NATIONAL RESEARCH
UNIVERSITY

HIGHER SCHOOL OF ECONOMICS

FACULTY OF COMPUTER SCIENCE
School of Data Analysis and Artificial Intelligence

Title project presented in accordance with the coursework requirements

“Methods for the Analysis of Covid-19 Medical Data”

PRIYANKA SINGH

Direction of Training : 01.04.02 Applied Mathematics & Informatics

Professor Guide: BORIS MIRKIN

Full Professor, Higher School of Economics (Moscow)

Moscow 2023

Abstract

The COVID-19 pandemic has had a significant impact on India, necessitating the analysis of medical data to understand the spread of the virus, identify high-risk populations, and devise effective strategies for controlling the outbreak. This abstract provides an overview of the methods used for COVID-19 medical data analysis in India. The analysis of COVID-19 medical data in India encompasses various techniques, including statistical analysis, machine learning, data mining, and epidemiological modeling. Statistical analysis enables researchers to examine the characteristics of COVID-19 cases, such as age, gender and outcomes providing insights into the demographic factors associated with severe illness and mortality. Moreover, analytical techniques are employed to analyze testing and vaccination data to assess the effectiveness of containment measures and vaccination campaigns.

Machine learning algorithms play a crucial role in COVID-19 data analysis by enabling disease progression, identification of high-risk individuals, and forecasting future case counts. Supervised learning models can be trained on Indian COVID-19 datasets to classify patients into different risk categories, helping healthcare providers allocate resources efficiently. Unsupervised learning algorithms identify patterns and clusters within the data, helping to discover potential hotspots and early warning signs of outbreaks. Data mining techniques are employed to extract valuable insights from large volumes of COVID-19 data in India. This includes electronic health records, contact tracing data, and social media data. This is to understand transmission dynamics, detect super-spreading events, and evaluate public sentiment towards preventive measures. Data mining also facilitates the identification of risk factors, such as demographic characteristics or preexisting conditions, that contribute to disease severity. Epidemiological modeling plays a pivotal role in understanding the COVID-19 pandemic trajectory in India. By utilizing

mathematical models, such as SEIR (Susceptible-Exposed-Infected-Recovered) or agent-based models, researchers can simulate various scenarios and assess the impact of interventions like lockdowns, travel restrictions, and vaccination strategies. These models aid policymakers in making informed decisions and predicting healthcare system capacity requirements.

In this coursework, I mainly focused on the analysis of COVID-19 India dataset based on regression model and neural networks model. Although a range of methods, such as statistical analysis, machine learning, data mining, and epidemiological modeling are available on the market, my task was to learn and apply regression on covid-19 India dataset. Regression approaches provide valuable insights into COVID-19 case characteristics, transmission dynamics, and public health interventions effectiveness. Regression methods are essential for guiding evidence-based decision-making and formulating targeted strategies to mitigate the pandemic impact in India.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Brief Introduction about the coursework

The COVID-19 pandemic has profoundly affected healthcare systems globally, including India. With a vast population and diverse healthcare infrastructure, India has faced significant challenges in managing the spread of the virus. This has included minimizing its impact on public health. In this coursework analysis, we focus on the application of regression models and neural networks to analyze COVID-19 medical data specific to India. We aim to uncover insights that inform effective pandemic control strategies.

This research investigates the application of regression models and neural networks to COVID-19 medical data. By utilizing these methods, we aim to uncover insights into the factors influencing the virus spread and its impact on different regions. Additionally, we will explore the predictive capabilities of these models, assessing their accuracy in forecasting COVID-19 case counts and identifying trends over time. Through this analysis, we intend to contribute to the understanding of COVID-19 in India. We also aim to provide valuable information for policy-makers, healthcare professionals, and researchers involved in the pandemic management.

To accomplish our objectives, we will employ a dataset specific to India. This dataset will encompass various variables related to COVID-19 cases, demographics, healthcare infrastructure, and socioeconomic factors. This dataset will serve as the foundation for developing and evaluating regression models and neural networks. By examining the relationships between these variables and their impact on COVID-19 outcomes, we can gain insights into the specific factors driving the spread of the virus in India. We can also identify areas where targeted interventions can be implemented to effectively control its transmission.

1.2 Formulating the Objective

Regression analysis is a statistical method for analyzing relationships between variables and making predictions. In the context of COVID-19, regression models offer a valuable tool to investigate the factors that influence the spread and severity of the virus in India. By examining various factors such as demographic information, socioeconomic indicators, and geographical variables, regression models can provide meaningful insights into the determinants of COVID-19 cases, hospitalizations, and mortality rates. Through regression analysis, researchers can identify demographic characteristics that contribute to COVID-19 vulnerability. Factors such as age, gender, and pre-existing health conditions can be explored to understand the extent to which these variables affect the likelihood of contracting the virus and experiencing severe symptoms. By identifying vulnerable populations, policymakers and healthcare professionals can focus resources and implement targeted interventions to protect those at high risk.

By employing regression models in the analysis of COVID-19 medical data in India, researchers can gain valuable insights into the determinants of the virus's spread and severity. This knowledge is essential for formulating evidence-based strategies to mitigate the pandemic impact. Identifying vulnerable populations, allocating resources effectively, and implementing targeted interventions can reduce transmission rates, minimize hospitalizations, and ultimately save lives.

Neural networks have gained considerable prominence in recent years due to their remarkable ability to model complex relationships and provide accurate predictions. Unlike traditional regression models, neural networks excel at capturing non-linear patterns and dependencies on multiple input variables. In the context of analyzing COVID-19 data, neural networks offer a powerful approach to exploring intricate relationships between factors such as population density, mobility patterns, healthcare infrastructure, and the spread of the virus in India. By training neural network models using COVID-19 datasets specific to India, we can predict the number of cases, evaluate the effectiveness of interventions, and identify potential hotspots within the country. Neural networks are an invaluable tool for comprehending pandemic dynamics and aiding decision-making processes. Neural networks are particularly adept at capturing complex interactions and patterns that might not be apparent using traditional statistical approaches. COVID-19 is a multifaceted disease influenced by numerous interconnected factors. By utilizing neural networks, we can account for intricate relationships between variables and uncover hidden patterns. This may significantly impact the virus' spread and severity. For example, neural networks can reveal how population density, mobility patterns, and healthcare infrastructure interact to influence COVID-19 transmission rates in different regions of India.

Training neural network models on COVID-19 India datasets enables us to predict the number of cases, which is crucial for assessing the pandemic trajectory. Accurate predictions can aid in resource allocation, allowing healthcare systems to prepare for future demands and allocate resources strategically. Additionally, neural networks can evaluate the effectiveness of interventions by analyzing how various factors contribute to the reduction or exacerbation of COVID-19 cases. This information can guide policymakers and public health officials in implementing targeted interventions and optimizing their effectiveness. Furthermore, neural networks can identify potential hotspots within India by analyzing the complex relationships between various factors and the localized spread of the virus. By identifying areas at high risk, authorities can allocate resources, implement targeted testing and vaccination campaigns, and enforce appropriate containment measures. This will mitigate the spread of the disease.

1.3 Socio- and Geo- Aspects

Socioeconomic factors also play a significant role in shaping COVID-19 impact. Regression models examine variables such as income levels, occupation types, and access to healthcare services. This is to understand how socioeconomic disparities contribute to COVID-19 outcomes variations. This information can guide policymakers in implementing equitable interventions and addressing social determinants of health to reduce disparities in COVID-19 outcomes. Geographical variables, including population density, urbanization, and regional characteristics, are critical to understanding the virus spread within India. Regression models can assess the impact of these variables on COVID-19 transmission rates and identify high-risk areas. This knowledge can help inform decisions about resource allocation, testing strategies, and targeted interventions to limit virus spread in specific regions.

1.4 Following Sections

The remainder of this report is organized as follows: Chapter 2 provides a review on COVID-19 India dataset description, data analysis, data preprocessing, and visualization of results. Chapter 3 describes the methodology employed in this study, including implementation of regression models, prediction model, and model performance evaluation and interpretation of the findings.. In the end section of this report, appendix is included for supplementary material. By conducting this analysis, we aim to contribute to the collective knowledge of COVID-19 data analysis in India and help devise effective strategies to combat the pandemic.

Chapter 2

Exploratory Data Analysis

2.1 Dataset Description

The COVID-19 India dataset provides detailed information on the COVID-19 pandemic at the state level within India. The dataset contains several key features and attributes that allow for a comprehensive analysis of the pandemic impact on different states. In this section, we will explore the dataset and discuss its various components.

- *State:* This attribute represents the name of each Indian state or territory included in the dataset.
- *Confirmed:* The ”confirmed” column indicates the cumulative count of confirmed COVID-19 cases in each state. It represents the total number of positive tests for the virus since the outbreak began.
- *Active:* The ”active” column signifies the current number of ongoing COVID-19 cases in each state. Active cases refer to individuals who are currently infected and undergoing treatment or isolation.
- *Passive:* The ”passive” column represents the number of individuals who have recovered from COVID-19 or completed their isolation period. These cases are classified as ”passive” since the individuals are no longer considered actively infected.

- *Deaths:* The ”deaths” column indicates the total number of COVID-19-related deaths reported in each state. It reflects the cumulative count of virus victims.
- *Dose1, Dose2, Dose3:* These columns provide information on the number of COVID-19 vaccine doses administered in each state. ”Dose1” represents the first dose administered, ”Dose2” represents the second dose (for two-dose vaccines), and ”Dose3” represents additional booster doses or third doses (if applicable).
- *Precaution-dose:* The ”precaution-dose” column signifies the number of precautionary or preventive doses administered. These doses may include booster shots or additional doses recommended for specific populations.
- *Total-doses:* The ”total-doses” column represents the cumulative count of all COVID-19 vaccine doses administered in each state. It includes all doses administered, including first, second, and precautionary doses.
- *Population:* The ”population” column indicates the estimated population of each state or territory. It provides context for understanding COVID-19 metrics in relation to population size.

The dataset captures data for multiple Indian states and territories, including Andaman and Nicobar, Andhra Pradesh, Arunachal Pradesh, Assam, Bihar, Chandigarh, Chhattisgarh, Dadra Nagar Haveli, Delhi, Goa, Gujarat, Haryana, Himachal Pradesh, Jammu and Kashmir, Jharkhand, Karnataka, Kerala, Ladakh, Lakshadweep, Madhya Pradesh, Maharashtra, Manipur, Meghalaya, Mizoram, Nagaland, Odisha, Puducherry, Punjab, Rajasthan, Sikkim, Tamil Nadu, Telangana, Tripura, Uttar Pradesh, Uttarakhand, and West Bengal.

The dataset provides valuable insights into the COVID-19 situation within each state, including the number of confirmed cases, active and passive cases, deaths, vaccination progress, and population data. Researchers, analysts, and policymakers can utilize this dataset to study and understand the patterns, trends, and impact of the pandemic at the state level.

It's imperative to note that the dataset has limitations. These limitations include potential reporting delays, variations in testing and reporting protocols across states, and possible data inconsistencies due to updates or revisions. Care should be taken to account for these limitations while conducting analyses or drawing conclusions from the dataset. The COVID-19 India dataset offers a comprehensive collection of information on the COVID-19 pandemic in different states within India.

2.2 Data Analysis

The figure.2.2 describes the top 10 states with highest population which are Uttar pradesh , maharashtra, bihar, west bengal, madhya pradesh,rajasthan, tamil nadu, karnataka, gujarat,andhra pradesh and top 10 states with the confirmed cases in that state which includes state maharashtra,kerala,karnataka,tamil nadu,andhra pradesh, uttar pradesh, west bengal, delhi, odisha,rajasthan .We found that 6 states out of 10 which has maximum population has maximum confirmed cases.

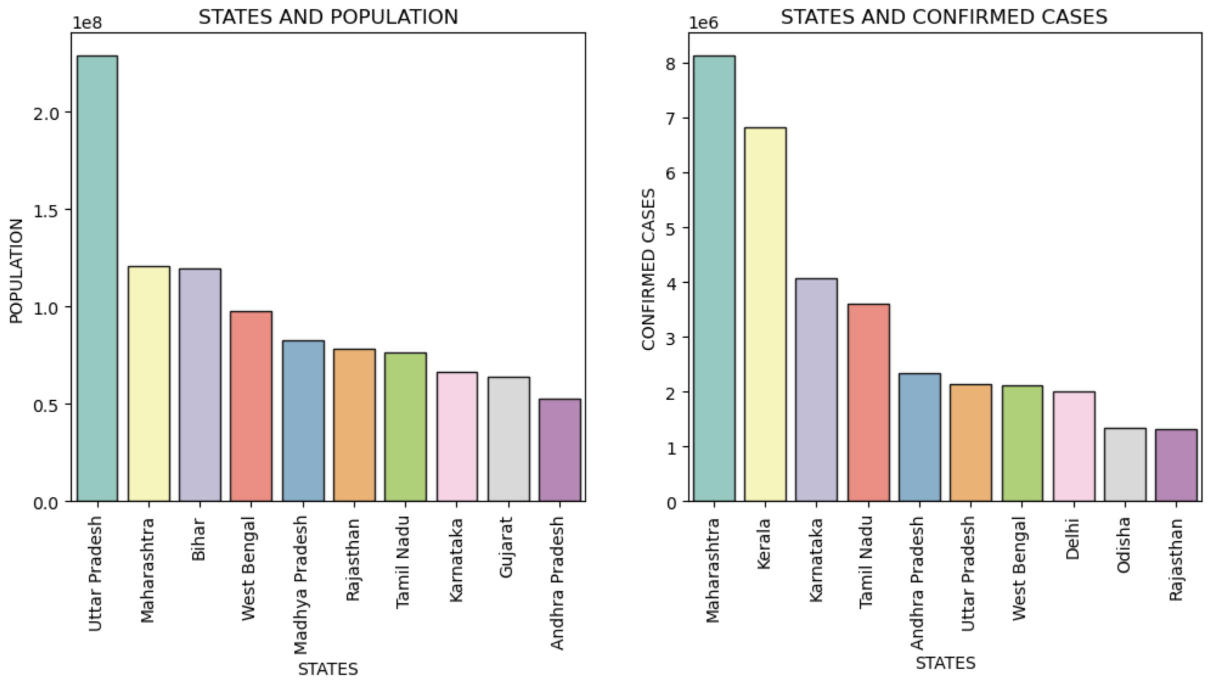


Figure 2.1: Top 10 states with population and confirmed cases

Table 2.1 describes states with high positivity rates for COVID-19. The data includes the state names and their corresponding positivity rates expressed as a percentage. The states listed in the table, along with their respective positivity rates, are as follows:

- Mizoram: The state of Mizoram has a positivity rate of 19.82%, indicating a significant proportion of COVID-19 tests conducted in the state return positive results.
- Kerala: Kerala has a positivity rate of 19.33%, suggesting a high rate of positive COVID-19 cases relative to the number of tests conducted in the state.
- Goa: With a positivity rate of 16.79%, Goa experiences a notable proportion of positive cases among the tested population.
- Lakshadweep: Lakshadweep has a positivity rate of 15.81%, indicating a considerable number of positive COVID-19 cases relative to the tests performed in the region.
- Puducherry: Puducherry exhibits a positivity rate of 12.56%, implying a significant proportion of positive cases compared to the tests conducted in the Union Territory.

This table serves as a reference for identifying states with higher positivity rates. This can inform public health measures, resource allocation, and targeted interventions to mitigate COVID-19 spread.
The table presented in 2.2 provides an overview of states in India with low COVID-19 positivity rates. It includes the names of the states and their corresponding positivity rates expressed as a percentage. The states listed in the table, along with their respective positivity rates, are Bihar (0.71%), Uttar Pradesh (0.92%), Jharkhand (1.18%), Madhya Pradesh (1.28%), and Nagaland (1.64%).

These states demonstrate a relatively low incidence of positive COVID-19 cases compared to the number of tests conducted. A low positivity rate indicates that a small proportion of individuals tested in these states are confirmed positive for the virus. This can be an encouraging sign, suggesting effective containment measures, robust testing strategies, and possibly fewer cases of spread of the virus within these regions.

Table 2.1: States with high positivity rates

State	**Positivity Rate (%)**
Mizoram	19.82
Kerala	19.33
Goa	16.79
Lakshadweep	15.81
Puducherry	12.56

Table 2.2: States with low positivity rates

State	**Positivity Rate (%)**
Bihar	0.71
Uttar Pradesh	0.92
Jharkhand	1.18
Madhya Pradesh	1.28
Nagaland	1.64

By considering the variations in positivity rates among states, public health measures can be tailored to address specific challenges and allocate resources effectively. Understanding the factors contributing to low positivity rates can help inform strategies for maintaining low transmission rates and preventing future outbreaks. Overall, high- low- positivity contributes to the broader understanding of the COVID-19 landscape in India and supports evidence-based decision-making to mitigate the impact of the pandemic.
Seems like the number of doses administered is more co-related to the population. total number of doses is not in likes of positivity rate but more correlated to population

Figure 2.2 describes states with highest death rates and lowest death rates.The state with highest death rates include Punjab, Nagaland, Maharashtra,Uttrakhand,Meghalaya and states with lowest death rate includes Dadra Nagar Haveli, Mizoram,Arunachal Pradesh, Lakshdweep, Telangana. from figure 2.1 and figure 2.2 we found that states with highest death rate and lowest death rate are not the states with top 10 highest population or top 10 confirmed cases states except Maha-

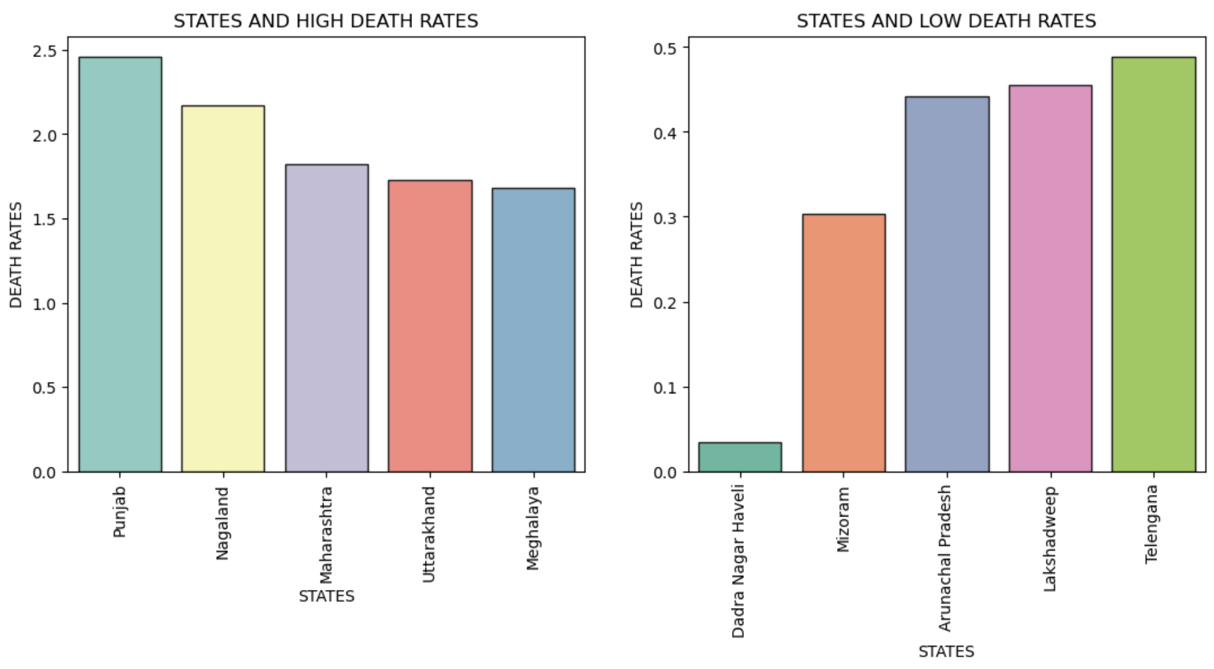


Figure 2.2: Top five states with high and low death rates

rashtra.

Figure 2.3 describes the distribution of dose1, dose2, dose3 in all states.The distribution of dose1 is 68.83%, dose2 is 64.61% and dose3 is 11.55% in all states.

Figure 2.4 describes the correlation heatmap of doses and vaccination with population of that state.

Figure 2.5 describe the states with highest and lowest vaccination rates.The highest vaccination states are Andaman and Nicobar, Telangana,Sikkim, Ladakh, Andhra pradesh. And the states with lowest vaccination rates are Meghalaya, Nagaland, Kerala, Punjab and Jharkhand.

From figure 2.2 and 2.5 we can compare that three states which have lowest vaccination rates are Meghalaya, Nagaland, Punjab have the highest death rate. Thus we can say that there is negative relation between vaccine and death.

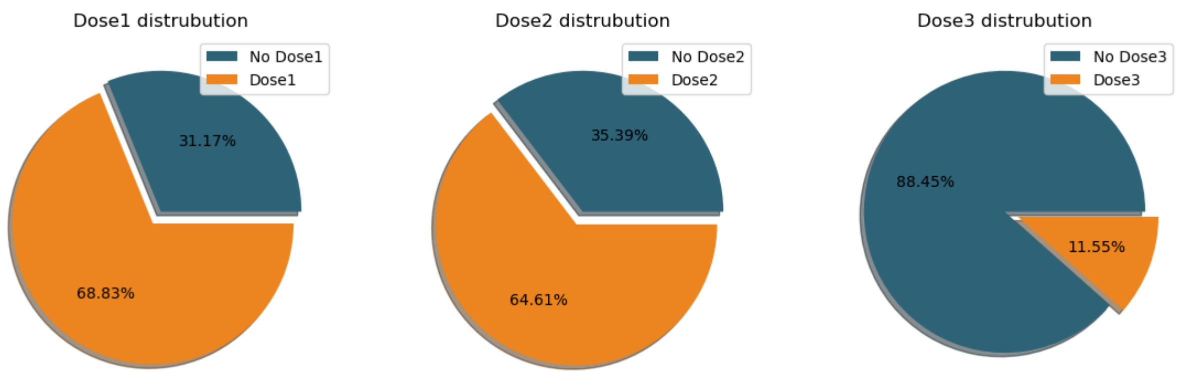
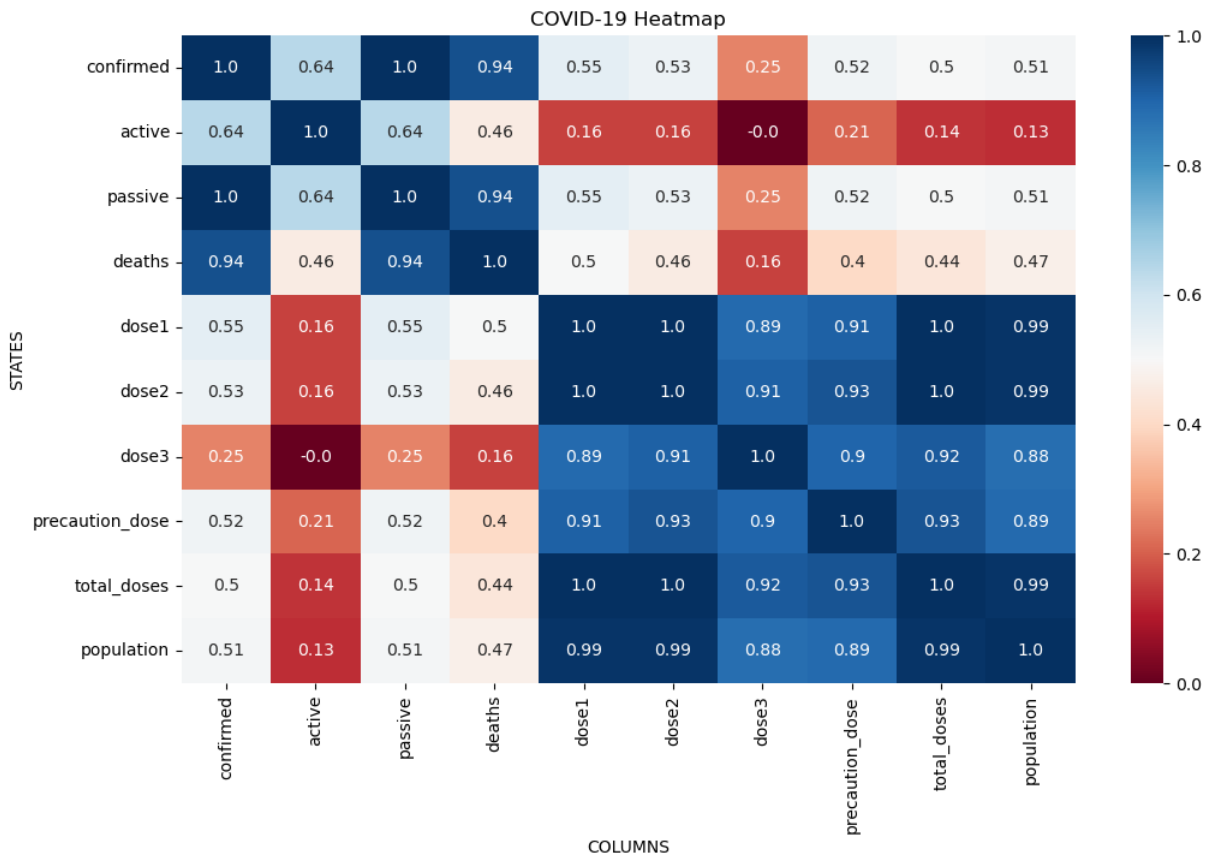


Figure 2.3: Distribution of three doses among the all states

2.3 Data Preprocessing

In our dataset first we check if there is any missing value or duplicate rows. For this our data has no duplicate rows and there is no missing data in the dataset. Now we check the values of active, passive, deaths and found that the total sum of these three columns is confirmed cases. Now we check the column of doses, we found that in some states the number of dose2 is greater than dose1 which is not clear as dose 2 is given to a person if he had taken dose1.Likewise sum of total doses in a state exceeds the sum of dose1, dose2, dose3, precaution dose. Again it is not clear how the total doses is greater than the sum of all doses.Hence we assume that the difference is because some doses are wasted and data of dose1 is accurate and we preprocess the data of dose2 according to dose1.

For this we change the values of dose2 if it exceeds dose1 then we change the value of dose2 to dose1 and if dose2 is smaller than dose1 than we don't change the value of dose2.And similarly we check the data of dose3 and if the value of dose3 is greater than dose2 then we replace dose3 with the value to dose2 and if it is smaller then we don't change the value of dose3. Similarly we check the data of precaution dose with dose3. Now we have seen that the total doses exceeds the sum of all doses. Here we will assume that the difference is because some doses are wasted.



STATES	confirmed	active	passive	deaths	dose1	dose2	dose3	precaution_dose	total_doses	population
confirmed	1.0	0.64	1.0	0.94	0.55	0.53	0.25	0.52	0.5	0.51
active	0.64	1.0	0.64	0.46	0.16	0.16	-0.0	0.21	0.14	0.13
passive	1.0	0.64	1.0	0.94	0.55	0.53	0.25	0.52	0.5	0.51
deaths	0.94	0.46	0.94	1.0	0.5	0.46	0.16	0.4	0.44	0.47
dose1	0.55	0.16	0.55	0.5	1.0	1.0	0.89	0.91	1.0	0.99
dose2	0.53	0.16	0.53	0.46	1.0	1.0	0.91	0.93	1.0	0.99
dose3	0.25	-0.0	0.25	0.16	0.89	0.91	1.0	0.9	0.92	0.88
precaution_dose	0.52	0.21	0.52	0.4	0.91	0.93	0.9	1.0	0.93	0.89
total_doses	0.5	0.14	0.5	0.44	1.0	1.0	0.92	0.93	1.0	0.99
population	0.51	0.13	0.51	0.47	0.99	0.99	0.88	0.89	0.99	1.0

Figure 2.4: Distribution of three doses among the all states

Now our data is cleaned and we will start our further process of normalization.We further divide the number of confirmed cases, active, passive and deaths with the population of each state. And convert the whole dataset to per 1000 person as each state has different population one has very smaller and one has very larger.After obtaining the new values we apply the formula for linear regression on total doses with deaths.

2.3.1 Evaluation and Update

According to our earlier analysis of covid-19 dataset, showed in the heatmap, there is single negative correlation between dose3 and active cases. However, after we modified the dataset the dose3 reduces the confirmed, active, passive and deaths cases. First we consider the total doses as

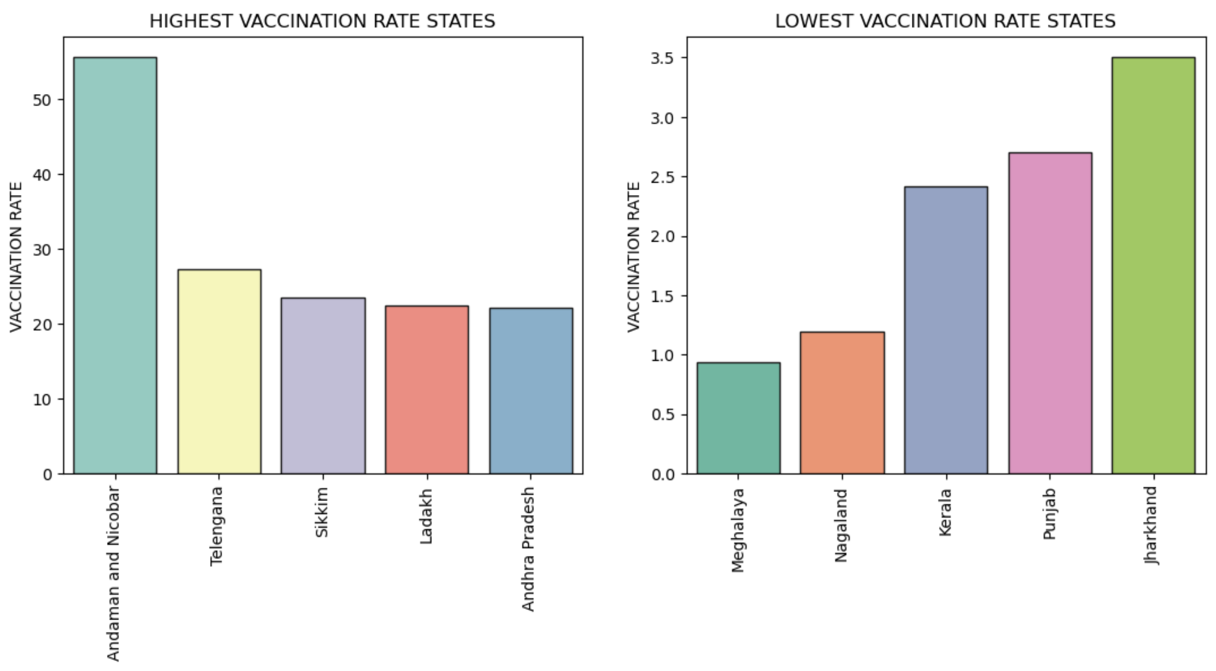


Figure 2.5: Top 5 states with highest and lowest Vaccination Rates

x and deaths as y. We measure the correlation ρ between them with the formula $\rho = corr(x, y)$. After calculating the correlation factor we calculate the linear regression coefficient a with the formula $a = \rho * std(y)/std(x)$, here $std(y)$ is the standard deviation of y i.e confirmed cases and $std(x)$ is the standard deviation of x i.e. total doses. After calculating the regression coefficient we calculate the intercept of y which is denoted by b. $b = mean(y) - a * mean(x)$. Now we put these values into the regression formula $y_c = ax + b$. Here y_c is the newly derived y. After finding the regression line we calculate the error with calculate y i.e. y_c and original y with the formula $error1 = 100 * (y - y_c)/y_c$ and $error2 = 100 * (y - y_c)/y$.

