COVID VACCINES ANALYSIS

TEAM MEMBERS

310821243012: DHANUUSH A

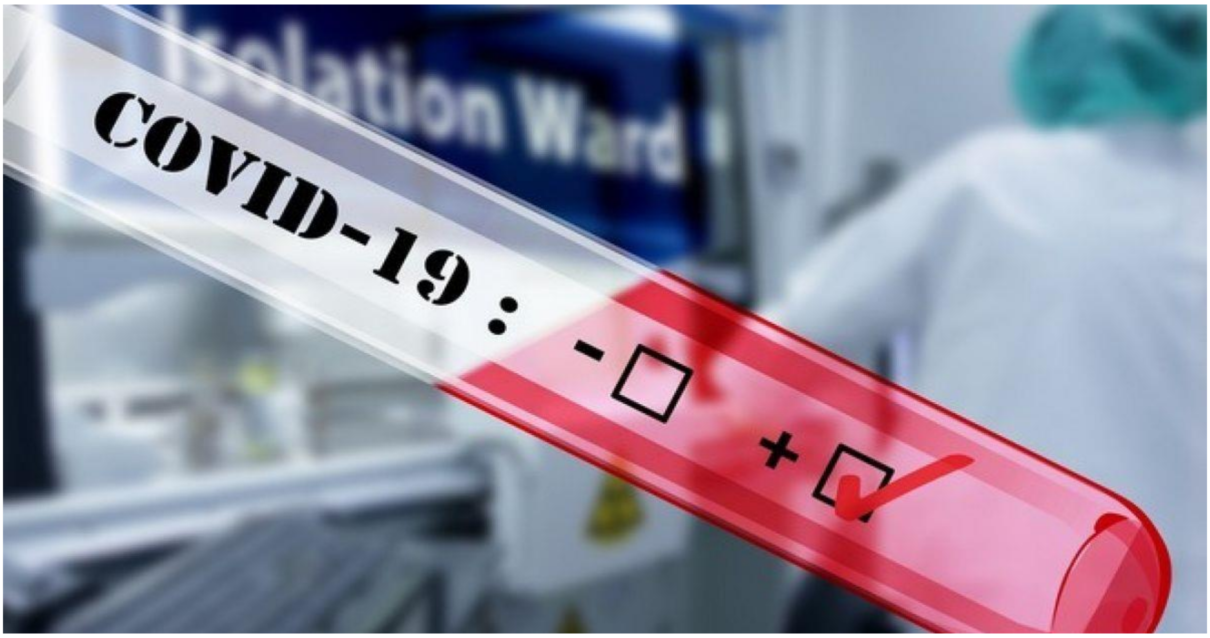
310821243048: SAI NITHISH KUMAR S

310821243034: MURALI KRISHNAN M

310821243033: LOKESH V

PHASE-III: Data preprocessing and visualization

PROJECT: *DATA ANALYSIS ON COVID VACCINATION DATA*



AIM:

A COVID-19 analysis report is a comprehensive document that presents data-driven insights and conclusions about the COVID-19 pandemic. It typically covers various aspects, such as the spread of the virus, its impact on public health, economic repercussions, and more. Here's a general structure of such a report:

Introduction:

The COVID 19 pandemic caused due to the Corona virus devastated the world by causing several fatalities around the world. This virus originated in Wuhan, China in 2019 and was later spread throughout the world due to human contact in one way or the other. An effort was made to find a cure or vaccine by several health organizations to bring a stop to this pandemic.

In later stages of 2020 several experimental vaccines were developed and was administered to humans. The efforts were successful as the vaccines were helpful in reducing the affects the virus and even if people were infected, they were not in any life threating situation and escaped the illness having only minor symptoms. Many countries later developed their own vaccines and also helped other countries without the resources by providing them with vaccines developed.

Given data set:

jupyter DAC_PHASE3 Last Checkpoint: 09/19/2023 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

Out[1]:

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	daily_vaccinations	total_vaccinations_per_
0	Afghanistan	AFG	2021-02-22	0.0	0.0	NaN	NaN	NaN	
1	Afghanistan	AFG	2021-02-23	NaN	NaN	NaN	NaN	1367.0	
2	Afghanistan	AFG	2021-02-24	NaN	NaN	NaN	NaN	1367.0	
3	Afghanistan	AFG	2021-02-25	NaN	NaN	NaN	NaN	1367.0	
4	Afghanistan	AFG	2021-02-26	NaN	NaN	NaN	NaN	1367.0	
5	Afghanistan	AFG	2021-02-27	NaN	NaN	NaN	NaN	1367.0	
6	Afghanistan	AFG	2021-02-28	8200.0	8200.0	NaN	NaN	1367.0	
7	Afghanistan	AFG	2021-03-01	NaN	NaN	NaN	NaN	1580.0	
8	Afghanistan	AFG	2021-03-02	NaN	NaN	NaN	NaN	1794.0	
9	Afghanistan	AFG	2021-03-03	NaN	NaN	NaN	NaN	2008.0	

Data Preprocessing:

It involves identifying and correcting errors or inconsistencies in the data, such as missing values, outliers, and duplicates. Various techniques can be used for data cleaning, such as imputation, removal, and transformation.

Necessary step to follow:

1.Import Libraries:

Start by importing all the necessary libraries.

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
```

2.Loading the DataSet:

```
In [1]: import pandas as pd
        data=pd.read_csv(r"C:\Users\murali\OneDrive\Desktop\country_vaccinations.csv")
        data.head(10)
```

Out[1]:

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	daily_vaccinations	total_vaccinations_per_
0	Afghanistan	AFG	2021-02-22	0.0	0.0	NaN	NaN	NaN	
1	Afghanistan	AFG	2021-02-23	NaN	NaN	NaN	NaN	1367.0	
2	Afghanistan	AFG	2021-02-24	NaN	NaN	NaN	NaN	1367.0	
3	Afghanistan	AFG	2021-02-25	NaN	NaN	NaN	NaN	1367.0	
4	Afghanistan	AFG	2021-02-26	NaN	NaN	NaN	NaN	1367.0	
5	Afghanistan	AFG	2021-02-27	NaN	NaN	NaN	NaN	1367.0	
6	Afghanistan	AFG	2021-02-28	8200.0	8200.0	NaN	NaN	1367.0	
7	Afghanistan	AFG	2021-03-01	NaN	NaN	NaN	NaN	1580.0	

3.Data Cleaning:

Data **cleaning**, **also** known as data **cleansing** or data preprocessing, is a crucial step in the data science pipeline that involves identifying and correcting or removing errors, inconsistencies, and inaccuracies in the data to improve its quality and usability.

Important steps :

1.Data inspection and exploration:

This step involves understanding the data by inspecting its structure and identifying missing values, outliers, and inconsistencies.

```
In [2]: data.shape
Out[2]: (86512, 15)

In [3]: data.columns
Out[3]: Index(['country', 'iso_code', 'date', 'total_vaccinations',
        'people_vaccinated', 'people_fully_vaccinated',
        'daily_vaccinations_raw', 'daily_vaccinations',
        'total_vaccinations_per_hundred', 'people_vaccinated_per_hundred',
        'people_fully_vaccinated_per_hundred', 'daily_vaccinations_per_million',
        'vaccines', 'source_name', 'source_website'],
       dtype='object')
```

.shape function returns the total no of rows and columns present in our dataset and .columns function gives us the columns names

Let’s see the descriptive structure of the data using data.describe() and data.info()

```
In [5]: data.describe()
Out[5]:
```

	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	daily_vaccinations	total_vaccinations_per_hundred	people_vaccinat
count	4.360700e+04	4.129400e+04	3.880200e+04	3.536200e+04	8.621300e+04	43607.000000	
mean	4.592964e+07	1.770508e+07	1.413830e+07	2.705996e+05	1.313055e+05	80.188543	
std	2.246004e+08	7.078731e+07	5.713920e+07	1.212427e+06	7.682388e+05	67.913577	
min	0.000000e+00	0.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	0.000000	
25%	5.264100e+05	3.494642e+05	2.439622e+05	4.668000e+03	9.000000e+02	16.050000	
50%	3.590096e+06	2.187310e+06	1.722140e+06	2.530900e+04	7.343000e+03	67.520000	
75%	1.701230e+07	9.152520e+06	7.559870e+06	1.234925e+05	4.409800e+04	132.735000	
max	3.263129e+09	1.275541e+09	1.240777e+09	2.474100e+07	2.242429e+07	345.370000	

Checking data Information using .info()

From the below data info, we can see that many columns have an unequal number of counts. And some of the columns have data type objects and some are float values.

```
In [4]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 86512 entries, 0 to 86511
Data columns (total 15 columns):
 #   Column                               Non-Null Count  Dtype  
---  ------                               --------------  -----  
 0   country                              86512 non-null  object 
 1   iso_code                             86512 non-null  object 
 2   date                                 86512 non-null  object 
 3   total_vaccinations                   43607 non-null  float64
 4   people_vaccinated                    41294 non-null  float64
 5   people_fully_vaccinated              38802 non-null  float64
 6   daily_vaccinations_raw               35362 non-null  float64
 7   daily_vaccinations                   86213 non-null  float64
 8   total_vaccinations_per_hundred       43607 non-null  float64
 9   people_vaccinated_per_hundred        41294 non-null  float64
 10  people_fully_vaccinated_per_hundred  38802 non-null  float64
 11  daily_vaccinations_per_million       86213 non-null  float64
 12  vaccines                             86512 non-null  object 
 13  source_name                          86512 non-null  object 
 14  source_website                       86512 non-null  object 
dtypes: float64(9), object(6)
memory usage: 9.9+ MB
```

2. Removal of unwanted observation:

This includes deleting duplicate/ redundant or irrelevant values from our dataset. Duplicate observations most frequently arise during data collection and Irrelevant observations are those that don't actually fit the specific problem

As we know our machines don't understand the text data. So, we have to either drop or convert the categorical column values into numerical types.Here we are dropping the columns because it hasn't a great influence on target variables

```
In [6]: data.drop(['iso_code', 'source_name', 'source_website'],axis=1,inplace=True)

In [7]: data.columns

Out[7]: Index(['country', 'date', 'total_vaccinations', 'people_vaccinated',
               'people_fully_vaccinated', 'daily_vaccinations_raw',
               'daily_vaccinations', 'total_vaccinations_per_hundred',
               'people_vaccinated_per_hundred', 'people_fully_vaccinated_per_hundred',
               'daily_vaccinations_per_million', 'vaccines'],
              dtype='object')
```

3. Handling missing data:

Missing data is a common issue in real-world datasets, and it can occur due to various reasons such as human errors, system failures, or data collection issues. Various techniques can be used to handle missing data, such as imputation, deletion, or substitution.

Let's check the missing values columns-wise for each row using data.isnull() it checks whether the values are null or not and gives returns boolean values and .sum() will sum the total number of null values rows

```
In [8]: data.isnull().sum()

Out[8]: country                                0
        date                                   0
        total_vaccinations                 42905
        people_vaccinated                  45218
        people_fully_vaccinated            47710
        daily_vaccinations_raw             51150
        daily_vaccinations                   299
        total_vaccinations_per_hundred     42905
        people_vaccinated_per_hundred      45218
        people_fully_vaccinated_per_hundred 47710
        daily_vaccinations_per_million       299
        vaccines                               0
        dtype: int64
```

We cannot just ignore or remove the missing observation. They must be handled carefully as they can be an indication of something important.

1.Dropping observations with missing values.

2.Inputing the missing values from past observation

```
In [9]: data.fillna(0,inplace=True)
        data.isnull().sum()

Out[9]: country                                0
        date                                   0
        total_vaccinations                     0
        people_vaccinated                      0
        people_fully_vaccinated                0
        daily_vaccinations_raw                 0
        daily_vaccinations                     0
        total_vaccinations_per_hundred         0
        people_vaccinated_per_hundred          0
        people_fully_vaccinated_per_hundred    0
        daily_vaccinations_per_million         0
        vaccines                               0
        dtype: int64
```

4. Data transformation:

Data transformation involves converting the data from one form to another to make it more suitable for analysis.

```
In [14]: data['date']=pd.to_datetime(data['date'])
```

```
In [15]: print(data['date'].dtype)

datetime64[ns]
```

Since some of the columns in our dataset contains float datatype we converted them into integer datatype.

jupyter DAC_PHASE3 Last Checkpoint: 09/19/2023 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

```
Out[16]: 0        2021-02-22
         1        2021-02-23
         2        2021-02-24
         3        2021-02-25
         4        2021-02-26
                     ...
         86507    2022-03-25
         86508    2022-03-26
         86509    2022-03-27
         86510    2022-03-28
         86511    2022-03-29
         Name: date, Length: 86512, dtype: datetime64[ns]
```

```
In [17]: data[['total_vaccinations','people_vaccinated','people_fully_vaccinated','daily_vaccinations_raw',
                 'daily_vaccinations', 'total_vaccinations_per_hundred',
                 'people_vaccinated_per_hundred', 'people_fully_vaccinated_per_hundred',
                 'daily_vaccinations_per_million']]=data[['total_vaccinations','people_vaccinated','people_fully_vaccinated','daily_vaccina
                 'daily_vaccinations', 'total_vaccinations_per_hundred',
                 'people_vaccinated_per_hundred', 'people_fully_vaccinated_per_hundred',
                 'daily_vaccinations_per_million']].astype(int)
```

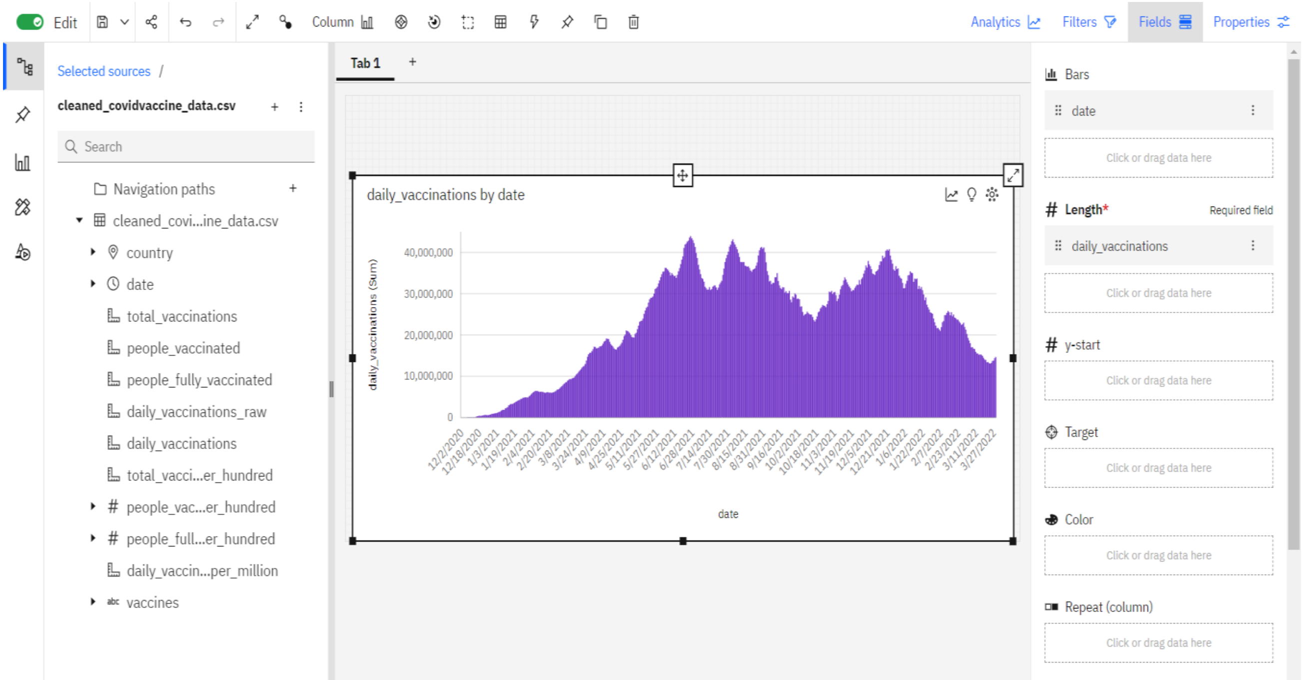
```
In [18]: data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 86512 entries, 0 to 86511
Data columns (total 12 columns):
 #   Column                               Non-Null Count  Dtype
---  ------                               --------------  -----
 0   country                              86512 non-null  object
 1   date                                 86512 non-null  datetime64[ns]
 2   total_vaccinations                   86512 non-null  int32
 3   people_vaccinated                    86512 non-null  int32
 4   people_fully_vaccinated              86512 non-null  int32
 5   daily_vaccinations_raw               86512 non-null  int32
 6   daily_vaccinations                   86512 non-null  int32
 7   total_vaccinations_per_hundred       86512 non-null  int32
 8   people_vaccinated_per_hundred        86512 non-null  int32
 9   people_fully_vaccinated_per_hundred  86512 non-null  int32
 10  daily_vaccinations_per_million       86512 non-null  int32
 11  vaccines                             86512 non-null  object
dtypes: datetime64[ns](1), int32(9), object(2)
memory usage: 5.6+ MB
```

4.Visualization:

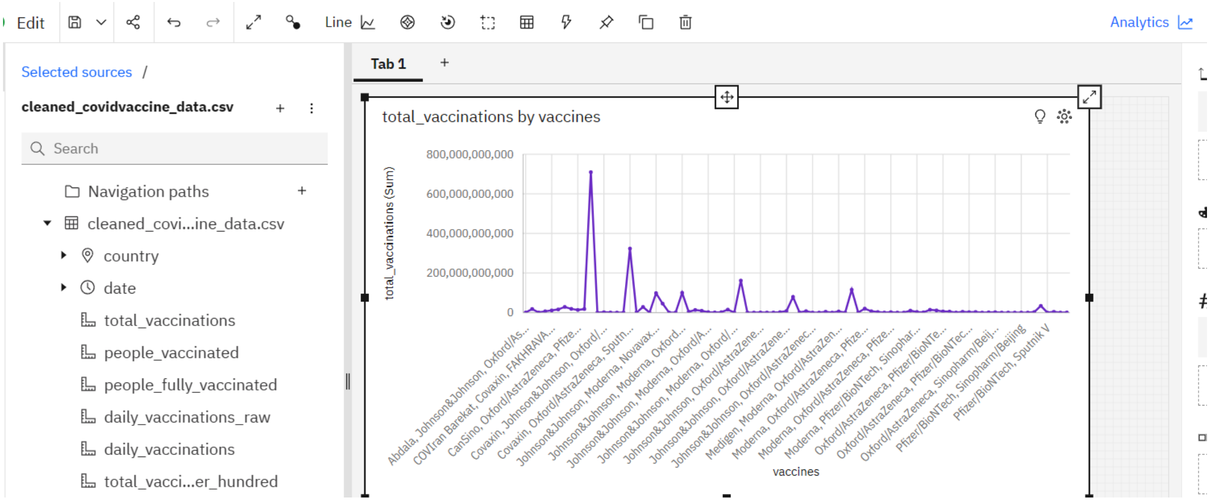
After loading the cleaned dataset into IBM Cognos , we visualized the relations between some of the columns using IBM Cognos analytics.

1.The number of daily vaccinations by date :

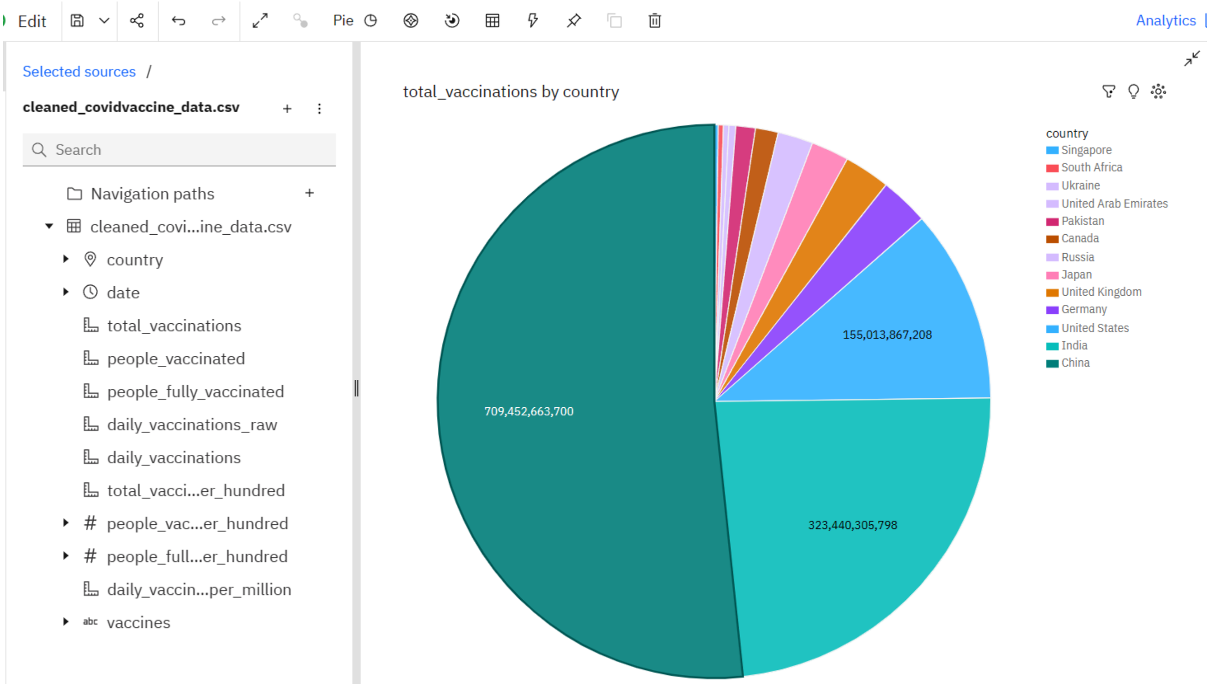


2.Some countries vaccination progresses:

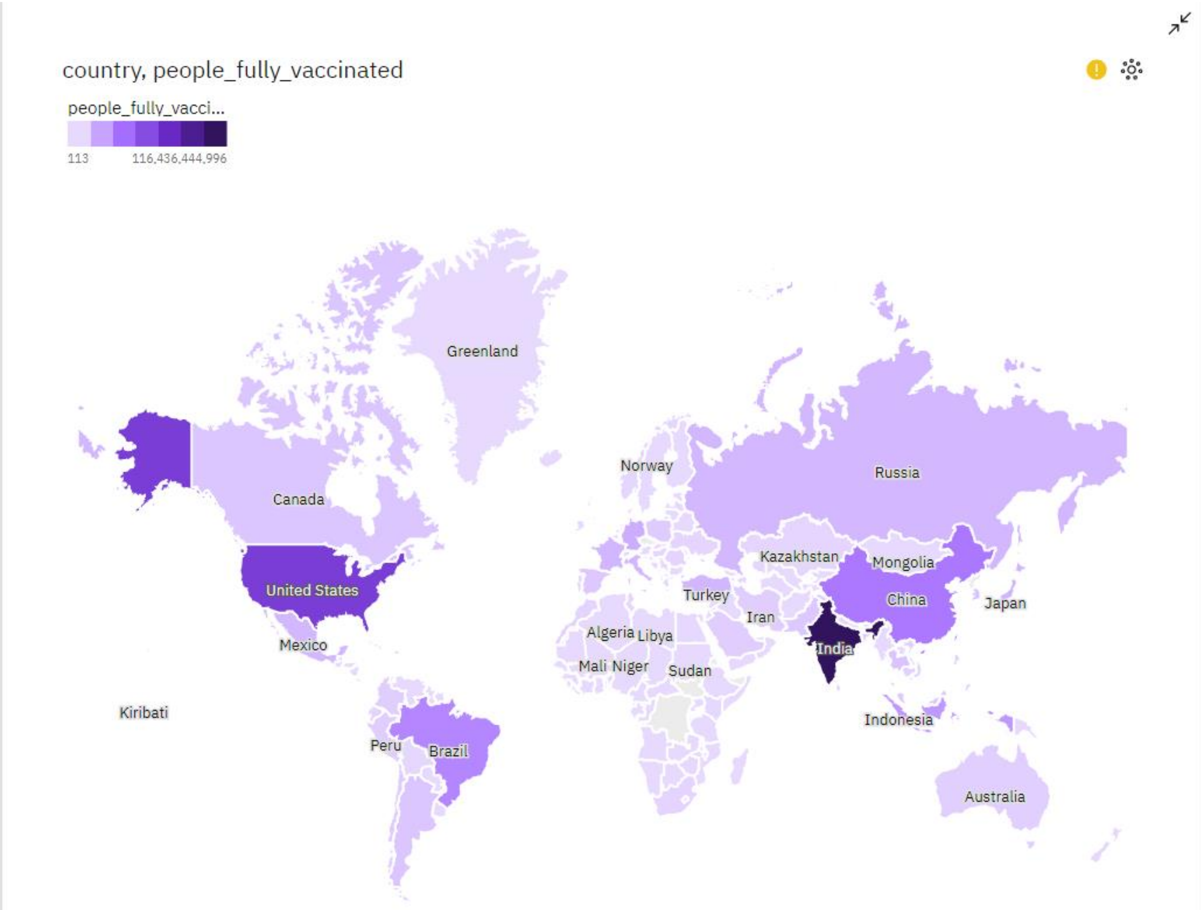
3.Total vaccinations by vaccines



4.Total Vaccinations by country:



4.Country vs people fully vaccinated:



Conclusion:

The preprocessing of our dataset on COVID vaccine analysis has been successfully completed, ensuring the integrity, reliability, and consistency of our data.

By managing missing values, outliers, and any anomalies present, the data is now primed for more intricate analytical and modelling tasks. Basic visualization is provided along with the trends and patterns associated with the vaccine data. While these initial visual findings have paved the way for deeper understanding, subsequent analyses will be pivotal in drawing more concrete and actionable conclusions. As we continue, it will be crucial to validate our hypotheses and findings, further utilizing this cleaned and structured dataset to its fullest potential.