```python
import pandas as pd

# Load the CSV file
df = pd.read_csv("owid-covid-data.csv")
# Check basic shape
print("Rows:", df.shape[0])
print("Columns:", df.shape[1])
```

imports the pandas library, loads a COVID-19 dataset from a CSV file, and prints out how many rows and columns the data has.

```
Rows: 429435
Columns: 67
```

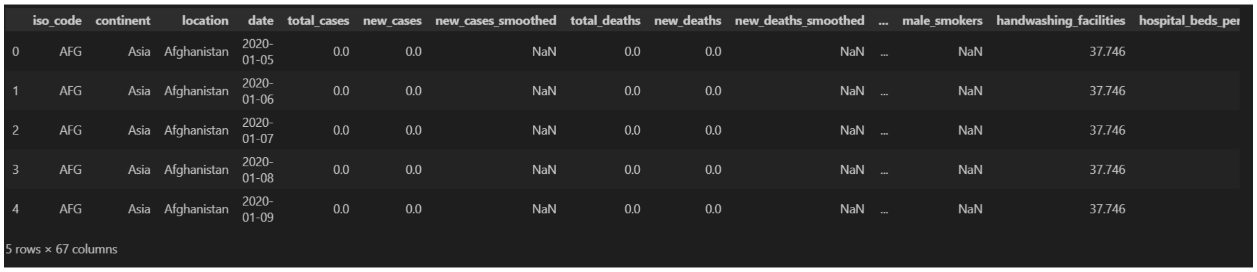
This line converts all the column names in the DataFrame into a Python list, allowing to see all the column headers in the dataset.

```python
df.columns.to_list()
```

```
['iso_code',
 'continent',
 'location',
 'date',
 'total_cases',
 'new_cases',
 'new_cases_smoothed',
 'total_deaths',
 'new_deaths',
 'new_deaths_smoothed',
 'total_cases_per_million',
 'new_cases_per_million',
 'new_cases_smoothed_per_million',
 'total_deaths_per_million',
 'new_deaths_per_million',
 'new_deaths_smoothed_per_million',
 'reproduction_rate',
 'icu_patients',
 'icu_patients_per_million',
 'hosp_patients',
 'hosp_patients_per_million',
 'weekly_icu_admissions',
 'weekly_icu_admissions_per_million',
 'weekly_hosp_admissions',
 'weekly_hosp_admissions_per_million',
...
'population',
 'excess_mortality_cumulative_absolute',
 'excess_mortality_cumulative',
 'excess_mortality',
 'excess_mortality_cumulative_per_million']
```
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

This line displays the first 5 rows of the DataFrame df, giving a quick preview of the data's structure and contents.

```
df.head(5)
```



	iso_code	continent	location	date	total_cases	new_cases	new_cases_smoothed	total_deaths	new_deaths	new_deaths_smoothed	...	male_smokers	handwashing_facilities	hospital_beds_per
0	AFG	Asia	Afghanistan	2020-01-05	0.0	0.0	NaN	0.0	0.0	NaN	...	NaN	37.746	
1	AFG	Asia	Afghanistan	2020-01-06	0.0	0.0	NaN	0.0	0.0	NaN	...	NaN	37.746	
2	AFG	Asia	Afghanistan	2020-01-07	0.0	0.0	NaN	0.0	0.0	NaN	...	NaN	37.746	
3	AFG	Asia	Afghanistan	2020-01-08	0.0	0.0	NaN	0.0	0.0	NaN	...	NaN	37.746	
4	AFG	Asia	Afghanistan	2020-01-09	0.0	0.0	NaN	0.0	0.0	NaN	...	NaN	37.746	

5 rows × 67 columns

This line counts the number of missing (NaN) values in each column of the DataFrame df and sorts the results in descending order to show which columns have the most missing data.

```
df.isnull().sum().sort_values(ascending=False)
```

```
weekly_icu_admissions                418442
weekly_icu_admissions_per_million      418442
excess_mortality                     416024
excess_mortality_cumulative_absolute    416024
excess_mortality_cumulative            416024
                                   ...
total_cases_per_million                17631
location                              0
iso_code                               0
date                                  0
population                             0
Length: 67, dtype: int64
```

This line calculates the **percentage of missing values** for each column in the DataFrame df and sorts the results in descending order, helping you identify which columns have the highest proportion of missing data.

```
(df.isnull().mean() * 100).sort_values(ascending=False)
```

```
weekly_icu_admissions                97.440125
weekly_icu_admissions_per_million      97.440125
excess_mortality                     96.877059
excess_mortality_cumulative_absolute    96.877059
excess_mortality_cumulative            96.877059
                                    ...
total_cases_per_million                4.105627
location                           0.000000
iso_code                            0.000000
date                               0.000000
population                          0.000000
Length: 67, dtype: float64
```

This code filters the DataFrame df to keep only the rows where the **location column matches one of the specified countries** (Kenya, United States, or India). It then creates a **copy** of this filtered data in a new DataFrame called df_filtered.

```
countries = ['Kenya', 'United States', 'India']
df_filtered = df[df['location'].isin(countries)].copy()
```

Converts the date column in the df_filtered DataFrame to a datetime format. This ensures the date values are properly recognized as date objects for easier manipulation and analysis (e.g., filtering by date, sorting, etc.).

```
df_filtered['date'] = pd.to_datetime(df_filtered['date'])
```

Removes rows from the df_filtered DataFrame where any of the values in the date, total_cases, or total_deaths columns are missing (NaN). This ensures that only rows with complete data in these key columns are retained for further analysis.

```
df_filtered = df_filtered.dropna(subset=['date', 'total_cases', 'total_deaths'])
```

This code sorts the df_filtered DataFrame first by the location column, and then by the date column. The sorting is done in ascending order by default, ensuring that the data is organized by location and then by the chronological order of the dates within each location.

```
df_filtered = df_filtered.sort_values(by=['location', 'date'])
```

This fills missing values in the dataset by using the most recent available data within each country. The index is reset to avoid any grouping structure.

```
# Fill forward missing values within each country
df_filtered = df_filtered.groupby('location').apply(lambda group:
group.fillna(method='ffill')).reset_index(drop=True)
```

performs linear interpolation to fill missing values in all numeric columns. It ensures that the interpolation only happens forward in time, filling gaps with estimated values based on surrounding data.

```
# Interpolate numeric columns (safely)
numeric_cols = df_filtered.select_dtypes(include='number').columns
df_filtered[numeric_cols] = df_filtered[numeric_cols].interpolate(method='linear',
limit_direction='forward')
```

resets the index of the DataFrame after all previous operations. The drop=True argument ensures the old index is discarded, and inplace=True modifies the DataFrame directly without creating a new one.

```
df_filtered.reset_index(drop=True, inplace=True)
```

calculates the total number of missing (null) values in each column of the filtered DataFrame. The result is sorted in descending order to show columns with the most missing values. It then displays the top 10 columns with the highest number of missing values.

```
df_filtered.isnull().sum().sort_values(ascending=False).head(10)
```

```
weekly_icu_admissions                   5022
weekly_icu_admissions_per_million       5022
weekly_hosp_admissions                  3546
weekly_hosp_admissions_per_million      3546
icu_patients_per_million                3540
hosp_patients_per_million               3540
icu_patients                            3540
hosp_patients                           3540
excess_mortality_cumulative_absolute    3348
excess_mortality                        3348
dtype: int64
```

import matplotlib.pyplot as plt: Imports the matplotlib library for creating visualizations, specifically for plotting graphs.

import seaborn as sns: Imports the seaborn library, which is used for making statistical graphics with enhanced visualization features.

sns.set(style="whitegrid"): Sets the default aesthetic style of the plots to a white grid, making the background cleaner and easier to read.

plt.rcParams["figure.figsize"] = (12, 6): Sets the default size of the plots to 12 inches wide and 6 inches tall. This ensures that the plots have a larger, more readable format.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Set style
sns.set(style="whitegrid")
plt.rcParams["figure.figsize"] = (12,6)
```

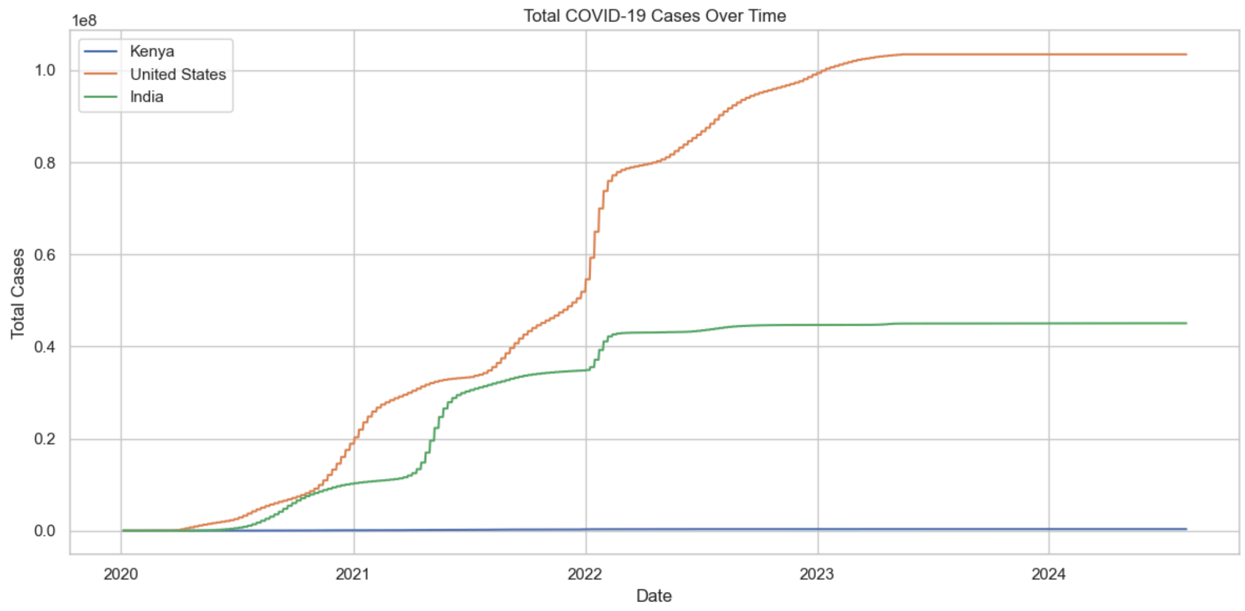
Filters data for specific countries (Kenya, United States, and India).

Plots total COVID-19 cases over time for each country.

Adds a title, axis labels, and a legend to the plot.

Displays the plot.

```
for country in countries:
    data = df_filtered[df_filtered['location'] == country]
    plt.plot(data['date'], data['total_cases'], label=country)

plt.title('Total COVID-19 Cases Over Time')
plt.xlabel('Date')
plt.ylabel('Total Cases')
plt.legend()
plt.tight_layout()
plt.show()
```



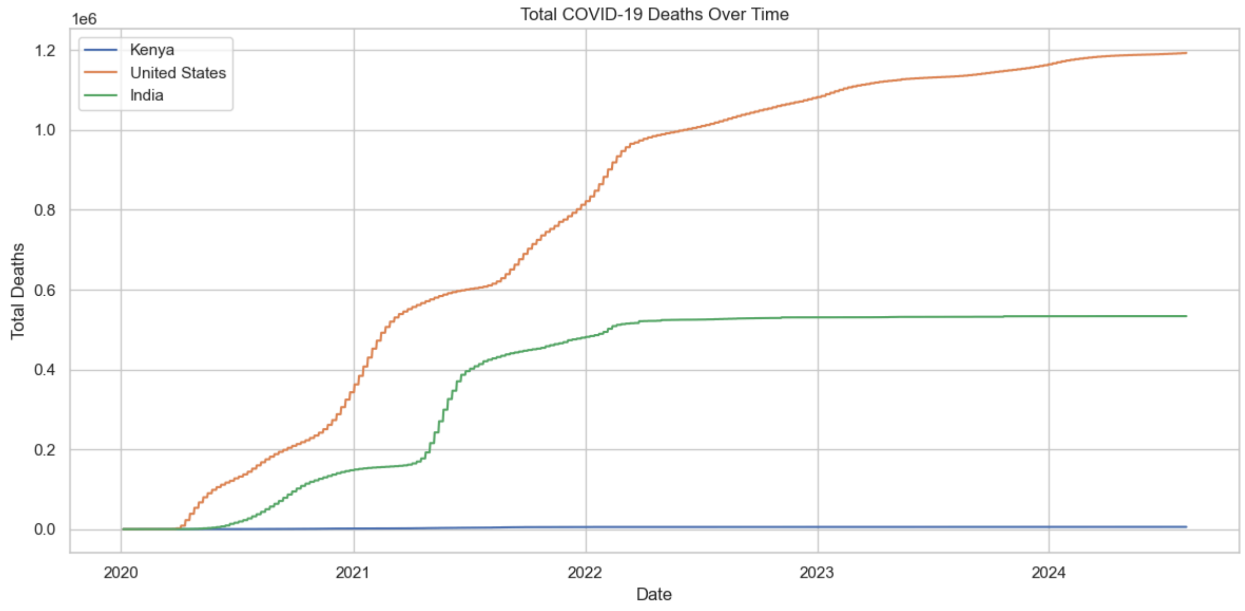
Filters the data for each country (Kenya, United States, and India) to focus on the total_deaths column.

Plots the total COVID-19 deaths over time for each country.

Adds a title, axis labels, and a legend to the plot.

Displays the plot showing the total deaths for each country over time.

