Final Project – Covid 19 Data Analysis

Group A

Alexandre Santana

João Tiago Dias

Paulo Sousa

Online Program in Big Data Talent Discovery and Upskilling

Data Analytics and Data Science

Instructor:

Vala Rohani

July 2022

Analytics Academy of Data Corner in collaboration with Polytechnique Institute of Setubal

Sponsored by Santander Portuguese Universities

Table of Contents

1 Introduction

According to (Binti Hamzah et al., 2020), COVID-19 outbreak was first reported in Wuhan, China and has spread to more than 50 countries. World Health Organization declared COVID-19 as a Public Health Emergency of International Concern (PHEIC) on 30 January 2020. Naturally, a rising infectious disease involves fast spreading, endangering the health of large numbers of people, and thus requires immediate actions to prevent the disease at the community level. An infectious disease outbreak is the occurrence of a disease that is not usually expected in a particular community, geographical region, or time-period. COVID-19 is caused by a new type of coronavirus which was previously named 2019-nCoV by the World Health Organization (WHO). It is the seventh member of the coronavirus family, together with MERS- nCoV and SARS-nCoV, that can spread to humans. The symptoms of the infection include fever, cough, shortness of breath, and diarrhea. In more severe cases, COVID-19 can cause pneumonia and even death. The incubation period of COVID-19 can last for 2 weeks or longer. During the period of latent infection, the disease may still be infectious (Binti Hamzah et al., 2020). The virus can spread from person to person through respiratory droplets and close contact. Since of its outbreak and expansion all around the world that it becomes even more important of analyzing these epidemiological data concerning the number of cases in each country, deaths, and other relevant information about this pandemic to guide strategies for situational awareness and intervention and predict the risk of infecting people (Dey et al., 2020). With the high demand for appropriate and trustworthy information about 2019-nCoV, organizations are fighting for managing, cleaning, and understanding all the data that arises from multiple sources of Covid-19 information to take the steps that will allow them to have insightful information about the pandemic and to predict and forecast COVID-19 cases, deaths, and recoveries through predictive modelling, and to decipher patterns on pandemic dissemination and at the same time, assess the political and economic impact of the virus spread (Binti Hamzah et al., 2020).

In this report, we present an effort to compile and analyze the epidemiological outbreak information on Covid-19 based on an open dataset on Covid-19 provided by "Our World in Data" which involves the global level data related to Covid-19 for all countries and ranges from 24/2/2020 to 5/3/2022. Feature engineering methods and techniques through Excel and Python were applied to clean the data and prepare it for data analysis. After the data cleaning step an exploratory data analysis with visualizations has been made through

Tableau Desktop to identify the most influential features and create insightful information to begin the necessary evaluation to understand the risks and apply the necessary measures and containment activities to control the Covid-19 pandemic.

2 Dataset Description

The open dataset on Covid-19 file provided by "Our World in Data" comes with the daily information on Covid-19 cases, deaths and recoveries for affected countries and regions, testing rates, people vaccinated, hospital bed occupancy and other relevant information which ranges from 24/2/2020 to 5/3/2022. This is a time series data and so the number of cases on any given day is the cumulative number. This dataset provides an opportunity for the data analyst to bring some insightful information about the pandemic and be a starting point for health and scientific community and governments to cooperate and decide the best path to go to adopt efficient policies and prevent its spread. Below in fig.1 we have a better visualization of all this important data.

iso_code	continent	location	date	total_cases	new_cases	new_cases_smoothed	total_deaths	new_deaths	new_deaths_smoothed	total_cases_per_million	new_cases_per_million	new_cases_smoothed_per_million	total_deaths_per_million	new_deaths_per_million	r
AFG	Asia	Afghanistan	2020-02-24	5.0	5.0					0.126	0.126				
AFG	Asia	Afghanistan	2020-02-25	5.0	0.0					0.126	0.0				
AFG	Asia	Afghanistan	2020-02-26	5.0	0.0					0.126	0.0				
AFG	Asia	Afghanistan	2020-02-27	5.0	0.0					0.126	0.0				
AFG	Asia	Afghanistan	2020-02-28	5.0	0.0					0.126	0.0				
AFG	Asia	Afghanistan	2020-02-29	5.0	0.0					0.126	0.0				
AFG	Asia	Afghanistan	2020-03-01	5.0	0.0	0.714				0.126	0.0	0.018			
AFG	Asia	Afghanistan	2020-03-02	5.0	0.0	0.0				0.126	0.0	0.0			
AFG	Asia	Afghanistan	2020-03-03	5.0	0.0	0.0				0.126	0.0	0.0			
AFG	Asia	Afghanistan	2020-03-04	5.0	0.0	0.0				0.126	0.0	0.0			
AFG	Asia	Afghanistan	2020-03-05	5.0	0.0	0.0				0.126	0.0	0.0			
AFG	Asia	Afghanistan	2020-03-06	5.0	0.0	0.0				0.126	0.0	0.0			
AFG	Asia	Afghanistan	2020-03-07	8.0	3.0	0.429				0.201	0.075	0.011			
AFG	Asia	Afghanistan	2020-03-08	8.0	0.0	0.429				0.201	0.0	0.011			
AFG	Asia	Afghanistan	2020-03-09	8.0	0.0	0.429				0.201	0.0	0.011			
AFG	Asia	Afghanistan	2020-03-10	8.0	0.0	0.429				0.201	0.0	0.011			
AFG	Asia	Afghanistan	2020-03-11	11.0	3.0	0.857				0.276	0.075	0.022			
AFG	Asia	Afghanistan	2020-03-12	11.0	0.0	0.857				0.276	0.0	0.022			
AFG	Asia	Afghanistan	2020-03-13	11.0	0.0	0.857				0.276	0.0	0.022			
AFG	Asia	Afghanistan	2020-03-14	14.0	3.0	0.857				0.351	0.075	0.022			
AFG	Asia	Afghanistan	2020-03-15	20.0	6.0	1.714				0.502	0.151	0.043			
AFG	Asia	Afghanistan	2020-03-16	25.0	5.0	2.429				0.628	0.126	0.061			
AFG	Asia	Afghanistan	2020-03-17	26.0	1.0	2.571				0.653	0.025	0.065			
AFG	Asia	Afghanistan	2020-03-18	26.0	0.0	2.143				0.653	0.0	0.054			
AFG	Asia	Afghanistan	2020-03-19	26.0	0.0	2.143				0.653	0.0	0.054			
AFG	Asia	Afghanistan	2020-03-20	24.0						0.602					
AFG	Asia	Afghanistan	2020-03-21	24.0	0.0					0.602	0.0				
AFG	Asia	Afghanistan	2020-03-22	34.0	10.0					0.854	0.251				
AFG	Asia	Afghanistan	2020-03-23	40.0	6.0		1.0	1.0		1.004	0.151		0.025	0.025	
AFG	Asia	Afghanistan	2020-03-24	42.0	2.0		1.0	0.0		1.054	0.05		0.025	0.0	

Figure 1 - Open dataset "Our World in Data" - Raw File

3 Purpose of the analysis

After analyzing the Covid-19 data raw file and finding some inaccuracies and features with no data or scarce data that could lead us to insufficient information or inadequate conclusions we decided to proceed to a reliable and specific analysis that will allow us to reach the main goals that we are looking for with this analysis. As so, we centered our analysis on the following questions that we wanted to be satisfied:

Q1 - *How have Covid cases and deaths evolved in the world and which continents have reached the highest numbers?*

Q2 - *Which are the Top 10 countries with the most cases and most deaths per million inhabitants?*

Q3 - *How did the evolution of positive cases affect the number of patients in hospitals in Europe (EU)?*

Q4 - *In Europe (EU), what is the correlation between GDP Per Capita vs New Cases and also Population Density vs Cases per Million?*

Q5 - *What is the influence of vaccination on new cases and deaths in Portugal?*

Q6 - *How was the evolution of new cases and deaths in Portugal and how can we predict the future?*

4 Data Cleaning

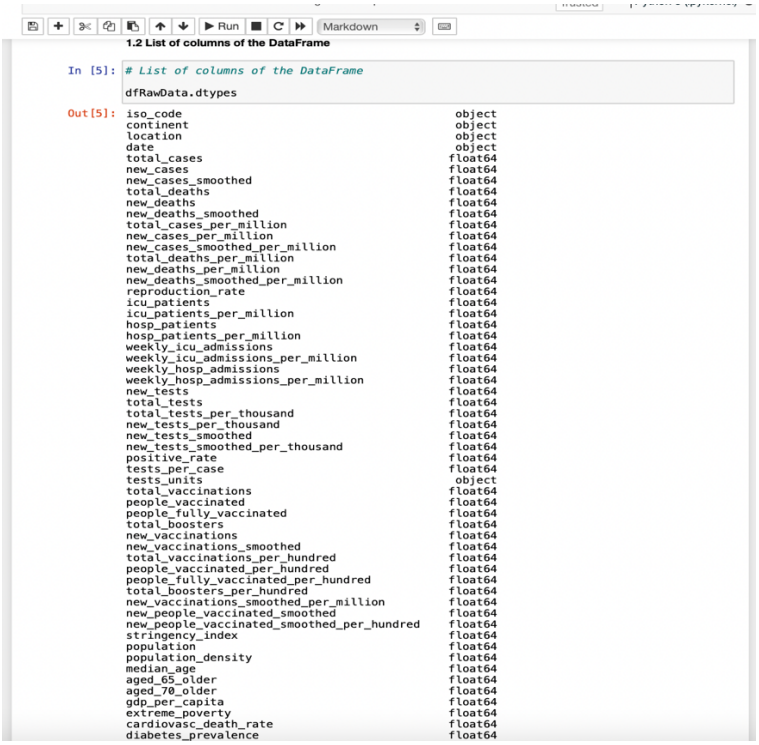
According to Christine Chai (Chai, 2020), data scientists spend a lot of their work time on cleaning and organizing data, leaving little time for data analysis. This importance of cleaning and preparing data is one of the most important steps in managing and leading with data, especially when emerging analysis reveals additional issues that must be resolved. Accurate and reliable data must be a priority on data analysis. Data analysis tools are powerful in business, but businesses need data to be cleaned appropriately before they can produce valid outputs. Otherwise, the whole data pipeline becomes inaccurate, and the result would not be as useful as a business team expect (Chai, 2020). Raw data is never perfect because data errors or missing values are inevitable and may lead to incorrect answers and understandings. Fortunately, many of these errors can be corrected through featured engineering methods and powerful analytical tools such as Excel or Python programming. As so, we have used these tools, especially Python programming to organize and proceed to the cleaning step of the raw dataset of Covid-19 that was given. This was one important step for the analytical process because it helped us to increase overall productivity and allow for the highest quality information in our decision making by handling null values and outliers, duplicate records, organizing data and choosing the most appropriated and influential features concerning our Covid-19 data analysis.

As so, to prepare and clean the data we first loaded the raw data file (covid-data.csv) into Jupyter Notebook. Then we isolated the columns containing the data necessary to answer

the previously defined questions. After we proceeded with data cleaning using various techniques. This entire process with the python programming code can be found in detail in the appendix section.

4.1 Preparing and cleaning of the Covid-19 dataset

Using Jupyter Notebook we started importing the required libraries and loading the provided dataset. To verify the dimensionality of the dataset we listed the features in columns (dfRawData (166326, 67)), as we can see below.



