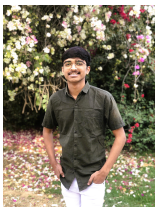
Exploratory Data Analysis on

COVID 19 IMPACT

by

Group 8

Akshat Kadia
ID: 202203029
Course:
BTech(MnC)



Pritesh Vadher
ID: 202203043
Course:
BTech(MnC)

Vivek Chaudhari
ID: 202201294
Course: BTech(ICT)

Course Code: IT 462
Semester: Autumn 2024

Under the guidance of

Dr. Gopinath Panda

Github-Repo

Dhirubhai Ambani Institute of Information and Communication Technology

December 3, 2024

Acknowledgment

I am writing this letter to express my heartfelt gratitude for your guidance and support throughout my project titled **"COVID 19 IMPACT"** Your invaluable assistance has played a pivotal role in shaping the successful completion of this endeavor.

I am incredibly fortunate to have had the opportunity to work under your mentorship. Your expertise, encouragement, and willingness to share your knowledge have been instrumental in elevating the quality and scope of my project. Your constructive feedback and insightful suggestions have helped me overcome challenges and develop a deeper understanding of the subject matter.

Furthermore, I would like to thank the entire team at Dhirubhai Ambani Institute of Information and Communication Technology for fostering an environment of collaboration and innovation. The resources and facilities provided have been crucial in conducting comprehensive research and analysis. I would also like to express my gratitude to my peers and colleagues who have been supportive throughout this journey. Their valuable input and camaraderie have been a constant source of motivation.

Completing this project has been a tremendous learning experience. I am confident that the knowledge and skills acquired during this endeavor will be a solid foundation for my future endeavors.

Sincerely,
Group 8,
Akshat Kadia, 202203029
Pritesh Vadher, 202203043
Vivek Chaudhari, 202201294

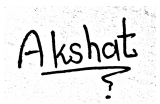
DECLARATION

We, Group 8 Akshat Kadia 202203029, Pritesh Vadher 202203043, Vivek Chaudhari 202201294 now declare that the EDA project work presented in this report is our original work and has not been submitted for any other academic degree. All the sources cited in this report have been appropriately referenced.

We acknowledge that the data utilized in this project has been sourced from Catalog.Data.gov. We affirm that we have complied with the terms and conditions specified on the website for accessing and using the dataset. We hereby confirm that the dataset employed in this project is accurate and authentic to the best of our knowledge.

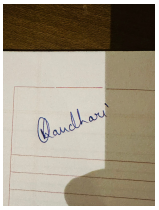
We acknowledge that we have received no external help or assistance in conducting this project except for the guidance provided by our mentor, Prof. Gopinath Panda. We declare no conflict of interest in conducting this EDA project.

We have now signed the declaration statement and confirmed the submission of this report on 2 December 2024.



Akshat Kadia
ID: 202203029
Course:
BTech(MnC)

Pritesh Vadher
ID: 202203043
Course:
BTech(MnC)



Vivek Chaudhari
ID: 202201294
Course: BTech(ICT)

CERTIFICATE

This is to certify that Group 8, comprising Akshat Kadia (202203029), Pritesh Vadher (202203043), and Vivek Chaudhari (202201294), has successfully completed an exploratory data analysis (EDA) project on the "COVID-19 Dataset." The project reflects the students' skills in data analysis, data preprocessing, and visualization techniques.

The project was undertaken as part of their coursework, and it involved a thorough examination of various aspects of the COVID-19 dataset. The students employed advanced techniques in data cleaning, transformation, and exploratory analysis, ensuring that the dataset was accurate, reliable, and ready for in-depth investigation. The report includes various visualizations, including bar charts, heat maps, and scatter plots, which effectively convey insights from the data, allowing for meaningful conclusions to be drawn.

Throughout the course of the project, the team demonstrated strong analytical skills, working collaboratively to clean and explore the data. They effectively communicated their findings through detailed analysis and visualizations, which are presented clearly in the final report. This project showcases their ability to handle complex datasets, uncover patterns, and generate insights using modern data science tools.

We recognize that the project was completed under the guidance of the course instructor, Prof. Gopinath Panda, who provided valuable support and direction throughout the project's execution. His guidance was instrumental in refining the methodologies used and ensuring the project met the required academic standards.

This certificate is issued to acknowledge the successful completion of the EDA project on the COVID-19 Dataset, demonstrating the analytical capabilities, technical proficiency, and understanding of data science exhibited by Group 8. The project not only highlights the students' technical skills but also their ability to apply those skills in a real-world context, drawing meaningful insights from publicly available data.

Signed,
Dr. Gopinath Panda,
IT 462 Course Instructor
Dhirubhai Ambani Institute of Information and Communication Technology
Gandhinagar, Gujarat, INDIA.

December 3, 2024

Contents

List of Figures 5

1 Introduction 1
- 1.1 Your Project idea . . . 1
- 1.2 Data Collection . . . 1
- 1.3 Dataset Description . . . 2
  - 1.3.1 Use Cases . . . 5
- 1.4 Packages required . . . 5

2 Data Cleaning 8
  - 2.0.1 Data Loading . . . 8
  - 2.0.2 Observing Data . . . 8
- 2.1 Data Encoding . . . 11
  - 2.1.1 Step 1: Encoding the `tests_units` Column . . . 11
  - 2.1.2 Step 2: Handling the `date` Column . . . 12
  - 2.1.3 Step 3: Verification and Sample Inspection . . . 12
  - 2.1.4 Key Takeaways . . . 12
- 2.2 Missing Data Analysis . . . 13
  - 2.2.1 Bar Plot of Missing Values . . . 13
  - 2.2.2 Matrix Plot of Missing Values . . . 14
  - 2.2.3 Dendrogram of Missing Values . . . 14
- 2.3 Imputation . . . 14
  - 2.3.1 Initial Missing Data Overview . . . 15
  - 2.3.2 Dropping Columns with Excessive Missing Data . . . 15
  - 2.3.3 Imputing Missing Data in Remaining Columns . . . 16
  - 2.3.4 Verification of Imputation . . . 17
- 2.4 Outlier Detection and Treatment . . . 18
  - 2.4.1 Identifying Outliers Using Boxplots . . . 19
  - 2.4.2 Using the Interquartile Range (IQR) for Outlier Detection . . . 19
  - 2.4.3 Decision to Retain Outliers . . . 20
  - 2.4.4 Visual Verification After Removing Outliers . . . 20

3 Visualization 21
- 3.1 Univariate analysis . . . 21
  - 3.1.1 Histogram . . . 21
  - 3.1.2 Boxplot . . . 21

3.1.3 Violin Plot . . . 22
3.1.4 Kernel Density Estimation (KDE) Plot . . . 22
3.1.5 Insights from Univariate Analysis . . . 22
3.2 Multivariate analysis . . . 24
3.2.1 Heatmap . . . 24
3.2.2 Line Plot . . . 25
3.2.3 Scatter Plot . . . 25
3.2.4 Pair Plot . . . 27
3.2.5 Insights from Multivariate Analysis . . . 27

4 Feature Engineering **29**
4.1 Feature selection . . . 29
4.1.1 Correlation Analysis . . . 29
4.1.2 Feature Importance Using Random Forest . . . 30
4.1.3 Feature Selection Using K best method . . . 30

5 Model fitting **31**
5.1 ARIMA a Time Series Model . . . 31
5.1.1 Process . . . 31
5.1.2 Testing for Stationarity (ADF Test) . . . 31
5.1.3 Autocorrelation and Partial Autocorrelation Analysis . . . 32
5.1.4 ARIMA Parameters: . . . 32
5.1.5 Model Diagnostics . . . 33
5.1.6 Forecasting and Fitted Values . . . 34
5.1.7 Error Metrics for Model Evaluation . . . 34
5.2 LSTM . . . 34
5.2.1 Data Preparation . . . 35
5.2.2 Building the LSTM Model: . . . 35
5.3 XGBoost . . . 36
5.3.1 Results . . . 37
5.3.2 Forecasted Value . . . 38
5.4 ML algorithms . . . 38

6 Conclusion & future scope **39**
6.1 Findings/observations . . . 39
6.2 Challenges . . . 39
6.3 Future plan . . . 40
6.3.1 Public Health Decision Support . . . 40
6.3.2 Real-Time Forecasting Systems Automation: . . . 40
6.3.3 Hybrid Model Development Combine strengths of multiple models: . . . 41

List of Figures

2.1 Datatype of columns . . . 9
2.2 The bar plot figure depicts the distribution of missing values across the columns of the dataset. The darker bars represent the columns with fewer missing values, while the lighter bars indicate the columns with a higher proportion of missing values. This allows for easy identification of which columns need further attention. . . . 13
2.3 The above figure depicts the Matrix plot of the Raw Data. The blank nodes in the Matrix represent the missing data. . . . 14
2.4 The above figure shows the Dendrogram plot of Raw Data. Dendrogram is a form of tree that shows the closeness/similarity of the columns. The closer two columns are placed in the Dendro gram, the more they are similar to each other. . . . 15
2.5 The output gives us the percentage of missing data per column initially, which allows us to assess the severity of missingness. . . . 16
2.6 This ensures that only columns with sufficient data are retained for further analysis. . . . 16
2.7 After Imputation, we can see that missing data has been handled. . . . 18
2.8 The above figure of box plots depicts the distribution of COVID-19 cases, tests, vaccinations, and deaths. Some metrics show a wide range of values, while others are more concentrated. . . . 19
2.9 After Removing Outliers . . . 20

3.1 total_cases . . . 23
3.2 total_tests . . . 23
3.3 people_vaccinated . . . 23
3.4 new_tests . . . 23
3.5 total_vaccinations . . . 24
3.6 total_deaths . . . 24
3.7 Caption . . . 25
3.8 total_cases . . . 26
3.9 new_cases . . . 26
3.10 total_tests . . . 26
3.11 total_vaccinations . . . 26
3.12 total_deaths . . . 27
3.13 new_deaths . . . 27
3.14 The above figure depicts the pairplot of some features. . . . 28

5.1 X represents days and y represents new deaths . . . 36
5.2 These are the forecasted new deaths predicted by our model . . . 36

Abstract

This project focuses on analyzing the impact of the COVID-19 pandemic on various socio-economic factors, utilizing an extensive COVID-19 dataset to detect trends, patterns. The study aims to uncover the effects of the pandemic on public health, economic activity, and daily life across different regions and time periods. The dataset, which includes information on infection rates, mortality rates, vaccination progress, and government policies, was subjected to rigorous data preprocessing, including missing value imputation, normalization, and outlier detection.

The project employs statistics to explore univariate and multivariate relationships within the data. Time-series forecasting was used to model the new deaths, predict future trends, and understand the factors influencing pandemic outcomes. The models performance was evaluated using various metrics such as accuracy, R-squared, and mean-squared error.

By identifying outliers and anomalies, the study aims to highlight regions or periods where the pandemic had unusual impacts, providing insights into how various interventions and factors may have altered the trajectory of the crisis. The results from this analysis can help inform future policy decisions, improve pandemic preparedness, and contribute to the broader understanding of COVID-19's multifaceted impact.

This project demonstrates the value of data science in analyzing public health crises, showcasing how comprehensive data analysis can support decision-making and improve response strategies during times of global uncertainty.

Chapter 1. Introduction

1.1 Your Project idea

The project focuses on forecasting the number of COVID-19-related deaths (new deaths) using historical data. This prediction is critical for making informed decisions about healthcare resource allocation, government interventions, and public health strategies. The project explores multiple machine learning models, including **XGBoost**, **Long Short-Term Memory (LSTM)** networks, and **ARIMA**, to predict the future trends of 'new_deaths' based on time-series data.

The main concept behind the project is to create more robust, data-driven models to predict COVID-19 mortality trends, which can further extend to public health outcomes forecasting in the near future. Real-time issues such as inconsistent data collection, feature selection complexity, and the nonlinear relationships existing in COVID-19 data would be addressed through the project.

This evaluates a range of algorithms for machine learning based on performance comparison and picks the model that best serves to predict good forecasting. This work can help improve the management of this pandemic and increase preparedness in case of a health crisis by using predictive analytics.

1.2 Data Collection

For this project, the data was sourced from Our World in Data, a comprehensive and widely used resource for COVID-19 statistics. The website provides a global dataset that includes various metrics such as the number of confirmed cases, recoveries, deaths, and other key pandemic-related indicators, collected from multiple countries and regions around the world.

Initially, the dataset was vast, containing more than **3,50,000** rows of data spanning numerous countries and regions. Given the size and complexity of this data, preprocessing it for global predictions proved to be a challenge. The extensive nature of the dataset made it difficult to manage and filter effectively for specific countries, especially when dealing with inconsistencies in the data reporting formats across regions.

To simplify the analysis, we chose to focus on only
textbfIndia in the predictive modeling part. We filtered the dataset such that we kept only the relevant data for India, which reduced the size and complexity of the dataset, allowing it to be more manageable for preprocessing and model building. The decision was of great importance because it allowed us to focus only on accurate predictions for the Indian context, which formed the focal point of this project.

1.3 Dataset Description

This dataset provides a comprehensive overview of COVID-19 cases and related factors in India. It encompasses a wide range of variables, enabling in-depth analysis of the pandemic's impact and the country's response.

- **iso_code:** The ISO country code for each region. This is a unique identifier for each location in the dataset.
- **continent:** The continent to which the country or region belongs. Examples include Asia, Europe, North America, and others.
- **location:** The specific country or region name where the data was recorded.
- **date:** The date on which the data was reported, formatted as YYYY-MM-DD.
- **total_cases:** The cumulative number of confirmed COVID-19 cases in the location by the date provided.
- **new_cases:** The number of new confirmed COVID-19 cases reported on the given date.
- **new_cases_smoothed:** The smoothed number of new cases, calculated to reduce fluctuations and provide a clearer trend.
- **total_deaths:** The total number of COVID-19 deaths recorded in the location by the specified date.
- **new_deaths:** The number of new deaths attributed to COVID-19 reported on the given date.
- **new_deaths_smoothed:** The smoothed number of new deaths, helping to remove daily fluctuations in reporting.
- **total_cases_per_million:** The total number of COVID-19 cases per million people in the location.
- **new_cases_per_million:** The number of new cases per million people on the given date.
- **new_cases_smoothed_per_million:** The smoothed number of new cases per million people.
- **total_deaths_per_million:** The total number of deaths per million people in the location.
- **new_deaths_per_million:** The number of new deaths per million people on the given date.
- **new_deaths_smoothed_per_million:** The smoothed number of new deaths per million people.
- **reproduction_rate:** The reproduction rate (R0) representing the average number of people an infected individual will transmit the virus to.
- **icu_patients:** The total number of patients in Intensive Care Units (ICU) for COVID-19 in the location.

- **icu_patients_per_million:** The number of ICU patients per million people.
- **hosp_patients:** The total number of hospitalized patients for COVID-19.
- **hosp_patients_per_million:** The number of hospitalized patients per million people.
- **weekly_icu_admissions:** The number of ICU admissions due to COVID-19 in a given week.
- **weekly_icu_admissions_per_million:** The number of ICU admissions per million people in a week.
- **weekly_hosp_admissions:** The number of hospital admissions due to COVID-19 in a given week.
- **weekly_hosp_admissions_per_million:** The number of hospital admissions per million people in a week.
- **total_tests:** The total number of COVID-19 tests conducted in the location.
- **new_tests:** The number of new tests conducted on the given date.
- **total_tests_per_thousand:** The total number of tests per thousand people in the location.
- **new_tests_per_thousand:** The number of new tests per thousand people on the given date.
- **new_tests_smoothed:** The smoothed number of new tests, adjusting for fluctuations in testing activity.
- **new_tests_smoothed_per_thousand:** The smoothed number of new tests per thousand people.
- **positive_rate:** The percentage of tests that returned positive results.
- **tests_per_case:** The number of tests conducted per confirmed case.
- **tests_units:** The unit of measurement used for the test counts (e.g., tests, samples).
- **total_vaccinations:** The total number of COVID-19 vaccinations administered in the location.
- **people_vaccinated:** The number of individuals who have received at least one dose of the COVID-19 vaccine.
- **people_fully_vaccinated:** The number of individuals who have received both doses (or the full course) of the COVID-19 vaccine.
- **total_boosters:** The total number of COVID-19 booster doses administered.
- **new_vaccinations:** The number of new COVID-19 vaccinations administered on the given date.
- **new_vaccinations_smoothed:** The smoothed number of new vaccinations administered, adjusting for daily variations.
- **total_vaccinations_per_hundred:** The total number of vaccinations per hundred people.
- **people_vaccinated_per_hundred:** The number of people vaccinated per hundred people in the population.

- **people_fully_vaccinated_per_hundred:** The number of people fully vaccinated per hundred people.
- **total_boosters_per_hundred:** The number of booster doses per hundred people.
- **new_vaccinations_smoothed_per_million:** The smoothed number of new vaccinations per million people.
- **new_people_vaccinated_smoothed:** The number of newly vaccinated people, smoothed for daily variations.
- **new_people_vaccinated_smoothed_per_hundred:** The number of newly vaccinated people per hundred, smoothed for fluctuations.
- **stringency_index:** A measure of government restrictions, with higher values indicating more stringent measures.
- **population_density:** The population density (people per square kilometer) of the location.
- **median_age:** The median age of the population in the location.
- **aged_65_older:** The percentage of the population aged 65 or older.
- **aged_70_older:** The percentage of the population aged 70 or older.
- **gdp_per_capita:** The Gross Domestic Product per capita, which provides an indicator of the country's economic status.
- **extreme_poverty:** The percentage of the population living in extreme poverty (earning less than $1.90 per day).
- **cardiovasc_death_rate:** The cardiovascular death rate, an important factor in assessing health outcomes.
- **diabetes_prevalence:** The prevalence of diabetes in the population.
- **female_smokers:** The percentage of female smokers in the population.
- **male_smokers:** The percentage of male smokers in the population.
- **handwashing_facilities:** The percentage of the population with access to handwashing facilities.
- **hospital_beds_per_thousand:** The number of hospital beds available per thousand people.
- **life_expectancy:** The average life expectancy in the location.
- **human_development_index:** A composite index of human development, which includes factors like life expectancy, education, and income.
- **population:** The total population of the location.
- **excess_mortality_cumulative_absolute:** The cumulative excess mortality, representing the number of deaths beyond what would be expected based on historical data.

- **excess_mortality_cumulative:** The cumulative excess mortality as a percentage of expected deaths.
- **excess_mortality:** The rate of excess mortality compared to expected values.
- **excess_mortality_cumulative_per_million:** The cumulative excess mortality per million people.

1.3.1 Use Cases

Epidemiological Analysis:

- Tracking the spread and severity of the virus over time.
- Identifying trends and patterns in case and death rates.

Public Health Policy Evaluation:

- Assessing the effectiveness of public health interventions, such as lockdowns and vaccination campaigns.
- Identifying areas where additional resources and support are needed.

Socioeconomic Impact Assessment:

- Analyzing the impact of the pandemic on various socioeconomic indicators.
- Understanding how factors like smoking, handwashing facilities, and healthcare infrastructure influence COVID-19 outcomes.

Predictive Modeling:

- Developing models to forecast future trends in cases and deaths.

By leveraging this dataset, researchers and policymakers can gain valuable insights into the COVID-19 pandemic in India and inform evidence-based decision-making.

1.4 Packages required

For the analysis of the COVID-19 impact dataset, several Python packages were used for data manipulation, visualization, statistical modeling, and machine learning tasks. Below is a list of the key packages and their purpose in the project:

1. **pandas**
   Used for data manipulation and analysis. It provides data structures like DataFrame, which makes it easy to work with structured data.
2. **numpy**
   Essential for numerical operations and handling arrays. It is used for performing mathematical calculations on large datasets.

3. **matplotlib.pyplot**
   A plotting library used for creating static, animated, and interactive visualizations in Python. It was used to generate basic plots and graphs.

4. **seaborn**
   Built on top of `matplotlib`, seaborn is used for more advanced visualizations, particularly for statistical graphics.

5. **missingno**
   A specialized library for visualizing missing data. It provides convenient methods for detecting and visualizing missing values.

6. **warnings**
   This library is used to suppress warnings during code execution, ensuring that the output remains clean.

7. **sklearn.preprocessing**
   Contains a variety of tools for preprocessing data, such as `OneHotEncoder`, `LabelEncoder`, `StandardScaler`, and `MinMaxScaler`, which are used for encoding categorical variables and scaling numerical data.

8. **sklearn.model_selection**
   Provides tools for splitting the data into training and test sets. Specifically, `train_test_split` was used for this purpose.

9. **sklearn.metrics**
   Includes metrics for model evaluation such as `mean_absolute_error`, `mean_squared_error`, `r2_score`, and `accuracy_score`, which are essential for assessing the performance of machine learning models.

10. **sklearn.feature_selection**
    Provides methods like `SelectKBest` and `mutual_info_regression` for feature selection to identify the most relevant features in the dataset.

11. **sklearn.ensemble**
    Contains `RandomForestRegressor`, an ensemble learning method used for regression tasks in this project.

12. **statsmodels.tsa.arima.model**
    Provides the `ARIMA` model for time series forecasting.

13. **statsmodels.tsa.stattools**
    Includes statistical tests like `adfuller` for checking stationarity in time series data.

14. **statsmodels.graphics.tsaplots**
    Used to create autocorrelation and partial autocorrelation plots, which help in understanding the time series data's behavior.

15. **xgboost**
    A powerful machine learning library used for training an XGBoost model for prediction tasks.

16. **tensorflow.keras.models**
    A deep learning framework used for creating neural network models. The `Sequential` model was used for building an LSTM (Long Short-Term Memory) network.

17. **tensorflow.keras.layers**
    Includes layers like `LSTM`, `Dense`, and `Dropout`, which were used for designing and training the LSTM model.

Chapter 2. Data Cleaning

In this section, we perform various data cleaning tasks to ensure that the dataset is well-prepared for analysis.

2.0.1 Data Loading

The dataset was loaded from an external source using the `pandas` library. We read the CSV file and then checked the first 10 rows of the dataset to gain an initial understanding.

```
df_india = pd.read_csv('https://raw.githubusercontent.com/akshat-11004/EDA_Project/main/
df_india.sample(10)
```

To get an overview of the columns and the general shape of the dataset, we used the following commands:

```
df_india.columns
df_india.shape
```

The dataset consists of 1390 rows and 67 columns.

2.0.2 Observing Data

After loading the data, we checked the dataset's general information using the `info()` method to understand the data types and the presence of any missing values.

```
df_india.info()
```

Dropping Unnecessary Columns

The columns `iso_code`, `continent`, and `location` were removed as they do not contribute to the analysis of the dataset's numerical and temporal attributes. These columns primarily served as identifiers and geographical labels, which are redundant for our analysis.

```
df_india.drop(columns=['iso_code', 'continent', 'location'], inplace=True)
```

#	Column	Non-Null Count	Dtype
0	iso_code	1390 non-null	object
1	continent	1390 non-null	object
2	location	1390 non-null	object
3	date	1390 non-null	object
4	total_cases	1358 non-null	float64
5	new_cases	1382 non-null	float64
6	new_cases_smoothed	1377 non-null	float64
7	total_deaths	1315 non-null	float64
8	new_deaths	1385 non-null	float64
9	new_deaths_smoothed	1380 non-null	float64
10	total_cases_per_million	1358 non-null	float64
11	new_cases_per_million	1382 non-null	float64
12	new_cases_smoothed_per_million	1377 non-null	float64
13	total_deaths_per_million	1315 non-null	float64
14	new_deaths_per_million	1385 non-null	float64
15	new_deaths_smoothed_per_million	1380 non-null	float64
16	reproduction_rate	1024 non-null	float64
17	icu_patients	0 non-null	float64
18	icu_patients_per_million	0 non-null	float64
19	hosp_patients	0 non-null	float64
20	hosp_patients_per_million	0 non-null	float64
21	weekly_icu_admissions	0 non-null	float64
22	weekly_icu_admissions_per_million	0 non-null	float64
23	weekly_hosp_admissions	0 non-null	float64
24	weekly_hosp_admissions_per_million	0 non-null	float64
25	total_tests	810 non-null	float64
26	new_tests	795 non-null	float64
27	total_tests_per_thousand	810 non-null	float64
28	new_tests_per_thousand	795 non-null	float64
29	new_tests_smoothed	824 non-null	float64
30	new_tests_smoothed_per_thousand	824 non-null	float64
31	positive_rate	816 non-null	float64
32	tests_per_case	816 non-null	float64
33	tests_units	831 non-null	object
34	total_vaccinations	985 non-null	float64
35	people_vaccinated	985 non-null	float64
36	people_fully_vaccinated	955 non-null	float64
37	total_boosters	632 non-null	float64
38	new_vaccinations	961 non-null	float64
39	new_vaccinations_smoothed	1011 non-null	float64
40	total_vaccinations_per_hundred	985 non-null	float64
41	people_vaccinated_per_hundred	985 non-null	float64
42	people_fully_vaccinated_per_hundred	955 non-null	float64
43	total_boosters_per_hundred	632 non-null	float64

Figure 2.1: Datatype of columns

Rationale for Dropping Columns

- `iso_code`: Redundant as the dataset pertains to a single country (India).
- `continent`: Not relevant for country-specific analysis.

- `location`: Provides no additional value as the dataset exclusively focuses on India.

These columns were removed to streamline the dataset and focus on the relevant features for analysis.

2.1 Data Encoding

After the initial data cleaning, the dataset still contained two columns with non-numeric formats: `tests_units` (categorical) and `date` (date). These columns needed to be appropriately encoded or converted to ensure compatibility with analytical and machine learning processes. The following steps were undertaken for encoding and handling these columns.

2.1.1 Step 1: Encoding the `tests_units` Column

The `tests_units` column represents the type of unit used for testing measurements. Upon inspection, the column had the following characteristics:

Missing Values

- There were 559 missing values in the `tests_units` column (`df['tests_units'].isnull().sum()`).

Unique Categories

- The column contained only one unique category (`df['tests_units'].nunique()`), making the missing values non-discriminative for analysis.

Handling Missing Values

Since the `tests_units` column had only one category, the missing values were initially replaced with the word `Missing`. This step ensured that the dataset remained consistent and avoided errors due to null values.

```
df['tests_units'] = df['tests_units'].fillna('Missing')
```

Label Encoding

Next, the `tests_units` column was encoded using the `LabelEncoder` from the `sklearn.preprocessing` library. Label encoding is a process where each unique category in the column is assigned a unique integer value. This transformation is useful for machine learning algorithms, as they generally require numeric inputs.

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
df['tests_units'] = encoder.fit_transform(df['tests_units'])
```

Explanation of LabelEncoder

The `LabelEncoder` assigns a unique integer to each category in the `tests_units` column. The `fit_transform()` method first fits the encoder to the unique categories in the column and then transforms each value into its corresponding encoded integer. The following is the code to validate the encoding process.

```
unique_encoded_values = df['tests_units'].unique()
unique_labels = encoder.classes_

for encoded_value, label in zip(unique_encoded_values, unique_labels):
    print(f"Encoded Value: {encoded_value}, Label: {label}")
```

Output:

```
Encoded Value: 0, Label: Missing
Encoded Value: 1, Label: [Original Category] (if applicable)
```

This output shows that the missing values were successfully encoded as a distinct numerical value (0), and the original category was also encoded as an integer (e.g., 1).

2.1.2 Step 2: Handling the `date` Column

The `date` column was originally in the `object` (string) format, which is unsuitable for time-series analysis. Therefore, it was converted into a `datetime` object to facilitate temporal analysis.

```
df['date'] = pd.to_datetime(df['date'], format='%d-%m-%Y')
```

Purpose of Conversion

- Enables sorting and filtering of data by date.
- Allows extraction of specific date components such as year, month, and day for further analysis.
- Facilitates time-series visualization and forecasting tasks.

2.1.3 Step 3: Verification and Sample Inspection

After the encoding and conversion steps, the dataset was inspected to ensure that the transformations were successful. The following commands were used to verify the changes:

- `df.info()`: This command confirmed that the `tests_units` column was now numeric and the `date` column had been successfully converted to the `datetime64` format.
- `df.sample(10)`: Displayed 10 random rows to visually verify the encoded values in `tests_units` and the proper format of the `date` column.

2.1.4 Key Takeaways

- `tests_units`: The missing values were encoded as a distinct category (`Missing`) and were successfully converted into a numeric format using label encoding.
- `date`: The `date` column was transformed into a `datetime` object, making it suitable for time-based analysis and modeling.

These transformations ensured that the dataset was appropriately prepared for subsequent analysis and modeling tasks.

2.2 Missing Data Analysis

In this section, we analyze the missing values in the dataset. Visualizing the missing data helps identify patterns, understand the extent of missingness, and guide appropriate imputation strategies. We use three plots to analyze the missing data: a bar plot, a matrix plot, and a dendrogram.

2.2.1 Bar Plot of Missing Values

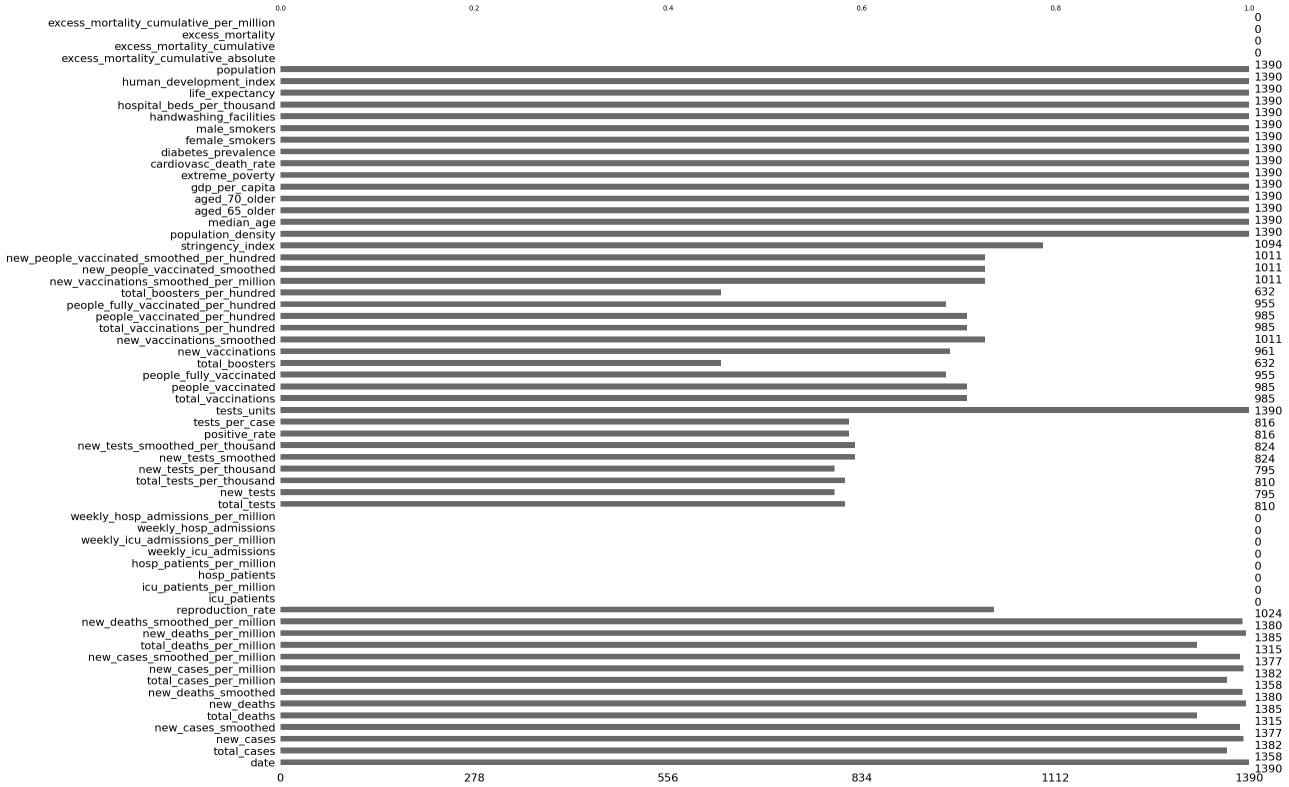


Figure 2.2: The bar plot figure depicts the distribution of missing values across the columns of the dataset. The darker bars represent the columns with fewer missing values, while the lighter bars indicate the columns with a higher proportion of missing values. This allows for easy identification of which columns need further attention.

The bar plot provides a simple visualization of the number of missing values in each column. This helps in identifying columns with a significant amount of missing data.

```
import missingno as msno
msno.bar(df)
```

This plot visualizes the presence or absence of data for each column. The x-axis represents the columns, and the y-axis shows the count of non-null values.

2.2.2 Matrix Plot of Missing Values

The matrix plot visualizes the missing values in the dataset as a heatmap. It allows us to observe any patterns or structure in the missing data. Darker cells represent missing data, while lighter cells indicate available data.

```
msno.matrix(df)
```

This plot helps identify if missingness is random or if there is any correlation between missing values across different columns.

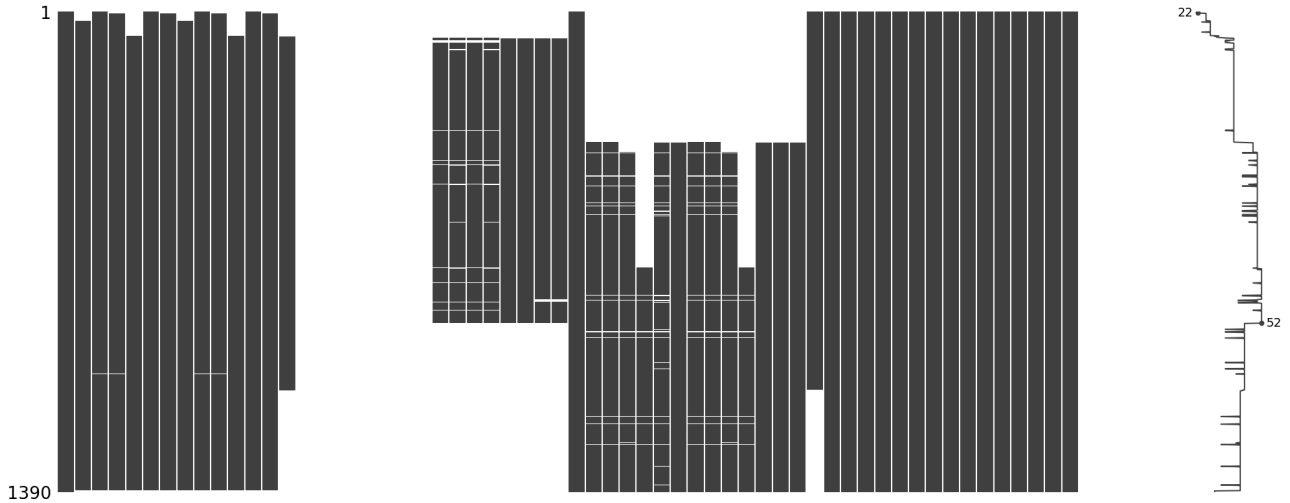


Figure 2.3: The above figure depicts the Matrix plot of the Raw Data. The blank nodes in the Matrix represent the missing data.

2.2.3 Dendrogram of Missing Values

The dendrogram shows a hierarchical clustering of columns based on the missing data pattern. It helps identify columns that exhibit similar patterns of missingness. This can be useful for deciding which columns to drop or impute.

```
msno.dendrogram(df)
```

This plot clusters columns with similar missing value patterns, which can provide insights into which features are likely to be related.

By using these visualizations, we gain a deeper understanding of the missing data, which guides us in the subsequent steps of data cleaning and imputation.

2.3 Imputation

After analyzing the missing data, it was necessary to handle the missing values to ensure the dataset is suitable for further analysis and modeling. In this step, we performed column removal and imputation to deal with missing values in the dataset.

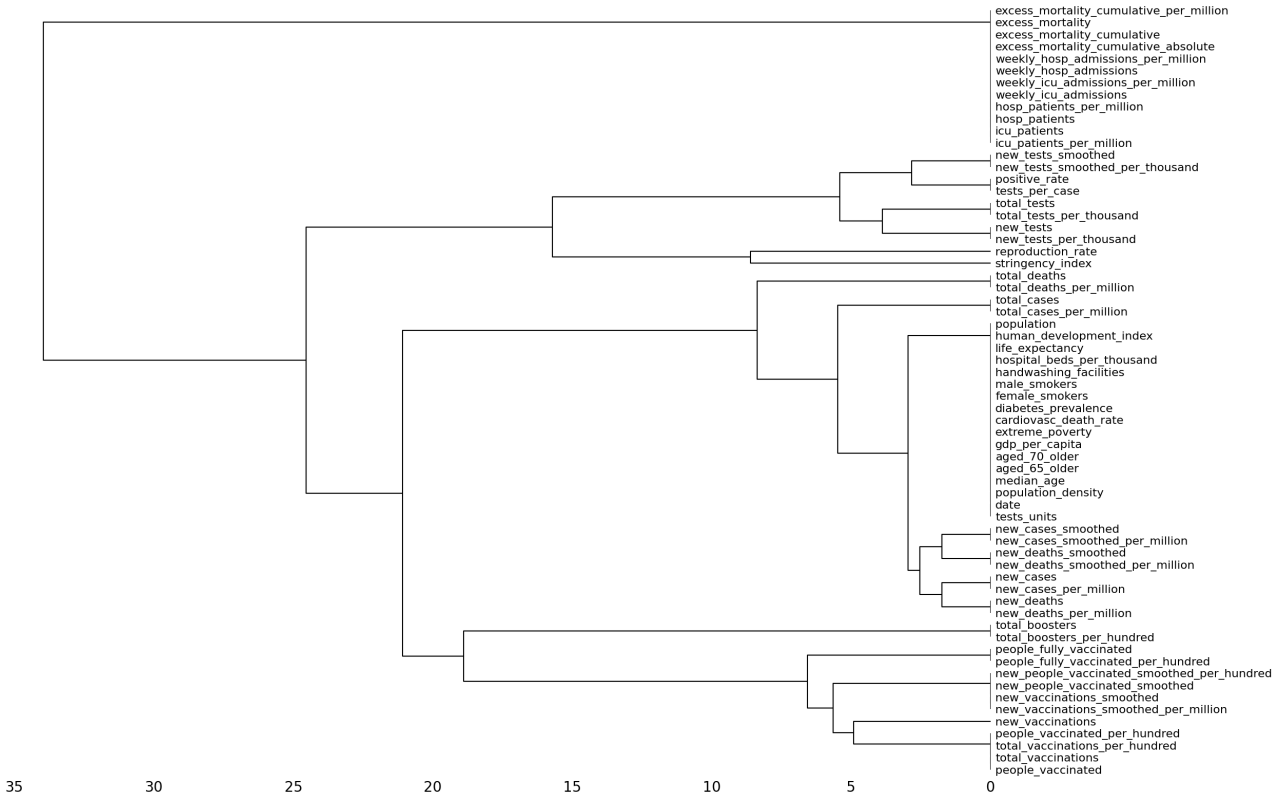


Figure 2.4: The above figure shows the Dendrogram plot of Raw Data. Dendrogram is a form of tree that shows the closeness/similarity of the columns. The closer two columns are placed in the Dendro gram, the more they are similar to each other.

2.3.1 Initial Missing Data Overview

To begin, we calculated the percentage of missing values for each column to identify the extent of missingness across the dataset. This helps in deciding which columns should be dropped entirely and which columns require imputation.

```
print(df.isnull().sum()/df.shape[0]*100)
missing_percentage = (df.isnull().sum()/df.shape[0]*100)
```

This code calculates the percentage of missing values for each column by dividing the number of missing values by the total number of rows (given by `df.shape[0]`) and multiplying by 100. The output gives us the percentage of missing data per column, which allows us to assess the severity of missingness.

2.3.2 Dropping Columns with Excessive Missing Data

Columns with more than 50% missing data were dropped from the dataset. It is generally considered inefficient to impute such a large amount of missing data, and it may not be worth retaining these columns in the analysis.

```
columns_to_drop = missing_percentage[missing_percentage > 50].index
```

```
date                                        0.000000
total_cases                                 2.302158
new_cases                                   0.575540
new_cases_smoothed                          0.935252
total_deaths                                5.395683
                                              ...
population                                  0.000000
excess_mortality_cumulative_absolute      100.000000
excess_mortality_cumulative               100.000000
excess_mortality                          100.000000
excess_mortality_cumulative_per_million   100.000000
Length: 64, dtype: float64
```

Figure 2.5: The output gives us the percentage of missing data per column initially, which allows us to assess the severity of missingness.

```
print(missing_percentage[missing_percentage > 50])
df = df.drop(columns=columns_to_drop)
```

In this code: - `missing_percentage[missing_percentage > 50].index` identifies columns where the missing data exceeds 50%. - The columns identified are then dropped from the dataframe using `df.drop()`. This ensures that only columns with sufficient data are retained for further analysis.

```
icu_patients                              100.000000
icu_patients_per_million                  100.000000
hosp_patients                             100.000000
hosp_patients_per_million                 100.000000
weekly_icu_admissions                     100.000000
weekly_icu_admissions_per_million         100.000000
weekly_hosp_admissions                    100.000000
weekly_hosp_admissions_per_million        100.000000
total_boosters                             54.532374
total_boosters_per_hundred                 54.532374
excess_mortality_cumulative_absolute      100.000000
excess_mortality_cumulative               100.000000
excess_mortality                          100.000000
excess_mortality_cumulative_per_million   100.000000
dtype: float64
```

Figure 2.6: This ensures that only columns with sufficient data are retained for further analysis.

2.3.3 Imputing Missing Data in Remaining Columns

After removing the columns with excessive missing data, we focus on imputing the missing values in the remaining columns. Different imputation strategies were applied depending on the column type.

Imputation Strategy for Numeric Columns

For numeric columns, we first applied a forward fill method (`ffill`) to propagate the last valid observation forward. This method is particularly useful when the missing data is small and can be reasonably assumed to be similar to the previous values. If any missing values remain after forward filling, we imputed them with the mean value of the respective columns.

```
numeric_columns = df.select_dtypes(exclude=['object']).columns
df[numeric_columns] = df[numeric_columns].fillna(method='ffill').fillna(df[numeric_colum
```

Here's what the code does:
- `df.select_dtypes(exclude=['object'])` selects all numeric columns in the dataframe.

- `df[numeric_columns].fillna(method='ffill')` fills any missing values in numeric columns by propagating the last valid observation forward.

-`df[numeric_columns].fillna(df[numeric_columns].mean())` then fills any remaining missing values with the mean of the respective column.

This two-step imputation process ensures that missing values in numeric columns are filled with appropriate values, minimizing the impact of missing data on subsequent analysis.

2.3.4 Verification of Imputation

After performing the imputation, we checked if any missing values remained in the dataset by running:

```
print(df.isnull().sum())
```

This code prints the number of missing values per column after the imputation process, confirming that all missing data has been handled, leaving a clean dataset for further exploration and modeling.

date	0
total_cases	0
new_cases	0
new_cases_smoothed	0
total_deaths	0
new_deaths	0
new_deaths_smoothed	0
total_cases_per_million	0
new_cases_per_million	0
new_cases_smoothed_per_million	0
total_deaths_per_million	0
new_deaths_per_million	0
new_deaths_smoothed_per_million	0
reproduction_rate	0
total_tests	0
new_tests	0
total_tests_per_thousand	0
new_tests_per_thousand	0
new_tests_smoothed	0
new_tests_smoothed_per_thousand	0
positive_rate	0
tests_per_case	0
tests_units	0
total_vaccinations	0
people_vaccinated	0
people_fully_vaccinated	0
new_vaccinations	0
new_vaccinations_smoothed	0
total_vaccinations_per_hundred	0
people_vaccinated_per_hundred	0
people_fully_vaccinated_per_hundred	0
new_vaccinations_smoothed_per_million	0
new_people_vaccinated_smoothed	0
new_people_vaccinated_smoothed_per_hundred	0
stringency_index	0
population_density	0
median_age	0
aged_65_older	0
aged_70_older	0
gdp_per_capita	0
extreme_poverty	0
cardiovasc_death_rate	0
diabetes_prevalence	0
female_smokers	0
male_smokers	0
handwashing_facilities	0
hospital_beds_per_thousand	0
life_expectancy	0
human_development_index	0

Figure 2.7: After Imputation, we can see that missing data has been handled.

2.4 Outlier Detection and Treatment

Outliers are data points that differ significantly from the majority of other data in a dataset. In the context of COVID-19 data, outliers can occur due to various reasons such as reporting anomalies, sudden spikes in cases, large-scale testing events, or significant changes in vaccination rates. Outliers

might represent key moments during the pandemic, such as new outbreaks or policy shifts, making them important to consider carefully before deciding whether to remove them.

2.4.1 Identifying Outliers Using Boxplots

To start the outlier detection process, we first visualize the data using **Boxplots** for key columns in the dataset, such as `total_cases`, `total_tests`, `people_vaccinated`, `new_tests`, `total_vaccinations`, `people_fully_vaccinated`, `new_vaccinations`, and `total_deaths`. Boxplots are a simple and effective way to identify potential outliers because they show the distribution of data, the interquartile range (IQR), and any values that fall outside the typical range (whiskers). Values outside the whiskers are considered potential outliers.

For each of these columns, we plotted boxplots to visually inspect the data for unusual spikes or drops that could indicate outliers.

The boxplot visualizations allowed us to observe how data points in these columns deviated from the median, and provided insight into the presence of extreme values.

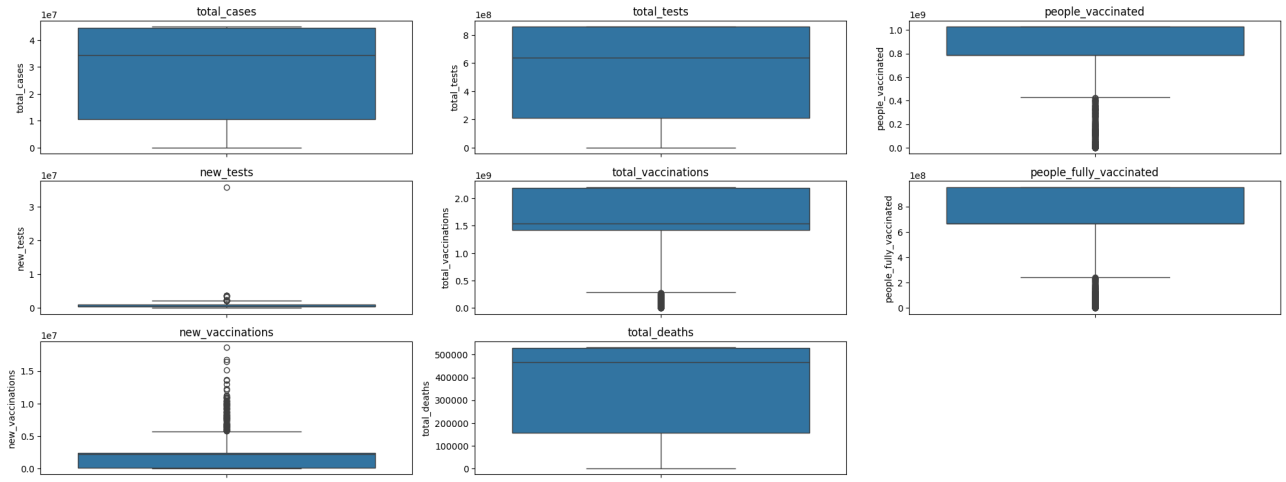


Figure 2.8: The above figure of box plots depicts the distribution of COVID-19 cases, tests, vaccinations, and deaths. Some metrics show a wide range of values, while others are more concentrated.

2.4.2 Using the Interquartile Range (IQR) for Outlier Detection

While boxplots provided a visual inspection of outliers, we also applied a more systematic approach using the **Interquartile Range (IQR)**. The IQR is a statistical measure that quantifies the spread of the middle 50% of the data, and is calculated as:

$$IQR = Q3 - Q1$$

Where:
- $Q1$ is the first quartile (25th percentile),
- $Q3$ is the third quartile (75th percentile).

Outliers are typically defined as values that fall below:

$$\text{Lower Bound} = Q1 - 1.5 \times IQR$$

or above:

$$\text{Upper Bound} = Q3 + 1.5 \times IQR$$

We calculated the IQR for each numeric column and identified values that fell outside of these bounds as outliers. This method helped us to systematically pinpoint extreme values in the dataset, based on statistical criteria rather than subjective judgment.

2.4.3 Decision to Retain Outliers

After detecting the outliers, we made a deliberate decision not to remove them. In typical data analysis, outliers are often removed to avoid distorting statistical models. However, in the context of COVID-19 data, outliers are not always errors but may represent important events such as: - Sudden surges in COVID-19 cases or deaths due to outbreaks or new variants. - Mass testing efforts leading to large increases in reported tests. - Government interventions that lead to abrupt increases in vaccinations or public health measures.

For example, if a particular day sees a dramatic increase in deaths or cases, it could reflect a significant event such as a new wave of infections or a backlog in reporting. These are crucial for understanding the course of the pandemic and should be included in the analysis, rather than discarded.

Thus, removing these outliers could distort our understanding of the true impact of COVID-19 and lead to inaccurate conclusions. Therefore, we decided to retain the outliers, allowing the dataset to maintain the integrity of the real-world phenomena it represents.

2.4.4 Visual Verification After Removing Outliers

After detecting and identifying outliers, we generated boxplots again to ensure that our decisions on handling outliers were appropriate. These visualizations provided a final check to verify that the outliers, even if they were retained, did not overly skew the data distribution.

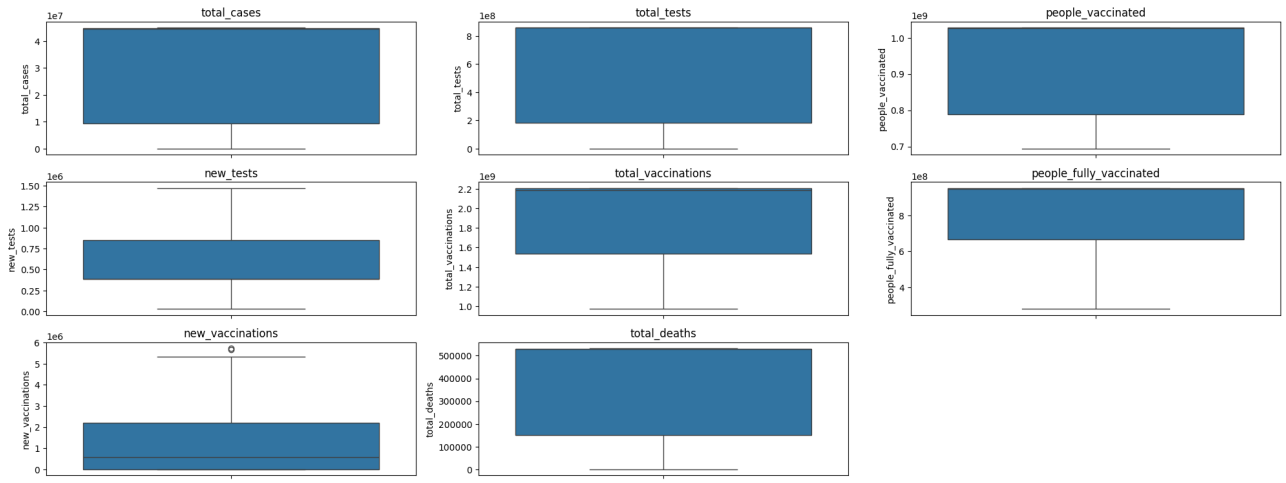


Figure 2.9: After Removing Outliers

Chapter 3. Visualization

3.1 Univariate analysis

Univariate analysis involves examining a single variable at a time to understand its distribution, central tendency, and spread. This type of analysis is particularly important for the COVID-19 dataset as it helps uncover the patterns and variability of key variables such as `total_cases`, `new_cases`, `total_deaths`, and `vaccination_rates`.

We employed four types of visualizations—histograms, boxplots, violin plots, and kernel density estimation (KDE) plots—to explore the characteristics of individual variables in the dataset. Each plot provides unique insights into the data, which are summarized below.

3.1.1 Histogram

A histogram is a graphical representation of the distribution of a dataset. It uses bins to group data into intervals, showing how often values occur within each interval. For the COVID-19 dataset, histograms are helpful in:

- Identifying the overall shape of the distribution (e.g., normal, skewed, etc).
- Spotting anomalies such as gaps or unusually high frequencies.
- Understanding the spread of data, such as daily new cases or vaccination rates.

For instance, a histogram of `daily_new_cases` may reveal periods of high infection rates or the existence of multiple peaks, indicating pandemic waves.

3.1.2 Boxplot

A boxplot, also known as a whisker plot, visually summarizes the distribution of a variable by showing:

- The median (central line of the box).
- The interquartile range (IQR), which spans the middle 50% of the data.
- Outliers, as points outside the whiskers.

In the context of COVID-19 data, boxplots can highlight:

- Variability in testing rates, case counts, or vaccination levels across different time periods.
- Extreme values or outliers that might correspond to significant events, such as mass testing campaigns or vaccination drives.

3.1.3 Violin Plot

A violin plot combines aspects of a boxplot and a kernel density plot, offering a more detailed view of the data's distribution. It shows:

- The density of data points across different ranges.
- The symmetry or skewness of the distribution.

Violin plots are especially useful for understanding the shape and variability of COVID-19 metrics, such as `daily_new_cases`. For example, they can reveal whether the majority of new cases cluster around a specific range or whether the distribution is more dispersed.

3.1.4 Kernel Density Estimation (KDE) Plot

A KDE plot provides a smooth, continuous estimate of the probability density function of a variable. Unlike histograms, KDE plots are not affected by binning, making them ideal for identifying the underlying distribution of data.

For the COVID-19 dataset, KDE plots can:

- Reveal trends in daily deaths or cases without the noise introduced by abrupt changes.
- Highlight whether a variable's distribution is unimodal (single peak) or multimodal (multiple peaks).

3.1.5 Insights from Univariate Analysis

Using these plots, we analyzed key variables such as:

- `Total Cases`: Histograms and KDE plots revealed the growth in cases over time and highlighted significant spikes during major waves of the pandemic.
- `New Tests`: Boxplots exposed outliers, potentially corresponding to mass testing campaigns.
- `Vaccination Rates`: Violin plots showed the density and spread of vaccination efforts, capturing the variability in rollout across different regions or time periods.
- `Total Deaths`: KDE plots highlighted key periods of increased mortality, providing insights into the most challenging phases of the pandemic.

Univariate analysis provides a foundation for understanding the distribution and variability of individual variables in the COVID-19 dataset. By leveraging multiple visualization techniques, we obtained a comprehensive view of the dataset, identified patterns, and detected anomalies. These insights form the basis for further analysis, including bivariate and multivariate explorations.

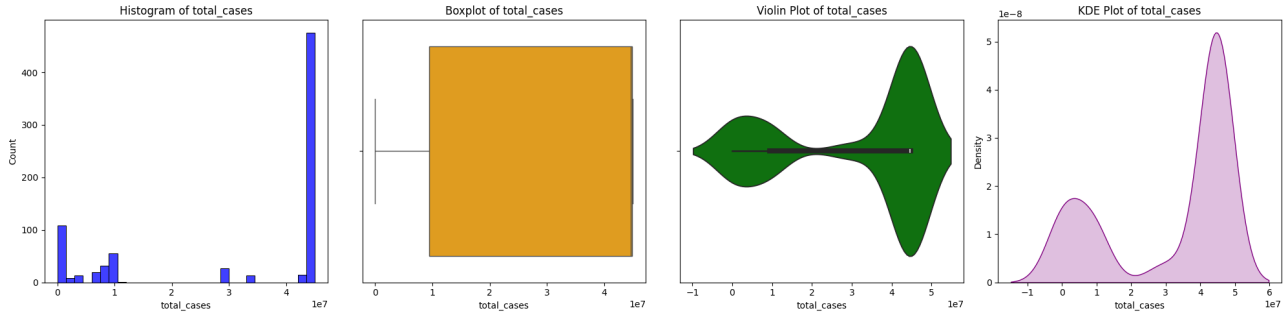


Figure 3.1: total_cases

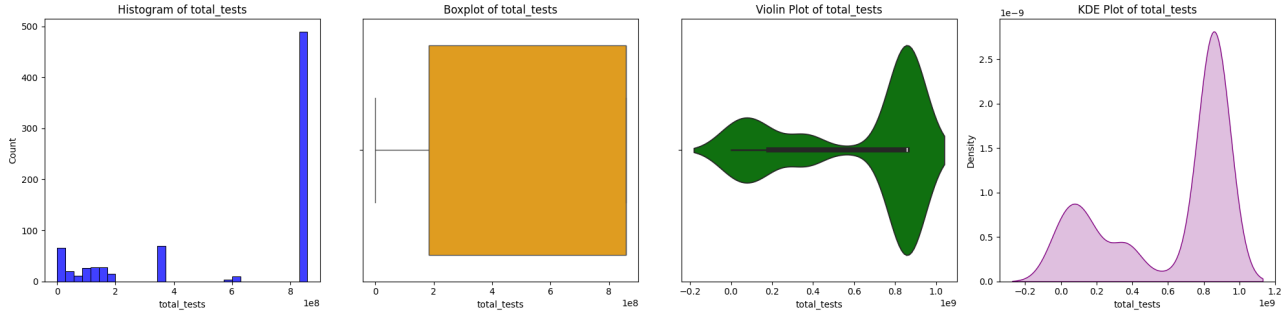


Figure 3.2: total_tests

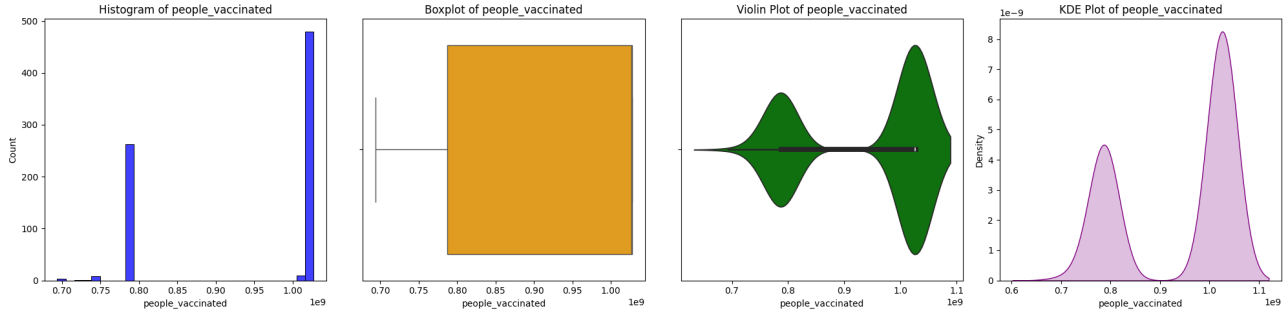


Figure 3.3: people_vaccinated

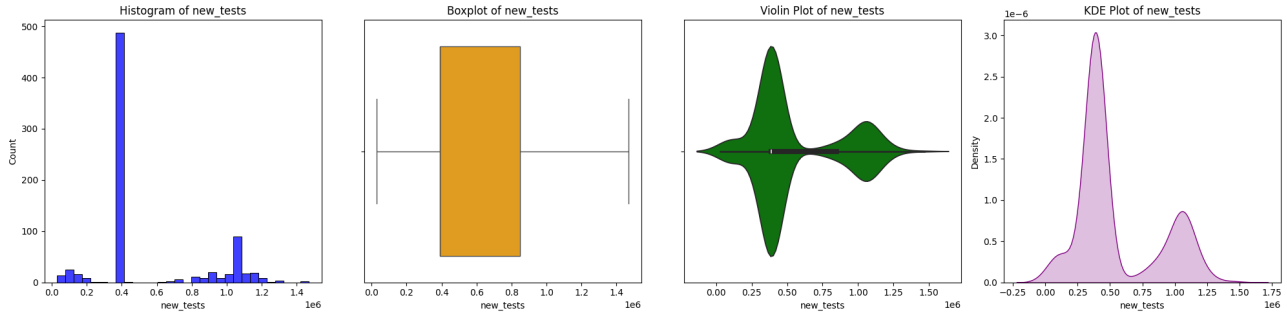


Figure 3.4: new_tests

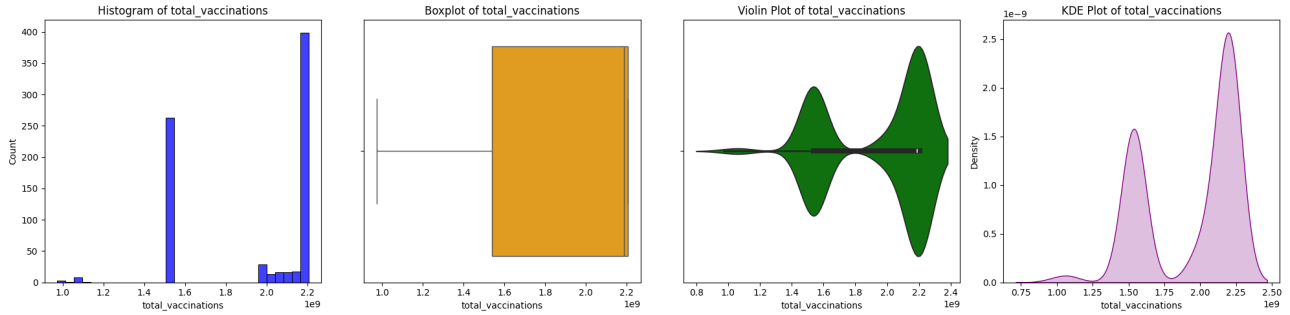


Figure 3.5: total_vaccinations

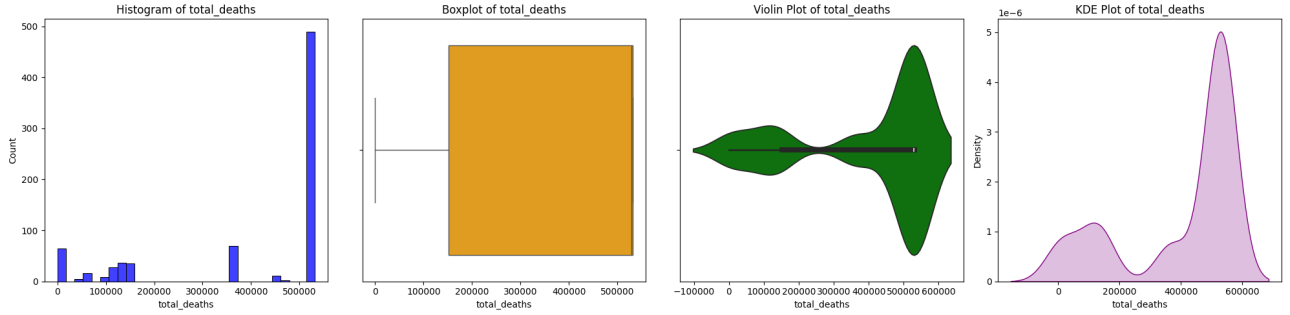


Figure 3.6: total_deaths

3.2 Multivariate analysis

Multivariate analysis involves examining relationships and interactions between two or more variables simultaneously. This analysis is crucial for understanding how different aspects of the COVID-19 pandemic, such as cases, deaths, and vaccination rates, are interconnected. To achieve this, we used four visualization techniques: heatmaps, line plots, scatter plots, and pair plots. Each plot provides unique insights into the relationships within the data.

3.2.1 Heatmap

A heatmap is a graphical representation of data where individual values are represented as colors. It is commonly used to display correlations between numerical variables in a dataset.

- For the COVID-19 dataset, the heatmap revealed the strength and direction of relationships between variables such as `total_cases`, `total_tests`, and `total_vaccinations`.
- Strong positive correlations, such as between `total_cases` and `total_deaths`, indicate that increases in one variable are closely associated with increases in the other.
- Weak or negative correlations can highlight areas where relationships are less evident or inverse.

The heatmap helps us identify key variable pairs for further exploration or modeling.

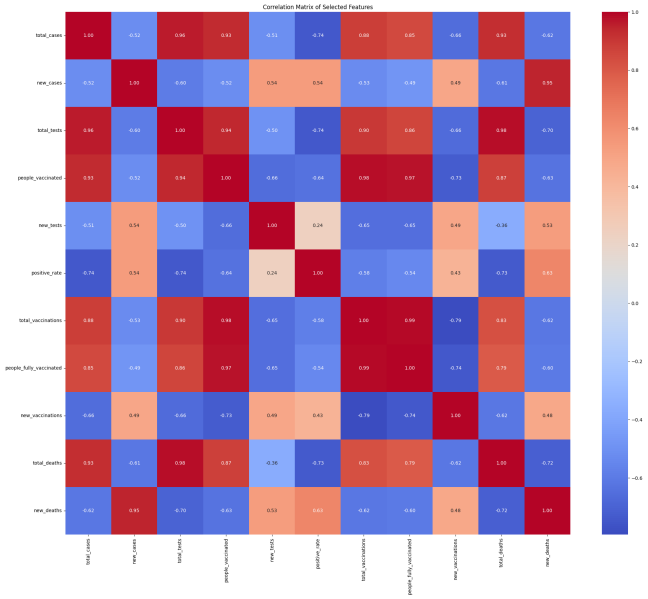


Figure 3.7: Caption

3.2.2 Line Plot

A line plot is used to visualize trends over time. It is particularly effective for time-series data, making it an ideal tool for analyzing the progression of COVID-19 metrics.

- For the COVID-19 dataset, line plots were used to examine trends in variables such as `daily_new_cases`, `daily_deaths`, and `vaccination_rates` over time.
- These plots help identify patterns like peaks during pandemic waves, the flattening of the curve due to interventions, or the steady rise in vaccination rates.
- Line plots also highlight anomalies or disruptions in data trends, such as sudden surges in testing or vaccination campaigns.

3.2.3 Scatter Plot

Scatter plots are used to explore relationships between two continuous variables. Each point in the plot represents an observation in the dataset, with its position determined by the values of the two variables being compared.

- In the COVID-19 dataset, scatter plots help visualize relationships such as the association between `total_tests` and `total_cases`.
- Patterns in the scatter plot, such as clustering or linear relationships, can indicate strong associations or reveal heterogeneity in the data.
- Outliers in scatter plots may correspond to unique events or regions, such as areas with disproportionately high testing rates or anomalous case counts.

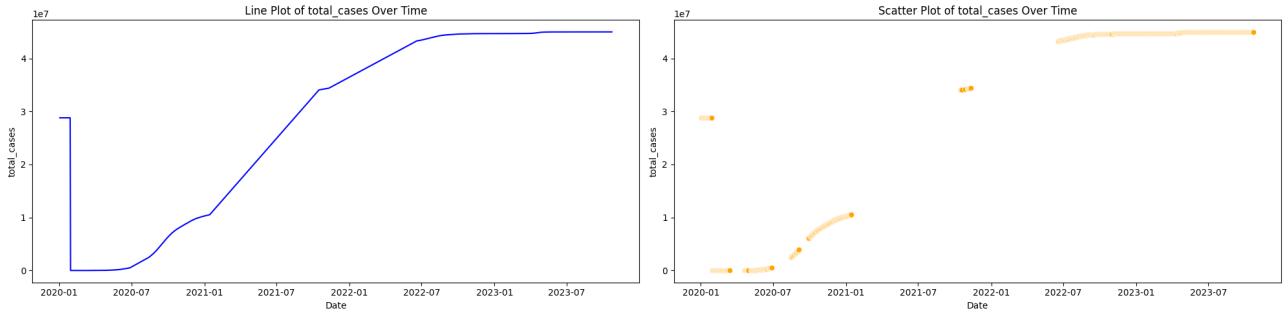


Figure 3.8: total_cases

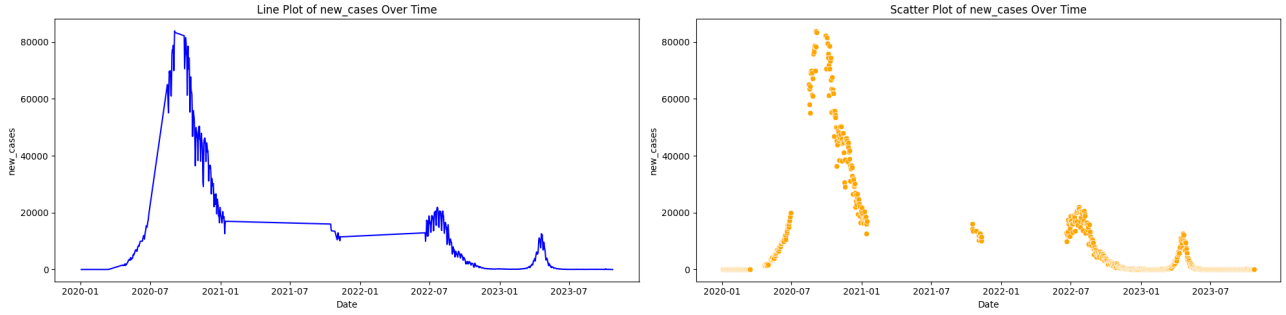


Figure 3.9: new_cases

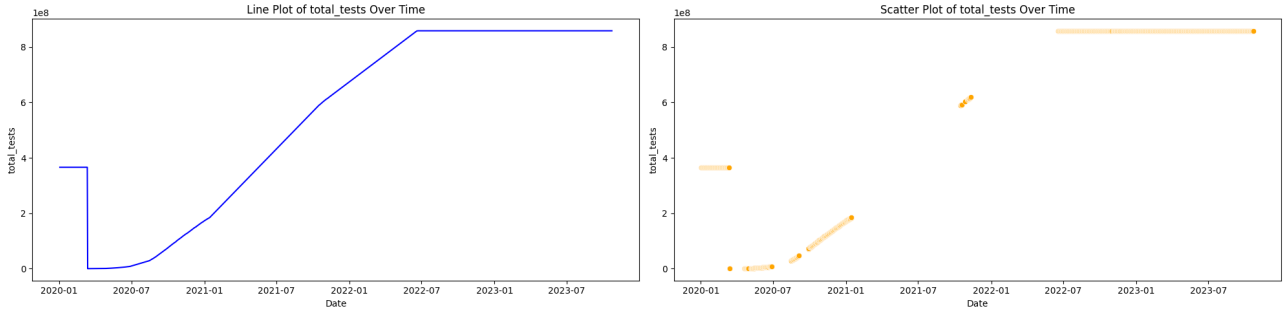


Figure 3.10: total_tests

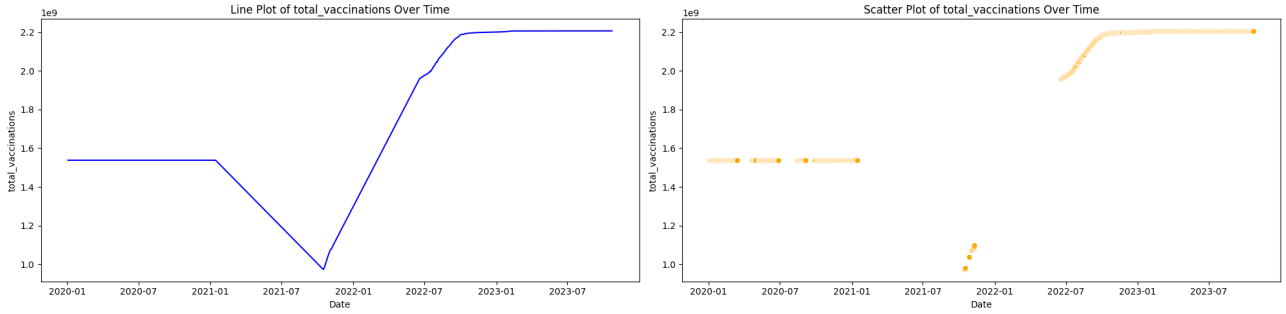


Figure 3.11: total_vaccinations

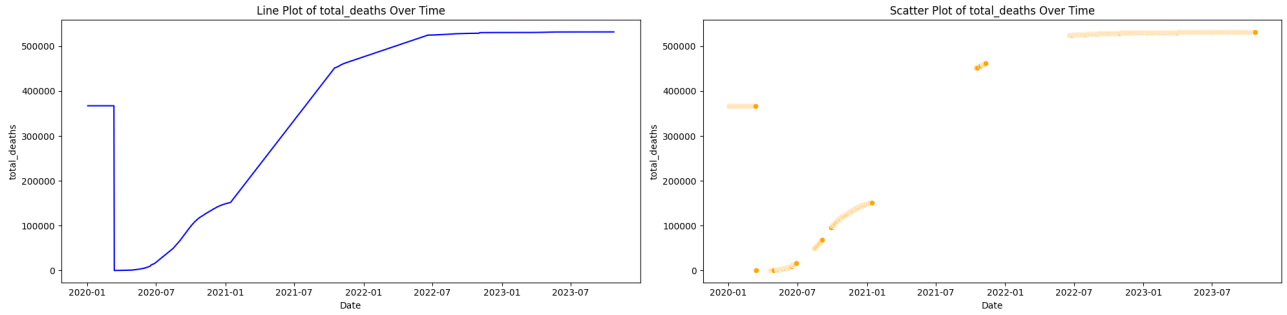


Figure 3.12: total_deaths

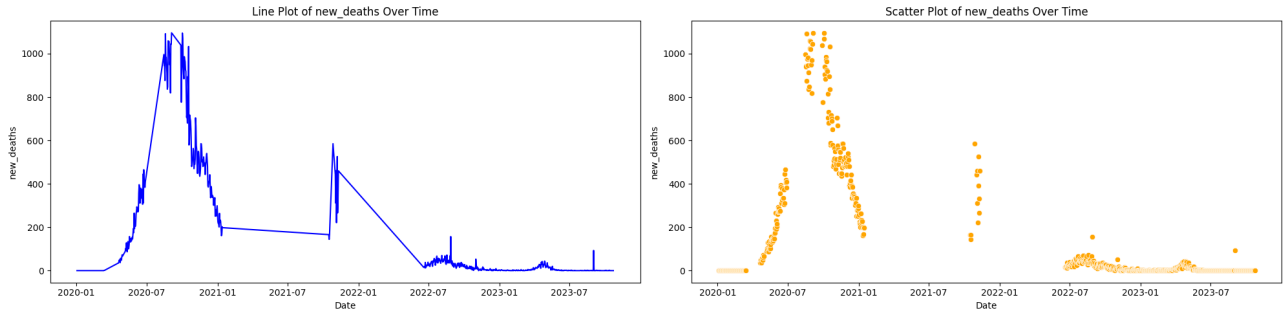


Figure 3.13: new_deaths

3.2.4 Pair Plot

A pair plot is a grid of scatter plots, combined with histograms or KDE plots for the diagonal elements. It allows for a comprehensive examination of relationships between multiple variables at once.

- For the COVID-19 dataset, the pair plot highlights how variables like `total_cases`, `total_deaths`, `total_tests`, and `vaccination_rates` interact.
- It provides a quick overview of correlations, distributions, and potential clusters within the data.
- Pair plots are particularly useful for spotting multicollinearity or variables with similar patterns.

3.2.5 Insights from Multivariate Analysis

Using these plots, we gained the following insights:

- The heatmap revealed a strong positive correlation between `total_cases` and `total_deaths`, consistent with the severity of the pandemic.
- Line plots demonstrated the progression of daily cases and vaccinations, helping identify key turning points such as the impact of lockdowns or vaccine rollouts.
- Scatter plots showed a clear relationship between `total_tests` and `total_cases`, emphasizing the role of widespread testing in identifying cases.
- The pair plot provided a holistic view of how key variables relate, with clusters suggesting differences in pandemic outcomes across regions or time periods.

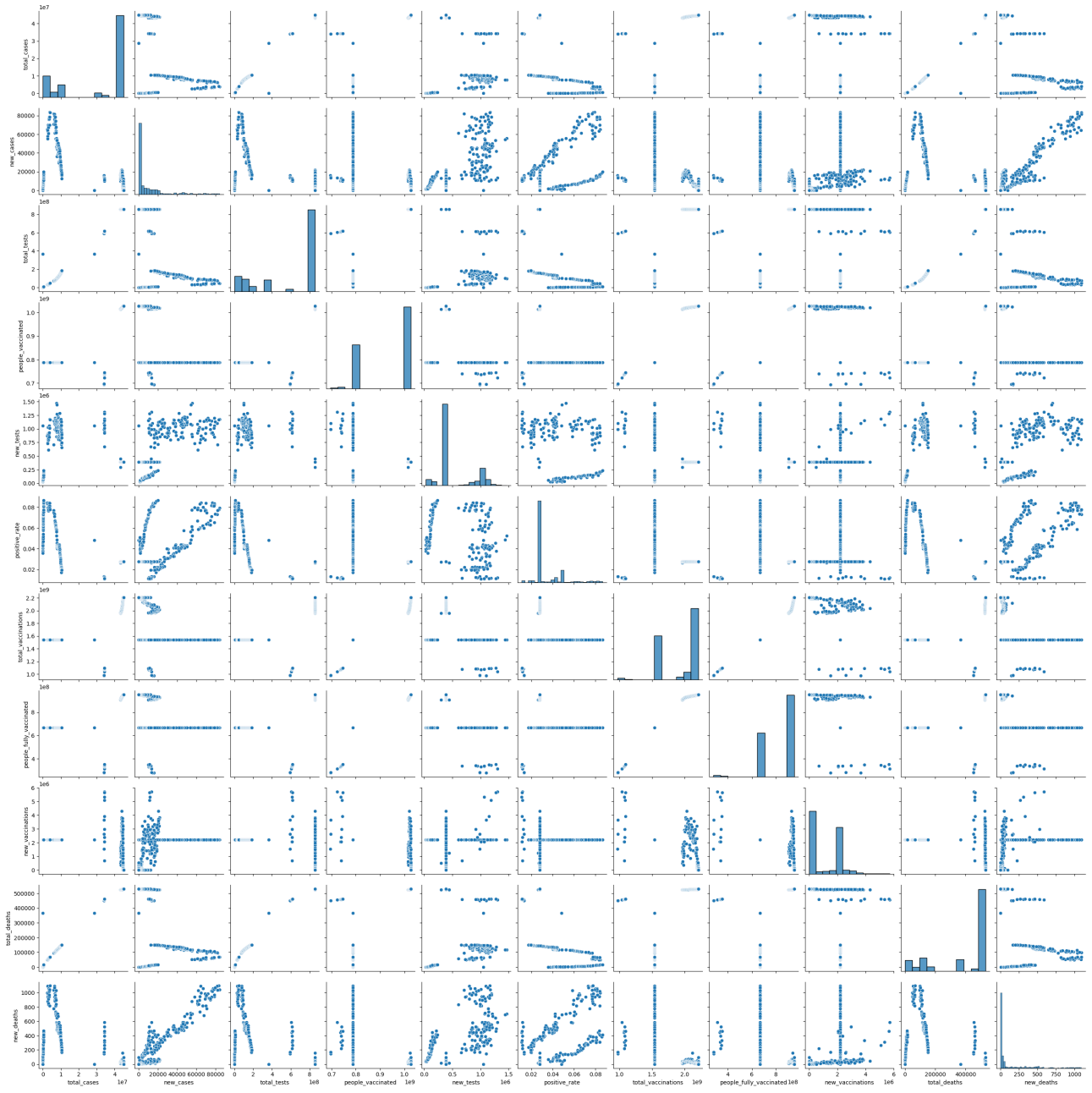


Figure 3.14: The above figure depicts the pairplot of some features.

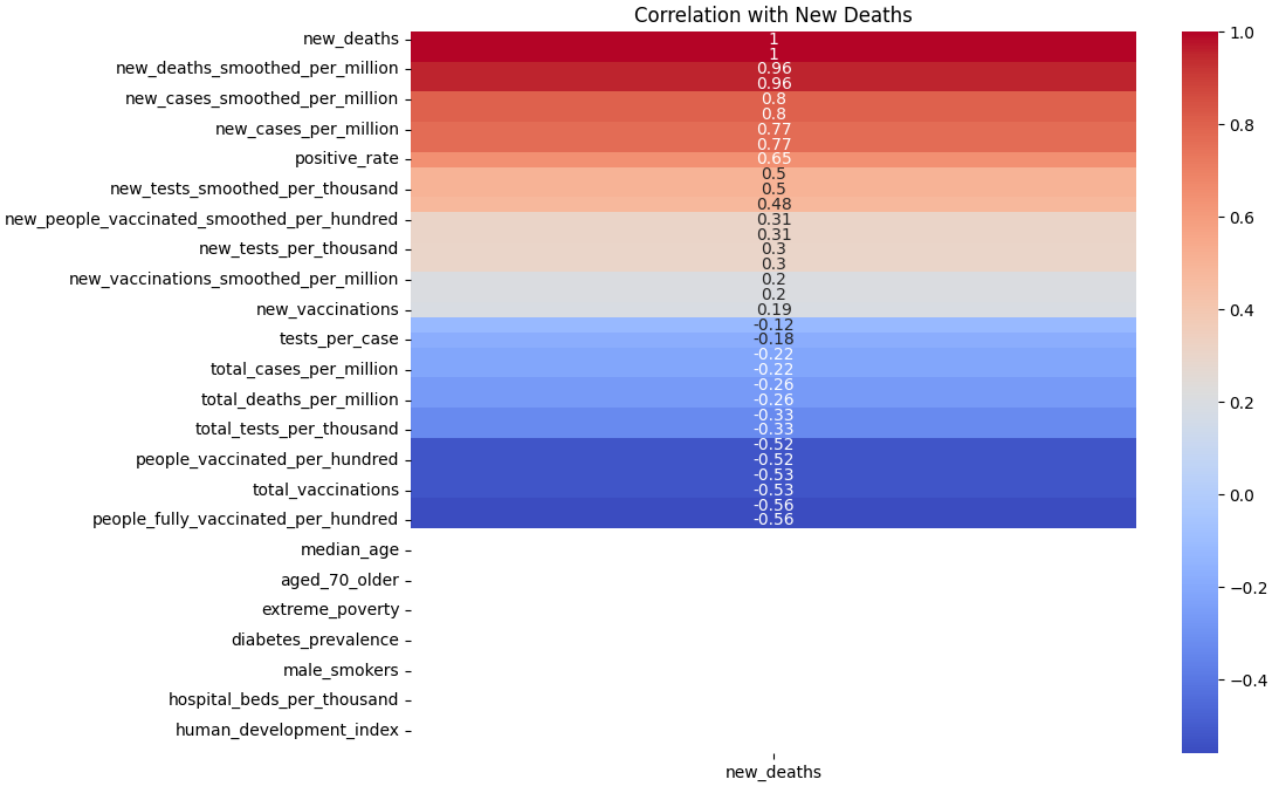
Multivariate analysis helps uncover the intricate relationships between variables in the COVID-19 dataset. By combining different visualization techniques, we identified correlations, trends, and patterns that offer valuable insights into the pandemic's dynamics and its impact.

Chapter 4. Feature Engineering

4.1 Feature selection

4.1.1 Correlation Analysis

- The code first calculates the correlation matrix for the dataset, focusing on how strongly each feature correlates with the target variable, new_deaths.
- Using Seaborn's heatmap, the correlation values for all features are visually presented in a sorted order to highlight the strength and direction (positive or negative) of the relationships.
- This visualization helps in identifying features that may have a linear relationship with the target variable, aiding in feature selection and understanding.



4.1.2 Feature Importance Using Random Forest

- A Random Forest Regressor model is trained to estimate the importance of each feature in predicting new_deaths.
- The target variable, new_deaths, is separated from the rest of the features (X contains all independent variables except new_deaths, and y contains new_deaths).
- The model is then fitted using RandomForestRegressor with a fixed random state for reproducibility.
- The feature importances generated by the model are stored in a Pandas Series and sorted in descending order. Features with higher importance scores are more influential in the prediction task.

4.1.3 Feature Selection Using K best method

- The K-Best Method is a statistical approach to feature selection where the top K features are selected based on a specific scoring function. It evaluates each feature's individual relationship with the target variable, assigns a score, and selects the best K features with the highest scores.
- This method is implemented in the SelectKBest class of the scikit-learn library.
- we use mutual_info_regression: Measures the dependency between a feature and the target, including non-linear relationships. Based on information theory.

Chapter 5. Model fitting

5.1 ARIMA a Time Series Model

- ARIMA stands for:
  - **AR (AutoRegressive):** Relates the variable to its own past values (lags).
  - **I (Integrated):** Differencing the data to make it stationary. MA (Moving Average): Models the dependency on past forecast errors.
  - In essence, ARIMA is a univariate model that internally uses lagged data and residuals for prediction, not independent explanatory variables like traditional regression.
- We implement an ARIMA (AutoRegressive Integrated Moving Average) model to analyze and forecast the time series data of new_deaths. It performs the following steps:
- Tests for stationarity using the Augmented Dickey-Fuller (ADF) test.
- Differentiates the series to make it stationary if necessary. Analyzes autocorrelation and partial autocorrelation to determine suitable ARIMA parameters (p, d, q).
- Fits the ARIMA model and evaluates its performance using various metrics.
- Forecasts future values and assesses the accuracy of predictions

5.1.1 Process

5.1.2 Testing for Stationarity (ADF Test)

- The ADF test checks if the time series is stationary.
- If the p-value $>$ 0.05, the data is non-stationary, and differencing is required to stabilize the mean.
- If p-value $\leq$ 0.05, the data is stationary, and no differencing is needed.
- ARIMA assumes the time series is stationary for accurate modeling. The ADF Statistic is less

```
ADF Statistic: -3.204677196079222
p-value: 0.019715536018882623
```

than the critical values (thresold, -2.9 for 5% significance). The p-value is much less than 0.05, so that we reject the null hypothesis of non-stationarity.

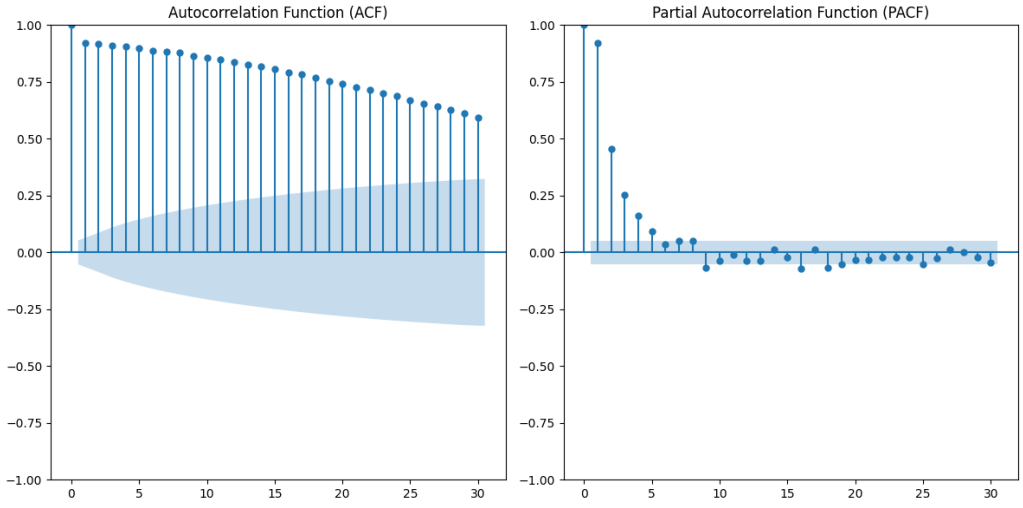
Differencing for Stationarity

- This transforms the non-stationary data into a stationary series by differencing.
- This computes the difference between consecutive values, effectively removing trends or seasonality.

```
ADF Statistic after differencing: -15.874021877326593
p-value after differencing: 8.913794439583317e-29
```

- The ADF test is re-run on the differenced data to confirm stationarity, Now its fit into our thresold.
- This means our target variable (new_deaths_diff) is ready for ARIMA modeling.

5.1.3 Autocorrelation and Partial Autocorrelation Analysis

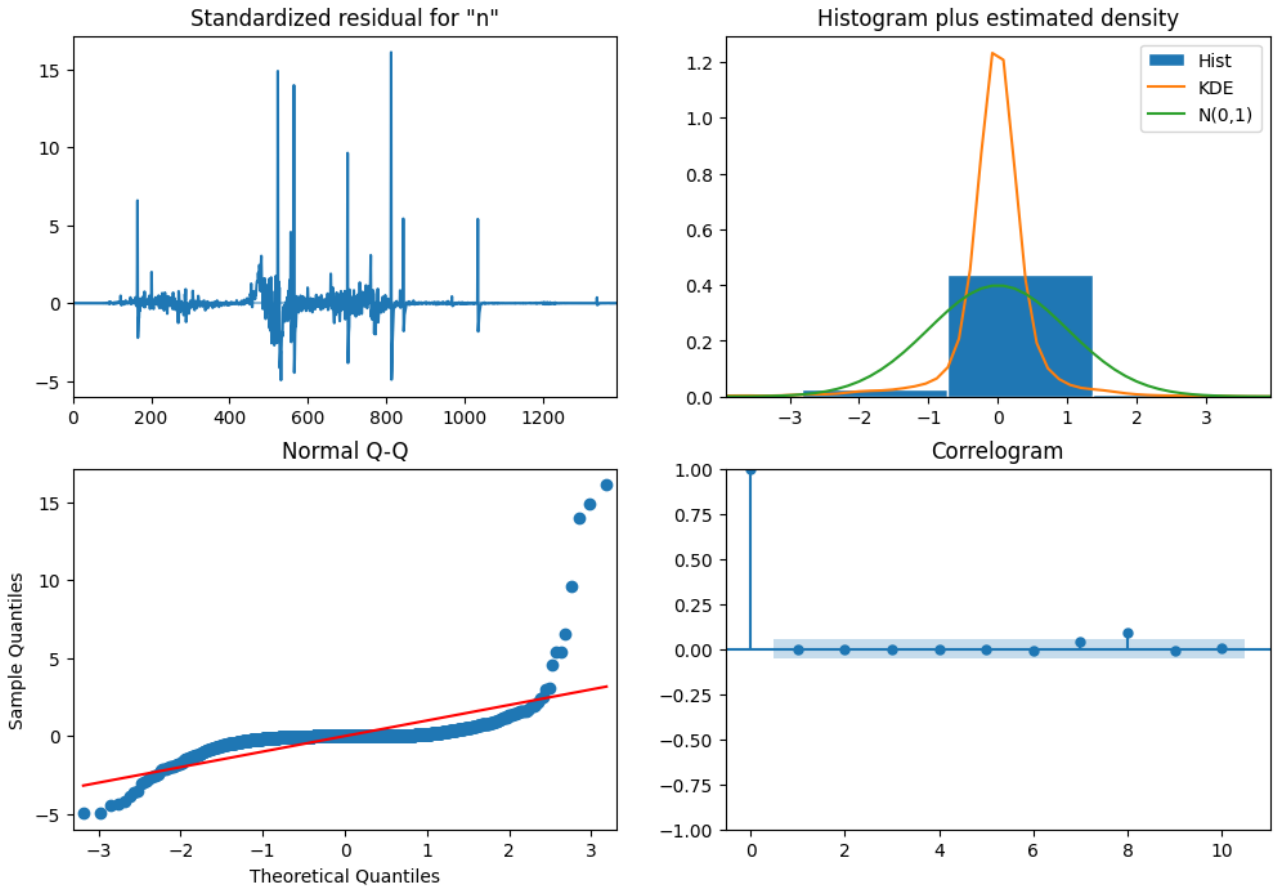


- p (AR order): Determined by the PACF plot.
- q (MA order): Determined by the ACF plot.
- This is for the number of significant lags in the plots to set p and q.

5.1.4 ARIMA Parameters:

-
- **p (AR Order):** Number of lagged terms to include.
- **d (Differencing Order):** Number of differencing steps applied to make the series stationary.
- **q (MA Order):** Number of lagged forecast errors to include.

5.1.5 Model Diagnostics



- **Standardized Residuals (Top-Left)**
  - What it shows: This plot displays the residuals (differences between actual and predicted values) standardized to have a mean of 0 and a standard deviation of 1.
  - What to look for: The residuals should resemble white noise (no clear pattern). They should be evenly distributed around zero without trends or clustering
- **Histogram + KDE (Top-Right)**
  - What it shows: A histogram of the residuals with a Kernel Density Estimate (KDE) overlay, and a QQ plot comparison with the normal distribution.
  - What to look for: The residuals should follow a normal distribution. The histogram should appear bell-shaped, and the QQ plot should form a straight diagonal line.
- **QQ Plot (Bottom-Left)**
  - What it shows: A Quantile-Quantile (QQ) plot comparing the quantiles of the residuals to the theoretical quantiles of a normal distribution.

  - **What to look for:** If the residuals are normally distributed, the points should align closely with the diagonal line. Deviations from the line suggest non-normality in the residuals.

- Correlogram (ACF of Residuals) (Bottom-Right)
  - What it shows: The autocorrelation of residuals plotted as a function of lag.
  - What to look for: The autocorrelations should be close to zero at all lags (no significant correlation). Residuals with significant autocorrelations suggest that the model has not captured all patterns in the data.

5.1.6 Forecasting and Fitted Values

```
Forecasted values: 1390    0.002481
1391    0.002038
1392    0.001903
1393    0.001822
1394    0.001823
1395    0.001847
1396    0.001850
1397    0.001849
1398    0.001848
1399    0.001847
```

- This is the next 10 days predicted values.
- Here data is scaled using MinMaxSacler so that values are in between 0 and 1.

5.1.7 Error Metrics for Model Evaluation

```
Mean Absolute Error (MAE): 76.8248588456453
```

```
Root Mean Squared Error (RMSE): 249.51225082695976
```

- The ARIMA model provides a systematic approach to time series analysis and forecasting. The differencing ensures stationarity, while ACF/PACF guides parameter selection. The evaluation metrics and diagnostics validate the model's effectiveness, although additional tuning or alternative models (e.g., LSTM) may further enhance accuracy.

5.2 LSTM

LSTM, short for Long Short-Term Memory, is a specialized type of Recurrent Neural Network (RNN) designed to model and predict sequences of data. Standard RNNs struggle with learning patterns in sequences when the relationships between elements span long distances (e.g., predicting the next word in a long sentence). LSTMs overcome this by using a unique internal structure that enables them to retain and manage information over long periods effectively.

5.2.1 Data Preparation

- **Normalization:**
  - Normalizing the target variable, new_deaths, between 0 and 1 ensures the data fits within a range suitable for the LSTM, which performs better with normalized inputs.
  - MinMaxScaler rescales the new_deaths column so that all values are mapped between 0 and 1.
- **Creating Lagged Sequences:**

```
def create_sequences(data, n_steps):
X, y = [], []
for i in range(len(data) - n_steps):
    X.append(data[i:i+n_steps])
    y.append(data[i+n_steps])
return np.array(X), np.array(y)

```

  - To prepare the data for time series forecasting by creating sequences of past observations (X) and corresponding targets (y).
  - This function slices the time series into sequences of length n_steps. For each sequence of n_steps values, the target is the next value (n_steps + 1).

5.2.2 Building the LSTM Model:

- **Architecture:**
  - Input: A sequence of n_steps past values. Hidden Layers:
    - Two LSTM layers with 50 units each and relu activation.
    - Dropout layers (20%) to prevent overfitting.

    Output: A single prediction .
- **Optimizer:** Adam optimizer, efficient for this task.
- **Loss:** Mean Squared Error (MSE) for regression.
- **epochs=50:** The model trains for 50 complete passes through the data.
- **batch_size=32**: Processes data in chunks of 32 samples at a time.
- **validation_split=0.2:** Reserves 20% of the training data for validation.

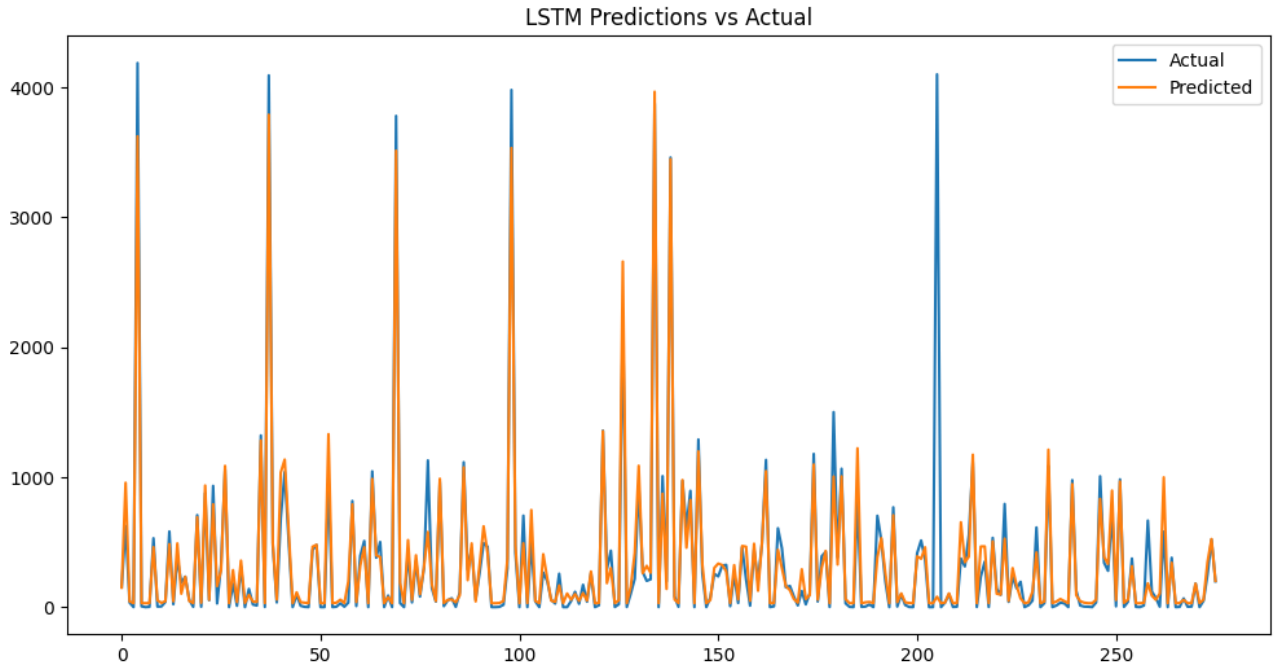


Figure 5.1: X represents days and y represents new deaths

- **Mean Squared Error (MSE):** 72277.70373126744
- **Mean Absolute Error (MAE):** 82.68699831893478
- In this figure how does our model fit with actual data.

	forecasted_new_deaths
2023-10-24	30.037979
2023-10-25	30.788078
2023-10-26	32.030914
2023-10-27	33.596592
2023-10-28	35.371147
2023-10-29	37.277966
2023-10-30	39.328125
2023-10-31	41.668774
2023-11-01	44.302238
2023-11-02	47.242203

Figure 5.2: These are the forecasted new deaths predicted by our model

5.3 XGBoost

XGBoost (Extreme Gradient Boosting) is a high-performance, scalable machine learning algorithm based on gradient boosting. It is particularly well-suited for both regression and classification tasks,

offering strong predictive performance. XGBoost builds an ensemble of decision trees in a sequential manner, where each tree attempts to correct the errors of the previous ones.

- XGBoost is often chosen for predictive modeling, especially when the data is complex or contains features with nonlinear relationships. Here's why it is particularly useful for forecasting tasks like predicting new_deaths in our dataset:
- XGBoost builds an ensemble of decision trees, which are inherently good at modeling nonlinear relationships.
- For predicting new_deaths, the relationship between input features (e.g., date-related variables like day, month, year) and the target can be highly nonlinear due to seasonal trends, policy changes, or unexpected events like outbreaks.
- **Smaller Dataset:** LSTMs require large datasets to train effectively. For smaller datasets, XGBoost performs better with limited overfitting.

5.3.1 Results

- **Mean Absolute Error (MAE):** 77.69904639727349
- **Mean Squared Error (MSE):** 28504.922938642056
- **Root Mean Squared Error (RMSE):** 168.83401001765625
- **R-squared (R^2):** 0.9243428338531494
  - **Mean Absolute Error (MAE):** 77.7
    * On average, the model's predictions deviate from the actual values by about 77.7 deaths.
    * A lower MAE indicates better model performance.
  - **Mean Squared Error (MSE):** 28,504.92
    * This measures the average squared difference between predicted and actual values.
    * It penalizes larger errors more heavily than MAE. While the raw value is not intuitive to interpret, its magnitude suggests some large prediction errors, though not excessively frequent.
  - **Root Mean Squared Error (RMSE):** 168.83
    * This is the square root of MSE, representing the standard deviation of prediction errors in the same unit as the target variable (deaths).
    * An RMSE of 168.83 deaths implies that most predictions fall within ±169 deaths from the actual values.
  - **R-squared (R^2):** 0.924
    * Indicates that 92.4% of the variance in new_deaths is explained by the model.
    * An R^2 value close to 1 signifies a strong fit between predictions and actual values, showing the model captures most of the patterns in the data

5.3.2 Forecasted Value

Here are the next 10 day's predicted new_deaths.

```
Forecasted Values for the next 10 days:
[449.11996, 393.33173, 275.8605, 275.8605, 275.8605, 275.8605, 275.8605, 275.8605, 275.8605, 275.8605]
```

5.4 ML algorithms

LSTM is designed to handle sequential and time-series data, making it the most appropriate choice for predicting new deaths, as COVID-19 data often exhibits the following characteristics:

1. Time-Dependent Patterns
    - COVID-19 death counts typically have patterns influenced by: Outbreak waves.
    - Seasonal effects or policy changes (e.g., lockdowns, vaccinations).
    - LSTMs are excellent at capturing such temporal dependencies and long-term trends compared to XGBoost or ARIMA, which can struggle with highly nonlinear temporal relationships
2. Ability to Learn Complex Relationships
    - LSTM uses gates (input, forget, and output gates) to manage and retain information over time.
    - This mechanism allows it to learn both short-term spikes (e.g., sudden outbreak) and long-term patterns (e.g., a declining trend due to vaccination).

Why LSTM Over XGBoost or ARIMA? Compared to XGBoost:

- XGBoost is great for tabular, structured data but lacks the sequential understanding needed for time-series trends. While XGBoost can provide good results, it might require heavy feature engineering to capture temporal dependencies, which LSTM does inherently.
- **Compared to ARIMA:** ARIMA assumes linear relationships and stationarity, which COVID-19 data often violates. It performs poorly when the data has complex, nonlinear, or non-stationary patterns.

Chapter 6. Conclusion & future scope

The predictive models developed for the prediction of new COVID-19 deaths show considerable promise for practical application. Precise mortality pattern forecasting using these models can facilitate resource distribution, inform policy development, and give early alerts for public health interventions. Further research may expand these models to include other diseases, integrate supplementary external variables such as vaccination rates and variants, and be incorporated into systems for real-time forecasting. Hybrid models and uncertainty quantification may also improve the accuracy or reliability of predictions. Thus, this research may be found useful in effective pandemic management and public health decision-making.

Our work in developing predictive models for new_deaths due to COVID-19 has significant implications and potential for future development. Below are some areas where this work can be extended or applied:

6.1 Findings/observations

1. Model Performance: The LSTM model demonstrated strong predictive accuracy for COVID-19 'new_deaths', the figure fitted the best, indicating it explained most of the variance in the data.
2. Error Metrics: The model's MAE (77.7 deaths) suggest moderate prediction errors, but the overall model performance is satisfactory for practical use.
3. Temporal Patterns: The LSTM's ability to capture time-dependent patterns and non-linear relationships was crucial for forecasting death trends, especially with COVID-19's fluctuating nature.
4. Model Strengths: LSTM outperformed XGBoost and ARIMA, particularly in handling the sequential and non-linear data dynamics typical of pandemic data.
5. Scalability: The model can be adapted for future pandemics or other diseases by incorporating additional features like vaccination rates and policy changes.
6. Potential for Real-Time Forecasting: The model can be integrated into live systems to provide ongoing predictions and support proactive health interventions.

6.2 Challenges

1. **Data Collection:**
    - One of the primary challenges encountered during this project was gathering comprehensive and accurate COVID-19 data.

- The data was often fragmented, inconsistent, or incomplete, especially regarding regional differences in reporting standards and data availability.
- This required extensive data cleaning and preprocessing to ensure that the dataset was both accurate and usable for predictive modeling.

2. **Feature Selection:**
   - Selecting the right features for model input was another hurdle. Early versions of the model struggled with underfitting due to a lack of relevant features.
   - The key challenge was identifying which variables (e.g., date-related features, mobility data, vaccination rates) had the most impact on predicting deaths and ensuring that the feature set was both comprehensive and not overly complex.
   - Additional external data, such as local government policies or new virus variants, could have further improved the model but was difficult to source.
3. **Model Building:**
   - Building effective models for forecasting new_deaths was complex due to the nature of the data, which included both non-linear relationships and time-series dependencies.
   - While traditional models like ARIMA struggled with the data's complexity, tuning more advanced models like LSTM and XGBoost to perform optimally required careful parameter selection and model evaluation.
   - Ensuring that the models did not overfit the data while still capturing key patterns was a significant challenge.

6.3 Future plan

6.3.1 Public Health Decision Support

4. **Resource Allocation:** Predictions can help allocate medical resources (e.g., hospital beds, oxygen supplies, ventilators) in areas likely to see increased deaths.
   - **Policy Making:** Governments can use forecasts to plan interventions like lockdowns, vaccination drives, or travel restrictions.
   - **Early Warnings:** Serve as an early warning system for upcoming waves, allowing for preemptive measures.

6.3.2 Real-Time Forecasting Systems Automation:

- Integrate the model into a real-time data pipeline to provide daily or weekly forecasts.
- Dashboard Integration: Build a live dashboard for healthcare providers, displaying current trends, forecasts, and uncertainty intervals.

6.3.3 Hybrid Model Development Combine strengths of multiple models:

- LSTM for time-series understanding + XGBoost for feature engineering and accuracy.
- Ensemble approaches could improve prediction performance.

Group Contribution

Akshat Kadia (202203029): Find problem statement, collected and organized the dataset, and led the implementation of machine learning models, Evaluating their performance.

Pritesh Vadher (202203043): Handled data cleaning, missing data analysis, outliers detection, and imputation. Contributed to data visualization and feature engineering toward raising model insights.

Vivek Chaudhary (202201294): Focused on data cleaning, exploratory data analysis (observations), and encoding categorical data into machine-readable formats.

Together, the team efficiently divided responsibilities, ensuring smooth progress from data preparation to model evaluation, resulting in a successful project.

Short Bio

1. **Akshat Kadia** is an enthusiastic learner with a keen interest in Artificial Intelligence, Machine Learning, and Web Development. Proficient in programming languages like C, C++, Java, and SQL, Akshat has hands-on experience with tools and technologies such as Power BI, VS Code, Overleaf, MATLAB, Jupyter Notebook, pgAdmin4, and GitHub.
His technical expertise is strengthened by coursework in Data Structures and Algorithms (DSA), Object-Oriented Programming (OOPs), Database Management Systems (DBMS), Parallel and Distributed Algorithms, and Operating Systems. Akshat is passionate about leveraging his skills to solve complex problems and contribute to cutting-edge advancements in technology.

2. **Pritesh Vadher** is a third-year B.Tech student in Mathematics and Computing, with a strong interest in data analysis and its practical applications. This course on Exploratory Data Analysis (EDA) has been particularly fascinating, as it has introduced me to the art of uncovering insights from raw data using Python. I have enjoyed learning essential concepts such as data preprocessing, feature selection, and visualizations, which have enhanced my ability to interpret and analyze complex datasets effectively.

My technical skills include programming languages like Python, Java, C++, and SQL, along with familiarity in tools such as Linux, MATLAB, and Scala. This course has further fueled my passion for solving logical and mathematical problems by showing how statistical techniques and programming can be combined to derive meaningful conclusions from data. It has also inspired me to delve deeper into advanced topics and explore innovative ways to approach data-driven challenges.

Outside academics, I enjoy playing cricket and various outdoor sports, which have taught me teamwork and strategic thinking. My inclination toward solving challenging problems and learning new concepts continues to grow, and this course has played a key role in strengthening my analytical mindset. I look forward to applying these skills to excel academically and make meaningful contributions in the field of data analysis.

3. **Vivek Chaudhari** is the Deputy Convenor of the Student Body Government, where I actively contribute to shaping a dynamic and collaborative environment for students.I am a dynamic software engineer with a keen interest in machine learning, exploratory data analysis (EDA), data visualization, and software development. I earned my Bachelor's degree in Computer Science from Dhirubhai Ambani University, where I developed a solid foundation in web development, spanning both front-end and back-end technologies.

I am proficient in programming languages like Python, JavaScript, and Java, and I excel in solving complex challenges related to software development, machine learning algorithms, and data visualization. In addition to my technical skills, I am passionate about organizing events, participating in music competitions, and performing as part of a music band. My strong communication skills and collaborative mindset enable me to thrive as a team player, consistently delivering high-quality solutions under tight deadlines.

Beyond my professional interests, I enjoy hiking, volunteering at local coding workshops, and exploring new musical horizons. I am deeply com-

mitted to inspiring the next generation of developers and fostering innovation within the tech community.

References

[1] Our World in Data. *COVID-19 Data*. *URL:* https://ourworldindata.org/coronavirus

[2] Statistical Software, Inc. *ARIMA Model for Time Series Forecasting*. *URL:* https://www.ibm.com/topics/arima-model

[3] Chris Albon. *XGBoost for Classification*. *URL:* https://xgboost.readthedocs.io/en/stable/tutorials/model.html

[4] Colah's Blog. *Understanding LSTM Networks*. *URL:* https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/