Machine Learning Final Project

University of Colombo

COVID-19 Cases prediction in the state of Chicago

S16099 – H.P.Kalubowila
S16033 – Pasindu Chamara
S16036 – Ridma Jayawardena
S16102 – Dinidu Dasun

Table of contents

Introduction

This report is analysis of the covid 19 dataset. We are specially focusing on predicting weekly covid 19 cases using machine learning models and performing clustering analysis on dataset. The dataset contains covid 19 cases, tests and deaths across different zip codes in the state of Chicago from the start of the pandemic to the end of 2024.

The main objective of this project is the **Prediction of Weekly COVID-19 Cases**: Using machine learning models to predict weekly case numbers based on historical data, including the impact of tests, case rates, and death rates.

Dataset Overview

The dataset can be downloaded from the following link

https://www.kaggle.com/datasets/mahdiehhajian/covid-19-cases-tests-and-deaths

The dataset includes the following key features:

- Tests - Weekly: Number of COVID-19 tests administered during the week.
- Tests - Cumulative: Total cumulative tests up to the week.
- Test Rate - Weekly: Tests per 100,000 population (weekly).
- Test Rate - Cumulative: Cumulative tests per 100,000 population.
- Percent Tested Positive - Weekly: Percentage of positive tests for the week.
- Percent Tested Positive - Cumulative: Cumulative percentage of positive tests.
- Deaths - Weekly: New COVID-19-related deaths for the week.
- Deaths - Cumulative: Total cumulative deaths up to the week.
- Death Rate - Weekly: Deaths per 100,000 population (weekly).
- Death Rate - Cumulative: Cumulative deaths per 100,000 population.
- Population: Total population of the ZIP code area.
- Row ID: Unique identifier for each record (e.g., 60622-2020-31).

The dataset is organized in weekly intervals, providing a time series for each region (zip code) over the course of the pandemic.

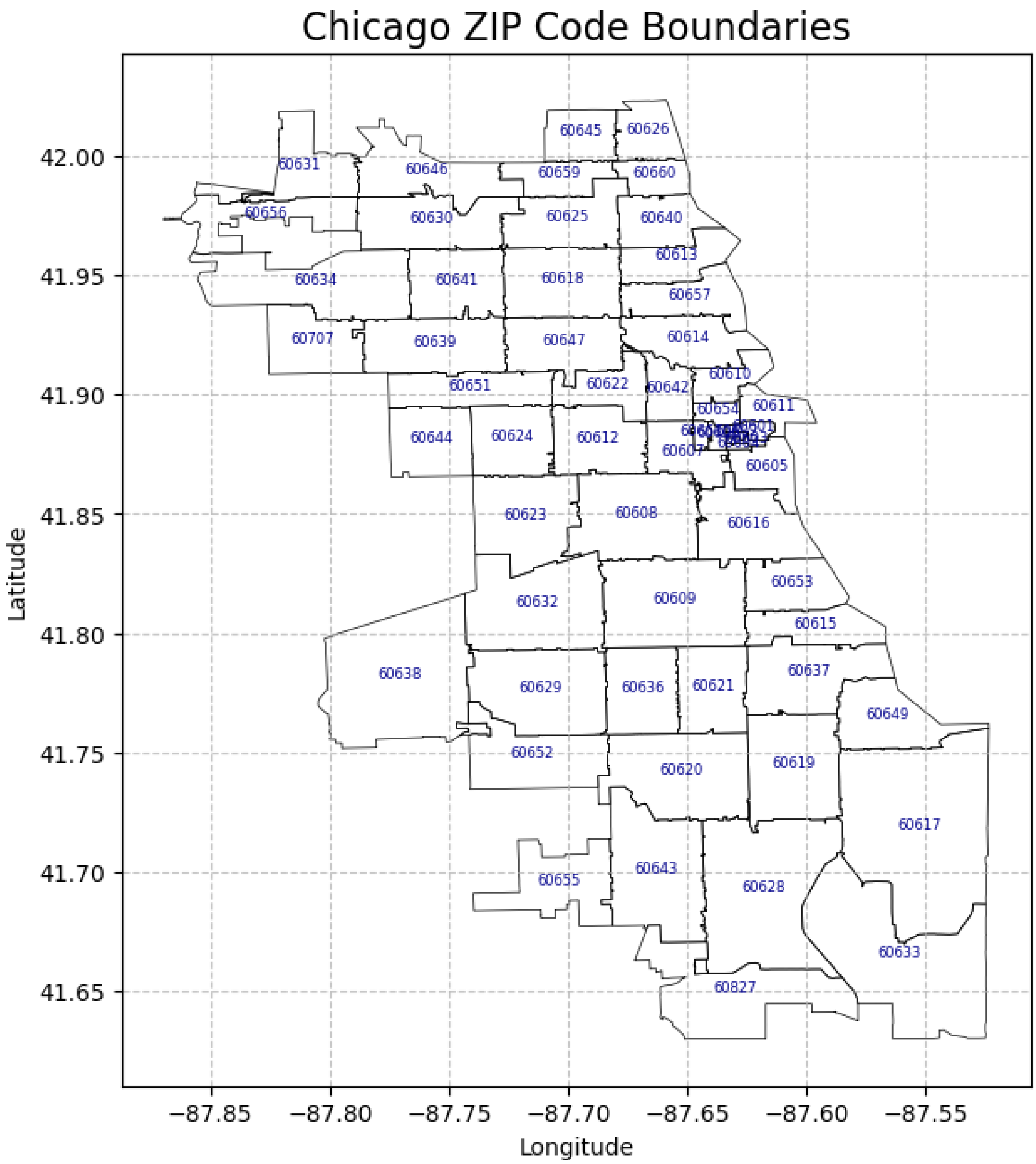


Figure 1: - State of Chicago with zip code boundaries

As a group project each member used a different machine learning model to predict Weekly COVID-19 Cases :

- Deep neural network model (DNN)
- Decision Tree Regression
- Random forest regression
- Gradient Boost Regression

Note that for each model the data pre-processing part might be a bit different from the others

Deep neural network model (DNN)

Data Preprocessing

The dataset initially required significant cleaning and preprocessing to ensure its suitability for analysis. The first step involved removing any rows where the ZIP Code was marked as "Unknown" to ensure that only valid geographic data was considered. Since compared to total no. of rows (13132) the rows with missing zip code values are mostly negligible (211) the removing of those rows can be justified.

To handle location information, the ZIP Code Location field, which contained coordinates in a string format, was parsed to extract latitude and longitude values. A custom function was used to split the string and extract the relevant geographic coordinates. Any rows with missing or invalid coordinates were dropped to maintain data integrity.

Next, the date columns, Week Start and Week End, were converted to pandas datetime objects to enable proper time-series analysis. This conversion allowed for the grouping of data by week and region, ensuring that temporal trends could be captured effectively.

For clustering, geographic coordinates (longitude and latitude) were scaled and used to apply the KMeans algorithm to group ZIP codes into 7 regions. This clustering was important for analyzing COVID-19 trends within different geographic areas, as it allowed for regional aggregation of cases, tests, and deaths. The clustering is done due to there being too less data rows per zip code. So we combined some zip codes into one region to get more data rows per region

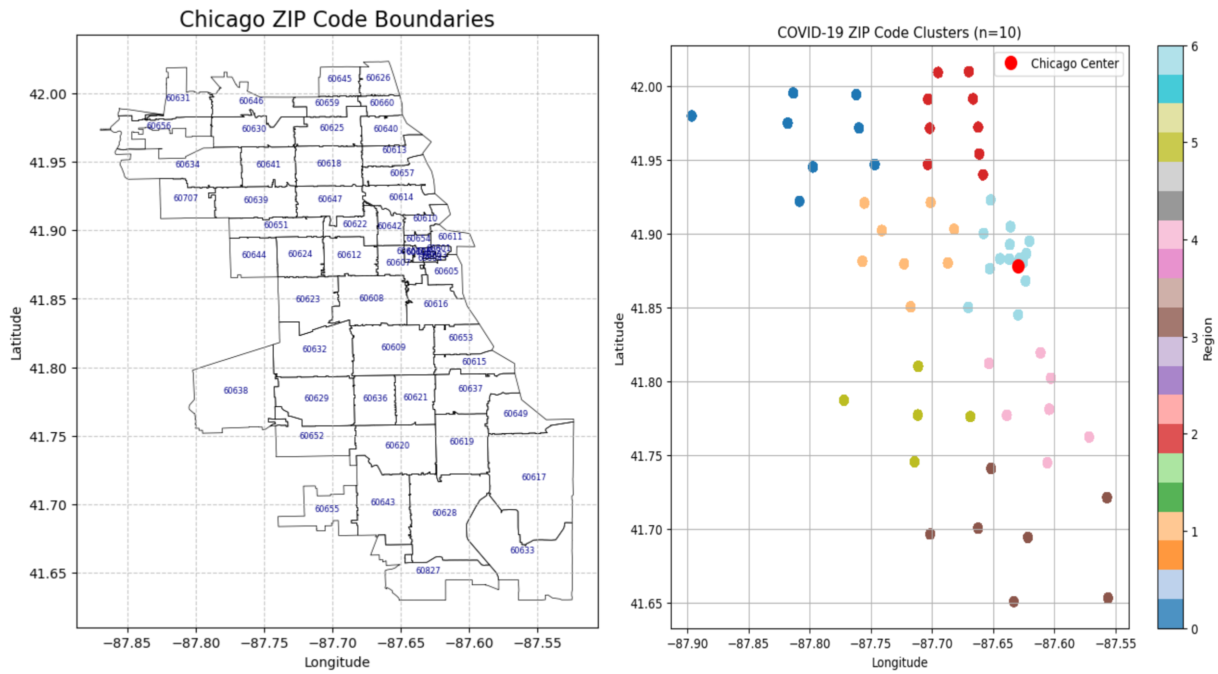


Figure 2: - Chicago area is divided into 7 main regions by KMeans clustering

The data was then aggregated by region and week, summing up weekly and cumulative cases, tests, and deaths. Positivity rates (percentage of tests returning positive results) were calculated for both weekly and cumulative data, with special handling for cases where the number of tests was zero. Additionally, rates per 100,000 people were calculated for case rates, test rates, and death rates, providing standardized metrics for easier comparison across regions. Finally, all columns were rounded appropriately, with integer values rounded to whole numbers and decimal values rounded to three places for clarity. Here the rest of the missing values in rows are dealt by functions used for aggregation. This preprocessing step ensured that the dataset was clean, consistent, and ready for further analysis and modeling.

In the next step, we calculate the correlation matrix to examine how different features in the dataset relate to the target variable, 'Cases - Weekly'. The correlation matrix computes the relationship between all numeric columns, with values ranging from -1 (perfect negative correlation) to 1 (perfect positive correlation), and 0 indicating no relationship. By isolating the correlation with the target column, we can identify which features have the strongest influence on the number of weekly cases. Features with a high positive correlation suggest a direct relationship with the target, while those with a negative correlation indicate an inverse relationship. Features with a low correlation may have little impact on predicting the target. This step helps us prioritize the most influential variables, which can be valuable for further analysis or model-building. Visualizing the correlations through bar plots or heatmaps can also provide a clearer understanding of the relationships, aiding in feature selection for predictive modeling.