```
for country in countries:
    data = df_filtered[df_filtered['location'] == country]
    plt.plot(data['date'], data['total_deaths'], label=country)

plt.title('Total COVID-19 Deaths Over Time')
plt.xlabel('Date')
plt.ylabel('Total Deaths')
plt.legend()
plt.tight_layout()
plt.show()
```



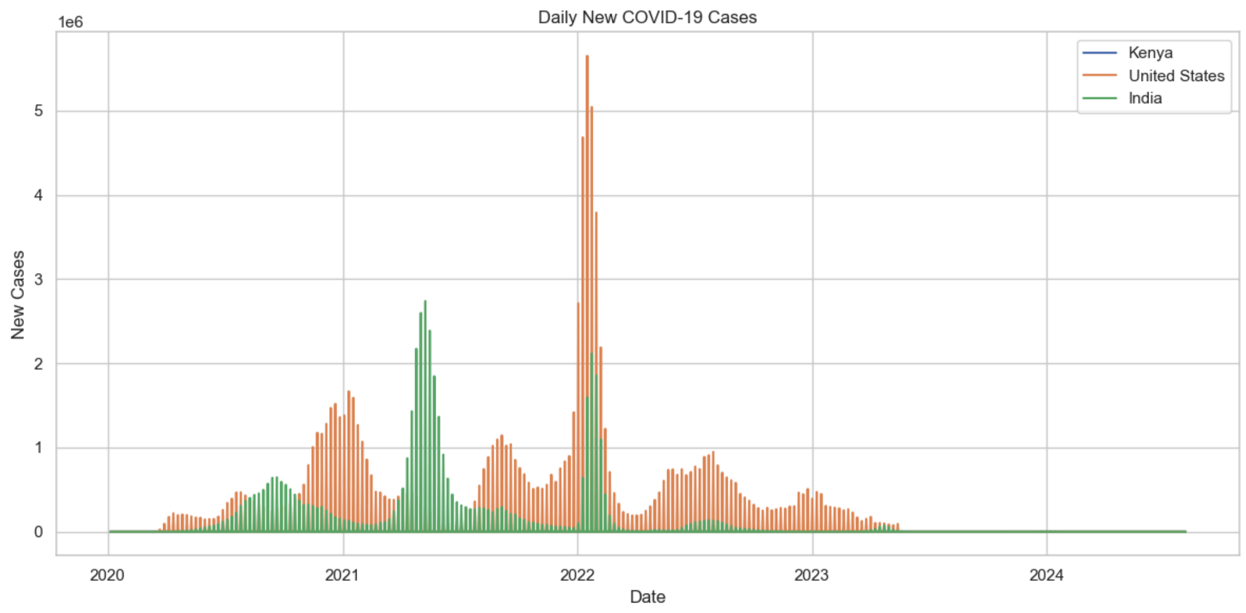
Filters the data for each country (Kenya, United States, and India) to focus on the new_cases column (daily new COVID-19 cases).

Plots the daily new COVID-19 cases over time for each country.

Adds a title, axis labels, and a legend to the plot.

Displays the plot showing the daily new cases for each country over time.

```
for country in countries:
    data = df_filtered[df_filtered['location'] == country]
    plt.plot(data['date'], data['new_cases'], label=country)

plt.title('Daily New COVID-19 Cases')
plt.xlabel('Date')
plt.ylabel('New Cases')
plt.legend()
plt.tight_layout()
plt.show()
```



Creates a new column, death_rate, which calculates the death rate by dividing the total_deaths by the total_cases for each entry in the dataset.

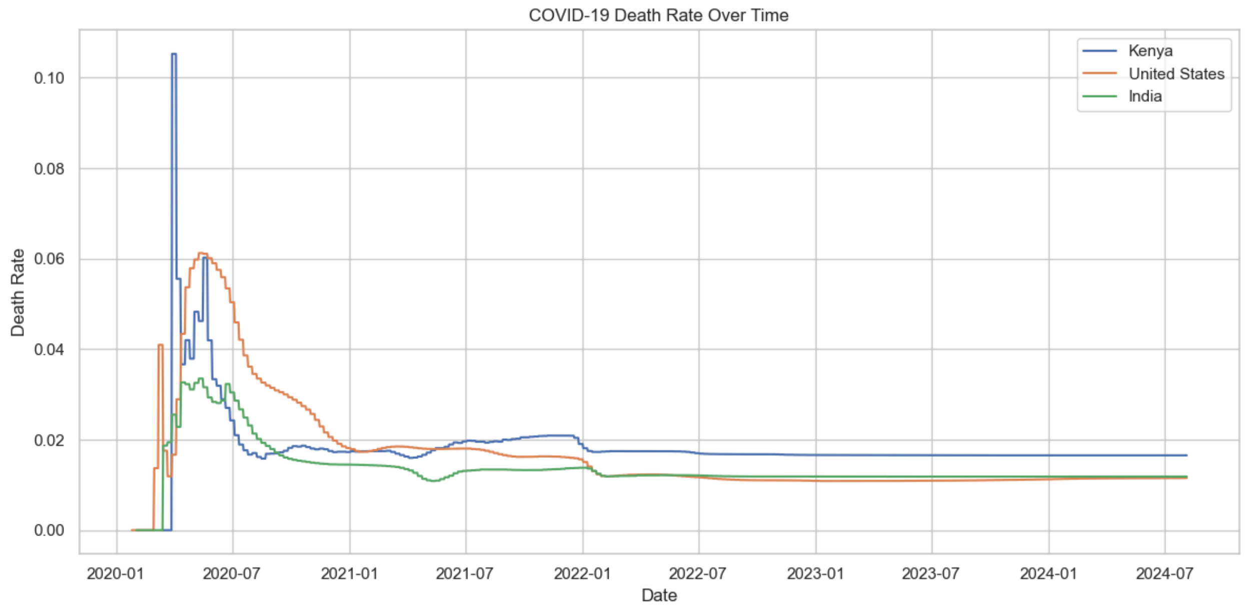
Filters the data for each country (Kenya, United States, and India) to plot the death_rate over time.

Plots the death rate for each country over time.

Adds a title, axis labels, and a legend to the plot.

Displays the plot showing the COVID-19 death rate over time for each country.

```
df_filtered['death_rate'] = df_filtered['total_deaths'] / df_filtered['total_cases']

for country in countries:
    data = df_filtered[df_filtered['location'] == country]
    plt.plot(data['date'], data['death_rate'], label=country)

plt.title('COVID-19 Death Rate Over Time')
plt.xlabel('Date')
plt.ylabel('Death Rate')
plt.legend()
plt.tight_layout()
plt.show()
```



Imports pandas, seaborn, and matplotlib for data handling and visualization.

Converts the date column to datetime format and finds the latest date in the data.

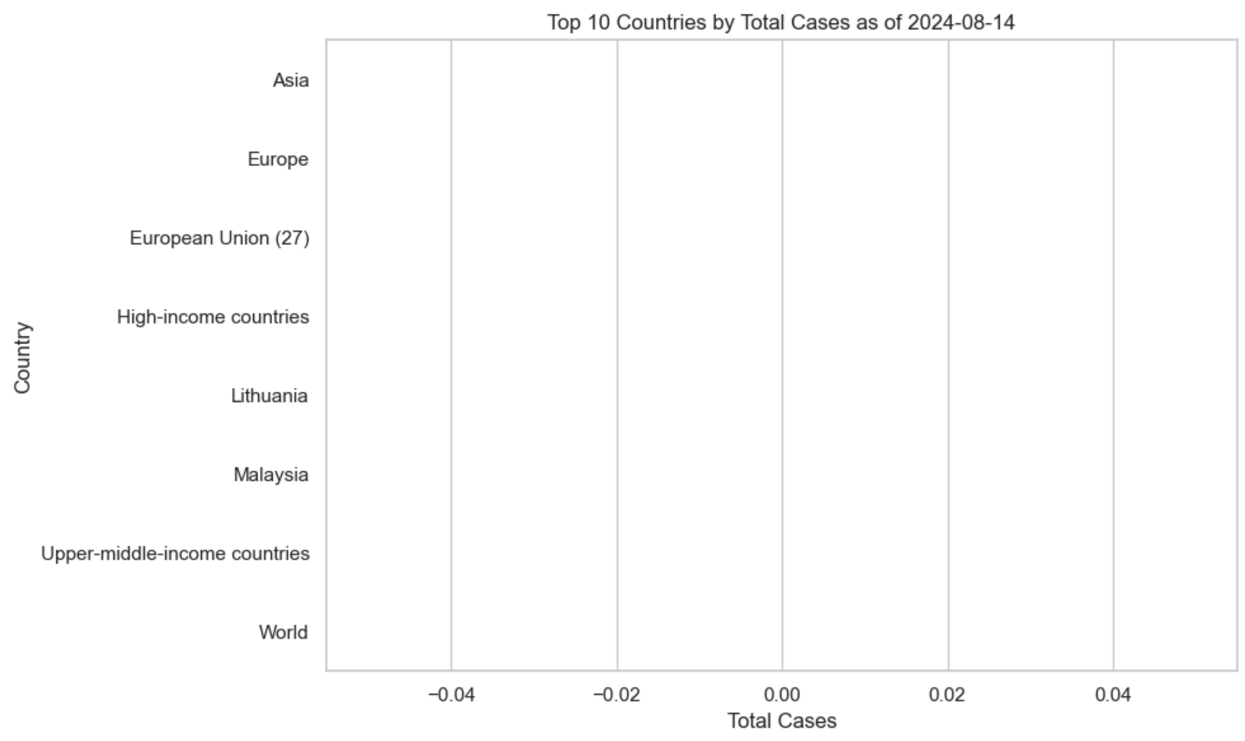
Filters the data to get the top 10 countries with the highest total cases on the most recent date.

Creates a horizontal bar chart to show the total cases for these top 10 countries using a red color scheme.

Displays the chart.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


# Ensure the date is in datetime format
```

```
latest_date = pd.to_datetime(df['date'].max())

# Filter top 10 countries by total cases on latest date
top_cases = df[df['date'] == latest_date.strftime('%Y-%m-%d')].sort_values('total_cases',
ascending=False).head(10)

plt.figure(figsize=(10, 6))
sns.barplot(
    data=top_cases,
    x='total_cases',
    y='location',
    hue='location',
    palette='Reds_r',
    dodge=False,
    legend=False
)

plt.title(f"Top 10 Countries by Total Cases as of {latest_date.strftime('%Y-%m-%d')}")
plt.xlabel("Total Cases")
plt.ylabel("Country")
plt.tight_layout()
plt.show()
```



Filters the dataset for data from Kenya and removes any missing values.

Selects key numeric columns: 'total_cases', 'new_cases', 'total_deaths', 'new_deaths', and 'total_vaccinations'.

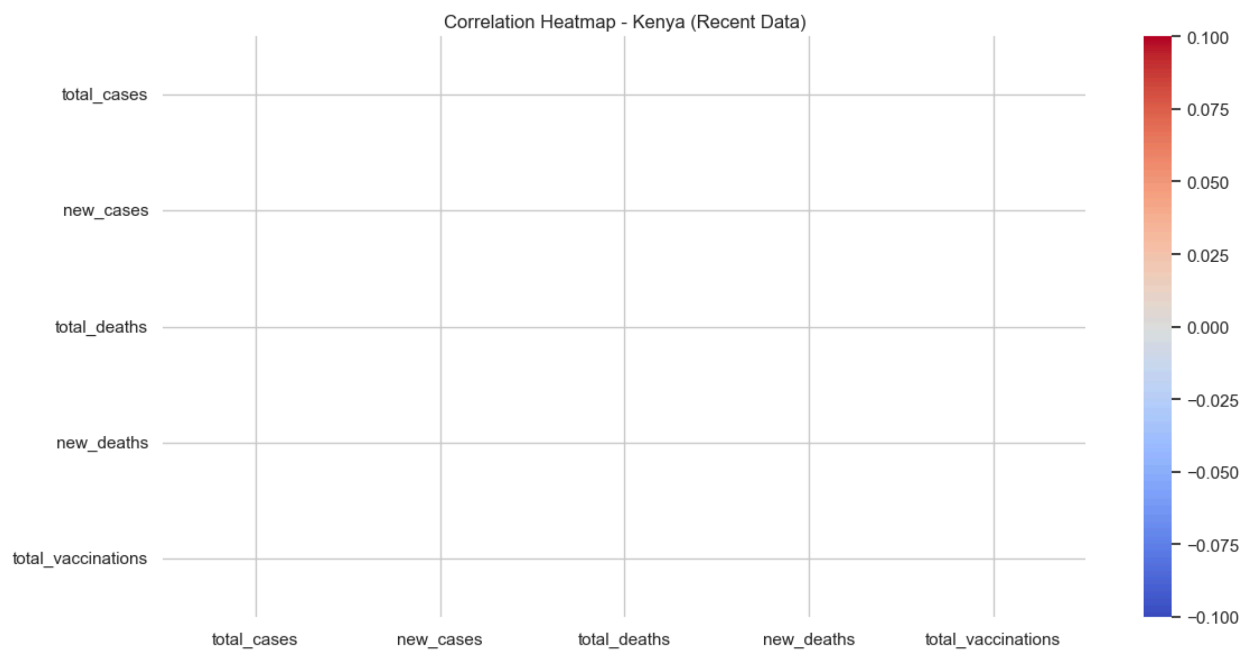
Calculates the correlation matrix between these selected features.

Uses a heatmap to visualize the correlations between the features, with annotated values and a color scheme.

Displays the heatmap with the title "Correlation Heatmap - Kenya (Recent Data)".

```
# Take a recent sample from one country to show trend correlations
kenya_data = df_filtered[df_filtered['location'] == 'Kenya'].dropna()

# Select key numeric features
features = ['total_cases', 'new_cases', 'total_deaths', 'new_deaths', 'total_vaccinations']
corr = kenya_data[features].corr()
```

```
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap - Kenya (Recent Data)")
plt.tight_layout()
plt.show()
```



Loops through a list of selected countries.

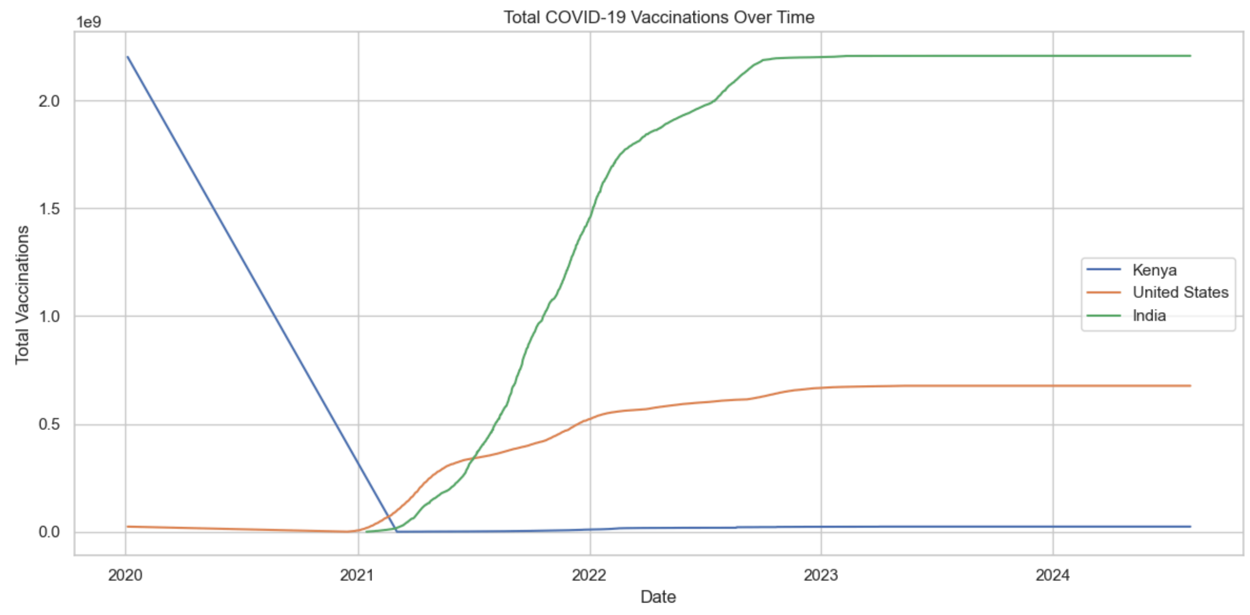
Filters the data for each country and plots the total number of COVID-19 vaccinations over time.

Sets the title, x-axis label ("Date"), and y-axis label ("Total Vaccinations").

Displays a legend to differentiate the countries on the plot.

Adjusts the layout for better spacing and shows the plot.

```
for country in countries:
    data = df_filtered[df_filtered['location'] == country]
    plt.plot(data['date'], data['total_vaccinations'], label=country)
plt.title('Total COVID-19 Vaccinations Over Time')
plt.xlabel('Date')
plt.ylabel('Total Vaccinations')
plt.legend()
plt.tight_layout()
plt.show()
```



Loops through a list of selected countries.

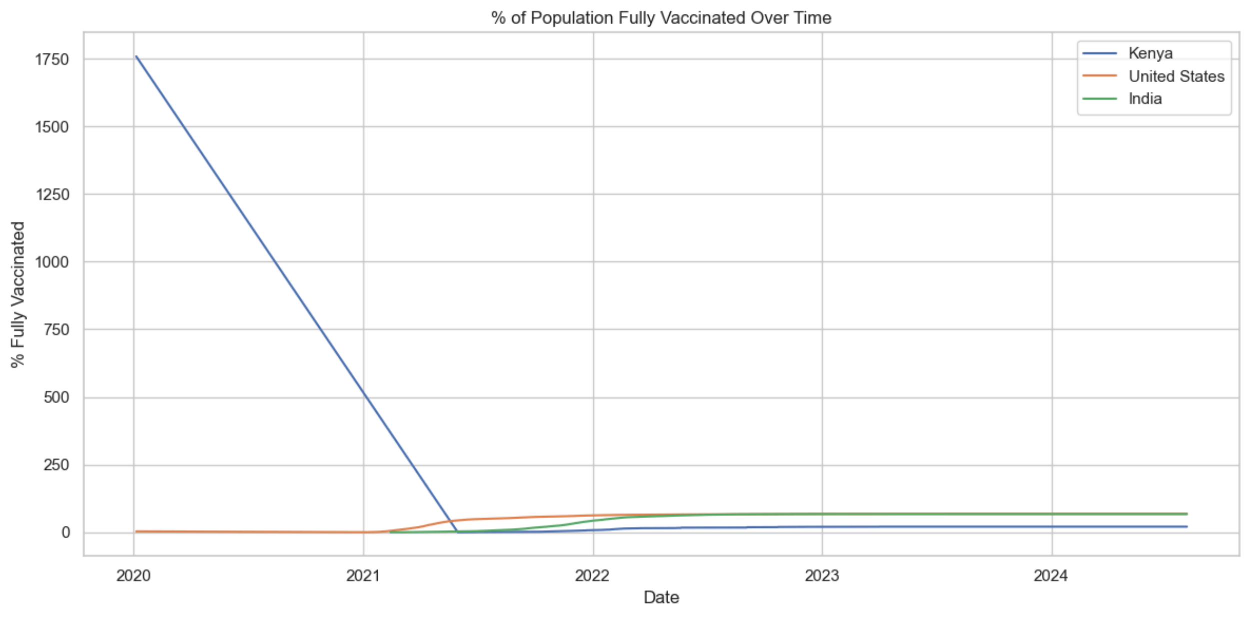
Filters the data for each country and plots the percentage of the population that is fully vaccinated over time.

Sets the title, x-axis label ("Date"), and y-axis label ("% Fully Vaccinated").

Displays a legend to differentiate the countries on the plot.

Adjusts the layout for better spacing and shows the plot.

```
for country in countries:
    data = df_filtered[df_filtered['location'] == country]
    plt.plot(data['date'], data['percent_fully_vaccinated'], label=country)
plt.title('% of Population Fully Vaccinated Over Time')
plt.xlabel('Date')
plt.ylabel('% Fully Vaccinated')
plt.legend()
plt.tight_layout()
plt.show()
```



Filters the dataset to get the latest data for each country by grouping the data by 'location' and selecting the row with the latest date (max()).

Loops through the selected countries.

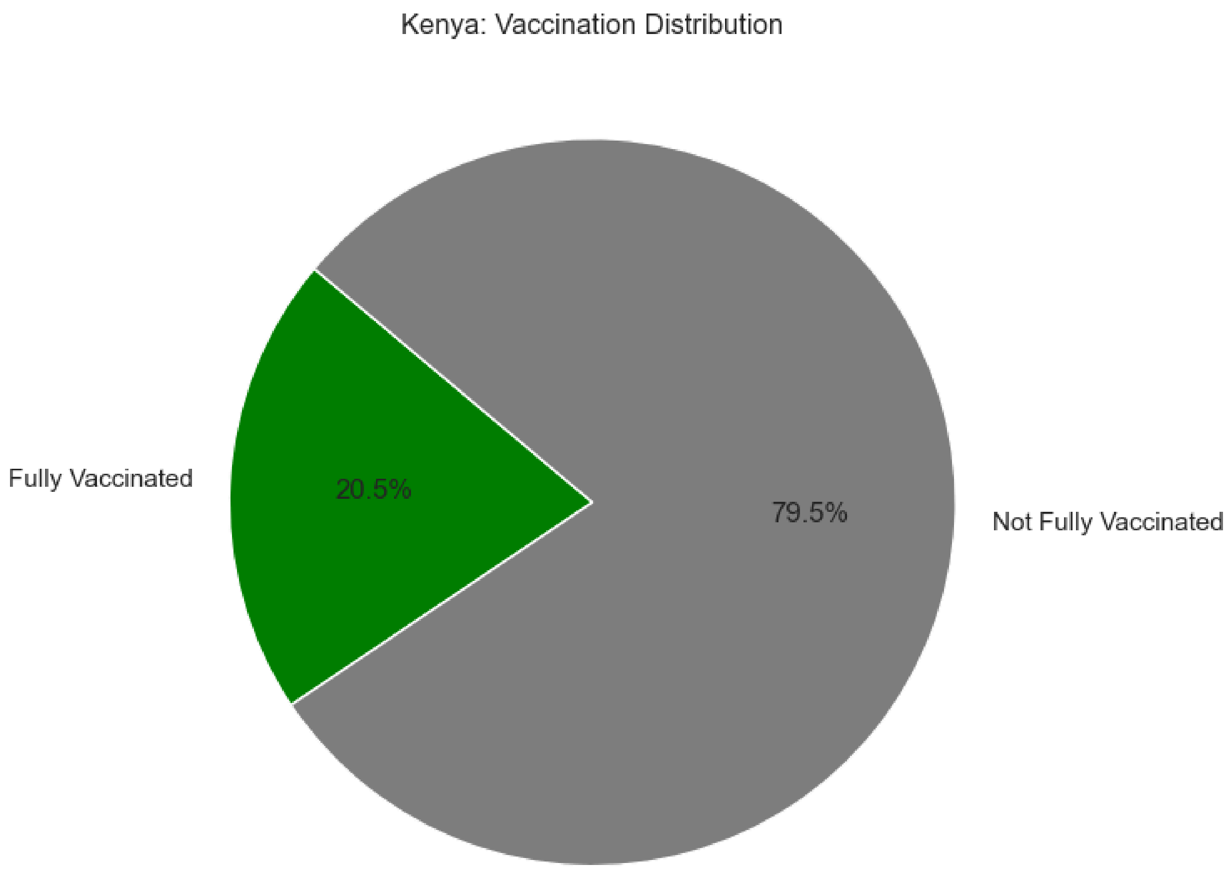
For each country, calculate the number of fully vaccinated and unvaccinated people based on the latest data available.

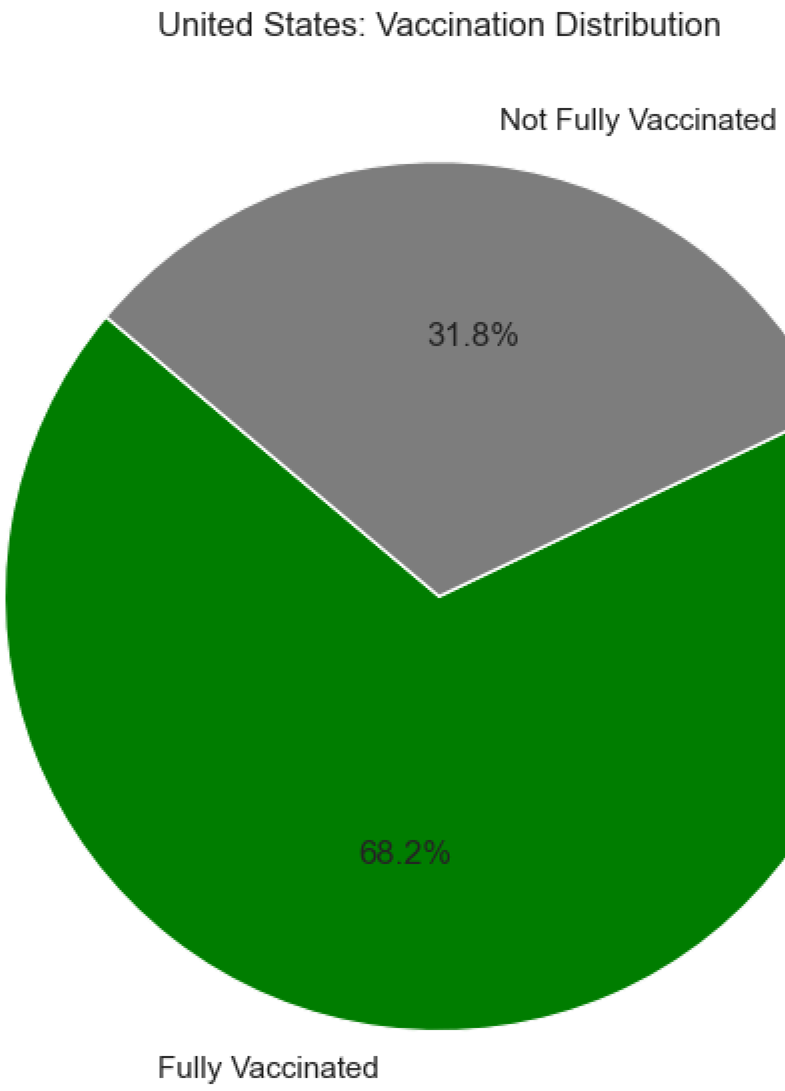
Creates a pie chart for each country showing the distribution between fully vaccinated and not fully vaccinated individuals.

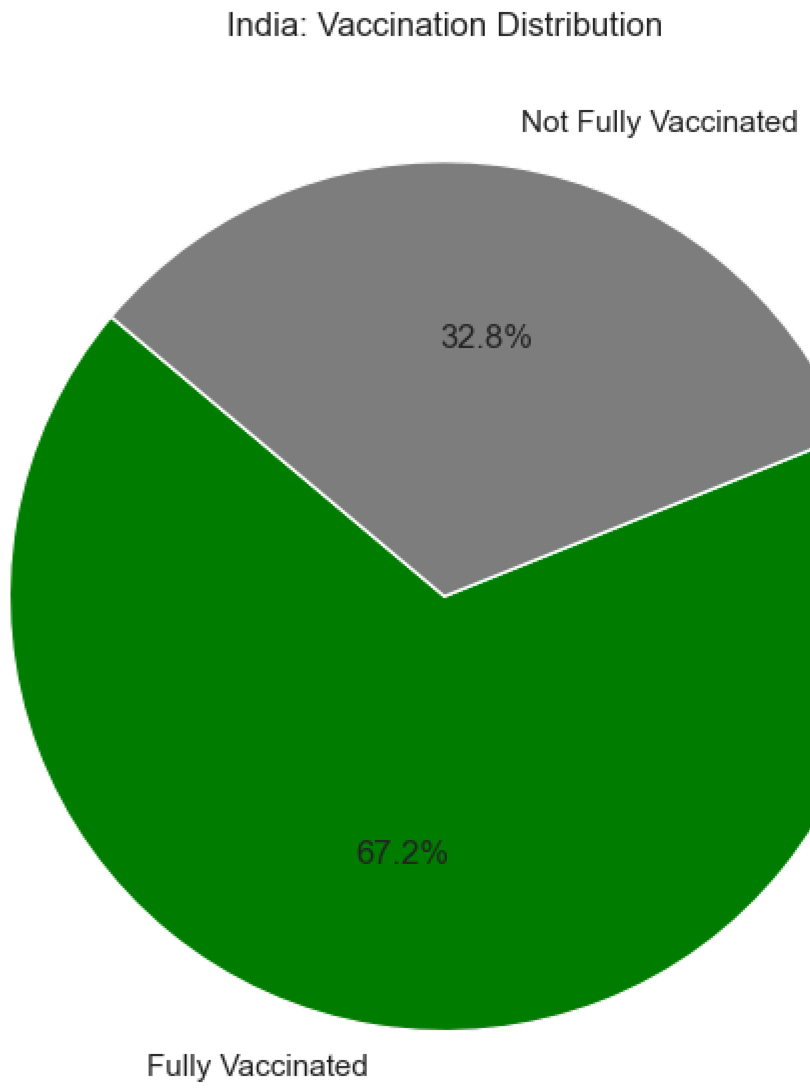
The chart is labeled with percentages and color-coded (green for fully vaccinated, gray for unvaccinated).

Each chart is displayed separately with a title indicating the country.

```
latest_data = df_filtered.groupby('location').apply(lambda g: g[g['date'] ==
g['date'].max()]).reset_index(drop=True)

for country in countries:
    row = latest_data[latest_data['location'] == country].iloc[0]
    vaccinated = row['people_fully_vaccinated']
    population = row['population']
    unvaccinated = population - vaccinated

    plt.figure()
    plt.pie([vaccinated, unvaccinated],
            labels=['Fully Vaccinated', 'Not Fully Vaccinated'],
            autopct='%1.1f%%',
            colors=['green', 'gray'],
            startangle=140)

    plt.title(f"{country}: Vaccination Distribution")
    plt.tight_layout()
    plt.show()
```


Kenya: Vaccination Distribution
Fully Vaccinated
20.5%
79.5%
Not Fully Vaccinated


United States: Vaccination Distribution
Not Fully Vaccinated
31.8%
68.2%
Fully Vaccinated



Extracts the latest data: It selects rows where the 'date' is the most recent date available in the dataset.

Handles missing data: Drops rows that do not have an 'iso_code', which is necessary for mapping country data.

Selects relevant columns: Creates a new dataframe (map_df) containing only the relevant columns, such as country ISO code, location (country name), total cases, total deaths, total vaccinations, people fully vaccinated, and population.

Calculates vaccination percentage: A new column, percent_fully_vaccinated, is added by dividing the number of fully vaccinated people by the total population, then multiplying by 100 to get the percentage of the population that is fully vaccinated.

```
# Get the latest date per country
latest_global = df[df['date'] == df['date'].max()].copy()

# Drop rows without ISO codes (required for mapping)
latest_global = latest_global.dropna(subset=['iso_code'])

# Select key columns
map_df = latest_global[['iso_code', 'location', 'total_cases', 'total_deaths',
                        'total_vaccinations', 'people_fully_vaccinated', 'population']].copy()

# Calculate % fully vaccinated
map_df['percent_fully_vaccinated'] = (map_df['people_fully_vaccinated'] / map_df['population'])
* 100
```

Sets the default Plotly renderer: It changes the default renderer for Plotly plots to be displayed in a web browser instead of inline or in other formats. This is useful when you want interactive visualizations that can be explored directly in your browser rather than in the Jupyter notebook or other environments.

```
import plotly.io as pio
pio.renderers.default = 'browser'  # Forces output in browser
```

px.choropleth: This function generates the choropleth map.

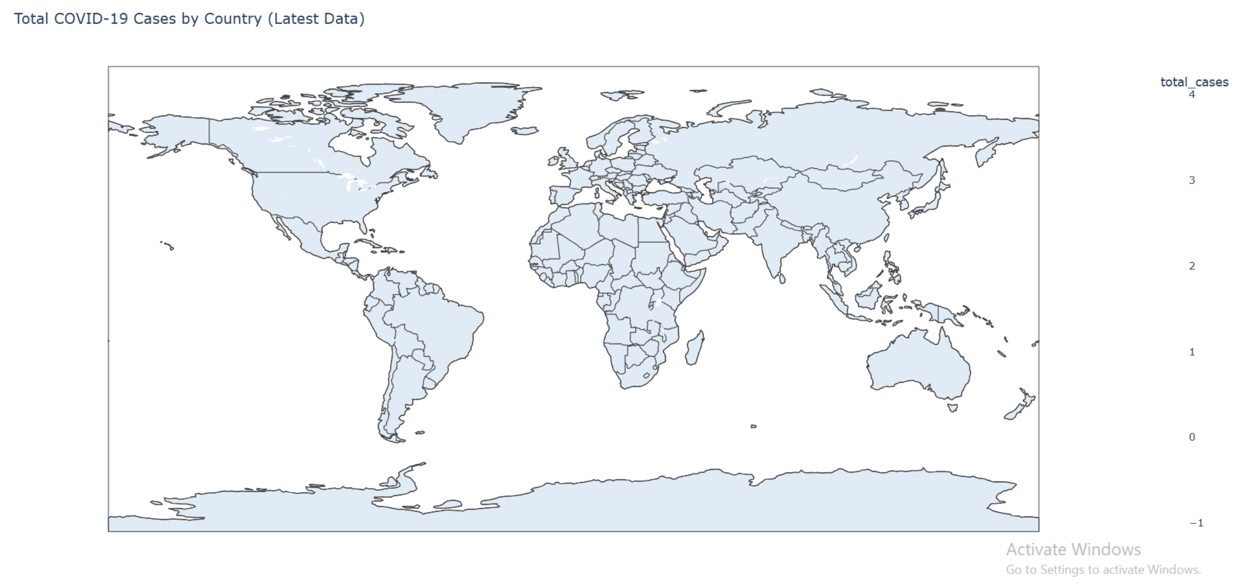
- **map_df**: The DataFrame containing the data to be visualized.

- **locations="iso_code"**: Specifies the geographical locations of each data point using the ISO country codes.
- **color="total_cases"**: The color of each country is determined by the total number of COVID-19 cases.
- **hover_name="location"**: When hovering over a country, it displays the country's name.
- **color_continuous_scale="Reds"**: The color scale used for representing the data, with red indicating higher numbers of cases.
- **title="Total COVID-19 Cases by Country (Latest Data)"**: Sets the title of the map.

fig_cases.update_geos(showcountries=True): This ensures that country boundaries are visible on the map.

fig_cases.show(): This displays the map in the browser or a notebook environment, allowing users to interact with it.

```
fig_cases = px.choropleth(
    map_df,
    locations="iso_code",
    color="total_cases",
    hover_name="location",
    color_continuous_scale="Reds",
    title="Total COVID-19 Cases by Country (Latest Data)",
)

fig_cases.update_geos(showcountries=True)
fig_cases.show()
```


Total COVID-19 Cases by Country (Latest Data)
total_cases
4
3
2
1
0
−1
Activate Windows
Go to Settings to activate Windows.

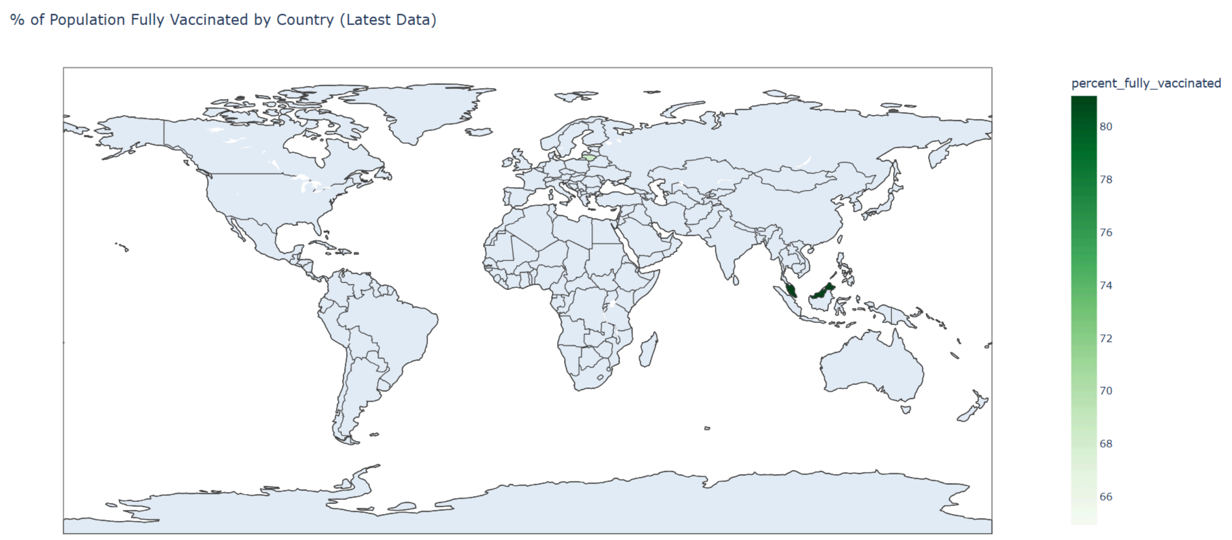
px.choropleth: This function is used to generate the choropleth map.

- **map_df**: The DataFrame containing the relevant data.
- **locations="iso_code"**: Specifies that the geographical locations are identified using ISO country codes.
- **color="percent_fully_vaccinated"**: The color of each country is determined by the percentage of the population that is fully vaccinated.
- **hover_name="location"**: When hovering over a country, it will show the country name.
- **color_continuous_scale="Greens"**: The color scale is set to green, where darker shades represent higher percentages of fully vaccinated populations.
- **title="% of Population Fully Vaccinated by Country (Latest Data)"**: The title for the map.

fig_vax.update_geos(showcountries=True): This makes sure the country boundaries are displayed on the map.

fig_vax.show(): This command will render the map in a browser or Jupyter notebook, allowing the user to interact with it.

```python
fig_vax = px.choropleth(
    map_df,
    locations="iso_code",
    color="percent_fully_vaccinated",
    hover_name="location",
    color_continuous_scale="Greens",
    title="% of Population Fully Vaccinated by Country (Latest Data)",
)

fig_vax.update_geos(showcountries=True)
fig_vax.show()
```


% of Population Fully Vaccinated by Country (Latest Data)
percent_fully_vaccinated
80
78
76
74
72
70
68
66