**Project title : COVID-19 CASES ANALYSIS**

**DAC\_PHASE 3 Submission document**

Team Member:N.palanikumar\au812121106019

Introduction

The goal of this project is the loading, preprocessing and exploratory data analysis of SARS-Cov-2 data on a global scale using [PySpark](https://spark.apache.org/docs/latest/api/python/), Apache Spark's Python API. The [dataset](https://github.com/owid/covid-19-data/blob/master/public/data/README.md) was curated and is maintained by [Our World in Data](https://ourworldindata.org/coronavirus) (OWiD). Typically, Spark is utilized when dealing with much larger datasets than the one seen here. In fact, all the tasks performed below could be performed using Pandas, in a somewhat cleaner and more familiar fashion. Nonetheless, the purpose of this notebook is not the analysis itself (especially since Covid-19 data have been extensively analyzed before), but the introduction of the reader to PySpark. In this sense, the dataset is more of a workhorse serving an educational purpose, so Pandas will be used only when absolutely necessary.

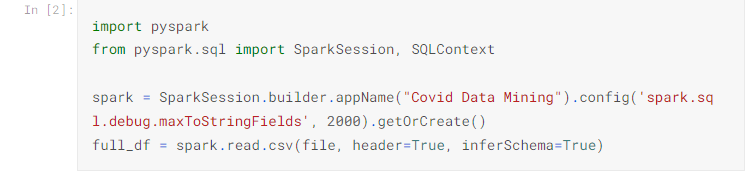
linkcode

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.



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



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



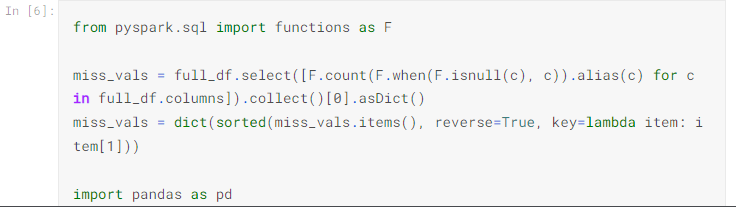
Most of the features are double types, i.e. numerical data. However, nominal features are also present:

* 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.



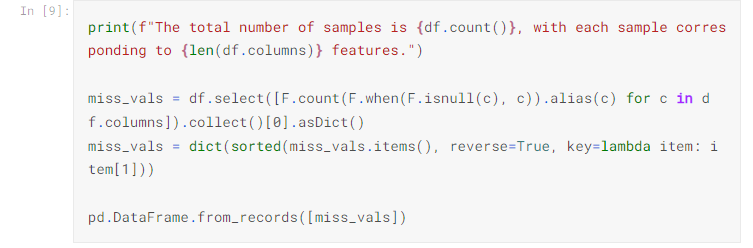
Note that the tests\_units column in the above command contains several null values, indicating that many of them may appear in the present dataset. We may import functions from pyspark.sql, which allows us to perform aggregations (among other things) and count the exact number of null values for each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





## 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.



### 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.

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 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.")

### 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.

#### 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%.

linkcode

A better way of approaching this is by developing a series of lollipop charts that depict the evolution of the top 10 countries with respect to the mortality rate. The python code used to extract each lollipop chart image can be found in the [Appendix](https://www.kaggle.com/code/spyrosrigas/analyzing-covid-19-data-with-pyspark#appendix) and all images were merged into a .gif file using Adobe Photoshop. The result can be seen below:

The lollipop chart provides the information that the code above provided. In addition to that, it presents the exact changes in ratings between the top 10 countries.

* **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]:

linkcode

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()

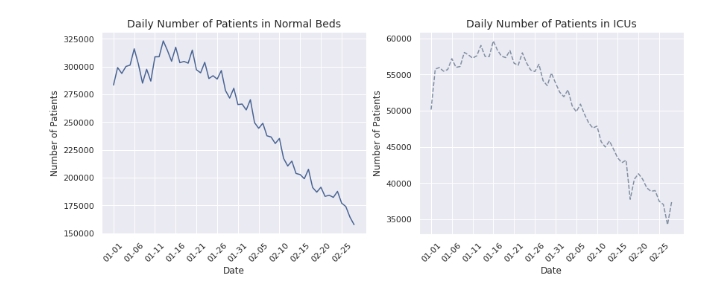
matplotlib.rc\_file\_defaults()

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.

linkcode

* **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/).

