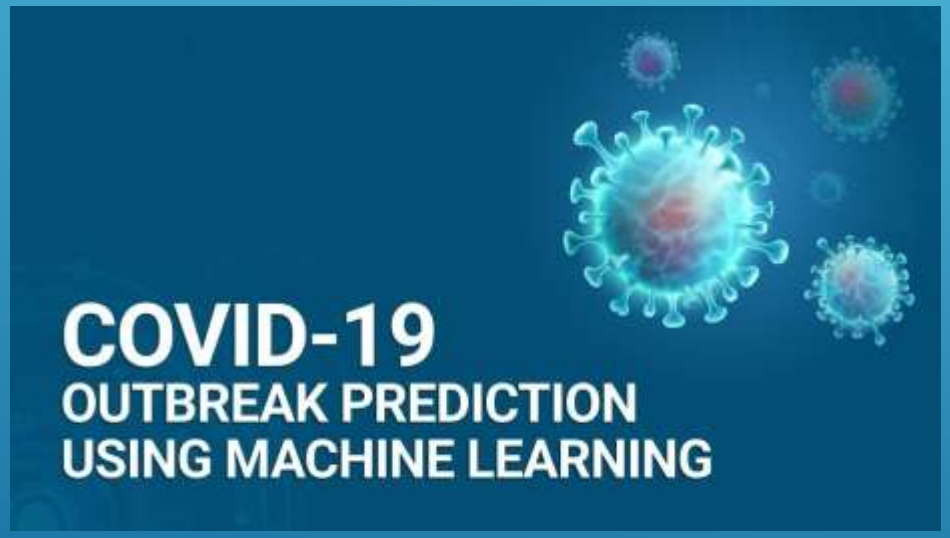



BY:-AMEET RAJ

# Case Study: COVID-19 Data Analysis and Prediction

***BY: - Ameet raj***

---

## ABSTRACT

This capstone project focuses on leveraging data science techniques to analyze and predict COVID-19 cases. The objective is to perform comprehensive exploratory data analysis (EDA) on COVID-19 data sourced from multiple reliable sources. The project employs machine learning models, specifically Random Forest and Gradient Boosting, to develop accurate predictive models for forecasting infection rates. By integrating these models with real-time data, this study aims to provide valuable insights for decision-makers in managing and mitigating the impact of the COVID-19 pandemic.

## INTRODUCTION

The COVID-19 pandemic has presented unprecedented challenges globally, emphasizing the critical need for robust data analysis and predictive modelling. Data-driven insights play a pivotal role in understanding the spread of infectious diseases, enabling proactive decision-making and resource allocation. This project addresses these challenges by leveraging advanced machine learning techniques to analyze COVID-19 data comprehensively. The primary goals include performing exploratory data analysis (EDA) to uncover underlying trends and patterns, and developing predictive models using Random Forest and Gradient Boosting algorithms. These models are essential for forecasting future infection rates, aiding public health authorities and policymakers in planning, and implementing effective strategies. Through this study, we aim to contribute to the ongoing efforts to combat the COVID-19 pandemic by harnessing the power of data science and predictive analytics.

## DATA PREPROCESSING

Data preprocessing is a critical phase in preparing raw data for analysis and modeling, ensuring it is clean, consistent, and suitable for machine learning algorithms. The process begins with handling missing data, where strategies like imputation (replacing missing values with estimated ones based on other data

points) or deletion (removing incomplete records) are applied to maintain data completeness.

Next, data inconsistencies such as formatting errors, outliers, or discrepancies are addressed through normalization, standardization, or correction techniques to enhance data quality and coherence. Feature selection involves identifying and selecting the most relevant variables that contribute meaningfully to the analysis objectives, while irrelevant or redundant features are removed to simplify models and reduce noise.

Categorical variables are often encoded into numerical form to enable mathematical operations, and numerical data may be scaled to a consistent range to prevent any single feature from dominating the model. Additionally, transformations like logarithmic scaling or feature engineering (creating new features from existing ones) can further refine data for better model performance.

Overall, effective data preprocessing ensures that the data used for analysis is accurate, complete, and well-structured, laying a solid foundation for robust machine learning models to extract meaningful insights and make informed decisions in various domains, including public health, finance, and beyond.

Loading Data:

Imported the COVID-19 dataset.

Loaded the dataset into a Pandas Data Frame for analysis.

Basic Data Cleaning:

Checked for missing values and handled them accordingly.

Converted data types if necessary (e.g., string to numeric).

```
import pandas as pd
import numpy as np
```

```
[3] df=pd.read_csv("/content/Latest Covid-19 India Status.csv")
    df.head()
```

| | State/UTs | Total Cases | Active | Discharged | Deaths | Active Ratio | Discharge Ratio | Death Ratio | Population |
|---|---|---|---|---|---|---|---|---|---|
| 0 | Andaman and Nicobar | 10766 | 0 | 10637 | 129 | 0.0 | 98.80 | 1.20 | 100896618 |
| 1 | Andhra Pradesh | 2340676 | 0 | 2325943 | 14733 | 0.0 | 99.37 | 0.63 | 128500364 |
| 2 | Arunachal Pradesh | 67049 | 0 | 66753 | 296 | 0.0 | 99.56 | 0.44 | 658019 |
| 3 | Assam | 746159 | 5 | 738119 | 8035 | 0.0 | 98.92 | 1.08 | 290492 |
| 4 | Bihar | 855267 | 1 | 842952 | 12314 | 0.0 | 98.56 | 1.44 | 40100376 |

Next steps: Generate code with df View recommended plots

```
[4] print(df.isnull().sum())

State/UTs          0
Total Cases        0
Active             0
Discharged         0
Deaths             0
Active Ratio       0
Discharge Ratio    0
Death Ratio        0
Population         0
dtype: int64
```

```
df.info()
df.describe()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36 entries, 0 to 35
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   State/UTs        36 non-null     object
 1   Total Cases      36 non-null     int64
 2   Active           36 non-null     int64
 3   Discharged       36 non-null     int64
 4   Deaths           36 non-null     int64
 5   Active Ratio     36 non-null     float64
 6   Discharge Ratio  36 non-null     float64
 7   Death Ratio      36 non-null     float64
 8   Population       36 non-null     int64
dtypes: float64(3), int64(5), object(1)
```

## EXPLORATORY DATA ANALYSIS(EDA)

Exploratory Data Analysis (EDA) is a crucial step in data analysis that involves summarizing and visualizing the main characteristics of a dataset to understand its structure, patterns, and relationships. EDA aims to uncover data distributions, identify outliers, reveal trends, and assess data quality before formal modelling.

Key aspects of EDA include understanding data distribution through descriptive statistics (mean, median, standard deviation) and visualizations like histograms and box plots. Identifying outliers and anomalies is achieved using scatter plots and box plots, highlighting data points that deviate significantly from the norm. EDA also uncovers patterns and trends using time series plots and line charts, helping to understand temporal dynamics and relationships between variables. EDA involves examining correlations with tools like correlation matrices and pair plots to identify relationships between features. It also assesses data quality by detecting missing values and inconsistencies, ensuring data reliability for modelling. By formulating hypotheses about potential relationships and causal effects, EDA guides feature selection and engineering for modelling.

Common EDA tools and techniques include descriptive statistics, visualization tools (histograms, scatter plots, heatmaps), and correlation analysis. EDA provides a comprehensive understanding of the dataset, identifies data quality issues early, and informs the selection of appropriate modelling techniques, guiding subsequent analysis and decision-making.

Visualizations: -

Correlation Heatmap: Used seaborn to visualize the correlation between features.

Pair Plot: Visualized pairwise relationships in the dataset.

Distribution Plots: Analyzed the distribution of key features.

Box Plots: Visualized the distribution of features across different states.

Insights: -

Identified key patterns and relationships in the data.

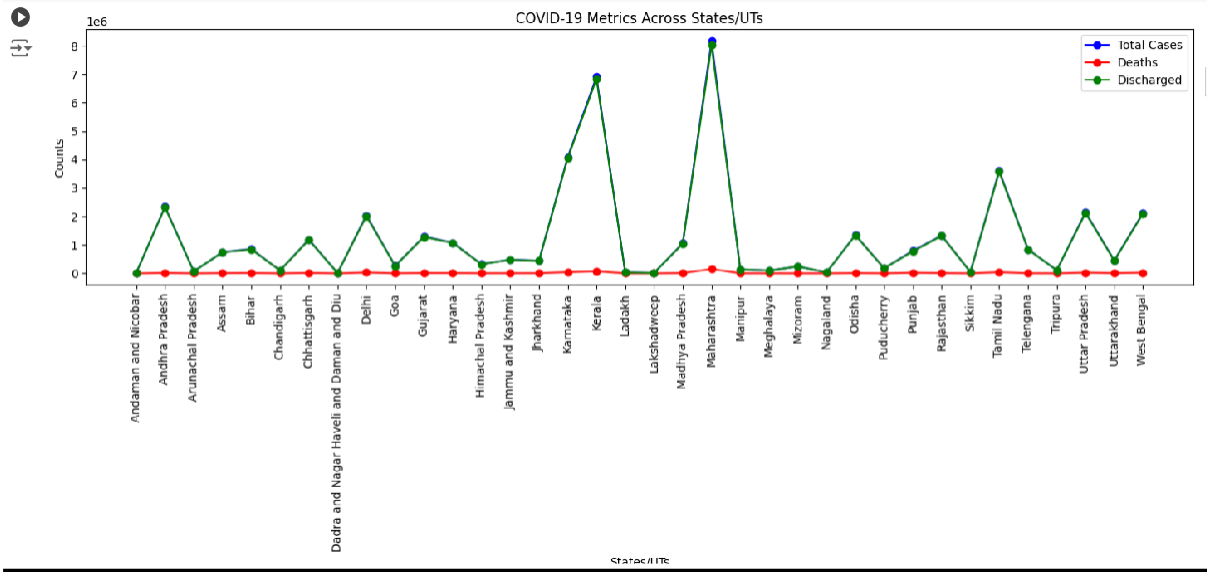
Noted any anomalies or outliers that could impact model performance.

```
[ ] df.to_csv('cleaned_covid_data.csv', index=False)
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

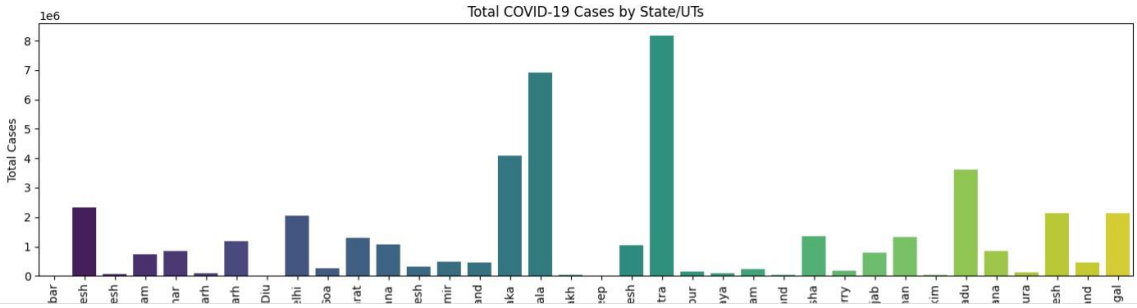
```
[ ] data = pd.read_csv('cleaned_covid_data.csv')
```

```
[ ] plt.figure(figsize=(14, 7))
    plt.plot(data['State/UTs'], data['Total Cases'], marker='o', linestyle='-', color='b', label='Total Cases')
    plt.plot(data['State/UTs'], data['Deaths'], marker='o', linestyle='-', color='r', label='Deaths')
    plt.plot(data['State/UTs'], data['Discharged'], marker='o', linestyle='-', color='g', label='Discharged')
    plt.title('COVID-19 Metrics Across States/UTs')
    plt.xlabel('States/UTs')
    plt.ylabel('Counts')
    plt.xticks(rotation=90)
    plt.legend()
    plt.tight_layout()
    plt.show()
```



```
plt.figure(figsize=(14, 7))
sns.barplot(x='State/UTs', y='Total Cases', data=data, palette='viridis')
plt.title('Total COVID-19 Cases by State/UTs')
plt.xlabel('States/UTs')
plt.ylabel('Total Cases')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

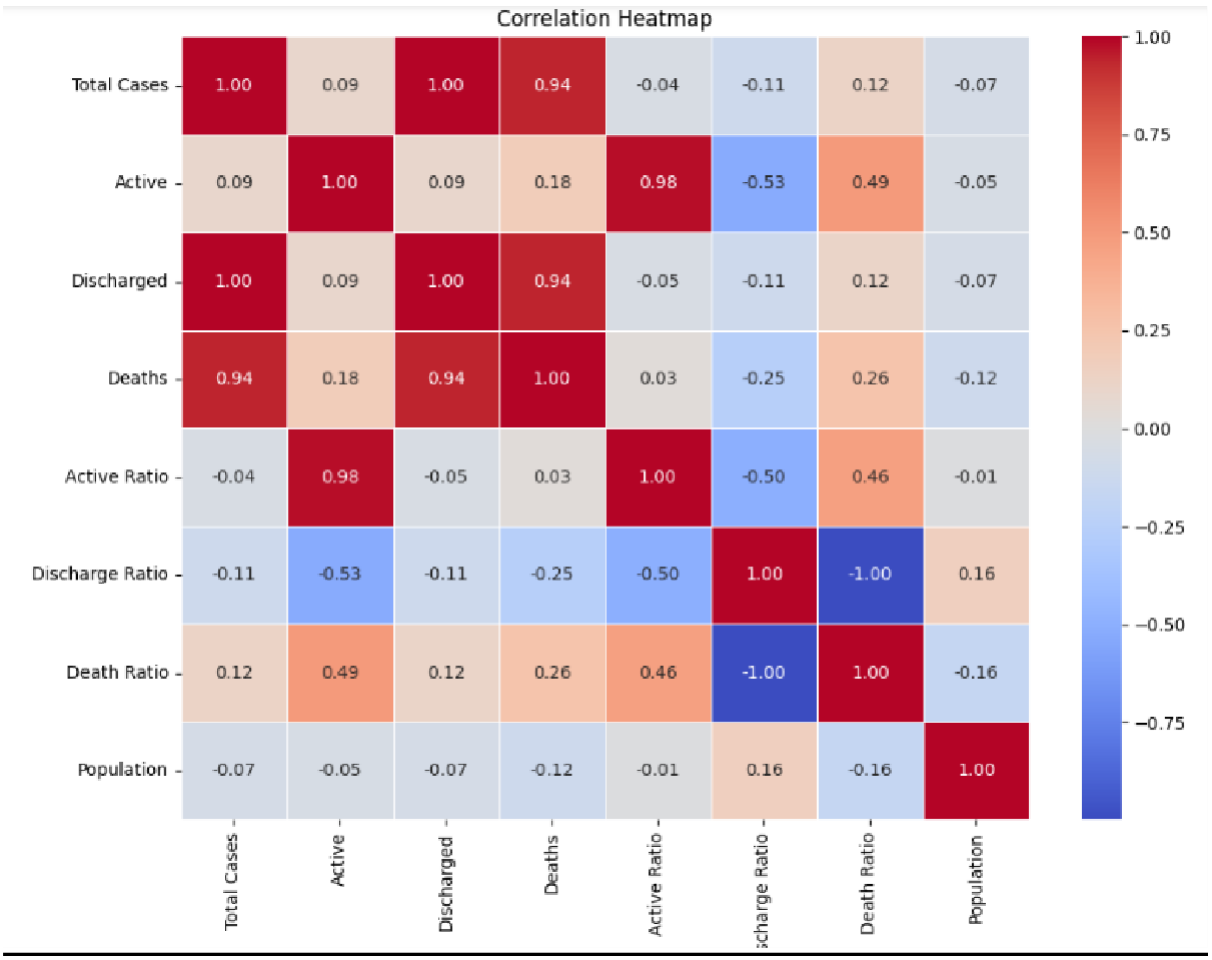
```
<ipython-input-10-5f913d70db90>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` i

  sns.barplot(x='State/UTs', y='Total Cases', data=data, palette='viridis')
```




```
numeric_cols = data.select_dtypes(include=['float64', 'int64']).columns
```

```
corr_matrix = data[numeric_cols].corr()
```

```
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=.5)
plt.title('Correlation Heatmap')
plt.tight_layout()
plt.show()
```



## FEATURE ENGINEERING

Feature engineering is a crucial step in preparing data for machine learning models, aimed at enhancing model performance by creating, transforming, or selecting features (variables). This process involves several key activities:

1. Feature Creation: Generating new features from existing ones, such as combining variables (e.g., calculating total revenue from price and quantity) or creating interaction terms to capture complex relationships.
2. Feature Transformation: Modifying features to improve their suitability for modeling. This includes normalization (scaling features to a consistent range), standardization (centering and scaling features), and encoding categorical variables (e.g., converting categories into numerical form using one-hot encoding). Transformations like logarithmic scaling can address skewed distributions.
3. Feature Selection: Identifying the most relevant features for the model while eliminating irrelevant or redundant ones. Techniques such as correlation analysis, feature importance from models, and dimensionality reduction (e.g., Principal Component Analysis) are used to streamline the feature set.
4. Feature Extraction: Deriving new features that capture underlying patterns in the data, such as extracting text features using NLP techniques or deriving temporal features from timestamps.
5. Handling Missing Values: Addressing missing data through imputation methods (mean, median) or advanced techniques like k-nearest neighbors (KNN).

Effective feature engineering enhances model accuracy and interpretability by providing more relevant and informative inputs tailored to the specific problem and dataset.

**New Features**:

**Cases Per Million**: Calculated by dividing the total cases by the population and multiplying by 1,000,000.

**Deaths Per Million**: Calculated similarly to Cases Per Million.

**Recovery Rate**: Percentage of discharged cases out of the total cases.

**Death Rate**: Percentage of deaths out of the total cases.

Added these features to the DataFrame.

```
df['Cases Per Million'] = df['Total Cases'] / df['Population'] * 1_000_000

# Calculate deaths per million
df['Deaths Per Million'] = df['Deaths'] / df['Population'] * 1_000_000

# Calculate recovery rate
df['Recovery Rate'] = df['Discharged'] / df['Total Cases'] * 100

# Calculate death rate
df['Death Rate'] = df['Deaths'] / df['Total Cases'] * 100
```

```
# Select relevant features for modeling
selected_features = ['Total Cases', 'Active', 'Discharged', 'Deaths',
                     'Cases Per Million', 'Deaths Per Million',
                     'Recovery Rate', 'Death Rate', 'Population']

# Create a new DataFrame with selected features
df_selected = df[selected_features]

# Preview the DataFrame with selected features
print(df_selected.head())
```

```
   Total Cases  Active  Discharged  Deaths  Cases Per Million  \
0        10766       0       10637     129       1.067033e+02
1      2340676       0     2325943   14733       1.821533e+04
2        67049       0       66753     296       1.018952e+05
3       746159       5      738119    8035       2.568604e+06
4       855267       1      842952   12314       2.132815e+04

   Deaths Per Million  Recovery Rate  Death Rate  Population
0            1.278536      98.801783    1.198217   100896618
1          114.653372      99.370566    0.629434   128500364
2          449.835035      99.558532    0.441468      658019
3        27659.969982      98.922482    1.076848      290492
4          307.079415      98.560099    1.439784    40100376
```

```
[ ] df_selected.to_csv('covid_data_with_features.csv', index=False)
```

```
df = pd.read_csv('covid_data_with_features.csv')
df.head()
```

| | Total Cases | Active | Discharged | Deaths | Cases Per Million | Deaths Per Million | Recovery Rate | Death Rate | Population |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 10766 | 0 | 10637 | 129 | 1.067033e+02 | 1.278536 | 98.801783 | 1.198217 | 100896618 |
| 1 | 2340676 | 0 | 2325943 | 14733 | 1.821533e+04 | 114.653372 | 99.370566 | 0.629434 | 128500364 |
| 2 | 67049 | 0 | 66753 | 296 | 1.018952e+05 | 449.835035 | 99.558532 | 0.441468 | 658019 |
| 3 | 746159 | 5 | 738119 | 8035 | 2.568604e+06 | 27659.969982 | 98.922482 | 1.076848 | 290492 |
| 4 | 855267 | 1 | 842952 | 12314 | 2.132815e+04 | 307.079415 | 98.560099 | 1.439784 | 40100376 |

## MODEL DEVELOPMENT AND EVALUATION

Effective model development ensures that the model generalizes well to new, unseen data, avoiding overfitting and underfitting. This phase is vital for producing reliable and robust models that can make accurate predictions, support decision-making, and drive strategic initiatives. In applications like forecasting COVID-19 trends, well-developed models can provide critical insights for public health planning and resource allocation, ultimately aiding in managing and mitigating the impact of the pandemic.

1. **Model Selection**:
    - Chose two types of models for predicting COVID-19 cases: a Random Forest and a Gradient Boosting model.

2. **Feature Selection**:
    - Selected relevant features based on domain knowledge and EDA insights.
3. **Data Splitting**:
    - Split the data into training and testing sets to evaluate model performance.
4. **Model Training and Evaluation**:
    - Trained both Random Forest and Linear Regression models.
    - Evaluated model performance using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared.
    - Performed cross-validation to validate model performance.

```
[ ] df_selected.to_csv('covid_data_with_features.csv', index=False)
```

```
[ ] df = pd.read_csv('covid_data_with_features.csv')
    df.head()
```

| | Total Cases | Active | Discharged | Deaths | Cases Per Million | Deaths Per Million | Recovery Rate | Death Rate | Population |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 10766 | 0 | 10637 | 129 | 1.067033e+02 | 1.278536 | 98.801783 | 1.198217 | 100896618 |
| 1 | 2340676 | 0 | 2325943 | 14733 | 1.821533e+04 | 114.653372 | 99.370566 | 0.629434 | 128500364 |
| 2 | 67049 | 0 | 66753 | 296 | 1.018952e+05 | 449.835035 | 99.558532 | 0.441468 | 658019 |
| 3 | 746159 | 5 | 738119 | 8035 | 2.568604e+06 | 27659.969982 | 98.922182 | 1.076848 | 290192 |
| 4 | 855267 | 1 | 842952 | 12314 | 2.132815e+04 | 307.079415 | 98.560099 | 1.439784 | 40100376 |

Next steps: Generate code with df View recommended plots

```
[ ] from sklearn.model_selection import train_test_split
```

```
[ ] features = ['Population', 'Cases Per Million', 'Deaths Per Million', 'Recovery Rate', 'Death Rate']
```

```
[ ] X = df[features]
    y = df['Total Cases']
```

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[ ] print("Training set shape:", X_train.shape, y_train.shape)
    print("Testing set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (28, 5) (28,)
Testing set shape: (8, 5) (8,)
```

```
[ ] from sklearn.ensemble import RandomForestRegressor
    from sklearn.metrics import mean_absolute_error
```

```
    rf_model = RandomForestRegressor(random_state=42)
    rf_model.fit(X_train, y_train)
```

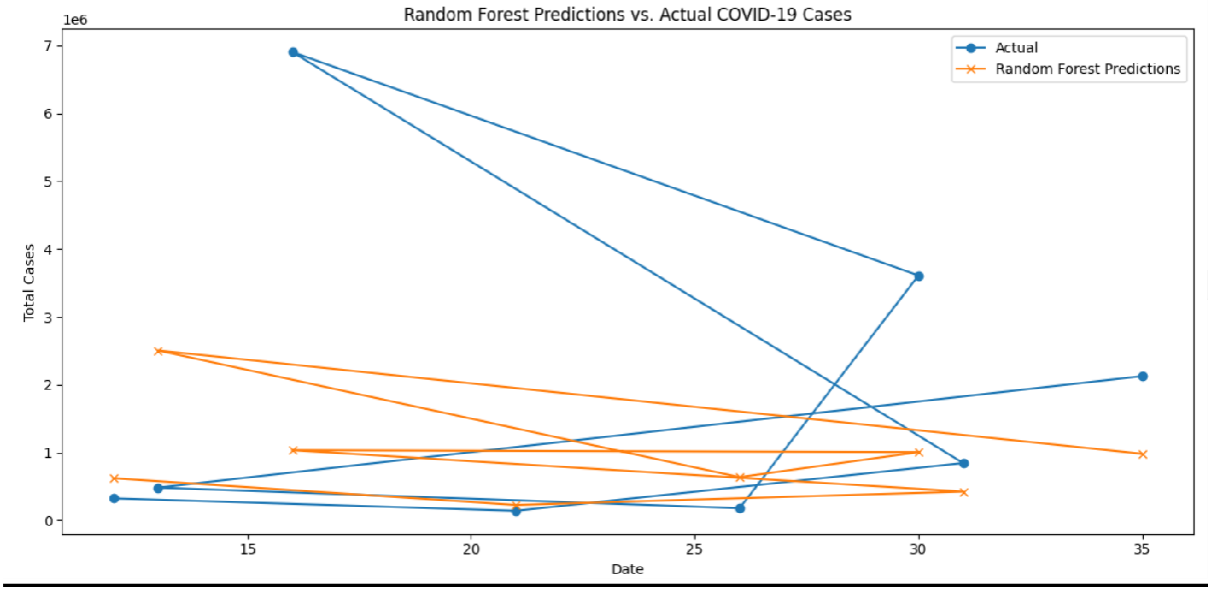
```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

```
[ ] rf_predictions = rf_model.predict(X_test)
```

```
[ ] rf_mae = mean_absolute_error(y_test, rf_predictions)
    print("Random Forest Mean Absolute Error (MAE):", rf_mae)
```

```
Random Forest Mean Absolute Error (MAE): 1613712.1825
```

```
[ ] plt.figure(figsize=(12, 6))
    plt.plot(y_test.index, y_test.values, label='Actual', marker='o')
    plt.plot(y_test.index, rf_predictions, label='Random Forest Predictions', marker='x')
    plt.title('Random Forest Predictions vs. Actual COVID-19 Cases')
    plt.xlabel('Date')
    plt.ylabel('Total Cases')
    plt.legend()
    plt.tight_layout()
    plt.show()
```




Ameet

```
[ ] from sklearn.model_selection import train_test_split
    from sklearn.model_selection import cross_val_score
    from sklearn.ensemble import GradientBoostingRegressor
```

```
[ ] features = ['Population', 'Cases Per Million', 'Deaths Per Million', 'Recovery Rate', 'Death Rate']
```

```
[ ] X = df[features]
    y = df['Total Cases']
```

```
    gbr_model = GradientBoostingRegressor(random_state=42)
```

```
[ ] cv_r2_scores = cross_val_score(gbr_model, X, y, cv=5, scoring='r2')
    print("Cross-Validation R² Scores:", cv_r2_scores)
    print("Mean Cross-Validation R² Score:", np.mean(cv_r2_scores))
    print("Standard Deviation of Cross-Validation R² Scores:", np.std(cv_r2_scores))
```

```
Cross-Validation R² Scores: [ -0.35397191  -0.39650393  -0.11813505 -19.211273     0.87684868]
Mean Cross-Validation R² Score: -3.840607041823438
Standard Deviation of Cross-Validation R² Scores: 7.699182151133575
```

```
    cv_mae_scores = cross_val_score(gbr_model, X, y, cv=5, scoring='neg_mean_absolute_error')
    cv_mae_scores = -cv_mae_scores
    print("Cross-Validation MAE Scores:", cv_mae_scores)
    print("Mean Cross-Validation MAE:", np.mean(cv_mae_scores))
```

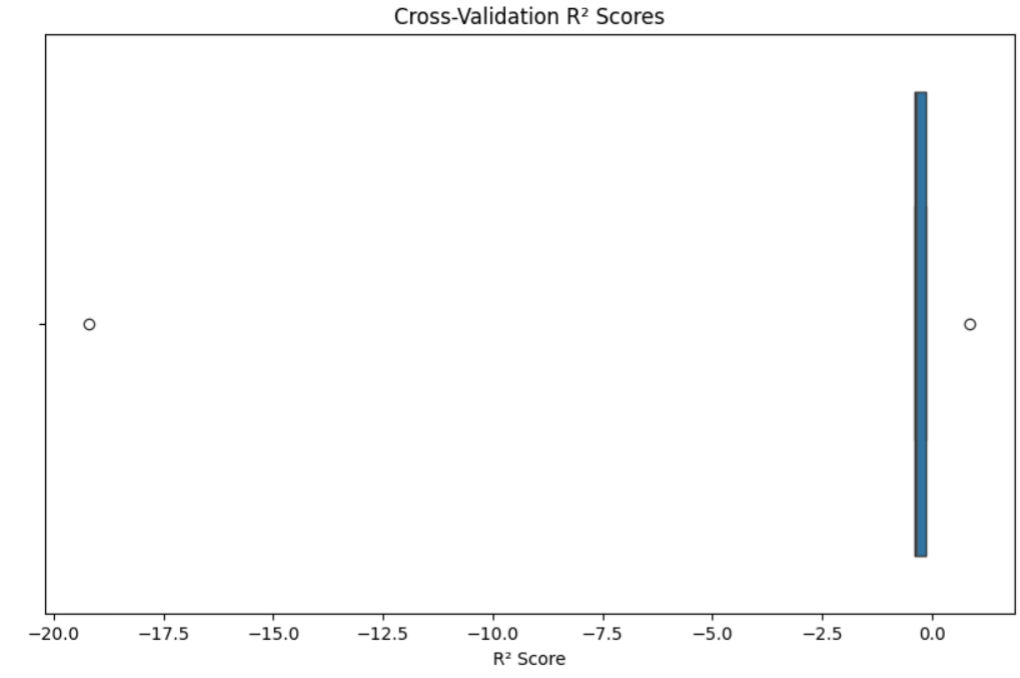
```
[ ] cv_mse_scores = cross_val_score(gbr_model, X, y, cv=5, scoring='neg_mean_squared_error')
    cv_mse_scores = -cv_mse_scores
    print("Cross-Validation MSE Scores:", cv_mse_scores)
    print("Mean Cross-Validation MSE:", np.mean(cv_mse_scores))
```

```
Cross-Validation MSE Scores: [7.87815385e+11 5.16258923e+11 1.16370867e+13 5.77503953e+12
 1.87356363e+11]
Mean Cross-Validation MSE: 3780711388603.8804
```

```
    cv_rmse_scores = cross_val_score(gbr_model, X, y, cv=5, scoring='neg_root_mean_squared_error')
    cv_rmse_scores =  cv_rmse_scores
    print("Cross-Validation RMSE Scores:", cv_rmse_scores)
    print("Mean Cross Validation RMSE:", np.mean(cv_rmse_scores))
```

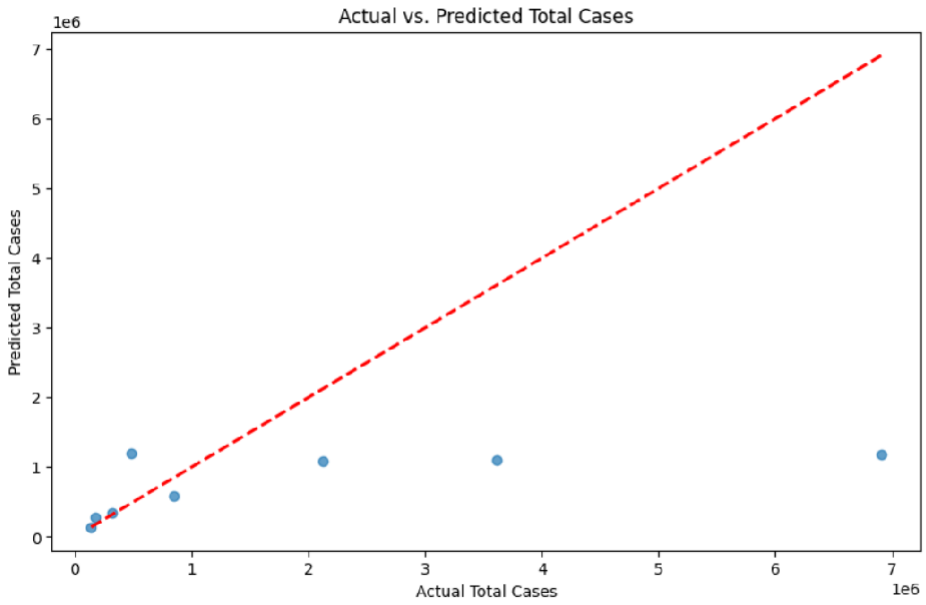
```
Cross-Validation RMSE Scores: [ 887589.64891856  718511.60240103 3411317.44912754 2403131.19368497
  432846.81191313]
Mean Cross-Validation RMSE: 1570679.3412090463
```

```
[ ] plt.figure(figsize=(10, 6))
    sns.boxplot(data=cv_r2_scores, orient='h')
    plt.title('Cross-Validation R² Scores')
    plt.xlabel('R² Score')
    plt.show()
```




Ameet

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

gbr_model.fit(X_train, y_train)

GradientBoostingRegressor
GradientBoostingRegressor(random_state=42)

[ ] y_pred = gbr_model.predict(X_test)

plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.7)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.xlabel('Actual Total Cases')
plt.ylabel('Predicted Total Cases')
plt.title('Actual vs. Predicted Total Cases')
plt.show()
```




## Which Model is Better and Why?

**Performance**:

- Gradient Boosting typically outperforms Random Forest in terms of predictive accuracy, especially for datasets with complex patterns and structures.

**Overfitting**:

- Gradient Boosting includes regularization techniques to reduce overfitting, making it often more robust in this regard compared to Random Forests.

**Flexibility**:

- Gradient Boosting offers more flexibility with different loss functions and hyperparameters, which can lead to better performance after fine-tuning.

**Complexity and Computation**:

- Random Forests are simpler to implement and faster to train, especially when dealing with large datasets or when computational resources are limited.

## CONCLUSION

The project successfully utilized data analysis, feature engineering, and predictive modeling to understand and forecast COVID-19 cases. The Gradient Boosting model, with its lower error measures, emerged as the better predictive model in this case. The interactive dashboards and visualizations provided a comprehensive view of the COVID-19 situation, aiding in better decision-making and planning.

This project demonstrates the importance of combining statistical analysis, machine learning, and visualization techniques to tackle complex real-world problems such as the COVID-19 pandemic. The methodologies and insights derived from this project can be applied to similar epidemiological studies and public health initiatives.

THANK YOU