1.2 List of columns of the DataFrame

```
In [5]: # List of columns of the DataFrame

        dfRawData.dtypes

Out[5]: iso_code                                  object
        continent                                 object
        location                                  object
        date                                      object
        total_cases                              float64
        new_cases                                float64
        new_cases_smoothed                       float64
        total_deaths                             float64
        new_deaths                               float64
        new_deaths_smoothed                      float64
        total_cases_per_million                  float64
        new_cases_per_million                    float64
        new_cases_smoothed_per_million           float64
        total_deaths_per_million                 float64
        new_deaths_per_million                   float64
        new_deaths_smoothed_per_million          float64
        reproduction_rate                        float64
        icu_patients                             float64
        icu_patients_per_million                 float64
        hosp_patients                            float64
        hosp_patients_per_million                float64
        weekly_icu_admissions                    float64
        weekly_icu_admissions_per_million        float64
        weekly_hosp_admissions                   float64
        weekly_hosp_admissions_per_million       float64
        new_tests                                float64
        total_tests                              float64
        total_tests_per_thousand                 float64
        new_tests_per_thousand                   float64
        new_tests_smoothed                       float64
        new_tests_smoothed_per_thousand          float64
        positive_rate                            float64
        tests_per_case                           float64
        tests_units                               object
        total_vaccinations                       float64
        people_vaccinated                        float64
        people_fully_vaccinated                  float64
        total_boosters                           float64
        new_vaccinations                         float64
        new_vaccinations_smoothed                float64
        total_vaccinations_per_hundred           float64
        people_vaccinated_per_hundred            float64
        people_fully_vaccinated_per_hundred      float64
        total_boosters_per_hundred               float64
        new_vaccinations_smoothed_per_million    float64
        new_people_vaccinated_smoothed           float64
        new_people_vaccinated_smoothed_per_hundred float64
        stringency_index                         float64
        population                               float64
        population_density                       float64
        median_age                               float64
        aged_65_older                            float64
        aged_70_older                            float64
        gdp_per_capita                           float64
        extreme_poverty                          float64
        cardiovasc_death_rate                    float64
        diabetes_prevalence                      float64
```

Figure 2 - List of columns of the data frame

Then we proceeded to the choosing of the data for the scope of our analysis by choosing the columns that are useful for our purpose in answering the questions that we are looking for.

2.2 Choosing the columns that are useful for the scope of our analysis

```
#-- Choosing the columns that are useful for the scope of our analysis
selected_col = [
    #Geography - Where and when
    'iso_code', 'continent', 'location', 'date',

    #Covid cases per million - How many
    'total_cases', 'total_cases_per_million', 'total_deaths', 'total_deaths_per_million',

    #Economy and People
    'gdp_per_capita', 'population', 'population_density',

    #Medical Facilites
    'hospital_beds_per_thousand', 'life_expectancy'
]
```

Figure 3 – Choosing the desired features for our analysis

In our analysis we will use 13 out of 67 columns (features). Then a new data frame with the same values was created but with the desired columns ((166326, 13).

2.4 List of columns of the new DataFrame

```
# List of columns of the DataFrame
dfToClean.dtypes
```

```
iso_code                      object
continent                     object
location                      object
date                          object
total_cases                  float64
total_cases_per_million      float64
total_deaths                 float64
total_deaths_per_million     float64
gdp_per_capita               float64
population                   float64
population_density           float64
hospital_beds_per_thousand   float64
life_expectancy              float64
dtype: object
```

Figure 4- Columns of the new data frame for our analysis

2.6 Display the first 5 rows of the new DataFrame

```
# The first 5 rows of the DataFrame
dfToClean.head(5)
```

	iso_code	continent	location	date	total_cases	total_cases_per_million	total_deaths	total_deaths_per_million	gdp_per_capita	population	population_c
0	AFG	Asia	Afghanistan	2020-02-24	5.0	0.126	NaN	NaN	1803.987	39835428.0	
1	AFG	Asia	Afghanistan	2020-02-25	5.0	0.126	NaN	NaN	1803.987	39835428.0	
2	AFG	Asia	Afghanistan	2020-02-26	5.0	0.126	NaN	NaN	1803.987	39835428.0	
3	AFG	Asia	Afghanistan	2020-02-27	5.0	0.126	NaN	NaN	1803.987	39835428.0	
4	AFG	Asia	Afghanistan	2020-02-28	5.0	0.126	NaN	NaN	1803.987	39835428.0	

Figure 5 - Example of the new data frame

After preparing this new dataset we proceeded to one of the most important steps of our analysis that was starting to clean the data and correcting the inaccuracies to have an understandable and reliable data frame that allow us to have correct and insightful information about the questions we want to answer with this analysis.

In the following step have started with the cleaning of World data. We started by handling with the null values:

3.1 Listing the columns with null values

```
# Listing the columns with null values

dfToClean.apply(lambda x: sum(x.isnull()),axis=0)
```

```
iso_code                        0
continent                    9956
location                        0
date                            0
total_cases                  3033
total_cases_per_million      3791
total_deaths                20875
total_deaths_per_million    21620
gdp_per_capita              27822
population                   1075
population_density          18398
hospital_beds_per_thousand  42662
life_expectancy             11058
dtype: int64
```

Figure 6 - Listing the columns with null values

We noticed that there were several records with an iso-code starting with OWID. The records are not from individual countries, so they are out of the scope of our analysis. As so, we remove the records with "iso-code" containing "OWID".

3.5 Verify that all records with 'iso_code' containing 'OWID' were removed

```
# 'iso_code' contains 'OWID'
dfToClean[dfToClean['iso_code'].str.contains('OWID')]
```

iso_code	continent	location	date	total_cases	total_cases_per_million	total_deaths	total_deaths_per_million	gdp_per_capita	population	population_density

Figure 7 -Verifying that all records with 'iso_code' containing 'OWID' were removed

By listing the columns with null values again, we have:

3.6 Listing the columns with null values

```
# Listing the columns with null values

dfToClean.apply(lambda x: sum(x.isnull()),axis=0)
```

```
iso_code                        0
continent                       0
location                        0
date                            0
total_cases                  2709
total_cases_per_million      2709
total_deaths                20336
total_deaths_per_million    20336
gdp_per_capita              18323
population                      0
population_density           8899
hospital_beds_per_thousand  32441
life_expectancy               837
dtype: int64
```

Figure 8 - Listing the columns with null values

As the dataset was imported from a csv file all blank cells were imported as null value (NaN). Handling with the null values we started by replacing it with zero in "total_cases" and "total_deaths":

3.8 Replace null values with zero in 'total_cases' ¶

```
# Replace NaN with zero in 'total_cases'
dfToClean['total_cases'] = dfToClean['total_cases'].replace(np.nan, 0)

# Replace NaN with zero in 'total_cases_per_million'
dfToClean['total_cases_per_million'] = dfToClean['total_cases_per_million'].replace(np.nan, 0)

# Replace NaN with zero in 'total_deaths'
dfToClean['total_deaths'] = dfToClean['total_deaths'].replace(np.nan, 0)

# Replace NaN with zero in 'total_deaths_per_million'
dfToClean['total_deaths_per_million'] = dfToClean['total_deaths_per_million'].replace(np.nan, 0)
```

Figure 9 - Replacing null values with zero

Some “gdp_per_capita” have null values as we can see below:

3.10 Showing records with null values in 'gdp_per_capita'

```
# 'gdp_per_capita' isnull

dfToClean[dfToClean['gdp_per_capita'].isnull()]
```

	iso_code	continent	location	date	total_cases	total_cases_per_million	total_deaths	total_deaths_per_million	gdp_per_capita	population	populatior
2973	AND	Europe	Andorra	2020-03-02	1.0	12.928	0.0	0.000	NaN	77354.0	
2974	AND	Europe	Andorra	2020-03-03	1.0	12.928	0.0	0.000	NaN	77354.0	
2975	AND	Europe	Andorra	2020-03-04	1.0	12.928	0.0	0.000	NaN	77354.0	
2976	AND	Europe	Andorra	2020-03-05	1.0	12.928	0.0	0.000	NaN	77354.0	
2977	AND	Europe	Andorra	2020-03-06	1.0	12.928	0.0	0.000	NaN	77354.0	
...	...	...	...	...	...	...	...	...	...	...	
163418	WLF	Oceania	Wallis and Futuna	2022-01-01	454.0	40923.021	7.0	630.972	NaN	11094.0	
163419	WLF	Oceania	Wallis and Futuna	2022-03-02	454.0	40923.021	7.0	630.972	NaN	11094.0	

Figure 10 -Listing "gdp_per_capita" with null values

The “gdp_per_capita” values are missing, and these values cannot be replaced with zero.

We started listing all the countries where “gdp_per_capita” isnull.

3.11 List the locations where 'gdp_per_capita' isnull

```
# create a new dataframe where all records where 'gdp_per_capita' isnull
gdpIsNull = dfToClean[dfToClean['gdp_per_capita'].isnull()]

# Remove records where 'location' have duplicate values, keep the first one
gdpIsNull = gdpIsNull.drop_duplicates(subset='location', keep='first')

# List of 'location' with missing 'gdp_per_capita'
gdpIsNull
```

	iso_code	continent	location	date	total_cases	total_cases_per_million	total_deaths	total_deaths_per_million	gdp_per_capita	population	popul
2973	AND	Europe	Andorra	2020-03-02	1.0	12.928	0.0	0.0	NaN	77354.0	
4423	AIA	North America	Anguilla	2020-03-28	2.0	132.231	0.0	0.0	NaN	15125.0	
19169	BES	North America	Bonaire Sint Eustatius and Saba	2020-04-02	2.0	75.629	0.0	0.0	NaN	26445.0	
22048	VGB	North America	British Virgin Islands	2020-03-28	2.0	65.740	0.0	0.0	NaN	30423.0	
34433	COK	Oceania	Cook Islands	2021-05-21	1.0	56.909	0.0	0.0	NaN	17572.0	
36917	CUB	North America	Cuba	2020-03-12	3.0	0.265	0.0	0.0	NaN	11317498.0	
37641	CUW	North America	Curacao	2020-03-14	1.0	6.068	0.0	0.0	NaN	164796.0	
50873	FRO	Europe	Faeroe Islands	2020-03-04	1.0	20.386	0.0	0.0	NaN	49053.0	

Figure 11 - Listing the locations where gdp_per_capita isnull

Then we replaced null values in 'gdp_per_capita' for locations classified as independent countries but we did a manual update in each value as there was no table to load and do it automatically. After doing this we proceeded to the removal of the records with null values in 'gdp_per_capita' for locations not classified as independent countries (python programming code found in detail in the appendix section).

3.14 Listing the columns with null values

```
# Listing the columns with null values
dfToClean.apply(lambda x: sum(x.isnull()),axis=0)
```

```
iso_code                        0
continent                       0
location                        0
date                            0
total_cases                     0
total_cases_per_million         0
total_deaths                    0
total_deaths_per_million        0
gdp_per_capita                  0
population                      0
population_density           2194
hospital_beds_per_thousand  19258
life_expectancy                 0
dtype: int64
```

Figure 12 - Listing the columns with null values

After this step we proceeded to the replacing of null values in “population_density”. We updated manually each value as there was no table to load and do it automatically (python programming code found in detail in the appendix section).

3.16 List the locations where 'population_density' isnull

```
# create a new dataframe where all records where 'population_density' isnull
popDensIsNull = dfToClean[dfToClean['population_density'].isnull()]

# Remove records where 'location' have duplicate values, keep the first one
popDensIsNull = popDensIsNull.drop_duplicates(subset='location', keep='first')

popDensIsNull
```

le	continent	location	date	total_cases	total_cases_per_million	total_deaths	total_deaths_per_million	gdp_per_capita	population	population_density	hospit
D	Africa	South Sudan	2020-04-05	1.0	0.088	0.0	0.0	1569.888	11381377.0	NaN	
R	Asia	Syria	2020-03-22	1.0	0.055	0.0	0.0	4684.720	18275704.0	NaN	
N	Asia	Taiwan	2020-01-16	0.0	0.000	0.0	0.0	24502.000	23855008.0	NaN	

Figure 13 - Listing locations where "population_density" isnull

3.18 Listing the columns with null values

```
# Listing the columns with null values
dfToClean.apply(lambda x: sum(x.isnull()),axis=0)
```

```
iso_code                        0
continent                       0
location                        0
date                            0
total_cases                     0
total_cases_per_million         0
total_deaths                    0
total_deaths_per_million        0
gdp_per_capita                  0
population                      0
population_density              0
hospital_beds_per_thousand  19258
life_expectancy                 0
dtype: int64
```

Figure 14 - Listing the columns with null values

After this step we proceeded to the replacing of null values in "hospital_beds_per_thousand". We updated manually each value as there was no table to load and do it automatically (python programming code found in detail in the appendix section).

3.19 List the locations where 'hospital_beds_per_thousand' isnull

```
# create a new dataframe where all records where 'population_density' isnull
hospBedsIsNull = dfToClean[dfToClean['hospital_beds_per_thousand'].isnull()]

# Remove records where 'location' have duplicate values, keep the first one
hospBedsIsNull = hospBedsIsNull.drop_duplicates(subset='location', keep='first')

hospBedsIsNull
```

otal_cases	total_cases_per_million	total_deaths	total_deaths_per_million	gdp_per_capita	population	population_density	hospital_beds_per_thousand	life_exp
1.0	12.928	0.0	0.0	49900.000	77354.0	163.755	NaN	
1.0	0.029	0.0	0.0	5819.495	33933611.0	23.890	NaN	
2.0	18.658	0.0	0.0	35973.781	107195.0	584.800	NaN	
2.0	32.210	0.0	0.0	50669.315	62092.0	1308.820	NaN	
1.0	15.038	0.0	0.0	49903.029	66498.0	256.496	NaN	
1.0	0.059	0.0	0.0	1768.153	16914985.0	11.833	NaN	
1.0	0.177	0.0	0.0	4881.406	5657017.0	15.405	NaN	

Figure 15 - Listing locations where "hospital_beds_per_thousand" isnull

3.22 Listing the columns with null values

```
# Listing the columns with null values
dfToClean.apply(lambda x: sum(x.isnull()),axis=0)
```

```
iso_code                    0
continent                   0
location                    0
date                        0
total_cases                 0
total_cases_per_million     0
total_deaths                0
total_deaths_per_million    0
gdp_per_capita              0
population                  0
population_density          0
hospital_beds_per_thousand  0
life_expectancy             0
dtype: int64
```

Figure 16 - Listing the columns with null values

After this step of data cleaning, we then proceeded to the removal of duplicates. First, we checked for duplicates.

3.23 Check for duplicates

```
# Copy data to a new dataframe and check for duplicates
duplicate = dfToClean[dfToClean.duplicated()]
print("Duplicate Rows :")
duplicate
```

```
Duplicate Rows :
```

iso_code	continent	location	date	total_cases	total_cases_per_million	total_deaths	total_deaths_per_million	gdp_per_capita	population	population_density

```
duplicate.shape
```

```
(0, 13)
```

Figure 17 - Checking duplicates

As we can see, there were no duplicates in this data frame. Then we proceeded to another important step of data cleaning that is handling with the outliers. So, we checked if there were any outliers.

3.24 Check for Outliers

```
dfToClean.describe()
```

	total_cases	total_cases_per_million	total_deaths	total_deaths_per_million	gdp_per_capita	population	population_density	hospital_beds_per_thou
count	1.411300e+05	141130.000000	141130.000000	141130.000000	141130.000000	1.411300e+05	141130.000000	141130.00
mean	6.995308e+05	28341.368170	14208.632417	451.435730	20741.584535	4.189247e+07	461.881418	2.92
std	3.332693e+06	50641.416283	58165.774583	767.965459	23334.161775	1.524558e+08	2183.108269	2.40
min	0.000000e+00	0.000000	0.000000	0.000000	661.240000	1.087300e+04	1.980000	0.10
25%	2.728000e+03	513.592500	42.000000	7.853000	4449.898000	2.159067e+06	36.253000	1.20
50%	2.984200e+04	4188.705000	502.000000	72.275000	13111.214000	9.291000e+06	87.176000	2.30
75%	2.685572e+05	35525.021000	4779.000000	581.969000	27936.896000	3.049064e+07	212.865000	4.00
max	7.926573e+07	496858.598000	958437.000000	6322.263000	139100.000000	1.444216e+09	20546.766000	13.80

Figure 18 - Checking outliers

After all these steps of data cleaning we can consider that the data is clean and suitable for analysis.

3.25 We now have clean data

```
# copy the clean data to a new DataFrame
dfClean = dfToClean
```

```
# concise summary of the DataFrame
dfClean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 141130 entries, 0 to 166325
Data columns (total 13 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   iso_code                    141130 non-null  object
 1   continent                   141130 non-null  object
 2   location                    141130 non-null  object
 3   date                        141130 non-null  datetime64[ns]
 4   total_cases                 141130 non-null  float64
 5   total_cases_per_million     141130 non-null  float64
 6   total_deaths                141130 non-null  float64
 7   total_deaths_per_million    141130 non-null  float64
 8   gdp_per_capita              141130 non-null  float64
 9   population                  141130 non-null  float64
 10  population_density          141130 non-null  float64
 11  hospital_beds_per_thousand  141130 non-null  float64
 12  life_expectancy             141130 non-null  float64
dtypes: datetime64[ns](1), float64(9), object(3)
memory usage: 15.1+ MB
```

Figure 19 - Showing the clean data frame

Another part of our analysis focuses on Europe. So, we selected the specific data related to Europe from our raw data file by removing all continents except Europe and then proceeded to the selection of European Union Countries, by dropping out the ones that don t belong to it.

```
# List of countries
# create a new dataframe
countries = dfEuropeToClean

# Remove records where 'location' have duplicate values, keep the first one
countries = countries.drop_duplicates(subset='location', keep='first')

countries
```

				02-25							
14066	BLR	Europe	Belarus	2020-02-28	1.0	1.0	NaN	NaN	NaN	NaN	...
14803	BEL	Europe	Belgium	2020-02-04	1.0	1.0	NaN	NaN	NaN	NaN	...
19872	BIH	Europe	Bosnia and Herzegovina	2020-03-05	2.0	2.0	NaN	NaN	NaN	NaN	...
23483	BGR	Europe	Bulgaria	2020-03-08	4.0	4.0	NaN	NaN	NaN	NaN	...
36177	HRV	Europe	Croatia	2020-02-25	1.0	1.0	NaN	NaN	NaN	NaN	...
38363	CYP	Europe	Cyprus	2020-03-08	NaN	NaN	NaN	NaN	NaN	NaN	...
39091	CZE	Europe	Czechia	2020-03-01	3.0	3.0	NaN	NaN	NaN	NaN	...
40551	DNK	Europe	Denmark	2020-	NaN	NaN	NaN	NaN	NaN	NaN	

Figure 20 - Selection of Europe data

```
# List of countries
# create a new dataframe
countries2 = dfEuropeToClean

# Remove records where 'location' have duplicate values, keep the first one
countries2 = countries2.drop_duplicates(subset='location', keep='first')

countries2

# It should list only all 27 EU countries
```

	iso_code	continent	location	date	total_cases	new_cases	new_cases_smoothed	total_deaths	new_deaths	new_deaths_smoothed	...	fema
9678	AUT	Europe	Austria	2020-02-25	2.0	2.0	NaN	NaN	NaN	NaN	...	
14803	BEL	Europe	Belgium	2020-02-04	1.0	1.0	NaN	NaN	NaN	NaN	...	
23483	BGR	Europe	Bulgaria	2020-03-08	4.0	4.0	NaN	NaN	NaN	NaN	...	
36177	HRV	Europe	Croatia	2020-02-25	1.0	1.0	NaN	NaN	NaN	NaN	...	
38363	CYP	Europe	Cyprus	2020-03-08	NaN	NaN	NaN	NaN	NaN	NaN	...	
39091	CZE	Europe	Czechia	2020-03-01	3.0	3.0	NaN	NaN	NaN	NaN	...	
40551	DNK	Europe	Denmark	2020-02-02	NaN	NaN	NaN	NaN	NaN	NaN	...	
47120	EST	Europe	Estonia	2020-01-06	NaN	NaN	NaN	NaN	NaN	NaN	...	

Figure 21 - List of selected European Union countries

Then we removed the columns with the data we don't use in the analysis and proceeded to the cleaning of it by handling with the null values, outliers, and duplicates of it with the same proceedings we did with world data (python programming code found in detail in the appendix section). Below we can see the summary of the cleaned data frame for Europe.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20153 entries, 9678 to 144692
Data columns (total 34 columns):
 #   Column                                      Non-Null Count  Dtype
---  ------                                      --------------  -----
 0   iso_code                                    20153 non-null  object
 1   location                                    20153 non-null  object
 2   date                                        20153 non-null  datetime64[ns]
 3   total_cases                                 20153 non-null  float64
 4   new_cases_smoothed                          20153 non-null  float64
 5   total_deaths                                20153 non-null  float64
 6   new_deaths_smoothed                         20153 non-null  float64
 7   total_cases_per_million                     20153 non-null  float64
 8   new_cases_smoothed_per_million              20153 non-null  float64
 9   total_deaths_per_million                    20153 non-null  float64
 10  new_deaths_smoothed_per_million             20153 non-null  float64
 11  reproduction_rate                           20153 non-null  float64
 12  icu_patients                                20153 non-null  float64
 13  icu_patients_per_million                    20153 non-null  float64
 14  hosp_patients                               20153 non-null  float64
 15  hosp_patients_per_million                   20153 non-null  float64
 16  tests_per_case                              20153 non-null  float64
 17  new_vaccinations_smoothed_per_million       20153 non-null  float64
 18  new_people_vaccinated_smoothed              20153 non-null  float64
 19  new_people_vaccinated_smoothed_per_hundred  20153 non-null  float64
 20  stringency_index                            20153 non-null  float64
 21  population                                  20153 non-null  float64
 22  population_density                          20153 non-null  float64
 23  median_age                                  20153 non-null  float64
 24  aged_65_older                               20153 non-null  float64
 25  aged_70_older                               20153 non-null  float64
 26  gdp_per_capita                              20153 non-null  float64
 27  cardiovasc_death_rate                       20153 non-null  float64
 28  diabetes_prevalence                         20153 non-null  float64
 29  female_smokers                              20153 non-null  float64
 30  male_smokers                                20153 non-null  float64
 31  hospital_beds_per_thousand                  20153 non-null  float64
 32  life_expectancy                             20153 non-null  float64
 33  human_development_index                     20153 non-null  float64
dtypes: datetime64[ns](1), float64(31), object(2)
memory usage: 5.4+ MB
```

Figure 22 - Concise summary of the cleaned data frame (EU countries)

After all these steps of data cleaning we can consider that the data is clean and suitable for analysis.

Another part of our analysis focuses on Portugal. So, we selected the specific data related to Portugal from our raw data file. Then we removed the columns with the data we don't use in the analysis and proceeded to the cleaning of it by handling with the null values, outliers, and duplicates of it with the same proceedings we did with world data (python programming code found in detail in the appendix section). Below we can see the summary of the cleaned data frame for Portugal.

```
In [124]: # concise summary of the DataFrame
          dfPortugalClean.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 735 entries, 121544 to 122278
Data columns (total 52 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   iso_code                                  735 non-null    object
 1   location                                  735 non-null    object
 2   date                                      735 non-null    datetime64[ns]
 3   total_cases                               735 non-null    float64
 4   new_cases_smoothed                        735 non-null    float64
 5   total_deaths                              735 non-null    float64
 6   new_deaths_smoothed                       735 non-null    float64
 7   total_cases_per_million                   735 non-null    float64
 8   new_cases_smoothed_per_million            735 non-null    float64
 9   total_deaths_per_million                  735 non-null    float64
 10  new_deaths_smoothed_per_million           735 non-null    float64
 11  reproduction_rate                         735 non-null    float64
 12  icu_patients                              735 non-null    float64
 13  icu_patients_per_million                  735 non-null    float64
                                               735 non-null    float64
                                               735 non-null    float64
 16  total_tests                               735 non-null    float64
 17  total_tests_per_thousand                  735 non-null    float64
 18  new_tests_smoothed                        735 non-null    float64
 19  new_tests_smoothed_per_thousand           735 non-null    float64
 20  positive_rate                             735 non-null    float64
 21  tests_per_case                            735 non-null    float64
 22  total_vaccinations                        735 non-null    float64
 23  people_vaccinated                         735 non-null    float64
 24  people_fully_vaccinated                   735 non-null    float64
 25  total_boosters                            735 non-null    float64
 26  new_vaccinations_smoothed                 735 non-null    float64
 27  total_vaccinations_per_hundred            735 non-null    float64
 28  people_vaccinated_per_hundred             735 non-null    float64
 29  people_fully_vaccinated_per_hundred       735 non-null    float64
 30  total_boosters_per_hundred                735 non-null    float64
 31  new_vaccinations_smoothed_per_million     735 non-null    float64
 32  new_people_vaccinated_smoothed            735 non-null    float64
 33  new_people_vaccinated_smoothed_per_hundred 735 non-null   float64
 34  stringency_index                          735 non-null    float64
 35  population                                735 non-null    float64
 36  population_density                        735 non-null    float64
 37  median_age                                735 non-null    float64
 38  aged_65_older                             735 non-null    float64
 39  aged_70_older                             735 non-null    float64
 40  gdp_per_capita                            735 non-null    float64
 41  cardiovasc_death_rate                     735 non-null    float64
 42  diabetes_prevalence                       735 non-null    float64
 43  female_smokers                            735 non-null    float64
 44  male_smokers                              735 non-null    float64
 45  hospital_beds_per_thousand                735 non-null    float64
 46  life_expectancy                           735 non-null    float64
 47  human_development_index                   735 non-null    float64
 48  excess_mortality_cumulative_absolute      735 non-null    float64
 49  excess_mortality_cumulative               735 non-null    float64
 50  excess_mortality                          735 non-null    float64
 51  excess_mortality_cumulative_per_million   735 non-null    float64
dtypes: datetime64[ns](1), float64(49), object(2)
memory usage: 304.3+ KB
```

click to scroll output; double click to hide

Figure 23 - Concise summary of clean data frame for Portugal

After all these steps of data cleaning we can consider that the data is clean and suitable for analysis.

Finally, we came to the end of our cleaning process of the Covid-19 raw dataset to start another important phase of our data analysis, which is the exploratory data analysis.

5 Data Exploratory and Visualization Analysis

We analyzed our datasets with different EDA methods and visualize those data to provide a sufficient consciousness regarding the outbreak of COVID-19 all over the globe. We analyzed the comparison between the world pandemic evolution, the EU countries pandemic evolution and Portugal pandemic evolution, regarding the specific features we have chosen for our analysis to answer our research questions. Our exploit data performed with the Covid-19 dataset from “Our World in Data”.

We exported our clean Covid-19 selected datasets (World, Europe, Portugal) to Excel and Tableau to ensure an effective data exploration. We also did some exploratory data

analysis on Python to test the data before moving to Excel and Tableau (this exploration analysis on Python programming can be found in detail in the appendix section with its coding and graphics).

We started our exploratory analysis in Excel by building a 3D map which lets us to discover insights we may not see in traditional 2D tables or charts. We plotted geographic and temporal data related to our selected datasets on a 3D globe so we can see time stamped data change over time. We can visualize the evolution of Covid-19 pandemic changing over time. It provides a knowledge of how SARS-CoV-2 spread all around the globe. Each map segment represents a region, by using visual data analytics it helps the individuals to understand the epidemiological nature of COVID-19. This method of analyzing data will certainly increase the understanding of the situation and inform interventions. Below we can see a print of the automated 3D map generated in Excel and which can be seen in the PowerPoint presentation of this report.

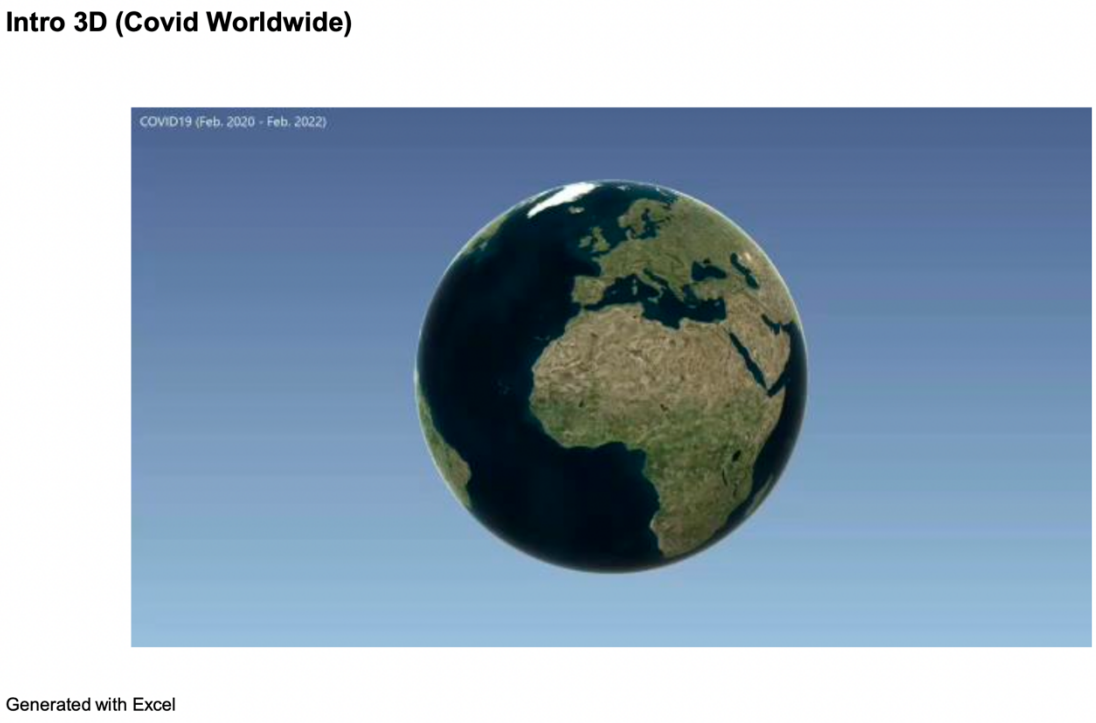


Figure 24 -World Covid-19 evolution 3D Map

After this first exploratory analysis and visualization in Excel and to interpret available data and reveal important aspects of the analysis, we loaded the sets into Tableau.

To clearly communicate some complex ideas, we found in the analysis and facilitate the identification and analysis of meaningful patterns we found in the data, we used Bar charts, Line charts, Scatter plots and Heat maps to communicate our findings. The dashboards and storytelling about our analysis can be found in the Tableau file of this report.

Following we will present some important exploratory dashboards made in Tableau that were essential in answering our questions.

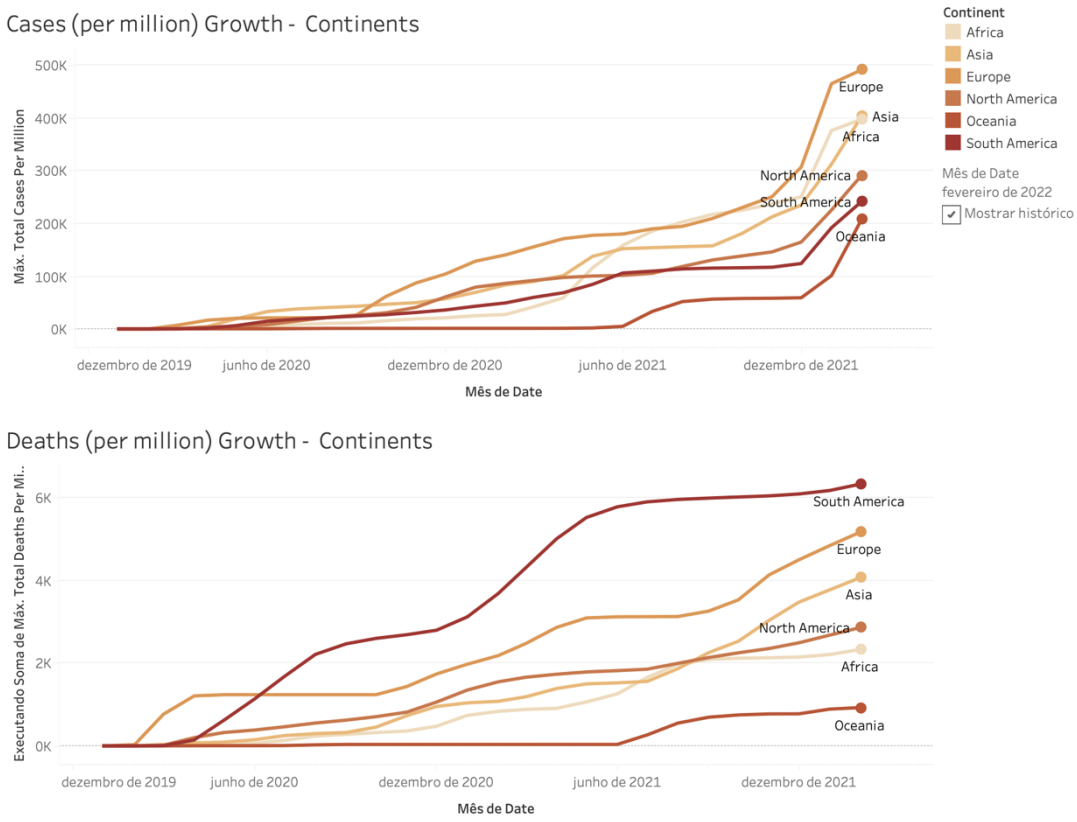


Figure 25 - Evolution of cases and deaths along time by Continent

The above figure presents how Covid cases and deaths evolved in the world along time and which continents have reached the highest values.

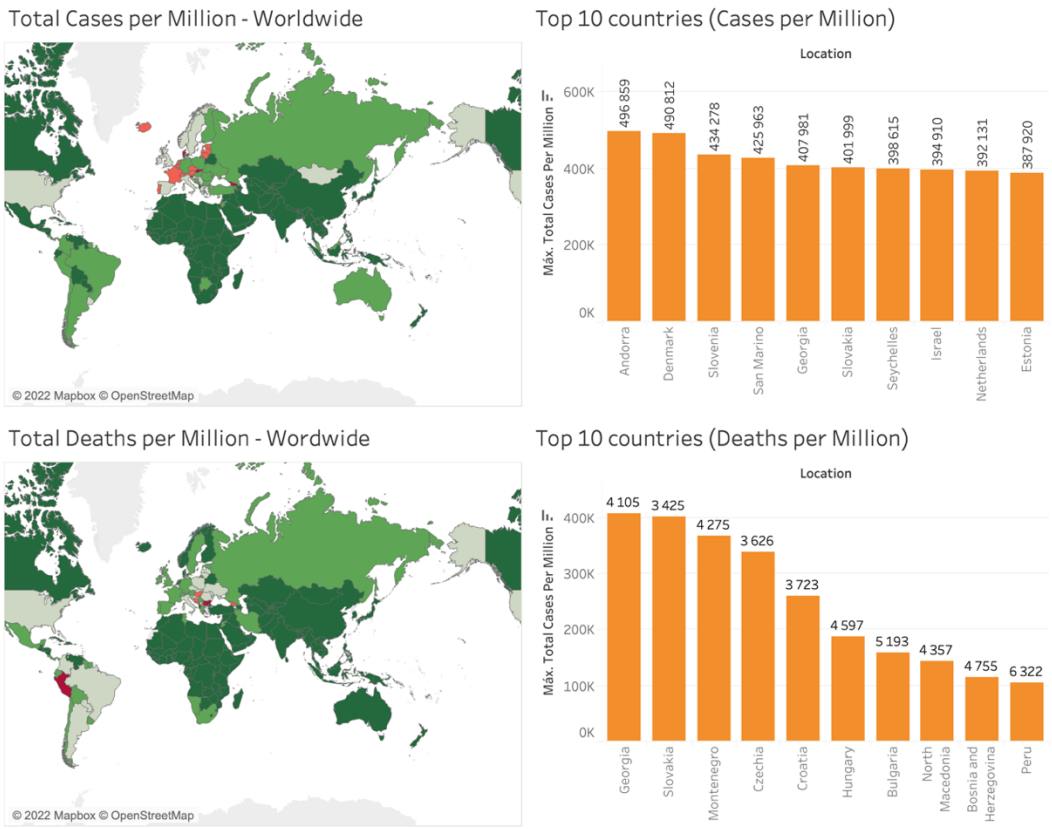


Figure 26 - Map of Total cases and deaths per million worldwide and the top 10 countries in cases and deaths

The above figure presents the top 10 countries with the most cases and the most deaths per million inhabitants and the total number worldwide.

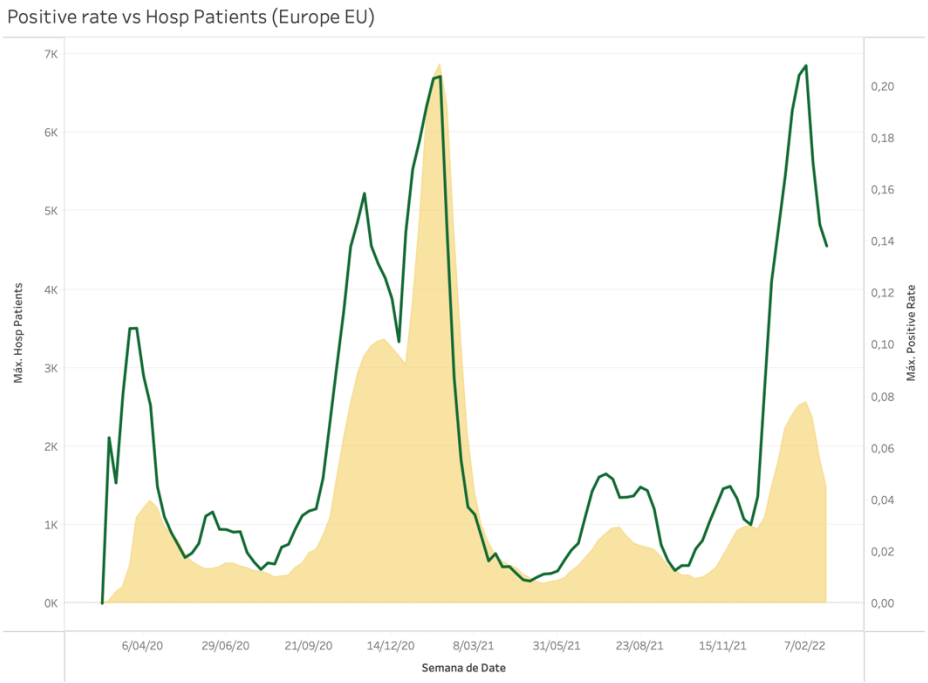


Figure 27 - Positive rate of cases Vs Hospital bed occupancy

The above figure presents how the evolution of positive cases affect the number of patients in hospitals in Europe (EU).

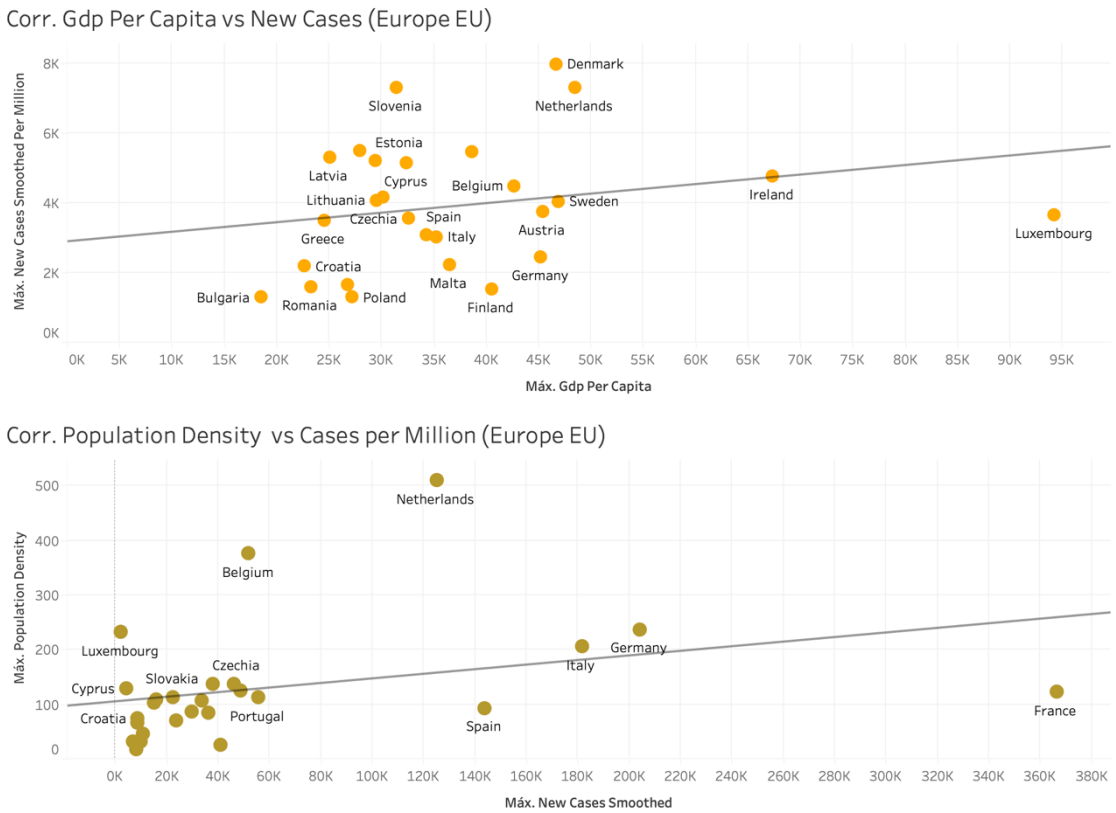


Figure 28 - Correlation between GDP per capita Vs New cases and Population density

The above figure presents what is the correlation between GDP per capita versus New cases and the Population density in Europe (EU).

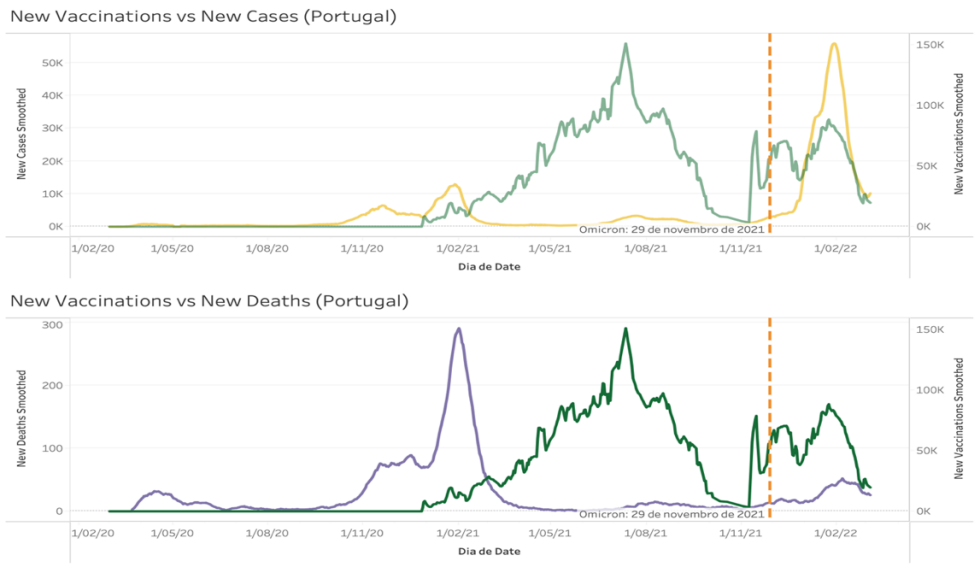


Figure 29 - Influence of vaccination on new cases and deaths in Portugal

The above figure presents what is the influence of vaccination on new cases and deaths in Portugal.

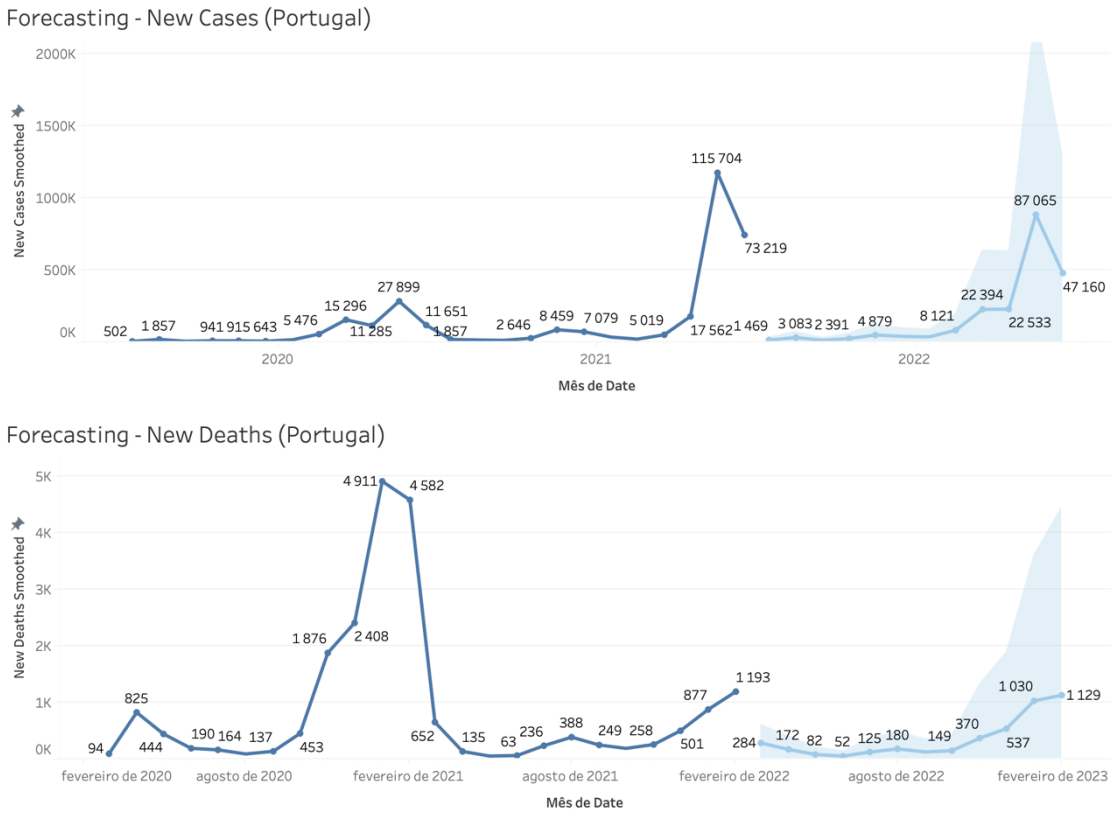


Figure 30 - Evolution and Forecasting of new cases and new deaths in Portugal

The above figure presents how was the evolution of new cases and deaths in Portugal and how we can predict the future.

By condensing multiple reports down into visuals, such as graphs charts, tables, and metrics, and displaying those visuals on analytics dashboards, it allows us to review significant amounts of valuable information briefly and intervene early during pandemic and adopt corrective and effective measures to stop Covid-19 spreading. Regarding all the data collected and information during this data analysis we came to some precious conclusion about our dataset which will be presented in the following section of this report.

6 Conclusion

This report presented current trends of COVID-19 outbreak from 24/2/2020 to 5/3/2022 as visualized in the dashboards and storytelling in Tableau file. The trajectory of the outbreak is also forecasted until February 2023. In a pandemic like this, providing timely information to the public is paramount. The outbreak spreads are largely influenced by each country's policy and social responsibility. A clear and insightful information will assist the government and authorities to disseminate corrective and effective measures to stop the spreading and allocate resources better. As so, and after exploration of all the selected data and deep analysis of it we came to several important conclusions that will answer our research questions and are stated below:

R1 - The evolution by continent of Covid19 cases per million inhabitants and deaths represented by the trend graph, shows that Europe, Asia, and Africa are on the list in cases and South America, Europe and Asia are at the top in terms of deaths.

R2 - The top 10 countries regarding both cases and deaths per million inhabitants, are represented using bar charts. Central and North European Countries are highlighted in this representation.

R3 - The line and area graph for the European Union Countries shows that the peaks of positive cases coincide with the peaks of hospitalized patients, justifying the overload of these services during the times when there were more cases.

R4 - Checking the correlation (GDP per capita vs New Cases) and (Population Density vs New Cases) using scatterplot with linear regression, we achieve on both cases as a result, a low R-Squared (near zero) and a high P-Value (>0.05) meaning the correlation is weak and evidence is not strong enough to suggest an effect exists in the population.

Concluding that both GDP per capita and Population Density in the European Union, are not factors influencing Covid19 new cases raise.

R5 - Accessing the relation between New Vaccinations and New Cases in Portugal using a line chart, it is possible to observe that although the first vaccination peak coincides with a time of few new cases, the same is not true after the appearance of the Omicron variant. On the other hand, comparing the evolution of vaccination with the emergence

of new deaths, it becomes evident that the increase in vaccination had a great influence on controlling the number of deaths.

R6 – Using run charts for both new cases and new deaths in Portugal, adding the Forecast function from Tableau, it is possible to predict the future for 1 year. For both new cases and new deaths, we can predict a sharp decline in March 2022, with a new rise at the end of the year around October and November, culminating in identical numbers of deaths to February 2022. Regarding cases, this rise but did not reach the pre-forecast values.

Despite all these conclusions, more work must be done to gather more data to work with and reach more insightful conclusions that can change the future of this worldwide spreading disease and help organizations and countries protect the most affected part of them which are the people.

References

- Binti Hamzah, F. A., Lau, C. H., Nazri, H., Ligot, D. C., Lee, G., Tan, C. L., & et al. (2020). CoronaTracker: World-wide Covid-19 outbreak data analysis and prediction. *Bulletin of the World Health Organization*, *March*.
- Chai, C. P. (2020). The Importance of Data Cleaning: Three Visualization Examples. *CHANCE*, *33*(1). https://doi.org/10.1080/09332480.2020.1726112
- Dey, S. K., Rahman, M. M., Siddiqi, U. R., & Howlader, A. (2020). Analyzing the epidemiological outbreak of COVID-19: A visual exploratory data analysis approach. *Journal of Medical Virology*, *92*(6). https://doi.org/10.1002/jmv.25743

Also:

- The Class Material provided by the instructor

Appendix

Data Cleaning and Data Exploration in Python programming code is shown in detail below:

Data Cleaning

```
# Importing the required libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

pd.set_option('display.max_rows', 100)

# Load the provided dataset file
dfRawData = pd.read_csv("covid-data.csv")

#-- Copying a backup from rawdata
dfToClean = dfRawData

# Comparing the dimensionality of the 2 DataFrames
print('dfRawData ',dfRawData.shape)
print('dfToClean ',dfToClean.shape)

#-- Choosing the columns that are useful for the scope of our analysis
selected_col = [
    #Geography - Where and when
    'iso_code', 'continent', 'location', 'date',

    #Covid cases per million - How many
    'total_cases', 'total_cases_per_million', 'total_deaths', 'total_deaths_per_million',

    #Economy and People
    'gdp_per_capita', 'population', 'population_density',

    #Medical Facilites
    'hospital_beds_per_thousand', 'life_expectancy'
]

# Create the new dataframe with the chosen columns
dfToClean = dfRawData[selected_col]

# Change the data format
dfToClean['date'] = pd.to_datetime(dfToClean['date'], format = '%Y-%m-%d')
```

```
# Remove the records with 'iso_code' containing 'OWID'
dfToClean = dfToClean[dfToClean["iso_code"].str.contains("OWID") == False]

# Replace NaN with zero in 'total_cases'
dfToClean['total_cases'] = dfToClean['total_cases'].replace(np.nan, 0)

# Replace NaN with zero in 'total_cases_per_million'
dfToClean['total_cases_per_million'] = dfToClean['total_cases_per_million'].replace(np.nan, 0)

# Replace NaN with zero in 'total_deaths'
dfToClean['total_deaths'] = dfToClean['total_deaths'].replace(np.nan, 0)

# Replace NaN with zero in 'total_deaths_per_million'
dfToClean['total_deaths_per_million'] = dfToClean['total_deaths_per_million'].replace(np.nan, 0)



# Replace null values in 'gdp_per_capita' for locations classified as independent countries

# Andorra gdp_per_capita to be replaced by 49900 according to CIA World Factbook,
# accessed in 16/07/2022
# https://www.cia.gov/the-world-factbook/countries/andorra/
dfToClean['gdp_per_capita'] = np.where((dfToClean['location'] == 'Andorra'), 49900, dfToClean['gdp_per_capita'])

# Cuba gdp_per_capita to be replaced by 21016.65 according to Trading Economics,
# accessed in 16/07/2022
# https://tradingeconomics.com/andorra/gdp-per-capita-ppp
dfToClean['gdp_per_capita'] = np.where((dfToClean['location'] == 'Cuba'), 21016.65, dfToClean['gdp_per_capita'])

# Liechtenstein gdp_per_capita to be replaced by 139100 according to CIA World Factbook,
# accessed in 16/07/2022
# https://www.cia.gov/the-world-factbook/countries/liechtenstein/
dfToClean['gdp_per_capita'] = np.where((dfToClean['location'] == 'Liechtenstein'), 139100, dfToClean['gdp_per_capita'])

# Monaco gdp_per_capita to be replaced by 115700 according to CIA World Factbook,
# accessed in 16/07/2022
# https://www.cia.gov/the-world-factbook/countries/monaco/
dfToClean['gdp_per_capita'] = np.where((dfToClean['location'] == 'Monaco'), 115700, dfToClean['gdp_per_capita'])

# Somalia gdp_per_capita to be replaced by 1302.45 according to Trading Economics,
# accessed in 16/07/2022
# https://tradingeconomics.com/somalia/gdp-per-capita-ppp
dfToClean['gdp_per_capita'] = np.where((dfToClean['location'] == 'Somalia'), 1302.45, dfToClean['gdp_per_capita'])

# Syria gdp_per_capita to be replaced by 4684.72 according to Trading Economics,
# accessed in 16/07/2022
# https://tradingeconomics.com/syria/gdp-per-capita-ppp
dfToClean['gdp_per_capita'] = np.where((dfToClean['location'] == 'Syria'), 4684.72, dfToClean['gdp_per_capita'])

# Taiwan gdp_per_capita to be replaced by 24502 according to CIA World Factbook,
# accessed in 16/07/2022
# https://www.cia.gov/the-world-factbook/countries/taiwan/
```

```
dfToClean['gdp_per_capita'] = np.where((dfToClean['location'] == 'Taiwan'), 24502, dfToClean['gdp_per_capita'])

# Remove the records where 'gdp_per_capita' isnull
# calculated loss of population data = 1289128
dfToClean = dfToClean.dropna(subset=['gdp_per_capita'])


#Replace null values in 'population_density'

# South Sudan population_density to be replaced by 18 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/EN.POP.DNST?locations=SS
dfToClean['population_density'] = np.where((dfToClean['location'] == 'South Sudan'), 18, dfToClean['population_density'])

# Syria population_density to be replaced by 100 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/EN.POP.DNST?locations=SY
dfToClean['population_density'] = np.where((dfToClean['location'] == 'Syria'), 100, dfToClean['population_density'])

# Taiwan population_density to be replaced by 673 according to Worldometer,
# accessed in 16/07/2022
# https://www.worldometers.info/world-population/taiwan-population/
dfToClean['population_density'] = np.where((dfToClean['location'] == 'Taiwan'), 673, dfToClean['population_density'])


#Replace null values in 'hospital_beds_per_thousand'

# Andorra hospital_beds_per_thousand to be replaced by 2.5 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=AD
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Andorra'), 2.5,
dfToClean['hospital_beds_per_thousand'])

# Angola hospital_beds_per_thousand to be replaced by 0.8 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=AO
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Angola'), 0.8,
dfToClean['hospital_beds_per_thousand'])

# Chad hospital_beds_per_thousand to be replaced by 0.4 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=TD
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Chad'), 0.4,
dfToClean['hospital_beds_per_thousand'])

# Congo hospital_beds_per_thousand to be replaced by 1.6 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=CG
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Congo'), 1.6,
dfToClean['hospital_beds_per_thousand'])
```

```
# "Cote d'Ivoire" hospital_beds_per_thousand to be replaced by 0.4 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=CI
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == "Cote d'Ivoire"), 0.4,
dfToClean['hospital_beds_per_thousand'])

# Democratic Republic of Congo hospital_beds_per_thousand to be replaced by 0.8 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=CD
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Democratic Republic of Congo'), 0.8,
dfToClean['hospital_beds_per_thousand'])

# Guinea-Bissau hospital_beds_per_thousand to be replaced by 1 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=GW
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Guinea-Bissau'), 1,
dfToClean['hospital_beds_per_thousand'])

# Lesotho hospital_beds_per_thousand to be replaced by 1.3 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=LS
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Lesotho'), 1.3,
dfToClean['hospital_beds_per_thousand'])

# Maldives hospital_beds_per_thousand to be replaced by 4.3 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=BM-MV
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Maldives'), 4.3,
dfToClean['hospital_beds_per_thousand'])

# Mauritania hospital_beds_per_thousand to be replaced by 0.4 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=MR
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Mauritania'), 0.4,
dfToClean['hospital_beds_per_thousand'])

# Micronesia (country) hospital_beds_per_thousand to be replaced by 3.2 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=FM
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Micronesia (country)'), 3.2,
dfToClean['hospital_beds_per_thousand'])

# Namibia (country) hospital_beds_per_thousand to be replaced by 2.7 according to The World Bank,
# accessed in 16/07/2022
# htts://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=NA
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Namibia'), 2.7,
dfToClean['hospital_beds_per_thousand'])

# Nigeria (country) hospital_beds_per_thousand to be replaced by 0.5 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=NG
```

```
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Nigeria'), 0.5, dfToClean['hospital_beds_per_thousand'])

# Papua New Guinea hospital_beds_per_thousand to be replaced by 4 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=PG
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Papua New Guinea'), 4, dfToClean['hospital_beds_per_thousand'])

# Rwanda hospital_beds_per_thousand to be replaced by 1.6 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=RW
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Rwanda'), 1.6, dfToClean['hospital_beds_per_thousand'])

# Samoa hospital_beds_per_thousand to be replaced by 1 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=WS

dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Samoa'), 1, dfToClean['hospital_beds_per_thousand'])

# Senegal hospital_beds_per_thousand to be replaced by 0.3 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=SN
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Senegal'), 0.3, dfToClean['hospital_beds_per_thousand'])

# Sierra Leone hospital_beds_per_thousand to be replaced by 0.4 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=SL
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Sierra Leone'), 0.4, dfToClean['hospital_beds_per_thousand'])

# Tuvalu hospital_beds_per_thousand to be replaced by 5.6 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=TV

dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Tuvalu'), 5.6, dfToClean['hospital_beds_per_thousand'])

# Vanuatu hospital_beds_per_thousand to be replaced by 1.7 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=VU

dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Vanuatu'), 1.7, dfToClean['hospital_beds_per_thousand'])

# Hong Kong hospital_beds_per_thousand to be replaced by 4.9 according to The World Bank,
# accessed in 16/07/2022
# https://data.worldbank.org/indicator/SH.MED.BEDS.ZS?locations=HK
```

```
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Hong Kong'), 4.9, dfToClean['hospital_beds_per_thousand'])

# Macao hospital_beds_per_thousand to be replaced by 5.3 according to WorldData.info,
# accessed in 16/07/2022
# https://www.worlddata.info/asia/macao/health.php
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Macao'), 5.3, dfToClean['hospital_beds_per_thousand'])

# Palestine hospital_beds_per_thousand to be replaced by 1.2 according to fanack.com,
# accessed in 16/07/2022
# https://fanack.com/palestine/politics-of-palestine/the-health-sector-in-palestine/
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Palestine'), 1.2, dfToClean['hospital_beds_per_thousand'])

# South Sudan hospital_beds_per_thousand to be replaced by 0.74 according to Trading Economics,
# accessed in 16/07/2022
# https://tradingeconomics.com/sudan/hospital-beds-per-1-000-people-wb-data.html
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'South Sudan'), 0.74, dfToClean['hospital_beds_per_thousand'])

# Taiwan hospital_beds_per_thousand to be replaced by 5.7 according to PWC,
# accessed in 16/07/2022
# https://www.pwc.tw/en/publications/assets/taiwan-health-industries.pdf
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Taiwan'), 5.7, dfToClean['hospital_beds_per_thousand'])

# Bermuda hospital_beds_per_thousand to be replaced by 6.3 according to WorldData.info,
# accessed in 16/07/2022
# https://www.worlddata.info/america/bermuda/health.php
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Bermuda'), 6.3, dfToClean['hospital_beds_per_thousand'])

# Cayman Islands hospital_beds_per_thousand to be replaced by 3 according to WorldData.info,
# accessed in 16/07/2022
# https://www.worlddata.info/america/cayman-islands/health.php
dfToClean['hospital_beds_per_thousand'] = np.where((dfToClean['location'] == 'Cayman Islands'), 3, dfToClean['hospital_beds_per_thousand'])


# Remove the records where 'hospital_beds_per_thousand for the remaining locations
# calculated loss of population data = 150616
dfToClean = dfToClean.dropna(subset=['hospital_beds_per_thousand'])


# Check for Outliers
dfToClean.describe()


# copy the clean data to a new DataFrame
dfClean = dfToClean
```

```
# concise summary of the DataFrame
dfClean.info()


# Export the clean data to Excel
dfClean.to_excel("Group_A_covid-data_World.xlsx", sheet_name='Group_A_covid-data_World', index=False)
print("file saved as Group_A_covid-data_World.xlsx")


# Excel file should have 141130+1 rows and 13 columns
dfClean.shape


# ---------------------------------------------
# *Europe EU*
# ---------------------------------------------

# Load the provided dataset file
dfEuropeRawData = pd.read_csv("covid-data.csv")

#-- Copying a backup from rawdata
dfEuropeToClean = dfEuropeRawData

# Comparing the dimensionality of the 2 DataFrames
print('EuropeRawData ',dfEuropeRawData.shape)
print('EuropeToClean ',dfEuropeToClean.shape)

# Change the data format
dfEuropeToClean['date'] = pd.to_datetime(dfEuropeToClean['date'], format = '%Y-%m-%d')

# Remove all records not from Europe
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['continent'] != 'Europe'].index, inplace = True)

# drop countries not in the EU
# Members of EU:
# Austria, Belgium, Bulgaria, Croatia, Cyprus, Czechia, Denmark, Estonia, Finland, France, Germany, Greece,
# Hungary, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Poland, Portugal, Romania,
# Slovakia, Slovenia, Spain and Sweden.

dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Albania'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Andorra'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Belarus'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Bosnia and Herzegovina'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Faeroe Islands'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Gibraltar'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Guernsey'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Iceland'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Isle of Man'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Jersey'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Kosovo'].index, inplace = True)
```

```
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Liechtenstein'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Moldova'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Monaco'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Montenegro'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'North Macedonia'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Norway'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Russia'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'San Marino'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Serbia'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Switzerland'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Ukraine'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'United Kingdom'].index, inplace = True)
dfEuropeToClean.drop(dfEuropeToClean[dfEuropeToClean['location'] == 'Vatican'].index, inplace = True)

# drop columns that will not be used in our analysis
dfEuropeToClean=dfEuropeToClean.drop(columns=['continent', 'new_cases', 'new_deaths', 'new_cases_per_million',
                                    'new_deaths_per_million', 'new_tests', 'new_tests_per_thousand',
                                    'new_vaccinations', 'extreme_poverty', 'handwashing_facilities',
                                    'weekly_icu_admissions', 'weekly_icu_admissions_per_million',
                                    'weekly_hosp_admissions', 'weekly_hosp_admissions_per_million',
                                    'tests_units'])


# Replace NaN with zero in 'total_cases'
dfEuropeToClean['total_cases'] = dfEuropeToClean['total_cases'].replace(np.nan, 0)

# Replace NaN with zero in 'new_cases_smoothed'
dfEuropeToClean['new_cases_smoothed'] = dfEuropeToClean['new_cases_smoothed'].replace(np.nan, 0)

# Replace NaN with zero in 'total_deaths'
dfEuropeToClean['total_deaths'] = dfEuropeToClean['total_deaths'].replace(np.nan, 0)

# Replace NaN with zero in 'new_deaths_smoothed'
dfEuropeToClean['new_deaths_smoothed'] = dfEuropeToClean['new_deaths_smoothed'].replace(np.nan, 0)

# Replace NaN with zero in 'total_cases_per_million'
dfEuropeToClean['total_cases_per_million'] = dfEuropeToClean['total_cases_per_million'].replace(np.nan, 0)

# Replace NaN with zero in 'new_cases_smoothed_per_million'
dfEuropeToClean['new_cases_smoothed_per_million'] = dfEuropeToClean['new_cases_smoothed_per_million'].replace(np.nan, 0)

# Replace NaN with zero in 'total_deaths_per_million'
dfEuropeToClean['total_deaths_per_million'] = dfEuropeToClean['total_deaths_per_million'].replace(np.nan, 0)

# Replace NaN with zero in 'new_deaths_smoothed_per_million'
dfEuropeToClean['new_deaths_smoothed_per_million']  =  dfEuropeToClean['new_deaths_smoothed_per_million'].replace(np.nan,
0)

# Replace NaN with zero in 'reproduction_rate'
dfEuropeToClean['reproduction_rate'] = dfEuropeToClean['reproduction_rate'].replace(np.nan, 0)
```

```
# Replace NaN with zero in 'icu_patients'
dfEuropeToClean['icu_patients'] = dfEuropeToClean['icu_patients'].replace(np.nan, 0)

# Replace NaN with zero in 'icu_patients_per_million'
dfEuropeToClean['icu_patients_per_million'] = dfEuropeToClean['icu_patients_per_million'].replace(np.nan, 0)

# Replace NaN with zero in 'hosp_patients'
dfEuropeToClean['hosp_patients'] = dfEuropeToClean['hosp_patients'].replace(np.nan, 0)

# Replace NaN with zero in 'hosp_patients_per_million'
dfEuropeToClean['hosp_patients_per_million'] = dfEuropeToClean['hosp_patients_per_million'].replace(np.nan, 0)

# Replace NaN with zero in 'tests_per_case'
dfEuropeToClean['tests_per_case'] = dfEuropeToClean['tests_per_case'].replace(np.nan, 0)

# Replace NaN with zero in 'new_vaccinations_smoothed_per_million'
dfEuropeToClean['new_vaccinations_smoothed_per_million']                                                    =
dfEuropeToClean['new_vaccinations_smoothed_per_million'].replace(np.nan, 0)

# Replace NaN with zero in 'new_people_vaccinated_smoothed'
dfEuropeToClean['new_people_vaccinated_smoothed'] = dfEuropeToClean['new_people_vaccinated_smoothed'].replace(np.nan, 0)

# Replace NaN with zero in 'new_people_vaccinated_smoothed_per_hundred'
dfEuropeToClean['new_people_vaccinated_smoothed_per_hundred']                                               =
dfEuropeToClean['new_people_vaccinated_smoothed_per_hundred'].replace(np.nan, 0)

# Replace NaN with zero in 'stringency_index'
dfEuropeToClean['stringency_index'] = dfEuropeToClean['stringency_index'].replace(np.nan, 0)



#-- Copying a copy for Europe (Portugal) for later use
dfPortugalToClean = dfEuropeToClean

# Comparing the dimensionality of the 2 DataFrames
print('dfPortugalToClean ',dfPortugalToClean.shape)
print('dfEuropeToClean ',dfEuropeToClean.shape)

# drop columns that will not be used in our analysis
dfEuropeToClean=dfEuropeToClean.drop(columns=['total_tests', 'total_tests_per_thousand', 'new_tests_smoothed',
                                              'new_tests_smoothed_per_thousand', 'positive_rate', 'total_vaccinations',
                                              'people_vaccinated', 'people_fully_vaccinated', 'total_boosters',
                                              'new_vaccinations_smoothed', 'total_vaccinations_per_hundred',
                                              'people_vaccinated_per_hundred', 'people_fully_vaccinated_per_hundred',
                                              'total_boosters_per_hundred', 'excess_mortality_cumulative_absolute',
                                              'excess_mortality_cumulative', 'excess_mortality',
                                              'excess_mortality_cumulative_per_million',])



#  Check for Outliers
dfEuropeToClean.describe()
```

```
# copy the clean data to a new DataFrame
dfEuropeClean = dfEuropeToClean

# concise summary of the DataFrame
dfEuropeClean.info()

# Export the clean data to Excel
dfEuropeClean.to_excel("Group_A_covid-data_Europe.xlsx", sheet_name='Group_A_covid-data_Europe', index=False)

print("file saved as Group_A_covid-data_Europe.xlsx")


# ---------------------------------------------
# *Portugal*
# ---------------------------------------------

dfPortugalToClean.shape

# Remove all records not from Portugal
dfPortugalToClean.drop(dfPortugalToClean[dfPortugalToClean['location'] != 'Portugal'].index, inplace = True)

# Replace null values with ffill
dfPortugalToClean['total_tests'].fillna(method='ffill', inplace=True)
dfPortugalToClean['total_tests_per_thousand'].fillna(method='ffill', inplace=True)
dfPortugalToClean['positive_rate'].fillna(method='ffill', inplace=True)
dfPortugalToClean['total_vaccinations'].fillna(method='ffill', inplace=True)
dfPortugalToClean['people_vaccinated'].fillna(method='ffill', inplace=True)
dfPortugalToClean['people_fully_vaccinated'].fillna(method='ffill', inplace=True)
dfPortugalToClean['total_boosters'].fillna(method='ffill', inplace=True)
dfPortugalToClean['total_vaccinations_per_hundred'].fillna(method='ffill', inplace=True)
dfPortugalToClean['people_vaccinated_per_hundred'].fillna(method='ffill', inplace=True)
dfPortugalToClean['people_fully_vaccinated_per_hundred'].fillna(method='ffill', inplace=True)
dfPortugalToClean['total_boosters_per_hundred'].fillna(method='ffill', inplace=True)
dfPortugalToClean['excess_mortality_cumulative_absolute'].fillna(method='ffill', inplace=True)
dfPortugalToClean['excess_mortality_cumulative'].fillna(method='ffill', inplace=True)
dfPortugalToClean['excess_mortality'].fillna(method='ffill', inplace=True)
dfPortugalToClean['excess_mortality_cumulative_per_million'].fillna(method='ffill', inplace=True)

# Replace null values with zero
dfPortugalToClean['new_tests_smoothed'] = dfPortugalToClean['new_tests_smoothed'].replace(np.nan, 0)
dfPortugalToClean['new_tests_smoothed_per_thousand']                                                =
dfPortugalToClean['new_tests_smoothed_per_thousand'].replace(np.nan, 0)
dfPortugalToClean['positive_rate'] = dfPortugalToClean['positive_rate'].replace(np.nan, 0)
dfPortugalToClean['total_vaccinations'] = dfPortugalToClean['total_vaccinations'].replace(np.nan, 0)
dfPortugalToClean['people_vaccinated'] = dfPortugalToClean['people_vaccinated'].replace(np.nan, 0)
dfPortugalToClean['people_fully_vaccinated'] = dfPortugalToClean['people_fully_vaccinated'].replace(np.nan, 0)
dfPortugalToClean['total_boosters'] = dfPortugalToClean['total_boosters'].replace(np.nan, 0)
dfPortugalToClean['new_vaccinations_smoothed'] = dfPortugalToClean['new_vaccinations_smoothed'].replace(np.nan, 0)
dfPortugalToClean['total_vaccinations_per_hundred'] = dfPortugalToClean['total_vaccinations_per_hundred'].replace(np.nan, 0)
dfPortugalToClean['people_vaccinated_per_hundred'] = dfPortugalToClean['people_vaccinated_per_hundred'].replace(np.nan, 0)
```

```
dfPortugalToClean['people_fully_vaccinated_per_hundred'] =
dfPortugalToClean['people_fully_vaccinated_per_hundred'].replace(np.nan, 0)
dfPortugalToClean['total_boosters_per_hundred'] = dfPortugalToClean['total_boosters_per_hundred'].replace(np.nan, 0)

# copy the clean data to a new DataFrame
dfPortugalClean = dfPortugalToClean

# concise summary of the DataFrame
dfPortugalClean.info()

# Export the clean data to Excel
dfPortugalClean.to_excel("Group_A_covid-data_Portugal.xlsx", sheet_name='Group_A_covid-data_Portugal', index=False)

print("file saved as Group_A_covid-data_Portugal.xlsx")


#  EOF
```

Data Exploration

```
# ----------------------------------------------------------------
# *Exploratory Data Analysis (EDA)*
# *We will test the data before moving to Excel and Tableau*
# ----------------------------------------------------------------

#Importing the required libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

pd.set_option('display.max_rows', 100)

# Load the excel files created before
dfWorld = pd.read_excel("Group_A_covid-data_World.xlsx")
dfEurope = pd.read_excel("Group_A_covid-data_Europe.xlsx")
dfPortugal = pd.read_excel("Group_A_covid-data_Portugal.xlsx")

# *** Optional ***
# Change the plot size
sns.set(rc={'figure.figsize':(12,7)})

# ----------------------------------------------------------------
# 10.2 Density plot - Total Deaths per Million in World
sns.set(style = "darkgrid")
df = dfWorld

# density plot with shaded area with kdeplot 'shade' parameter
sns.kdeplot(df['total_deaths_per_million'], shade = True)
```

```
# add title
plt.title("Total Deaths per Million in World", loc = "left")

plt.show()

print(dfWorld['total_deaths_per_million'].describe())


# ---------------------------------------------------------------  #
# 10.3 Density plot - Total Deaths per Million in Europe
sns.set(style = "darkgrid")
df = dfWorld

# density plot with shaded area with kdeplot 'shade' parameter
sns.kdeplot(df['total_deaths_per_million'], shade = True)

# add title
plt.title("Total Deaths per Million in Europe", loc = "center")

plt.show()

print(dfEurope['total_deaths_per_million'].describe())


# ---------------------------------------------------------------  #
# 10.3 Density plot - Total Cases in Portugal

# Usual boxplot
sns.boxplot(x = 'date', y = 'total_cases', data = dfPortugal)

# add title
plt.title("Total Cases in Portugal", loc = "center")

plt.show()

print(dfPortugal['total_cases'].describe())


# ---------------------------------------------------------------  #
# 10.4 Density plot - People Fully Vaccinated per Hundred in Portugal

# Usual boxplot
sns.boxplot(x = 'date', y = 'people_fully_vaccinated_per_hundred', data = dfPortugal)

# add title
plt.title("People Fully Vaccinated per Hundred in Portugal", loc = "center")

plt.show()

print(dfPortugal['people_fully_vaccinated_per_hundred'].describe())
```

```
# ---------------------------------------------------------------- #
# 10.5 Density plot - Total Cases per Million and Total Deaths per Million in Portugal

# boxplot
ax = sns.boxplot(x = 'date', y = 'total_cases_per_million', data = dfPortugal)
# add stripplot
ax = sns.stripplot(x = 'date', y = 'total_deaths_per_million', data = dfPortugal, color = "blue", jitter = 0.2, size = 2.5)

# add title
plt.title("Portugal - Total Cases p. Mil. (Black) vs Total Deaths p. Mil. (Blue)", loc = "center")

# show the graph
plt.show()

print('Portugal - total_cases_per_million')
print(dfPortugal['total_cases_per_million'].describe())


print('total_deaths_per_million')
print(dfPortugal['total_deaths_per_million'].describe())


# ---------------------------------------------------------------- #
# 10.6 Density plot - Total Cases per Million in Europe and Portugal

# boxplot
ax = sns.boxplot(x = 'date', y = 'total_cases_per_million', data = dfEurope)
# add stripplot
ax = sns.stripplot(x = 'date', y = 'total_cases_per_million', data = dfPortugal, color = "blue", jitter = 0.2, size = 2.5)

# add title
plt.title("Total Cases per Million in Europe (Black) vs Portugal (Blue)", loc = "left")

# show the graph
plt.show()

print('Europe - total_cases_per_million')
print(dfEurope['total_cases_per_million'].describe())


print('Portugal - total_cases_per_million')
print(dfPortugal['total_cases_per_million'].describe())


# ---------------------------------------------------------------- #
# 10.7 Density plot - Total Cases per Million in World, Europe and Portugal

# boxplot
ax = sns.boxplot(x = 'date', y = 'total_cases_per_million', data = dfWorld)
```

```
# add stripplot
ax = sns.stripplot(x = 'date', y = 'total_cases_per_million', data = dfEurope, color = "green", jitter = 0.2, size = 2.5)
# add stripplot
ax = sns.stripplot(x = 'date', y = 'total_cases_per_million', data = dfPortugal, color = "blue", jitter = 0.2, size = 2.5)

# add title
plt.title("Total Cases p.Mil. in World(Black) vs Europe(Green) vs Portugal(Blue)", loc = "left")

# show the graph
plt.show()

print('World - total_cases_per_million')
print(dfWorld['total_cases_per_million'].describe())


print('Europe - total_cases_per_million')
print(dfEurope['total_cases_per_million'].describe())


print('Portugal - total_cases_per_million')
print(dfPortugal['total_cases_per_million'].describe())


# ---------------------------------------------------------------  #
# 10.8 Scatter - Life Expectancy vs GDP per Capita PPP in the World

dfWorld.plot(kind='scatter', x='life_expectancy', y='gdp_per_capita',
        title='World - Life Expectancy vs GDP per Capita PPP');


print('World - life_expectancy')
print(dfWorld['life_expectancy'].describe())


print('World - gdp_per_capita')
print(dfWorld['gdp_per_capita'].describe())


# ---------------------------------------------------------------  #
# 10.9 Distribution - Total Cases per Million in Europe

plt.figure(figsize=(9, 8))
sns.distplot(dfEurope['total_cases_per_million'], color='g', bins=100,
        hist_kws={'alpha': 0.4}).set(title='Total Cases per Million in Europe');


print(dfEurope['total_cases_per_million'].describe())


# ---------------------------------------------------------------  #
# 10.10 Numerical Data Distribution
```

```
# List the numerical data types in the dataset

dfNumDistPT = dfPortugal.select_dtypes(include = ['float64', 'int64'])
dfNumDistPT.head()

# Select the numerical data types to be used
dfNumDistPT1 = dfNumDist[["total_cases", "total_cases_per_million", "excess_mortality_cumulative_absolute"]]

# Confirm that the choosen data types are present
dfNumDistPT1.head()

# plot the choosen data types
dfNumDistPT1.hist(figsize=(16, 20), bins=50, xlabelsize=8, ylabelsize=8);


# ---------------------------------------------------------------  #
# 10.11 Correlation testing

# Select the numerical data types to be used
dfCorrelPT1 = dfNumDistPT[['total_vaccinations', 'total_cases', 'total_deaths', 'new_cases_smoothed',
                           'new_deaths_smoothed', 'reproduction_rate', 'total_tests']]


plt.figure(figsize=(16, 6))
heatmap = sns.heatmap(dfCorrelPT1.corr(), vmin=-1, vmax=1, annot=True)
heatmap.set_title('Correlation Heatmap PT 1', fontdict={'fontsize':18}, pad=12);


# ---------------------------------------------------------------  #

# Select the numerical data types to be used
dfCorrelPT2 = dfNumDistPT[['people_fully_vaccinated_per_hundred', 'new_deaths_smoothed', 'icu_patients_per_million']]


plt.figure(figsize=(8, 6))
heatmap = sns.heatmap(dfCorrelPT2.corr(), vmin=-1, vmax=1, annot=True)
heatmap.set_title('Correlation Heatmap PT 2', fontdict={'fontsize':18}, pad=12);


# ---------------------------------------------------------------  #

# Select the numerical data types to be used
dfCorrelPT3 = dfNumDistPT[['new_cases_smoothed_per_million', 'people_fully_vaccinated_per_hundred']]


plt.figure(figsize=(8, 6))
heatmap = sns.heatmap(dfCorrelPT3.corr(), vmin=-1, vmax=1, annot=True)
heatmap.set_title('Correlation Heatmap PT 3', fontdict={'fontsize':18}, pad=12);
```

```
# ------------------------------------------------------------  # In[249]:

dfNumDistW = dfWorld.select_dtypes(include = ['float64', 'int64'])
dfNumDistW.head()


# Select the numerical data types to be used
dfCorrelW1 = dfNumDistW[['gdp_per_capita', 'life_expectancy', 'total_cases_per_million', 'hospital_beds_per_thousand']]

# ------------------------------------------------------------


plt.figure(figsize=(8, 6))
heatmap = sns.heatmap(dfCorrelW1.corr(), vmin=-1, vmax=1, annot=True)
heatmap.set_title('Correlation Heatmap World 1', fontdict={'fontsize':18}, pad=12);


# ------------------------------------------------------------  #
# 10.12 Country distribution by continent

CountDist = dfWorld.groupby(['location','continent'], as_index=False).max('date')

sns.countplot(y=CountDist.continent)
plt.xticks(rotation=75);
plt.title('Country distribution by continent')
plt.ylabel(None);


# ------------------------------------------------------------  #
# 10.13 Total cases per million distribution by continent

caseDist = dfWorld.groupby(['continent','total_cases_per_million'], as_index=False).max('date')

sns.countplot(y=caseDist.continent)
plt.xticks(rotation=75);
plt.title('Total cases per million distribution by continent')
plt.ylabel(None);


# ------------------------------------------------------------  #
# 10.14 Total cases and total deaths per million per continent

f, ax = plt.subplots(figsize=(12, 8))
data = dfWorld[['continent','total_cases_per_million','total_deaths_per_million']]
data.sort_values('total_cases_per_million',ascending=False,inplace=True)
sns.set_color_codes("pastel")
sns.barplot(x="total_cases_per_million", y="continent", data=dfWorld,label="Total Cases per million", color="green")

sns.set_color_codes("muted")
sns.barplot(x="total_deaths_per_million", y="continent", data=dfWorld, label="Total Deaths per million", color="red")
ax.legend(ncol=2, loc="upper right", frameon=True)
```

```
ax.set_title('Continents - Total Cases and Total Deaths per million');


# ------------------------------------------------------------  #
#  10.15 Total cases and total deaths per continent

f, ax = plt.subplots(figsize=(12, 8))
data = dfWorld[['continent','total_cases','total_deaths']]
data.sort_values('total_cases',ascending=False,inplace=True)
sns.set_color_codes("pastel")
sns.barplot(x="total_cases", y="continent", data=dfWorld,label="Total Cases", color="green")

sns.set_color_codes("muted")
sns.barplot(x="total_deaths", y="continent", data=dfWorld, label="Total Deaths", color="red")
ax.legend(ncol=2, loc="upper right", frameon=True)
ax.set_title('Continents - Total Cases and Total Deaths');


# EOF
```