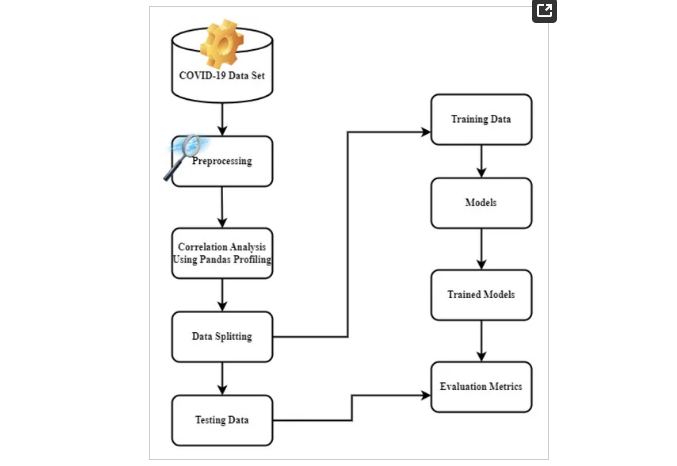


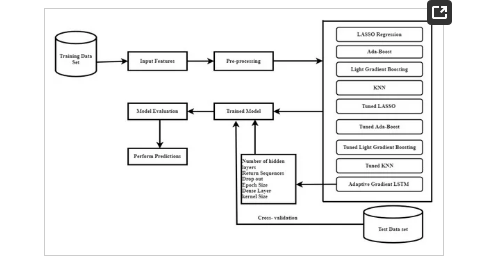
* **Geographic Correlation of Excess Mortality**

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).





Due to how complicated the COVID-19 outbreak is, its uncertainty, and how many countries do not have important data because they do not have as many ways to collect data as a country like India, the main concern is not only how accurate the models are, but also how well they can be used in a wide range of situations [[**12**](https://www.mdpi.com/1660-4601/20/11/5943#B12-ijerph-20-05943)]. This work attempts to solve this complex problem in this environment with the least amount of training and prediction time possible. By reusing previously developed prediction models, we also take data fusion and transfer learning into consideration. This is because; deep neural networks and deep ensemble learning training require large amounts of computational time, massive data, and computer resources. In fact, the focus of this work is data fusion, a method that integrates data to create knowledge that is more accurate, consistent, and informative, while real data could be inaccurate, unclear, inconsistent, and insufficient [[**13**](https://www.mdpi.com/1660-4601/20/11/5943#B13-ijerph-20-05943)]. We can develop precise predictions that might run into problems when gathering COVID-19 data by concentrating on data fusion.

The article is organized in the following manner. [**Section 2**](https://www.mdpi.com/1660-4601/20/11/5943#sec2-ijerph-20-05943) discusses the research works that employ machine learning and deep learning models for the prediction of COVID-19. The proposed state-of-the-art using multivariate data for the prediction of COVID-19 is outlined in [**Section 3**](https://www.mdpi.com/1660-4601/20/11/5943#sec3-ijerph-20-05943), which will be followed by the experimental setting used in this research. Subsequently, the results of the conducted trials are given and discussed, and the article ends with a conclusion, future work and limitation section.

**Literature Review**

This section’s main goal is to assess some of the most significant recent relevant attempts at COVID-19 outbreak prediction utilizing machine and deep learning methods. This review of the literature focuses, in particular, on studies that forecast everyday events that are verified or positive. M. Li et al. [[**14**](https://www.mdpi.com/1660-4601/20/11/5943#B14-ijerph-20-05943)] suggest a machine learning technique for estimating the daily numbers of cumulative confirmed cases, newly confirmed cases, and death cases of COVID-19 in China from 20 January 2020 to 1 March 2020, using data from the National Health Committee of China. A comparison of machine learning and soft computing models for COVID-19 outbreak prediction in five counties was conducted by S.F. Ardabili et al. [[**15**](https://www.mdpi.com/1660-4601/20/11/5943#B15-ijerph-20-05943)]. Two machine learning models’ results—the multi-layered perceptron and the adaptive network-based fuzzy inference system—were promising and had a high capacity for long-term prediction. S. Bandyopadhyay et al. [[**16**](https://www.mdpi.com/1660-4601/20/11/5943#B16-ijerph-20-05943)] used recurrent neural networks (RNNs) to predict COVID-19 confirmed (positive), negative, released, and death cases. RNNs can represent the prediction of temporal (sequential) data. Three models—a combined LSTM-GRU model, a gated-recurrent unit (GRU) model, and a long short-term memory (LSTM) model—were presented. According to experimental findings on the COVID-19 dataset for South Korea from 20 January 2020 to 12 March 2020, the combined model achieves the highest level of accuracy. A convolutional neural network (CNN) model was proposed by C.J. Huang et al. [[**17**](https://www.mdpi.com/1660-4601/20/11/5943#B17-ijerph-20-05943)] to predict the number of COVID-19 verified cases in China from 23 January 2020 to 2 March 2020, using information from Growing News Network and WHO. According to experiments, the recommended CNN model outperforms MLP (multilayer perceptron), LSTM, and GRU. For predicting the number of novel coronavirus (COVID-19)-positive reported cases for 32 Indian states and union territories, P.H. Kumar et al. [[**18**](https://www.mdpi.com/1660-4601/20/11/5943#B18-ijerph-20-05943)] used deep learning-based models, specifically LSTM variants such as deep LSTM, convolutional LSTM, and bi-directional LSTM models. They found that bi-directional LSTM gave the best results, while convolutional LSTM gave the worst. For predicting the number of new and recovered cases for six countries—Italy, Spain, France, China, the United States, and Australia—A. Zeroual et al. [[**19**](https://www.mdpi.com/1660-4601/20/11/5943#B19-ijerph-20-05943)] presented a comparison of five deep learning models (basic RNN, LSTM, Bidirectional-LSTM, gated recurrent units (GRUs), and VariationalAutoEncoder (VAE)). Their results demonstrated the VAE’s superior performance over the other methods and the deep learning models’ promising potential in forecasting COVID-19 cases. A. Hernandez-Matamoros, et al. [[**20**](https://www.mdpi.com/1660-4601/20/11/5943#B20-ijerph-20-05943)] made a way to run and analyze the ARIMA model for 145 countries spread out over 6 continents. The goal was to link countries in the same area so that the spread of the virus could be predicted. S. Chae et al. [[**21**](https://www.mdpi.com/1660-4601/20/11/5943#B21-ijerph-20-05943)] previously compared DNN and LSTM models to the auto-regressive integrated moving average (ARIMA) for the prediction of infectious diseases, and the findings showed that DNN and LSTM models outperformed ARIMA. [**Table 1**](https://www.mdpi.com/1660-4601/20/11/5943#table_body_display_ijerph-20-05943-t001) summarizes a quick comparison study and review of various deep learning models. This validates our choice to use deep learning techniques, which have been shown to be precise and efficient in predicting COVID-19 outbreaks. The study’s main contribution is the creation of a standardized, data-driven, accurate, and generic COVID-19 outbreak prediction technique. In the part after this one, we will discuss the models’ historical contexts. It is discovered that machine learning is a useful method for simulating the COVID-19 epidemic due to its highly complex structure and variance in behavior from country to country.

## Materials and Methods

#### 3.1. Data Set Description and Data Preparation

In this study, real-time observations are incorporated for up-to-date analysis and for the prediction of COVID-19 results. There are 2 types of datasets used in this study: (1) The global dataset from January 2020 to August 2021, which is being gathered from covid19india.org and is available via the online source Kaggle. Three separate time series datasets were gathered, including confirmed, recovered, and death cases. It also includes information such as the name of the province, country, and the number of cases by date. (2) Second, data for COVID-19 is collected from Indiastathealth.org, which includes parameters like confirmed cases by date, confirmed deaths, vaccination, policy responses, mobility, generic, hospitalizations, discharged or migrated, and the number of Asha deaths.

The panda profiling feature is used for Exploratory Data Analysis, and the interaction between 3 important features is shown in [**Figure 4**](https://www.mdpi.com/1660-4601/20/11/5943#fig_body_display_ijerph-20-05943-f004). Correlation factors like Spearman’s (ρ), Pearson’s (r), and Kendall’s (τ) are used to do the statistical analysis, which is presented in [**Figure 4**](https://www.mdpi.com/1660-4601/20/11/5943#fig_body_display_ijerph-20-05943-f004)a–c. Pearson’s correlation specifies the linear correlation while testing the similarities in the ordering of the data; when it is ranked by quantity Kendall’s correlation is used, which is highly reflected in the below variables. Also, Spearman’s correlation specifies the strength and direction of the association between ranked variables. From the above correlation factors, we can see that the variables are highly correlated linearly and in terms of quantity, they have similarities. The scatter diagrams for confirmed, active, and cured cases and deaths are depicted in [**Figure 5**](https://www.mdpi.com/1660-4601/20/11/5943#fig_body_display_ijerph-20-05943-f005).

**Figure 4.** (**a**) Spearman’s, (**b**) Pearson’s, and (**c**) Kendall’s correlations.

**Figure 5.** Scatter diagrams of confirmed, active, and cured cases and deaths.

Real-world datasets can be unreliable, and studying raw data might lead to incorrect conclusions. As a result, data must be pre-processed before being analyzed. There are a variety of pre-processing approaches available to deal with messy data in order to ensure consistency in knowledge discovery data [[**34**](https://www.mdpi.com/1660-4601/20/11/5943#B34-ijerph-20-05943)]. Multiple files’ properties can be concatenated to make a single file in a usable format [[**35**](https://www.mdpi.com/1660-4601/20/11/5943#B35-ijerph-20-05943)]. Data reduction procedures can be used to reduce the number of attributes by reducing redundancy in the dataset [[**36**](https://www.mdpi.com/1660-4601/20/11/5943#B36-ijerph-20-05943),[**37**](https://www.mdpi.com/1660-4601/20/11/5943#B37-ijerph-20-05943)].

The support and resistance levels are determined by technical analysis indicators. The support level indicates when the number of cases has decreased, and the resistance level indicates when it has risen [[**38**](https://www.mdpi.com/1660-4601/20/11/5943#B38-ijerph-20-05943)]. They aid in recognizing both upward and downward trends. To extract noise-free features from the existing raw features, the TA-lib software is used to reveal significant patterns. The indications utilized to execute feature engineering are listed here [[**39**](https://www.mdpi.com/1660-4601/20/11/5943#B39-ijerph-20-05943)]:

SMA (simple moving average): The average of a chosen range of cases is determined by the number of periods in that range.

Weighted moving average (WMA): The formula for calculating the weighted moving average (WMA) is to multiply the current cases by the corresponding weights and then add the results.

The exponential moving average (EMA): It is a sort of weighted moving average that emphasizes current case data, but the rate of decline between one case and its prior case is not linear but rather exponential.

EMA=Cases(t)∗k+EMA(y)∗(1−k),EMA=Casest∗k+EMAy∗1−k,

(1)

where t is today, y is yesterday, N is the number of days in EMA (i.e., the smoothening range), and k = 2/(N + 1)

After feature engineering, certain undesirable features were deleted using linear interpolation. The feature selection method is depicted in [**Figure 6**](https://www.mdpi.com/1660-4601/20/11/5943#fig_body_display_ijerph-20-05943-f006) where missing value imputation was applied when necessary after dividing all features into blocks for each set of periods (smoothing range) and technical indicators. Then, using a random forest regressor as an estimator, each set is fitted with Recursive Feature Elimination, Cross-Validated (RFECV). The most significant characteristic is chosen by RFECV after the features are ranked. At each iteration, the step size is reduced to 0.6, removing 60% of the least significant features. From each block, the feature with the highest rank is selected. The Variance Inflation Factor (VIF) is used to leave out characteristics that are strongly linked to other independent characteristics.

**Figure 6.** Feature Selection Process.

After preparing the dataset, all date columns are converted to data-time objects to group the data by ‘Date’ to find the cumulative sum of cases. The description and time series plotting of 3 variables, cases, deaths, and cured, are shown in [**Table 2**](https://www.mdpi.com/1660-4601/20/11/5943#table_body_display_ijerph-20-05943-t002) and [**Figure 7**](https://www.mdpi.com/1660-4601/20/11/5943#fig_body_display_ijerph-20-05943-f007), respectively. Then re-sampling the number of cases is done on a monthly and weekly basis, and is shown in [**Figure 8**](https://www.mdpi.com/1660-4601/20/11/5943#fig_body_display_ijerph-20-05943-f008) and [**Figure 9**](https://www.mdpi.com/1660-4601/20/11/5943#fig_body_display_ijerph-20-05943-f009). Also, the time series visualization for 200 days is shown in [**Figure 10**](https://www.mdpi.com/1660-4601/20/11/5943#fig_body_display_ijerph-20-05943-f010). Then we set up helper functions for forecasting, extracting the last n days from the time series and plotting the last n days from the time series. Afterward, multivariate data is prepared with a Keras format series, which is used to convert the numpy series into a 3D form, and then data splitting is done for training and testing.

**Figure 7.** Time series plotting of three individual features.