plot the population, total vaccinations, and people_fully_vaccinated by country, my approach is to:

- Create a new DataFrame from the df_final data (from phase ii) just for this question, to plot the required chart. Note: Our expected DataFrame for question 2 should have, **'country' as the x-axis & 'people_fully_vaccinated', 'total_vaccinations', 'population' for the y-axis.**

Create q2_df DataFrame for question 2's plot

I will keep the NaN values as NaN values, and won't change them to 0, in order to prevent the reader from being confused as to why they might be 0. For now, please assume that the NaN values are missing data.

In [43]:

```python
# iloc to get specfic columns
q2_df = df_final.iloc[:, [0, 1, 2, 4]]

# The values in the dataframe were represented in thousands, so we want to convert them bac
q2_df.iloc[:, 3] = q2_df[['population (thousands)']].apply(lambda x: x*1000)

# Renaming the columns
q2_df.columns=['country', 'people_fully_vaccinated', 'total_vaccinations', 'population']
q2_df
```

```
C:\Users\AaronKhoo\anaconda3\lib\site-packages\pandas\core\indexing.py:966:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  self.obj[item] = s
```

Out[43]:

	country	people_fully_vaccinated	total_vaccinations	population
0	Cambodia	NaN	8171.0	16487000
1	Indonesia	825650.0	2022788.0	270626000
2	Myanmar	NaN	82823.0	54045000
3	Singapore	110000.0	329630.0	5804000
4	Brunei Darussalam	NaN	NaN	433000
5	Lao PDR	NaN	NaN	7169000
6	Malaysia	NaN	NaN	31950000
7	Philippines	NaN	NaN	108117000
8	Thailand	NaN	NaN	69626000
9	Timor-Leste	NaN	NaN	1293000
10	Vietnam	NaN	NaN	96462000

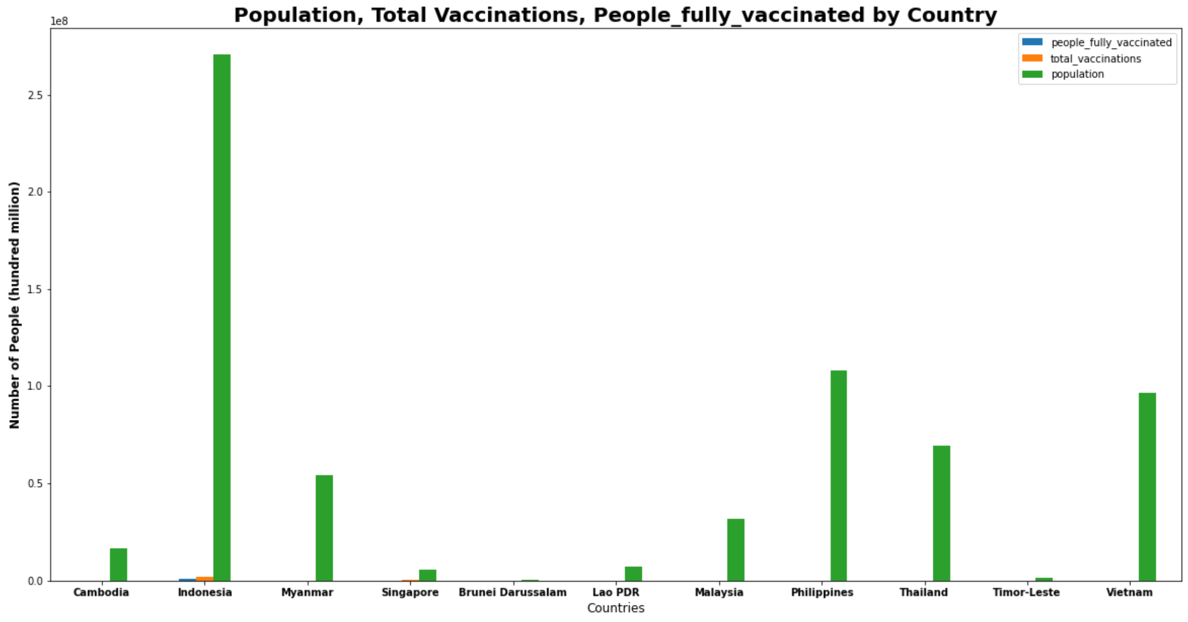
Plotting the graph.

As mentioned in the assignment specifications, the graph shown below will not be in the best in terms of visualisation. Thus it is expected that we should plot another one.

In [44]:

```
fig = plt.figure()
ax = q2_df.plot.bar(figsize=(20, 10))

# title
plt.title('Population, Total Vaccinations, People_fully_vaccinated by Country', fontsize=20

# x-label
plt.xlabel("Countries", fontsize=12)
ax.set_xticklabels(q2_df['country'], rotation=0, fontsize=10, fontweight='bold') # xticklab

# y-label
ax.set_ylabel('Number of People (hundred million)', fontsize=12, fontweight='bold')

plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



Plotting a better chart that improves the first chart

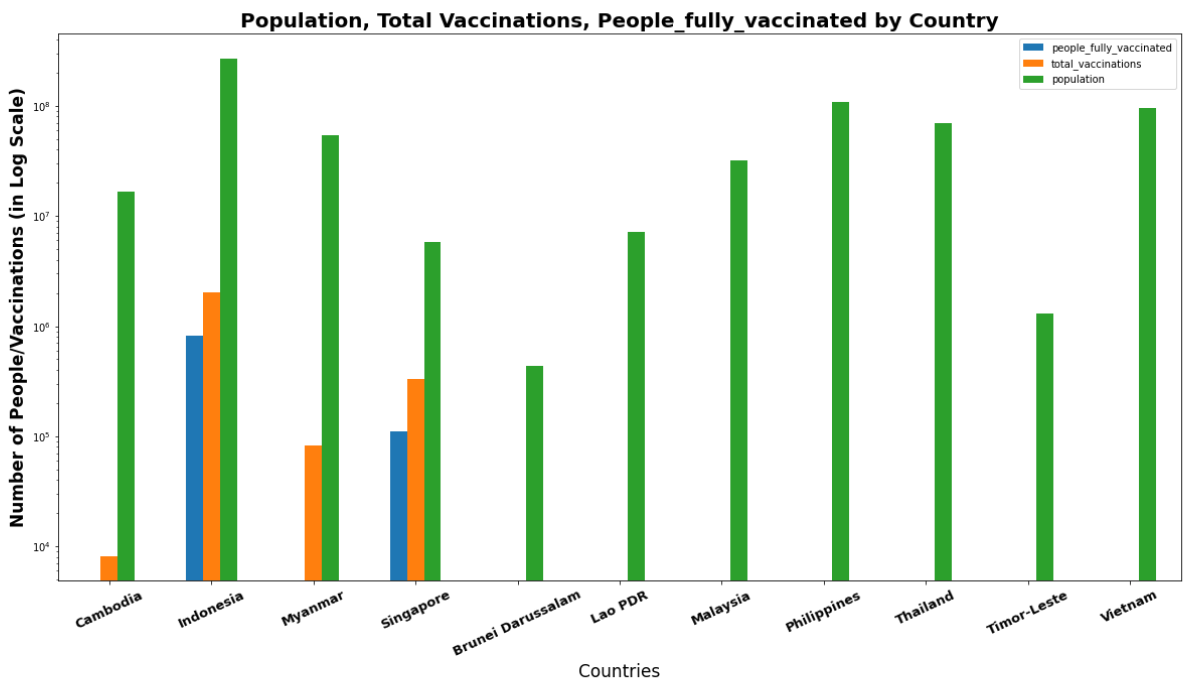
My approach is to convert the y-axis's scale to be in **log** scale. This would help make the following improvements:

- Previously we can observe that the values vary immensely and it is hard to see the bars for the smaller values. Now, with a log scale, we are able to **display numerical data over a very wide range of values in a compact manner.**
- Previously, it was hard for us to visualise the large discrepancies. Now, we can **visualize between large descrepancies of values on a single axis.**

In [45]:

```
# Better Chart!
fig2 = plt.figure()
ax2 = q2_df.plot.bar(figsize=(20, 10))

# title
plt.title('Population, Total Vaccinations, People_fully_vaccinated by Country', fontsize=20

# x-label
plt.xlabel("Countries", fontsize=17)
ax2.set_xticklabels(q2_df['country'], rotation=25, fontsize=13, fontweight='bold') # xtickl

# y-label
ax2.set_ylabel('Number of People/Vaccinations (in Log Scale)', fontsize=17, fontweight='bol
# set y-axis's scale as log
ax2.set_yscale('log')

plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



Question 2: Provide some basic insights & explain why the data used for the graph may be misleading. (some general knowledge / domain needed).

From the chart above, there a few key points & relevants insights we can draw:

(1) For countries that have values for people_fully_vaccinated, the total_vaccinations bars are higher than the people_fully_vaccinated bar. This can be explained as such with regards to the definition of a person who is **fully vaccinated:**

People are considered fully vaccinated: 1) 2 weeks after their second dose in a 2-dose series, such as the Pfizer or Moderna vaccines, or 2) 2 weeks after a single-dose vaccine, such as Johnson & Johnson's Janssen vaccine. (CDC, 2021)

Therefore, a person must take >= 1 vaccine and only after 2 weeks/longer will they be considered fully vaccinated. Thus, this explains why the number of people fully vaccinated would be lower than the current total occuring vaccinations. This was also explained from the description of the Kaggle dataset's link page provided. (Preda, 2021)

(2) Note that only 4 countries have values for total_vaccinations & people_fully vaccinated. This could be because only these 4 countries have implemented vaccine programmes before 24/2/2021 (the last date for the data in Vaccinations dataset). From external resources, we can find this fact to be true:

- Malaysia only started their vaccine programme on 26 February (Kwan, 2021)
- Brunei has yet to start their vaccine programme (Brunei MOH, 2021)
- Lao PDR started their programme in March (WHO, 2021)
- Phillippines started their programme in March (Tomacruz, 2021)
- Thailand started their programme on 28 Feb (Nyugen, 2021)
- Timor-Leste has yet to start their programme (Timor-Leste MOF, 2021)
- Vietnam started their programme in March (Nikkei Asia, 2021)

Therefore, there shouldn't seem to be any missing vaccination values in the vaccinations dataset for countries that have not implemented vaccinations yet.

(3) Indonesia is has the highest total_vaccinations & people_fully_vaccinated for 2 reasons:

- They started phase 1 of their vaccination programme right off the bat with large numbers targeting 1.5 million medical workers (New Straits Times, 2021) frontline workers and textile factory workers. Therefore, their initial vaccinations from phase 1 till end february (which was the start of phase 2) targeted a huge amount of people. They are continuing to aim for large numbers in phase 2 & 3, with phase 2's target to vaccinate at least 38.5 million people. (Soeriaatmadja, 2021)
- They have the biggest population in south east asia.

(4) Singapore & Indonesia has the highest total_vaccination values:

- They were the earliest to start vaccinations: Singapore started vaccinations on 30/12/2020, Indonesia 15/1/2021, Myanmar 27/1/2021 and Cambodia on 10/2/2021. Note how Singapore started the earliest; this will shed some light on the following point (5).

(5) Singapore, managed to be 2nd highest in total_vaccinations & people_fully_vaccinated despite having a much lower population compared to Cambodia, Indonesia & Myanmar. This could likely be due to:

- The **high efficiency and goal set by Singapore**, that is to have enough vaccines for all 5.7 million people in the city by the third quarter of 2021, with the voluntary vaccine free for all Singaporeans and long-term residents. This goal of theirs has led to them to purchase vaccines beforehand, with maximum cautions on the vaccines to be picked, to prepare for its vaccination programme.(Khalik, 2021) Therefore, their efficiency in purchasing vaccines and going ahead with the vaccination programme very early on is a good take as the reason why their vaccination numbers are high as compared to the other countries who have a larger population (except for Indonesia). (AFP, 2020)
- They **started vaccinations the earliest** out of all South East Asian countries in December 2020, as mentioned in point(4).
- **perCapitaGDP of Singapore is the highest** out of all other South East Asian countries (refering to the df_final dataframe constructed in phase ii). Thus, this implies that a high perCapitaGDP is associated with a higher number of vaccinations. We will not be looking into this potential correlation for this assignment.

(6) To further corroborate my findings from point 5, let us find out the accurate **total_vaccinations:population ratio** for each country with the cell below:

In [46]:

```
ratio = q2_df['total_vaccinations']/q2_df['population']
q2_df.insert(4, 'vaccinations:population (ratio)', ratio, True)
q2_df
```

Out[46]:

	country	people_fully_vaccinated	total_vaccinations	population	vaccinations:population (ratio)
0	Cambodia	NaN	8171.0	16487000	0.000496
1	Indonesia	825650.0	2022788.0	270626000	0.007474
2	Myanmar	NaN	82823.0	54045000	0.001532
3	Singapore	110000.0	329630.0	5804000	0.056794
4	Brunei Darussalam	NaN	NaN	433000	NaN
5	Lao PDR	NaN	NaN	7169000	NaN
6	Malaysia	NaN	NaN	31950000	NaN
7	Philippines	NaN	NaN	108117000	NaN
8	Thailand	NaN	NaN	69626000	NaN
9	Timor-Leste	NaN	NaN	1293000	NaN
10	Vietnam	NaN	NaN	96462000	NaN

(6 cont.) Thus, from the dataframe shown above, we can see that the vaccinations:population ratio for Singapore is the highest at 0.056794. This shows how Singapore is on the lead in terms of making sure much of their population gets vaccinated.

(7) Another important thing to note is **how the data used for the graph may be misleading**. The reason for this is because:

1. A lot of data is left out. The analysis we drew from the graph is mainly done on the four south east asian countries that have implemented the COVID vaccination programmes then. Therefore, the insight drawn from the graph may not faithfully reflect the situation in South East Asia, and thus analysis done might be shorthanded and could be proven wrong when more data is available.
2. The scale used for the y-axis is enlarged with log values. Arguably, this might cause viewers to think that the differences between the values may be small, rather than what it actually is, which are big differences. However, the reason why I put it in log scale in the first place was explained earlier, which are reasonable reasons that I am repeating here:

    - to display numerical data over a very wide range of values in a compact manner
    - to visualize between large descrepancies of values on a single axis

QUESTION 3

For the final question, you will probably need the non-aggregated data from "Vaccination.csv". You are to extract the data that's related only to Singapore and then plot a line graph on the daily_vaccinations over time. Like earlier, you are to discard the original total_vaccinations and create the total vaccinations for each day using the cumulative sum of the daily vaccinations (up to that day). Again,

plot a line graph to visualise the cumulative vaccinations over time. Explain in what circumstances would the first line graph be useful (if at all) and in what circumstances that the second (cumulative) line graph would be useful?

Plotting of line graph 1 for question 3

To plot the **first** line graph on the daily_vaccinations over time for Singapore, my approach is to:

- Create a new DataFrame from the vaccinations dataset just for this question, to plot the required chart. Note: Our expected DataFrame for question 3 should have, 'date' & 'daily_vaccinations' columns. We will name this dataframe q3_df.
- Import matplotlib.dates's DateFormatter function as we might need to use them when accounting for labels

The cell below first reads the vaccinations dataset again into a dataframe.

In [47]:

```
vc = pd.read_csv("C:\\Users\\Aaron Khoo\\Documents\\Vanessa\\1_Monash\\Y1S2\\FIT1043\\Assig
```

In [48]:

```
# Importing libraries
from matplotlib.dates import DateFormatter
```

Now, let us create the DataFrame for the plotting of the first line. I will put NaN values as 0 it is assumed that the NaN values illustrate that there were not any vaccinations that day.

In [52]:

```
cols = ['date']
vc[cols] = vc[cols].apply(pd.to_datetime, errors='coerce')
q3_df = vc.loc[vc['country'] == 'Singapore'].reset_index()
q3_df = q3_df.iloc[:, [1, 3, 4, 7, 8]]
q3_df.head()
```

Out[52]:

	country	date	total_vaccinations	daily_vaccinations_raw	daily_vaccinations
0	Singapore	2021-01-11	3400.0	NaN	NaN
1	Singapore	2021-01-12	6200.0	2800.0	2800.0
2	Singapore	2021-01-13	NaN	NaN	4090.0
3	Singapore	2021-01-14	NaN	NaN	4520.0
4	Singapore	2021-01-15	NaN	NaN	4735.0

In [53]:

```
# Update the dataframe to remove the country column, I will assu
q3_df = q3_df.iloc[:, [1, 4]].fillna(0)
q3_df.head()
```

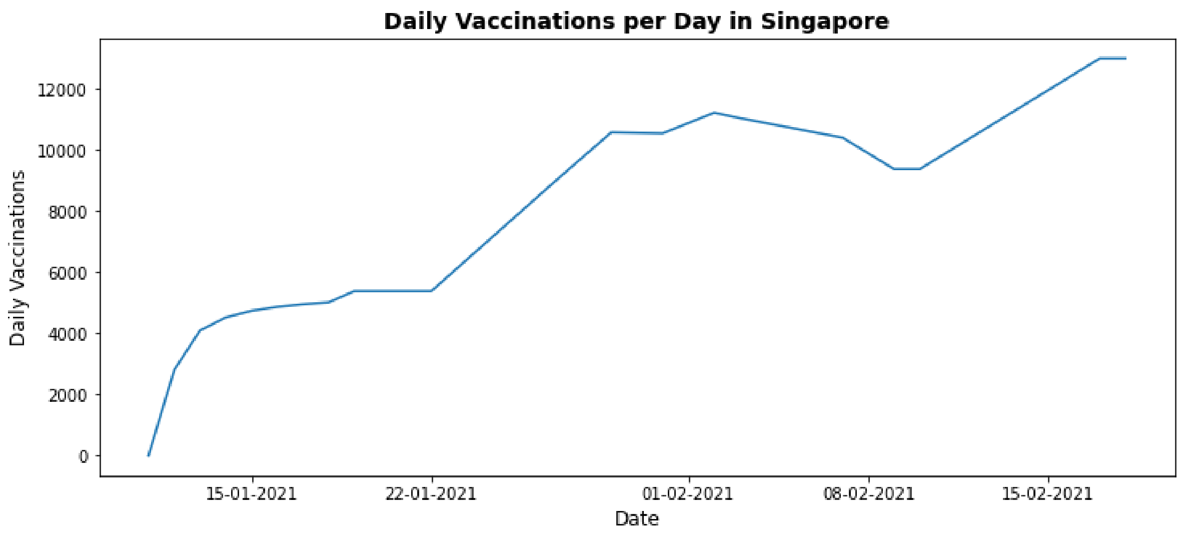
Out[53]:

	date	daily_vaccinations
0	2021-01-11	0.0
1	2021-01-12	2800.0
2	2021-01-13	4090.0
3	2021-01-14	4520.0
4	2021-01-15	4735.0

Plotting of line graph 1 for question 3

In [55]:

```
# plot graph
plt.figure(figsize=(12,5))
plt.plot(q3_df['date'].astype('datetime64[ns]'), q3_df['daily_vaccinations'])

# Set title
plt.title('Daily Vaccinations per Day in Singapore',fontsize=14, fontweight='bold')

# Set x-label
plt.xlabel('Date', fontsize=12)

# Set y-label
plt.ylabel('Daily Vaccinations', fontsize=12)

# Define formatter function for dates
formatter = DateFormatter('%d-%m-%Y')
# Set format for the xaxis's labels
plt.gcf().axes[0].xaxis.set_major_formatter(formatter)

# Show plot
plt.show()
```



Plotting of line graph 2 for question 3

To plot the **second** line graph on the daily_vaccinations (accumulated) over time for Singapore, my approach is to:

- Create a new DataFrame from the vaccinations dataset just for this question, to plot the required chart.
- Iterate over the rows of the q3_df used for the previous plot, with the function iterrows(), which returns a tuple with the column name and content in form of series. This iteration will help me to accumulate the data I need for the accumulated daily vaccinations required for this second line graph.

Note: Our expected DataFrame for question 3 should have, 'date' & 'daily_vaccinations' (accumulated) columns. We will name this dataframe q3b_df.

In [56]:

```
q3b_df = q3_df

# Iterate over the rows with for loop
for index, row in q3b_df.iterrows():
    if index < q3b_df.shape[0]-1:
        prev = q3b_df.iloc[index, 1]
        curr = q3b_df.iloc[index+1, 1]
        q3b_df.iloc[index+1, 1] = prev+curr  # accumulating previous accumulated value with
q3b_df.head()
```

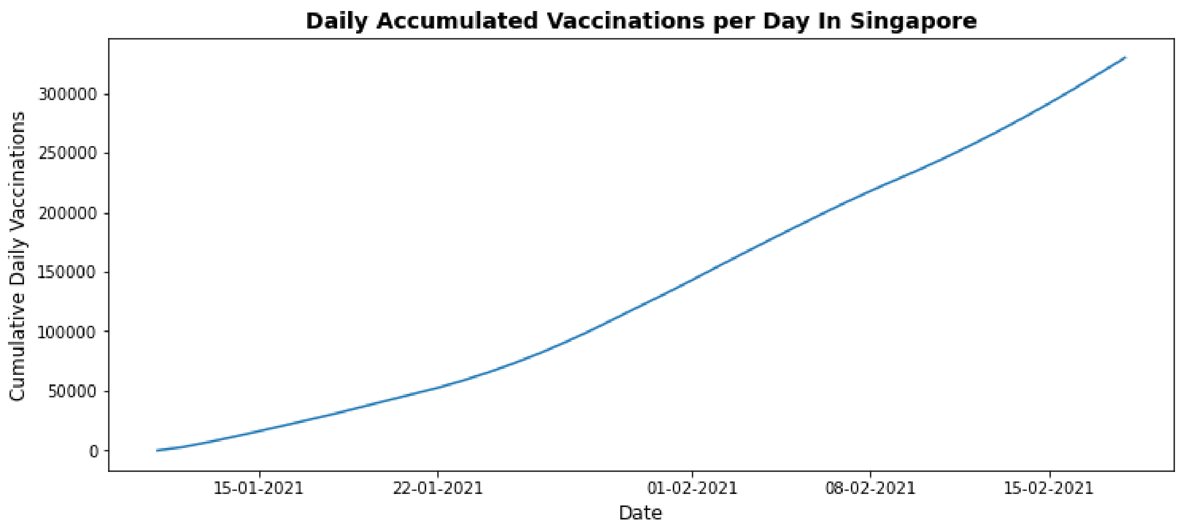
Out[56]:

	date	daily_vaccinations
0	2021-01-11	0.0
1	2021-01-12	2800.0
2	2021-01-13	6890.0
3	2021-01-14	11410.0
4	2021-01-15	16145.0

Plotting of line graph 2 for question 3

In [57]:

```
# Plot graph
plt.figure(figsize=(12,5))
plt.plot(q3b_df['date'], q3b_df['daily_vaccinations'])

# Set title
plt.title('Daily Accumulated Vaccinations per Day In Singapore',fontsize=14, fontweight='bo

# Set x-label
plt.xlabel('Date',fontsize=12)

# Set y-label
plt.ylabel('Cumulative Daily Vaccinations',fontsize=12)

# Define formatter function for dates
formatter = DateFormatter('%d-%m-%Y')
# Set format for the xaxis's labels
plt.gcf().axes[0].xaxis.set_major_formatter(formatter)

# Show plot
plt.show()
```



Question 3: Explain in what circumstances would the first line graph be useful (if at all) and in what circumstances that the second (cumulative) line graph would be useful?

The first graph can be useful when:

1) We wish to find the individual values per day during the time period, which helps us to identify the **minimum** and **maximum** values and on which dates those happened during the time period.

For example, if there are drastic discrepancies between the usual values with the min or max, analysing this kind of graph could help us further identify why there was a minimum or maximum on that certain day and if something might have happened during the period for such a huge discrepancy to happen. Thus, it can then help us to identify the outliers and what might have caused them during the time period.

Another example: When the rate of increase in daily accumulated vaccinations seem to drop, there might be a need for us to check back on that certain period of reduced rate with the daily vaccinations graph to see how the specific dates & daily individual vaccinations were like to help identify the root cause of the reduced rate of increase.

The second graph can be useful when:

1) Anaysing the growth of a certain variable/result since the start of a certain **time** period. This is because the cumulative graph shows us at what rate of increase or decrease that the variable is growing at.

For example, in this COVID-19 situation, using this graph to analyse the number of vaccinations up-to-date with a cumulative line graph helps us identify the specific trend of the vaccainations, thus we can answer questions like this while infering from the graph: i) Is the number of vaccinations generally increasing?, ii) When does the rate of increase in vaccinations drop?

2) This can also help in identifying the general trend and outlook of the situation during a time period.

References

AFP. (2020, December 30). Singapore begins coronavirus vaccination campaign. Retrieved from

`https://www.nst.com.my/world/world/2020/12/653175/singapore-begins-coronavirus-vaccination-campaign`

Brunei Ministry of Health. (2021). FREQUENTLY ASKED QUESTIONS ON COVID-19 VACCINES. Retrieved from

`http://www.moh.gov.bn/Shared%20Documents/COVID-19%20Vaccine/FAQs%20General%20Covid%2019%20Vaccine_04032021.pdf`

Khalik, S. (2021, February 7) How Singapore picked its Covid-19 vaccines. Retrieved from

`https://www.straitstimes.com/singapore/health/how-singapore-picked-its-covid-19-vaccines`

Kwan, F. (2021, February 26). Covid-19 vaccine to arrive on Feb 21, rollout from Feb 26. Retrieved from

`https://www.freemalaysiatoday.com/category/nation/2021/02/16/covid-19-vaccine-to-arrive-on-feb-21-rollout-from-feb-26/`

Mckinney, T. (2020, August). Adding value labels on a matplotlib bar chart. Retrieved from

`https://stackoverflow.com/questions/28931224/adding-value-labels-on-a-matplotlib-bar-chart`

New Straits Times. (2021, February 5). Jokowi witnesses vaccination of thousands of health workers. Retrieved from

`https://www.nst.com.my/world/region/2021/02/663279/jokowi-witnesses-vaccination-thousands-health-workers`

Nguyen, A. (2021, February 28). Thailand Kicks Off Covid-19 Vaccine Program With Sinovac Shots. Retrieved from

`https://www.bloomberg.com/news/articles/2021-02-28/thailand-kicks-off-covid-19-vaccine-program-with-sinovac-shots`

Nikkei Asia. (2021, March 8). Vietnam begins COVID vaccinations with AstraZeneca shots. Retrieved from https://asia.nikkei.com/Spotlight/Coronavirus/COVID-vaccines/Vietnam-begins-COVID-vaccinations-with-AstraZeneca-shots

Preda, G. (2021). COVID-19 World Vaccination Progress. Retrieved from

https://www.kaggle.com/gpreda/covid-world-vaccination-progress

Soeriaatmadja, W. (2021, February 17). Indonesia starts 2nd phase of Covid-19 vaccination drive at biggest textile market. Retrieved from

https://www.straitstimes.com/asia/se-asia/indonesia-starts-2nd-phase-of-covid-19-vaccination-drive-at-biggest-textile-market

Soeriaatmadja, W. (2021, February 18). Indonesia kicks off jabs for 38.5m people. Retrieved from

https://www.straitstimes.com/asia/se-asia/indonesia-kicks-off-jabs-for-385m-people

Ministry of Finance. (2021). Government of Timor-Leste Presents the 2021 State Budget and the National Vaccine Deployment Plan to the Development

Partners. Retrieved from https://www.mof.gov.tl/government-of-timor-leste-presents-the-2021-state-budget-and-the-national-vaccine-deployment-plan-to-the-development-partners/?lang=en

Tomacruz, S. (2021, February 4). The Philippines' 2021 COVID-19 vaccine plan. Retrieved from

https://www.rappler.com/newsbreak/iq/timeline-philippines-2021-covid-19-vaccination-plan

World Health Organisation. (2021, March 1). Lao PDR Prepares for Roll-out of COVID-19 Vaccines. Retrieved from

https://www.who.int/laos/news/detail/01-03-2021-lao-pdr-prepares-for-roll-out-of-covid-19-vaccines

CDC, (2021, March 23). When You've Been Fully Vaccinated. Retrieved from

https://www.cdc.gov/coronavirus/2019-ncov/vaccines/fully-vaccinated.html