Now we apply our second approach, multi-linear regression. We would like to see if we consider doses separately, and how will it affect confirmed cases. For this we consider the independent variables as dose1, dose2, dose3, precaution dose. These are the input variables for x. The confirmed cases are represented by y.Now we apply the same linear regression formula for correlation calculation, a, b, y calculated, error1 and error2. The error we found is also known as the loss function. Figure 2.6 describes how much vaccine is wasted across the country which is

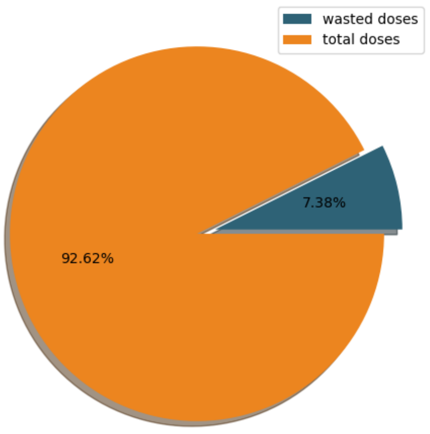
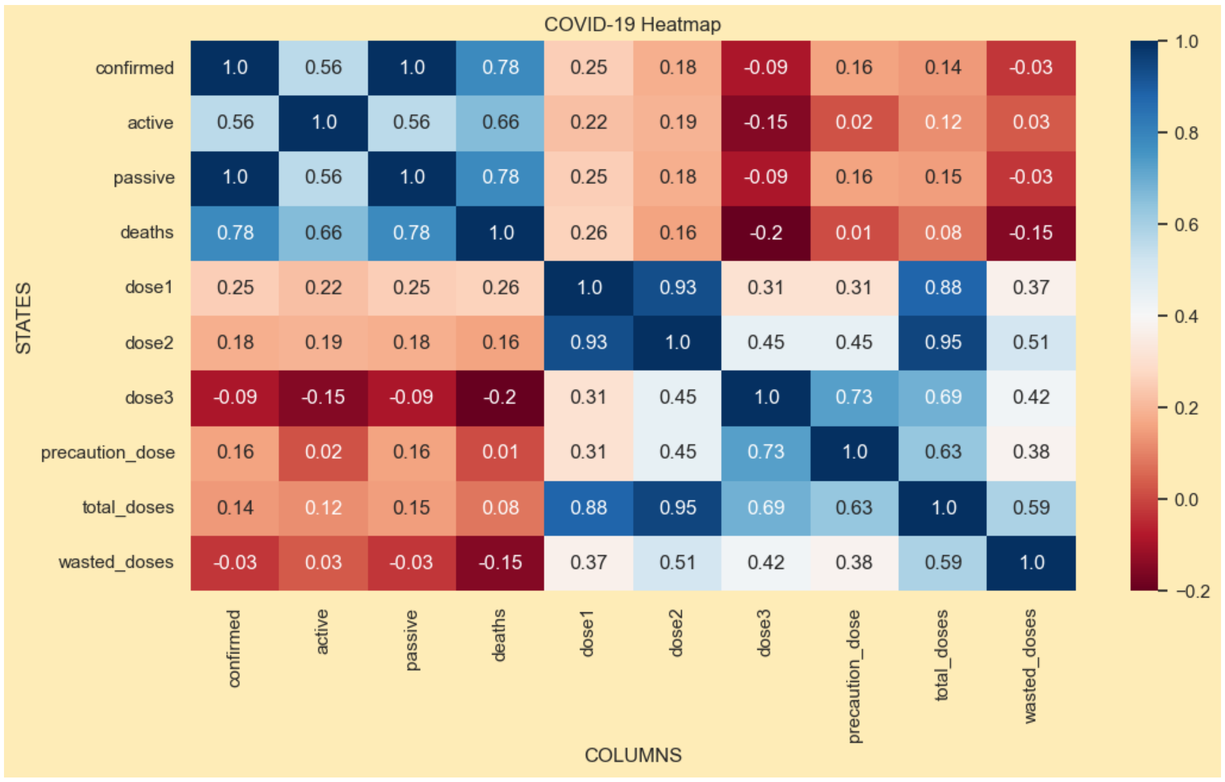


Figure 2.6: Distribution of dose wastage across country



	confirmed	active	passive	deaths	dose1	dose2	dose3	precaution_dose	total_doses	wasted_doses
confirmed	1.0	0.56	1.0	0.78	0.25	0.18	-0.09	0.16	0.14	-0.03
active	0.56	1.0	0.56	0.66	0.22	0.19	-0.15	0.02	0.12	0.03
passive	1.0	0.56	1.0	0.78	0.25	0.18	-0.09	0.16	0.15	-0.03
deaths	0.78	0.66	0.78	1.0	0.26	0.16	-0.2	0.01	0.08	-0.15
dose1	0.25	0.22	0.25	0.26	1.0	0.93	0.31	0.31	0.88	0.37
dose2	0.18	0.19	0.18	0.16	0.93	1.0	0.45	0.45	0.95	0.51
dose3	-0.09	-0.15	-0.09	-0.2	0.31	0.45	1.0	0.73	0.69	0.42
precaution_dose	0.16	0.02	0.16	0.01	0.31	0.45	0.73	1.0	0.63	0.38
total_doses	0.14	0.12	0.15	0.08	0.88	0.95	0.69	0.63	1.0	0.59
wasted_doses	-0.03	0.03	-0.03	-0.15	0.37	0.51	0.42	0.38	0.59	1.0

Figure 2.7: Heatmap after updating the missing information in the dataset

7.38% of total vaccine.

Chapter 3

Analytical Methodology

3.1 Introduction

The Covid-19 India dataset will be examined using regression models and Artificial Neural Networks (ANN), and the key ideas and procedures guiding these analytical methods are outlined in this part. Additionally, it describes how important they are for comprehending and forecasting the dynamics of the Covid-19 pandemic in India. We can investigate the relationships between different parameters and the quantity of verified instances by using regression models, such as multiple regression and linear regression. Artificial Neural Networks also reveal complex patterns and nonlinear interactions in the collection. This improves our comprehension of the underlying dynamics causing the virus to spread. We seek to obtain important insights into the variables driving the transmission and impact of the virus in various Indian states by using regression models and ANN on the Covid-19 India dataset. The use of regression models, as well as the construction and training of ANN architecture, will be covered in this part on analytical technique.

3.2 Regression

In regression, associations between variables are looked for. For instance, in covid-19 dataset, watch a number of active covid cases in each state to try to understand how factors like positive covid-19 rate, doses injected to population, recovery rate, and so on affect population deaths.

The data associated with each state represents one observation in this regression issue. The assumption is that while the active cases depends on these factors, dose1, dose2, and dose3 are independent characteristics. The mathematical relationship between total population, active case, passive cases, etc. may be established in a similar manner.

Regression analysis often involves the consideration of an interesting phenomena and a number of observations. There are at least two characteristics in each observation. We attempt to build a relationship between them under the presumption that at least one of the traits depends on the others. To put it another way, we need to discover a function that accurately maps certain traits or variables to others. The dependent variables, outputs, or reactions are referred to as the dependent features. The independent variables, inputs, regressors, or predictors are names for the independent characteristics.

3.2.1 Linear Regression

One of the most significant and often employed methods of regression analysis is linear regression. It is the simplest regression techniques. The simplicity of analyzing the results is one of its key advantages. We presume a linear connection between y and x when we execute linear regression of a dependent variable y on the collection of independent variables $x = (x_1, ..., x_r)$, where r is the number of predictors. The regression equation is represented by this equation:

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_r x_r + \epsilon \tag{3.1}$$

The regression coefficients are $\beta_0, \beta_1, ...\beta_r$ and ϵ is the random error. With $b_0, b_1, ..., b_r$, linear regression simply determines the anticipated weights or estimators of the regression coefficients. The estimated regression function is specified by these estimators as $f(x) = b_0 + b_1 x_1 + ... + b_r x_r$. The interdependence between the inputs and outputs should be adequately captured by this function. For each observation $i = 1, ..., n$, the estimated or anticipated response, $f(x_i)$, should be as near as feasible to the corresponding actual response, y_i. The residuals are the differences $y_i - f(x_i)$ for all observations $i = 1, ..., n$. Finding the weights with the least residuals (best projected weights) is the goal of regression.

The optimal weights are often obtained by minimizing the sum of squared residuals (SSR) for all observations $i = 1, ..., n$: $SSR = \sum_i (y_i - f(x_i))^2$. This strategy is named as method of ordinary least squares.

Regression Performance

Because of the reliance on the predictors x_i , there is some fluctuation in the actual replies $y_i, i = 1, ..., n$. However, there is also an extra intrinsic output variance. The amount of variation in y that can be described by the reliance on x using the specific regression model is shown by the coefficient of determination, abbreviated as R^2. A better fit and the ability of the model to account for the variance in the output with respect to the inputs are both indicated by a larger R^2. Sum of squared residuals $SSR = 0$ is equivalent to the value $R^2 = 1$. Given that the expected and actual response values are a perfect match, this is the best fit possible.

3.2.1.1 Simple Linear Regression

The simplest case of linear regression is simple or single-variate linear regression, as it has a single independent variable, $X = x$.

```python
def mean(values):
    return sum(values) / len(values)

def variance(values, mean):
    return sum([(x - mean) ** 2 for x in values])

def covariance(x, y, x_mean, y_mean):
    return sum([(x[i] - x_mean) * (y[i] - y_mean) for i in range(len(x))
])

def coefficients(x, y):
    x_mean = mean(x)
    y_mean = mean(y)
    b1 = covariance(x, y, x_mean, y_mean) / variance(x, x_mean)
    b0 = y_mean - b1 * x_mean
    return b0, b1
```

```python

def simple_linear_regression(x, y):
    b0, b1 = coefficients(x, y)
    predictions = [b0 + b1 * x[i] for i in range(len(x))]
    return b0, b1, predictions
```

Listing 3.1: Code implimentation of Linear Regression Equation

The provided Python code implements simple linear regression, a technique used to model the relationship between two variables by fitting a straight line to the data. The code consists of several functions: *mean, variance, covariance, coefficients, and simple linear regression*. The *mean* function calculates the average value of a given set of values. The *variance* function computes the *variance* of a set of values using the mean value. The *covariance* function calculates the covariance between two sets of values using their respective mean values. The *coefficients* function combines the variance and covariance calculations to determine the slope and intercept of the regression line. It uses the least squares method to find the best-fitting line.

Finally, the *simple linear regression* function applies the calculated coefficients to generate predictions for the given data points. It uses the regression equation $y = b0 + b1 * x$ to calculate the predicted values for each x value in the input. Overall, this code provides a basic implementation of simple linear regression and can be used to analyze and predict the relationship between two variables.

```
# Result: Regression slope for confirmed cases against the dose1
    Intercept: -11.295686815529749
    Slope: 0.09501872297595909
```

The figure 3.1 represents the results of the simple linear regression model applied to the given covid-19 data. The calculated intercept value for the linear regression line is approximately -11.2957. The calculated slope value for the linear regression line is approximately 0.0950. These are the predicted values of the dependent variable (in this case, 'confirmed' cases) based on the independent variable (in this case, 'dose1'). The predictions represent the y-values of the regression line for each corresponding x-value. The predictions list contains the predicted

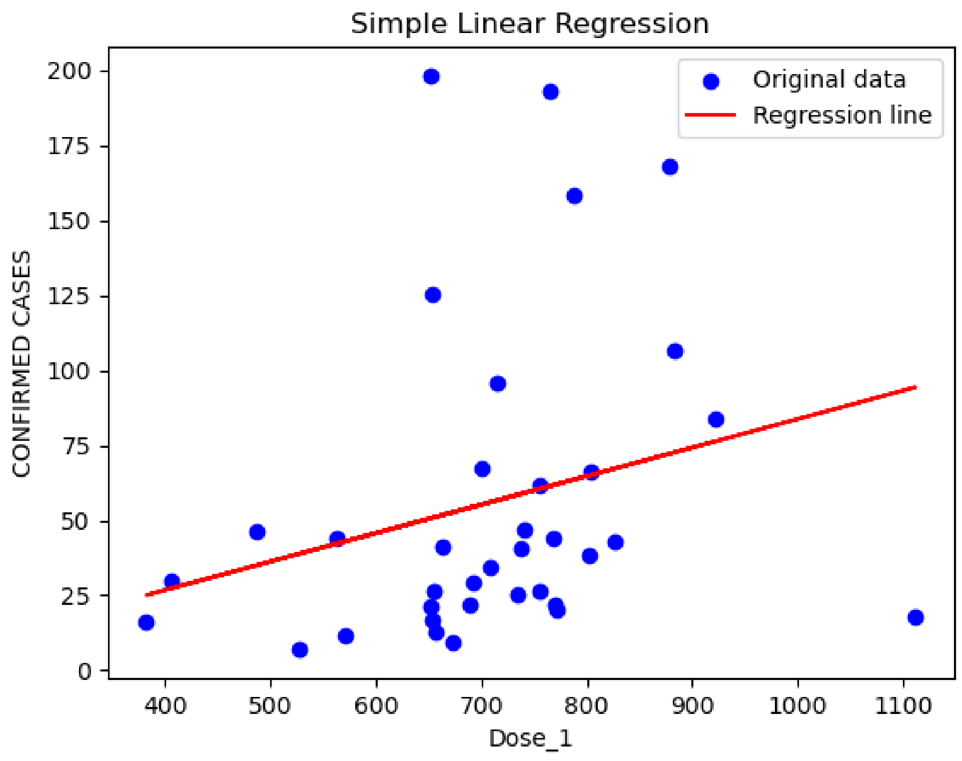


Figure 3.1: Regression slope for confirmed cases against the dose1

'confirmed' values for each 'dose1' value in the dataset. For example, the first prediction value (index 0) is approximately 58.5407, which corresponds to the first 'dose1' value in the dataset. The intercept represents the expected 'confirmed' value when the 'dose1' value is zero, and the slope represents the change in 'confirmed' for each unit increase in 'dose1'. The predictions can be used to estimate 'confirmed' values for new 'dose1' values that were not present in the original dataset.

```
# Result: Regression slope for number of deaths against the total doses
    Intercept: -11.295686815529749
    Slope: 0.09501872297595909
```

In figure 3.2, performs a simple linear regression analysis on the relationship between the number of total vaccine doses administered *total doses* and the number of deaths. The calculated intercept value of approximately 0.3696 represents the estimated number of deaths when the number of total vaccine doses administered is zero. However, this interpretation may not be practically meaningful in the context of COVID-19 since it is highly unlikely to have zero vaccine doses

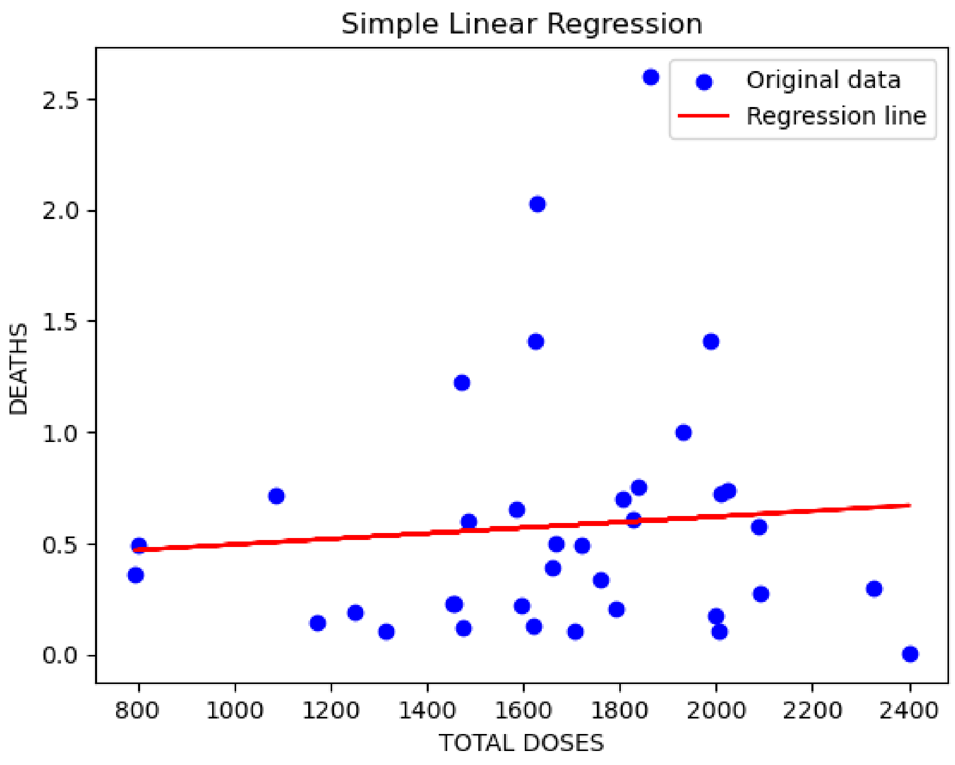


Figure 3.2: Regression slope for number of deaths against the total doses

administered. The calculated slope value of approximately 0.0001254 indicates that, on average, there is a positive association between the number of total vaccine doses administered and the number of deaths. However, the slope is very small, suggesting a weak relationship between these variables. For each additional total vaccine dose administered, the number of deaths is estimated to increase by approximately 0.0001254.

The predictions obtained from the regression analysis provide estimated values of deaths corresponding to each total vaccine dose value. These estimates are based on the fitted regression line. For example, a total vaccine dose value of 734.9754018 is associated with a predicted number of deaths of approximately 0.6613. The small slope value suggests that the number of total vaccine doses administered may not be a strong predictor of the number of deaths. Additionally, caution should be exercised when making predictions outside the range of total vaccine dose values in the dataset, as the model's accuracy may decrease for extrapolated values. These findings contribute to our understanding of the potential impact of vaccination efforts on reducing mortality rates.

3.2.1.2 Multiple Linear Regression

Regression with two or more independent variables is known as multiple or multivariate regression. The estimated regression function is $f(x_1, x_2) = b_0 + b_1x_1 + b_2x_2$ if there are only two independent variables. In three dimensions, it depicts a regression plane. In order to produce the least amount of sum of squared residuals (SSR) while keeping this plane as near to the real replies as feasible, regression must be used to estimate the weights b_0, b_1, and b_2 values. Similar, but more generic, is the scenario of more than two independent variables. There are $r + 1$ weights to be found when the number of inputs is r, and the predicted regression function is $f(x_1, ..., x_r) = b_0 + b_1x_1 + b_rx_r$.

