

COVID-19 VACCINES ANALYSIS

Member name: GOKULA KANNAN M

Code:au723721205018

Phase 3 Submission

INTRODUCTION:

Data loading and preprocessing are essential steps in the field of data science and machine learning. These processes are crucial for preparing raw data into a format that can be effectively used for analysis and model training.

Data Loading:

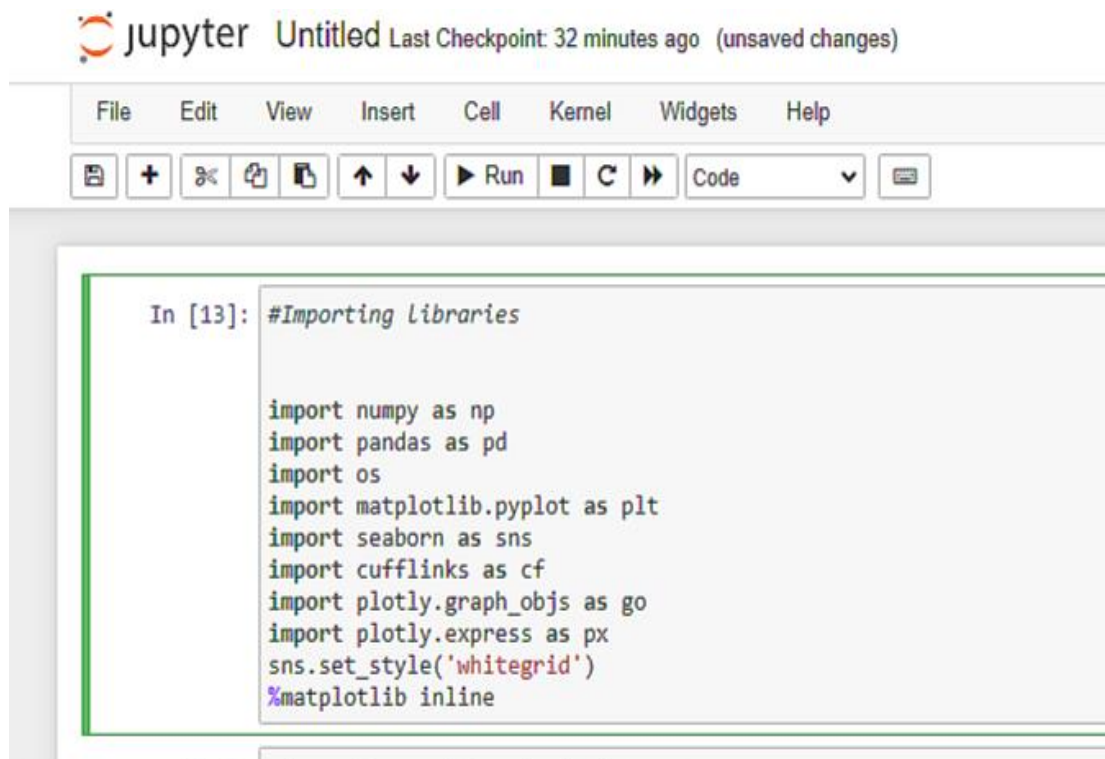
Data loading involves acquiring data from various sources, such as databases, files, or web services. This step ensures that the required data is accessible and ready for further processing. Common data loading techniques include reading data from CSV files, querying databases, or using APIs to fetch data from the web.

Data Preprocessing:

Data preprocessing, on the other hand, involves cleaning, transforming, and organizing the data to make it suitable for analysis and modeling. This may include handling missing values, normalizing or scaling features, encoding categorical variables, and splitting the data into training and testing sets. Data preprocessing is critical as it helps improve the quality of the data and ensures that machine learning algorithms can work effectively.

We can get sources from Github, Kaggle etc.

