Analysis of Covid-19 Data According to the Countries by Using K-Means Clustering and Regression Method

Abstract— The COVID-19 pandemic has affected the world as a whole and caused millions of deaths. This paper aims to provide a better understanding of how various Machine Learning models can be implemented in real-world situations. The study analyzes the trend or pattern of Covid-19 transmission in different parts of the world using datasets. The data studied was obtained for 3 years from 2020 to 2023. The paper also reviews studies that use machine and deep learning approaches in the estimation of COVID-19 spreading trend. Machine learning techniques have been used effectively in combating the COVID-19 epidemic. And we apply the K-Means clustering method to cluster the countries according to death rates. The COVID-19 crisis has put a spotlight on the power and potential of analytics and artificial intelligence. A systematic analysis has been conducted to highlight the latest developments in analyzing the COVID-19 data using machine learning.

I. INTRODUCTION

The COVID-19 pandemic has had a huge influence on global health and has presented various hurdles to healthcare systems around the world. Predicting and assessing COVID-19 data is critical for understanding the virus's transmission and making informed steps to reduce its impact. In this study, we will investigate the use of linear regression and decision tree approaches for predicting and analyzing COVID-19 data. In addition, we will investigate how countries are clustered based on death rates to acquire insight into how the pandemic affects different regions. Linear regression is a popular statistical method for modeling the connection between dependent variables. By applying linear regression to COVID-19 data, we can analyze trends and patterns in the virus's propagation and make predictions about future cases, hospitalizations, hotspot division or other important metrics. Another effective tool for analyzing and forecasting COVID-19 data is decision trees. To create predictions based on a collection of input features, decision trees employ a hierarchical structure of nodes and branches. We also compare the regression summary of both prediction methods and analyze which is better for this prediction. In addition to prediction and evaluation, we will investigate country grouping based on death rates. Clustering is a technique for grouping together comparable data elements based on their qualities. We can identify locations with comparable patterns of COVID-19 impact and acquire insights into the elements that contribute to greater or lower mortality rates by clustering countries based on death rates.

The purpose of this study report is to help policymakers, healthcare practitioners, and researchers better understand COVID-19 dynamics. We may acquire a complete picture of the pandemic's impact and guide evidence-based solutions for managing and minimizing its consequences by using linear regression, decision tree methods, and clustering techniques.

II. METHODS AND MATERIALS

A. Data Preprocessing

The data we collected from the world in data has 3 lakhs rows and 67 columns. Hence, we conducted a detailed processing of data. Finally, we compressed the data into 243 rows and 14 attributes which are total cases, total deaths total tests, total vaccinations, median age, aged_65_older, aged_70_older, gdp_per_capita, extreme poverty, population density, cardiovasc_death_rate, diabetes prevalence, life expectancy, population.

1. Finding and Removing Outliers

We plotted a boxplot with all 14 attributes to find the outliers. Outliers were prominent in total tests, total vaccinations, and population.

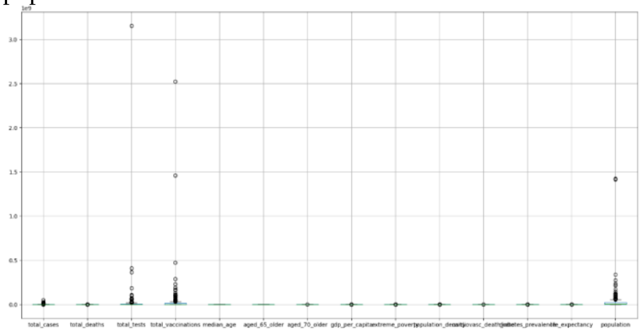
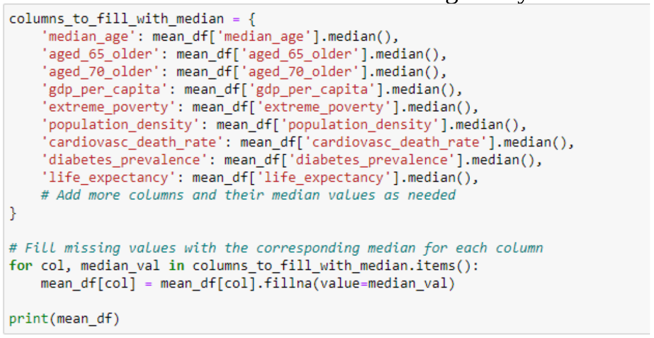


Fig.1.The boxplot for all the attributes which chose to analyze the outliers

Fig.1. shows the boxplot of all the attributes to visualize the outliers. Even though total cases, total tests and total vaccinations have null values, we substituted those null values with zeros logically. Using the IQR method, we removed all the outliers in the attributes.

2.Performing the imputation technique

Apart from total cases, total deaths, total vaccinations, and total tests which can be zero value, other attributes are replaced with median because from the histogram we concluded that attributes are negatively skewed.



```
columns_to_fill_with_median = {
    'median_age': mean_df['median_age'].median(),
    'aged_65_older': mean_df['aged_65_older'].median(),
    'aged_70_older': mean_df['aged_70_older'].median(),
    'gdp_per_capita': mean_df['gdp_per_capita'].median(),
    'extreme_poverty': mean_df['extreme_poverty'].median(),
    'population_density': mean_df['population_density'].median(),
    'cardiovasc_death_rate': mean_df['cardiovasc_death_rate'].median(),
    'diabetes_prevalence': mean_df['diabetes_prevalence'].median(),
    'life_expectancy': mean_df['life_expectancy'].median(),
    # Add more columns and their median values as needed
}

# Fill missing values with the corresponding median for each column
for col, median_val in columns_to_fill_with_median.items():
    mean_df[col] = mean_df[col].fillna(value=median_val)

print(mean_df)
```

Fig.2. Code for replacing the null value with the mean of each attribute.

Fig.2. shows the code for replacing null values with the median of the column. Hence, all our null values or missing values are replaced with the mean of each attribute.

3. Feature scaling the data

Feature scaling is an important stage in the data pre-processing process before building a machine learning model. It entails transforming the features in a dataset so that their values are all on the same scale. In our dataset, there are attributes with values zero. So, there is a huge scale difference in values, so we found that feature scaling is an unavoidable process.

4. Finding the correlation of attributes using Heat Map visualizations.

We found strong positive and negative correlations between variables by visualizing the correlation matrix as a hotspot. The darker the color, the stronger the association, and the lighter the color, the weaker the link.

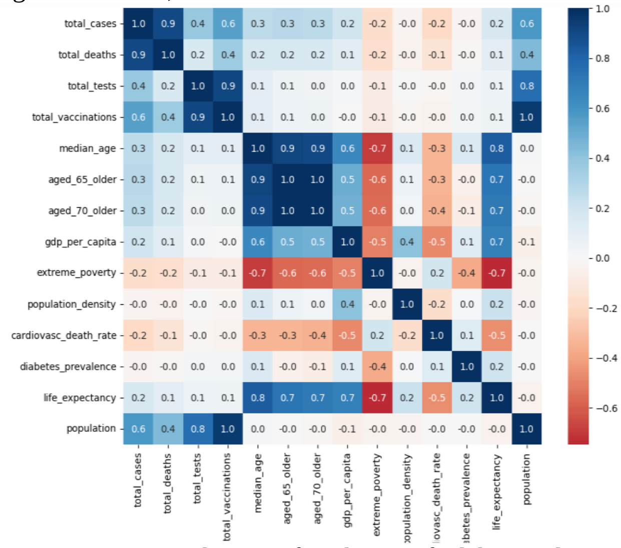


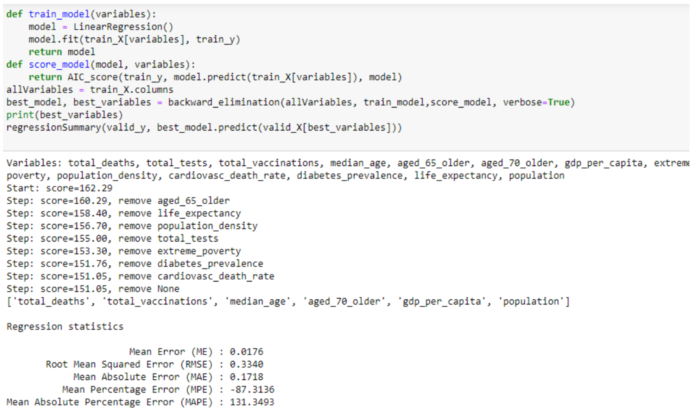
Fig.3. Heatmap visualization of attributes to find the correlation

This strategy aids in analyzing the correlations and patterns in our dataset. we found it. Fig.3. shows the heat map for analyzing the correlation between attributes.Strong correlation between total vaccinations, total tests, populations, total deaths. And we concluded that the linear regression can be done with these as predictors.

B. Linear Regression Model to predict total cases and Evaluation of Performance

1.Backward and Forward selection to find the suitable predictors.

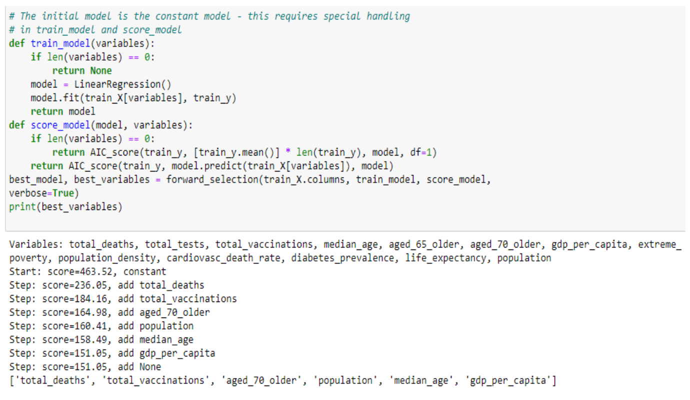
Backward and forward selection are two popular procedures used when analyzing predictors in a dataset. Backward selection begins with all predictor variables in the model. It then iteratively removes the least significant predictor variable at each step until none of the remaining predictors fulfill the stated requirement.Fig.4. shows the code for backward elimination and the result of significant predictors chosen.



```
def train_model(variables):
    model = LinearRegression()
    model.fit(train_X[variables], train_y)
    return model
def score_model(model, variables):
    return AIC_score(train_y, model.predict(train_X[variables]), model)
allVariables = train_X.columns
best_model, best_variables = backward_elimination(allVariables, train_model,score_model, verbose=True)
print(best_variables)
regressionSummary(valid_y, best_model.predict(valid_X[best_variables]))
```

```
Variables: total_deaths, total_tests, total_vaccinations, median_age, aged_65_older, aged_70_older, gdp_per_capita, extreme
poverty, population_density, cardiovasc_death_rate, diabetes_prevalence, life_expectancy, population
Start: score=162.29
Step: score=160.29, remove aged_65_older
Step: score=158.40, remove life_expectancy
Step: score=156.70, remove population_density
Step: score=155.00, remove total_tests
Step: score=153.30, remove extreme_poverty
Step: score=151.76, remove diabetes_prevalence
Step: score=151.05, remove cardiovasc_death_rate
Step: score=151.05, remove None
['total_deaths', 'total_vaccinations', 'median_age', 'aged_70_older', 'gdp_per_capita', 'population']

Regression statistics

                      Mean Error (ME) : 0.0176
       Root Mean Squared Error (RMSE) : 0.3340
            Mean Absolute Error (MAE) : 0.1718
          Mean Percentage Error (MPE) : -87.3136
Mean Absolute Percentage Error (MAPE) : 131.3493
```

Fig.4. Backward elimination to remove the least significant predictors.



```
# The initial model is the constant model - this requires special handling
# in train_model and score_model
def train_model(variables):
    if len(variables) == 0:
        return None
    model = LinearRegression()
    model.fit(train_X[variables], train_y)
    return model
def score_model(model, variables):
    if len(variables) == 0:
        return AIC_score(train_y, [train_y.mean()] * len(train_y), model, df=1)
    return AIC_score(train_y, model.predict(train_X[variables]), model)
best_model, best_variables = forward_selection(train_X.columns, train_model, score_model,
verbose=True)
print(best_variables)
```

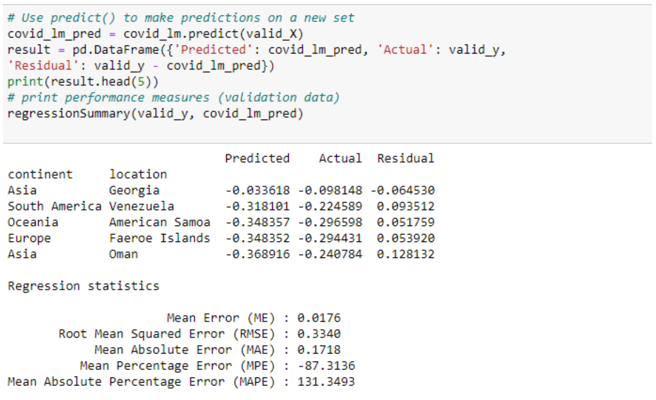
```
Variables: total_deaths, total_tests, total_vaccinations, median_age, aged_65_older, aged_70_older, gdp_per_capita, extreme_
poverty, population_density, cardiovasc_death_rate, diabetes_prevalence, life_expectancy, population
Start: score=463.52, constant
Step: score=236.05, add total_deaths
Step: score=184.16, add total_vaccinations
Step: score=164.98, add aged_70_older
Step: score=160.41, add population
Step: score=158.49, add median_age
Step: score=151.05, add gdp_per_capita
Step: score=151.05, add None
['total_deaths', 'total_vaccinations', 'aged_70_older', 'population', 'median_age', 'gdp_per_capita']
```

Fig.5. Forward selection to add the most significant predictors.

Forward selection begins with a null model that has no predictors. It then adds the most important predictor variable at each stage until no more predictors fulfill the required condition.Fig4. shows the code for forward selection and the result of significant predictors chosen. Finally, we got the predictors as total deaths, total vaccinations, median age, aged_70_older, gdp_per_capita, population. Using these predictors, we splitted the data into training and validation sets.

2. Finding coefficients for attributes and making predictions on the test set.

We fitted the linear regression model to the test set by finding coefficients. Assessed the model's performance by evaluating the regression summary. Once the model has been trained and validated, we use it to generate predictions on the test dataset.



```
# Use predict() to make predictions on a new set
covid_lm_pred = covid_lm.predict(valid_X)
result = pd.DataFrame({'Predicted': covid_lm_pred, 'Actual': valid_y,
'Residual': valid_y - covid_lm_pred})
print(result.head(5))
# print performance measures (validation data)
regressionSummary(valid_y, covid_lm_pred)
```

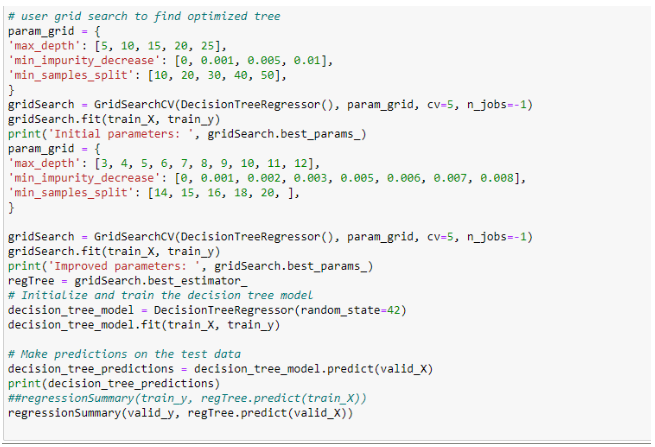
```
                              Predicted    Actual  Residual
continent     location
Asia          Georgia         -0.033618 -0.098148 -0.064530
South America Venezuela       -0.318101 -0.224589  0.093512
Oceania       American Samoa  -0.348357 -0.296598  0.051759
Europe        Faeroe Islands  -0.348352 -0.294431  0.053920
Asia          Oman            -0.368916 -0.240784  0.128132

Regression statistics

                      Mean Error (ME) : 0.0176
       Root Mean Squared Error (RMSE) : 0.3340
            Mean Absolute Error (MAE) : 0.1718
          Mean Percentage Error (MPE) : -87.3136
Mean Absolute Percentage Error (MAPE) : 131.3493
```

Fig.6. Code for making predictions on test set and regression statistics

Fig.6. shows the code for making predictions on test set and regression statistics.After we inputted the test cases properties into the trained model and made predictions of total cases based on the learnt coefficients.

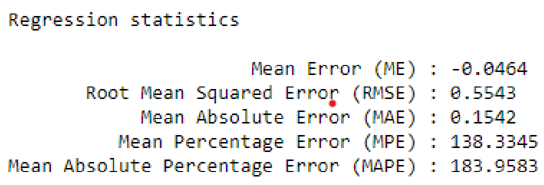
C. Decision Tree for making Prediction on total cases

Decision trees are well-known for their interpretability and ability to handle both numerical and categorical inputs. They can capture non-linear correlations and interactions between variables, making them useful for analyzing complicated datasets such as COVID-19 data.



```
# user grid search to find optimized tree
param_grid = {
'max_depth': [5, 10, 15, 20, 25],
'min_impurity_decrease': [0, 0.001, 0.005, 0.01],
'min_samples_split': [10, 20, 30, 40, 50],
}
gridSearch = GridSearchCV(DecisionTreeRegressor(), param_grid, cv=5, n_jobs=-1)
gridSearch.fit(train_X, train_y)
print('Initial parameters: ', gridSearch.best_params_)
param_grid = {
'max_depth': [3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
'min_impurity_decrease': [0, 0.001, 0.002, 0.003, 0.005, 0.006, 0.007, 0.008],
'min_samples_split': [14, 15, 16, 18, 20, ],
}

gridSearch = GridSearchCV(DecisionTreeRegressor(), param_grid, cv=5, n_jobs=-1)
gridSearch.fit(train_X, train_y)
print('Improved parameters: ', gridSearch.best_params_)
regTree = gridSearch.best_estimator_
# Initialize and train the decision tree model
decision_tree_model = DecisionTreeRegressor(random_state=42)
decision_tree_model.fit(train_X, train_y)

# Make predictions on the test data
decision_tree_predictions = decision_tree_model.predict(valid_X)
print(decision_tree_predictions)
##regressionSummary(train_y, regTree.predict(train_X))
regressionSummary(valid_y, regTree.predict(valid_X))
```

Fig.7. Code for Initializing and train the decision model and make predictions on test set



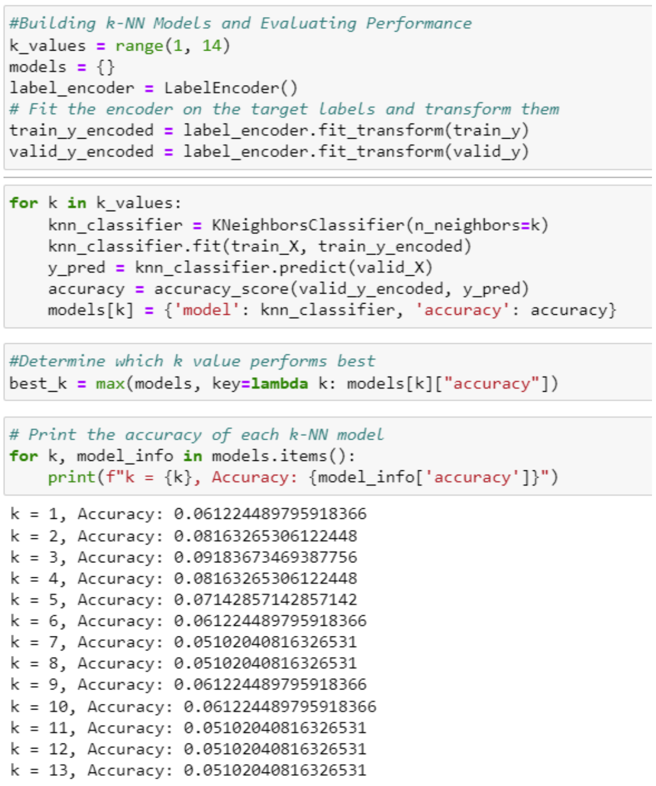
```
Regression statistics

                      Mean Error (ME) : -0.0464
       Root Mean Squared Error (RMSE) : 0.5543
            Mean Absolute Error (MAE) : 0.1542
          Mean Percentage Error (MPE) : 138.3345
Mean Absolute Percentage Error (MAPE) : 183.9583
```

Fig.8. Regression summary of Decision tree model prediction

In the Decision tree, we used grid search to find the optimum tree.The initial param_grid defines a set of hyperparameters to search over using the GridSearchCV function. The hyperparameters being tuned are max_depth, min_impurity_decrease, and min_samples_split. It considers different values for each of these hyperparameters.The GridSearchCV instance is created with a DecisionTreeRegressor() estimator, the defined param_grid, and other settings like cross-validation (cv=5) and parallelism (n_jobs=-1).Fig.7.shows the code for initializing and training the decision model and making predictions on the test set. At last, we made the predictions on the test set. Finally, we estimated the regression to evaluate and compare with the decision tree regression model. Fig.8.shows the regression statistics.

D. K Nearest Neighbors Regression to predict Total Cases

K-Nearest Neighbors (KNN) regression is used to predict a continuous numeric value based on input features. Here we use KNN Regression to predict the total number of COVID-19 cases. KNN regression works by finding the "k" nearest neighbors to a given data point in the feature space and using their target values to predict the target value for the new data point. In KNN regression model we start by defining a range of k values and a dictionary to store model accuracy information. The code transforms categorical target data into numerical values using Label Encoding. Through a loop, the code builds k-NN classifiers for different k values, trains them on the training data, and evaluates their accuracy on validation data. The best-performing model is obtained when using k=3, which has the highest accuracy. Figure 8 shows the KNN model with accuracy for each value of k.



```
#Building k-NN Models and Evaluating Performance
k_values = range(1, 14)
models = {}
label_encoder = LabelEncoder()
# Fit the encoder on the target labels and transform them
train_y_encoded = label_encoder.fit_transform(train_y)
valid_y_encoded = label_encoder.fit_transform(valid_y)
```

```
for k in k_values:
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(train_X, train_y_encoded)
    y_pred = knn_classifier.predict(valid_X)
    accuracy = accuracy_score(valid_y_encoded, y_pred)
    models[k] = {'model': knn_classifier, 'accuracy': accuracy}
```

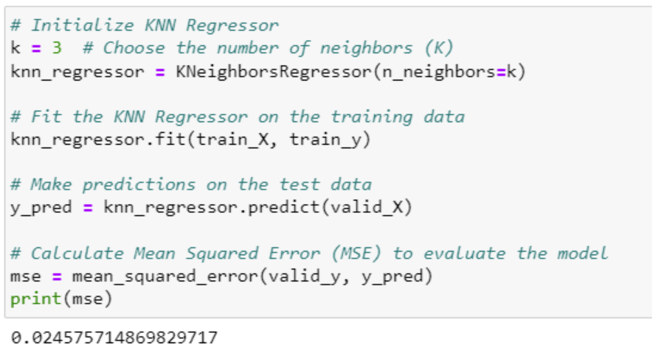
```
#Determine which k value performs best
best_k = max(models, key=lambda k: models[k]["accuracy"])
```

```
# Print the accuracy of each k-NN model
for k, model_info in models.items():
    print(f"k = {k}, Accuracy: {model_info['accuracy']}")
```

```
k = 1, Accuracy: 0.061224489795918366
k = 2, Accuracy: 0.08163265306122448
k = 3, Accuracy: 0.09183673469387756
k = 4, Accuracy: 0.08163265306122448
k = 5, Accuracy: 0.07142857142857142
k = 6, Accuracy: 0.061224489795918366
k = 7, Accuracy: 0.05102040816326531
k = 8, Accuracy: 0.05102040816326531
k = 9, Accuracy: 0.061224489795918366
k = 10, Accuracy: 0.061224489795918366
k = 11, Accuracy: 0.05102040816326531
k = 12, Accuracy: 0.05102040816326531
k = 13, Accuracy: 0.05102040816326531
```

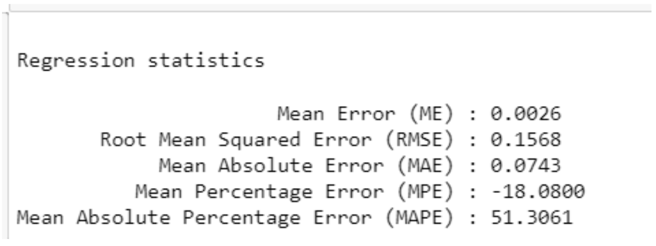
Fig 8: Code for evaluating and selecting the best value for k

Then the KNeighborsRegressor is initialized using k=3 and the regressor is then trained on the training data, predictions are made on the test data and these predictions are compared to the actual target values. Figure 9 shows the KNN Regression model. The values for the total case are predicted and Mean Squared Error (MSE), is calculated using the predicted and actual values.



```
# Initialize KNN Regressor
k = 3  # Choose the number of neighbors (K)
knn_regressor = KNeighborsRegressor(n_neighbors=k)

# Fit the KNN Regressor on the training data
knn_regressor.fit(train_X, train_y)

# Make predictions on the test data
y_pred = knn_regressor.predict(valid_X)

# Calculate Mean Squared Error (MSE) to evaluate the model
mse = mean_squared_error(valid_y, y_pred)
print(mse)

0.024575714869829717
```

Fig.9. K Nearest Neighbors Regression model



```
Regression statistics

                      Mean Error (ME) : 0.0026
       Root Mean Squared Error (RMSE) : 0.1568
            Mean Absolute Error (MAE) : 0.0743
          Mean Percentage Error (MPE) : -18.0800
Mean Absolute Percentage Error (MAPE) : 51.3061
```

Fig.10. Regression Statistics for KNN Regression model

MSE is computed to evaluate and compare with other regression models.

E. K-MEANS CLUSTERING

An unsupervised machine learning algorithm, K-Means clustering is used for partitioning a dataset into distinct groups or clusters. The primary goal of K-means is to classify data points based on their similarity to a certain number (K) of cluster centroids. It operates iteratively to assign each data point to the nearest centroid and then updates the centroids by calculating the mean of the points assigned to each cluster. This process continues until convergence, resulting in well-defined clusters.
Here we are using K-means clustering to cluster the location based on COVID-19 death rates to identify hot spots in the location based on the death rate. First we generate an Elbow plot to determine the optimal number of clusters (K) for KMeans clustering. The plot helps find the point at which adding more clusters does not significantly improve the model's fit.

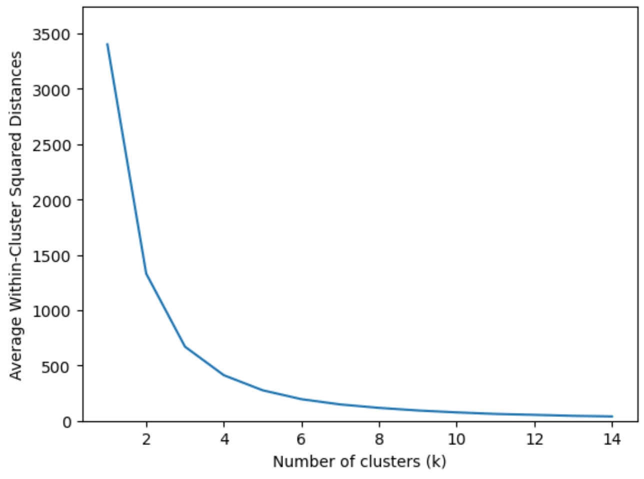
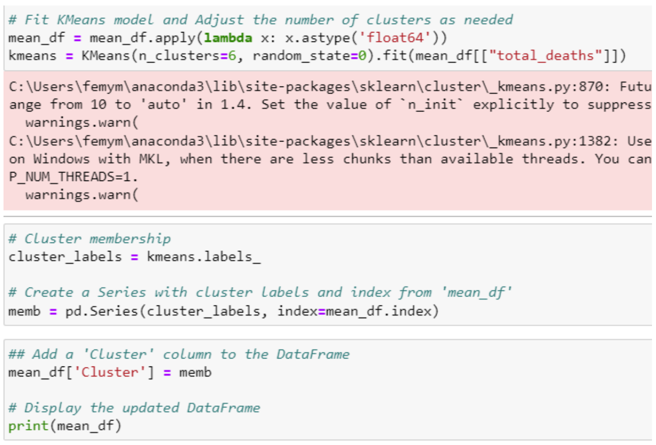


Fig.11. Elbow Plot generated

This Elbow plot assists in making an informed decision about the optimal number of clusters for the KMeans algorithm and k = 6 is selected for the model.

Then the data points are segmented based on the total deaths column, to get underlying patterns or similarities in the dataset. The number of clusters is set to 6. After fitting the KMeans model, the resulting cluster labels are assigned to each data point. The KMeans algorithm is then applied to the total deaths column, and cluster labels are obtained.



```
# Fit KMeans model and Adjust the number of clusters as needed
mean_df = mean_df.apply(lambda x: x.astype('float64'))
kmeans = KMeans(n_clusters=6, random_state=0).fit(mean_df[["total_deaths"]])
```

```
C:\Users\femym\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: Futu
ange from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
  warnings.warn(
C:\Users\femym\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: Use
on Windows with MKL, when there are less chunks than available threads. You can
P_NUM_THREADS=1.
  warnings.warn(
```

```
# Cluster membership
cluster_labels = kmeans.labels_

# Create a Series with cluster labels and index from 'mean_df'
memb = pd.Series(cluster_labels, index=mean_df.index)
```

```
## Add a 'Cluster' column to the DataFrame
mean_df['Cluster'] = memb

# Display the updated DataFrame
print(mean_df)
```

Fig.12. Code for K means model and generating clusters

Then a scatterplot is generated to visualize the relationship between location and total death variables, taking into consideration the clusters assigned through K-means clustering.

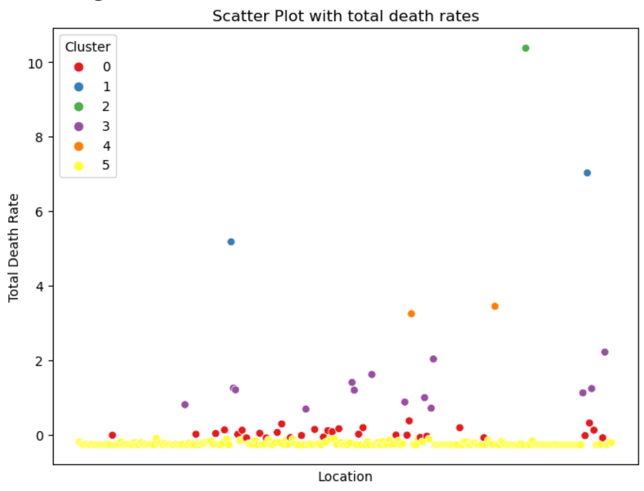
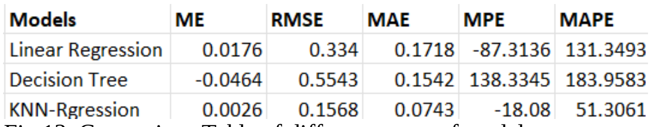


Fig.12. Scatterplot of Total Death rates with Location

By examining scatter plots, we can gain a better understanding of how the total number of death values are distributed among different location categories within the identified clusters.

III. RESULTS AND DISCUSSION

We have successfully used K-NN, Linear Regression and Decision Tree models to predict the total cases. Fig.13 shows the comparison of different errors of three models. K-NN Regression appears to perform the best among these models in terms of error metrics. It has the lowest RMSE and MAE values and a more reasonable percentage error.



Models	ME	RMSE	MAE	MPE	MAPE
Linear Regression	0.0176	0.334	0.1718	-87.3136	131.3493
Decision Tree	-0.0464	0.5543	0.1542	138.3345	183.9583
KNN-Rgression	0.0026	0.1568	0.0743	-18.08	51.3061

Fig.13. Comparison Table of different errors of models.

Linear Regression is moderate but has significant percentage errors, making it potentially less suitable if proportional mistakes are critical. Decision Tree seems to perform the worst, with the highest RMSE and alarming percentage errors. It could be overfitting the training data or may need better tuning. We did K-means to cluster the death rates according to the countries to identify the hotspots and do appropriate actions.

IV. CONCLUSION

In this study, we analyzed the COVID-19 data to predict the total cases using linear regression, decision tree, and K-nearest neighbors (KNN) regression methods. We also evaluated the regression summary to assess the performance of each method. Additionally, we clustered the death rate according to countries to gain insights into the patterns and variations.Furthermore, our clustering analysis allowed us to identify distinct patterns in the death rate among different countries. This information can be crucial for understanding the variations in mortality rates and informing targeted interventions and healthcare resource allocation.These insights can support decision-making in public health and aid in the development of effective strategies to mitigate the impact of the pandemic Future research should focus on incorporating additional variables and data sources to enhance the accuracy of the predictions. Moreover, exploring other machine learning algorithms and ensemble methods could provide further insights into the analysis of COVID-19.

V. REFERENCES

[1] Shmueli, G., Bruce, P. C., Gedeck, P., & Patel, N. R. (2020). Data mining for Business Analytics: Concepts, techniques and applications in Python. Wiley-Blackwell.

[2] Mathieu, E., Ritchie, H., Rodés-Guirao, L., Appel, C., Giattino, C., Hasell, J., Macdonald, B., Dattani, S., Beltekian, D., Ortiz-Ospina, E., & Roser, M. (2020, March 5). *Coronavirus (COVID-19) deaths*. Our World in Data. https://ourworldindata.org/covid-deaths

[3] Shih, D.-H., Shih, P.-L., Wu, T.-W., Li, C.-J., & Shih, M.-H. (2022, July 2). *Cluster analysis of US covid-19 infected states for vaccine distribution*. Healthcare (Basel, Switzerland).https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9323689/

[4] Varghese, D. (2019, May 10). Comparative study on classic machine learning algorithms. Medium. https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222