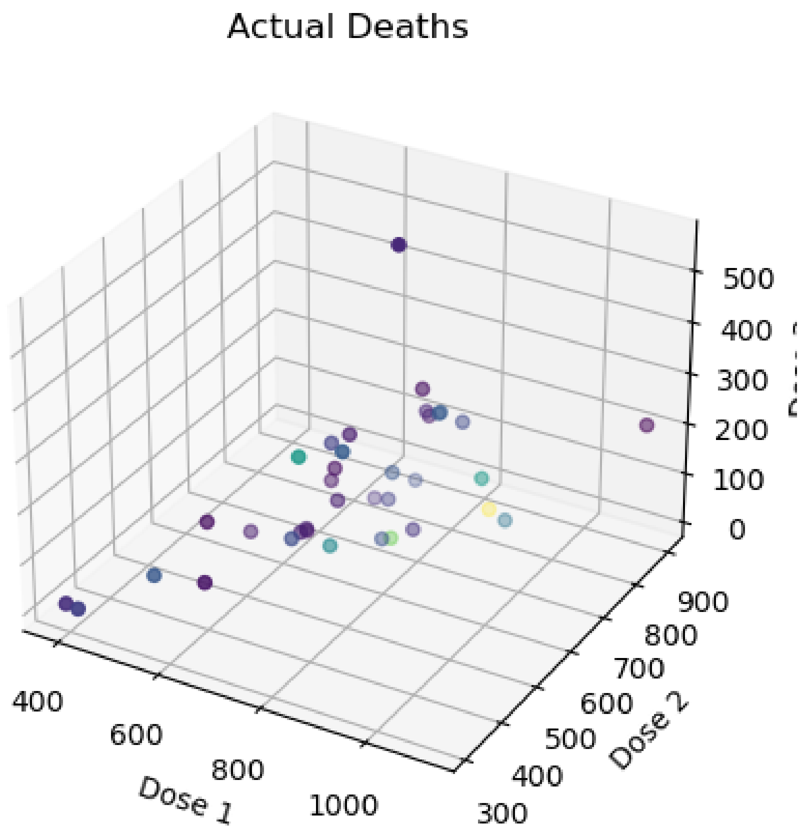


Figure 3.3: 3D-Curve fitting on the predictions for the Covid-19 data

The figure 3.4 represents the curve fitting on the predictions for the covid-19 dataset. The scatter points in the plot represent the actual number of deaths for each data point in the dataset. Each

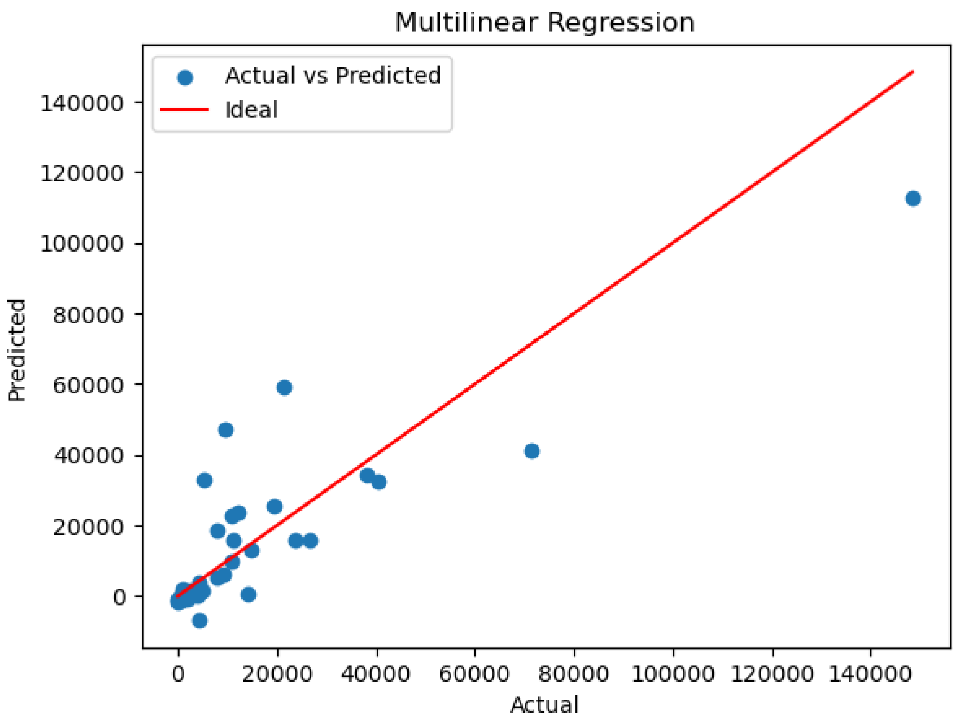


Figure 3.4: Curve fitting on the predictions for the Covid-19 data

point corresponds to a specific data index. The red line in the plot represents the predicted number of deaths based on the multiple linear regression model. The line is fitted to approximate the relationship between the independent variables (dose1, dose2, dose3,precaution dose) and the dependent variable (deaths). By comparing the scatter points (actual deaths) with the red line (predicted deaths), you can observe the following:

- If the scatter points and the red line closely align, it indicates that the linear regression model has successfully captured the underlying relationship between the independent variables and the dependent variable. This suggests that the model's predictions are in good agreement with the actual data.

- If there is a significant deviation between the scatter points and the red line, it indicates that the linear regression model may not be accurately capturing the relationship in the data. In such cases, alternative models or additional features may need to be considered to improve the accuracy of the predictions.

Overall, figure 3.4 helps visualize how well the linear regression model is fitting the data and provides insights into the relationship between the independent variables and the dependent variable.

Table 3.1: Multi Linear Regression Prediction: Actual vs. Predicted Values

Y (Actual)	Y_Prediction	Y (Actual)	Y_Prediction
129	-1731.61	12302	23732.33
14733	12966.75	1181	256.54
296	-249.02	14146	428.03
8035	18681.98	4	-796.85
26521	15885.60	10714	22735.21
4013	175.85	4213	3949.69
11043	15885.30	4785	1415.29
10714	22735.21	5331	33120.39
40307	32674.20	231	-843.48
71552	41434.96	52	-1333.83
231	-843.48	10776	9651.91
52	-1333.83	148416	112811.79
2149	1514.41	7751	5253.51
1624	1051.25	21532	59293.51
726	-175.82		
782	-4.99		

To analyze the statistical performance of the predicted values in table 3.1 compared to the actual values, we can calculate the (i) Mean Absolute Error (MAE), (ii) Mean Squared Error (MSE), and (iii) Root Mean Squared Error (RMSE) metrics as follows:

```
# Calculate MAE, MSE, and RMSE

   MAE = np.mean(np.abs(Y_actual - Y_prediction))
   MSE = np.mean((Y_actual - Y_prediction)**2)
   RMSE = np.sqrt(MSE)

```

```
Output: Staticial Performance of Y (Actual) and Y (Prediction)

    Mean Absolute Error (MAE): 7371.421599999999
    Mean Squared Error (MSE): 150379706.038576
    Root Mean Squared Error (RMSE): 12262.940350445158
```

These metrics provide an indication of the accuracy and performance of the multi-linear regression model in predicting the COVID-19 data. Lower values for MAE, MSE, and RMSE indicate better predictive performance, with RMSE being the most commonly used metric for evaluating regression models.

3.3 Artificial Neural Network (ANN)

Neural networks consist of artificial neurons arranged in layers, connected by weights. Input units receive external information, while output units indicate the network's response. Hidden units form the majority of the network and are located between input and output units. Most neural networks are fully connected, with each unit connected to all units in adjacent layers. The connections are represented by weights, which can be positive or negative, determining the influence between units. The weights resemble the way brain cells interact through synapses. Neural networks can vary in size, with dozens to millions of units, enabling them to learn, recognize, and process information from the external world.

Neural network works in two ways. When it is being trained or after being trained, the input units feed the network with information patterns, which then cause the layers of hidden units to be activated and bring those patterns to the output units. A feedforward network is the term used for this popular layout.Some units don't always ”fire” at all. The inputs are multiplied by the weights of the connections they travel along as they are received by each unit from the units to its left. Each unit adds up all the inputs it gets in this manner, and (in the most basic sort of network) if the sum is more than a predetermined threshold value, the unit ”fires” and activates the units it is connected to (those to its right).

A node layer of an artificial neural network (ANN) consists of an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, is connected to others and has a weight and threshold that go along with it. Any node whose output exceeds the defined threshold value is activated and begins providing data to the network's uppermost layer.

Training data is essential for neural networks to develop and enhance their accuracy over time.Weights are assigned after a determination of the input layer. These weights aid in determining the significance of each variable, with bigger weights having a greater impact on the final result than smaller ones. Next, each input is multiplied by its corresponding weight before being added together. The output is then determined by an activation function once the output has been passed through it. This ”fires” (or activates) the node, sending data to the network's next layer, if the output rises beyond a certain threshold. As a result, the input of one node becomes the output of the following node. This neural network is a feedforward network since data is sent from one layer to the next layer throughout this procedure.

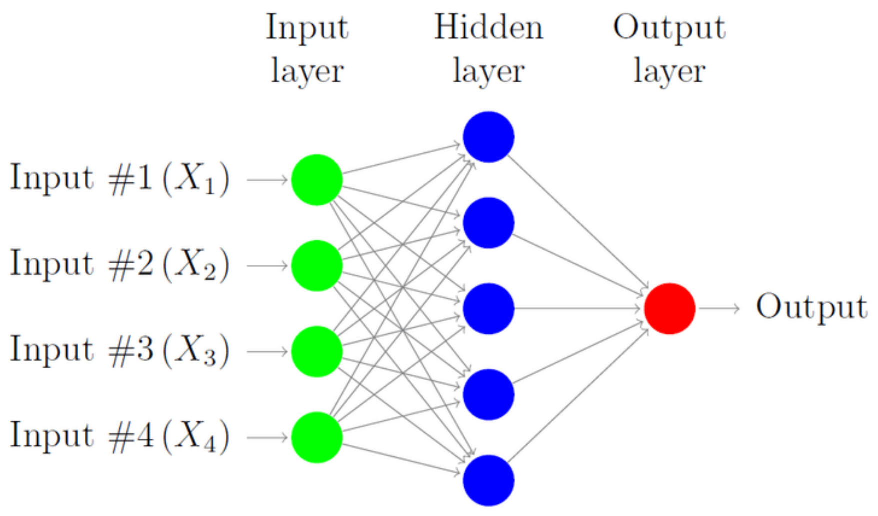


Figure 3.5: Layer Architecture of Neural Network

Algorithm for neural network:

- Initialize the weights and biases for the input layer, hidden layer, and output layer.
- Define the activation function which is used for the hidden layer and output layer (e.g., sigmoid, ReLU, or tanh).
- Set the learning rate and the number of epochs for training.
- Perform Forward Propagation for each epochs.

- Perform Backpropagation
- After training, use the trained model to make predictions on new data by performing forward propagation with the updated weights and biases.
- Evaluate the performance of the model using appropriate evaluation metrics (e.g., accuracy, precision, recall, or mean squared error).

Formulas

$$U = f(w_1x_1 + w_2x_2 + \ldots + w_nx_n) \tag{3.2}$$

The neuron fires if

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n > a \tag{3.3}$$

That is,

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n + a(-1) > 0 \tag{3.4}$$

Alternatively, we can rewrite it as

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n + w_{n+1}x_{n+1} > 0 \tag{3.5}$$

Here, a represents the bias.As U is the input

$$\begin{aligned} \text{Observed } y &= (y_1, y_2) \\ \text{Computed } \hat{y} &= (\hat{y}_1, \hat{y}_2) \\ \text{Error } e &= y - \hat{y} = (y_1 - \hat{y}_1, y_2 - \hat{y}_2) \\ \text{Squared Error } E &= \langle y - \hat{y}, y - \hat{y} \rangle = \langle y - \text{th}(x \cdot W) \cdot V, y - \text{th}(x \cdot W) \cdot V \rangle \end{aligned}$$

Here, y represents the observed values, $\hat{y}$ represents the computed values, e represents the error, and E represents the squared error. The symbols x, W, and V are variables involved in the calculations, and th$(\cdot)$ denotes a threshold function. We aim to minimize the quadratic error over the unknown matrices W and V. The quadratic error is given by:

$$E = \frac{\langle y - \hat{y}, y - \hat{y} \rangle}{2} = \frac{\langle y - \text{th}(x \cdot W) \cdot V, y - \text{th}(x \cdot W) \cdot V \rangle}{2}$$

Here, y represents the observed values, $\hat{y}$ represents the computed values, and x is the input vector. The function th$(\cdot)$ is defined as:

$$\text{th}(u) = \frac{2(1 + e^{-2u}) - 1}{1 + e^{-2u}}$$

For steepest descent

$$E = \frac{[y_1 - \sum_j v_{j1}\text{th}(w_{1j}x_1 + w_{2j}x_2 + w_{3j}x_3)]^2}{2} + \frac{[y_2 - \sum_j v_{j2}\text{th}(w_{1j}x_1 + w_{2j}x_2 + w_{3j}x_3)]^2}{2}$$

Error back-propagation mimicking Steepest descent for squared error E

$$v_{jk}(t+1) = v_{jk}(t) - \mu\frac{\partial E}{\partial v_{jk}}, \quad w_{ij}(t+1) = w_{ij}(t) - \mu\frac{\partial E}{\partial w_{ij}} \quad (i \in I, j \in II, k \in III)$$

$$\frac{\partial E}{\partial v_{jk}} = -(y_k - \hat{y}_k) \cdot \text{th}\left(\sum_i x_i \cdot w_{ij}\right)$$

$$\frac{\partial E}{\partial w_{ij}} = \sum_k \left[-(y_k - \hat{y}_k) \cdot v_{jk} \cdot (1 + \text{th}\left(\sum_i x_i \cdot w_{ij}\right)) \cdot (1 - \text{th}\left(\sum_i x_i \cdot w_{ij}\right)) \cdot x_i\right]$$

Algorithm for back error propagation

- Specify NE, number of epochs. Standardize features to [-1,1] interval. Generate W, V N(0,1) randomly.
- Epoch. In a loop over objects (x,y)
- 1.1. Forward computation: Find ŷ=F(x) and error e=y – ŷ
- 1.2. Error back-propagation: Use formulas to estimate derivatives of E
- 1.3. Weights update: Use formulas to update V and W

Halting test: If the number of epochs reached NE, output e, E, V, W and halt; else: randomize the order of objects and go to 1.

here for our data we have divided the data into two parts training and test. The test part includes 20% of data and training has 80%. The number of hidden layers we have taken are 50.Number of epoch=1000 and the activation function we have used is ”relu”.After computing the mean squared error loss we found that the loss is 40%

Chapter 4

Results and Discussion

In simple linear regression model the mean squared error is which is very high. In multilinear regression model the which is also high but compared to simple linear regression the loss is reduced. In neural network the losses are 40%. Hence we can say that for this type of dataset the neural network approach is better.
This dataset has some limitations as there is no clarity what are precaution doses and a large number of doses are wasted as we assumed because there is a difference between total doses given and the doses actually given.Further the dose2 was higher than dose1 which is not possible.Maybe we have this much error because we modified the dataset as there are some discrepencies in the dataset. We can further try other approaches also to find out how can we improve the large losses occured during regression.

Appendix

Appendix I

Exploratory Analysis Supplementary Code

I.1 Exploratory Data Analysis on Covid-19 dataset

Exploratory Data Analysis

June 19, 2023

1 Exploratory Data Analysis on Covid-19 dataset

[1]:
```
# Import all required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

[2]:
```
# Read the CSV file into a pandas DataFrame
df = pd.read_csv('covid.csv')
```

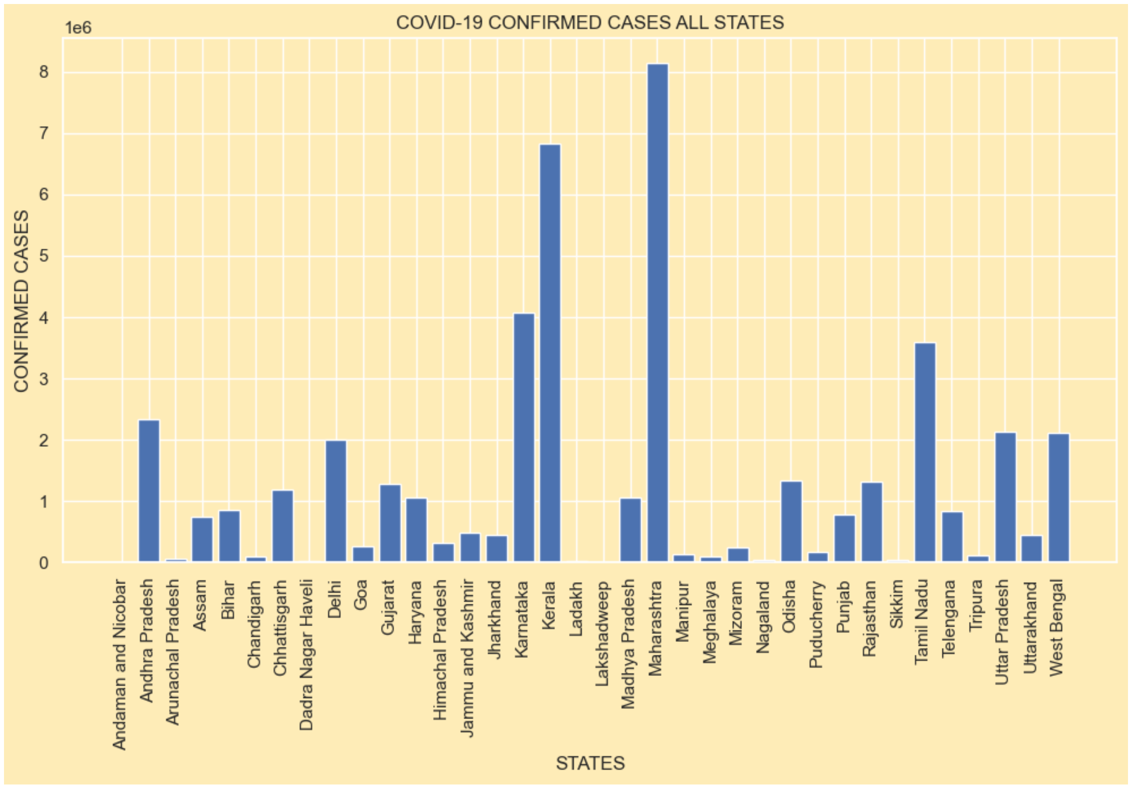
[3]:
```
df.head()
```

[3]:
```
                 state  confirmed  active  passive  deaths     dose1  \
0  Andaman and Nicobar      10742       1    10612     129    313284
1       Andhra Pradesh    2339067       3  2324331   14733  40643161
2    Arunachal Pradesh      66890       0    66594     296    860442
3                Assam     746100       0   738065    8035  22549957
4                Bihar     851379      15   839062   12302  62944633

      dose2     dose3  precaution_dose  total_doses  population
0    320383    236936            53427       991263      426251
1  43549055  11703273          6579565    110556756    52883163
2    747177     72403            58618      1911760     1528296
3  20561790   2082670          1259853     50284713    34586234
4  59144387  11983504          3868082    157197041   119461013
```

[4]:
```
sns.set(rc={'axes.facecolor':'#FEECB7', 'figure.facecolor':'#FEECB7'})
```

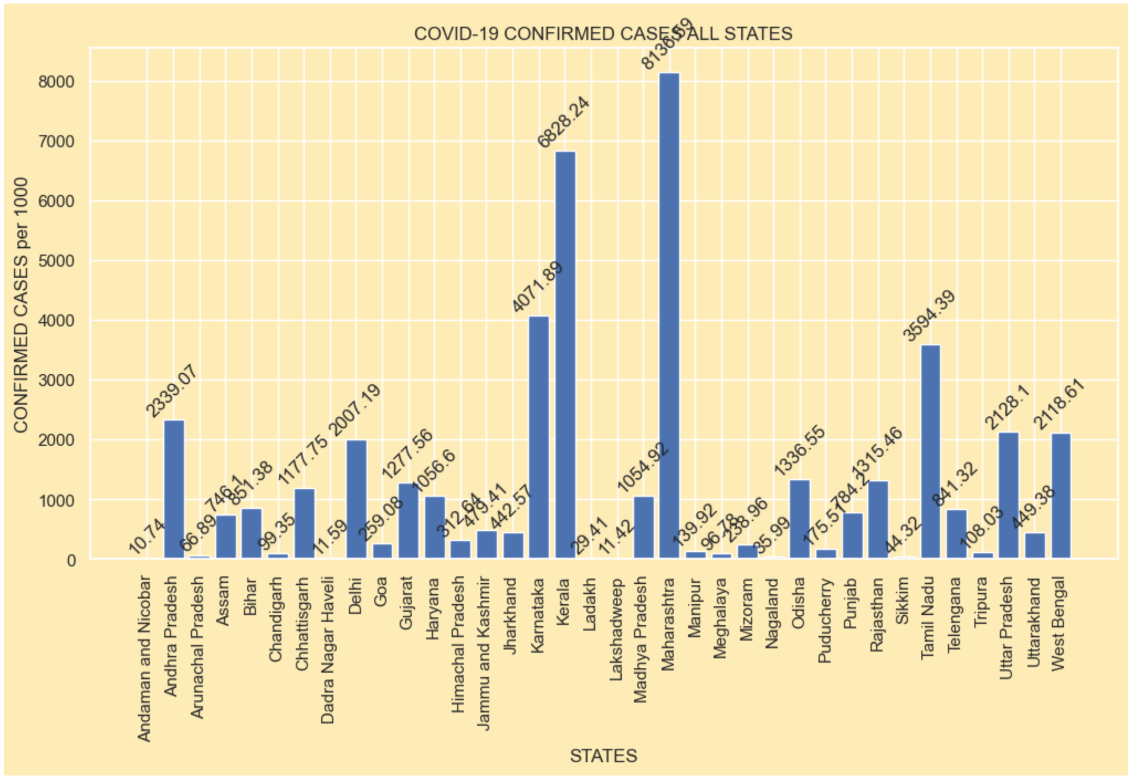
1.1 Visualization

[5]:
```
# Example: Bar plot of confirmed cases by state
plt.figure(figsize=(12, 6))
plt.bar(df['state'], df['confirmed'])
plt.xlabel('STATES')
plt.ylabel('CONFIRMED CASES')
plt.title('COVID-19 CONFIRMED CASES ALL STATES')
plt.xticks(rotation='vertical')
plt.show()
```

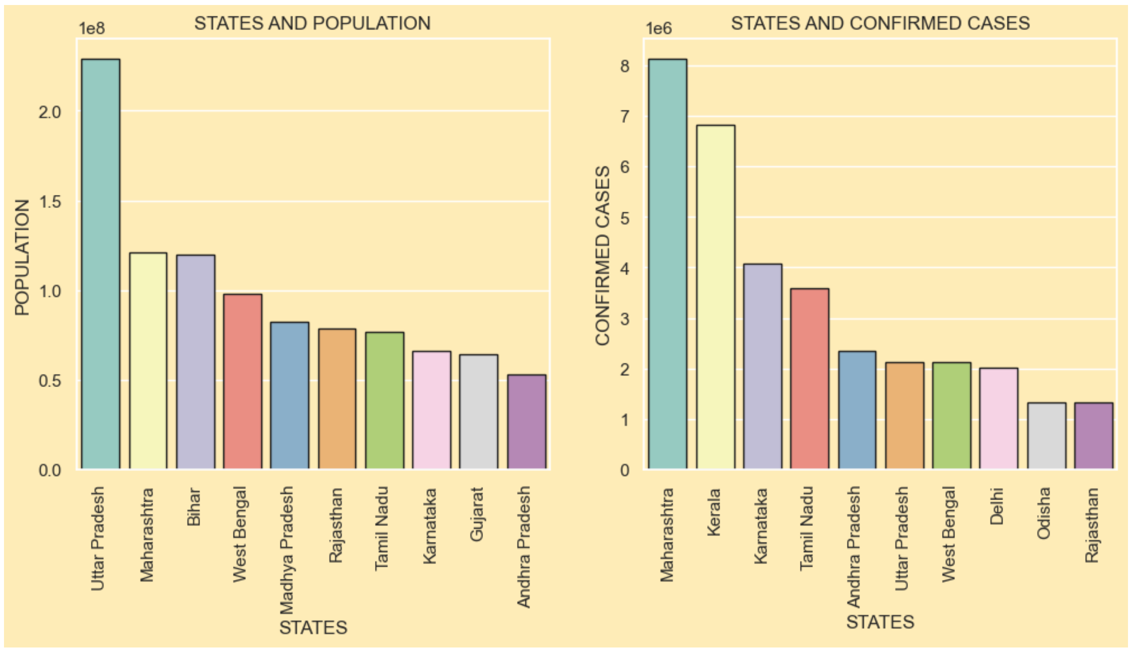


```
[6]: import pandas as pd
     import matplotlib.pyplot as plt

     # Read the CSV file
     df = pd.read_csv("covid.csv")

     plt.figure(figsize=(12, 6))
     bars = plt.bar(df['state'], (df['confirmed']/1000).round(2))
     plt.xlabel('STATES')
     plt.ylabel('CONFIRMED CASES per 1000')
     plt.title('COVID-19 CONFIRMED CASES ALL STATES')
     plt.xticks(rotation='vertical')

     # Write values on top of each bar
     for bar in bars:
         yval = bar.get_height()
         plt.text(bar.get_x() + bar.get_width() / 2, yval, yval, ha='center',␣
       ↪va='bottom', rotation=45)
```

```
plt.show()
```



[7]:
```
#Top states by population?
states = list(df.sort_values(by='population',ascending=False).state)[:10]
values = df.sort_values(by='population',ascending=False)['population'].values[:
 ↪10]
plt.figure(figsize=(12,6))
plt.subplot(121)
sns.barplot(x=states,y=values,palette='Set3',edgecolor='.1')
plt.xticks(rotation='vertical')
plt.xlabel('STATES')
plt.ylabel('POPULATION')
plt.title('STATES AND POPULATION')

#Top states by number of cases?
states = list(df.sort_values(by='confirmed',ascending=False).state)[:10]
values = df.sort_values(by='confirmed',ascending=False)['confirmed'].values[:10]
# plt.figure(figsize=(12,5))
plt.subplot(122)
sns.barplot(x=states,y=values,palette='Set3',edgecolor='.1')
plt.xticks(rotation='vertical')
plt.xlabel('STATES')
plt.ylabel('CONFIRMED CASES')
plt.title('STATES AND CONFIRMED CASES')
plt.show()
```



[8]:
```
#Which states have highest postivity rate?
df['Positivity rate'] = df['confirmed']*100/df['population']

states_pos = df.sort_values(by='Positivity rate',ascending=False)['state'].
 ↪values[:5]

print('States with high postivity are')
for i in states_pos:
    print(i,' ------> postivity rate of ',df[df['state']==str(i)]['Positivity␣
 ↪rate'].values[0],'%')
```

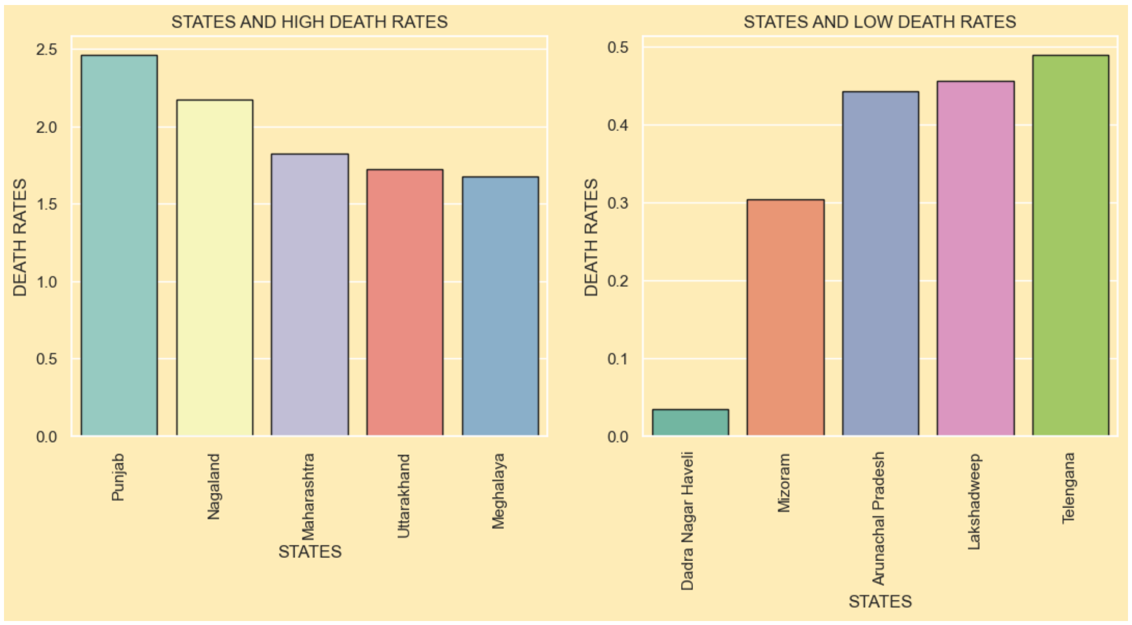
```
States with high postivity are
Mizoram  ------> postivity rate of  19.81502088768083 %
Kerala  ------> postivity rate of  19.32655924187357 %
Goa  ------> postivity rate of  16.79332361043591 %
Lakshadweep  ------> postivity rate of  15.808059825508932 %
Puducherry  ------> postivity rate of  12.55685204409794 %
```

[9]:
```
#Which states have least postivity rate?

states_pos = df.sort_values(by='Positivity rate',ascending=True)['state'].
 ↪values[:5]
```

```
print('States with low postivity are')
for i in states_pos:
    print(i,' ------> postivity rate of ',df[df['state']==str(i)]['Positivity␣
↪rate'].values[0],'%')
```

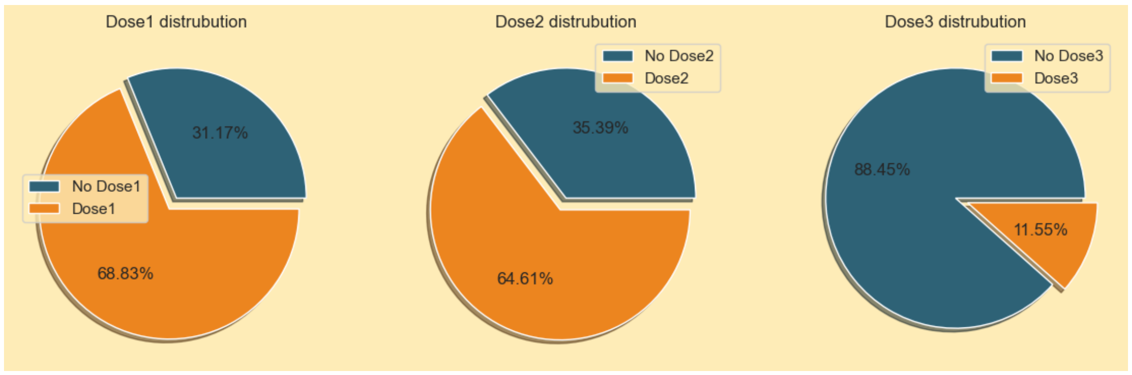
```
States with low postivity are
Bihar  ------> postivity rate of  0.7126835597819684 %
Uttar Pradesh  ------> postivity rate of  0.929466599913114 %
Jharkhand  ------> postivity rate of  1.185588905264543 %
Madhya Pradesh  ------> postivity rate of  1.2811297280139624 %
Nagaland  ------> postivity rate of  1.6437239899383227 %
```

[10]:
```
#States with high and low death rates?
df['death rate']=df['deaths']*100/df['confirmed']
states_death_high = df.sort_values(by='death rate',ascending=False)['state'].
↪values[:5]
values_high = df.sort_values(by='death rate',ascending=False)['death rate'].
↪values[:5]

states_death_low = df.sort_values(by='death rate',ascending=True)['state'].
↪values[:5]
values_low = df.sort_values(by='death rate',ascending=True)['death rate'].
↪values[:5]

plt.figure(figsize=(12,6))
plt.subplot(121)
sns.barplot(x=states_death_high,y=values_high,palette='Set3',edgecolor='.1')
plt.xticks(rotation='vertical')
plt.xlabel('STATES')
plt.ylabel('DEATH RATES')
plt.title('STATES AND HIGH DEATH RATES')

plt.subplot(122)
sns.barplot(x=states_death_low,y=values_low,palette='Set2',edgecolor='.1')
plt.xticks(rotation='vertical')
plt.xlabel('STATES')
plt.ylabel('DEATH RATES')
plt.title('STATES AND LOW DEATH RATES')
plt.show()
```



[11]:
```
#Total vaccinated stats in india

pop = df.sum()['population']
dose1 = df.sum()['dose1']
dose2 = df.sum()['dose2']
dose3 = df.sum()['dose3']


plt.figure(figsize=(14,14))
plt.subplot(131)
plt.pie(x=[pop-dose1,dose1],autopct='%1.2f%%',shadow=True,explode=(0,0.
 ↪1),colors=('#2E6276','#EC851F'))
plt.legend(['No Dose1','Dose1'])
plt.title('Dose1 distrubution')
plt.subplot(132)
plt.pie(x=[pop-dose2,dose2],autopct='%1.2f%%',shadow=True,explode=(0,0.
 ↪1),colors=('#2E6276','#EC851F'))
plt.legend(['No Dose2','Dose2'])
plt.title('Dose2 distrubution')
plt.subplot(133)
plt.pie(x=[pop-dose3,dose3],autopct='%1.2f%%',shadow=True,explode=(0,0.
 ↪1),colors=('#2E6276','#EC851F'))
plt.legend(['No Dose3','Dose3'])
plt.title('Dose3 distrubution')
```

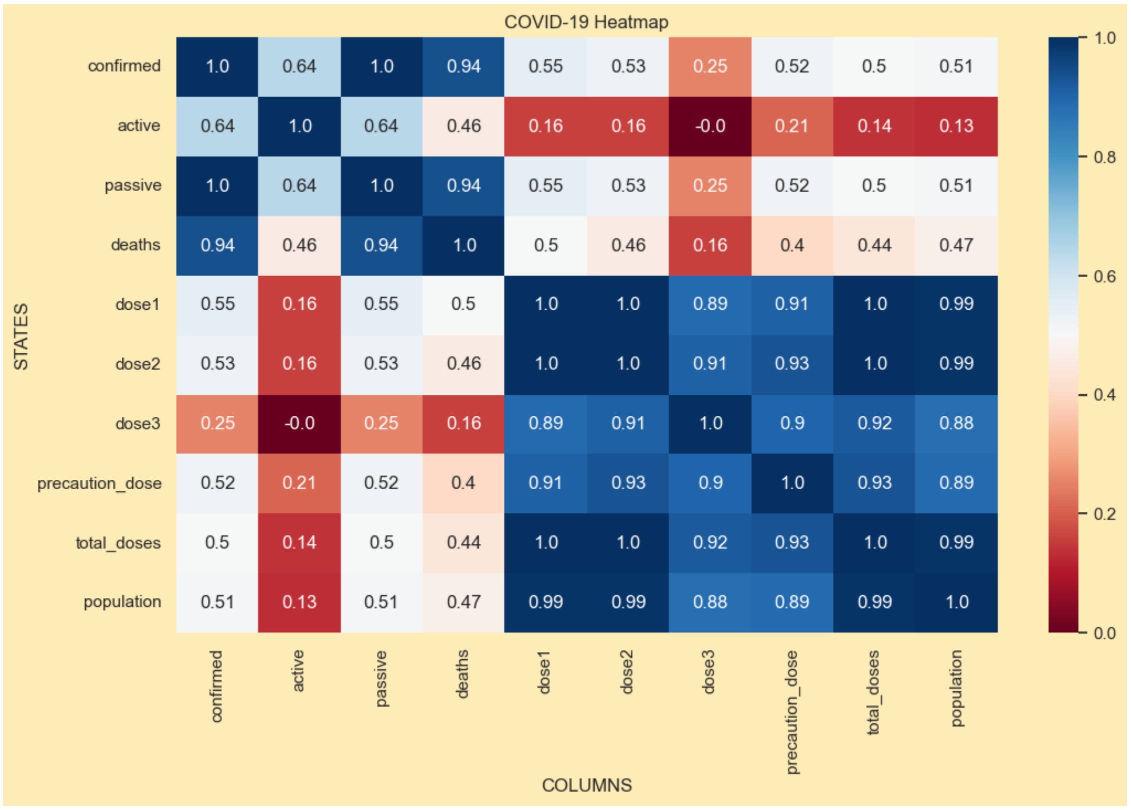
[11]:
```
Text(0.5, 1.0, 'Dose3 distrubution')
```



```
[12]: #Which states adminstered most number of doses
      states_doses = df.sort_values(by='total_doses',ascending=False)['state'].
       ↪values[:5]

      print(states_doses)
```

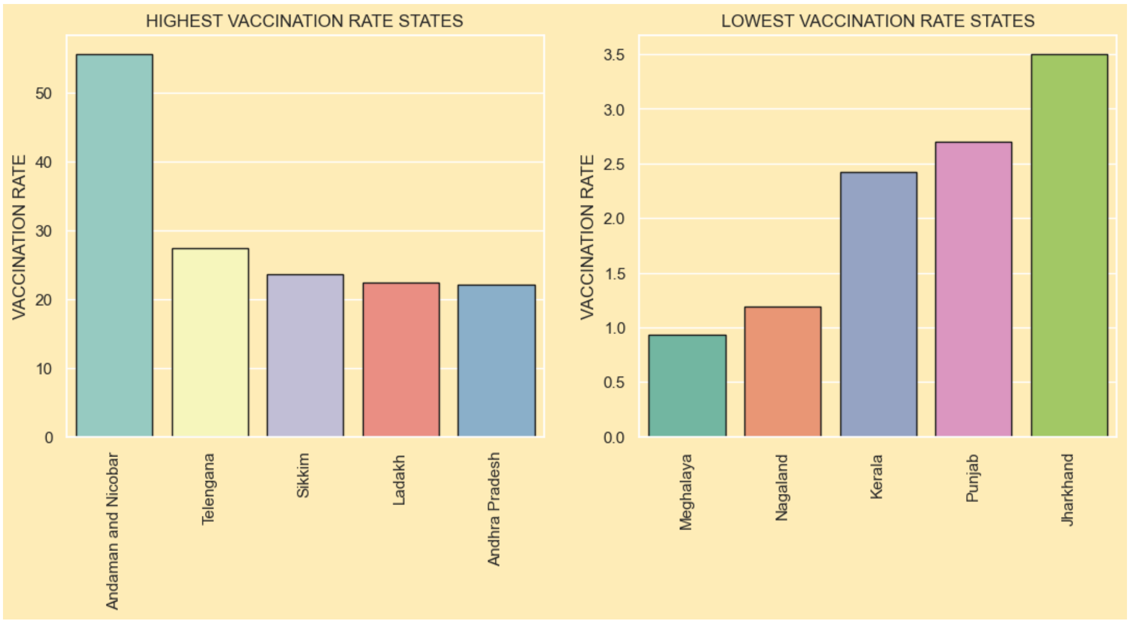
```
['Uttar Pradesh' 'Maharashtra' 'Bihar' 'West Bengal' 'Madhya Pradesh']
```

```
[13]: df = pd.read_csv('covid.csv')

      # set the column as index
      df_index = df.set_index('state')

      # create heatmap using seaborn
      plt.figure(figsize=(12, 7))
      sns.heatmap(df_index.corr().round(2), annot=True, cmap='RdBu', fmt=',')
      plt.title('COVID-19 Heatmap')
      plt.xlabel('COLUMNS')
      plt.ylabel('STATES')
      plt.show()
```



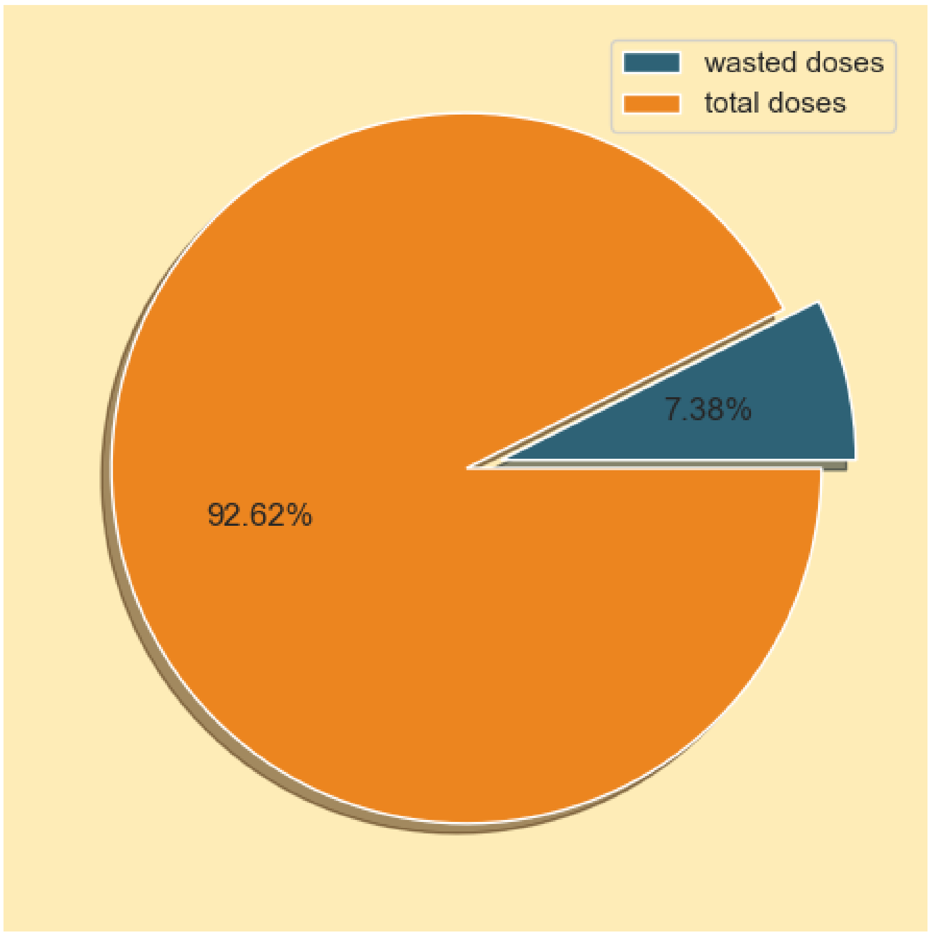
	confirmed	active	passive	deaths	dose1	dose2	dose3	precaution_dose	total_doses	population
confirmed	1.0	0.64	1.0	0.94	0.55	0.53	0.25	0.52	0.5	0.51
active	0.64	1.0	0.64	0.46	0.16	0.16	-0.0	0.21	0.14	0.13
passive	1.0	0.64	1.0	0.94	0.55	0.53	0.25	0.52	0.5	0.51
deaths	0.94	0.46	0.94	1.0	0.5	0.46	0.16	0.4	0.44	0.47
dose1	0.55	0.16	0.55	0.5	1.0	1.0	0.89	0.91	1.0	0.99
dose2	0.53	0.16	0.53	0.46	1.0	1.0	0.91	0.93	1.0	0.99
dose3	0.25	-0.0	0.25	0.16	0.89	0.91	1.0	0.9	0.92	0.88
precaution_dose	0.52	0.21	0.52	0.4	0.91	0.93	0.9	1.0	0.93	0.89
total_doses	0.5	0.14	0.5	0.44	1.0	1.0	0.92	0.93	1.0	0.99
population	0.51	0.13	0.51	0.47	0.99	0.99	0.88	0.89	0.99	1.0

[14]:
```
#Which states had most/least vaccinated population

df['Vaccinated rate'] = df['dose3']*100/df['population']
states_most_vaccinated = df.sort_values(by='Vaccinated rate',ascending=False).
  ↪state.values[:5]
values_most = df.sort_values(by='Vaccinated rate',ascending=False)['Vaccinated␣
  ↪rate'].values[:5]

states_least_vaccinated = df.sort_values(by='Vaccinated rate',ascending=True).
  ↪state.values[:5]
values_least = df.sort_values(by='Vaccinated rate',ascending=True)['Vaccinated␣
  ↪rate'].values[:5]

plt.figure(figsize=(13,5))
plt.subplot(121)
sns.barplot(x=states_most_vaccinated,y=values_most,palette='Set3',edgecolor='.
  ↪1')
plt.xticks(rotation='vertical')
plt.xlabel('')
plt.ylabel('VACCINATION RATE')
plt.title('HIGHEST VACCINATION RATE STATES')
plt.subplot(122)
sns.barplot(x=states_least_vaccinated,y=values_least,palette='Set2',edgecolor='.
  ↪1')
plt.xticks(rotation='vertical')
plt.xlabel('')
plt.ylabel('VACCINATION RATE')
plt.title('LOWEST VACCINATION RATE STATES')
plt.show()
```



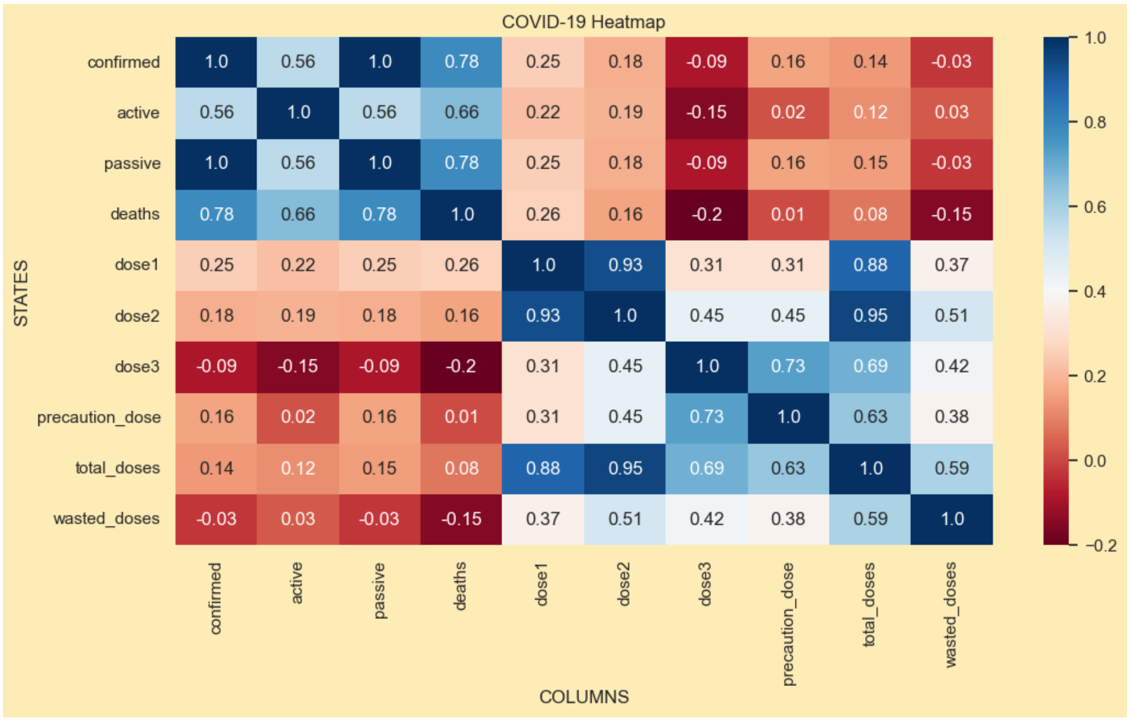
```
[15]: df_new = pd.read_csv('covid_new.csv')

      #distributuion of dose wastage across country
      plt.figure(figsize=(6,6))

      wasted = df_new.sum()['wasted_doses']
      total = df_new.sum()['total_doses']

      plt.pie(x=[wasted,total],colors=('#2E6276','#EC851F'),shadow=True,explode=(0.
        ↪1,0),autopct='%1.2f%%')
      plt.legend(['wasted doses','total doses'])
```

```
[15]: <matplotlib.legend.Legend at 0x17c3924f130>
```



1.2 Generating heamap on Covid-19 India updated data

[17]:
```python
df = pd.read_csv('covid_new.csv')

# set the column as index
df_index = df.set_index('state')

# create heatmap using seaborn
plt.figure(figsize=(12, 6))
sns.heatmap(df_index.corr().round(2), annot=True, cmap='RdBu', fmt=',')
plt.title('COVID-19 Heatmap')
plt.xlabel('COLUMNS')
plt.ylabel('STATES')
plt.show()
```



	confirmed	active	passive	deaths	dose1	dose2	dose3	precaution_dose	total_doses	wasted_doses
confirmed	1.0	0.56	1.0	0.78	0.25	0.18	-0.09	0.16	0.14	-0.03
active	0.56	1.0	0.56	0.66	0.22	0.19	-0.15	0.02	0.12	0.03
passive	1.0	0.56	1.0	0.78	0.25	0.18	-0.09	0.16	0.15	-0.03
deaths	0.78	0.66	0.78	1.0	0.26	0.16	-0.2	0.01	0.08	-0.15
dose1	0.25	0.22	0.25	0.26	1.0	0.93	0.31	0.31	0.88	0.37
dose2	0.18	0.19	0.18	0.16	0.93	1.0	0.45	0.45	0.95	0.51
dose3	-0.09	-0.15	-0.09	-0.2	0.31	0.45	1.0	0.73	0.69	0.42
precaution_dose	0.16	0.02	0.16	0.01	0.31	0.45	0.73	1.0	0.63	0.38
total_doses	0.14	0.12	0.15	0.08	0.88	0.95	0.69	0.63	1.0	0.59
wasted_doses	-0.03	0.03	-0.03	-0.15	0.37	0.51	0.42	0.38	0.59	1.0

```
[ ]:
```

Appendix II

Analytical Methodology Supplementary Code

II.1 Linear Regression

linear_regression

June 19, 2023

1 Linear Regression on Covid-19 India dataset

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
# Read the CSV data file
df = pd.read_csv('covid_new.csv')
```

[3]:
```
# Implement simple linear regression using regression formula

def mean(values):
    return sum(values) / len(values)

def variance(values, mean):
    return sum([(x - mean) ** 2 for x in values])

def covariance(x, y, x_mean, y_mean):
    return sum([(x[i] - x_mean) * (y[i] - y_mean) for i in range(len(x))])

def coefficients(x, y):
    x_mean = mean(x)
    y_mean = mean(y)
    b1 = covariance(x, y, x_mean, y_mean) / variance(x, x_mean)
    b0 = y_mean - b1 * x_mean
    return b0, b1

def simple_linear_regression(x, y):
    b0, b1 = coefficients(x, y)
    predictions = [b0 + b1 * x[i] for i in range(len(x))]
    return b0, b1, predictions
```

1.1 Linear Regression: Confirmed Vs Dose1

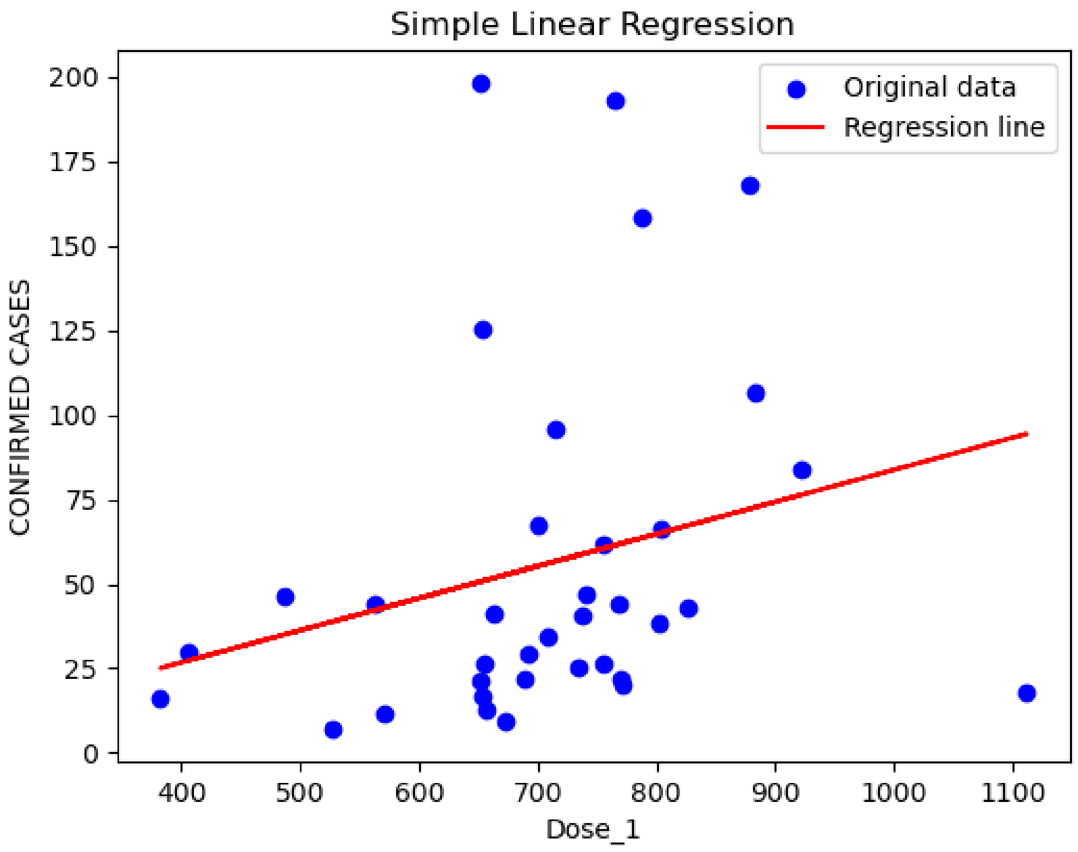
[4]:
```
# Copy the values of the "confirmed" column into a variable
x_dose1 = df['dose1'].tolist()
y_confirmed = df['confirmed'].tolist()
```

```
# Convert the data to arrays
x_dose1 = np.array(x_dose1)
y_confirmed = np.array(y_confirmed)
```

[5]:
```
# Perform simple linear regression
intercept, slope, predictions = simple_linear_regression(x_dose1, y_confirmed)

# Print the coefficients
print("Intercept:", intercept)
print("Slope:", slope)
print("Predictions:", predictions)
```

```
Intercept: -11.295686815529749
Slope: 0.09501872297595909
Predictions: [58.54073728224867, 61.73060429398646, 42.20056000950762,
50.655785471449015, 38.77017563899728, 76.25965863078578, 51.75401603342522,
94.3257039744132, 72.59296818831643, 72.14882876580928, 62.0263985763072,
64.87649012244657, 67.16454801259503, 55.95731781554201, 42.923135760535374,
60.53168977422742, 61.354951136363795, 56.61583416697751, 63.587646339064705,
51.17886921404944, 55.262979633589524, 34.94167182699262, 27.337448842439066,
50.61349742679606, 25.064985415603545, 54.49181151667305, 50.765022064486935,
60.42955517510276, 50.773804207965476, 65.08014543323235, 59.09668712695424,
61.82832419021544, 50.96410800089977, 52.638831579013264, 58.79024765004979,
54.16093940993807]
```

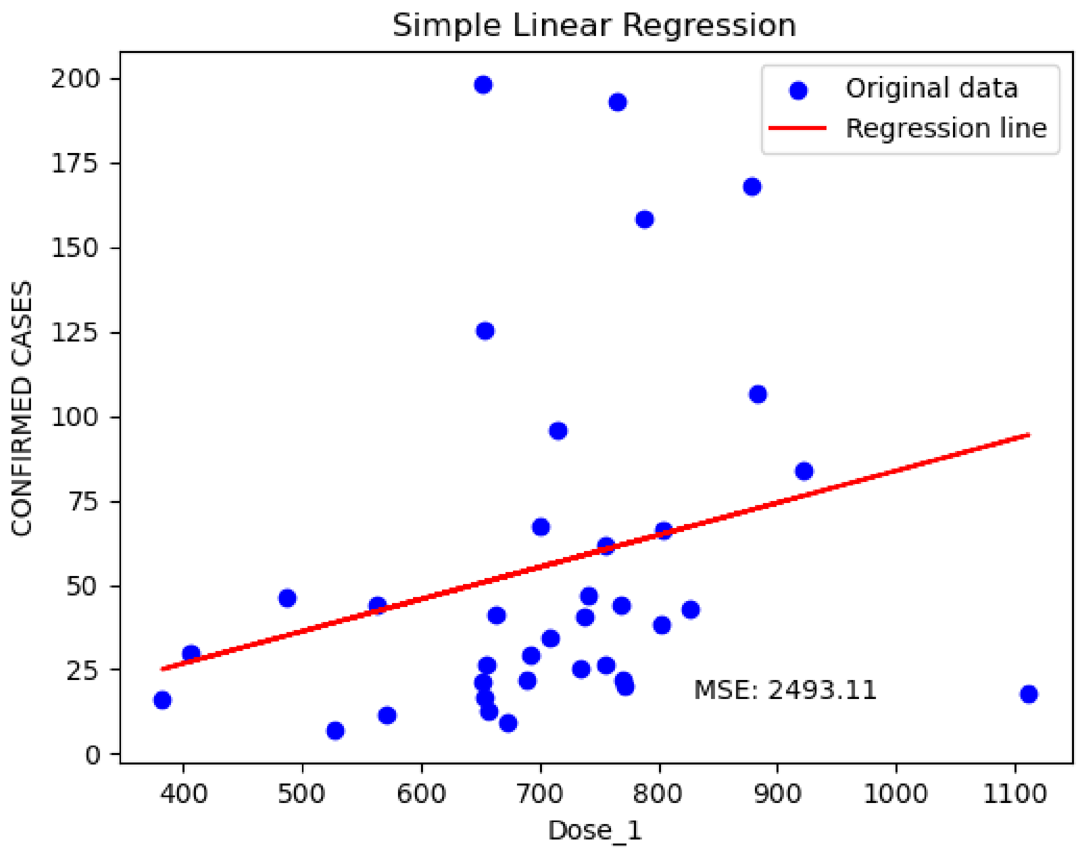
[6]:
```
# Plot the original data points
plt.scatter(x_dose1, y_confirmed, color='blue', label='Original data')

# Plot the regression line
plt.plot(x_dose1, predictions, color='red', label='Regression line')

# Add labels and title
plt.xlabel('Dose_1')
plt.ylabel('CONFIRMED CASES')
plt.title('Simple Linear Regression')

# Add a legend
plt.legend()

# Display the plot
plt.show()
```



[7]:
```
# Calculate the mean squared error (MSE)
mse = mean((y_confirmed - predictions) ** 2)

# Perform simple linear regression
intercept, slope, predictions = simple_linear_regression(x_dose1, y_confirmed)

# Calculate the mean squared error (MSE)
mse = mean((y_confirmed - predictions) ** 2)

# Print the coefficients and MSE
print("Intercept:", intercept)
print("Slope:", slope)
print("Predictions:", predictions)
print("Mean Squared Error:", mse)


# Perform simple linear regression
intercept, slope, predictions = simple_linear_regression(x_dose1, y_confirmed)

# Calculate the mean squared error (MSE)
```

```python
mse = mean((y_confirmed - predictions) ** 2)

# Print the coefficients and MSE
print("Intercept:", intercept)
print("Slope:", slope)
print("Predictions:", predictions)
print("Mean Squared Error:", mse)

# Plot the original data points
plt.scatter(x_dose1, y_confirmed, color='blue', label='Original data')

# Plot the regression line
plt.plot(x_dose1, predictions, color='red', label='Regression line')

# Add labels and title
plt.xlabel('Dose_1')
plt.ylabel('CONFIRMED CASES')
plt.title('Simple Linear Regression')

# Add MSE text
mse_text = f"MSE: {mse:.2f}"
plt.text(0.7, 0.1, mse_text, ha='center', va='center', transform=plt.gca().
 ↪transAxes)

# Add a legend
plt.legend()

# Display the plot
plt.show()
```

```
Intercept: -11.295686815529749
Slope: 0.09501872297595909
Predictions: [58.54073728224867, 61.73060429398646, 42.20056000950762,
50.655785471449015, 38.77017563899728, 76.25965863078578, 51.75401603342522,
94.3257039744132, 72.59296818831643, 72.14882876580928, 62.0263985763072,
64.87649012244657, 67.16454801259503, 55.95731781554201, 42.923135760535374,
60.53168977422742, 61.354951136363795, 56.61583416697751, 63.587646339064705,
51.17886921404944, 55.262979633589524, 34.94167182699262, 27.337448842439066,
50.61349742679606, 25.064985415603545, 54.49181151667305, 50.765022064486935,
60.42955517510276, 50.773804207965476, 65.08014543323235, 59.09668712695424,
61.82832419021544, 50.96410800089977, 52.638831579013264, 58.79024765004979,
54.16093940993807]
Mean Squared Error: 2493.105500451706
Intercept: -11.295686815529749
Slope: 0.09501872297595909
Predictions: [58.54073728224867, 61.73060429398646, 42.20056000950762,
50.655785471449015, 38.77017563899728, 76.25965863078578, 51.75401603342522,
94.3257039744132, 72.59296818831643, 72.14882876580928, 62.0263985763072,
64.87649012244657, 67.16454801259503, 55.95731781554201, 42.923135760535374,
60.53168977422742, 61.354951136363795, 56.61583416697751, 63.587646339064705,
51.17886921404944, 55.262979633589524, 34.94167182699262, 27.337448842439066,
50.61349742679606, 25.064985415603545, 54.49181151667305, 50.765022064486935,
60.42955517510276, 50.773804207965476, 65.08014543323235, 59.09668712695424,
61.82832419021544, 50.96410800089977, 52.638831579013264, 58.79024765004979,
54.16093940993807]
Mean Squared Error: 2493.105500451706
```



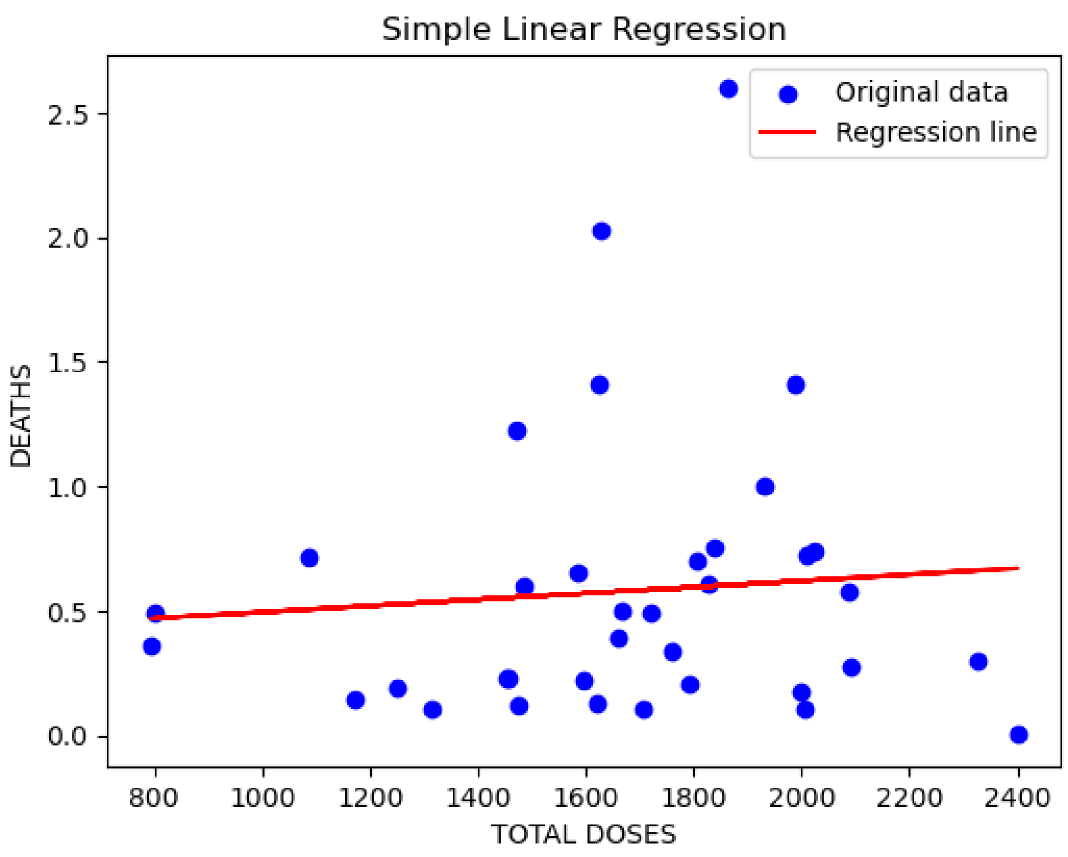
1.2 Linear Regression: Total Doses Vs Deaths

[8]:
```
# Copy the values of the "confirmed" column into a variable
x_total_doses = df['total_doses'].tolist()
y_deaths = df['deaths'].tolist()

# Convert the data to arrays
x_total_doses = np.array(x_total_doses)
y_deaths = np.array(y_deaths)

# Perform simple linear regression
```

```
intercept, slope, predictions = simple_linear_regression(x_total_doses,␣
 ↪y_deaths)

# Print the coefficients
print("Intercept:", intercept)
print("Slope:", slope)
print("Predictions:", predictions)
```

```
Intercept: 0.3695947491543371
Slope: 0.00012543691878402416
Predictions: [0.6613031003689227, 0.631831310582091, 0.5265049837663706,
0.55196671431369, 0.5346553992175973, 0.6117750449670017, 0.5854067052911096,
0.6707871488478432, 0.6188778305898298, 0.6031567876906601, 0.6204285358431499,
0.5777891914044789, 0.6315309289251064, 0.590512244905866, 0.5164054690104828,
0.5988499440741202, 0.5736615763223529, 0.6002711374649127, 0.6214900090153426,
0.573100466001713, 0.5540362415900942, 0.5057630419458214, 0.4699442688657086,
0.5558630203307815, 0.46914881603902725, 0.5944770837598459, 0.573550280660045,
0.5685621488723381, 0.5545093492570596, 0.6236439118114776, 0.5786126626993984,
0.6213038086969827, 0.5524808826270873, 0.5836089904099415, 0.5963872499536547,
0.569845592080088]
```

[9]:
```
# Plot the original data points
plt.scatter(x_total_doses, y_deaths, color='blue', label='Original data')

# Plot the regression line
plt.plot(x_total_doses, predictions, color='red', label='Regression line')

# Add labels and title
plt.xlabel('TOTAL DOSES')
plt.ylabel('DEATHS')
plt.title('Simple Linear Regression')

# Add a legend
plt.legend()

# Display the plot
plt.show()
```



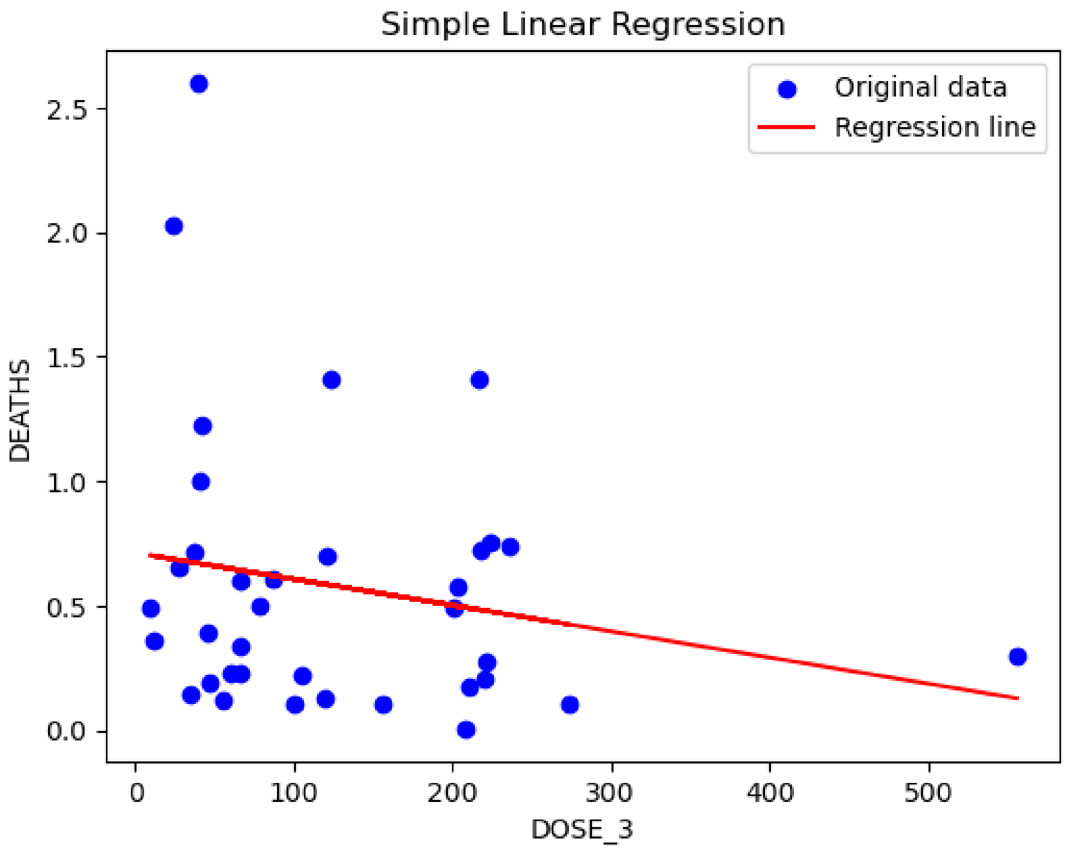
1.3 Linear Regression: Dose3 Vs Deaths

[10]:
```
# Copy the values of the "column" into a variable
x_dose3 = df['dose3'].tolist()
y_deaths = df['deaths'].tolist()

# Convert the data to arrays
x_dose3 = np.array(x_dose3)
y_deaths = np.array(y_deaths)

# Perform simple linear regression
intercept, slope, predictions = simple_linear_regression(x_dose3, y_deaths)

# Print the coefficients
print("Intercept:", intercept)
print("Slope:", slope)
print("Predictions:", predictions)
```

```
Intercept: 0.7127586872583541
Slope: -0.0010510400772896491
```

```
Predictions: [0.12852725613042815, 0.48015896630978805, 0.6629656794149101,
0.6494684881089552, 0.6073256036108702, 0.669653831045822, 0.5016573224190506,
0.4937751504664485, 0.5834753781081956, 0.6713111577058718, 0.4913971201599928,
0.6645850845174401, 0.49939234051185216, 0.6431623455285796, 0.6759675678532135,
0.6217852344912522, 0.6873357883228886, 0.4774888927462086, 0.4834104183828559,
0.5869674979687854, 0.6690720440390971, 0.6729312181213795, 0.7029082431432915,
0.6434085917638638, 0.7002477742088609, 0.48150992224969236, 0.484553213943581,
0.6843641447487999, 0.6538160797584052, 0.4651042787789453, 0.6298438545265097,
0.42519889194198485, 0.6424534213507777, 0.5492286470548722, 0.5851839911096488,
0.6024064376588767]
```

[11]:
```
# Plot the original data points
plt.scatter(x_dose3, y_deaths, color='blue', label='Original data')

# Plot the regression line
plt.plot(x_dose3, predictions, color='red', label='Regression line')

# Add labels and title
plt.xlabel('DOSE_3')
plt.ylabel('DEATHS')
plt.title('Simple Linear Regression')

# Add a legend
plt.legend()

# Display the plot
plt.show()
```


Simple Linear Regression
Original data
Regression line
DEATHS
DOSE_3
0.0
0.5
1.0
1.5
2.0
2.5
0
100
200
300
400
500

```
[ ]:
```

II.2 Multi Linear Regression

Multi_Linear_Regression

June 19, 2023

1 Multi Linear Regression on Covid-19 dataset

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from mpl_toolkits.mplot3d import Axes3D
```

[2]:
```
# Read the dataset
data = pd.read_csv('covid_new.csv')
```

[3]:
```
# Select the features and target variable
X = data[['dose1', 'dose2', 'dose3','precaution_dose']]
y = data['deaths']
```

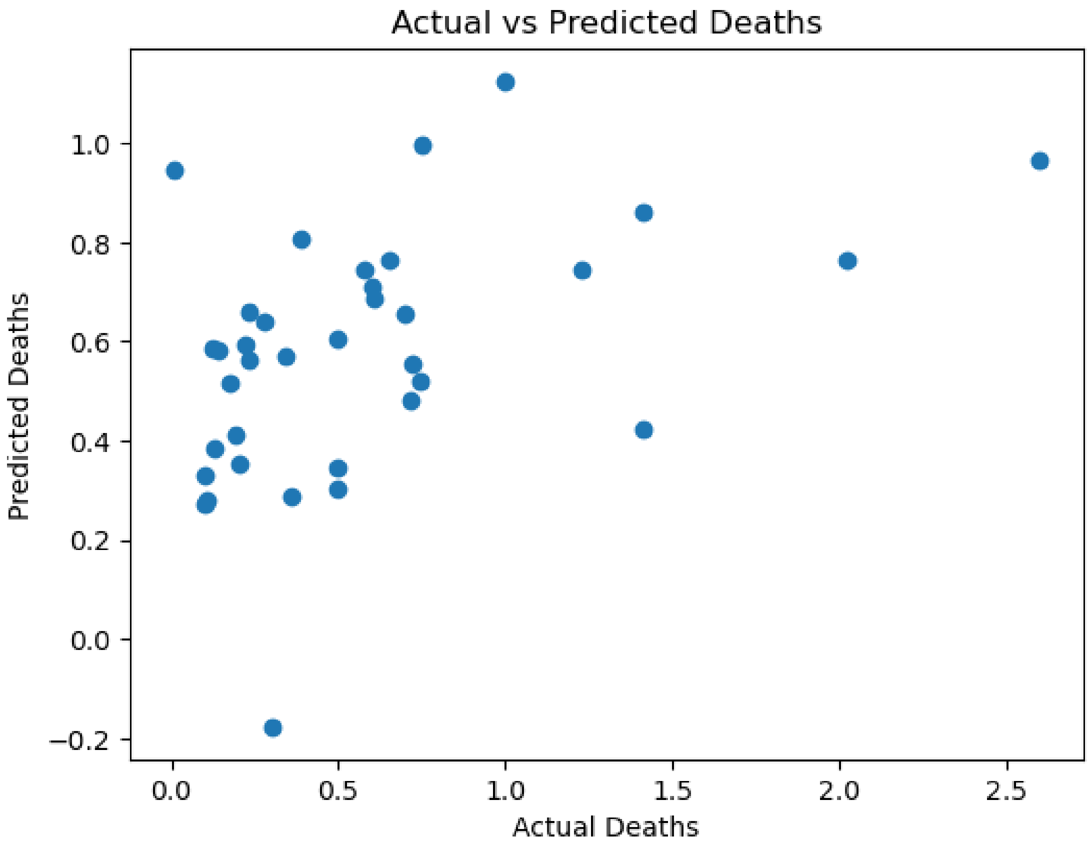
[4]:
```
# Create and fit the linear regression model
model = LinearRegression()
model.fit(X, y)
```

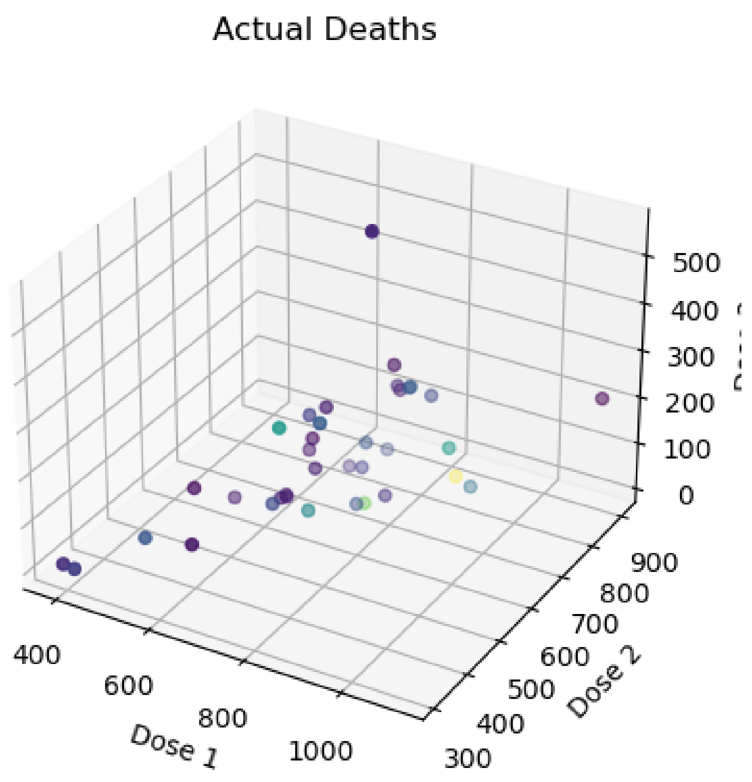
[4]:
```
LinearRegression()
```

[5]:
```
# Get the coefficients and intercept of the model
coefficients = model.coef_
intercept = model.intercept_

# Predict the target variable
y_pred = model.predict(X)

# Print the coefficients and intercept
print('Coefficients:', coefficients)
print('Intercept:', intercept)
```

```
Coefficients: [ 0.00272403 -0.00164311 -0.00234812  0.00481871]
Intercept: -0.2699504012682905
```

[6]:
```
# Plot the actual and predicted values
plt.scatter(y, y_pred)
plt.xlabel('Actual Deaths')
plt.ylabel('Predicted Deaths')
plt.title('Actual vs Predicted Deaths')
plt.show()

# Plot the actual and predicted values in 3D space
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X['dose1'], X['dose2'],X['dose3'], c=y, marker='o')
ax.set_xlabel('Dose 1')
ax.set_ylabel('Dose 2')
ax.set_zlabel('Dose 3')
ax.set_title('Actual Deaths')
plt.show()
```



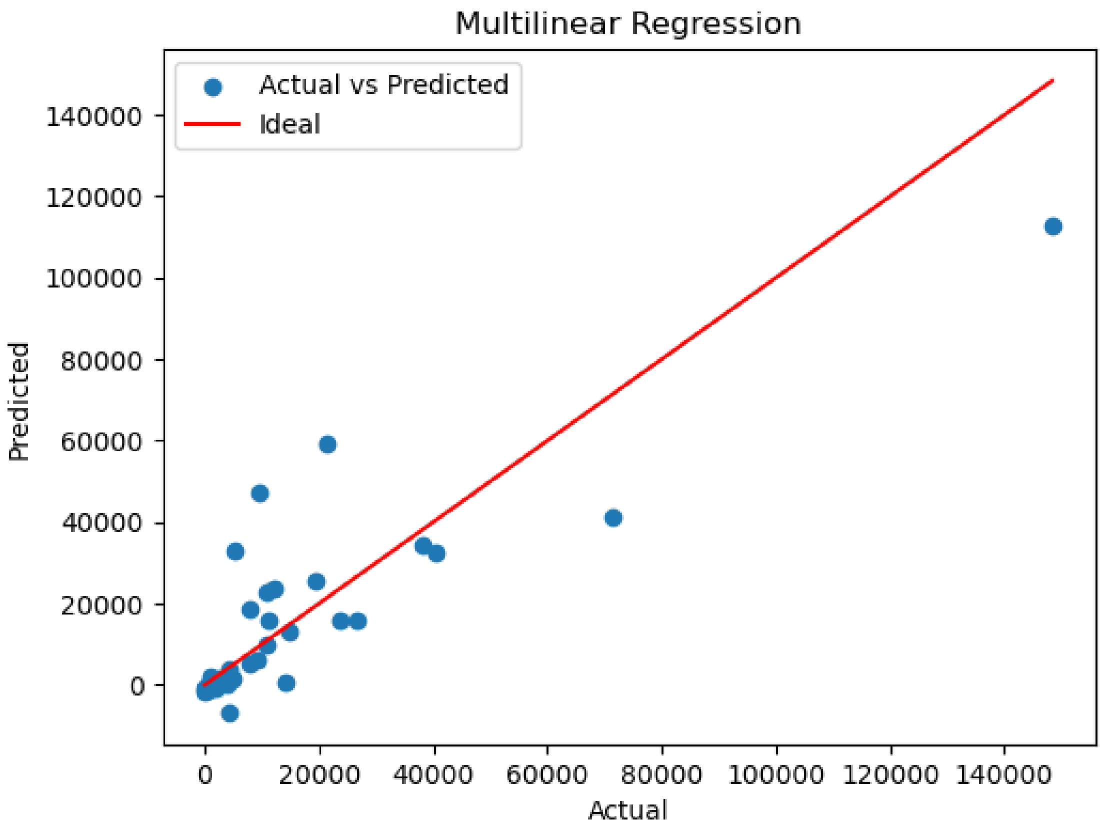


[7]:
```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Read the dataset
data = pd.read_csv('covid.csv')

# Select the features and target variable
X = data[['dose1', 'dose2', 'dose3','precaution_dose']]
y = data['deaths']

# Create and fit the linear regression model
model = LinearRegression()
model.fit(X, y)

# Get the coefficients and intercept of the model
coefficients = model.coef_
intercept = model.intercept_

# Print the coefficients and intercept
print('Coefficients:', coefficients)
print('Intercept:', intercept)
```

```
Coefficients: [ 0.00565748 -0.00547254 -0.00405594  0.01066642]
Intercept: -1359.574937039155
```

[10]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression

# Read the dataset
data = pd.read_csv('covid.csv')

# Select the features and target variable
x = data[['dose1', 'dose2', 'dose3', 'precaution_dose']]
y = data['deaths']

# Create and fit the linear regression model
reg_model = LinearRegression()
reg_model.fit(x, y)

# Generate predicted values
y_pred = reg_model.predict(x)
print("y_pred:", y_pred)

# Create scatter plot of the actual values
plt.scatter(y, y_pred, label='Actual vs Predicted')

# Add a diagonal line for reference
plt.plot([y.min(), y.max()], [y.min(), y.max()], c='red', label='Ideal')

plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Multilinear Regression')
plt.legend()
plt.show()
```

```
y_pred: [-1.73161131e+03  1.29667547e+04 -2.49019119e+02  1.86819795e+04
  2.37323293e+04  2.56541237e+02  4.28028453e+02 -7.96854679e+02
  1.58855961e+04  1.75851368e+02  1.58852975e+04  2.27352141e+04
  3.94969399e+03  1.41529088e+03  3.31203896e+04  3.26742019e+04
  4.14349597e+04 -8.43475533e+02 -1.33382631e+03  9.65191181e+03
  1.12811787e+05  1.51440863e+03  1.05124904e+03 -1.75821024e+02
 -4.98999162e+00  6.27298647e+03 -6.73071843e+02  2.56819546e+04
  4.73862802e+04 -1.13971800e+03  3.41690768e+04 -6.72285906e+03
  2.02223618e+03  1.59182019e+04  5.25351427e+03  5.92935120e+04]
```



[18]:
```
y.values
```

[18]:
```
array([   129,  14733,    296,   8035,  12302,   1181,  14146,      4,
        26521,   4013,  11043,  10714,   4213,   4785,   5331,  40307,
        71552,    231,     52,  10776, 148416,   2149,   1624,    726,
          782,   9205,   1975,  19289,   9653,    499,  38049,   4111,
          940,  23633,   7751,  21532], dtype=int64)
```

[20]:
```
y_pred
```

[20]:
```
array([-1.73161131e+03,  1.29667547e+04, -2.49019119e+02,  1.86819795e+04,
        2.37323293e+04,  2.56541237e+02,  4.28028453e+02, -7.96854679e+02,
        1.58855961e+04,  1.75851368e+02,  1.58852975e+04,  2.27352141e+04,
        3.94969399e+03,  1.41529088e+03,  3.31203896e+04,  3.26742019e+04,
        4.14349597e+04, -8.43475533e+02, -1.33382631e+03,  9.65191181e+03,
        1.12811787e+05,  1.51440863e+03,  1.05124904e+03, -1.75821024e+02,
       -4.98999162e+00,  6.27298647e+03, -6.73071843e+02,  2.56819546e+04,
        4.73862802e+04, -1.13971800e+03,  3.41690768e+04, -6.72285906e+03,
        2.02223618e+03,  1.59182019e+04,  5.25351427e+03,  5.92935120e+04])
```

[21]:
```
import numpy as np

# Actual values
Y_actual = np.array([129, 14733, 296, 8035, 12302, 1181, 14146, 4, 26521, 4013,␣
↪11043, 10714, 4213, 4785, 5331, 40307, 71552, 231, 52, 10776, 148416, 2149,␣
↪1624, 726, 782])

# Predicted values
Y_prediction = np.array([-1731.61, 12966.75, -249.02, 18681.98, 23732.33, 256.
↪54, 428.03, -796.85, 15885.60, 175.85, 15885.30, 22735.21, 3949.69, 1415.29,␣
↪33120.39, 32674.20, 41434.96, -843.48, -1333.83, 9651.91, 112811.79, 1514.
↪41, 1051.25, -175.82, -4.99])

# Calculate MAE, MSE, and RMSE
MAE = np.mean(np.abs(Y_actual - Y_prediction))
MSE = np.mean((Y_actual - Y_prediction)**2)
RMSE = np.sqrt(MSE)

print("Mean Absolute Error (MAE):", MAE)
print("Mean Squared Error (MSE):", MSE)
print("Root Mean Squared Error (RMSE):", RMSE)
```

```
Mean Absolute Error (MAE): 7371.421599999999
Mean Squared Error (MSE): 150379706.038576
Root Mean Squared Error (RMSE): 12262.940350445158
```

[ ]: