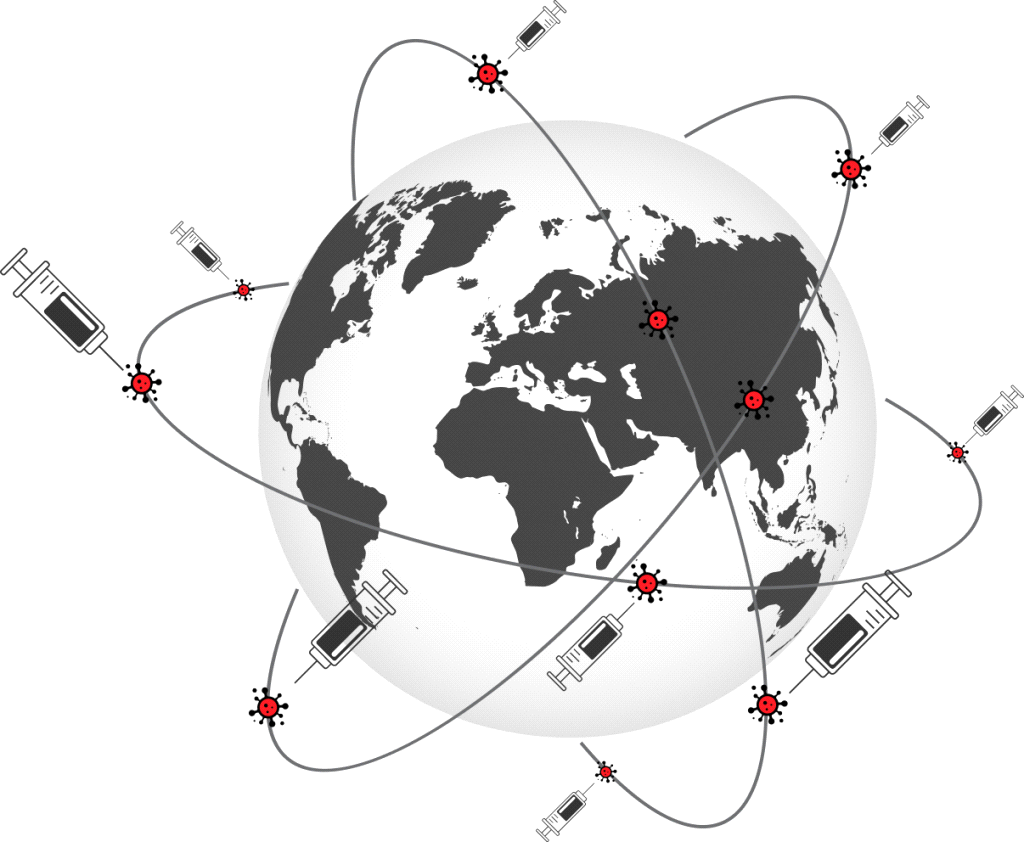
**COVID Vaccines Analysis**

**Phase5: Submission Document**

**Project Title:** COVID Vaccines Analysis

**Phase 5:** Project Documentation & Submission





COVID Vaccines Analysis

**COVID Vaccines Analysis**

**Introduction:**

* The "COVID Vaccines Analysis" project is a comprehensive exploration of critical aspects related to COVID-19 vaccines. This project delves into the data surrounding vaccine distribution, administration, and the occurrence of adverse effects, aiming to provide insights that contribute to informed decision-making and public health strategies.
* Our analysis encompasses a wide range of methodologies, from data mining and machine learning techniques to statistical modelling and trend analysis. By leveraging advanced data analytics, we seek to uncover hidden patterns, identify areas of concern, and make meaningful predictions regarding the trajectory of the pandemic and the success of vaccination efforts.
* Exploring advanced machine learning techniques like clustering and time series forecasting can be instrumental in uncovering hidden patterns in vaccine distribution and adverse effects data. Here's a more detailed explanation of how you can utilize these techniques.
* Exploring advanced machine learning techniques like clustering and time series forecasting can be instrumental in uncovering hidden patterns in vaccine distribution and adverse effects data. Here's a more detailed explanation of how you can utilize these techniques.

Advanced Machine Learning Techniques:

**Clustering Analysis:**

**Objective**: Clustering analysis groups similar data points together to reveal patterns and relationships within the data. In the context of your project on vaccine distribution and adverse effects, clustering can help identify geographical, temporal, or demographic clusters with similar characteristics.

**Implementation**:

* **Feature Selection**: Choose relevant features from your dataset, such as location (e.g., countries, states), time (e.g., months, weeks), vaccine types, and adverse effects.
* **Data Preprocessing**: Clean and preprocess your data to handle missing values, outliers, and normalize data if necessary.
* **Algorithm Selection**: Utilize clustering algorithms like K-Means, Hierarchical Clustering, or DBSCAN, depending on the nature of your data and the desired cluster shape.
* **Clustering Interpretation**: Analyze and interpret the clusters. For instance, you might discover clusters of regions with similar vaccination rates, clusters of time periods with similar adverse effects, or groups of people with similar vaccine response patterns.
* **Visualization**: Create visualizations, such as heatmaps or geographic maps, to illustrate the clusters and make it easier to communicate your findings.

**Time Series Forecasting:**

**Objective**: Time series forecasting involves predicting future values based on historical data. In the context of your project, time series forecasting can help in predicting future vaccine distribution trends and identifying potential outbreaks of adverse effects.

**Implementation**:

* **Data Preparation**: Organize your data into time series format with timestamps and relevant variables (e.g., vaccine distribution counts or adverse effect occurrences).
* **Train-Test Split**: Split your time series data into training and testing sets to evaluate the accuracy of your forecasting model.
* **Model Selection**: Choose an appropriate time series forecasting model like ARIMA (AutoRegressive Integrated Moving Average), Exponential Smoothing, or Long Short-Term Memory (LSTM) networks for deep learning.
* **Hyperparameter Tuning**: Fine-tune the model hyperparameters to optimize performance.
* **Forecasting and Evaluation**: Train the model on the training data and use it to make predictions for the test set. Evaluate the model's accuracy using metrics like Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE).
* **Visualization**: Visualize the predicted values alongside the actual data to assess the model's performance and identify trends or anomalies.

Data Loading & Overview:

The present notebook was written in a local environment using Jupyter Lab. As always, in case any of the libraries used in the present notebook is not installed in the environment where the reader intends to run it, please ensure their proper installation beforehand. The following command downloads the dataset from the corresponding GitHub repository.

Data Preprocessing:

Taking the above into consideration, the next step is the construction of a filtered version of the full DataFrame, which includes only the January - February 2021 time period. The reason for this is that - up to the day that this notebook is written - the OWiD dataset on Covid-19 is still being expanded. Consequently, any conclusions that may be drawn as part of the present analysis on the full dataset may be altered in the future, when more data become available and a reader attempts to run the notebook as it is. Before doing that, we make sure that the date feature is transformed into a date type object.

Exploratory Data Analysis:

Before diving into the EDA, we import some libraries and also present some helper functions and commands that will be utilized further down the road for visualizations.

**Exploratory data analysis:**

"In this comprehensive exploratory data analysis (EDA), we delve into the wealth of information provided by the 'COVID-19 World Vaccination Progress' dataset from Kaggle. Our aim is to gain a deep understanding of the global landscape of COVID-19 vaccination efforts. Through this analysis, we explore the trends, patterns, and insights that emerge from the data. We examine the distribution of vaccine types, demographics, and regional disparities in vaccination rates. We also investigate potential correlations between vaccine distribution and COVID-19 infection rates. Our data-driven approach not only provides a snapshot of the progress made in the fight against the pandemic but also offers valuable insights to inform public health policies and interventions. Join us on this data journey as we uncover the narrative behind the numbers and visualize the path towards global herd immunity."

**Statistical analysis:**

"In addition to the exploratory data analysis, we performed a comprehensive statistical analysis to extract meaningful insights from the 'COVID-19 World Vaccination Progress' dataset. This analysis involved conducting various statistical tests, such as t-tests and chi-squared tests, to evaluate significant differences in vaccination rates between different demographic groups and regions. We also employed correlation analysis to assess the relationships between variables, including vaccine distribution and COVID-19 infection rates. Time series analysis allowed us to identify trends and seasonality in vaccination progress over time. Furthermore, we used regression analysis to model the factors influencing vaccination rates, shedding light on the impact of various variables, such as healthcare infrastructure, population density, and socioeconomic factors. Our statistical analysis aims to provide a quantitative perspective on the patterns and determinants of COVID-19 vaccine distribution, contributing to evidence-based decision-making in the ongoing fight against the pandemic."

**Visualization:**

"As part of our analysis of the 'COVID-19 World Vaccination Progress' dataset, we leveraged various visualization techniques to bring the data to life. We employed bar charts and pie charts to vividly illustrate the distribution of vaccine types, age groups, and gender among those receiving vaccinations. Time series line plots and heatmaps were utilized to visualize trends and correlations over time and between variables. Choropleth maps enabled us to portray the geographic distribution of vaccination rates, highlighting regional disparities. We also created interactive dashboards that offer a user-friendly interface for exploring the data visually and dynamically. These visualizations serve as powerful tools to convey the intricate narrative within the dataset, making it more accessible and informative for a broad audience. By combining these techniques, we aim to provide a holistic and insightful perspective on the global COVID-19 vaccination progress."

**Given data set:**

**Dataset Link:** <https://www.kaggle.com/datasets/gpreda/covid-world-vaccination-progress>

**Program:**

IN[1]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import os

for dirname, \_, filenames **in** os.walk('/kaggle/input'):

for filename **in** filenames:

print(os.path.join(dirname, filename))

/opt/conda/lib/python3.10/site-packages/scipy/\_\_init\_\_.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

warnings.warn(f"A NumPy version >={np\_minversion} and <{np\_maxversion}"

/kaggle/input/covid-world-vaccination-progress/country\_vaccinations\_by\_manufacturer.csv

/kaggle/input/covid-world-vaccination-progress/country\_vaccinations.csv

In [2]:

data = pd.read\_csv("/kaggle/input/covid-world-vaccination-progress/country\_vaccinations.csv")

data.head()

Out[2]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 |
| 0 | Afghanistan | AFG | 2021-02-22 | 0.0 | 0.0 | NaN | NaN | NaN | 0.0 | 0.0 | NaN | NaN | Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi... | World Health Organization | <https://covid19.who.int/> |
| 1 | Afghanistan | AFG | 2021-02-23 | NaN | NaN | NaN | NaN | 1367.0 | NaN | NaN | NaN | 34.0 | Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi... | World Health Organization | <https://covid19.who.int/> |
| 2 | Afghanistan | AFG | 2021-02-24 | NaN | NaN | NaN | NaN | 1367.0 | NaN | NaN | NaN | 34.0 | Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi... | World Health Organization | <https://covid19.who.int/> |
| 3 | Afghanistan | AFG | 2021-02-25 | NaN | NaN | NaN | NaN | 1367.0 | NaN | NaN | NaN | 34.0 | Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi... | World Health Organization | <https://covid19.who.int/> |
| 4 | Afghanistan | AFG | 2021-02-26 | NaN | NaN | NaN | NaN | 1367.0 | NaN | NaN | NaN | 34.0 | Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi... | World Health Organization | <https://covid19.who.int/> |

In [3]:

data.describe()

Out[3]:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 |
| count | 4.360700e+04 | 4.129400e+04 | 3.880200e+04 | 3.536200e+04 | 8.621300e+04 | 43607.000000 | 41294.000000 | 38802.000000 | 86213.000000 |
| mean | 4.592964e+07 | 1.770508e+07 | 1.413830e+07 | 2.705996e+05 | 1.313055e+05 | 80.188543 | 40.927317 | 35.523243 | 3257.049157 |
| std | 2.246004e+08 | 7.078731e+07 | 5.713920e+07 | 1.212427e+06 | 7.682388e+05 | 67.913577 | 29.290759 | 28.376252 | 3934.312440 |
| min | 0.000000e+00 | 0.000000e+00 | 1.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 5.264100e+05 | 3.494642e+05 | 2.439622e+05 | 4.668000e+03 | 9.000000e+02 | 16.050000 | 11.370000 | 7.020000 | 636.000000 |
| 50% | 3.590096e+06 | 2.187310e+06 | 1.722140e+06 | 2.530900e+04 | 7.343000e+03 | 67.520000 | 41.435000 | 31.750000 | 2050.000000 |
| 75% | 1.701230e+07 | 9.152520e+06 | 7.559870e+06 | 1.234925e+05 | 4.409800e+04 | 132.735000 | 67.910000 | 62.080000 | 4682.000000 |
| max | 3.263129e+09 | 1.275541e+09 | 1.240777e+09 | 2.474100e+07 | 2.242429e+07 | 345.370000 | 124.760000 | 122.370000 | 117497.000000 |

In [4]:

pd.to\_datetime(data.date)

data.country.value\_counts()

Out[4]:

Norway 482

Latvia 480

Denmark 476

United States 471

Russia 470

...

Bonaire Sint Eustatius and Saba 146

Tokelau 114

Saint Helena 92

Pitcairn 85

Falkland Islands 67

Name: country, Length: 223, dtype: int64

In [5]:

data = data[data.country.apply(lambda x: x **not** **in** ["England", "Scotland", "Wales", "Northern Ireland"])]

data.country.value\_counts()

Out[5]:

Norway 482

Latvia 480

Denmark 476

United States 471

Canada 470

...

Bonaire Sint Eustatius and Saba 146

Tokelau 114

Saint Helena 92

Pitcairn 85

Falkland Islands 67

Name: country, Length: 219, dtype: int64

In [6]:

data.vaccines.value\_counts()

Out[6]:

Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech 7608

Oxford/AstraZeneca 6022

Oxford/AstraZeneca, Pfizer/BioNTech 4629

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech 4491

Johnson&Johnson, Moderna, Novavax, Oxford/AstraZeneca, Pfizer/BioNTech 3564

...

Johnson&Johnson, Oxford/AstraZeneca, Sinovac 312

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac, Sputnik V 311

Johnson&Johnson, Moderna 251

Johnson&Johnson, Pfizer/BioNTech, Sinopharm/Beijing 228

EpiVacCorona, Oxford/AstraZeneca, QazVac, Sinopharm/Beijing, Sputnik V, ZF2001 190

Name: vaccines, Length: 84, dtype: int64

In [7]:

df = data[["vaccines", "country"]]

df.head()

Out[7]:

|  |  |  |
| --- | --- | --- |
|  | vaccines | country |
| 0 | Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi... | Afghanistan |
| 1 | Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi... | Afghanistan |
| 2 | Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi... | Afghanistan |
| 3 | Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi... | Afghanistan |
| 4 | Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi... | Afghanistan |

In [8]:

dict\_ = {}

for i **in** df.vaccines.unique():

dict\_[i] = [df["country"][j] for j **in** df[df["vaccines"]==i].index]

vaccines = {}

for key, value **in** dict\_.items():

vaccines[key] = set(value)

for i, j **in** vaccines.items():

print(f"**{**i**}**:>>**{**j**}**")

Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing:>>{'Cameroon', 'Afghanistan', 'Belize', 'Namibia', 'Trinidad and Tobago'}

Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac, Sputnik V:>>{'Oman', 'Bosnia and Herzegovina', 'Albania', 'Azerbaijan'}

Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac, Sputnik V:>>{'Algeria', 'Zimbabwe'}

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech:>>{'Guernsey', 'United Kingdom', 'Fiji', 'Sweden', 'Australia', 'Jersey', 'Sint Maarten (Dutch part)', 'Finland', 'Andorra', 'Japan', 'Isle of Man'}

Oxford/AstraZeneca:>>{'Montserrat', 'Kiribati', 'Saint Helena', 'Saint Vincent and the Grenadines', 'Liberia', 'Falkland Islands', 'Solomon Islands', 'Tuvalu', 'Vanuatu', 'Democratic Republic of Congo', 'Pitcairn', 'Mali', 'Papua New Guinea', 'Nigeria', 'Samoa', 'Nauru', 'Togo', 'Angola', 'Tonga', 'Sao Tome and Principe'}

Oxford/AstraZeneca, Pfizer/BioNTech:>>{'Anguilla', 'Saudi Arabia', 'Gibraltar', 'Saint Lucia', 'Cayman Islands', 'New Zealand', 'Saint Kitts and Nevis', 'Panama', 'Bermuda', 'Costa Rica', 'Kosovo'}

Oxford/AstraZeneca, Pfizer/BioNTech, Sputnik V:>>{'Antigua and Barbuda'}

CanSino, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V:>>{'Argentina'}

Moderna, Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac, Sputnik V:>>{'Armenia'}

Pfizer/BioNTech:>>{'Monaco', 'Tokelau', 'Cook Islands', 'New Caledonia', 'Turks and Caicos Islands', 'Niue', 'Aruba'}

Johnson&Johnson, Moderna, Novavax, Oxford/AstraZeneca, Pfizer/BioNTech:>>{'Czechia', 'Slovenia', 'Netherlands', 'Germany', 'Austria', 'South Korea', 'Lithuania', 'Italy'}

Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech:>>{'Bahamas', 'Eswatini', 'Grenada'}

Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik Light, Sputnik V:>>{'Bahrain'}

Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac:>>{'Bangladesh'}

Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing:>>{'Maldives', 'Peru', 'Suriname', 'Barbados', 'Dominica'}

Sinopharm/Beijing, Sputnik V:>>{'Belarus', 'Kyrgyzstan'}

Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech:>>{'Cyprus', 'Portugal', 'Iceland', 'Malta', 'Belgium', 'Croatia', 'Jamaica', 'Luxembourg', 'Poland', 'France', 'Greece', 'Spain', 'Romania', 'Bulgaria', 'Estonia', 'Ireland', 'Canada'}

Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac:>>{'Benin', 'Brazil'}

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing:>>{'Cape Verde', 'Bhutan'}

Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V:>>{'Moldova', "Cote d'Ivoire", 'Morocco', 'Bolivia'}

Moderna, Pfizer/BioNTech:>>{'Faeroe Islands', 'Norway', 'Bonaire Sint Eustatius and Saba', 'Curacao', 'Qatar', 'Israel'}

Covaxin, Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac:>>{'Botswana'}

Johnson&Johnson, Oxford/AstraZeneca:>>{'British Virgin Islands', 'South Sudan', 'Malawi'}

Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing:>>{'Nepal', 'Brunei', 'Kenya', 'Kuwait'}

Johnson&Johnson, Oxford/AstraZeneca, Sinopharm/Beijing:>>{'Gambia', 'Mozambique', 'Madagascar', 'Senegal', 'Lesotho', 'Zambia', 'Burkina Faso'}

Sinopharm/Beijing:>>{'Equatorial Guinea', 'Burundi', 'Chad'}

Johnson&Johnson, Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac:>>{'Somalia', 'Cambodia'}

Covaxin, Oxford/AstraZeneca:>>{'Central African Republic'}

CanSino, Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac:>>{'Chile', 'Ecuador'}

CanSino, Sinopharm/Beijing, Sinopharm/Wuhan, Sinovac, ZF2001:>>{'China'}

Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac:>>{'Uganda', 'Ukraine', 'Colombia'}

Covaxin, Oxford/AstraZeneca, Sinopharm/Beijing:>>{'Mauritius', 'Comoros'}

Moderna, Oxford/AstraZeneca, Sinopharm/Beijing, Sputnik V:>>{'Congo'}

Abdala, Soberana Plus, Soberana02:>>{'Cuba'}

Johnson&Johnson, Moderna, Pfizer/BioNTech:>>{'United States', 'Liechtenstein', 'Denmark', 'Switzerland'}

Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac, Sputnik V:>>{'Egypt', 'Djibouti', 'Guinea'}

Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac:>>{'Dominican Republic', 'Georgia', 'El Salvador'}

Covaxin, Johnson&Johnson, Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac:>>{'Ethiopia'}

Johnson&Johnson, Pfizer/BioNTech:>>{'South Africa', 'French Polynesia'}

Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V:>>{'Gabon'}

Oxford/AstraZeneca, Sputnik V:>>{'Ghana'}

Moderna:>>{'Greenland', 'Wallis and Futuna'}

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sputnik V:>>{'Guatemala'}

Oxford/AstraZeneca, Sinopharm/Beijing:>>{'Niger', 'Myanmar', 'Mauritania', 'Sierra Leone', 'Guinea-Bissau'}

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V:>>{'Sri Lanka', 'Guyana'}

Johnson&Johnson, Moderna:>>{'Haiti'}

Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sputnik V:>>{'Honduras'}

Pfizer/BioNTech, Sinovac:>>{'Hong Kong'}

Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V:>>{'Hungary', 'Jordan'}

Covaxin, Oxford/AstraZeneca, Sputnik V:>>{'India'}

Johnson&Johnson, Moderna, Novavax, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac:>>{'Indonesia'}

COVIran Barekat, Covaxin, FAKHRAVAC, Oxford/AstraZeneca, Razi Cov Pars, Sinopharm/Beijing, Soberana02, SpikoGen, Sputnik V:>>{'Iran'}

Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V:>>{'Lebanon', 'Iraq', 'Montenegro', 'Mongolia', 'Serbia'}

QazVac, Sinopharm/Beijing, Sputnik V:>>{'Kazakhstan'}

Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac, Sputnik Light, Sputnik V:>>{'Laos'}

Johnson&Johnson, Moderna, Novavax, Pfizer/BioNTech:>>{'Latvia'}

Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac, Sputnik V:>>{'North Macedonia', 'Libya'}

Pfizer/BioNTech, Sinopharm/Beijing:>>{'Macao'}

CanSino, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac:>>{'Malaysia'}

CanSino, Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac, Sputnik V:>>{'Mexico'}

Abdala, Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech, Soberana02, Sputnik Light, Sputnik V:>>{'Nicaragua'}

Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac:>>{'Northern Cyprus', 'Timor', 'Uruguay'}

CanSino, Covaxin, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac, Sputnik V:>>{'Pakistan'}

Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac, Sputnik Light, Sputnik V:>>{'Palestine', 'Philippines'}

Covaxin, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac, Sputnik V:>>{'Paraguay'}

EpiVacCorona, Sputnik V:>>{'Russia'}

Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac, Sputnik V:>>{'Tunisia', 'Rwanda'}

Pfizer/BioNTech, Sputnik V:>>{'San Marino'}

Oxford/AstraZeneca, Sinopharm/Beijing, Sputnik V:>>{'Seychelles'}

Moderna, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac:>>{'Singapore'}

Johnson&Johnson, Moderna, Novavax, Oxford/AstraZeneca, Pfizer/BioNTech, Sputnik V:>>{'Slovakia'}

Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac:>>{'Sudan'}

Johnson&Johnson, Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac, Sputnik Light, Sputnik V:>>{'Syria'}

Medigen, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech:>>{'Taiwan'}

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac, Sputnik V:>>{'Tajikistan'}

Johnson&Johnson, Pfizer/BioNTech, Sinopharm/Beijing:>>{'Tanzania'}

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac:>>{'Thailand'}

Pfizer/BioNTech, Sinovac, Turkovac:>>{'Turkey'}

EpiVacCorona, Oxford/AstraZeneca, QazVac, Sinopharm/Beijing, Sputnik V, ZF2001:>>{'Turkmenistan'}

Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinopharm/Wuhan, Sputnik V:>>{'United Arab Emirates'}

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac, Sputnik Light, Sputnik V, ZF2001:>>{'Uzbekistan'}

Abdala, Sinopharm/Beijing, Sinovac, Soberana02, Sputnik Light, Sputnik V:>>{'Venezuela'}

Abdala, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V:>>{'Vietnam'}

Johnson&Johnson, Oxford/AstraZeneca, Sinovac:>>{'Yemen'}

In [9]:

linkcode

import plotly.express as px

import plotly.offline as py

vaccine\_map = px.choropleth(data, locations = 'iso\_code', color = 'vaccines')

vaccine\_map.update\_layout(height=300, margin={"r":0,"t":0,"l":0,"b":0})

vaccine\_map.show()

Advanced Machine Learning Techniques:

IN[1]: import numpy as np

IN[2]: import pandas as pd

import os

for dirname, \_, filenames **in** os.walk('/kaggle/input'):

for filename **in** filenames:

print(os.path.join(dirname, filename))

IN[3]: df = pd.read\_csv("/kaggle/input/covid-world-vaccination-progress/country\_vac cinations\_by\_manufacturer.csv")

IN[4]: df.head(10)

OUT[4]:

|  |  |  |  |
| --- | --- | --- | --- |
| location | date | vaccine | total\_vaccinations |
| 0 | Argentina | 2020-12-29 | Moderna | 2 |
| 1 | Argentina | 2020-12-29 | Oxford/AstraZeneca | 3 |
| 2 | Argentina | 2020-12-29 | Sinopharm/Beijing | 1 |
| 3 | Argentina | 2020-12-29 | Sputnik V | 20481 |
| 4 | Argentina | 2020-12-30 | Moderna | 2 |
| 5 | Argentina | 2020-12-30 | Oxford/AstraZeneca | 3 |
| 6 | Argentina | 2020-12-30 | Sinopharm/Beijing | 1 |
| 7 | Argentina | 2020-12-30 | Sputnik V | 40583 |
| 8 | Argentina | 2020-12-31 | Moderna | 2 |
| 9 | Argentina | 2020-12-31 | Oxford/AstraZeneca | 3 |

IN[5]: df["location"].nunique()

OUT[5]: 43

IN[6]: df.isnull().sum()

OUT[6]:

location 0

date 0

vaccine 0

total\_vaccinations 0

dtype: int64

IN[7]: df.dtypes

OUT[7]:

location object

date object

vaccine object

total\_vaccinations int64

dtype: object

IN[8]: df['date'] = pd.to\_datetime(df['date'])

IN[9]: data=pd.DataFrame(columns=['Country', 'Vaccine', 'Total\_vaccine'])

for country **in** df["location"].unique():

for vaccine **in** df["vaccine"].unique():

filtered\_data = df[(df['location'] == country) & (df['vaccine'] == vaccine)]

total\_count = filtered\_data['total\_vaccinations'].max()

data = pd.concat([data, pd.DataFrame({'Country': [country], 'Vaccine': [vaccine], 'Total\_vaccine': [total\_count]})], ignore\_index=True)

IN[10]: data.head(10)

OUT[10]:

|  |  |  |
| --- | --- | --- |
| Country | Vaccine | Total\_vaccine |
| 0 | Argentina | Moderna | 6507561 |
| 1 | Argentina | Oxford/AstraZeneca | 25977231 |
| 2 | Argentina | Sinopharm/Beijing | 28322602 |
| 3 | Argentina | Sputnik V | 20405678 |
| 4 | Argentina | CanSino | 610540 |
| 5 | Argentina | Pfizer/BioNTech | 14681054 |
| 6 | Argentina | Johnson&Johnson | NaN |
| 7 | Argentina | Novavax | NaN |
| 8 | Argentina | Sinovac | NaN |
| 9 | Argentina | Covaxin | NaN |

IN[11]:data.dropna(axis=0,inplace=True)

In [12]:

IN[12]:data.head(20)

OUT[12]:

|  |  |  |
| --- | --- | --- |
| Country | Vaccine | Total\_vaccine |
| 0 | Argentina | Moderna | 6507561 |
| 1 | Argentina | Oxford/AstraZeneca | 25977231 |
| 2 | Argentina | Sinopharm/Beijing | 28322602 |
| 3 | Argentina | Sputnik V | 20405678 |
| 4 | Argentina | CanSino | 610540 |
| 5 | Argentina | Pfizer/BioNTech | 14681054 |
| 10 | Austria | Moderna | 1585063 |
| 11 | Austria | Oxford/AstraZeneca | 1588222 |
| 15 | Austria | Pfizer/BioNTech | 14584985 |
| 16 | Austria | Johnson&Johnson | 363548 |
| 17 | Austria | Novavax | 3682 |
| 20 | Belgium | Moderna | 4267394 |
| 21 | Belgium | Oxford/AstraZeneca | 2846716 |
| 25 | Belgium | Pfizer/BioNTech | 17451842 |
| 26 | Belgium | Johnson&Johnson | 425639 |
| 27 | Belgium | Novavax | 36 |
| 30 | Bulgaria | Moderna | 491663 |
| 31 | Bulgaria | Oxford/AstraZeneca | 478541 |
| 35 | Bulgaria | Pfizer/BioNTech | 2852218 |
| 36 | Bulgaria | Johnson&Johnson | 511702 |

IN[13]: data\_2=pd.DataFrame(columns=['Country', 'Vaccine'])

data["Total\_vaccine"] = pd.to\_numeric(data["Total\_vaccine"], errors="coerce")

for country **in** data["Country"].unique():

new\_data = data[data["Country"] == country]

max\_vaccine = new\_data.loc[new\_data["Total\_vaccine"].idxmax(), "Vaccine"]

data\_2 = pd.concat([data\_2, pd.DataFrame({'Country': [country], 'Vaccine': [max\_vaccine]})], ignore\_index=True)

IN[14]: data\_2.head()

OUT[14]:

Out[14]:

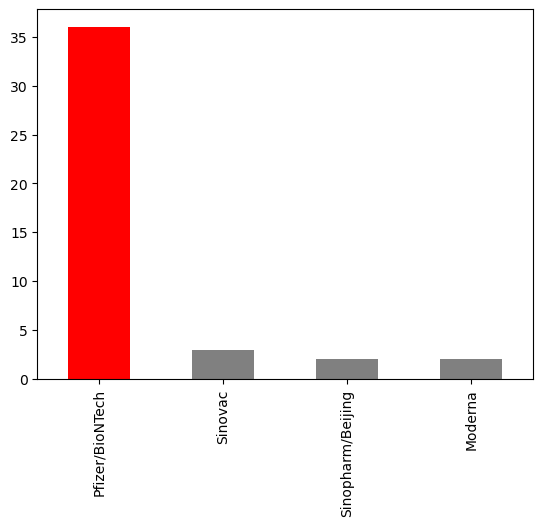
|  |  |  |
| --- | --- | --- |
|  | Country | Vaccine |
| 0 | Argentina | Sinopharm/Beijing |
| 1 | Austria | Pfizer/BioNTech |
| 2 | Belgium | Pfizer/BioNTech |
| 3 | Bulgaria | Pfizer/BioNTech |
| 4 | Chile | Sinovac |

IN[15]:data\_2["Vaccine"].value\_counts().plot(kind="bar",

color=["Red","Gray","Gray","Gray"])

OUT[15]:

<Axes: >



IN[16]:number\_of\_days = (df["date"].max() -df["date"].min() ).days

In [17]:

dtfrm=data[data["Vaccine"]=="Pfizer/BioNTech"]

dtfrm = dtfrm.drop(dtfrm[dtfrm['Country'] == 'European Union'].index)

In [18]:

dtfrm.head(10)

dtfrm.head(10)

Out[18]:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Country | Vaccine | Total\_vaccine |
| 5 | Argentina | Pfizer/BioNTech | 14681054 |
| 15 | Austria | Pfizer/BioNTech | 14584985 |
| 25 | Belgium | Pfizer/BioNTech | 17451842 |
| 35 | Bulgaria | Pfizer/BioNTech | 2852218 |
| 45 | Chile | Pfizer/BioNTech | 7910264 |
| 55 | Croatia | Pfizer/BioNTech | 3921503 |
| 65 | Cyprus | Pfizer/BioNTech | 1188656 |
| 75 | Czechia | Pfizer/BioNTech | 14604323 |
| 85 | Denmark | Pfizer/BioNTech | 10259219 |
| 95 | Ecuador | Pfizer/BioNTech | 8552679 |

In [19]:

dtfrm["average\_vaccination\_count"] = dtfrm["Total\_vaccine"] / number\_of\_days

dtfrm["average\_vaccination\_count"] =dtfrm["average\_vaccination\_count"].astype(int)

In [20]:

dtfrm.head(15)

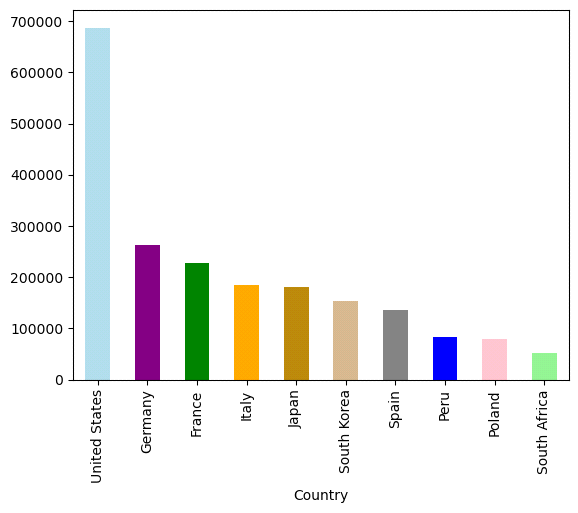
IN[21]:

color=["Lightblue","Purple","Green","Orange","darkgoldenrod","tan","Gray","Blue","Pink","Lightgreen"]

dtfrm["average\_vaccination\_count"].sort\_values(ascending=False).head(10).plot(kind="bar",color=color)

Out[22]:

<Axes: xlabel='Country'>



**Number of countries where vaccines are used**

In [23]:

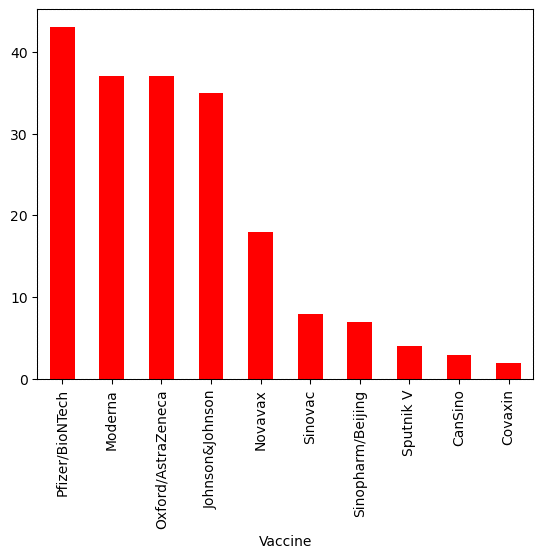
number\_of\_vaccines = data.groupby('Vaccine')['Country'].nunique()

In [24]:

number\_of\_vaccines.sort\_values(ascending=False).plot(kind="bar",color="r")

Out[24]:

<Axes: xlabel='Vaccine'>



IN[25]:

fig = px.choropleth(data\_frame=dtfrm,

locations=dtfrm.index,

locationmode='country names',

color='Total\_vaccine',

color\_continuous\_scale='YlOrRd',

title='Ülkelerde Yapılan Biontech Aşıları')

fig.update\_layout(title\_x=0.5)

IN[26]:

m = folium.Map(location=[0, 0], zoom\_start=2)

Choropleth(

geo\_data='https://raw.githubusercontent.com/johan/world.geo.json/master/countries.geo.json',

name='choropleth',

data=dtfrm,

columns=[dtfrm.index, 'Total\_vaccine'],

key\_on='feature.properties.name',

fill\_color='YlOrRd',

fill\_opacity=0.7,

line\_opacity=0.2,

legend\_name='Aşı Sayısı',

).add\_to(m)

Out[26]:

<folium.features.Choropleth at 0x7d414f2b7430>

Data Loading & Overview:

In [1]:

import warnings

warnings.filterwarnings("ignore")

!pip install pyspark

file = '../input/our-world-in-data-covid19-dataset/owid-covid-data.csv'

Collecting pyspark

Downloading pyspark-3.2.1.tar.gz (281.4 MB)

|████████████████████████████████| 281.4 MB 30 kB/s

Collecting py4j==0.10.9.3

Downloading py4j-0.10.9.3-py2.py3-none-any.whl (198 kB)

|████████████████████████████████| 198 kB 31.9 MB/s

Building wheels for collected packages: pyspark

Building wheel for pyspark (setup.py) ... - \ | / - \ | / - \ | / - \ | / - \ | / - \ | / - \ | / - \ | / - \ | / - \ | / - \ | / - \ | / - \ | / - \ | / - \ | / - done

Created wheel for pyspark: filename=pyspark-3.2.1-py2.py3-none-any.whl size=281853642 sha256=ede66e36a6e40ca9a3d6ce73a552588e3d117402922c41e77f61cefd5ba329b6

Stored in directory: /root/.cache/pip/wheels/9f/f5/07/7cd8017084dce4e93e84e92efd1e1d5334db05f2e83bcef74f

Successfully built pyspark

Installing collected packages: py4j, pyspark

Successfully installed py4j-0.10.9.3 pyspark-3.2.1

WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: <https://pip.pypa.io/warnings/venv>

After initializing a PySpark session, the data are transformed into a PySpark DataFrame.

In [2]:

import pyspark

from pyspark.sql import SparkSession, SQLContext

spark = SparkSession.builder.appName("Covid Data Mining").config('spark.sql.debug.maxToStringFields', 2000).getOrCreate()

full\_df = spark.read.csv(file, header=True, inferSchema=True)

WARNING: An illegal reflective access operation has occurred

WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform ([file:/opt/conda/lib/python3.7/site-packages/pyspark/jars/spark-unsafe\_2.12-3.2.1.jar](file://\opt\conda\lib\python3.7\site-packages\pyspark\jars\spark-unsafe_2.12-3.2.1.jar)) to constructor java.nio.DirectByteBuffer(long,int)

WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform

WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations

WARNING: All illegal access operations will be denied in a future release

Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties

Setting default log level to "WARN".

To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

22/04/10 14:02:16 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

Let's first identify the total number of samples, as well as the number of each sample's features.

In [3]:

print(f"The total number of samples is **{**full\_df.count()**}**, with each sample corresponding to **{**len(full\_df.columns)**}** features.")

The total number of samples is 176224, with each sample corresponding to 67 features.

In order to identify each feature, as well as its type, full\_df.dtypes can be used. Alternatively, they are available as part of the Schema's information via the following:

In [4]:

full\_df.printSchema()

root

|-- iso\_code: string (nullable = true)

|-- continent: string (nullable = true)

|-- location: string (nullable = true)

|-- date: string (nullable = true)

|-- total\_cases: double (nullable = true)

|-- new\_cases: double (nullable = true)

|-- new\_cases\_smoothed: double (nullable = true)

|-- total\_deaths: double (nullable = true)

|-- new\_deaths: double (nullable = true)

|-- new\_deaths\_smoothed: double (nullable = true)

|-- total\_cases\_per\_million: double (nullable = true)

|-- new\_cases\_per\_million: double (nullable = true)

|-- new\_cases\_smoothed\_per\_million: double (nullable = true)

|-- total\_deaths\_per\_million: double (nullable = true)

|-- new\_deaths\_per\_million: double (nullable = true)

|-- new\_deaths\_smoothed\_per\_million: double (nullable = true)

|-- reproduction\_rate: double (nullable = true)

|-- icu\_patients: double (nullable = true)

|-- icu\_patients\_per\_million: double (nullable = true)

|-- hosp\_patients: double (nullable = true)

|-- hosp\_patients\_per\_million: double (nullable = true)

|-- weekly\_icu\_admissions: double (nullable = true)

|-- weekly\_icu\_admissions\_per\_million: double (nullable = true)

|-- weekly\_hosp\_admissions: double (nullable = true)

|-- weekly\_hosp\_admissions\_per\_million: double (nullable = true)

|-- total\_tests: double (nullable = true)

|-- new\_tests: double (nullable = true)

|-- total\_tests\_per\_thousand: double (nullable = true)

|-- new\_tests\_per\_thousand: double (nullable = true)

|-- new\_tests\_smoothed: double (nullable = true)

|-- new\_tests\_smoothed\_per\_thousand: double (nullable = true)

|-- positive\_rate: double (nullable = true)

|-- tests\_per\_case: double (nullable = true)

|-- tests\_units: string (nullable = true)

|-- total\_vaccinations: double (nullable = true)

|-- people\_vaccinated: double (nullable = true)

|-- people\_fully\_vaccinated: double (nullable = true)

|-- total\_boosters: double (nullable = true)

|-- new\_vaccinations: double (nullable = true)

|-- new\_vaccinations\_smoothed: double (nullable = true)

|-- total\_vaccinations\_per\_hundred: double (nullable = true)

|-- people\_vaccinated\_per\_hundred: double (nullable = true)

|-- people\_fully\_vaccinated\_per\_hundred: double (nullable = true)

|-- total\_boosters\_per\_hundred: double (nullable = true)

|-- new\_vaccinations\_smoothed\_per\_million: double (nullable = true)

|-- new\_people\_vaccinated\_smoothed: double (nullable = true)

|-- new\_people\_vaccinated\_smoothed\_per\_hundred: double (nullable = true)

|-- stringency\_index: double (nullable = true)

|-- population: double (nullable = true)

|-- population\_density: double (nullable = true)

|-- median\_age: double (nullable = true)

|-- aged\_65\_older: double (nullable = true)

|-- aged\_70\_older: double (nullable = true)

|-- gdp\_per\_capita: double (nullable = true)

|-- extreme\_poverty: double (nullable = true)

|-- cardiovasc\_death\_rate: double (nullable = true)

|-- diabetes\_prevalence: double (nullable = true)

|-- female\_smokers: double (nullable = true)

|-- male\_smokers: double (nullable = true)

|-- handwashing\_facilities: double (nullable = true)

|-- hospital\_beds\_per\_thousand: double (nullable = true)

|-- life\_expectancy: double (nullable = true)

|-- human\_development\_index: double (nullable = true)

|-- excess\_mortality\_cumulative\_absolute: double (nullable = true)

|-- excess\_mortality\_cumulative: double (nullable = true)

|-- excess\_mortality: double (nullable = true)

|-- excess\_mortality\_cumulative\_per\_million: double (nullable = true)

Most of the features are double types, i.e. numerical data. However, nominal features are also present:

* iso\_code: a string corresponding to each country's code.
* location: a string corresponding to each location's name.
* continent: a string corresponding to the continent where the location belongs.
* tests\_units: a string corresponding to the units used in each location in order to count the number of tests (more details below).

There's also the date feature, the type of which is string, however it will be properly transformed into a datetime object in what follows. The following command gives some examples for each of these features.

In [5]:

full\_df.select("iso\_code","location","continent","date","tests\_units").show(5)

+--------+-----------+---------+----------+-----------+

|iso\_code| location|continent| date|tests\_units|

+--------+-----------+---------+----------+-----------+

| AFG|Afghanistan| Asia|2020-02-24| null|

| AFG|Afghanistan| Asia|2020-02-25| null|

| AFG|Afghanistan| Asia|2020-02-26| null|

| AFG|Afghanistan| Asia|2020-02-27| null|

| AFG|Afghanistan| Asia|2020-02-28| null|

+--------+-----------+---------+----------+-----------+

only showing top 5 rows

Note that the tests\_units column in the above command contains several null values, indicating that many of them may appear in the present dataset. We may import functions from pyspark.sql, which allows us to perform aggregations (among other things) and count the exact number of null values for each feature. The results can be stored in a dictionary, which can then be sorted to present the results in an orderly fashion.

In [6]:

from pyspark.sql import functions as F

miss\_vals = full\_df.select([F.count(F.when(F.isnull(c), c)).alias(c) for c **in** full\_df.columns]).collect()[0].asDict()

miss\_vals = dict(sorted(miss\_vals.items(), reverse=True, key=lambda item: item[1]))

import pandas as pd

pd.DataFrame.from\_records([miss\_vals])

Out[6]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | weekly\_icu\_admissions | weekly\_icu\_admissions\_per\_million | excess\_mortality\_cumulative\_absolute | excess\_mortality\_cumulative | excess\_mortality | excess\_mortality\_cumulative\_per\_million | weekly\_hosp\_admissions | weekly\_hosp\_admissions\_per\_million | total\_boosters | total\_boosters\_per\_hundred | ... | new\_cases\_smoothed\_per\_million | new\_cases\_smoothed | new\_cases\_per\_million | total\_cases\_per\_million | new\_cases | total\_cases | population | iso\_code | location | date |
| 0 | 170498 | 170498 | 170201 | 170201 | 170201 | 170201 | 164813 | 164813 | 155973 | 155973 | ... | 9320 | 8539 | 7331 | 7106 | 6544 | 6319 | 1104 | 0 | 0 | 0 |

1 rows × 67 columns

Note that in the above Pandas is utilized only for the visualization of the dataframe, so it's not considered as "cheating".

Data Preprocessing:

In [7]:

full\_df = full\_df.withColumn('date',F.to\_date(F.unix\_timestamp(F.col('date'), 'yyyy-MM-dd').cast("timestamp")))

At this point, full\_df is filtered in order to keep only the data for the aforementioned two months.

In [8]:

dates = ("2021-01-01", "2021-02-28")

df = full\_df.where(F.col('date').between(\*dates))

For completeness, we perform for this filtered version the basic tasks that were performed for the full DataFrame:

In [9]:

print(f"The total number of samples is **{**df.count()**}**, with each sample corresponding to **{**len(df.columns)**}** features.")

miss\_vals = df.select([F.count(F.when(F.isnull(c), c)).alias(c) for c **in** df.columns]).collect()[0].asDict()

miss\_vals = dict(sorted(miss\_vals.items(), reverse=True, key=lambda item: item[1]))

pd.DataFrame.from\_records([miss\_vals])

The total number of samples is 13515, with each sample corresponding to 67 features.

Out[9]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | weekly\_icu\_admissions | weekly\_icu\_admissions\_per\_million | excess\_mortality\_cumulative\_absolute | excess\_mortality\_cumulative | excess\_mortality | excess\_mortality\_cumulative\_per\_million | total\_boosters | total\_boosters\_per\_hundred | weekly\_hosp\_admissions | weekly\_hosp\_admissions\_per\_million | ... | new\_cases\_smoothed\_per\_million | new\_cases\_per\_million | total\_cases\_per\_million | new\_cases\_smoothed | new\_cases | total\_cases | population | iso\_code | location | date |
| 0 | 13029 | 13029 | 13010 | 13010 | 13010 | 13010 | 12765 | 12765 | 12581 | 12581 | ... | 412 | 382 | 378 | 353 | 323 | 319 | 105 | 0 | 0 | 0 |

1 rows × 67 columns

2.1. Handling Missing Values

Even in this filtered version, there's a sizeable number of null values present. Before investigating how to deal with them, it's important that we understand the reason why they're missing. As far as the continent feature is concerned, the following command sheds light into the reason why it contains null values.

In [10]:

df.sort("continent").select("iso\_code","continent","location").show(5)

[Stage 15:>(0 + 4) / 4]

+--------+---------+-------------+

|iso\_code|continent| location|

+--------+---------+-------------+

|OWID\_SAM| null|South America|

|OWID\_SAM| null|South America|

|OWID\_SAM| null|South America|

|OWID\_SAM| null|South America|

|OWID\_SAM| null|South America|

+--------+---------+-------------+

only showing top 5 rows

Clearly, OWiD have performed a series of aggregations based on criteria such as income, or general aggregations (for example on the continent level). Since they may prove to be useful later on, there is no reason to discard them. The null values can simply be set equal to the 'OWID' value, in order to be able to invoke them later on if we need to.

In [11]:

df = df.fillna({'continent':'OWID'})

Another column which corresponds to a nominal feature with missing values is tests\_units. The distinct values that this feature assumes are:

In [12]:

df.select("tests\_units").distinct().show()

[Stage 16:>(0 + 4) / 4]

+---------------+

| tests\_units|

+---------------+

| null|

| people tested|

| samples tested|

|tests performed|

| units unclear|

+---------------+

In other words, tests\_units is simply a variable that indicates how each country/location reports on the performed tests. For example, in the case of people tested, the reported number of total tests is expected to be lower compared to the same report in the case of tests performed, since one person can be tested more than once during the same day. This implies that the missing values are due to some countries/locations not providing the relevant information on how they count the total number of daily tests. Of course, this is not a reason to discard the relevant data, therefore the missing values will be replaced by the string 'no info'.

In [13]:

df = df.fillna({'tests\_units':'no info'})

Moving on to the quantitative features, most missing values are due to the fact that the relevant data were either not available during the studied time period for some locations, or were simply equal to zero. For example, there are 10272 missing values in the new\_vaccinations column, which are either due to the fact that vaccines were not available in some locations, or due to the fact that these locations reported no vaccinations for specific dates. The best approach in this case is replacing all these values with 0. In the few cases where the missing values are not due to any of these two reasons, but due to wrong reports, bugs, or other reasons, we expect to find it out during their analysis and especially their visualization. In this case, we will be able to re-handle them or discard them completely.

In [14]:

df = df.fillna(0)

The following confirms that there are no missing values left in the dataset.

In [15]:

miss\_vals = df.select([F.count(F.when(F.isnull(c), c)).alias(c) for c **in** df.columns]).collect()[0].asDict()

if any(list(miss\_vals.values())) != 0:

print("There are still missing values in the DataFrame.")

else:

print("All missing values have been taken care of.")

[Stage 19:>(0 + 4) / 4]

All missing values have been taken care of.

2.2. Outlier Detection

Having discussed the case of missing values, perhaps it's a good idea to also discuss the case of outliers. Typically, the identification of outliers requires further analysis, such as visualizations, since it is not a trivial matter (in fact, more often than not it's a case of a supervised learning problem on its own). Furthermore, there are several types of outliers, such as global outliers or context-based outliers (i.e. points that are outliers only given a specific condition or context), which means that dealing with outliers in a universal manner is ill-advised. Nonetheless, if one chooses to do so, a systematic way to deal with outliers is based on [interquartile range methods](https://en.wikipedia.org/wiki/Interquartile_range). The interquartile range, R, is defined as

R=Q3−Q1=3−1

where Qi is the i-th quartile. Every point for which the studied feature has a value higher than Q3+αR3+ or lower than Q1−αR1− is classified as an outlier for this specific feature, where α is a scalar that defines a "decision boundary" in units of R. This is essentially how [Box plots](https://en.wikipedia.org/wiki/Box_plot) are constructed, where R corresponds to the Box's height and αR is equal to the whiskers' length. One very common choice for α is α=1.5=1.5.

Based on these, one can define a function that identifies all outliers with respect to specific features.

In [16]:

def OutlierDetector(dataframe, features, alpha=1.5):

*"""*

*Args:*

*dataframe (pyspark.sql.dataframe.DataFrame):*

*the DataFrame hosting the data*

*features (string or List):*

*List of features (columns) for which we wish to identify outliers.*

*If set equal to 'all', outliers are identified with respect to all features.*

*alpha (double):*

*The parameter that defines the decision boundary (see markdown above)*

*"""*

feat\_types = dict(dataframe.dtypes)

if features == 'all':

features = dataframe.columns

outliers\_cols = []

for feat **in** features:

*# We only care for quantitative features*

if feat\_types[feat] == 'double':

Q1, Q3 = dataframe.approxQuantile(feat, [0.25, 0.75], 0)

R = Q3 - Q1

lower\_bound = Q1 - (R \* alpha)

upper\_bound = Q3 + (R \* alpha)

*# In this way we construct a query, which can be matched to a DataFrame column, thus returning a new*

*# column where every point that corresponds to an Outlier has a boolean value set to True*

outliers\_cols.append(F.when(~F.col(feat).between(lower\_bound, upper\_bound), True).alias(feat + '\_outlier'))

*# Sample points that do not correspond to outliers correspond to a False value for the new column*

outlier\_df = dataframe.select(\*outliers\_cols)

outlier\_df = outlier\_df.fillna(False)

return outlier\_df

For example, we can check if any of 5 random DataFrame rows correspond to outliers with respect to the new\_cases feature:

In [17]:

out\_df = OutlierDetector(dataframe=df, features=['new\_cases'], alpha=1.5)

out\_df.show(5)

+-----------------+

|new\_cases\_outlier|

+-----------------+

| false|

| false|

| false|

| false|

| false|

+-----------------+

only showing top 5 rows

2.3. Duplicate Entries

Before proceeding to the exploratory data analysis, the final step of the preprocessing phase is to locate possible duplicate entries and discard the duplicates. When speaking of duplicates we do not actually refer to a whole row, but rather the combined entries of the date **and** location columns. A duplicate entry on both of these features would imply that the location has provided more than one daily report on a given date. The following command shows that no duplicates exist in the filtered DataFrame, however, even if they did, they could be removed using df = df.dropDuplicates(['location','date']).

In [18]:

if df.count() != df.select(['location','date']).distinct().count():

print("There are duplicate entries present in the DataFrame.")

else:

print("Either there are no duplicate entries present in the DataFrame, or all of them have already been removed).")

[Stage 27:==============>(1 + 3) / 4]

Either there are no duplicate entries present in the DataFrame, or all of them have already been removed).

Exploratory Data Analysis:

In [19]:

import matplotlib

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from matplotlib.colors import ListedColormap, LinearSegmentedColormap, TwoSlopeNorm

from mpl\_toolkits.axes\_grid1 import make\_axes\_locatable

def CustomCmap(from\_rgb,to\_rgb):

*# from color r,g,b*

r1,g1,b1 = from\_rgb

*# to color r,g,b*

r2,g2,b2 = to\_rgb

cdict = {'red': ((0, r1, r1),

(1, r2, r2)),

'green': ((0, g1, g1),

(1, g2, g2)),

'blue': ((0, b1, b1),

(1, b2, b2))}

cmap = LinearSegmentedColormap('custom\_cmap', cdict)

return cmap

mycmap = CustomCmap([1.0, 1.0, 1.0], [72/255, 99/255, 147/255])

mycmap\_r = CustomCmap([72/255, 99/255, 147/255], [1.0, 1.0, 1.0])

mycol = (72/255, 99/255, 147/255)

mycomplcol = (129/255, 143/255, 163/255)

othercol1 = (135/255, 121/255, 215/255)

othercol2 = (57/255, 119/255, 171/255)

othercol3 = (68/255, 81/255, 91/255)

othercol4 = (73/255, 149/255, 139/255)

**Evolution of top countries with respect to mortality**

Herein, the mortality rate is calculated as the total number of deaths divided by each location's population (another common definition is the total number of deaths by Covid divided by the total number of Covid cases). For this purpose, a column named mortality is constructed. Using this column, we identify the top 10 countries in terms of mortality rates, for every day of the studied time interval.

In [20]:

dates\_frame = df.select("date").distinct().orderBy('date').collect()

dates\_list = [str(dates\_frame[x][0]) for x **in** range(len(dates\_frame))]

In [21]:

df\_for\_mort = df.filter(F.col('population') != 0.0).withColumn("mortality", F.col("total\_deaths")/F.col("population"))

for i, this\_day **in** enumerate(dates\_list):

this\_day\_top\_10 = df\_for\_mort.filter(F.col('date') == this\_day).orderBy("mortality", ascending=False).select(["location","mortality"]).take(10)

if i == 0:

ct\_list = [(this\_day\_top\_10[x][0],this\_day\_top\_10[x][1]) for x **in** range(10)]

print("During "+this\_day+", the top 10 countries with the highest mortality rate were:")

for country, instance **in** ct\_list:

print(f"▶ **{**country**}**, with mortality rate **{**100\*instance**:**.2f**}**%.")

new\_set = set(ct\_list[x][0] for x **in** range(10))

elif i == len(dates\_list)-1:

ct\_list = [(this\_day\_top\_10[x][0],this\_day\_top\_10[x][1]) for x **in** range(10)]

print("During "+this\_day+", the top 10 countries with the highest mortality rate were:")

for country, instance **in** ct\_list:

print(f"▶ **{**country**}**, with mortality rate **{**100\*instance**:**.2f**}**%.")

else:

new\_set = set(this\_day\_top\_10[x][0] for x **in** range(10))

if new\_set != old\_set:

left\_out = old\_set-new\_set

new\_additions = new\_set-old\_set

print("This was the top ten until "+this\_day+", when "+", ".join(str(s) for s **in** new\_additions)+" joined the list, replacing "+", ".join(str(s) for s **in** left\_out)+".")

new\_set, old\_set = set(), new\_set

During 2021-01-01, the top 10 countries with the highest mortality rate were:

▶ Peru, with mortality rate 0.28%.

▶ San Marino, with mortality rate 0.17%.

▶ Belgium, with mortality rate 0.17%.

▶ Slovenia, with mortality rate 0.13%.

▶ Bosnia and Herzegovina, with mortality rate 0.13%.

▶ Italy, with mortality rate 0.12%.

▶ North Macedonia, with mortality rate 0.12%.

▶ Liechtenstein, with mortality rate 0.12%.

▶ Bulgaria, with mortality rate 0.11%.

▶ Czechia, with mortality rate 0.11%.

This was the top ten until 2021-01-08, when United Kingdom joined the list, replacing Bulgaria.

This was the top ten until 2021-01-18, when Gibraltar joined the list, replacing North Macedonia.

This was the top ten until 2021-02-03, when North Macedonia joined the list, replacing Liechtenstein.

This was the top ten until 2021-02-05, when United States joined the list, replacing North Macedonia.

This was the top ten until 2021-02-08, when North Macedonia joined the list, replacing United States.

This was the top ten until 2021-02-09, when Portugal joined the list, replacing North Macedonia.

This was the top ten until 2021-02-23, when Montenegro joined the list, replacing Bosnia and Herzegovina.

During 2021-02-28, the top 10 countries with the highest mortality rate were:

▶ Peru, with mortality rate 0.37%.

▶ Gibraltar, with mortality rate 0.28%.

▶ San Marino, with mortality rate 0.22%.

▶ Belgium, with mortality rate 0.19%.

▶ Czechia, with mortality rate 0.19%.

▶ Slovenia, with mortality rate 0.18%.

▶ United Kingdom, with mortality rate 0.18%.

▶ Italy, with mortality rate 0.16%.

▶ Portugal, with mortality rate 0.16%.

▶ Montenegro, with mortality rate 0.16%.

**Evolution of top countries with respect to total cases per million**

The same procedure can be performed for the number of total cases per million. We choose to normalize the total number of cases in this way in order to be able to compare locations with different populations.

In [22]:

for i, this\_day **in** enumerate(dates\_list):

this\_day\_top\_10 = df.filter(F.col('date') == this\_day).orderBy("total\_cases\_per\_million", ascending=False).select(["location","total\_cases\_per\_million"]).take(10)

if i == 0:

ct\_list = [(this\_day\_top\_10[x][0],this\_day\_top\_10[x][1]) for x **in** range(10)]

print("During "+this\_day+", the top 10 countries with the highest number of total cases per million were:")

for country, instance **in** ct\_list:

print(f"▶ **{**country**}**, with **{**instance**}** total cases per million.")

new\_set = set(ct\_list[x][0] for x **in** range(10))

elif i == len(dates\_list)-1:

ct\_list = [(this\_day\_top\_10[x][0],this\_day\_top\_10[x][1]) for x **in** range(10)]

print("During "+this\_day+", the top 10 countries with the highest number of total cases per million were:")

for country, instance **in** ct\_list:

print(f"▶ **{**country**}**, with **{**instance**}** total cases per million.")

else:

new\_set = set(this\_day\_top\_10[x][0] for x **in** range(10))

if new\_set != old\_set:

left\_out = old\_set-new\_set

new\_additions = new\_set-old\_set

print("This was the top ten until "+this\_day+", when "+", ".join(str(s) for s **in** new\_additions)+" joined the list, replacing "+", ".join(str(s) for s **in** left\_out)+".")

new\_set, old\_set = set(), new\_set

During 2021-01-01, the top 10 countries with the highest number of total cases per million were:

▶ Andorra, with 104933.164 total cases per million.

▶ Montenegro, with 77364.736 total cases per million.

▶ Luxembourg, with 73115.905 total cases per million.

▶ San Marino, with 70097.03 total cases per million.

▶ Czechia, with 68256.644 total cases per million.

▶ Gibraltar, with 65655.516 total cases per million.

▶ United States, with 61189.671 total cases per million.

▶ French Polynesia, with 59907.834 total cases per million.

▶ Slovenia, with 59627.954 total cases per million.

▶ Liechtenstein, with 58686.673 total cases per million.

This was the top ten until 2021-01-07, when Panama joined the list, replacing French Polynesia.

This was the top ten until 2021-01-13, when Slovakia joined the list, replacing Liechtenstein.

This was the top ten until 2021-02-05, when Portugal joined the list, replacing Panama.

This was the top ten until 2021-02-10, when Israel joined the list, replacing Portugal.

During 2021-02-28, the top 10 countries with the highest number of total cases per million were:

▶ Andorra, with 140471.081 total cases per million.

▶ Gibraltar, with 125819.952 total cases per million.

▶ Montenegro, with 120744.971 total cases per million.

▶ Czechia, with 115201.072 total cases per million.

▶ San Marino, with 109261.982 total cases per million.

▶ Slovakia, with 108027.497 total cases per million.

▶ Slovenia, with 91441.236 total cases per million.

▶ Luxembourg, with 86812.83 total cases per million.

▶ United States, with 86351.972 total cases per million.

▶ Israel, with 83500.915 total cases per million.

The corresponding .gif image can be seen below.

**Hospitalized Patients and ICU Admissions**

Moving on, we study the hosp\_patients and icu\_patients features by visualizing the corresponding timeseries for the total number of hospitalized and ICU patients on a global scale.

In [23]:

dt\_ord = df.orderBy("date", ascending=True).groupBy("date")

hosps = dt\_ord.agg(F.sum("hosp\_patients")).collect()

hosps = [hosps[i][1] for i **in** range(len(hosps))]

icus = dt\_ord.agg(F.sum("icu\_patients")).collect()

icus = [icus[i][1] for i **in** range(len(icus))]

sns.set(style = "darkgrid")

alt\_dts\_list = [dt.replace('2021-', '') for dt **in** dates\_list]

tick\_marks = np.arange(len(alt\_dts\_list))

fig, [ax1,ax2] = plt.subplots(1, 2, figsize=(14,5))

for pat, col, style, ax, where **in** zip([hosps,icus], [mycol, mycomplcol],

['solid', 'dashed'], [ax1,ax2], ['Normal Beds','ICUs']):

ax.plot(alt\_dts\_list, pat, linestyle=style, color=col)

ax.set\_xlabel("Date")

ax.set\_ylabel("Number of Patients")

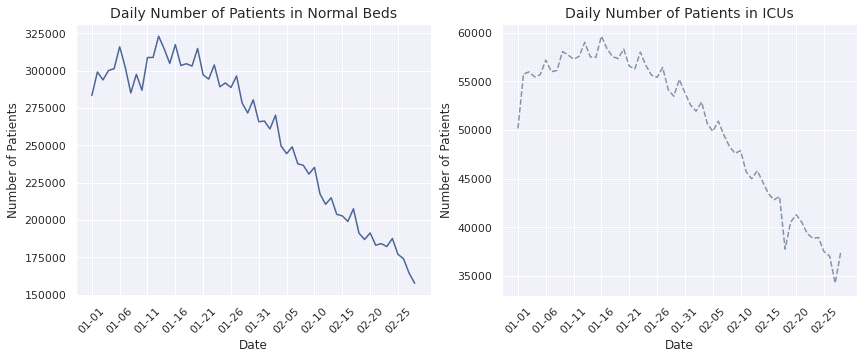
ax.set\_title(f"Daily Number of Patients in **{**where**}**", fontsize=14)

ax.set\_xticks(tick\_marks[::5])

ax.set\_xticklabels(alt\_dts\_list[::5], rotation=45)

plt.show()

matplotlib.rc\_file\_defaults()



It becomes evident that the overall trend is downwards for both hospital and ICU admissions, since both numbers have declined to almost half their initial value by the end of the two-month period under study. It is worth noting that an upwards trend seems to appear near the end of February in the case of ICU patients. Of course, without further information we can't know if it is the beginning of a monotonically increasing trend, or simply a momentary increase, as the one identified between February 15-20. Finally, notice that both diagrams have a similar behavior, which hints at a correlation between the number of hospital patients and the number of ICU patients (which is probably expected). An important difference is that the absolute value of the number of hospital patients is considerably higher compared to the number of ICU admissions, which is reasonable, since the number of milder cases is higher compared to the number of more severe ones.

**Geographic Heatmap of Total Cases**

An interesting visualization is the geographic heatmap, which is a 2D representation of countries world-wide which are colored depending on their intensity as far as a specific feature is concerned. Below, we construct the geographic heatmap for the number of total cases on a global scale. A heatmap image is extracted for each day and afterwards all images are merged into a .gif file. The heatmap is constructed using the geopandas library, as seen below. Note that to do this, we must first download a shapefile (.shp) which is the foundation for the construction of the heatmap and can be found [here](https://www.naturalearthdata.com/downloads/10m-cultural-vectors/).

In [24]:

import requests, zipfile

from io import BytesIO

zip\_file\_url = "<https://srigas.me/kaggle/owid-nb-data.zip>"

request = requests.get(zip\_file\_url)

zipDocument = zipfile.ZipFile(BytesIO(request.content))

zipDocument.extractall()

In [25]:

import geopandas as gpd

shapefile = 'countries.shp'

geo\_df = gpd.read\_file(shapefile)[['ADMIN','ADM0\_A3','geometry']]

geo\_df.columns = ['location', 'iso\_code', 'geometry']

geo\_df = geo\_df.drop(geo\_df.loc[geo\_df['location'] == 'Antarctica'].index) *# exclude Antarctica*

In [26]:

print('Initializing the construction of heatmaps for every day.')

ct = 0

for this\_day **in** dates\_list:

*# The conversion of the required columns into a Pandas df is necessary to perform the mapping*

day\_df = df.filter(F.col('date') == this\_day).select(["iso\_code","total\_cases"]).toPandas()

merged\_df = pd.merge(left=geo\_df, right=day\_df, how='left', left\_on='iso\_code', right\_on='iso\_code')

title = f'Total COVID-19 Cases as of **{**this\_day**}**'

col = 'total\_cases'

vmin, vmax = merged\_df[col].min(), merged\_df[col].max()

cmap = mycmap

divnorm = TwoSlopeNorm(vcenter=0.08\*20365726)

*# Create figure and axes for Matplotlib*

fig, ax = plt.subplots(1, figsize=(20, 8))

*# Remove the axis*

ax.axis('off')

merged\_df.plot(column=col, ax=ax, edgecolor='1.0', linewidth=1, norm=divnorm, cmap=cmap)

*# Add a title*

ax.set\_title(title, fontdict={'fontsize': '25', 'fontweight': '3'})

*# Create colorbar as a legend*

sm = plt.cm.ScalarMappable(norm=plt.Normalize(vmin=vmin, vmax=vmax), cmap=cmap)

*# Empty array for the data range*

sm.\_A = []

*# Add the colorbar to the figure*

cbaxes = fig.add\_axes([0.15, 0.25, 0.01, 0.4])

cbar = fig.colorbar(sm, cax=cbaxes)

plt.savefig(f'world\_map\_**{**this\_day**}**.png', bbox\_inches='tight')

plt.close(fig)

ct += 1

print(f'Process complete. **{**ct**}** heatmap(s) were extracted, ready to be converted into a .gif file.')

Initializing the construction of heatmaps for every day.

Process complete. 59 heatmap(s) were extracted, ready to be converted into a .gif file.

The final .gif can be seen below.

**Geographic Correlation of Excess Mortality**

Based on the previous visualization it appears that some neighbouring countries are correlated with respect to the total number of cases (for example France and Germany). A reasonable hypothesis is that the same may be true for other features as well, such as the excess mortality.

The excess mortality is a feature for which the reports are weekly and not daily. It is equal to the total number of deaths for a specific week minus the mean number of deaths, based on reports from previous years. While it is not a feature directly connected with Covid, it's expected that during a global pandemic the excess mortality can be mainly attributed to this pandemic.

In order to investigate the correlation between neighbouring countries, we must first develop a list of dates for which reports on excess mortality are available (for all other dates, the entries are equal to zero due to our preprocessing).

In [27]:

exc\_dates\_list = df.filter(F.col('excess\_mortality') != 0.0).select(['date']).distinct().orderBy('date').collect()

exc\_dates\_list = [str(exc\_dates\_list[i][0]) for i **in** range(len(exc\_dates\_list))]

For brevity, we shall focus our study only on European countries. First, we construct a geographic heatmap of Europe with respect to excess mortality for each date calculated in the previous cell and merge the results into a .gif file, as done previously.

In [28]:

print('Initializing the construction of heatmaps for every day.')

ct = 0

for this\_day **in** exc\_dates\_list:

europe\_df = df.filter(F.col('date') == this\_day).filter(F.col('continent') == 'Europe').filter(F.col('excess\_mortality') != 0.0).select(["iso\_code","excess\_mortality"])

geo\_eu = pd.merge(left=geo\_df, right=europe\_df.toPandas(), how='inner', on='iso\_code')

fig, ax = plt.subplots(1,1)

col = 'excess\_mortality'

cmap = mycmap

vmin, vmax = geo\_eu[col].min(), geo\_eu[col].max()

sm = plt.cm.ScalarMappable(norm=plt.Normalize(vmin=vmin, vmax=vmax), cmap=cmap)

ax.axis('off')

ax.axis([-13, 44, 33, 72])

geo\_eu.plot(column=col, ax=ax, edgecolor='1.0', linewidth=1, norm=None, cmap=cmap)

ax.set\_title(f'Excess Mortality in Europe as of **{**this\_day**}**', fontdict={'fontsize': '14', 'fontweight': '3'})

divider = make\_axes\_locatable(ax)

cax = divider.append\_axes("right", size="5%", pad=.2)

fig.add\_axes(cax)

fig.colorbar(sm, cax=cax)

plt.savefig(f'europe\_**{**this\_day**}**.png', bbox\_inches='tight')

plt.close(fig)

ct += 1

print(f'Process complete. **{**ct**}** heatmap(s) were extracted, ready to be converted into a .gif file.')

Initializing the construction of heatmaps for every day.

Process complete. 9 heatmap(s) were extracted, ready to be converted into a .gif file.

Based on this visualization, it's safe to assume that there indeed are neighbouring countries for which the excess mortality values appear to be significantly correlated. Germany and Switzerland are an example of one such pair of countries, as they appear to have highs and lows with respect to excess mortality at the same time.

In order to produce these results with more mathematical rigor, we need to construct a new PySpark DataFrame including all the reports on excess mortality for each European country that has provided reports on **all** of the previously calculated dates. Countries with even 1 missing value will not be taken into consideration, in order to be able to draw conclusions that are as safe as possible, since the volume of the available data is very small with regards to this feature. Then, using this newly created DataFrame, a Pearson correlation matrix can be constructed, thus revealing not only pairs of correlated countries that share the same geographical borders, but also the exact value of this correlation.

In [29]:

european\_df = df.filter(F.col('continent') == 'Europe').filter(F.col('excess\_mortality') != 0.0)

european\_cts = european\_df.select(['location']).distinct().collect()

european\_cts = [european\_cts[i][0] for i **in** range(len(european\_cts)) if european\_df.filter(F.col('location') == european\_cts[i][0]).count() == len(exc\_dates\_list)]

print(f'**{**len(european\_cts)**}** European countries are chosen for this analysis.')

31 European countries are chosen for this analysis.

In [30]:

from pyspark.sql.functions import monotonically\_increasing\_id, row\_number

from pyspark.sql.window import Window

eu\_cts\_df = european\_df.filter(F.col('location') == european\_cts[0]).select(['excess\_mortality']).withColumnRenamed("excess\_mortality", european\_cts[0])

*# required for the proper join of the following DataFrames*

eu\_cts\_df = eu\_cts\_df.withColumn('row\_index', row\_number().over(Window.partitionBy(F.lit(0)).orderBy(monotonically\_increasing\_id())))

for country **in** european\_cts[1:]:

new\_ct\_df = european\_df.filter(F.col('location') == country).select(['excess\_mortality']).withColumnRenamed("excess\_mortality", country)

new\_ct\_df = new\_ct\_df.withColumn('row\_index', row\_number().over(Window.partitionBy(F.lit(0)).orderBy(monotonically\_increasing\_id())))

eu\_cts\_df = eu\_cts\_df.join(new\_ct\_df, on=["row\_index"])

eu\_cts\_df = eu\_cts\_df.drop("row\_index")

In [31]:

from pyspark.ml.stat import Correlation

from pyspark.ml.feature import VectorAssembler

vector\_col = "corr\_features"

assembler = VectorAssembler(inputCols=eu\_cts\_df.columns, outputCol=vector\_col)

df\_vector = assembler.transform(eu\_cts\_df).select(vector\_col)

matrix = Correlation.corr(df\_vector, vector\_col, 'pearson')

cor\_np = matrix.collect()[0][matrix.columns[0]].toArray()

The calculated correlation matrix can be seen in the following heatmap, where only values of Pearson correlation that are higher than 0.8 are depicted (since we are looking for neighbouring countries with high correlation). This is why the lower limit of the colorbar is set to 0.8.

In [32]:

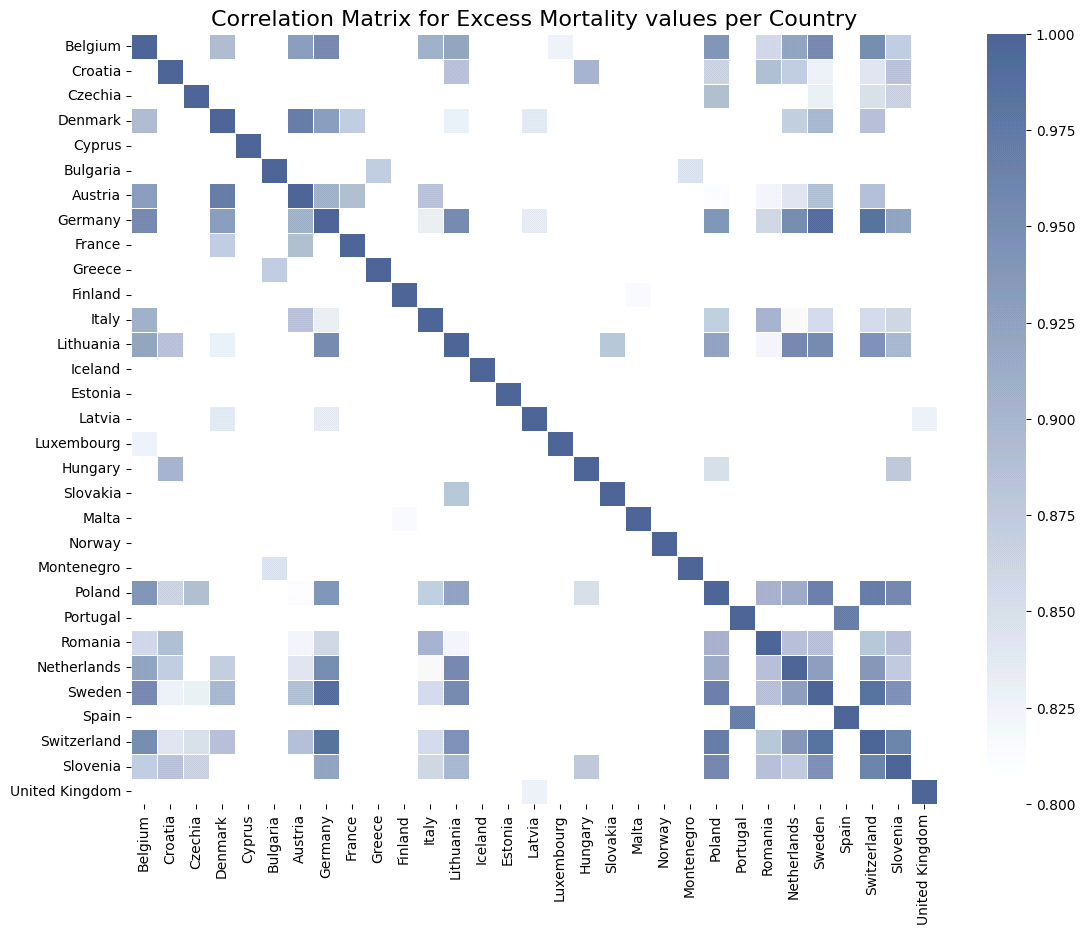
fig, ax = plt.subplots(figsize=(13,10))

sns.heatmap(cor\_np, linewidths=.5, ax=ax, vmin=0.8, vmax=1, cmap=mycmap,

xticklabels=european\_cts, yticklabels=european\_cts)

ax.set\_title('Correlation Matrix for Excess Mortality values per Country', fontsize=16)

plt.show()



As noted before, Switzerland and Germany indeed correspond to a pair of highly correlated neighbouring countries with respect to excess mortality. In fact, it appears that in most cases **only** neighbouring countries (for example Belgium and Germany, or Luxemburg and Netherlands) and second neighbours thereof show high values of correlation, with the value of correlation declining significantly as the neighbour index (i.e. how many countries apart two countries are) increases beyond 2. Some additional examples of correlated pairs can be seen in the following table, along with the corresponding Pearson Correlation.

|  |  |
| --- | --- |
|  | Pearson Correlation for excess mortality (%) |
| **Germany & Switzerland** | 98.29 |
| **Germany & Belgium** | 95.44 |
| **Lithuania & Poland** | 92.50 |
| **Netherlands & Belgium** | 92.30 |
| **Czechia & Poland** | 89.08 |
| **Greece & Bulgaria** | 86.87 |
| **Italy & Slovenia** | 86.23 |

For a more extensive list of European countries with high correlation that is not limited only to neighbouring countries, one can run the following snippet of code:

for i in range(len(european\_cts)):

for j in range(i+1,len(european\_cts)):

corr\_val = cor\_np[i][j]

if corr\_val > 0.8:

print(f'{european\_cts[i]} and {european\_cts[j]} show a correlation of {100\*corr\_val:.2f}.')

**Reproduction Rate on the Continent Level**

Moving on, we continue the study of the pandemic's features on a geographic viewpoint by grouping the countries together into continents. For this purpose, the DataFrame is split into continent-level DataFrames, in order to be able to draw the geographic heatmaps separately. The studied feature is now the daily reproduction rate, corresponding to the heatmap's intensity, with a common scale in order to be able to compare different continents. As usual, a .gif image is constructed using separate heatmap images for each day in the January-February interval.

In [33]:

daily\_means = {'AS': [], 'EU' : [], 'NAM' : [], 'SAM' : [], 'OC' : [], 'AF' : []}

print('Initializing the construction of heatmaps for every day.')

ct = 0

for this\_day **in** dates\_list:

asia\_df = df.filter(F.col('date') == this\_day).filter(F.col('continent') == 'Asia').filter(F.col('reproduction\_rate') != 0.0).select(["iso\_code","reproduction\_rate"])

europe\_df = df.filter(F.col('date') == this\_day).filter(F.col('continent') == 'Europe').filter(F.col('reproduction\_rate') != 0.0).select(["iso\_code","reproduction\_rate"])

namerica\_df = df.filter(F.col('date') == this\_day).filter((F.col('continent') == 'North America')).filter(F.col('reproduction\_rate') != 0.0).select(["iso\_code","reproduction\_rate"])

samerica\_df = df.filter(F.col('date') == this\_day).filter((F.col('continent') == 'South America')).filter(F.col('reproduction\_rate') != 0.0).select(["iso\_code","reproduction\_rate"])

oceania\_df = df.filter(F.col('date') == this\_day).filter(F.col('continent') == 'Oceania').filter(F.col('reproduction\_rate') != 0.0).select(["iso\_code","reproduction\_rate"])

africa\_df = df.filter(F.col('date') == this\_day).filter(F.col('continent') == 'Africa').filter(F.col('reproduction\_rate') != 0.0).select(["iso\_code","reproduction\_rate"])

daily\_means['AS'].append(asia\_df.select(F.mean(F.col('reproduction\_rate'))).collect()[0][0])

daily\_means['EU'].append(europe\_df.select(F.mean(F.col('reproduction\_rate'))).collect()[0][0])

daily\_means['NAM'].append(namerica\_df.select(F.mean(F.col('reproduction\_rate'))).collect()[0][0])

daily\_means['SAM'].append(samerica\_df.select(F.mean(F.col('reproduction\_rate'))).collect()[0][0])

daily\_means['OC'].append(oceania\_df.select(F.mean(F.col('reproduction\_rate'))).collect()[0][0])

daily\_means['AF'].append(africa\_df.select(F.mean(F.col('reproduction\_rate'))).collect()[0][0])

geo\_as = pd.merge(left=geo\_df, right=asia\_df.toPandas(), how='inner', on='iso\_code')

geo\_eu = pd.merge(left=geo\_df, right=europe\_df.toPandas(), how='inner', on='iso\_code')

geo\_sam = pd.merge(left=geo\_df, right=samerica\_df.toPandas(), how='inner', on='iso\_code')

geo\_nam = pd.merge(left=geo\_df, right=namerica\_df.toPandas(), how='inner', on='iso\_code')

geo\_oc = pd.merge(left=geo\_df, right=oceania\_df.toPandas(), how='inner', on='iso\_code')

geo\_af = pd.merge(left=geo\_df, right=africa\_df.toPandas(), how='inner', on='iso\_code')

fig, axes = plt.subplots(2,3, figsize=(18,14))

col = 'reproduction\_rate'

cmap = mycmap

vmin = min(geo\_as[col].min(),geo\_eu[col].min(),geo\_sam[col].min(),geo\_nam[col].min(),geo\_oc[col].min(),geo\_af[col].min())

vmax = max(geo\_as[col].max(),geo\_eu[col].max(),geo\_sam[col].max(),geo\_nam[col].max(),geo\_oc[col].max(),geo\_af[col].max())

sm = plt.cm.ScalarMappable(norm=plt.Normalize(vmin=vmin, vmax=vmax), cmap=cmap)

for ax, data, cont, dims **in** zip(axes.flat,

[geo\_eu, geo\_nam, geo\_af, geo\_as, geo\_sam, geo\_oc],

['Europe','North America','Africa','Asia','South America','Oceania'],

[[-13, 44, 33, 72],[-170, -50, 5, 85],[-20, 55, -38, 40],[25, 145, -10, 60],[-85, -32, -58, 15],[110,160,-45,0]]):

title = f'**{**cont**}**'

ax.axis('off')

ax.axis(dims)

data.plot(column=col, ax=ax, edgecolor='1.0', linewidth=1, norm=None, cmap=cmap)

ax.set\_title(title, fontdict={'fontsize': '18', 'fontweight': '3'})

clb = fig.colorbar(sm, ax=axes.flat, location='bottom', fraction=0.056)

clb.ax.set\_title(f'COVID-19 Reproduction Rate as of **{**this\_day**}**', fontsize=22)

plt.savefig(f'cont\_maps\_**{**this\_day**}**.png', bbox\_inches='tight')

plt.close(fig)

ct += 1

print(f'Process complete. **{**ct**}** heatmap(s) were extracted, ready to be converted into a .gif file.')

Initializing the construction of heatmaps for every day.

Process complete. 59 heatmap(s) were extracted, ready to be converted into a .gif file.

In [34]:

sns.set(style = "darkgrid")

fig, ax = plt.subplots(1, 1, figsize=(10,5))

for key, col, lab **in** zip(daily\_means,

[othercol1, othercol2, othercol3, mycol, mycomplcol, othercol4],

['Asia', 'Europe', 'Africa', 'N. America', 'S. America', 'Oceania']):

ax.plot(alt\_dts\_list, daily\_means[key], color=col, label = lab)

ax.set\_xlabel("Date")

ax.set\_ylabel("Reproduction Rate")

ax.set\_title("Daily Mean Reproduction Rate per Continent", fontsize=14)

tick\_marks = np.arange(len(alt\_dts\_list))

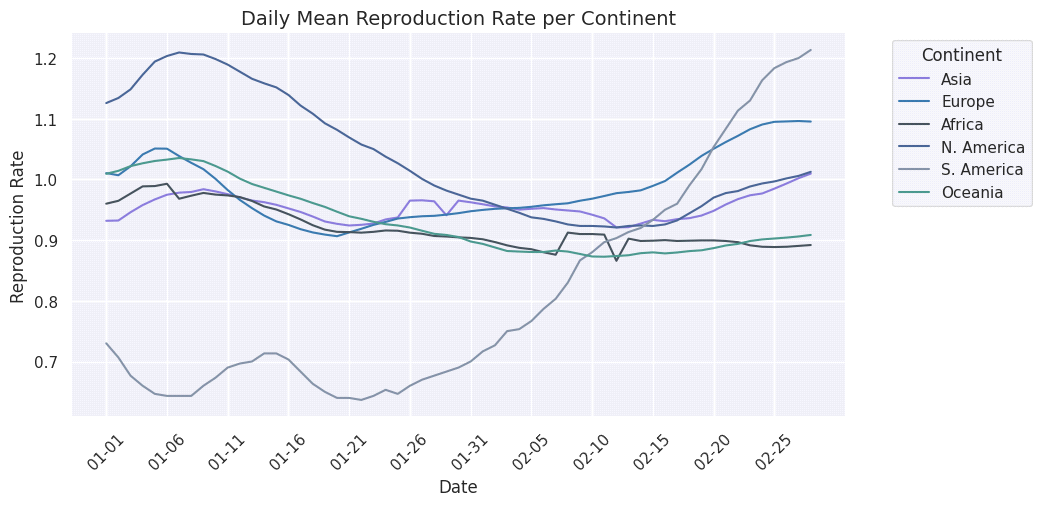
ax.set\_xticks(tick\_marks[::5])

ax.set\_xticklabels(alt\_dts\_list[::5], rotation=45)

plt.legend(bbox\_to\_anchor=(1.05, 1), loc=2, title="Continent")

plt.show()

matplotlib.rc\_file\_defaults()



The trends and fluctuations that were previously observed can be seen in this graph as well. For example, the overall trend towards higher values of reproduction rate is evident in all continents with the exception of North America. Nonetheless, an additional piece of information provided by this new graph is that during the last days of February the mean reproduction rate for the countries in North America has a tendency to increase. The same can be said for the countries of Asia and South America as well. Especially for the countries in South America, the overall increase of the reproduction rate was approximately 60%, indicating that the overall increase may be even higher by the end of March, based on the aforementioned tendency. On the other hand, when it comes to Oceania, Africa and Europe, there is a tendency towards stabilization of the reproduction rate to a constant value.

Closing this part of the analysis, it's worth noting that the intense fluctuations that were observed for Oceania in the .gif above do not seem to appear in this graph, where the corresponding timeseries is close to being constant, if compared to the other ones. However, upon a closer inspection, it becomes clear that there is no inconsistency: the mean reproduction rate is calculated as an unweighted mean value by diving with the total number of countries, instead of a weighted mean with respect to each country's population. This means that the contribution of Australia is considered equal to that of Papua New Guinea for the calculation of the mean reproduction rate. As a result, the almost constant trend of the timeseries corresponding to Oceania can be attributed to the fact that the trends of Australia and Papua New Guinea as far as reproduction rate is concerned are inverse: whenever the rate is high for Australia, it is low for Papua New Guinea and vice versa.

**Correlation between different features**

Moving away from the geographic visualizations we proceed to study the correlation between different features on the country level. More specifically, we study the correlation between excess mortality and number of daily tests performed. The columns used for this purpose are excess\_mortality and new\_tests\_smoothed. The reason why new\_tests\_smoothed is used instead of new\_tests is because it contains less missing values compared to new\_tests. Since the analysis is performed on the country level, there is no reason to use normalized features, since the final metric is simply the correlation between different features corresponding to the same country.

It's important to note that not all countries are taken into account, but only these with more than 5 **non zero** entries for the excess\_mortality feature, for the reasons discussed previously as far as this feature is concerned. Without this filter, a lot of countries seem to show the maximum/minimum value of +1/-1 for the studied correlation, simply because there are very few entries. A typical example is that of Albania, for which the correlation is found equal to +1, simply because it has only 2 entries for the excess\_mortality feature during the months January-February 2021.

In [35]:

countries\_frame = df.select("location").distinct().collect()

exclusion\_list = ['Europe', 'World', 'Asia', 'North America', 'South America', 'Africa', 'Oceania', 'Upper middle income']

countries\_list = [str(countries\_frame[x][0]) for x **in** range(len(countries\_frame)) if str(countries\_frame[x][0]) **not** **in** exclusion\_list]

In [36]:

country\_dict = {}

for country **in** countries\_list:

filtered = df.filter(F.col('location') == country).filter(F.col('excess\_mortality') != 0.0)

if filtered.count() > 5:

value = filtered.stat.corr("excess\_mortality", "new\_tests\_smoothed")

else:

value = np.nan

if **not** np.isnan(value):

country\_dict[country] = value

In [37]:

country\_dict = dict(sorted(country\_dict.items(), reverse=True, key=lambda item: item[1]))

print("As far as the correlation between new tests and excess mortality is concerned:**\n**")

print("The ten countries with the highest correlation are:")

for i, ct **in** enumerate(country\_dict):

if i == 10: break

print(f"**{**ct**}**, with correlation equal to **{**country\_dict[ct]**:**.3f**}**.")

country\_dict = dict(sorted(country\_dict.items(), reverse=False, key=lambda item: item[1]))

print("**\n**The ten countries with the lowest correlation are:")

for i, ct **in** enumerate(country\_dict):

if i == 10: break

print(f"**{**ct**}**, with correlation equal to **{**country\_dict[ct]**:**.3f**}**.")

As far as the correlation between new tests and excess mortality is concerned:

The ten countries with the highest correlation are:

Spain, with correlation equal to 0.976.

South Africa, with correlation equal to 0.968.

Peru, with correlation equal to 0.962.

Portugal, with correlation equal to 0.960.

Mexico, with correlation equal to 0.921.

Colombia, with correlation equal to 0.916.

United States, with correlation equal to 0.794.

Lithuania, with correlation equal to 0.762.

South Korea, with correlation equal to 0.723.

Ecuador, with correlation equal to 0.670.

The ten countries with the lowest correlation are:

Slovakia, with correlation equal to -0.883.

Denmark, with correlation equal to -0.828.

Romania, with correlation equal to -0.762.

New Zealand, with correlation equal to -0.729.

Switzerland, with correlation equal to -0.715.

Italy, with correlation equal to -0.698.

Sweden, with correlation equal to -0.643.

Luxembourg, with correlation equal to -0.543.

Guatemala, with correlation equal to -0.483.

In [38]:

country\_dict = {}

for country **in** countries\_list:

filtered = df.filter(F.col('location') == country).filter(F.col('excess\_mortality') != 0.0)

if filtered.count() > 5:

value = filtered.stat.corr("excess\_mortality", "total\_vaccinations")

else:

value = np.nan

if **not** np.isnan(value):

country\_dict[country] = value

In [39]:

country\_dict = dict(sorted(country\_dict.items(), reverse=True, key=lambda item: item[1]))

print("As far as the correlation between excess mortality and the course of the vaccinations is concerned:**\n**")

print("The ten countries with the highest correlation are:")

for i, ct **in** enumerate(country\_dict):

if i == 10: break

print(f"**{**ct**}**, with correlation equal to **{**country\_dict[ct]**:**.3f**}**.")

country\_dict = dict(sorted(country\_dict.items(), reverse=False, key=lambda item: item[1]))

print("**\n**The ten countries with the lowest correlation are:")

for i, ct **in** enumerate(country\_dict):

if i == 10: break

print(f"**{**ct**}**, with correlation equal to **{**country\_dict[ct]**:**.3f**}**.")

As far as the correlation between excess mortality and the course of the vaccinations is concerned:

The ten countries with the highest correlation are:

Croatia, with correlation equal to 0.937.

Cyprus, with correlation equal to 0.694.

Malta, with correlation equal to 0.667.

Peru, with correlation equal to 0.584.

Ecuador, with correlation equal to 0.410.

Chile, with correlation equal to 0.410.

Finland, with correlation equal to 0.380.

Greece, with correlation equal to 0.246.

Australia, with correlation equal to 0.215.

Bulgaria, with correlation equal to 0.206.

The ten countries with the lowest correlation are:

Germany, with correlation equal to -0.986.

United States, with correlation equal to -0.985.

Switzerland, with correlation equal to -0.955.

Denmark, with correlation equal to -0.952.

Sweden, with correlation equal to -0.945.

Belgium, with correlation equal to -0.925.

Lithuania, with correlation equal to -0.921.

Latvia, with correlation equal to -0.905.

Canada, with correlation equal to -0.898.

Slovenia, with correlation equal to -0.896.

In this case, most countries show a negative correlation (and more specifically close to -1), since the increase in total vaccinations is expected to lead to a reduction in excess mortality, as vaccinations have proven to prevent serious infections from Covid. However, there are still countries such as Cyprus or Croatia, where the correlation is positive. There, it's possible that vaccines became available for the general population during the studied time interval and as a result their efficiency on combating the pandemic has not yet been observed on large scales.

**Covid and general health conditions on the country level**

Another interesting aspect of excess mortality is how it correlates with the general health conditions of a country's population. For this reason, we will first calculate the mean value of the 'female\_smokers', 'male\_smokers', 'diabetes\_prevalence' and 'cardiovasc\_death\_rate' features, using the data on the last available date of our filtered DataFrame. Then, we will sort all countries with respect to their excess mortality per million, since a normalization is required when comparing different countries (and hence different populations). Finally, we will compare the values of the aforementioned features for the top 5 and the bottom 5 countries with their calculated mean values.

In [40]:

this\_day = dates\_list[-1]

filtered\_df = df.filter(F.col('date') == this\_day)

mean\_fem\_smokers = filtered\_df.filter(F.col('female\_smokers') != 0.0).select(F.mean(F.col('female\_smokers'))).collect()[0][0]

mean\_male\_smokers = filtered\_df.filter(F.col('male\_smokers') != 0.0).select(F.mean(F.col('male\_smokers'))).collect()[0][0]

mean\_diabetes = filtered\_df.filter(F.col('diabetes\_prevalence') != 0.0).select(F.mean(F.col('diabetes\_prevalence'))).collect()[0][0]

mean\_card = filtered\_df.filter(F.col('cardiovasc\_death\_rate') != 0.0).select(F.mean(F.col('cardiovasc\_death\_rate'))).collect()[0][0]

print(f'Based on data up to **{**this\_day**}**, the mean percentage of female smokers is **{**mean\_fem\_smokers**:**.2f**}**%, while the corresponding number for male smokers is **{**mean\_male\_smokers**:**.2f**}**%.')

print(f'In addition, the mean percentage of people suffering from diabetes (aged 20-79) is **{**mean\_diabetes**:**.2f**}**%, while the mean number of deaths per 100.000 people due to cardiovascular conditions is **{**mean\_card**:**.2f**}**.')

Based on data up to 2021-02-28, the mean percentage of female smokers is 10.40%, while the corresponding number for male smokers is 32.69%.

In addition, the mean percentage of people suffering from diabetes (aged 20-79) is 8.27%, while the mean number of deaths per 100.000 people due to cardiovascular conditions is 263.17.

In [41]:

filtered\_df = filtered\_df.filter(F.col('diabetes\_prevalence') != 0.0).filter(F.col('cardiovasc\_death\_rate') != 0.0).filter(F.col('female\_smokers') != 0.0).filter(F.col('male\_smokers') != 0.0)

filtered\_df.orderBy("excess\_mortality\_cumulative\_per\_million", ascending=False).select(["location", "excess\_mortality\_cumulative\_per\_million", "female\_smokers", "male\_smokers", "diabetes\_prevalence", "cardiovasc\_death\_rate"]).toPandas().head(5)

Out[41]:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | location | excess\_mortality\_cumulative\_per\_million | female\_smokers | male\_smokers | diabetes\_prevalence | cardiovasc\_death\_rate |
| 0 | Armenia | 3613.725554 | 1.5 | 52.1 | 7.11 | 341.010 |
| 1 | Mexico | 3437.694368 | 6.9 | 21.4 | 13.06 | 152.783 |
| 2 | Belarus | 3287.984465 | 10.5 | 46.1 | 5.18 | 443.129 |
| 3 | Russia | 3151.540865 | 23.4 | 58.3 | 6.18 | 431.297 |
| 4 | Albania | 2854.503445 | 7.1 | 51.2 | 10.08 | 304.195 |

The table below depicts the divergence of these characteristics from their mean values for each of the top 5 countries with respect to excess mortality (per million). The red color is used for the entries which are higher than their corresponding mean value.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Female Smokers (%) | Male Smokers (%) | Diabetic Population (%) | Cardiovascular-related Deaths per 100.000 (%) |
| **Armenia** | -85.58 | 59.38 | -12.87 | 29.17 |
| **Mexico** | -33.65 | -34.54 | 60.05 | -42.14 |
| **Belarus** | 0.96 | 41.02 | -36.52 | 67.82 |
| **Russia** | 125 | 78.34 | -24.26 | 63.34 |
| **Albania** | -31.73 | 56.62 | 23.53 | 15.20 |

First and foremost, it's evident that all of the above countries have at least one of the mentioned features assume a value higher than it's mean. With the exception of Mexico, the countries with the highest excess mortality are characterized by numbers of cardiovascular related deaths that are higher compared to their mean value. The same countries also show significantly increased percentages of male smokers, which is definitely correlated to the number of deaths due to cardiovascular causes. The percentages of female smokers do not show the same tendency, excluding Russia, where the percentage of the divergence from the mean is higher than 100%. Finally, as far as the diabetic population is concerned, Mexico (for which we observed lower numbers with respect to the other features, compared to their mean values) has a value higher than the mean value by 60%.

In [42]:

filtered\_df.orderBy("excess\_mortality\_cumulative\_per\_million", ascending=True).select(["location", "excess\_mortality\_cumulative\_per\_million", "female\_smokers", "male\_smokers", "diabetes\_prevalence", "cardiovasc\_death\_rate"]).toPandas().head(5)

Out[42]:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | location | excess\_mortality\_cumulative\_per\_million | female\_smokers | male\_smokers | diabetes\_prevalence | cardiovasc\_death\_rate |
| 0 | Seychelles | -1678.293398 | 7.1 | 35.7 | 10.55 | 242.648 |
| 1 | Barbados | -832.093651 | 1.9 | 14.5 | 13.57 | 170.050 |
| 2 | Uruguay | -682.380568 | 14.0 | 19.9 | 6.93 | 160.708 |
| 3 | Mongolia | -605.836333 | 5.5 | 46.5 | 4.82 | 460.043 |
| 4 | Malaysia | -465.227889 | 1.0 | 42.4 | 16.74 | 260.942 |

Moving on to the bottom 5 countries as far as excess mortality is concerned, it is worth noting that the negative values in these cases is not because of a bug or missing data. On the contrary, it is because the reported numbers of deaths from these countries during the studied period were lower compared to their expected number, based on previous years' reports. A table similar to the one shown for the top 5 can be seen below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Female Smokers (%) | Male Smokers (%) | Diabetic Population (%) | Cardiovascular-related Deaths per 100.000 (%) |
| **Seychelles** | -31.73 | 9.21 | 29.29 | -8.11 |
| **Barbados** | -81.73 | -55.64 | 66.30 | -35.60 |
| **Uruguay** | 34.62 | -39.13 | -15.07 | -39.14 |
| **Mongolia** | -47.12 | 42.25 | -40.93 | 74.21 |
| **Malaysia** | -90.38 | 29.70 | 105.15 | -1.18 |

In this case as well, all countries have at least 1 studied feature with value higher than the corresponding mean. Nonetheless, the most important issue that was present for the top 5 countries, i.e. the elevated numbers of death by cardiovascular causes, does not seem to appear in this case as well, with the exception of Mongolia. When it comes to the percentage of smokers, it appears significantly reduced for both sexes. In contrast to what was observed for the top 5 countries, the countries with the lowest excess mortality tend to have increased diabetic populations.

Based on these, one could conclude that the excess mortality due to Covid can be connected to high percentages of smokers in the general population, as well as cardiovascular diseases. Similar conclusions cannot be drawn for the case of diabetes, which may be uncorrelated with deaths due to Covid. Of course, more extensive studies need to be performed in order to draw such conclusions safely, as well as more tests on target groups.

**k-Means Clustering**

Moving on to the final part of this EDA, we incorporate unsupervised learning methods, and more specifically k-Means clustering, in order to draw some additional information from our data. This is our final study on excess mortality and we intend to cluster countries together with respect to it, as well as the total number of cases - both normalized. This clustering will be performed on two different dates: the first and the final date present in our filtered DataFrame, in order to be able to see the evolution of the initial state. As previously done, we will only take into account countries with no missing values (i.e. zeroes) on excess mortality.

As is always the case with k-Means clustering, the question that needs to be answered is "what is the optimal value of k?". While we could use methods such as the Elbow method to determine a good value for k, the purpose of this notebook is not an extensive study of clustering, but rather the presentation of a few basic methods for EDA using PySpark instead of widely used libraries such as Pandas or scikit-learn, for relatively small datasets. For this reason, we will simply create a scatterplot of the data and determine an optimal value for k through the visualization.

In [43]:

sns.set(style = "darkgrid")

fig, [ax1,ax2] = plt.subplots(1, 2, figsize=(12,5))

for idx, (ax,this\_day) **in** enumerate(zip([ax1,ax2],[exc\_dates\_list[0],exc\_dates\_list[-1]])):

eff\_df = df.filter(F.col('excess\_mortality\_cumulative\_per\_million') != 0.0).filter(F.col('date') == this\_day).select(['total\_cases\_per\_million','excess\_mortality\_cumulative\_per\_million','location'])

pdf = eff\_df.select(['total\_cases\_per\_million','excess\_mortality\_cumulative\_per\_million']).toPandas()

points = ax.scatter(pdf.total\_cases\_per\_million, pdf.excess\_mortality\_cumulative\_per\_million,

color=mycol, alpha=0.5)

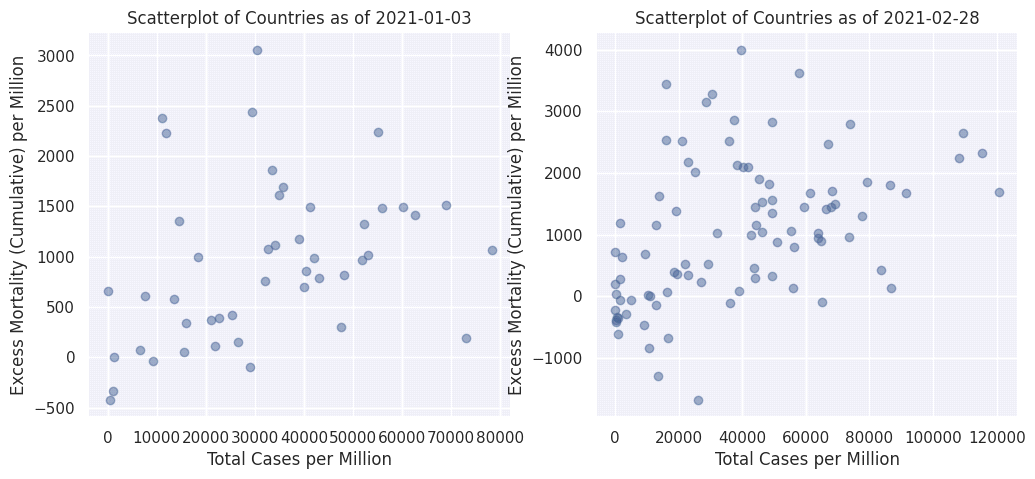
ax.set\_title(f'Scatterplot of Countries as of **{**this\_day**}**')

ax.set\_xlabel('Total Cases per Million')

ax.set\_ylabel('Excess Mortality (Cumulative) per Million')

plt.show()

matplotlib.rc\_file\_defaults()



Even through this preliminary visualization we can extract a very important conclusion as far as the data themselves are concerned: the number of countries which report on excess mortality has increased by the end of February, since the second scatterplot includes more points.

As far as the choice of k is concerned, i.e. the number of clusters to be taken into account, a reasonable hypothesis for the first date is k = 2: one cluster that includes the countries with fewer covid cases per million and one that includes the countries with more covid cases per million, since - with the exception of some outliers - it seems that the excess mortality is proportional to the number of total cases. On the other hand, things are somewhat more complex for the second date. We will be choosing k = 3 for this case, expecting to see the four countries with a very high number of cases (see the second scatterplot) in the same cluster.

In [44]:

from pyspark.ml.clustering import KMeans

sns.set(style = "darkgrid")

numclusters = [2,3]

colors = [mycol, mycomplcol, othercol1, othercol2, othercol3, othercol4]

fig, [ax1,ax2] = plt.subplots(1, 2, figsize=(14,5))

for idx, (ax,this\_day) **in** enumerate(zip([ax1,ax2],[exc\_dates\_list[0],exc\_dates\_list[-1]])):

eff\_df = df.filter(F.col('excess\_mortality\_cumulative\_per\_million') != 0.0).filter(F.col('date') == this\_day).filter(F.col('date') == this\_day).select(['total\_cases\_per\_million','excess\_mortality\_cumulative\_per\_million','location'])

vectorAssembler = VectorAssembler(inputCols = ['total\_cases\_per\_million','excess\_mortality\_cumulative\_per\_million'], outputCol = "features")

feat\_df = vectorAssembler.transform(eff\_df)

feat\_df = feat\_df.select(['features','location'])

kmeans = KMeans().setK(numclusters[idx]).setSeed(1).setFeaturesCol("features").setPredictionCol("cluster")

model = kmeans.fit(feat\_df)

transformed = model.transform(feat\_df)

centroids = model.clusterCenters()

transformed = transformed.join(eff\_df, 'location')

clusters, centers, images = {}, {}, {}

for i **in** range(numclusters[idx]):

clusters[i] = transformed.filter(F.col('cluster')==i).select(['location','cluster','total\_cases\_per\_million',

'excess\_mortality\_cumulative\_per\_million']).toPandas().set\_index('location')

images[i] = ax.scatter(clusters[i].total\_cases\_per\_million, clusters[i].excess\_mortality\_cumulative\_per\_million,

color=colors[i], alpha=0.5)

centers[i] = ax.scatter(centroids[i][0], centroids[i][1], color=colors[i], marker='x')

clusttuple = (images[i] for i **in** range(numclusters[idx]))

clustnames = ('Cluster '+str(i+1) for i **in** range(numclusters[idx]))

ax.legend(clusttuple, clustnames, loc='best')

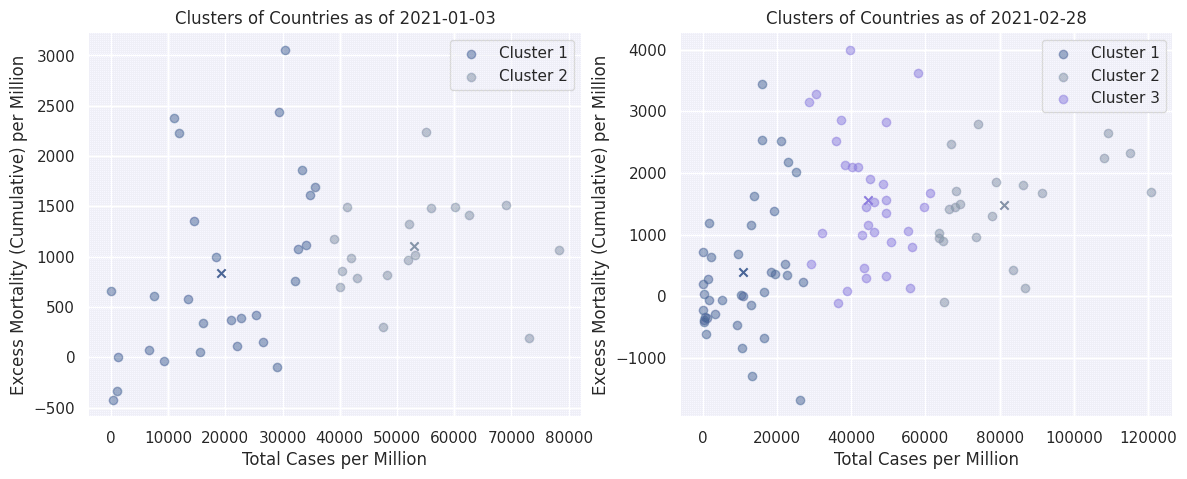
ax.set\_title(f'Clusters of Countries as of **{**this\_day**}**')

ax.set\_xlabel('Total Cases per Million')

ax.set\_ylabel('Excess Mortality (Cumulative) per Million')

plt.show()

matplotlib.rc\_file\_defaults()



In all cases, different clusters are shown with different colors, with each cluster's centroid depicted with an X. As far as the first date (early January) is concerned, it is indeed observed that all countries are split into the two clusters in a way that is expected from the initial visualization. As far as the second date (late February) is concerned, it appears that the 4 countries with more than 100.000 total cases per million (> 10%) indeed belong into the same cluster. All of the countries in this cluster are:

In [45]:

print(\*clusters[2].index, sep=', ')

Albania, Armenia, Austria, Belarus, Bosnia and Herzegovina, Brazil, Bulgaria, Chile, Colombia, Croatia, Cyprus, Denmark, Estonia, France, Germany, Hungary, Ireland, Italy, Kosovo, Latvia, Lebanon, Malta, Moldova, Monaco, North Macedonia, Peru, Poland, Qatar, Romania, Russia, Ukraine, United Kingdom

As far as the other two clusters are concerned, it appears that the clustering is based solely on the horizontal axis, i.e. the total number of cases per million. In the cluster with the centroid at ~53000 cases per million, 30 out of 39 countries are European countries. On the other hand, in the cluster with the centroid at ~13000 cases per million, there are countries from all over the world. Some of them are European, but under no circumstances do they correspond to the majority, as was the case in the other cluster.

Closing our investigation on clustering and the project itself, we perform the same steps in order to cluster countries with respect to the virus' reproduction rate and the countries' human development index.

In [46]:

sns.set(style = "darkgrid")

fig, [ax1,ax2] = plt.subplots(1, 2, figsize=(14,5))

for idx, (ax,this\_day) **in** enumerate(zip([ax1,ax2],[dates\_list[0],dates\_list[-1]])):

eff\_df = df.filter(F.col('human\_development\_index') != 0.0).filter(F.col('reproduction\_rate') != 0.0).filter(F.col('date') == this\_day).select(['human\_development\_index','reproduction\_rate','location'])

pdf = eff\_df.select(['human\_development\_index','reproduction\_rate']).toPandas()

points = ax.scatter(pdf.human\_development\_index, pdf.reproduction\_rate, color=mycol, alpha=0.5)

ax.set\_title(f'Scatterplot of Countries as of **{**this\_day**}**')

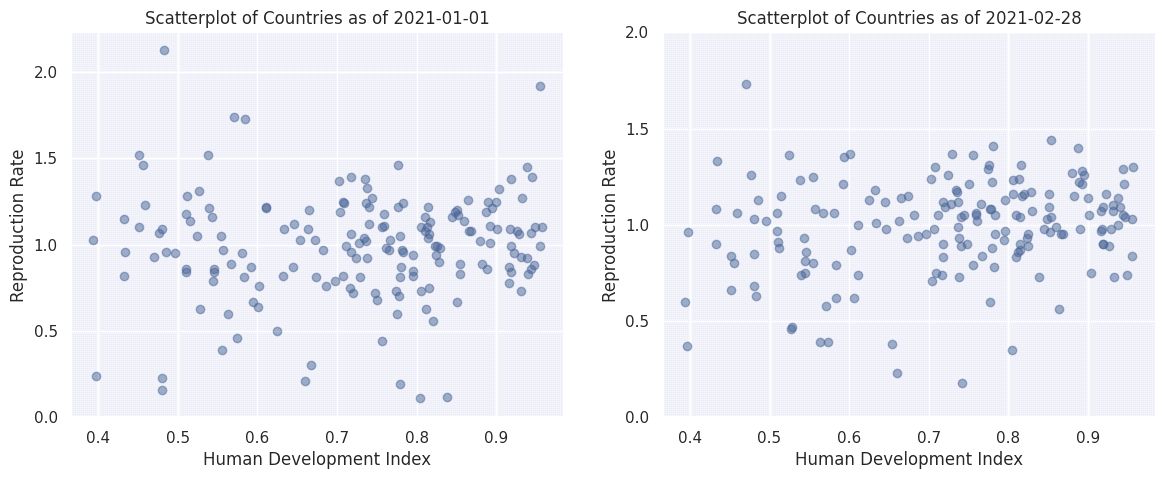
ax.set\_yticks([0.0,0.5,1.0,1.5,2.0])

ax.set\_xlabel('Human Development Index')

ax.set\_ylabel('Reproduction Rate')

plt.show()

matplotlib.rc\_file\_defaults()



In this case the number of studied countries seems to be constant in the two cases, unlike what we saw with excess mortality. Furthermore, especially in the final date, there seems to be a homogeneity as far as the reproduction rate is concerned. Before we move on with the clustering, we provide a .gif image (see [Appendix](https://www.kaggle.com/code/spyrosrigas/analyzing-covid-19-data-with-pyspark) for the code) where the time evolution of the scatterplots is shown for all dates in our filtered DataFrame.

As expected, all changes happen only along the vertical axis, since the human development index of a country is not expected to change during such a short period of time. The countries seem to perform an overall oscillatory motion. In addition, during the first days of January there is a sizeable number of countries with a reproduction rate higher than 1.5, however from a point forward the oscillatory motion seems to be limited in the [0.5,1.5] window, in terms of reproduction rate.

Moving on to the clustering for the first and the final day of the two-month period, k = 3 is chosen for both cases now, since tests performed using different numbers of k did not result into qualitatively interesting results.

In [47]:

sns.set(style = "darkgrid")

numclusters = [3,3]

colors = [mycol, mycomplcol, othercol1, othercol2, othercol3, othercol4]

fig, [ax1,ax2] = plt.subplots(1, 2, figsize=(14,5))

for idx, (ax,this\_day) **in** enumerate(zip([ax1,ax2],[dates\_list[0],dates\_list[-1]])):

eff\_df = df.filter(F.col('human\_development\_index') != 0.0).filter(F.col('reproduction\_rate') != 0.0).filter(F.col('date') == this\_day).select(['human\_development\_index','reproduction\_rate','location'])

vectorAssembler = VectorAssembler(inputCols = ['human\_development\_index','reproduction\_rate'], outputCol = "features")

feat\_df = vectorAssembler.transform(eff\_df)

feat\_df = feat\_df.select(['features','location'])

kmeans = KMeans().setK(numclusters[idx]).setSeed(1).setFeaturesCol("features").setPredictionCol("cluster")

model = kmeans.fit(feat\_df)

transformed = model.transform(feat\_df)

centroids = model.clusterCenters()

transformed = transformed.join(eff\_df, 'location')

clusters, centers, images = {}, {}, {}

for i **in** range(numclusters[idx]):

clusters[i] = transformed.filter(F.col('cluster')==i).select(['location','cluster','reproduction\_rate',

'human\_development\_index']).toPandas().set\_index('location')

images[i] = ax.scatter(clusters[i].human\_development\_index, clusters[i].reproduction\_rate, color=colors[i], alpha=0.5)

centers[i] = ax.scatter(centroids[i][0], centroids[i][1], color=colors[i], marker='x')

clusttuple = (images[i] for i **in** range(numclusters[idx]))

clustnames = ('Cluster '+str(i+1) for i **in** range(numclusters[idx]))

ax.legend(clusttuple, clustnames, loc='best')

ax.set\_title(f'Clusters of Countries as of **{**this\_day**}**')

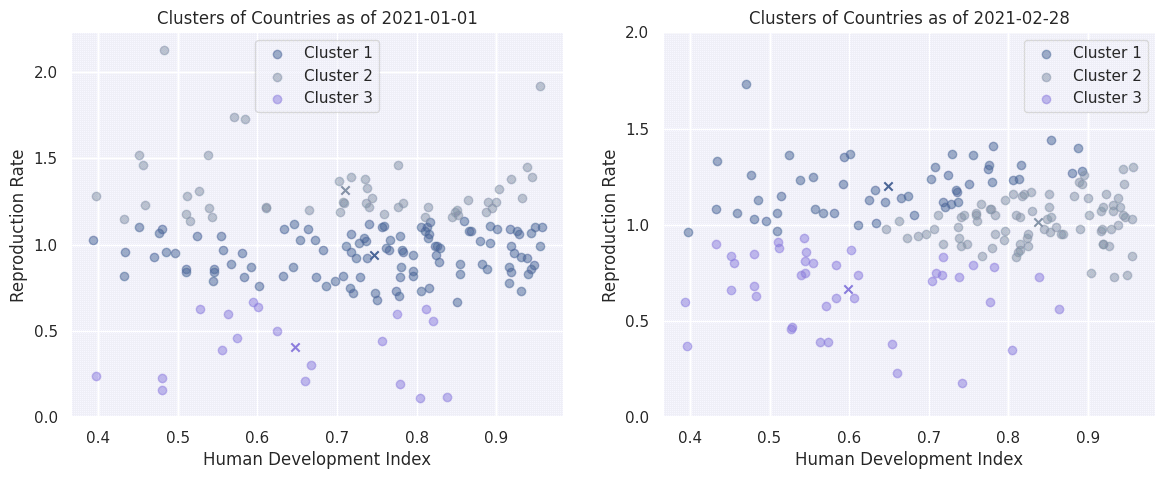
ax.set\_yticks([0.0,0.5,1.0,1.5,2.0])

ax.set\_xlabel('Human Development Index')

ax.set\_ylabel('Reproduction Rate')

plt.show()

matplotlib.rc\_file\_defaults()



linkcode

It's obvious that the k-Means clustering is performed with respect to the values of the vertical axis (reproduction rate). In both cases, one of the three clusters includes countries with a relatively small reproduction rate (less than 0.5), while the other two seem to split the countries using R = 1.0 as the decision boundary, as far as the reproduction rate is concerned.

As previously stated, this clustering analysis needs to be more carefully performed, perhaps using clustering algorithms with a different principles (for example density-based methods, such as DBSCAN). Unfortunately, PySpark does not yet include such implementations, therefore this task would require the use of scikit-learn in collaboration with Pandas.

**Exploratory data analysis:**

**Program:**

In [1]:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.express as px

from plotly.subplots import make\_subplots

from datetime import datetime

**Exploratory Data Analysis of Covid\_19\_India Dataset**

In [2]:

covid\_data = pd.read\_csv("../input/covid19-in-india/covid\_19\_india.csv")

covid\_data['Date'] = covid\_data['Date'].astype('datetime64[ns]')

covid\_data.head()

Out[2]:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Sno | Date | Time | State/UnionTerritory | ConfirmedIndianNational | ConfirmedForeignNational | Cured | Deaths | Confirmed |
| 0 | 1 | 2020-01-30 | 6:00 PM | Kerala | 1 | 0 | 0 | 0 | 1 |
| 1 | 2 | 2020-01-31 | 6:00 PM | Kerala | 1 | 0 | 0 | 0 | 1 |
| 2 | 3 | 2020-02-01 | 6:00 PM | Kerala | 2 | 0 | 0 | 0 | 2 |
| 3 | 4 | 2020-02-02 | 6:00 PM | Kerala | 3 | 0 | 0 | 0 | 3 |
| 4 | 5 | 2020-02-03 | 6:00 PM | Kerala | 3 | 0 | 0 | 0 | 3 |

In [3]:

covid\_data.shape

Out[3]:

(14150, 9)

In [4]:

covid\_data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 14150 entries, 0 to 14149

Data columns (total 9 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Sno 14150 non-null int64

1 Date 14150 non-null datetime64[ns]

2 Time 14150 non-null object

3 State/UnionTerritory 14150 non-null object

4 ConfirmedIndianNational 14150 non-null object

5 ConfirmedForeignNational 14150 non-null object

6 Cured 14150 non-null int64

7 Deaths 14150 non-null int64

8 Confirmed 14150 non-null int64

dtypes: datetime64[ns](1), int64(4), object(4)

memory usage: 995.0+ KB

In [5]:

covid\_data.describe()

Out[5]:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Sno | Cured | Deaths | Confirmed |
| count | 14150.00000 | 1.415000e+04 | 14150.000000 | 1.415000e+04 |
| mean | 7075.50000 | 1.540126e+05 | 2471.903816 | 1.681579e+05 |
| std | 4084.89749 | 3.105335e+05 | 6607.935316 | 3.384063e+05 |
| min | 1.00000 | 0.000000e+00 | 0.000000 | 0.000000e+00 |
| 25% | 3538.25000 | 1.225250e+03 | 10.000000 | 2.355250e+03 |
| 50% | 7075.50000 | 1.677250e+04 | 318.000000 | 2.102550e+04 |
| 75% | 10612.75000 | 1.782590e+05 | 1915.000000 | 2.059975e+05 |
| max | 14150.00000 | 3.330747e+06 | 62479.000000 | 4.094840e+06 |

**Statewise Analysis**

In [6]:

state\_wise = covid\_data.groupby('State/UnionTerritory')['Confirmed','Cured','Deaths'].sum().reset\_index()

state\_wise["Death\_percentage"] = ((state\_wise["Deaths"] / state\_wise["Confirmed"]) \* 100)

state\_wise.style.background\_gradient(cmap='magma')

/opt/conda/lib/python3.7/site-packages/ipykernel\_launcher.py:1: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a list instead.

"""Entry point for launching an IPython kernel.

Out[6]:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | State/UnionTerritory | Confirmed | Cured | Deaths | Death\_percentage |
| 0 | Andaman and Nicobar Islands | 1161624 | 1095239 | 14794 | 1.273562 |
| 1 | Andhra Pradesh | 205627560 | 195009537 | 1684138 | 0.819023 |
| 2 | Arunachal Pradesh | 3582775 | 3335949 | 10816 | 0.301889 |
| 3 | Assam | 51095731 | 47806964 | 231434 | 0.452942 |
| 4 | Bihar | 59502973 | 56326201 | 325733 | 0.547423 |
| 5 | Cases being reassigned to states | 345565 | 0 | 0 | 0.000000 |
| 6 | Chandigarh | 4497154 | 4098897 | 67640 | 1.504062 |
| 7 | Chhattisgarh | 60074333 | 53723225 | 694144 | 1.155475 |
| 8 | Dadra and Nagar Haveli and Daman and Diu | 866005 | 817775 | 582 | 0.067205 |
| 9 | Daman & Diu | 2 | 0 | 0 | 0.000000 |
| 10 | Delhi | 134827188 | 126242510 | 2399310 | 1.779545 |
| 11 | Goa | 11678197 | 10837849 | 161118 | 1.379648 |
| 12 | Gujarat | 58279923 | 52900802 | 1177367 | 2.020193 |
| 13 | Haryana | 55563899 | 51919242 | 605094 | 1.089006 |
| 14 | Himachal Pradesh | 10373705 | 9370148 | 166062 | 1.600797 |
| 15 | Jammu and Kashmir | 26971141 | 24677307 | 420238 | 1.558102 |
| 16 | Jharkhand | 26446302 | 24562965 | 236267 | 0.893384 |
| 17 | Karnataka | 209493180 | 192754571 | 2813076 | 1.342801 |
| 18 | Kerala | 166898288 | 151310286 | 657076 | 0.393698 |
| 19 | Ladakh | 2018868 | 1830434 | 25147 | 1.245599 |
| 20 | Lakshadweep | 46403 | 34913 | 57 | 0.122837 |
| 21 | Madhya Pradesh | 54165387 | 49471433 | 857399 | 1.582928 |
| 22 | Maharashtra | 494842275 | 432346106 | 12168696 | 2.459106 |
| 23 | Manipur | 5691040 | 5196442 | 63765 | 1.120445 |
| 24 | Meghalaya | 2721780 | 2485346 | 26987 | 0.991520 |
| 25 | Mizoram | 877555 | 807083 | 1442 | 0.164320 |
| 26 | Nagaland | 2557096 | 2337373 | 15235 | 0.595793 |
| 27 | Odisha | 72946835 | 69441871 | 384313 | 0.526840 |
| 28 | Puducherry | 8743641 | 8055805 | 146461 | 1.675057 |
| 29 | Punjab | 39895309 | 35785710 | 1215922 | 3.047782 |
| 30 | Rajasthan | 64864781 | 59970518 | 606176 | 0.934523 |
| 31 | Sikkim | 1219550 | 1111371 | 23817 | 1.952933 |
| 32 | Tamil Nadu | 198338601 | 186328161 | 2969358 | 1.497116 |
| 33 | Telengana | 66667750 | 61921079 | 383400 | 0.575091 |
| 34 | Tripura | 7596122 | 7049046 | 84812 | 1.116517 |
| 35 | Unassigned | 161 | 0 | 0 | 0.000000 |
| 36 | Uttar Pradesh | 133829373 | 122698961 | 1930554 | 1.442549 |
| 37 | Uttarakhand | 19218385 | 17580606 | 311929 | 1.623076 |
| 38 | West Bengal | 115907271 | 108036513 | 2097080 | 1.809274 |

In [7]:

px.bar(x=state\_wise.nlargest(10,"Confirmed")["State/UnionTerritory"],

y = state\_wise.nlargest(10,"Confirmed")["Confirmed"],

color\_discrete\_sequence=px.colors.diverging.Picnic,

title="Top 10 states with highest number of Confirmed cases")

MaharashtraKarnatakaAndhra PradeshTamil NaduKeralaDelhiUttar PradeshWest BengalOdishaTelengana0100M200M300M400M500M

Top 10 states with highest number of Confirmed casesxy

In [8]:

px.bar(x=state\_wise.nlargest(10,"Cured")["State/UnionTerritory"],

y = state\_wise.nlargest(10,"Cured")["Cured"],

color\_discrete\_sequence=px.colors.sequential.Sunset,

title="Top 10 states with highest number of Cured cases")

MaharashtraAndhra PradeshKarnatakaTamil NaduKeralaDelhiUttar PradeshWest BengalOdishaTelengana050M100M150M200M250M300M350M400M450M

Top 10 states with highest number of Cured casesxy

In [9]:

px.bar(x=state\_wise.nlargest(10,"Deaths")["State/UnionTerritory"],

y = state\_wise.nlargest(10,"Deaths")["Deaths"],

color\_discrete\_sequence=px.colors.diverging.curl,

title="Top 10 states with highest number of Deaths")

MaharashtraTamil NaduKarnatakaDelhiWest BengalUttar PradeshAndhra PradeshPunjabGujaratMadhya Pradesh02M4M6M8M10M12M

Top 10 states with highest number of Deathsxy

In [10]:

px.bar(x=state\_wise.nlargest(10,"Death\_percentage")["State/UnionTerritory"],

y = state\_wise.nlargest(10,"Death\_percentage")["Death\_percentage"],

color\_discrete\_sequence=px.colors.diverging.Portland,

title="Top 10 states with highest of Death percentage")

PunjabMaharashtraGujaratSikkimWest BengalDelhiPuducherryUttarakhandHimachal PradeshMadhya Pradesh00.511.522.53

Top 10 states with highest of Death percentagexy

**Monthwise Analysis**

In [11]:

month\_wise = covid\_data.groupby(pd.Grouper(key='Date',freq='M')).sum()

month\_wise = month\_wise.drop(['Sno'], axis = 1)

month\_wise['Date'] = month\_wise.index

first\_column = month\_wise.pop('Date')

month\_wise.insert(0, 'Date', first\_column)

index = [x for x **in** range(len(month\_wise))]

month\_wise['index'] = index

month\_wise = month\_wise.set\_index('index')

second\_column = month\_wise.pop('Confirmed')

month\_wise.insert(1, 'Confirmed', second\_column)

month\_wise["Death\_percentage"] = ((month\_wise["Deaths"] / month\_wise["Confirmed"]) \* 100)

month\_wise.style.background\_gradient(cmap='twilight\_shifted')

Out[11]:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Date | Confirmed | Cured | Deaths | Death\_percentage |
| index |  |  |  |  |  |
| 0 | 2020-01-31 00:00:00 | 2 | 0 | 0 | 0.000000 |
| 1 | 2020-02-29 00:00:00 | 86 | 0 | 0 | 0.000000 |
| 2 | 2020-03-31 00:00:00 | 9687 | 808 | 202 | 2.085269 |
| 3 | 2020-04-30 00:00:00 | 422442 | 75443 | 13270 | 3.141260 |
| 4 | 2020-05-31 00:00:00 | 2938234 | 1133341 | 89834 | 3.057415 |
| 5 | 2020-06-30 00:00:00 | 10558374 | 5668946 | 319690 | 3.027834 |
| 6 | 2020-07-31 00:00:00 | 31726501 | 19980130 | 793511 | 2.501098 |
| 7 | 2020-08-31 00:00:00 | 80749620 | 58580895 | 1553468 | 1.923808 |
| 8 | 2020-09-30 00:00:00 | 149113758 | 118592934 | 2443374 | 1.638597 |
| 9 | 2020-10-31 00:00:00 | 226770312 | 198824412 | 3457615 | 1.524721 |
| 10 | 2020-11-30 00:00:00 | 264556412 | 246213201 | 3894165 | 1.471960 |
| 11 | 2020-12-31 00:00:00 | 307177353 | 292244085 | 4457379 | 1.451077 |
| 12 | 2021-01-31 00:00:00 | 326469747 | 315332019 | 4709167 | 1.442451 |
| 13 | 2021-02-28 00:00:00 | 305631803 | 297133802 | 4359434 | 1.426368 |
| 14 | 2021-03-31 00:00:00 | 356305616 | 342610397 | 4935253 | 1.385118 |
| 15 | 2021-04-30 00:00:00 | 317003781 | 282887825 | 3951077 | 1.246382 |

pIn [12]:

fig = px.bar(month\_wise, x='Date', y='Confirmed',

hover\_data=['Cured', 'Deaths'], color='Date',

labels={'Date':'Date(monthwise)'},

title="Monthwise Increase in Confirmed cases")

fig.show()

Jul 2020Jan 2021050M100M150M200M250M300M350M

Date(monthwise)2020-01-31 00:00:002020-02-29 00:00:002020-03-31 00:00:002020-04-30 00:00:002020-05-31 00:00:002020-06-30 00:00:002020-07-31 00:00:002020-08-31 00:00:002020-09-30 00:00:002020-10-31 00:00:002020-11-30 00:00:002020-12-31 00:00:002021-01-31 00:00:002021-02-28 00:00:002021-03-31 00:00:002021-04-30 00:00:00Monthwise Increase in Confirmed casesDate(monthwise)Confirmed

In [13]:

fig = px.bar(month\_wise, x='Date', y='Cured',

hover\_data=['Confirmed','Deaths'], color='Date',

labels={'Date':'Date(monthwise)'},

title="Monthwise Increase in Cured cases")

fig.show()

Jul 2020Jan 2021050M100M150M200M250M300M350M

Date(monthwise)2020-01-31 00:00:002020-02-29 00:00:002020-03-31 00:00:002020-04-30 00:00:002020-05-31 00:00:002020-06-30 00:00:002020-07-31 00:00:002020-08-31 00:00:002020-09-30 00:00:002020-10-31 00:00:002020-11-30 00:00:002020-12-31 00:00:002021-01-31 00:00:002021-02-28 00:00:002021-03-31 00:00:002021-04-30 00:00:00Monthwise Increase in Cured casesDate(monthwise)Cured

In [14]:

fig = px.bar(month\_wise, x='Date', y='Deaths',

hover\_data=['Confirmed','Cured'], color='Date',

labels={'Date':'Date(monthwise)'},

title="Monthwise Increase in Deaths cases")

fig.show()

Jul 2020Jan 202101M2M3M4M5M

Date(monthwise)2020-01-31 00:00:002020-02-29 00:00:002020-03-31 00:00:002020-04-30 00:00:002020-05-31 00:00:002020-06-30 00:00:002020-07-31 00:00:002020-08-31 00:00:002020-09-30 00:00:002020-10-31 00:00:002020-11-30 00:00:002020-12-31 00:00:002021-01-31 00:00:002021-02-28 00:00:002021-03-31 00:00:002021-04-30 00:00:00Monthwise Increase in Deaths casesDate(monthwise)Deaths

In [15]:

fig = px.bar(month\_wise ,

x='Date',

y='Death\_percentage' ,

hover\_data=['Confirmed','Deaths'],color='Date',

labels={'Death\_percentage':'Death percentage'},

title="Top 10 states with highest of Death percentage")

fig.show()

Jul 2020Jan 202100.511.522.53

Date2020-01-31 00:00:002020-02-29 00:00:002020-03-31 00:00:002020-04-30 00:00:002020-05-31 00:00:002020-06-30 00:00:002020-07-31 00:00:002020-08-31 00:00:002020-09-30 00:00:002020-10-31 00:00:002020-11-30 00:00:002020-12-31 00:00:002021-01-31 00:00:002021-02-28 00:00:002021-03-31 00:00:002021-04-30 00:00:00Top 10 states with highest of Death percentageDateDeath percentage

**Exploratory Data Analysis of StatewiseTestingDetails Dataset**

In [16]:

covid\_testing = pd.read\_csv("../input/covid19-in-india/StatewiseTestingDetails.csv")

covid\_testing['Date'] = covid\_testing['Date'].astype('datetime64[ns]')

covid\_testing.head()

Out[16]:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Date | State | TotalSamples | Negative | Positive |
| 0 | 2020-04-17 | Andaman and Nicobar Islands | 1403.0 | 1210.0 | 12.0 |
| 1 | 2020-04-24 | Andaman and Nicobar Islands | 2679.0 | NaN | 27.0 |
| 2 | 2020-04-27 | Andaman and Nicobar Islands | 2848.0 | NaN | 33.0 |
| 3 | 2020-05-01 | Andaman and Nicobar Islands | 3754.0 | NaN | 33.0 |
| 4 | 2020-04-02 | Andhra Pradesh | 1800.0 | 1175.0 | 132.0 |

In [17]:

covid\_testing.shape

Out[17]:

(926, 5)

In [18]:

covid\_testing.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 926 entries, 0 to 925

Data columns (total 5 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Date 926 non-null datetime64[ns]

1 State 926 non-null object

2 TotalSamples 926 non-null float64

3 Negative 756 non-null float64

4 Positive 918 non-null float64

dtypes: datetime64[ns](1), float64(3), object(1)

memory usage: 36.3+ KB

In [19]:

covid\_testing['Negative'] = covid\_testing['TotalSamples'] - covid\_testing['Positive']

covid\_testing = covid\_testing.dropna()

covid\_testing.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 918 entries, 0 to 925

Data columns (total 5 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Date 918 non-null datetime64[ns]

1 State 918 non-null object

2 TotalSamples 918 non-null float64

3 Negative 918 non-null float64

4 Positive 918 non-null float64

dtypes: datetime64[ns](1), float64(3), object(1)

memory usage: 43.0+ KB

**Statewise Analysis**

In [20]:

covid\_testing\_state = covid\_testing.groupby('State')['TotalSamples','Negative','Positive'].max().reset\_index()

covid\_testing\_state["Positive\_percentage"] = ((covid\_testing["Positive"] / covid\_testing["TotalSamples"]) \* 100)

covid\_testing\_state.style.background\_gradient(cmap='gist\_earth\_r')

/opt/conda/lib/python3.7/site-packages/ipykernel\_launcher.py:1: FutureWarning:

Indexing with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a list instead.

Out[20]:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | State | TotalSamples | Negative | Positive | Positive\_percentage |
| 0 | Andaman and Nicobar Islands | 3754.000000 | 3721.000000 | 33.000000 | 0.855310 |
| 1 | Andhra Pradesh | 173735.000000 | 171755.000000 | 1980.000000 | 1.007839 |
| 2 | Arunachal Pradesh | 1823.000000 | 1821.000000 | 2.000000 | 1.158708 |
| 3 | Assam | 18002.000000 | 17940.000000 | 62.000000 | 0.879062 |
| 4 | Bihar | 36053.000000 | 35346.000000 | 707.000000 | 7.333333 |
| 5 | Chandigarh | 2142.000000 | 1969.000000 | 173.000000 | 5.726388 |
| 6 | Chhattisgarh | 25282.000000 | 25223.000000 | 59.000000 | 5.475711 |
| 7 | Delhi | 97678.000000 | 90445.000000 | 7233.000000 | 5.820638 |
| 8 | Goa | 4848.000000 | 4841.000000 | 7.000000 | 4.934323 |
| 9 | Gujarat | 113493.000000 | 105298.000000 | 8195.000000 | 4.502618 |
| 10 | Haryana | 56983.000000 | 56280.000000 | 703.000000 | 4.520796 |
| 11 | Himachal Pradesh | 10791.000000 | 10736.000000 | 55.000000 | 2.638992 |
| 12 | Jammu and Kashmir | 47080.000000 | 46219.000000 | 861.000000 | 2.826785 |
| 13 | Jharkhand | 20832.000000 | 20675.000000 | 157.000000 | 2.811189 |
| 14 | Karnataka | 107311.000000 | 106463.000000 | 848.000000 | 2.400030 |
| 15 | Kerala | 37464.000000 | 36952.000000 | 512.000000 | 2.349266 |
| 16 | Ladakh | 3503.000000 | 3461.000000 | 42.000000 | 2.117186 |
| 17 | Madhya Pradesh | 72069.000000 | 68455.000000 | 3614.000000 | 1.958470 |
| 18 | Maharashtra | 225524.000000 | 205296.000000 | 20228.000000 | 1.859177 |
| 19 | Meghalaya | 2287.000000 | 2275.000000 | 13.000000 | 1.757518 |
| 20 | Mizoram | 201.000000 | 200.000000 | 1.000000 | 1.658342 |
| 21 | Nagaland | 862.000000 | 862.000000 | 0.000000 | 1.612429 |
| 22 | Odisha | 59780.000000 | 59418.000000 | 362.000000 | 1.578785 |
| 23 | Puducherry | 4364.000000 | 4347.000000 | 17.000000 | 1.567207 |
| 24 | Punjab | 40962.000000 | 39139.000000 | 1823.000000 | 1.512588 |
| 25 | Rajasthan | 166424.000000 | 162610.000000 | 3814.000000 | 1.483745 |
| 26 | Sikkim | 219.000000 | 219.000000 | 0.000000 | 1.427874 |
| 27 | Tamil Nadu | 243037.000000 | 235833.000000 | 7204.000000 | 1.406788 |
| 28 | Telangana | 19278.000000 | 18262.000000 | 1016.000000 | 1.377276 |
| 29 | Tripura | 9091.000000 | 8955.000000 | 136.000000 | 1.317586 |
| 30 | Uttar Pradesh | 129955.000000 | 126488.000000 | 3467.000000 | 1.286219 |
| 31 | Uttarakhand | 9668.000000 | 9600.000000 | 68.000000 | 1.257839 |
| 32 | West Bengal | 43414.000000 | 41475.000000 | 1939.000000 | 1.227228 |

In [21]:

px.bar(x=covid\_testing\_state.nlargest(10,"TotalSamples")["State"],

y = covid\_testing\_state.nlargest(10,"TotalSamples")["TotalSamples"],

labels={'y':'Total Samples','x':'State'},

color\_discrete\_sequence=px.colors.sequential.haline,

title="Top 10 states with highest number of Total Samples")

Tamil NaduMaharashtraAndhra PradeshRajasthanUttar PradeshGujaratKarnatakaDelhiMadhya PradeshOdisha050k100k150k200k250k

Top 10 states with highest number of Total SamplesStateTotal Samples

In [22]:

px.bar(x=covid\_testing\_state.nlargest(10,"Negative")["State"],

y = covid\_testing\_state.nlargest(10,"Negative")["Negative"],

labels={'y':'Total Negative cases','x':'State'},

color\_discrete\_sequence=px.colors.sequential.turbid,

title="Top 10 states with highest number of Negative cases")

Tamil NaduMaharashtraAndhra PradeshRajasthanUttar PradeshKarnatakaGujaratDelhiMadhya PradeshOdisha050k100k150k200k

Top 10 states with highest number of Negative casesStateTotal Negative cases

In [23]:

px.bar(x=covid\_testing\_state.nlargest(10,"Positive")["State"],

y = covid\_testing\_state.nlargest(10,"Positive")["Positive"],

labels={'y':'Total Positive Cases','x':'State'},

color\_discrete\_sequence=px.colors.sequential.solar,

title="Top 10 states with highest number of Positive cases")

MaharashtraGujaratDelhiTamil NaduRajasthanMadhya PradeshUttar PradeshAndhra PradeshWest BengalPunjab05k10k15k20k

Top 10 states with highest number of Positive casesStateTotal Positive Cases

In [24]:

px.bar(x=covid\_testing\_state.nlargest(10,"Positive\_percentage")["State"],

y = covid\_testing\_state.nlargest(10,"Positive\_percentage")["Positive\_percentage"],

labels={'y':'Positive Percentage','x':'State'},

color\_discrete\_sequence=px.colors.sequential.Aggrnyl,

title="Top 10 states with highest Positive percentage",

height = 420)

BiharDelhiChandigarhChhattisgarhGoaHaryanaGujaratJammu and KashmirJharkhandHimachal Pradesh0246

Top 10 states with highest Positive percentageStatePositive Percentage

**Statistical analysis & Visualization:**

**Program:**

*# This Python 3 environment comes with many helpful analytics libraries installed*

*# It is defined by the kaggle/python Docker image:* [*https://github.com/kaggle/docker-python*](https://github.com/kaggle/docker-python)

*# For example, here's several helpful packages to load*

import numpy as np *# linear algebra*

import pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*

*# Input data files are available in the read-only "../input/" directory*

*# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory*

import os

for dirname, \_, filenames **in** os.walk('/kaggle/input'):

for filename **in** filenames:

print(os.path.join(dirname, filename))

*# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"*

*# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session*

/kaggle/input/covid-world-vaccination-progress/country\_vaccinations\_by\_manufacturer.csv

/kaggle/input/covid-world-vaccination-progress/country\_vaccinations.csv

In [2]:

*#importing essential libraries*

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings("ignore")

In [3]:

*#reading dataset*

df= pd.read\_csv("../input/covid-world-vaccination-progress/country\_vaccinations.csv")

In [4]:

df.head()

Out[4]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 |
| 0 | Afghanistan | AFG | 2021-02-22 | 0.0 | 0.0 | NaN | NaN | NaN | 0.0 | 0.0 | NaN | NaN | Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi... | World Health Organization | <https://covid19.who.int/> |
| 1 | Afghanistan | AFG | 2021-02-23 | NaN | NaN | NaN | NaN | 1367.0 | NaN | NaN | NaN | 34.0 | Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi... | World Health Organization | <https://covid19.who.int/> |
| 2 | Afghanistan | AFG | 2021-02-24 | NaN | NaN | NaN | NaN | 1367.0 | NaN | NaN | NaN | 34.0 | Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi... | World Health Organization | <https://covid19.who.int/> |
| 3 | Afghanistan | AFG | 2021-02-25 | NaN | NaN | NaN | NaN | 1367.0 | NaN | NaN | NaN | 34.0 | Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi... | World Health Organization | <https://covid19.who.int/> |
| 4 | Afghanistan | AFG | 2021-02-26 | NaN | NaN | NaN | NaN | 1367.0 | NaN | NaN | NaN | 34.0 | Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi... | World Health Organization | <https://covid19.who.int/> |

In [5]:

df.describe()

Out[5]:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 |
| count | 4.360700e+04 | 4.129400e+04 | 3.880200e+04 | 3.536200e+04 | 8.621300e+04 | 43607.000000 | 41294.000000 | 38802.000000 | 86213.000000 |
| mean | 4.592964e+07 | 1.770508e+07 | 1.413830e+07 | 2.705996e+05 | 1.313055e+05 | 80.188543 | 40.927317 | 35.523243 | 3257.049157 |
| std | 2.246004e+08 | 7.078731e+07 | 5.713920e+07 | 1.212427e+06 | 7.682388e+05 | 67.913577 | 29.290759 | 28.376252 | 3934.312440 |
| min | 0.000000e+00 | 0.000000e+00 | 1.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 5.264100e+05 | 3.494642e+05 | 2.439622e+05 | 4.668000e+03 | 9.000000e+02 | 16.050000 | 11.370000 | 7.020000 | 636.000000 |
| 50% | 3.590096e+06 | 2.187310e+06 | 1.722140e+06 | 2.530900e+04 | 7.343000e+03 | 67.520000 | 41.435000 | 31.750000 | 2050.000000 |
| 75% | 1.701230e+07 | 9.152520e+06 | 7.559870e+06 | 1.234925e+05 | 4.409800e+04 | 132.735000 | 67.910000 | 62.080000 | 4682.000000 |
| max | 3.263129e+09 | 1.275541e+09 | 1.240777e+09 | 2.474100e+07 | 2.242429e+07 | 345.370000 | 124.760000 | 122.370000 | 117497.000000 |

In [6]:

df.dtypes

Out[6]:

country object

iso\_code object

date object

total\_vaccinations float64

people\_vaccinated float64

people\_fully\_vaccinated float64

daily\_vaccinations\_raw float64

daily\_vaccinations float64

total\_vaccinations\_per\_hundred float64

people\_vaccinated\_per\_hundred float64

people\_fully\_vaccinated\_per\_hundred float64

daily\_vaccinations\_per\_million float64

vaccines object

source\_name object

source\_website object

dtype: object

In [7]:

*#converting date column datatype to date*

df["date"]= pd.to\_datetime(df.date)

In [8]:

df["Total\_vaccinations(count)"]= df.groupby("country").total\_vaccinations.tail(1)

In [9]:

*#Top countries with most vaccinations*

df.groupby("country")["Total\_vaccinations(count)"].mean().sort\_values(ascending= False).head(20)

Out[9]:

country

China 3.263129e+09

India 1.834501e+09

United States 5.601818e+08

Brazil 4.135596e+08

Indonesia 3.771089e+08

Japan 2.543456e+08

Bangladesh 2.436427e+08

Pakistan 2.193686e+08

Vietnam 2.031444e+08

Mexico 1.919079e+08

Germany 1.719400e+08

Russia 1.636012e+08

Philippines 1.487991e+08

Turkey 1.468819e+08

Iran 1.467926e+08

France 1.416662e+08

United Kingdom 1.409683e+08

Italy 1.358709e+08

Thailand 1.288824e+08

South Korea 1.206045e+08

Name: Total\_vaccinations(count), dtype: float64

In [10]:

*#barplot visualization of top countries with most vaccinations*

x= df.groupby("country")["Total\_vaccinations(count)"].mean().sort\_values(ascending= False).head(20)

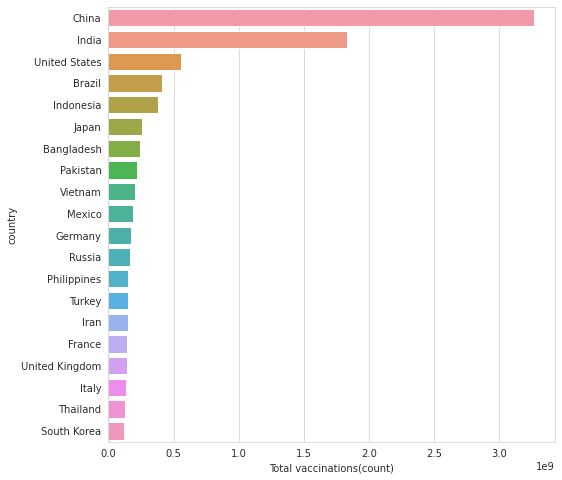
sns.set\_style("whitegrid")

plt.figure(figsize= (8,8))

ax= sns.barplot(x.values,x.index)

ax.set\_xlabel("Total vaccinations(count)")

plt.show()



In [11]:

*#Top countries with fully vaccinated peoples*

df["Full\_vaccinations(count)"]= df.groupby("country").people\_fully\_vaccinated.tail(1)

df.groupby("country")["Full\_vaccinations(count)"].mean().sort\_values(ascending= False).head(20)

Out[11]:

country

India 828229455.0

United States 217498967.0

Brazil 160272858.0

Indonesia 158830466.0

Bangladesh 107712737.0

Pakistan 101881176.0

Japan 100633737.0

Mexico 79711762.0

Vietnam 77754108.0

Russia 72841232.0

Philippines 65804988.0

Germany 63142649.0

Iran 56810058.0

Turkey 52968985.0

France 52438706.0

Thailand 50159803.0

United Kingdom 49404026.0

Italy 47817555.0

South Korea 44482876.0

England 41501690.0

Name: Full\_vaccinations(count), dtype: float64

In [12]:

*#barplot visualization of top countries with most full vaccinations*

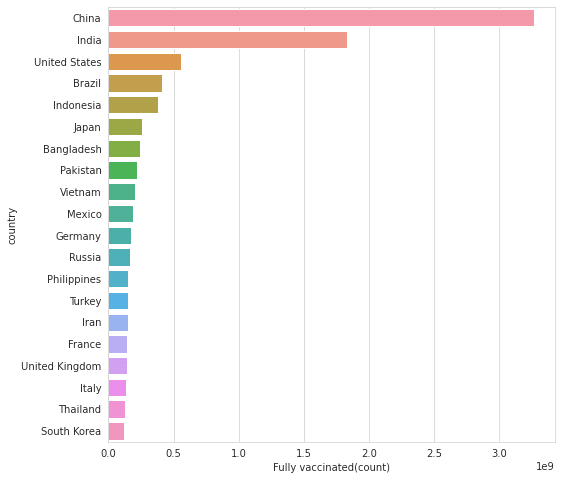
sns.set\_style("whitegrid")

plt.figure(figsize= (8,8))

ax= sns.barplot(x.values,x.index)

ax.set\_xlabel("Fully vaccinated(count)")

plt.show()



In [13]:

*#Vaccine types*

x=df.vaccines.unique()

y= list(x)

for i **in** y: print(i)

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

Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing

Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V

Moderna, Pfizer/BioNTech

Covaxin, Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac

Johnson&Johnson, Oxford/AstraZeneca

Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing

Johnson&Johnson, Oxford/AstraZeneca, Sinopharm/Beijing

Sinopharm/Beijing

Johnson&Johnson, Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac

Covaxin, Oxford/AstraZeneca

CanSino, Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac

CanSino, Sinopharm/Beijing, Sinopharm/Wuhan, Sinovac, ZF2001

Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac

Covaxin, Oxford/AstraZeneca, Sinopharm/Beijing

Moderna, Oxford/AstraZeneca, Sinopharm/Beijing, Sputnik V

Abdala, Soberana Plus, Soberana02

Johnson&Johnson, Moderna, Pfizer/BioNTech

Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac, Sputnik V

Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac

Covaxin, Johnson&Johnson, Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac

Johnson&Johnson, Pfizer/BioNTech

Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V

Oxford/AstraZeneca, Sputnik V

Moderna

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sputnik V

Oxford/AstraZeneca, Sinopharm/Beijing

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V

Johnson&Johnson, Moderna

Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sputnik V

Pfizer/BioNTech, Sinovac

Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V

Covaxin, Oxford/AstraZeneca, Sputnik V

Johnson&Johnson, Moderna, Novavax, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac

COVIran Barekat, Covaxin, FAKHRAVAC, Oxford/AstraZeneca, Razi Cov Pars, Sinopharm/Beijing, Soberana02, SpikoGen, Sputnik V

Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V

QazVac, Sinopharm/Beijing, Sputnik V

Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac, Sputnik Light, Sputnik V

Johnson&Johnson, Moderna, Novavax, Pfizer/BioNTech

Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac, Sputnik V

Pfizer/BioNTech, Sinopharm/Beijing

CanSino, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac

CanSino, Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac, Sputnik V

Abdala, Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech, Soberana02, Sputnik Light, Sputnik V

Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac

CanSino, Covaxin, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac, Sputnik V

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

Covaxin, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac, Sputnik V

EpiVacCorona, Sputnik V

Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac, Sputnik V

Pfizer/BioNTech, Sputnik V

Oxford/AstraZeneca, Sinopharm/Beijing, Sputnik V

Moderna, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac

Johnson&Johnson, Moderna, Novavax, Oxford/AstraZeneca, Pfizer/BioNTech, Sputnik V

Johnson&Johnson, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac

Johnson&Johnson, Oxford/AstraZeneca, Sinopharm/Beijing, Sinovac, Sputnik Light, Sputnik V

Medigen, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac, Sputnik V

Johnson&Johnson, Pfizer/BioNTech, Sinopharm/Beijing

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinovac

Pfizer/BioNTech, Sinovac, Turkovac

EpiVacCorona, Oxford/AstraZeneca, QazVac, Sinopharm/Beijing, Sputnik V, ZF2001

Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sinopharm/Wuhan, Sputnik V

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac, Sputnik Light, Sputnik V, ZF2001

Abdala, Sinopharm/Beijing, Sinovac, Soberana02, Sputnik Light, Sputnik V

Abdala, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinopharm/Beijing, Sputnik V

Johnson&Johnson, Oxford/AstraZeneca, Sinovac

In [14]:

*#most common vaccines*

df.vaccines.value\_counts()

Out[14]:

Johnson&Johnson, Moderna, Oxford/AstraZeneca, Pfizer/BioNTech 7608

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech 6263

Oxford/AstraZeneca 6022

Oxford/AstraZeneca, Pfizer/BioNTech 4629

Johnson&Johnson, Moderna, Novavax, Oxford/AstraZeneca, Pfizer/BioNTech 3564

...

Johnson&Johnson, Oxford/AstraZeneca, Sinovac 312

Moderna, Oxford/AstraZeneca, Pfizer/BioNTech, Sinovac, Sputnik V 311

Johnson&Johnson, Moderna 251

Johnson&Johnson, Pfizer/BioNTech, Sinopharm/Beijing 228

EpiVacCorona, Oxford/AstraZeneca, QazVac, Sinopharm/Beijing, Sputnik V, ZF2001 190

Name: vaccines, Length: 84, dtype: int64

In [15]:

from wordcloud import WordCloud, STOPWORDS

plt.figure(figsize= (20,20))

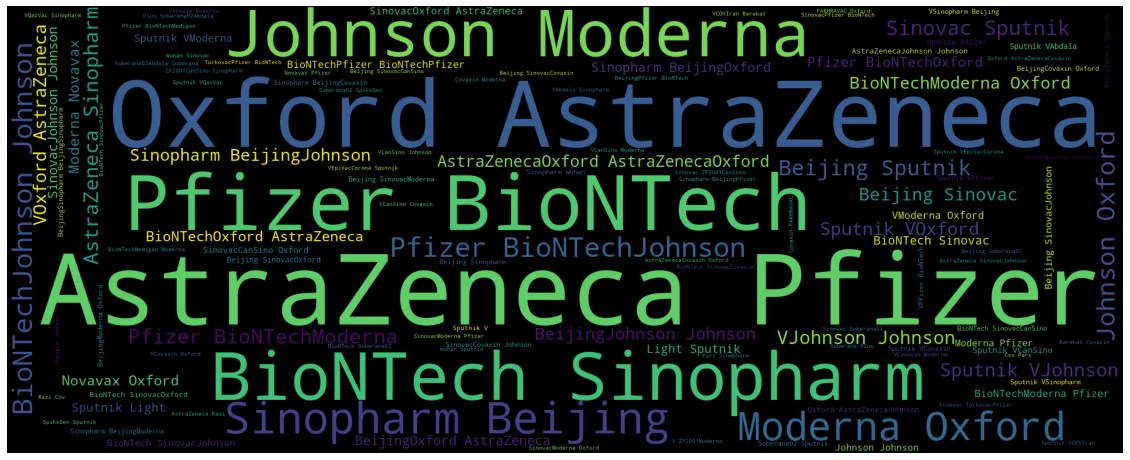
words= "".join(df["vaccines"])

final = WordCloud(width = 2000, height = 800, background\_color ="black",min\_font\_size = 10).generate(words)

plt.imshow(final)

plt.axis("off")

plt.show()

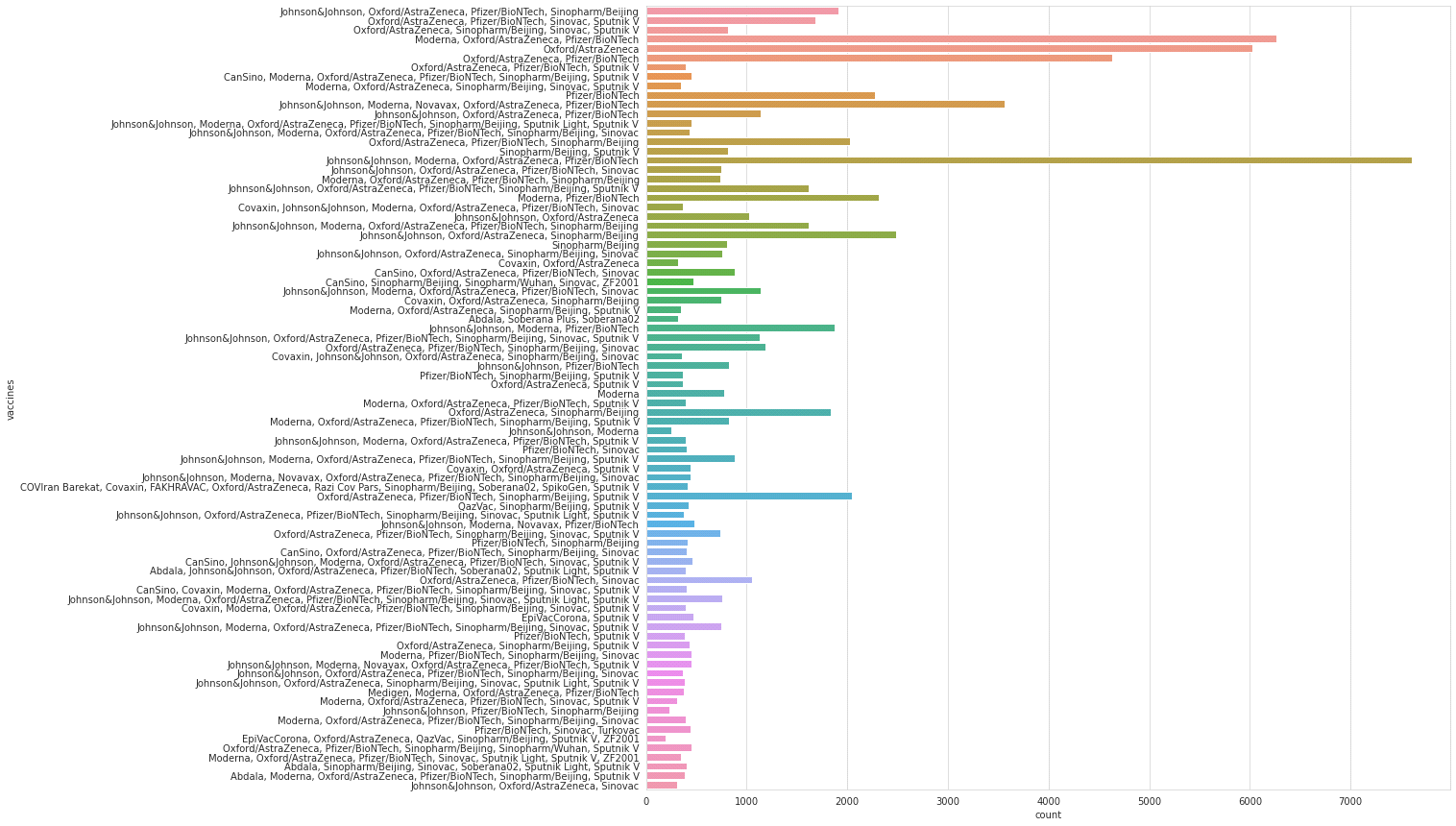


In [16]:

plt.figure(figsize=(15,15))

sns.countplot(y= "vaccines",data= df)

plt.show()



In [17]:

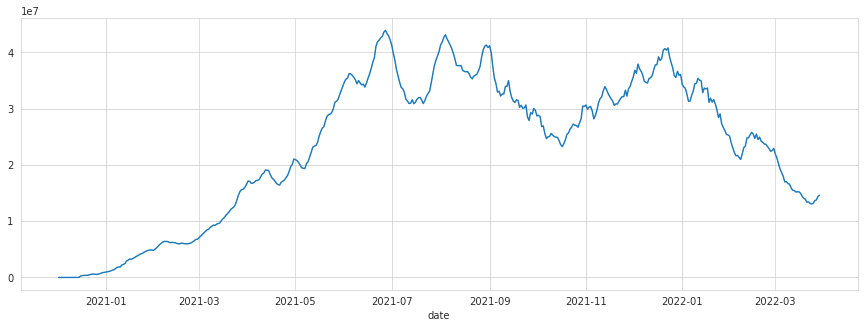
*#daily vaccinations*

x= df.groupby("date").daily\_vaccinations.sum()

plt.figure(figsize= (15,5))

sns.lineplot(x.index,x.values)

plt.show()



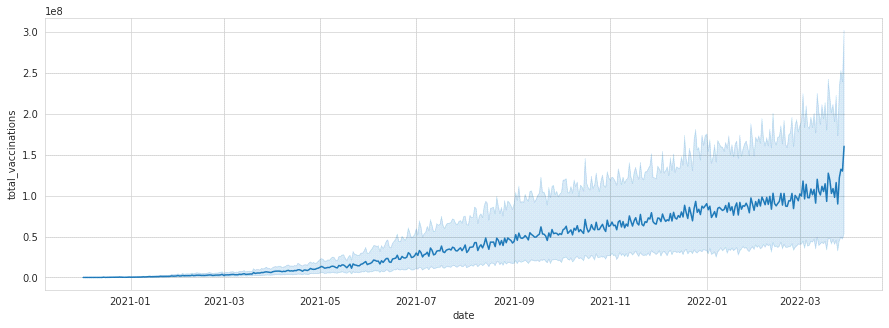
In [18]:

*#total vaccinations*

plt.figure(figsize= (15,5))

sns.lineplot(x= "date",y= "total\_vaccinations",data= df)

plt.show()



In [19]:

*#Countries with best daily average vaccinations*

x= df.groupby("country").daily\_vaccinations.mean().sort\_values(ascending= False).head(20)

x

Out[19]:

country

China 6.930368e+06

India 4.175994e+06

United States 1.191727e+06

Brazil 9.435287e+05

Indonesia 8.462893e+05

Japan 6.215795e+05

Bangladesh 5.453055e+05

Pakistan 5.430051e+05

Vietnam 5.310949e+05

Mexico 4.134253e+05

Germany 3.761575e+05

Philippines 3.665658e+05

Iran 3.535194e+05

Russia 3.480843e+05

Turkey 3.351917e+05

Thailand 3.251471e+05

United Kingdom 3.140841e+05

France 3.104963e+05

South Korea 3.042512e+05

Italy 2.970580e+05

Name: daily\_vaccinations, dtype: float64

In [20]:

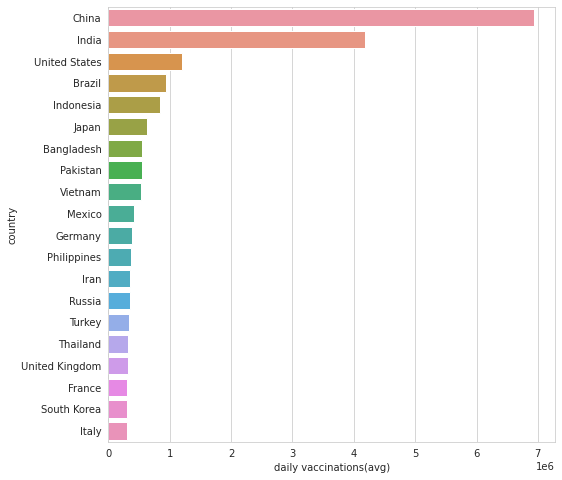
*#daily vaccinations barplot*

plt.figure(figsize= (8,8))

ax= sns.barplot(x.values,x.index)

ax.set\_xlabel("daily vaccinations(avg)")

plt.show()



In [21]:

*#vaccination per hundred top countries*

df["Total\_vaccinations\_per\_hundred"]= df.groupby("country").total\_vaccinations\_per\_hundred.tail(1)

In [22]:

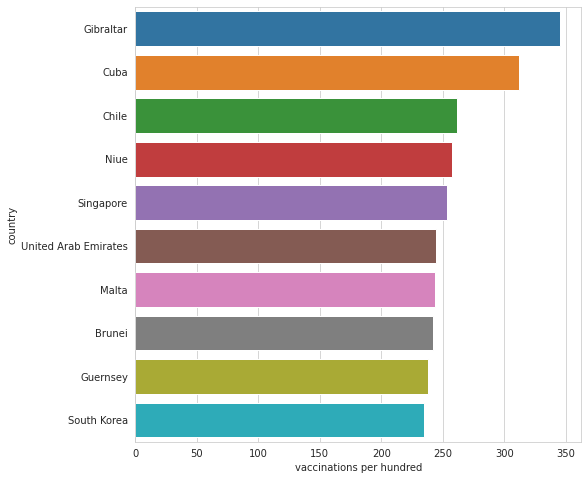
x= df.groupby("country")["Total\_vaccinations\_per\_hundred"].mean().sort\_values(ascending= False).head(10)

plt.figure(figsize= (8,8))

ax= sns.barplot(x.values,x.index)

ax.set\_xlabel("vaccinations per hundred")

plt.show()



In [23]:

*#daily vaccinations per million top countries*

df.groupby("country")["daily\_vaccinations\_per\_million"].mean().sort\_values(ascending= False).head(20)

Out[23]:

country

Falkland Islands 21185.393939

Saint Helena 13915.164835

Tokelau 12718.106195

Pitcairn 10891.797619

Niue 10109.509434

Cuba 9955.943333

Gibraltar 8000.463470

Bonaire Sint Eustatius and Saba 7412.000000

Bhutan 7241.676880

Brunei 6906.782857

Turkmenistan 6618.888889

South Korea 5930.227273

Uruguay 5829.491139

Chile 5764.154525

Singapore 5585.536424

Malta 5553.986207

Taiwan 5545.517426

Guernsey 5437.624113

Australia 5422.241895

Vietnam 5410.000000

Name: daily\_vaccinations\_per\_million, dtype: float64

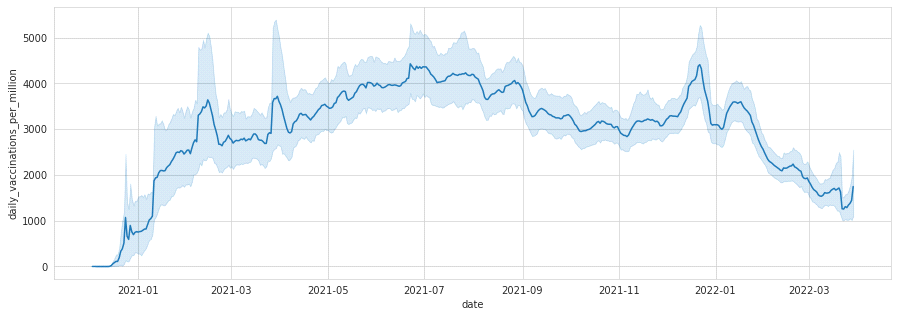
In [24]:

*#daily vaccination per million*

plt.figure(figsize= (15,5))

sns.lineplot(x= "date",y= "daily\_vaccinations\_per\_million",data= df)

plt.show()



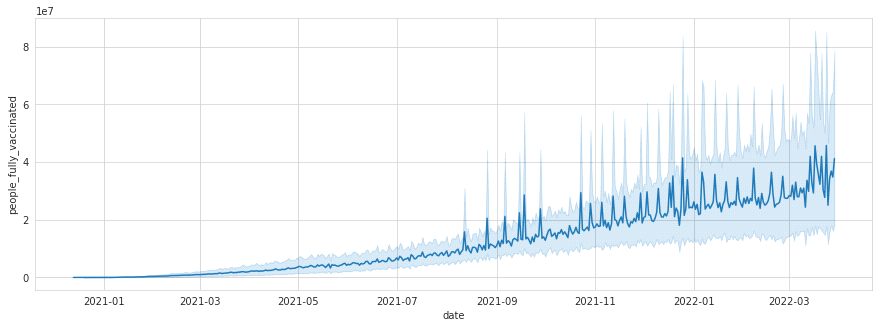
In [25]:

*#people fully vaccinated*

plt.figure(figsize= (15,5))

sns.lineplot(x= "date",y= "people\_fully\_vaccinated",data= df)

plt.show()



In [26]:

*#covid 19 vaccinations INDIA*

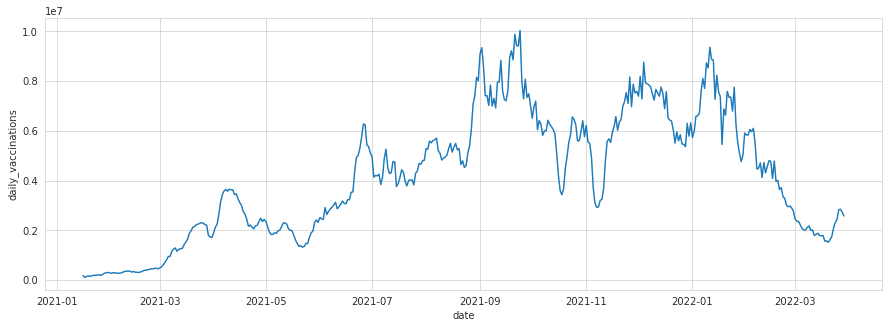
In [27]:

*#daily vaccinations in India*

plt.figure(figsize= (15,5))

sns.lineplot(x= "date",y= "daily\_vaccinations",data= df[df.country== "India"])

plt.show()



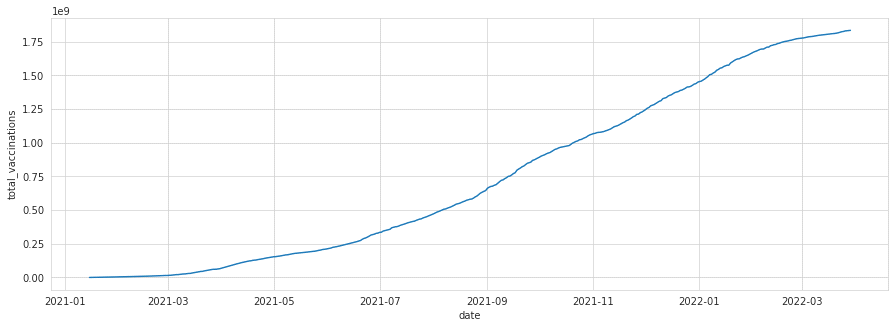
In [28]:

*#Total vaccinations in India*

plt.figure(figsize= (15,5))

sns.lineplot(x= "date",y= "total\_vaccinations",data= df[df["country"]=="India"])

plt.show()



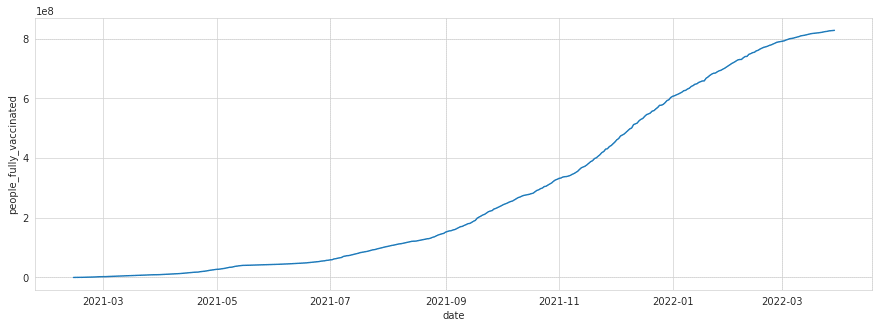
In [29]:

*#full vaccinations in India*

plt.figure(figsize= (15,5))

sns.lineplot(x= "date",y= "people\_fully\_vaccinated",data= df[df["country"]=="India"])

plt.show()



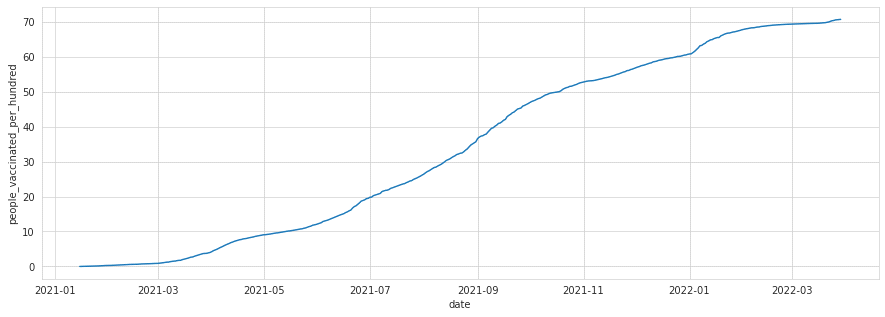
In [30]:

*#people\_vaccinated per hundred in India*

plt.figure(figsize= (15,5))

sns.lineplot(x= "date",y= "people\_vaccinated\_per\_hundred",data= df[df["country"]=="India"])

plt.show()



In [31]:

*#preferred vaccine in India*

x= df[df["country"]=="India"]

z= x.vaccines.value\_counts()

c= list(z.index)

c

Out[31]:

['Covaxin, Oxford/AstraZeneca, Sputnik V']

In [32]:

*#COMPARING TOP 5 COUNTRIES WITH MOST VACCINATIONS*

In [33]:

df.groupby("country")["Total\_vaccinations(count)"].mean().sort\_values(ascending= False).head()

Out[33]:

country

China 3.263129e+09

India 1.834501e+09

United States 5.601818e+08

Brazil 4.135596e+08

Indonesia 3.771089e+08

Name: Total\_vaccinations(count), dtype: float64

In [34]:

*#creating dataframe for top 5 vaccinated countries*

x= df.loc[(df.country== "United States") | (df.country== "China")| (df.country== "India")| (df.country== "Unted Kingdom")|(df.country== "England")]

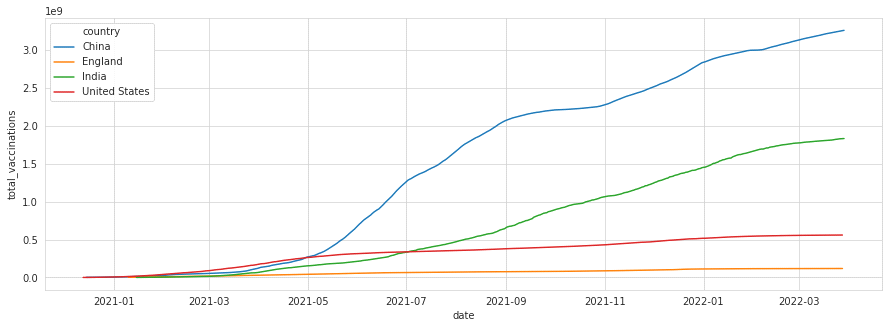
In [35]:

*#total vaccination comparison*

plt.figure(figsize= (15,5))

sns.lineplot(x= "date",y= "total\_vaccinations" ,data= x,hue= "country")

plt.show()



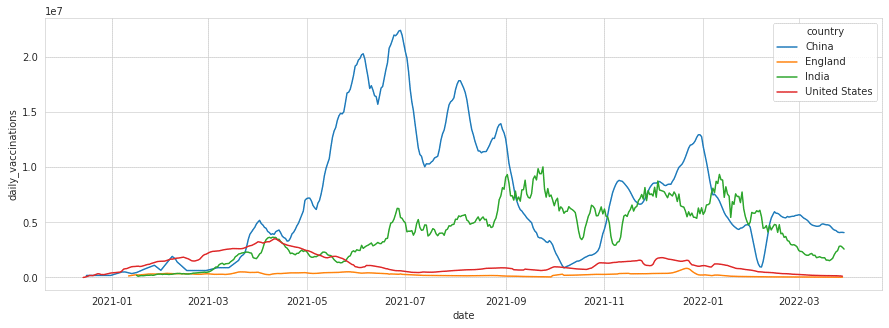
In [36]:

*#daily vaccination comparison*

plt.figure(figsize= (15,5))

sns.lineplot(x= "date",y= "daily\_vaccinations" ,data= x,hue= "country")

plt.show()



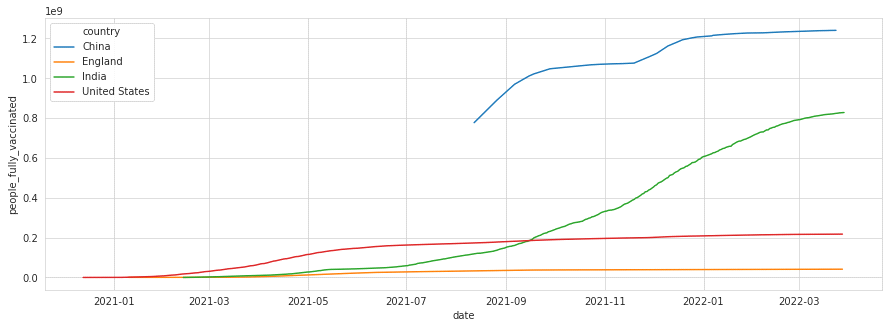
In [37]:

*#full vaccinations comparison*

plt.figure(figsize= (15,5))

sns.lineplot(x= "date",y= "people\_fully\_vaccinated" ,data= x,hue= "country")

plt.show()



In [38]:

*#daily vaccination per million comparison*

plt.figure(figsize= (15,5))

sns.lineplot(x= "date",y= "daily\_vaccinations\_per\_million" ,data= x,hue= "country")

plt.show()