Importing Necessary Libraries:



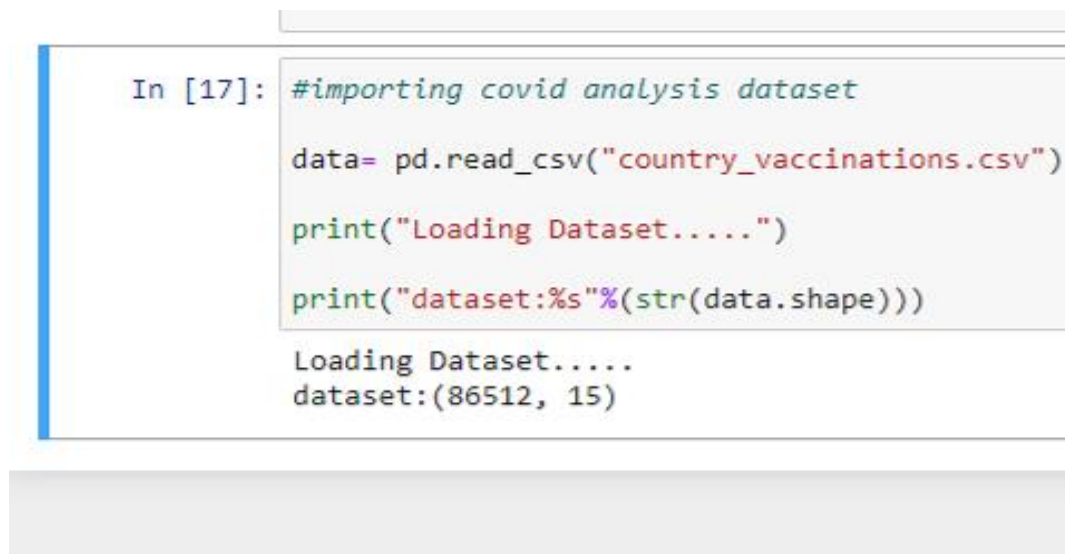
The image shows a Jupyter Notebook interface. At the top, the title bar says "jupyter Untitled Last Checkpoint: 32 minutes ago (unsaved changes)". Below the title bar is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Under the menu bar is a toolbar with icons for saving, adding cells, undo, redo, and running code. The main area shows a code cell labeled "In [13]:". The code in the cell is for importing various libraries: numpy, pandas, os, matplotlib.pyplot, seaborn, cufflinks, plotly.graph_objs, and plotly.express. It also sets the plotly style to 'whitegrid' and enables inline matplotlib plots.

```
In [13]: #Importing Libraries

import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
import cufflinks as cf
import plotly.graph_objs as go
import plotly.express as px
sns.set_style('whitegrid')
%matplotlib inline
```

These libraries are requiring for performing necessary operations and plots for analysis purpose.

Importing Dataset (Covid analysis):



The image shows a Jupyter Notebook interface with a code cell labeled "In [17]:". The code imports the pandas library and reads a CSV file named "country_vaccinations.csv". It then prints the dataset's shape. The output shows the dataset has 86512 rows and 15 columns.

```
In [17]: #importing covid analysis dataset

data= pd.read_csv("country_vaccinations.csv")

print("Loading Dataset.....")

print("dataset:%s"%(str(data.shape)))

Loading Dataset.....
dataset:(86512, 15)
```

The dataset has a total of 15 columns and 86512 records . Once the dataset is loaded we will

take a look at the dataset entries. Head () is used to get the starting records from the dataset and tail () is used to get the ending records from the dataset. We can also specify how much records are to be displayed.

```
In [19]: data.head(6)
```

```
Out[19]:
```

	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	

In the above our dataset is defined as 'data' and data.head(5) has returned the top 5 records.

```
In [20]: data.tail(6)
```

```
Out[20]:
```

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	daily_vaccinations	total_vaccinations_
86506	Zimbabwe	ZWE	2022-03-24	8552429.0	4704720.0	3461926.0	137952.0	51151.0	
86507	Zimbabwe	ZWE	2022-03-25	8691642.0	4814582.0	3473523.0	139213.0	69579.0	
86508	Zimbabwe	ZWE	2022-03-26	8791728.0	4886242.0	3487962.0	100086.0	83429.0	
86509	Zimbabwe	ZWE	2022-03-27	8845039.0	4918147.0	3493763.0	53311.0	90629.0	
86510	Zimbabwe	ZWE	2022-03-28	8934360.0	4975433.0	3501493.0	89321.0	100614.0	
86511	Zimbabwe	ZWE	2022-03-29	9039729.0	5053114.0	3510256.0	105369.0	103751.0	

In the above figure data.tail(5) has returned the last 5 records.

Preprocessing Steps:

Here are common data preprocessing steps:

1. Data Cleaning:

- Handling Missing Values: Fill in or remove missing data points.
- Outlier Detection: Identify and deal with outliers that may skew analysis.

2. Data Transformation:

- Feature Scaling: Normalize or standardize numerical features to a common scale.
- Encoding Categorical Data: Convert categorical variables into numerical representations (e.g., one-hot encoding).
- Feature Engineering: Create new features or transform existing ones to improve model performance.

3. Data Reduction:

- Dimensionality Reduction: Reduce the number of features while retaining meaningful information (e.g., using techniques like Principal Component Analysis).

4. Data Splitting:

- Split the dataset into training and testing sets to assess model performance.

5. Data Normalization:

- Normalize the data to have a standard distribution (e.g., using Z-score normalization).

6. Data Balancing:

- Address class imbalances, especially in classification tasks, to prevent bias in the model.

7. Text Data Preprocessing:

- Tokenization: Split text into words or tokens.
- Stop Word Removal: Eliminate common, non-informative words.
- Lemmatization or Stemming: Reduce words to their base form.

8. Time Series Data Preprocessing:

- Resampling: Aggregate or interpolate time series data into different time intervals.
- Feature Extraction: Create meaningful features from time-based data.

9. Handling Duplicates:

- Identify and remove duplicate records to avoid bias in analysis.

10. Data Imputation:

- Fill in missing values using statistical techniques, such as mean, median, or regression imputation.

11. Data Scaling:

- Scale data to a specific range (e.g., [0, 1]) to ensure that features with different units have equal influence.

12. Data Integration:

- Combine data from multiple sources into a single, coherent dataset.

These preprocessing steps are implemented for machine learning prediction model.

Since we are only going to analyse the dataset, we need only few preprocessing steps.

Missing values:

```
In [21]: #checking null values
         data.isnull().sum(axis=0)

Out[21]: country                0
         iso_code                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
         source_name              0
         source_website           0
         dtype: int64
```

isnull() returns true or false in presence or absence of NaN values in the records. Using sum() we can get the total number of null values in the records.

Removing null values:

```
In [23]: #Removing null values for totvacc and peopvacc
data.dropna(subset=["total_vaccinations", "people_vaccinated"], how="all", inplace=True)
data.head()
```

```
Out[23]:
```

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations
0	Afghanistan	AFG	2021-02-22	0.0	0.0	NaN	
6	Afghanistan	AFG	2021-02-28	8200.0	8200.0	NaN	
22	Afghanistan	AFG	2021-03-16	54000.0	54000.0	NaN	
44	Afghanistan	AFG	2021-04-07	120000.0	120000.0	NaN	
59	Afghanistan	AFG	2021-04-22	240000.0	240000.0	NaN	

dropna() is used to remove NaN values from columns mentioned. Here we have removed null values from columns 'total_vaccinations' and 'people_vaccinated'.

Removing unnecessary columns:

```
In [26]: #Dropping redundant columns
data.drop(['daily_vaccinations_raw', 'source_website'], axis=1)
```

```
Out[26]:
```

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations
0	Afghanistan	AFG	2021-02-22	0.0	0.0	NaN	
6	Afghanistan	AFG	2021-02-28	8200.0	8200.0	NaN	
22	Afghanistan	AFG	2021-03-16	54000.0	54000.0	NaN	
44	Afghanistan	AFG	2021-04-07	120000.0	120000.0	NaN	
59	Afghanistan	AFG	2021-04-22	240000.0	240000.0	NaN	

Since there is no use of 'daily_vaccinations_raw' and 'source_website' we have dropped these columns using drop() by defining axis=1 which denotes the columns.

Displaying vaccines used by countries:

```
In [29]: #vaccines used by country
vacc_country=data[['country','iso_code','vaccines']]
vacc_country
```

Out[29]:

	country	iso_code	vaccines
0	Afghanistan	AFG	Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi...
6	Afghanistan	AFG	Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi...
22	Afghanistan	AFG	Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi...
44	Afghanistan	AFG	Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi...
59	Afghanistan	AFG	Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi...
...
86507	Zimbabwe	ZWE	Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac...
86508	Zimbabwe	ZWE	Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac...
86509	Zimbabwe	ZWE	Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac...
86510	Zimbabwe	ZWE	Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac...
86511	Zimbabwe	ZWE	Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac...

44011 rows x 3 columns

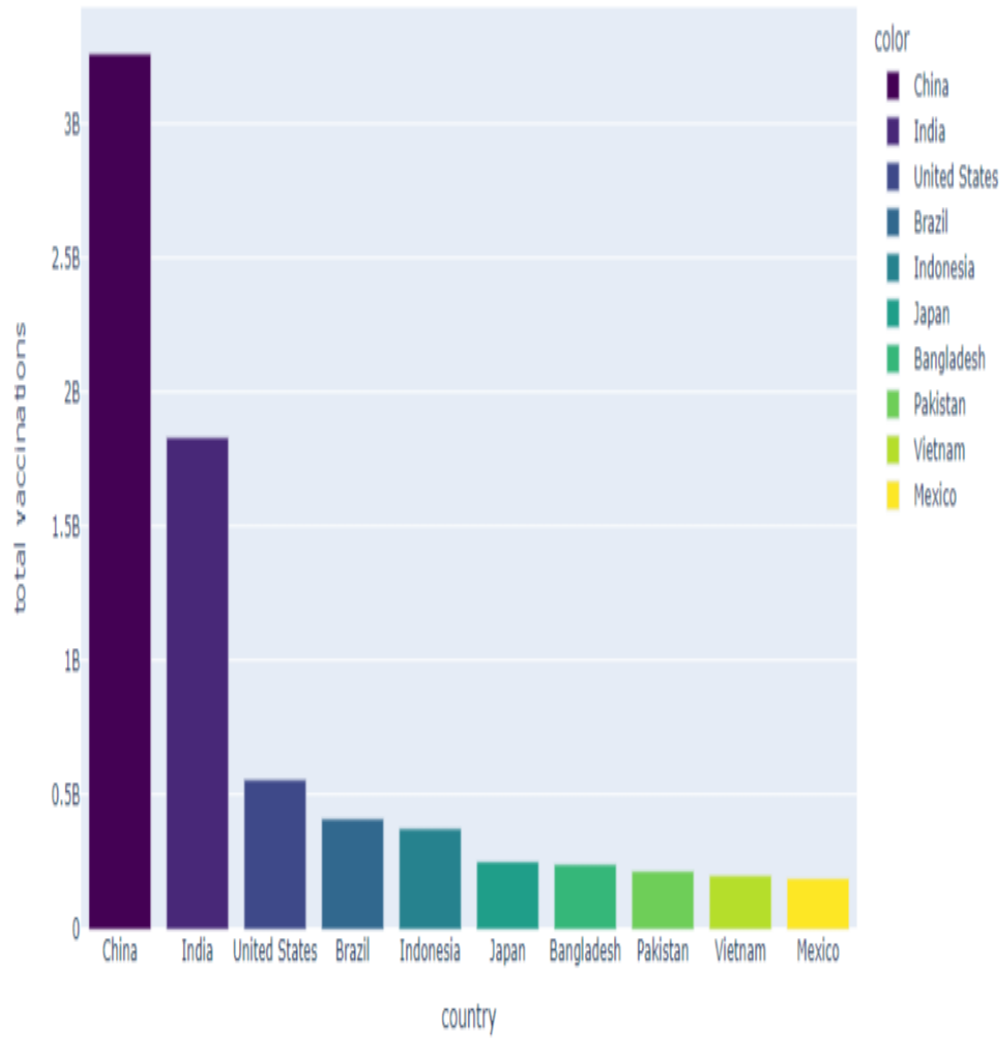
Bar chart for top 10 countries vaccinated:

```
In [*]: #bar chart for top 10 countries vaccinated

fig = px.bar(x=total_vaccinations_top10.index ,y=total_vaccinations_top10.values,
             color=total_vaccinations_top10.index,
             labels={"x": "country", "y": "total vaccinations"},
             color_discrete_sequence =px.colors.sequential.Viridis)

fig.show()
```

Based on country and total vaccinations.



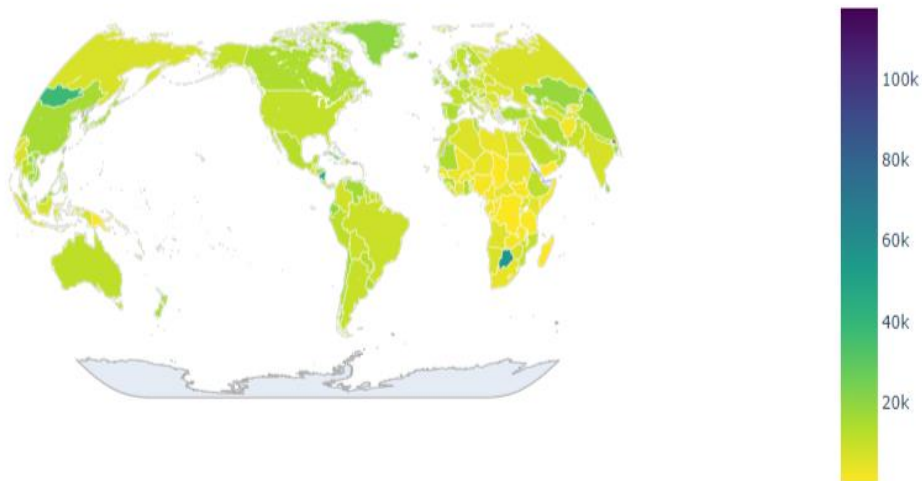
Map visualisation using plotly:

```

In [56]: #vaccinated per million
def create_choropleth(loc,z,text, title):
    fig = go.Figure(data=go.Choropleth(
        locations = loc,
        z=z,
        text=text,
        colorscale = 'viridis',
        autocolorscale=False,
        reversescale=True,
        marker_line_color='white',
        marker_line_width=0.5
    ))
    fig.update_geos(
        visible=True,
        resolution=50,
        showcountries=True,
        countrycolor = 'darkgrey'
    )
    fig.update_layout(
        title_text = title,
        geo=dict(
            showframe=False,
            showcoastlines=False,
            projection_type='natural earth'
        ))
    fig.show()
daily_vac_million = data[['country', 'iso_code', 'daily_vaccinations_per_million']]
daily_vac_million['daily_vaccinations_per_million'] = daily_vac_million['daily_vaccinations_per_million'].fillna(0)
daily_vac_million = daily_vac_million.groupby(['country', 'iso_code']).max().reset_index()
create_choropleth(daily_vac_million['iso_code'],
    daily_vac_million['daily_vaccinations_per_million'],
    daily_vac_million['country'],
    'Daily vaccinations per million')

```

Daily vaccinations per million



Vaccines used by each country:

```

In [65]: #vaccines used by each country

vaccinebycountry_df = data[['country', 'iso_code', 'vaccines', 'total_vaccinations']]
total_vaccinations= vaccinebycountry_df.groupby(['country']).max()['total_vaccinations',
        'vaccines', 'iso_code']].reset_index()

fig = px.choropleth(total_vaccinations, locations = 'country', locationmode = 'country names', color = 'vaccines',
        title = 'total Vaccines used for each country', hover_data= ['total_vaccinations'],
        color_discrete_map=dict(zip(total_vaccinations['vaccines'], px.colors.sequential.Viridis)),
        labels={'vaccines': 'Name of vaccine', 'country': 'Country',
        'total_vaccinations': 'Number of vaccinations'})

fig.update_geos(
    visible=True,
    resolution=50,
    showcountries=True,
    countrycolor="darkgrey"
)
fig.update_layout(
    geo=dict(
        showframe=False,
        showcoastlines=False,
        projection_type='equiangular'
    ),
)
fig.show()

```

total Vaccines used for each country

Name of vaccine

- Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing
- Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac, Sputnik V
- Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac, Sputnik V
- Moderna, Oxford/AstraZeneca, Pfizer/BioNTech
- Oxford/AstraZeneca
- Oxford/AstraZeneca, Pfizer/BioNTech
- Oxford/AstraZeneca, Pfizer/BioNTech, Sputnik V
- CanSino, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V
- Moderna, Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac, Sputnik V
- Pfizer/BioNTech
- Johnson&Johnson, Moderna, Novavax, Oxford/AstraZeneca, Pfizer/BioNTech
- Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech
- Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik Light, Sputnik V
- Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac
- Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing
- Sinopharm/Beijing, Sputnik V
- Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech