**COVID Vaccines Analysis**

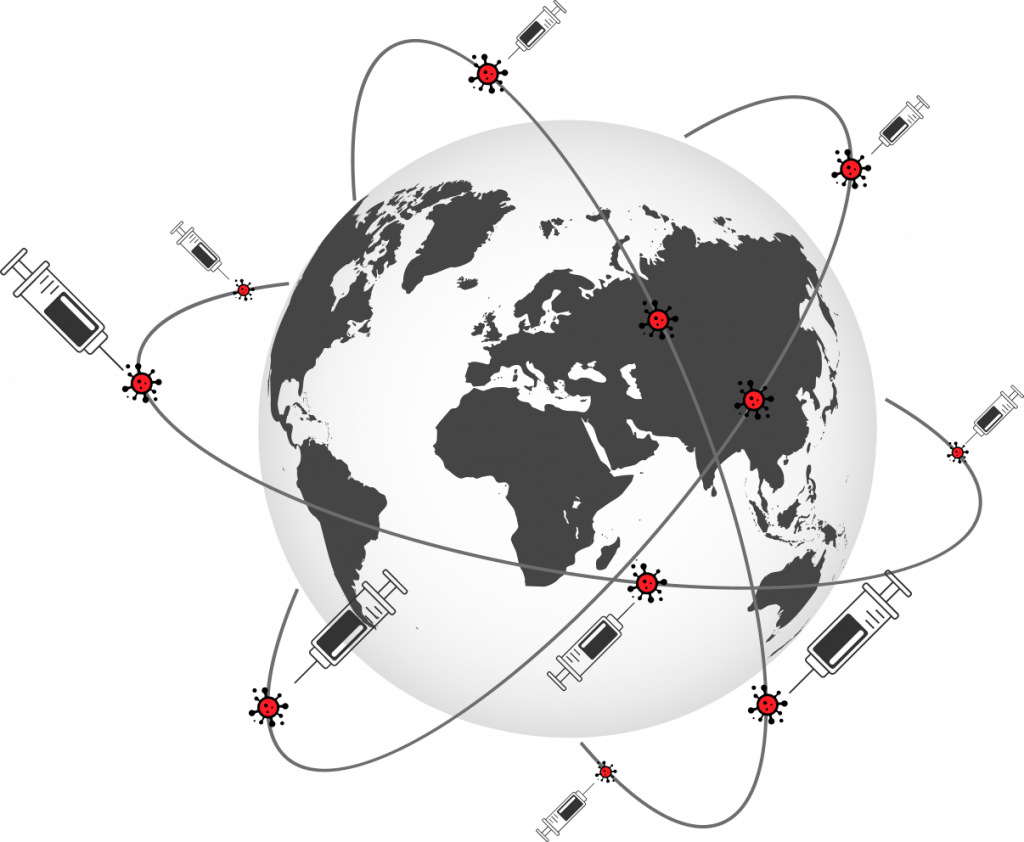
**Phase 3 Submission Document**

**Project Title:** COVID Vaccines Analysis

**Phase 3 : *Development part 1***

**Topic :***Start bulding the COVID Vaccines Analysis model by loading and pre-processing dataset.*

****

****

COVID Vaccines Analysis

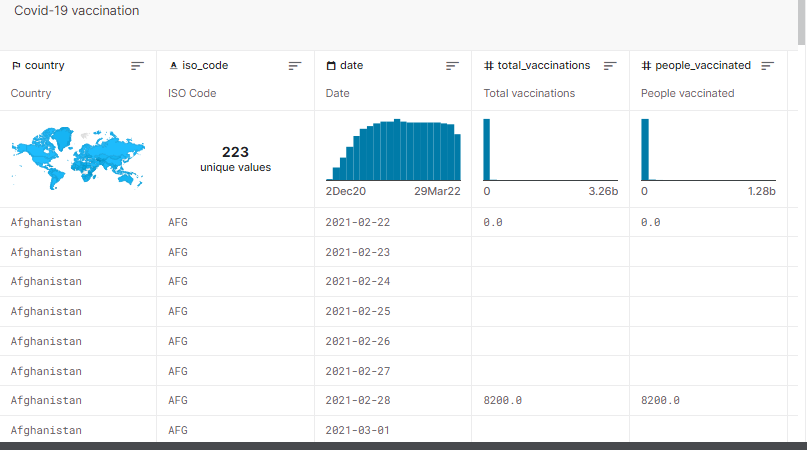
**COVID Vaccines Analysis**

**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.

**Given data set:**

**Dataset Link:**[**https://www.kaggle.com/datasets/gpreda/covid-world-vaccination-progress**](https://www.kaggle.com/datasets/gpreda/covid-world-vaccination-progress)



**1. 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.

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) 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".

## **2. 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.

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+αR�3+�� or lower than Q1−αR�1−�� 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).

## **3. 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.

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")

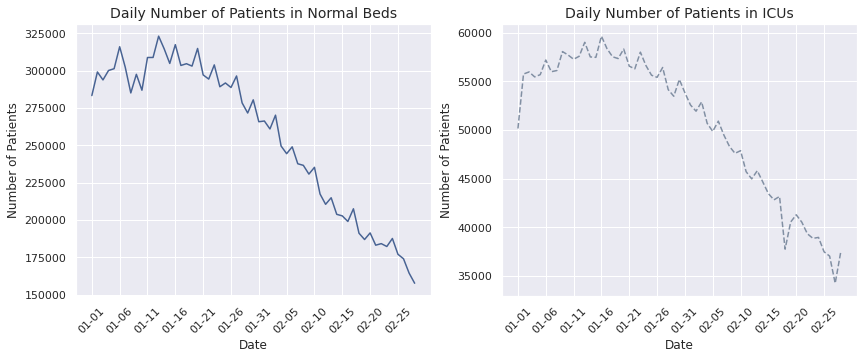
ax.set\_title(f"Daily Number of Patients in **{**where**}**", fontsize=14)

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]:

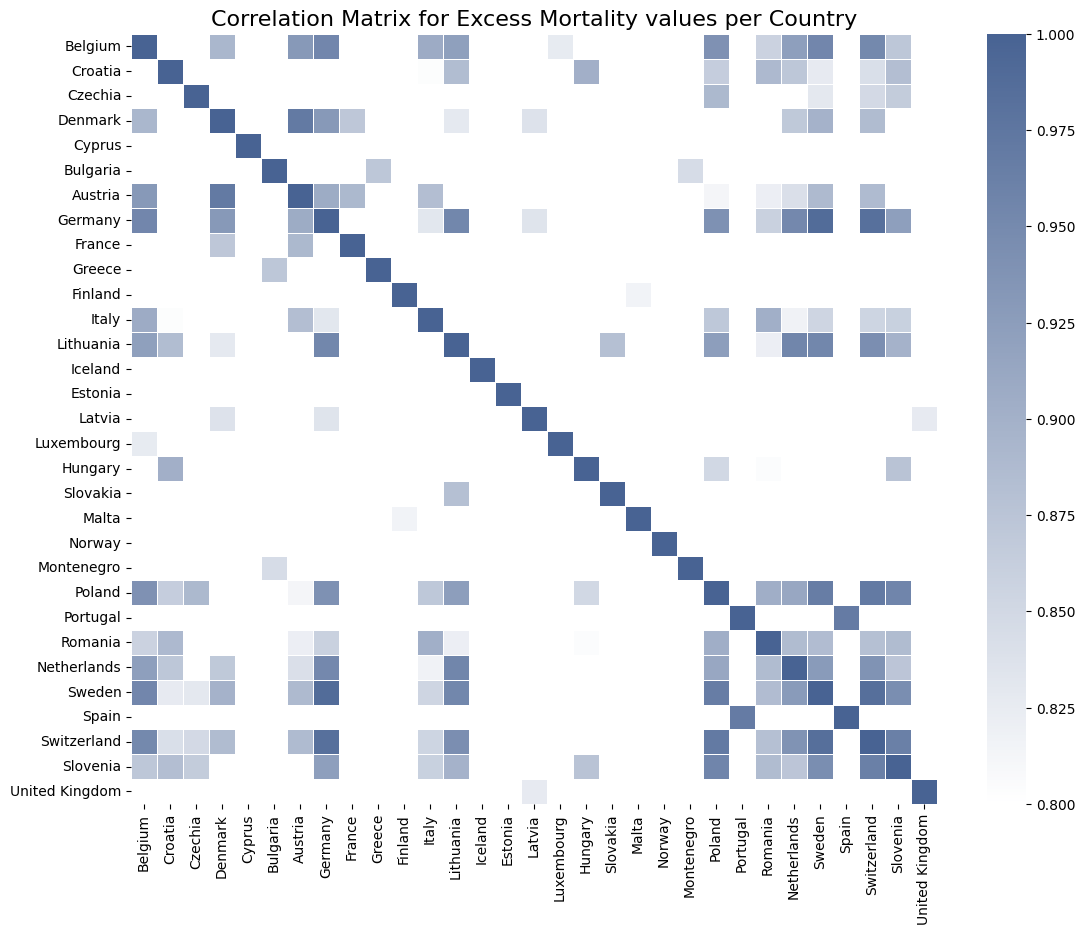
fig, ax = plt.subplots(figsize=(13,10))

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))

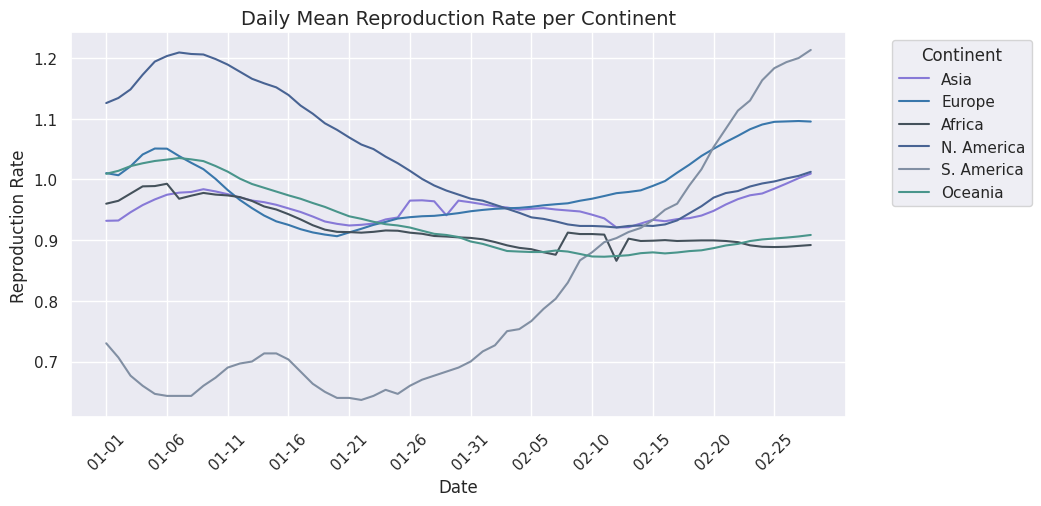
ax.set\_xticks(tick\_marks[::5])

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)

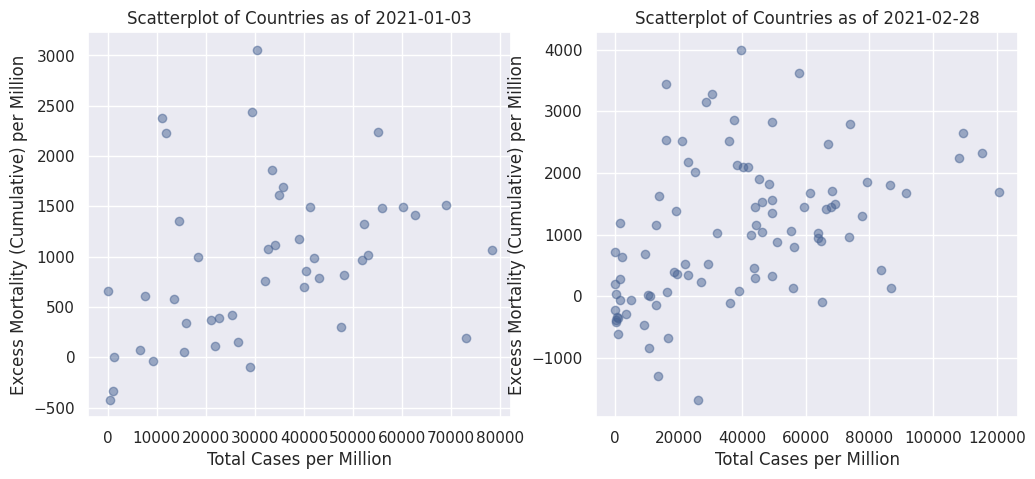
ax.set\_title(f'Scatterplot of Countries as of **{**this\_day**}**')

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')

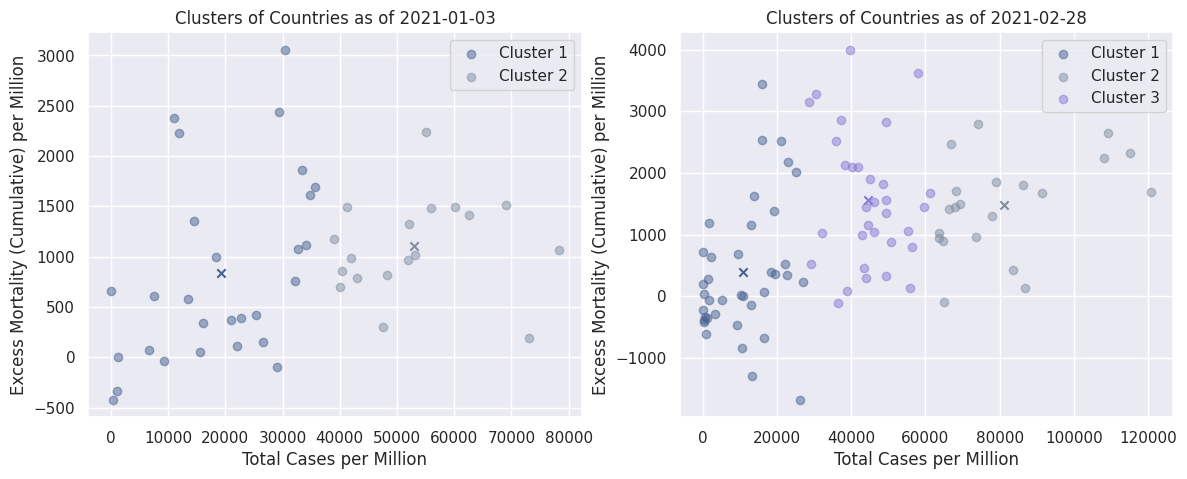
ax.set\_title(f'Clusters of Countries as of **{**this\_day**}**')

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#appendix) 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**}**')

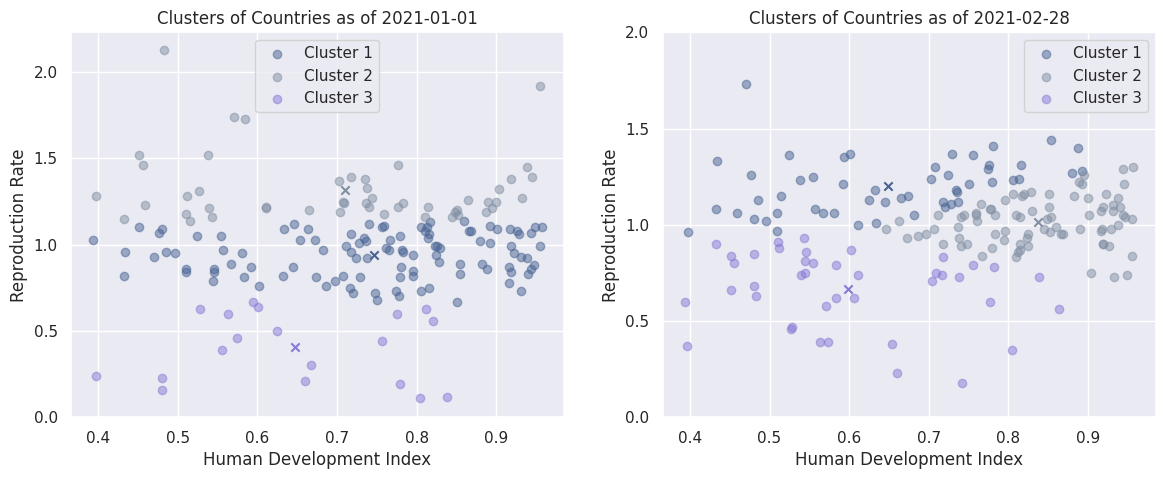
ax.set\_yticks([0.0,0.5,1.0,1.5,2.0])

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.