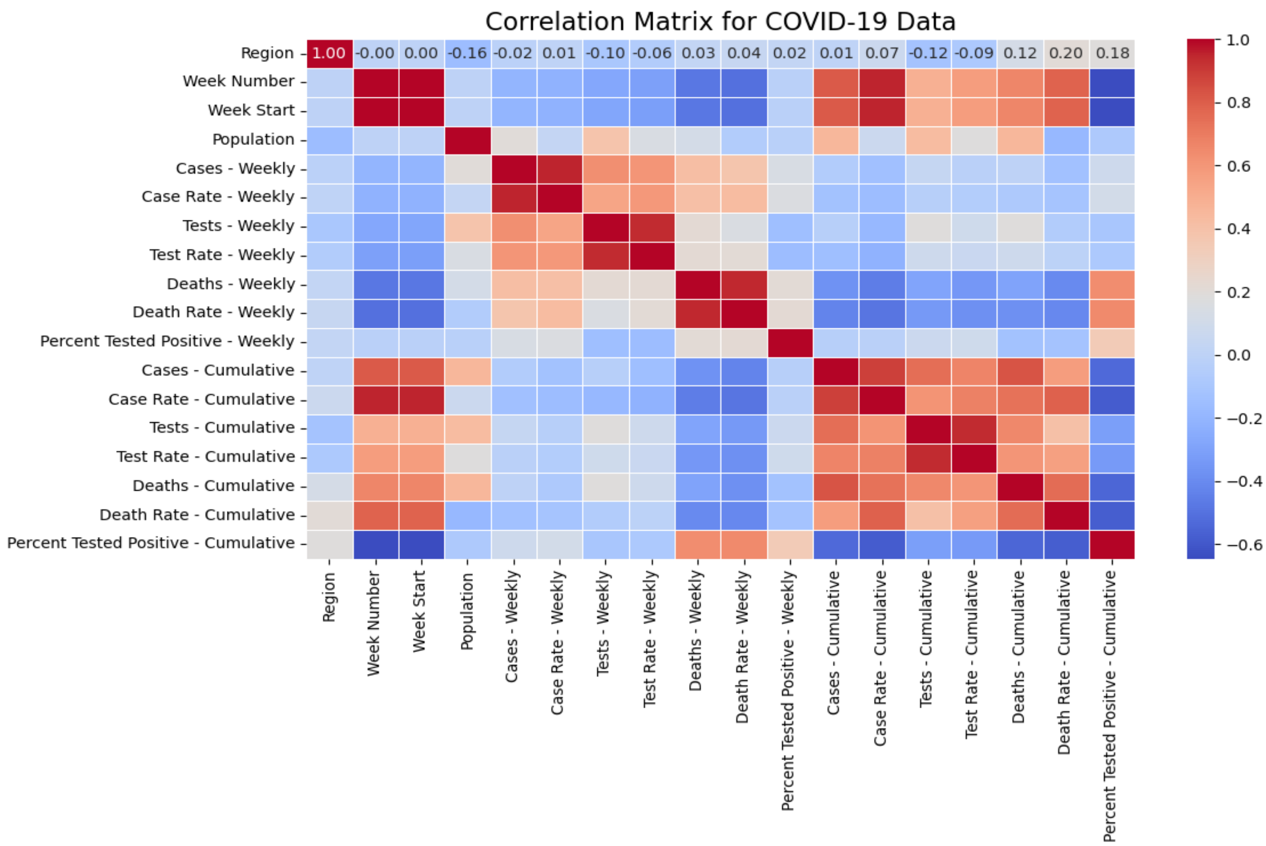


Figure 2 :- the correlation matrix

Creating Lag features and data-scaling

In this section, we perform feature engineering by creating lag features to improve the predictive model for weekly cases. Lag features are constructed by shifting the values of relevant variables, such as 'Tests - Weekly', 'Test Rate - Weekly', and 'Case Rate - Weekly', by one week within each region. This allows the model to capture temporal dependencies, as the number of tests, case rates, and other factors from the previous week could influence the current week's case numbers. After generating these lag features, we drop any rows containing missing values that result from the shifting process. The next step involves defining the target variable ('Cases - Weekly') and selecting a set of correlated features that are most likely to influence the target. These include variables like 'Case Rate - Weekly', 'Tests - Weekly', and 'Deaths - Weekly'.

We then prepare the data for modeling by separating the features (X) and the target (y), followed by standardization to scale the data, which is important for most machine learning algorithms. The dataset is then split into training and testing sets, with 80% used for training and 20% reserved for testing, ensuring that no shuffling occurs to maintain the temporal order. The corresponding dates for the test set are also stored to facilitate time-aware evaluation. This approach helps build a model that can predict future cases based on past trends, with a clear distinction between training and testing data for robust evaluation.

Construction of the DNN and output graphs

We developed an enhanced Deep Neural Network (DNN) architecture to predict weekly COVID-19 cases. The model consists of several dense layers with ReLU activation functions, providing the network with the ability to learn complex patterns in the data. The architecture includes three main layers: the first with 128 neurons, the second with 256 neurons, and the third with 128 neurons. Dropout layers are strategically added after each dense layer to prevent overfitting, with a dropout rate of 30% (0.3). The final layer has a single neuron, which produces the predicted number of cases.

The model is compiled with the Adam optimizer and a custom Huber loss function, which is less sensitive to outliers than the traditional Mean Squared Error (MSE), making it well-suited for the prediction task. The Huber loss includes a delta parameter, set to 1.5, that defines the point at which the loss function transitions from quadratic to linear, reducing the impact of large errors.

We use early stopping during training, which monitors the validation loss and stops the training if the loss does not improve for 15 consecutive epochs. This helps prevent overfitting and ensures that the best model weights are restored. The model is trained for up to 200 epochs with a batch size of 32, and the training process is validated on the test set.

After training, predictions are made on the test set, and the predicted values are inverse-transformed using the same scaler used during data preprocessing. A plot is generated to compare the actual vs. predicted cases over time. The x-axis represents the start date of each week, and the y-axis shows the number of cases. This visual comparison provides insights into the model's performance and its ability to predict trends in COVID-19 cases.

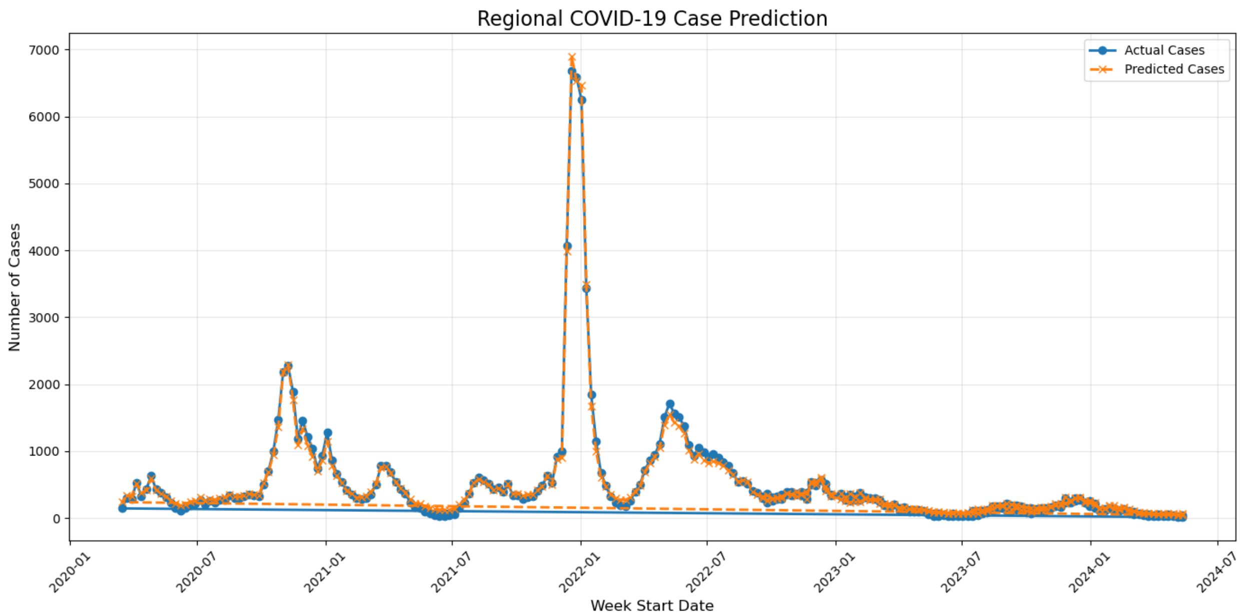


Figure 4: - The actual and predicted graphs of cases varying with time in an average region in Chicago (Here Chicago has been divided into 7 regions)

Model evaluation

The model performance report shows the following results:

- **Test Huber Loss**: 0.0017
  This value reflects the model's overall error during prediction. A Huber loss close to zero indicates that the model is performing well in terms of minimizing errors, particularly when outliers are present.
- **MAE (Mean Absolute Error)**: 0.0414 (Scaled), 567.5 (Actual Cases)
  The **MAE of 0.0414 (scaled)** represents the average absolute error in the standardized data. When translated back to the actual scale, the **MAE of 567.5 cases** means that, on average, the model's predictions deviate by approximately 568 cases from the true number of COVID-19 cases in the test set. This is a reasonable error margin depending on the scale of the problem, but there may still be room for improvement in the model's prediction accuracy, especially when dealing with larger numbers of cases.

This performance suggests that the model can make reasonable predictions with a manageable level of error, but it could be further fine-tuned to reduce the MAE, depending on the desired precision for forecasting COVID-19 cases.

Decision Tree Regression

Data Preprocessing

- Data preprocessing is crucial for building a reliable machine learning model. We performed several steps to clean and prepare the dataset, ensuring it was in a suitable format for model training.
- **Handling Missing Values:**
  - Missing data is a common issue in real-world datasets. To handle this, we applied different imputation strategies:
    - For **categorical variables**, missing values in the 'ZIP Code' column were filled with the **mode** (the most frequent value) of that column. This ensures that the missing ZIP Code data is replaced with the most common location, maintaining the integrity of geographic data.
    - For **numerical variables**, such as 'Cases - Weekly', 'Tests - Weekly', 'Case Rate - Weekly', and 'Percent Tested Positive - Weekly', we filled missing values with the **mean** of each respective column. The mean is a simple but effective way to preserve the general distribution of the data without introducing significant bias, especially when a small percentage of values are missing.
- **Removing Irrelevant Columns:**
  - To streamline the dataset and improve model performance, we removed **irrelevant columns** that did not contribute to the prediction task. Specifically:
    - **'Row ID'**: This was a unique identifier for each row, which has no predictive value and could introduce unnecessary complexity into the model.
    - **'ZIP Code Location'**: This column contained additional location details that were not needed for the model. Instead, we focused on the 'ZIP Code' itself, which was the relevant feature for geographic analysis.
- **Encoding Categorical Variables:**
  - **'ZIP Code'** is a categorical variable with multiple unique values. Since machine learning algorithms generally require numerical inputs, we performed **one-hot encoding** on the 'ZIP Code' column. This process converts each unique ZIP Code into a separate binary feature. This ensures that the model can treat different ZIP Codes as separate entities without introducing any ordinal relationships.
- **Converting Date Columns:**
  - The dataset included columns with date information, namely **'Week Start'** and **'Week End'**. These columns were originally in string format, which is not suitable for analysis or modeling. To address this, we converted these columns

into **datetime** format, allowing us to easily extract useful time-related features, such as:

- **Week Start Month** and **Year**: This can be valuable for identifying seasonal trends or changes over time in COVID-19 cases.
- The conversion to datetime also facilitates **time series analysis** if needed, where we could further analyze trends and patterns in the dataset over time.

Model Development

1. Handling Missing Values
   - Numerical Columns: Missing values in columns like 'Cases - Weekly', 'Cases - Cumulative', 'Case Rate - Weekly', and 'Tests - Weekly' were filled with the mean of each respective column.
   - Categorical Column: For categorical variables like 'ZIP Code', missing values were filled with the most frequent value.

```
#Data Cleaning and Preprocessing
df['ZIP Code'].fillna(df['ZIP Code'].mode()[0], inplace=True)
df['Cases - Weekly'].fillna(df['Cases - Weekly'].mean(), inplace=True)
df['Cases - Cumulative'].fillna(df['Cases - Cumulative'].mean(), inplace=True)
df['Case Rate - Weekly'].fillna(df['Case Rate - Weekly'].mean(), inplace=True)
df['Case Rate - Cumulative'].fillna(df['Case Rate - Cumulative'].mean(), inplace=True)
df['Tests - Weekly'].fillna(df['Tests - Weekly'].mean(), inplace=True)
df.isnull().sum()
```

```
df['ZIP Code'] = df['ZIP Code'].astype(str)
# Convert date columns to datetime format
df['Week Start'] = pd.to_datetime(df['Week Start'])
df['Week End'] = pd.to_datetime(df['Week End'])
```

2. One-Hot Encoding: Categorical features such as 'ZIP Code' were encoded using one hot encoding to convert them into numerical format suitable for machine learning models.

```
df = pd.get_dummies(df, columns=['ZIP Code'], drop_first=True)
```

Feature Selection and Correlation Analysis

In the dataset it included both numerical and categorical features. A correlation analysis was conducted to identify the most relevant features to predicting Cases - Weekly.

```
plt.figure(figsize=(10, 8))
correlation_matrix = df[numerical_columns].corr()
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Heatmap of Features')
plt.show()
```

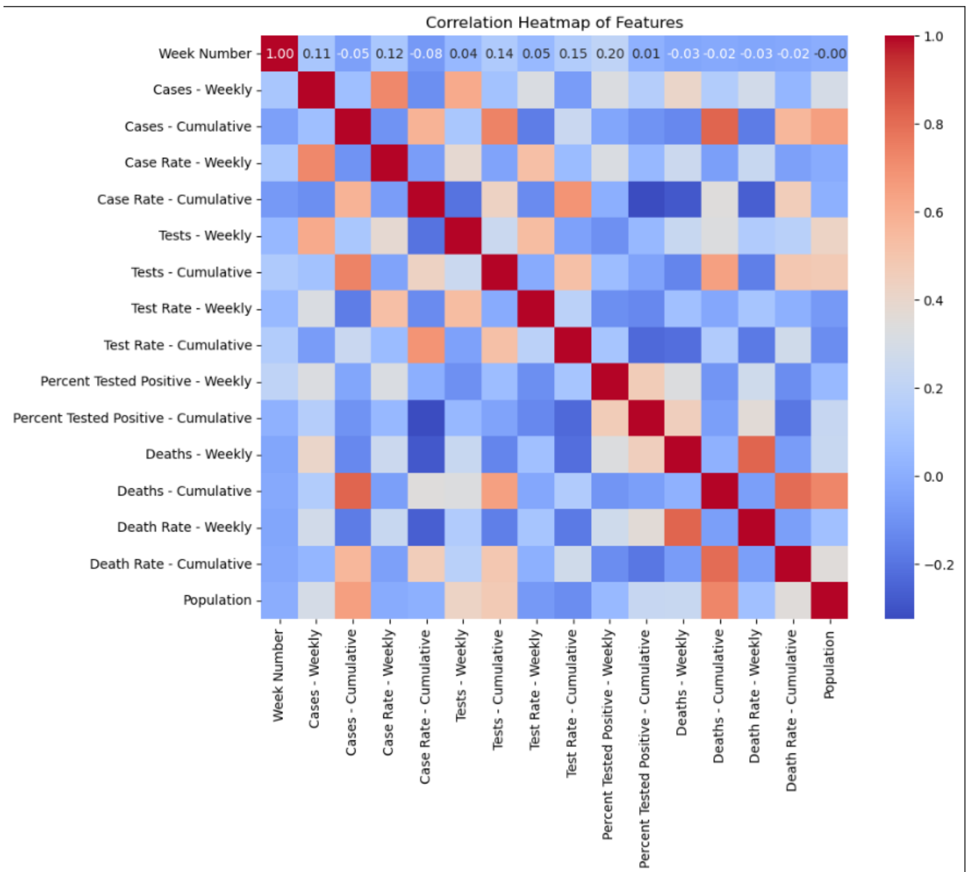


Figure 5: Correlation heatmap of features

Model Building

We used the Decision Tree Regressor to predict weekly COVID-19 cases based on historical data. The first step was to split the data into training and testing sets. The model was then trained on the training data, learning patterns between the features (like case rates, tests, and ZIP codes) and the target variable (weekly cases).

```python
X = df.drop(columns=["Cases - Weekly"])
y = df["Cases - Weekly"]

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

```python
model = DecisionTreeRegressor(random_state=42)
model.fit(X_train, y_train)
```

```python
y_pred = model.predict(X_test)
```

```python
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"R-squared (R² Score): {r2*100:.4f}%")
```

- Mean Absolute Error (MAE): 4.1203
- Mean Squared Error (MSE): 711.8736
- R^2 Score: 95.8450%

Hyperparameter Tuning

Hyperparameter tuning is the process of finding the best settings for the model to improve its performance. We used GridSearchCV to test different combinations of hyperparameters, such as decision tree, the number of samples required to split a node, and the maximum number of features to consider at each split. We found the best set of parameters that helped the model make more accurate predictions.

The result, increase in the R^2 score and a reduction in the Mean Squared Error. This shows that tuning helped the model make better use of the data, improving its ability to predict weekly COVID-19 cases.

- Optimized Mean Absolute Error (MAE): 4.1368
- Optimized Mean Squared Error (MSE): 674.2795
- Optimized R^2 Score: 96.06%

```python
param_grid = {
    'max_depth': [5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 5, 10],
    'max_features': [None, 'sqrt', 'log2']
}

grid_search = GridSearchCV(DecisionTreeRegressor(random_state=42), param_grid, cv=5, scoring='r2', n_jobs=-1)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
print(f"Best Parameters: {best_params}")

best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)

mse_best = mean_squared_error(y_test, y_pred_best)
r2_best = r2_score(y_test, y_pred_best)
mae_best = mean_absolute_error(y_test, y_pred_best)

print(f"Optimized MSE: {mse_best:.4f}")
print(f"Optimized MAE: {mae_best:.4f}")
print(f"R-squared (R² Score): {r2_best*100:.2f}%")
```

Visualizations

- Actual vs Predicted Weekly Cases: This graph showed how closely the model's predictions matched the actual weekly COVID-19 cases. It helped us see that the model was tracking the overall trends well, even though there were some fluctuations.

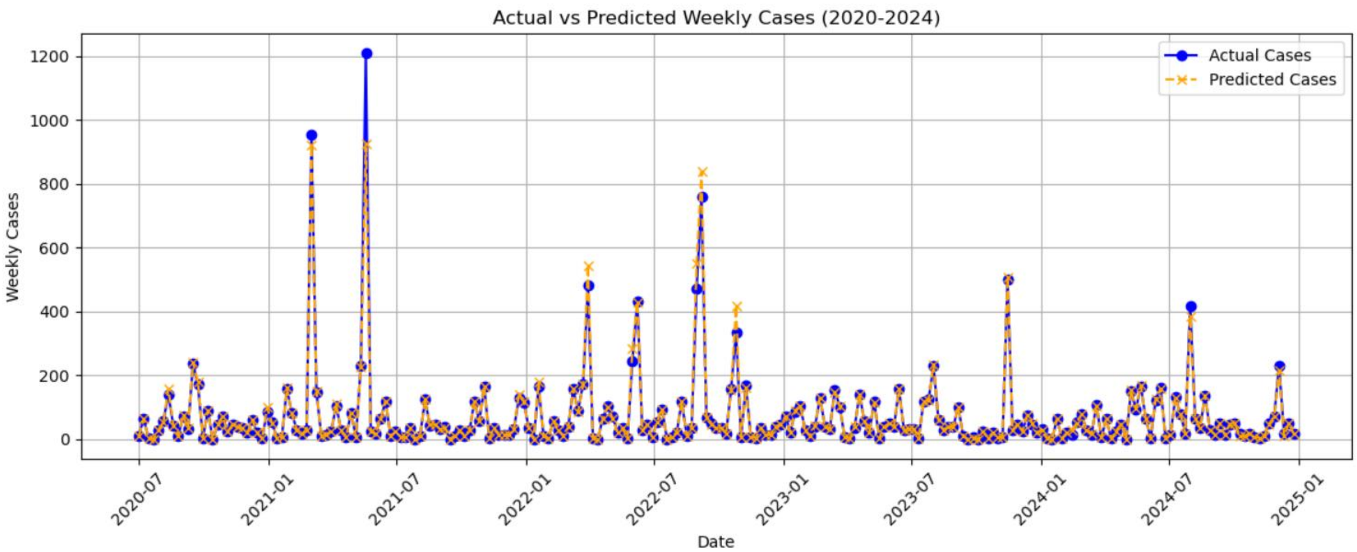


Figure 6: Actual vs Predicted Weekly Cases (2020-2024)

- Predicted vs Actual Weekly Cases: The scatter plot compared predicted values to the actual values, with the closer the points are to the red line, the better the model’s predictions. This gave us a clear picture of how accurate the model was across different weeks.

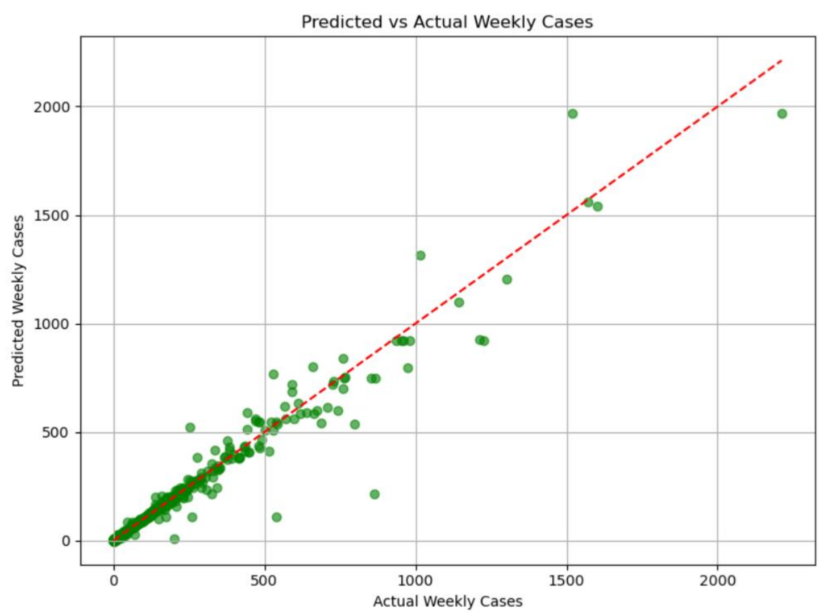


Figure 6: Predicted vs actual weekly cases

Model Evaluation

The model was trained using a Decision Tree Regressor to predict weekly COVID-19 cases based on features most closely related to the target variable. Initially, the model was trained with default settings, and the results were promising. After tuning the hyperparameters using GridSearchCV, the model's performance improved.

Model	**MSE**	**R^2 Score**
Initial Decision Tree Regressor	711.87	95.8450%
Tuned Decision Tree Regressor	674.2795	96.06%

The tuning process led to a noticeable improvement in the model's ability to predict weekly COVID-19 cases, reducing the MSE and increasing the R^2 Score. This indicates that the adjustments to the model's settings helped it generalize better to new, unseen data.

Conclusion

The Decision Tree Regressor model performed well in predicting weekly COVID-19 cases, explaining 96.06% of the variance in the data after hyperparameter optimization. The model was able to capture the trends in the data with high accuracy, making it a valuable tool for predicting future COVID-19 cases.

Feature Importance is the Decision Trees provided useful insights into which features most influenced case predictions. In the clustering of data based on weekly cases and testing rates highlighted distinct groups in the dataset, which may be useful for targeted interventions.

Random Forest Regression

Data Preprocessing

Before proceeding with analysis and modeling, the dataset underwent a comprehensive preprocessing phase to ensure data quality and relevance. Initially, rows with "Unknown" ZIP Codes were removed to retain only geographically valid entries. Given that these constituted a minimal portion of the data their exclusion was deemed justified. Additionally, the ZIP Code Location field, which contained coordinates in string format, was parsed using a custom function to extract latitude and longitude values. Any rows with missing or invalid geographic data were also discarded to maintain the integrity of location-based analysis.

To support time-series analysis, the date columns "Week Start" and "Week End" were converted into pandas datetime objects, facilitating the grouping of data by both week and region to better capture temporal trends. Furthermore, geographic coordinates were scaled and clustered using the KMeans algorithm, grouping ZIP Codes into seven regions. This step addressed the issue of sparse data at the individual ZIP Code level by aggregating them into broader regions, thereby enabling more robust regional analysis of COVID-19 cases, tests, and deaths.

Additional preprocessing measures were implemented to prepare the data for modeling. Missing values in various features were handled by either imputing with mean or median values or removing the records entirely, depending on the extent of missing data. Feature selection was performed to enhance model efficiency, with fields such as "Week Start," "Week End," and "ZIP Code Location" excluded, as they were not directly informative for predicting case counts. Both numerical and categorical variables relevant to prediction were retained. Finally, the dataset was split into training and testing subsets, typically allocating 70%–80% of the data for training and the remainder for testing, to ensure proper evaluation of model performance.

Model Selection

A Random Forest Regressor was selected for this prediction task due to its ability to handle complex, non-linear relationships in the data and its resilience against overfitting, especially when a large number of trees are used. The model can handle a variety of features and does not require heavy preprocessing or scaling of input data. The following parameters were configured for the Random Forest model:

- **n_estimators**: The number of trees in the forest, set to 100.

- **max_depth**: The maximum depth of each tree, which helps control the complexity of the model and reduces the likelihood of overfitting.
- **min_samples_split**: The minimum number of samples required to split an internal node, preventing the model from learning overly specific details from the data.

Model Training

The training process included several important steps:

- **Data Preparation**: The dataset was split into training and test sets using an 80/20 ratio, where 80% of the data was used to train the model, and the remaining 20% was used for testing. This allows the model to generalize well while being evaluated on unseen data.
- **Feature Selection**: Unnecessary columns (such as dates) were removed from the feature set, leaving only the relevant predictors. Categorical features were encoded using **LabelEncoder**.
- **Model Training**: The model was trained using the RandomForestRegressor with the selected features. The training process involved fitting the model to the training data and adjusting its internal parameters based on the input features.

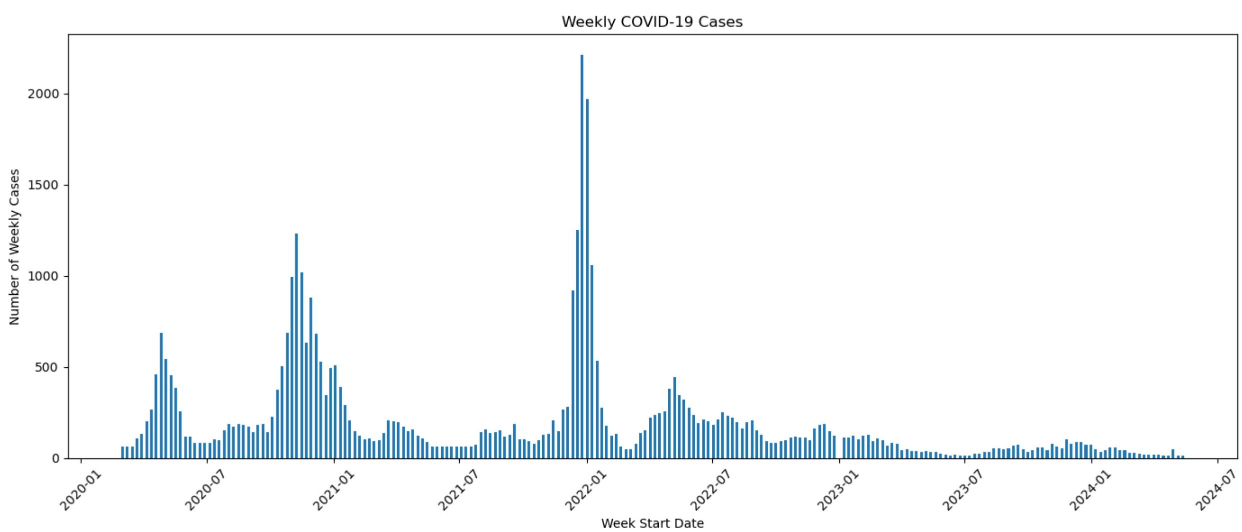


Figure 7: Weekly Covid 19 Cases Time bar chart

Model Visualization

The plot presents a time series bar chart where the x-axis represents the week start date, spanning from early 2020 to mid-2024, and the y-axis shows the number of weekly cases.

Key observations from the plot:

- A sharp spike in cases occurred in early 2022, indicating a major wave of COVID-19 infections during that period.
- The trend shows seasonal fluctuations with higher numbers of cases in certain months and a general decline as we move towards later 2023 and 2024.
- There are periods of relatively low cases between the major waves, highlighting the effectiveness of lockdown measures and other interventions during those times.

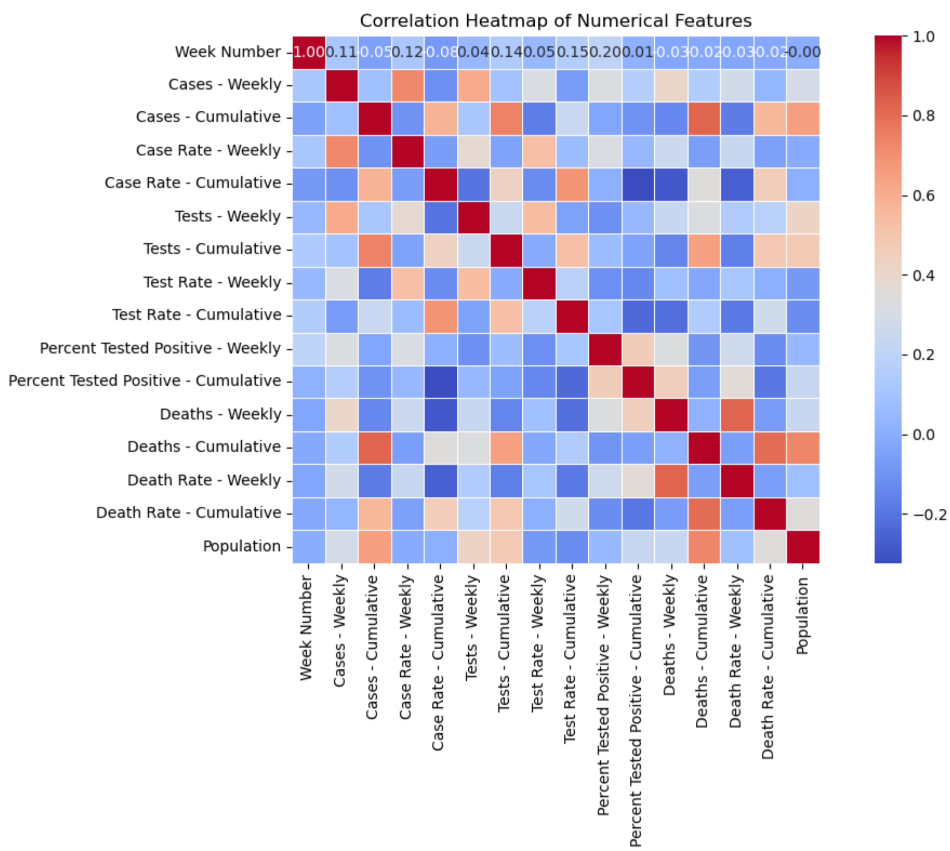


Figure 8: Correlation heatmap of Numerical Features

This chart aligns with the model's evaluation, showing a fluctuation pattern that the Random Forest Regressor has been trained to predict. The prediction task aims to estimate the number of cases based on weekly data, and the model leverages this trend in the temporal data to make accurate predictions, as evidenced by the R^2 score of 0.96, showing how well it captures this fluctuation.

The heatmap provides a visual representation of pairwise correlations between the features, with color intensity indicating the strength of the correlation.

Key insights from the heatmap:

- There is a strong positive correlation between Cases - Weekly and Cases - Cumulative, reflecting that as the cumulative cases increase, the weekly cases tend to increase as well. This is expected, as new cases tend to build on the prior cases in ongoing outbreaks.
- Tests - Cumulative shows a positive correlation with Test Rate - Weekly, indicating that the rate of testing increases as more cumulative tests are performed.
- There are negative correlations between features like Deaths - Weekly and Population, suggesting that population size does not necessarily correlate directly with the number of deaths in a given week. Other negative correlations might also exist, such as between Death Rate - Weekly and Test Rate - Cumulative, showing a weaker relationship between deaths and the overall testing rate.

This correlation analysis aids in model development by identifying which features are highly related and can help the Random Forest Regressor learn better. By selecting relevant features with strong correlations, the model can make more accurate predictions. For instance, Cases - Weekly is directly related to the target variable, making it a crucial feature for the model prediction task.

Model Evaluation

After training, performance of the model was evaluated using the following metrics:

- **Mean Absolute Error (MAE)**: Measures the average magnitude of errors in the predictions without considering their direction. This gives us an idea of the average prediction error.
- **Mean Squared Error (MSE)**: Measures the average squared difference between predicted and actual values. It penalizes larger errors more than MAE.

- **R-squared (R^2)**: Indicates how well the predictions fit the actual data. A value of 1 represents a perfect fit, while a value closer to 0 suggests that the model does not explain much of the variance in the target variable.

Results

The Random Forest Regressor model performed well with the following evaluation metrics:

- Mean Absolute Error (MAE): 6.28
- Mean Squared Error (MSE): 708.12
- R-squared (R^2): 0.96, or 96%. This indicates that 96% of the variance in the target variable (weekly cases) is explained by the model, which is a strong performance.

These results show that the Random Forest model can predict the weekly COVID-19 cases with high accuracy. The high R^2 and low MAE and MSE values suggest that the model has effectively learned from the data and makes reliable predictions.

Gradient Boost Regression

Suitability of the model

Gradient Boosting Regression is an ensemble machine learning algorithm that constructs a predictive model in a stage-wise fashion. It combines multiple weak learners, typically decision trees, to create a strong and accurate model. The key idea behind gradient boosting is that each new model attempts to correct the residual errors made by the previous ones. This is achieved by minimizing a loss function (such as mean squared error for regression tasks) using gradient descent techniques.

The Gradient Boosting Regressor is especially effective for:

- Capturing non-linear relationships between variables.
- Handling complex datasets with a mixture of numerical and categorical features
- It is relatively robust to outliers and noise, especially with proper preprocessing

- Providing feature importance scores that help identify which features contribute most to the predictions

This model is well-suited for the COVID-19 dataset used in our project because the dataset contains multiple interrelated features—such as weekly tests, cumulative cases, and case rates—that influence the weekly case count. These relationships are not necessarily linear, and the model benefits from the Gradient Boosting Regressor's ability to capture non-linear dependencies. Furthermore, the dataset may contain some noise and missing values, which gradient boosting is relatively robust against, especially after preprocessing. The model's built-in feature importance of ranking also aids in selecting the most impactful predictors, improving interpretability and predictive power. Overall, the Gradient Boosting Regression model provides a reliable and effective solution for predicting weekly COVID-19 cases based on historical and demographic trends in the data.

Data Preprocessing

To ensure the model's performance and reliability, I performed several preprocessing steps. I filled in missing values in important numerical columns such as 'Cases—Weekly', 'Cases—Cumulative', and testing rates using the mean of each respective column. For categorical data such as 'ZIP Code Location', the most frequent value was used to fill in missing entries. I also converted the 'Week Start' and 'Week End' columns into the date-time format for better time-based analysis.

```
print("Dataset Null Value")
print(df.isnull().sum())
print("Number of duplicates values: ", df.duplicated().sum())

df['Cases - Weekly'] = df['Cases - Weekly'].fillna(df['Cases - Weekly'].mean())
df['Cases - Cumulative'] = df['Cases - Cumulative'].fillna(df['Cases - Cumulative'].mean())
df['Case Rate - Weekly'] = df['Case Rate - Weekly'].fillna(df['Case Rate - Weekly'].mean())
df['Case Rate - Cumulative'] = df['Case Rate - Cumulative'].fillna(df['Case Rate - Cumulative'].mean())
df['Tests - Weekly'] = df['Tests - Weekly'].fillna(df['Tests - Weekly'].mean())

df['ZIP Code Location'] = df['ZIP Code Location'].fillna(df['ZIP Code Location'].mode()[0])

df['Week Start'] = pd.to_datetime(df['Week Start'], errors='coerce')
df['Week End'] = pd.to_datetime(df['Week End'], errors='coerce')

print("Dataset Null Value")
print(df.isnull().sum())
```

Feature Selection and Correlation Analysis

And performed a correlation analysis to identify the most relevant features for the model. A heatmap was generated to visualize relationships among numerical variables, and features with a correlation threshold greater than 0.05 with the target variable 'Cases - Weekly' were selected. This step helped narrow down the input space for the model and improved training efficiency.

```
df['Week_Start_Month'] = df['Week Start'].dt.month_name()
df['Week Start Year'] = df['Week Start'].dt.year

num_cols = df.select_dtypes(include=np.number).columns
cat_cols = df.select_dtypes(exclude=np.number).columns

plt.figure(figsize=(12, 8))
correlation_matrix = df[num_cols].corr()
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Heatmap of Features')
plt.show()
```

```
threshold = 0.05

correlation_matrix = df.corr(numeric_only=True)
high_corr_features = correlation_matrix.index[abs(correlation_matrix["Cases - Weekly"]) > threshold].tolist()
high_corr_features.remove("Cases - Weekly")
print(high_corr_features)
```

Model Building

Used the Gradient Boosting Regressor for this task due to its effectiveness in handling structured tabular data. The dataset was split into training and testing subsets using an 80-20 ratio. The initial model was trained with default hyperparameters such as a learning rate of 0.1, a maximum depth of 3, and 100 estimators. The model achieved a good baseline R^2 score, demonstrating its capacity to learn patterns in the deterministic approach to Mean Squared 541.1198 and R^2 square of 96.84%.

```
X_selected = df[high_corr_features]
Y = df['Cases - Weekly']
X_train, X_test, y_train, y_test = train_test_split(X_selected, Y, test_size=0.2, random_state=42)

model1 = GradientBoostingRegressor(learning_rate=0.1, max_depth=3, min_samples_leaf=1, min_samples_split=2, n_estimators=100)
model1.fit(X_train, y_train)

y_pred = model1.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"{model1.__class__.__name__}")
print(f"MSE: {mse}")
print(f"R2 Score: {r2 * 100:.2f}%"+'\n')
```

Hyperparameter Tuning

To enhance the model's predictive performance, conducted hyperparameter tuning using RandomizedSearchCV. A grid of potential values for parameters such as 'n_estimators', 'learning_rate', 'max_depth', and 'min_samples_split' was defined. The RandomizedSearchCV performed 50 random searches with 5-fold cross-validation, optimizing for the R^2 score. The best model obtained from this tuning step was retrained and evaluated on the test set.

```
param_dist = {
    "n_estimators": [50, 100, 200, 500],
    "learning_rate": [0.01, 0.05, 0.1, 0.2],
    "max_depth": [3, 4, 5, 6, 7],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "subsample": [0.7, 0.8, 0.9, 1.0],
    "max_features": ["sqrt", "log2", None]  # Removed 'auto'
}

model = GradientBoostingRegressor(random_state=42)

#RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=model,
    param_distributions=param_dist,
    n_iter=50,
    cv=5,
    scoring='r2',  # Optimize for R2 score
    n_jobs=-1,  # Use all available CPU cores
    verbose=2,
    random_state=42
)

random_search.fit(X_train, y_train)

best_params = random_search.best_params_
print(f"Best Hyperparameters: {best_params}")

best_model = GradientBoostingRegressor(**best_params)
best_model.fit(X_train, y_train)

# Evaluate
y_pred_best = best_model.predict(X_test)
mse_best = mean_squared_error(y_test, y_pred_best)
r2_best = r2_score(y_test, y_pred_best)

print("Tuned Gradient Boosting Regressor")
print(f"MSE: {mse_best}")
print(f"R2 Score: {r2_best * 100:.2f}%")
```

Model Evaluation

The final model was evaluated using Mean Squared Error (MSE) and R^2 score. The tuned model achieved a lower MSE and a higher R^2 score compared to the initial version, indicating improved performance. Additionally, a line plot was generated to visualize actual versus predicted weekly case counts over time, which showed strong alignment and temporal consistency.

```
y_test_reset = y_test.reset_index()
y_pred_series = pd.Series(y_pred_best, index=y_test_reset.index, name='Predicted')

# Combine Week Start with actual and predicted values
comparison_df = pd.DataFrame({
    'Week Start': df.loc[y_test_reset['index'], 'Week Start'].values,
    'Actual': y_test.values,
    'Predicted': y_pred_series.values
})

# Sort by date to
comparison_df = comparison_df.sort_values(by='Week Start')

sns.set(style='whitegrid')

plt.figure(figsize=(18, 6))

# Smoother lines using alpha
plt.plot(comparison_df['Week Start'], comparison_df['Actual'], label='Actual', color='dodgerblue', linewidth=1.5)
plt.plot(comparison_df['Week Start'], comparison_df['Predicted'], label='Predicted', color='darkorange', linewidth=1.5, linestyle='--')

plt.title(' Actual vs Predicted Weekly COVID-19 Cases Over Time', fontsize=14, fontweight='bold')
plt.xlabel('Week Start', fontsize=12)
plt.ylabel('Weekly Cases', fontsize=12)

# Rotate and limit ticks for clarity
plt.xticks(rotation=45)
plt.locator_params(axis='x', nbins=12)

plt.grid(True, linestyle='--', alpha=0.4)
plt.legend()
plt.tight_layout()
plt.show()
```

The Gradient Boosting Regressor proved to be an effective approach for predicting weekly COVID-19 cases based on the given dataset. With careful preprocessing, feature selection, and hyperparameter tuning, the model delivered accurate and meaningful predictions. This approach can be further enhanced by incorporating more contextual data and extending the temporal analysis.

Evaluation of GBR model

The model was trained using Gradient Boosting Regressor to predict weekly COVID-19 cases based on features with high correlation to the target variable. The initial model was trained with default hyperparameters, while the tuned model was optimized using RandomizedSearchCV.

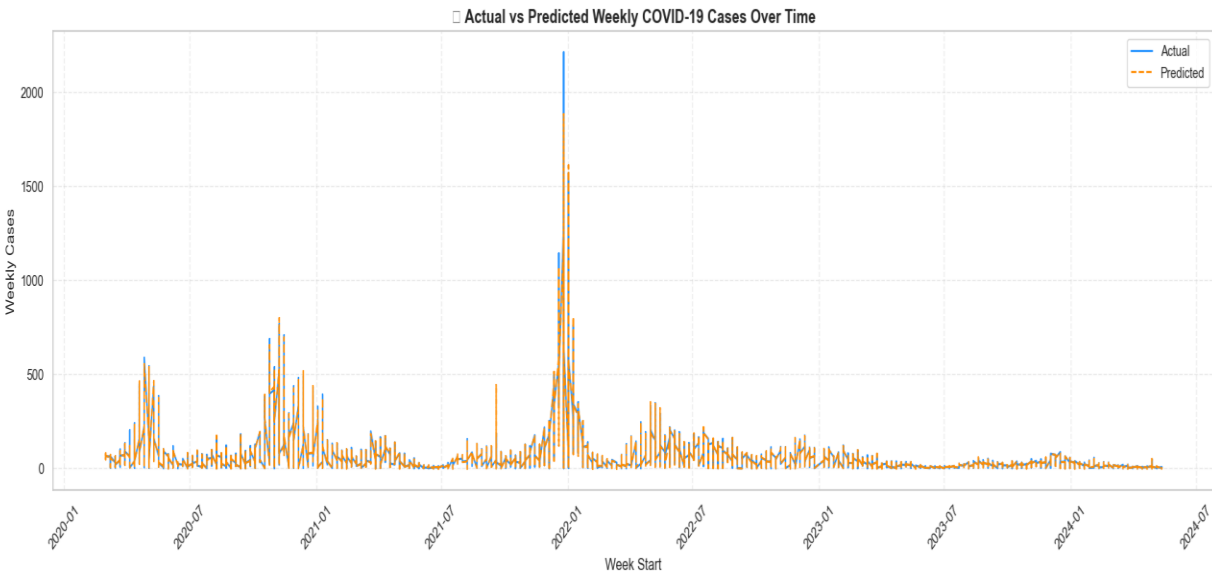


Figure 9: Actual vs Predicted weekly COVID 19 cases over time

- This plot shows the Actual vs Predicted comparison over time using Week Start as the x-axis.

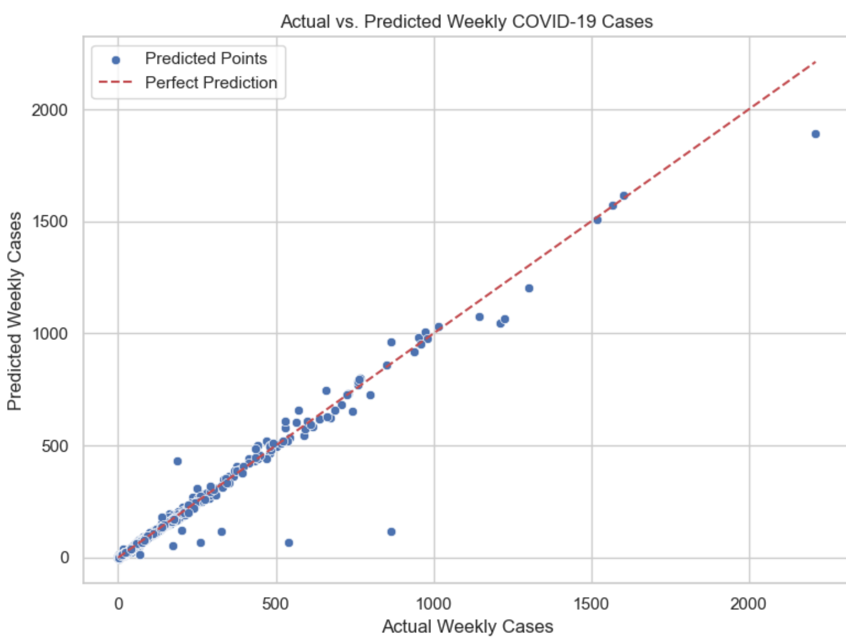


Figure 10: Actual vs Predicted weekly COVID 19 cases over time

- This scatter plot compares the actual weekly COVID-19 cases with the predicted cases generated by a model. Each dot represents a weekly data point, showing how closely the predicted values align with the actual observed values. The red dashed line represents the line of perfect prediction

Most of the data points are **closely** clustered around the red line, indicating that your model is performing quite well in predicting weekly cases. The linear alignment suggests a strong positive correlation between the predicted and actual values.

Model	MSE	R^2 Score(%)
Initial Gradient Boosting	541.12	96.84
Tuned Gradient Boosting	476.46	97.22

The tuned model showed a noticeable performance improvement, reducing the Mean Squared Error (MSE) and increasing the R^2 score. This suggests that hyperparameter optimization significantly enhanced the model's ability to generalize to unseen data.

Insights and Considerations

- The high R^2 values (>96%) confirm that the models explain most of the variance in weekly cases.
- Minor errors during rapid increases or decreases could be due to sudden real-world changes (e.g., new variants, policy changes) not captured in the features.
- Further improvements could involve adding external variables such as vaccination rates, mobility data, or policy interventions.

The Gradient Boosting Regressor, particularly after fine-tuning its parameters, showed impressive predictive capability in tracking weekly COVID-19 cases. It achieved a low Mean Squared Error and a high R^2 score of over 97%, demonstrating its strength and reliability. A scatter plot reveals that the predicted values closely match the actual case numbers, confirming the model's effectiveness. While there are some discrepancies during periods of sudden case spikes, which are to be expected given the unpredictable nature of a pandemic, these are manageable. Adding more external factors, such as mobility trends, vaccination rates, or policy changes, could improve the model’s ability to predict these sudden variations. In conclusion, the optimized Gradient Boosting model serves as a strong tool for accurate and timely forecasting of COVID-19 trends.

Conclusion

In this analysis, we compared the performance of four machine learning models to predict weekly COVID-19 cases: Deep Neural Network (DNN), Decision Tree Regression, Random Forest Regression, and Gradient Boosting Regression.

Deep Neural Network (DNN): The DNN model demonstrated exceptional predictive accuracy, achieving an R^2 score of 0.9960, indicating that it explained 99.6% of the variance in the target variable. The model performed with a Test Huber Loss of 0.0017 and a Mean Absolute Error (MAE) of 567.5 actual cases. These results highlight the ability of DNN to capture complex, non-linear relationships in the data, making it highly effective for predicting weekly COVID-19 cases. Despite the complexity and computational requirements, the performance of DNN makes it the best-suited model for this task.

Decision Tree Regression: The Decision Tree Regressor achieved an R^2 score of 96.06%, explaining over 96% of the variance in the target variable. Its MAE was 4.1368, and performance improved after hyperparameter optimization. While the model effectively captured trends in the data, it could not match the DNN in terms of overall predictive accuracy.

Random Forest Regression: The Random Forest model performed well with an R^2 score of 96% and an MAE of 6.28. This ensemble model demonstrated strength in handling complex, non-linear relationships and was resilient to overfitting. However, it was outperformed by the Gradient Boosting and DNN models in predictive accuracy.

Gradient Boosting Regression: The Gradient Boosting Regressor showed strong results with an R^2 score of 97.22% and a reduced Mean Squared Error (MSE) of 476.46 after hyperparameter optimization. While it captured the temporal and non-linear patterns in the data effectively, the DNN model still provided superior accuracy in predicting weekly COVID-19 cases.

Best Performing Model: Deep Neural Network (DNN)

The Deep Neural Network (DNN) emerged as the best-performing model in this analysis, achieving the highest R^2 score of 0.9960. With a Test Huber Loss of 0.0017 and an MAE of 567.5 actual cases, the DNN exhibited robust performance, demonstrating its capacity to model complex, non-linear relationships in the data effectively. Despite the computational cost and complexity associated with training the model, its predictive accuracy and reliability make it the optimal choice for forecasting COVID-19 trends.

Final Remarks

The performance of the machine learning models in this analysis underscores the potential of these algorithms to forecast COVID-19 case trends. While the DNN provided the most accurate predictions, there are opportunities for further enhancement. Incorporating external data, such as vaccination rates, mobility trends, or policy interventions, could help refine the models, particularly during periods of rapid change or sudden spikes in cases. The DNN model, in particular, offers a solid foundation for future improvements in pandemic forecasting.

Appendix: - python codes

Deep neural network

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
import geopandas as gpd
import folium
import matplotlib.pyplot as plt
import seaborn as sns

# Load CSV
df = pd.read_csv('C:/Users/hasal/Desktop/final_project/COVID-
19_Cases__Tests__and_Deaths_by_ZIP_Code_-_Historical.csv')

# Clean data first before processing
# ==================================

# Remove ZIP Codes marked as "Unknown"
df = df[df['ZIP Code'] != "Unknown"]

# Extract coordinates first
# ==========================
def extract_coordinates(coord_str):
    try:
        if pd.isna(coord_str):
            return (np.nan, np.nan)
        lon = float(coord_str.split()[1][1:])
        lat = float(coord_str.split()[2][:-1])
        return (lon, lat)
    except:
        return (np.nan, np.nan)

# Apply coordinate extraction
df['Longitude'], df['Latitude'] = zip(*df['ZIP Code
Location'].apply(extract_coordinates))

# Remove rows with missing coordinates
df = df.dropna(subset=['Longitude', 'Latitude'])

# Convert dates before filtering
# ===============================
df["Week Start"] = pd.to_datetime(df["Week Start"])
df["Week End"] = pd.to_datetime(df["Week End"])

# Cluster regions
```

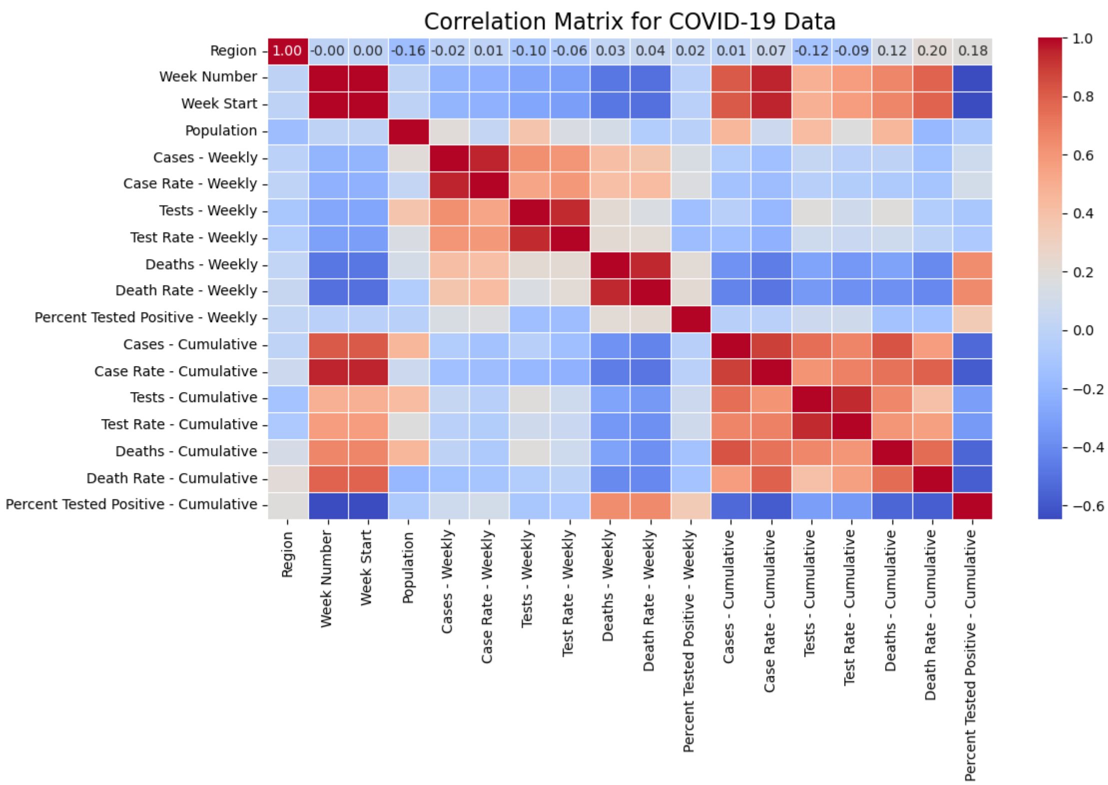
```
# ================
# Scale coordinates first
coords = df[['Longitude', 'Latitude']].values
kmeans = KMeans(n_clusters=7, random_state=42, n_init=10)
df['Region'] = kmeans.fit_predict(coords)

# Aggregate data by Region and Week Start
# ========================================
agg_dict = {
    "Cases - Weekly": "sum",
    "Cases - Cumulative": "sum",
    "Tests - Weekly": "sum",
    "Tests - Cumulative": "sum",
    "Deaths - Weekly": "sum",
    "Deaths - Cumulative": "sum",
    "Population": "sum"  # CORRECTED: Changed from mean to sum
}

df_time_series_region = df.groupby(['Region', 'Week
Start']).agg(agg_dict).reset_index()
# Calculate positivity rates FIRST
# =================================
# Handle zero-test cases
df_time_series_region['Percent Tested Positive - Weekly'] = np.where(
    df_time_series_region['Tests - Weekly'] > 0,
    (df_time_series_region['Cases - Weekly'] / df_time_series_region['Tests -
Weekly']) * 100,
    0
)

df_time_series_region['Percent Tested Positive - Cumulative'] = np.where(
    df_time_series_region['Tests - Cumulative'] > 0,
    (df_time_series_region['Cases - Cumulative'] /
df_time_series_region['Tests - Cumulative']) * 100,
    0
)
# Calculate rates per 100,000
# ============================
for time_type in ['Weekly', 'Cumulative']:
    df_time_series_region[f'Case Rate - {time_type}'] = (
        df_time_series_region[f'Cases - {time_type}'] /
        df_time_series_region['Population']
    ) * 100000

    df_time_series_region[f'Test Rate - {time_type}'] = (
        df_time_series_region[f'Tests - {time_type}'] /
        df_time_series_region['Population']
    ) * 100000
```

```
    df_time_series_region[f'Death Rate - {time_type}'] = (
        df_time_series_region[f'Deaths - {time_type}'] /
        df_time_series_region['Population']
    ) * 100000

# Integer columns (no decimals)
integer_columns = [
    "Cases - Weekly", "Cases - Cumulative",
    "Tests - Weekly", "Tests - Cumulative",
    "Deaths - Weekly", "Deaths - Cumulative",
    "Population"
]

# Decimal columns (round to 3 places)
decimal_columns = [
    "Percent Tested Positive - Weekly",
    "Percent Tested Positive - Cumulative",
    "Case Rate - Weekly",
    "Case Rate - Cumulative",
    "Test Rate - Weekly",
    "Test Rate - Cumulative",
    "Death Rate - Weekly",
    "Death Rate - Cumulative"
]

# Apply rounding
df_time_series_region[integer_columns] =
df_time_series_region[integer_columns].astype(int)
df_time_series_region[decimal_columns] =
df_time_series_region[decimal_columns].round(3)

# Add region-specific week numbers
# ==================================
df_time_series_region = df_time_series_region.sort_values(['Region', 'Week
Start'])
df_time_series_region['Week Number'] =
df_time_series_region.groupby('Region').cumcount() + 1

# Final column ordering
# =====================
column_order = [
    'Region', 'Week Number', 'Week Start', 'Population',
    'Cases - Weekly', 'Case Rate - Weekly',
    'Tests - Weekly', 'Test Rate - Weekly',
    'Deaths - Weekly', 'Death Rate - Weekly',
    'Percent Tested Positive - Weekly',
    'Cases - Cumulative', 'Case Rate - Cumulative',
    'Tests - Cumulative', 'Test Rate - Cumulative',
    'Deaths - Cumulative', 'Death Rate - Cumulative',
```

```python
    'Percent Tested Positive - Cumulative'
]

df_time_series_region = df_time_series_region[column_order]

# Final validation check
print("\nFinal validation:")
print("Max Weekly Case Rate:", df_time_series_region['Case Rate -
Weekly'].max())
print("Typical Chicago rates should be < 2000 in peak weeks")

# Step 1: Calculate the correlation matrix
correlation_matrix = df_time_series_region.corr()

# Step 2: Extract correlation with target 'Cases - Weekly'
correlation_with_target = correlation_matrix['Cases -
Weekly'].sort_values(ascending=False)

# Step 3: Print correlation with 'Cases - Weekly'
print("\nCorrelation with 'Cases - Weekly':")
print(correlation_with_target)

# Step 4: Plot the heatmap of the correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',
linewidths=0.5)
plt.title("Correlation Matrix for COVID-19 Data", fontsize=16)
plt.tight_layout()
plt.show()

# Display the final dataset
df_time_series_region.head()
```

```
Number of rows with missing values: 0

Final validation:
Max Weekly Case Rate: 1978.832
Typical Chicago rates should be < 2000 in peak weeks

Correlation with 'Cases - Weekly':
Cases - Weekly                         1.000000
Case Rate - Weekly                     0.952873
Tests - Weekly                         0.629610
Test Rate - Weekly                     0.605444
Deaths - Weekly                        0.417392
Death Rate - Weekly                    0.378141
Population                             0.196177
Percent Tested Positive - Weekly       0.138738
Percent Tested Positive - Cumulative   0.081098
```

```
Tests - Cumulative                         0.041052
Deaths - Cumulative                        0.003071
Region                                    -0.015782
Test Rate - Cumulative                    -0.016863
Cases - Cumulative                        -0.054739
Death Rate - Cumulative                   -0.132507
Case Rate - Cumulative                    -0.143272
Week Number                               -0.203768
Week Start                                -0.204698
Name: Cases - Weekly, dtype: float64
```

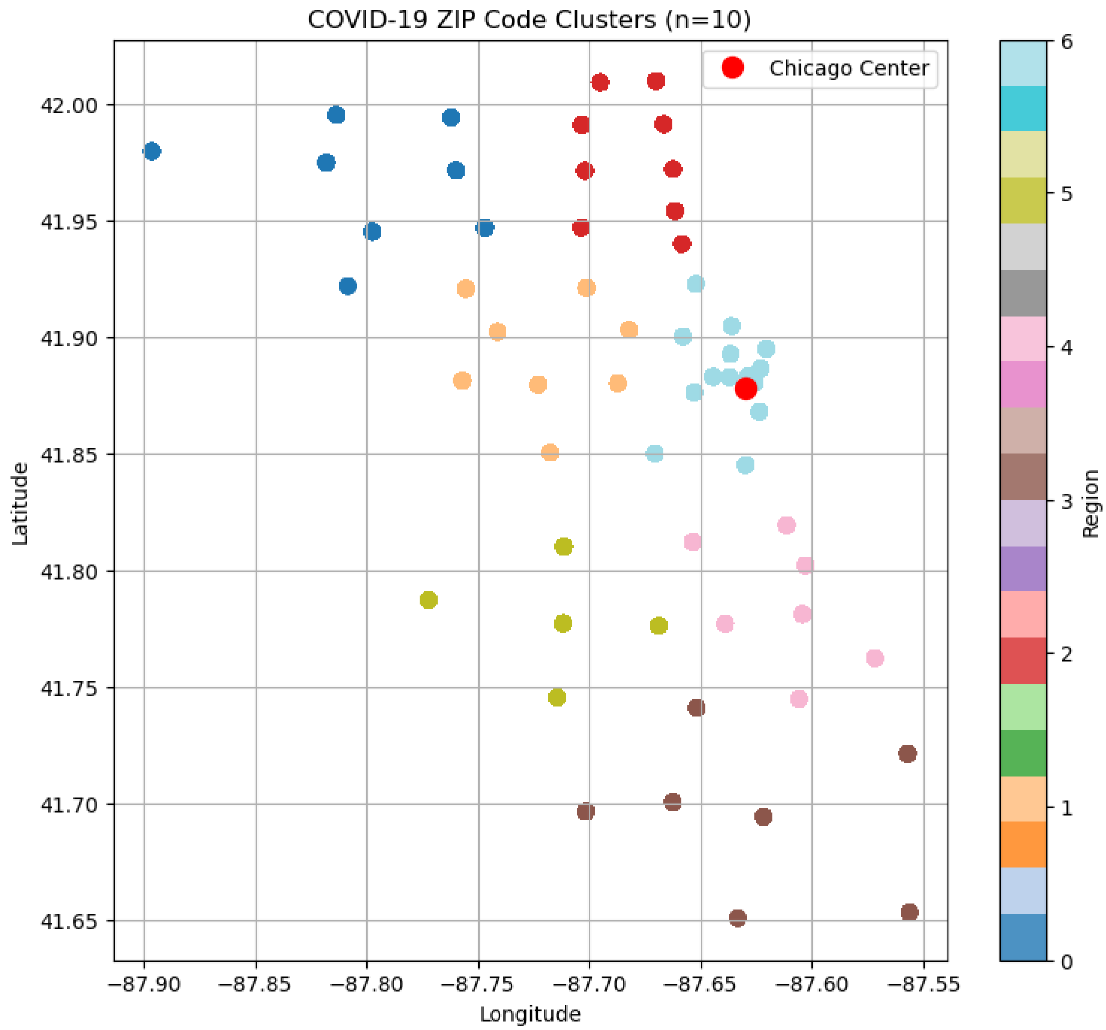



```
   Region  Week Number Week Start  Population  Cases - Weekly  \
0       0            1 2020-03-01      490874               0
1       0            2 2020-03-08      490874              24
2       0            3 2020-03-15      490874             185
3       0            4 2020-03-22      490874             401
4       0            5 2020-03-29      490874             373

   Case Rate - Weekly  Tests - Weekly  Test Rate - Weekly  Deaths - Weekly  \
0               0.000              17               3.463                0
1               4.889             122              24.854                0
2              37.688             950             193.532                1
3              81.691            1517             309.041                1
4              75.987            1287             262.185                9

   Death Rate - Weekly  Percent Tested Positive - Weekly  Cases - Cumulative
```

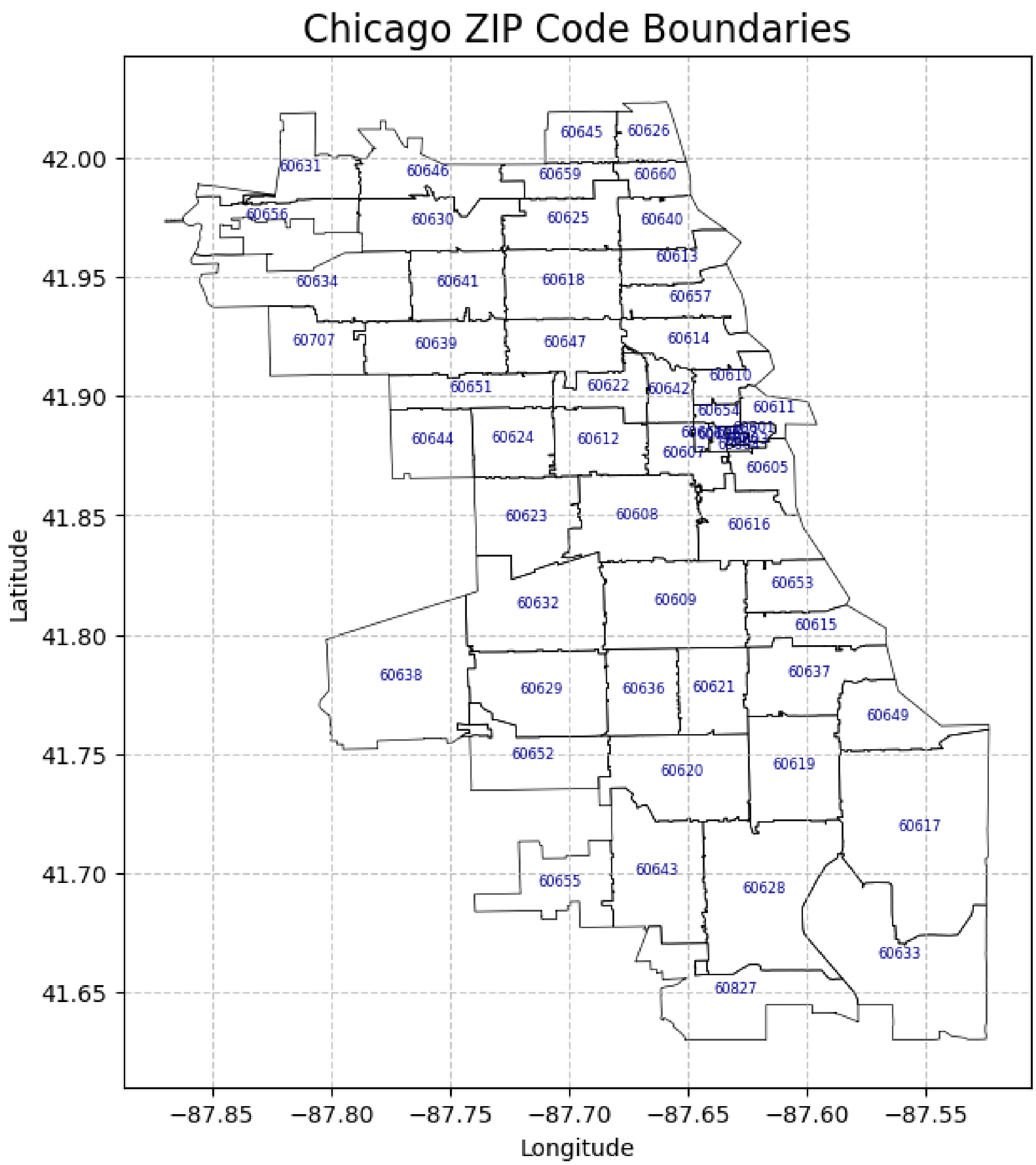
```
\
0              0.000                          0.000             0
1              0.000                         19.672            24
2              0.204                         19.474           228
3              0.204                         26.434           636
4              1.833                         28.982          1013

   Case Rate - Cumulative  Tests - Cumulative  Test Rate - Cumulative  \
0                   0.000                  17                   3.463
1                   4.889                 139                  28.317
2                  46.448                1089                 221.849
3                 129.565                2606                 530.890
4                 206.367                3893                 793.075

   Deaths - Cumulative  Death Rate - Cumulative  \
0                    0                    0.000
1                    0                    0.000
2                    1                    0.204
3                    2                    0.407
4                   11                    2.241

   Percent Tested Positive - Cumulative
0                                 0.000
1                                17.266
2                                20.937
3                                24.405
4                                26.021
```

```python
# 1. Generate Cluster Scatter Plot
# ================================
plt.figure(figsize=(9, 8))
scatter = plt.scatter(df['Longitude'], df['Latitude'], c=df['Region'],
                     cmap='tab20', s=50, alpha=0.8)
plt.colorbar(scatter, label='Region')
plt.title('COVID-19 ZIP Code Clusters (n=10)')
plt.xlabel('Longitude')
plt.ylabel('Latitude')

# Add Chicago reference points
plt.plot(-87.6298, 41.8781, 'ro', markersize=10, label='Chicago Center')  # Chicago coordinates
plt.legend()
plt.grid(True)
plt.show()


# Get unique ZIP codes from the dataset
chicago_zips = df["ZIP Code"].unique().astype(str)
```

```python
# Load Chicago ZIP code boundaries
geojson_url = "https://raw.githubusercontent.com/OpenDataDE/State-zip-code-GeoJSON/master/il_illinois_zip_codes_geo.min.json"
geojson_data = gpd.read_file(geojson_url)

# Filter to only Chicago ZIP codes from the dataset
geojson_data["ZCTA5CE10"] = geojson_data["ZCTA5CE10"].astype(str)
chicago_map = geojson_data[geojson_data["ZCTA5CE10"].isin(chicago_zips)]

# Create blank map with boundaries
fig, ax = plt.subplots(figsize=(12, 8))
chicago_map.plot(ax=ax,
                facecolor="none",  # No fill
                edgecolor="black",  # Boundary color
                linewidth=0.5,     # Boundary line thickness
                aspect="equal")

# Add context
ax.set_title("Chicago ZIP Code Boundaries", fontsize=16)
ax.set_xlabel("Longitude")
ax.set_ylabel("Latitude")

# Add grid
ax.grid(True, linestyle='--', alpha=0.7)

# Optional: Add text labels for ZIP codes
for idx, row in chicago_map.iterrows():
    ax.text(row.geometry.centroid.x,
           row.geometry.centroid.y,
           row.ZCTA5CE10,
           fontsize=6,
           ha='center',
           color='darkblue')

plt.show()
```



COVID-19 ZIP Code Clusters (n=10)
Chicago Center
Latitude
Longitude
Region
42.00
41.95
41.90
41.85
41.80
41.75
41.70
41.65
−87.90
−87.85
−87.80
−87.75
−87.70
−87.65
−87.60
−87.55
0
1
2
3
4
5
6




```python
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Load preprocessed regional data
df_time_series = df_time_series_region

# Convert dates and sort
df_time_series["Week Start"] = pd.to_datetime(df_time_series["Week Start"])
```

```python
df_time_series = df_time_series.sort_values(['Region', 'Week Start'])

# Feature Engineering - Lag Features for Test Prediction
# -----------------------------------------------------
# Create region-specific lag features
df_time_series['Tests - Weekly Lag1'] =
df_time_series.groupby('Region')['Tests - Weekly'].shift(1)
df_time_series['Test Rate - Weekly Lag1'] =
df_time_series.groupby('Region')['Test Rate - Weekly'].shift(1)
df_time_series['Case Rate - Weekly Lag1'] =
df_time_series.groupby('Region')['Case Rate - Weekly'].shift(1)



# Drop rows with missing values from lag features
df_time_series = df_time_series.dropna()
# New target
target = 'Cases - Weekly'

# Select features correlated with the new target
features = [
    'Week Number',                  # This can still be useful for trends
over time
    'Case Rate - Weekly',           # Strongly correlated with 'Cases -
Weekly'
    'Tests - Weekly',               # Moderately correlated with 'Cases -
Weekly'
    'Test Rate - Weekly',           # Moderately correlated with 'Cases -
Weekly'
    'Deaths - Weekly',              # Useful feature for predicting cases
    'Death Rate - Weekly'           # Another relevant feature
]

# Prepare data
X = df_time_series[features].values
y = df_time_series[target].values.reshape(-1, 1)

# Standardization
scaler_X = StandardScaler()
scaler_y = StandardScaler()

X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y)

# Temporal Train-Test Split (No Shuffling)
test_size = int(len(X_scaled) * 0.2)
X_train, X_test = X_scaled[:-test_size], X_scaled[-test_size:]
y_train, y_test = y_scaled[:-test_size], y_scaled[-test_size:]
```

```python
# Get corresponding dates for the test set
test_dates = df_time_series['Week Start'].iloc[-
test_size:].reset_index(drop=True)

# Enhanced DNN Architecture for Cases Prediction
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu',
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(1)
])

# Custom Huber Loss Configuration
huber_delta = 1.5  # Adjusted delta for case prediction
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss=tf.keras.losses.Huber(delta=huber_delta),
              metrics=['mae'])

# Early Stopping and Training
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=15,
    restore_best_weights=True
)

history = model.fit(
    X_train, y_train,
    epochs=200,
    batch_size=32,
    validation_data=(X_test, y_test),
    callbacks=[early_stopping],
    verbose=1
)

# Prediction and Inverse Scaling
y_pred_scaled = model.predict(X_test)
y_pred = scaler_y.inverse_transform(y_pred_scaled)
y_test_actual = scaler_y.inverse_transform(y_test)

# Plot actual vs predicted cases
plt.figure(figsize=(14, 7))
plt.plot(test_dates, y_test_actual, label='Actual Cases', marker='o',
linestyle='-', linewidth=2)
plt.plot(test_dates, y_pred, label='Predicted Cases', marker='x',
linestyle='--', linewidth=2)
plt.title('Regional COVID-19 Case Prediction', fontsize=16)
```

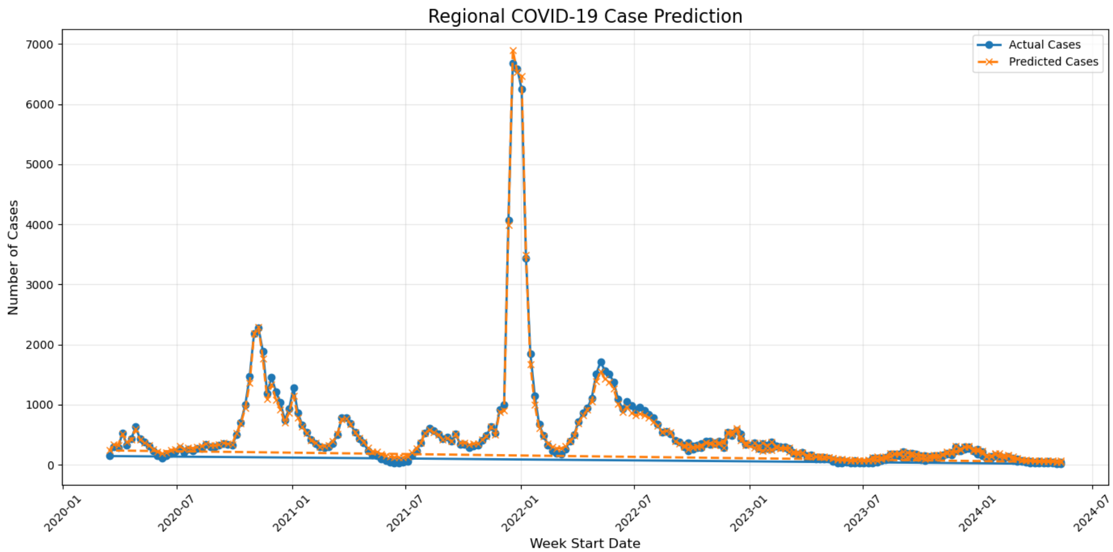
```
plt.xlabel('Week Start Date', fontsize=12)
plt.ylabel('Number of Cases', fontsize=12)
plt.legend()
plt.grid(True, alpha=0.3)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
C:\Users\hasal\AppData\Roaming\Python\Python311\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Epoch 1/200
38/38 ━━━━━━━━━━━━━━━━━━━━4s 20ms/step - loss: 0.1644 - mae: 0.3449 -
val_loss: 0.0094 - val_mae: 0.1106
Epoch 2/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0423 - mae: 0.1769 -
val_loss: 0.0055 - val_mae: 0.0789
Epoch 3/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 7ms/step - loss: 0.0283 - mae: 0.1458 -
val_loss: 0.0152 - val_mae: 0.1033
Epoch 4/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0254 - mae: 0.1359 -
val_loss: 0.0053 - val_mae: 0.0595
Epoch 5/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 7ms/step - loss: 0.0329 - mae: 0.1430 -
val_loss: 0.0062 - val_mae: 0.0647
Epoch 6/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0390 - mae: 0.1421 -
val_loss: 0.0153 - val_mae: 0.1310
Epoch 7/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0277 - mae: 0.1392 -
val_loss: 0.0090 - val_mae: 0.0740
Epoch 8/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 9ms/step - loss: 0.0229 - mae: 0.1127 -
val_loss: 0.0127 - val_mae: 0.0828
Epoch 9/200
38/38 ━━━━━━━━━━━━━━━━━━━━1s 10ms/step - loss: 0.0288 - mae: 0.1193 -
val_loss: 0.0057 - val_mae: 0.0777
Epoch 10/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0189 - mae: 0.1114 -
val_loss: 0.0044 - val_mae: 0.0498
Epoch 11/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 9ms/step - loss: 0.0185 - mae: 0.1009 -
val_loss: 0.0145 - val_mae: 0.1112
Epoch 12/200
```

```
38/38 ━━━━━━━━━━━━━━━━━━━━0s 9ms/step - loss: 0.0188 - mae: 0.1036 -
val_loss: 0.0038 - val_mae: 0.0476
Epoch 13/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0207 - mae: 0.1032 -
val_loss: 0.0115 - val_mae: 0.0822
Epoch 14/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 9ms/step - loss: 0.0130 - mae: 0.0944 -
val_loss: 0.0076 - val_mae: 0.0577
Epoch 15/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 9ms/step - loss: 0.0140 - mae: 0.0973 -
val_loss: 0.0055 - val_mae: 0.0514
Epoch 16/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 9ms/step - loss: 0.0157 - mae: 0.0932 -
val_loss: 0.0072 - val_mae: 0.0550
Epoch 17/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 9ms/step - loss: 0.0271 - mae: 0.1024 -
val_loss: 0.0020 - val_mae: 0.0461
Epoch 18/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0225 - mae: 0.0980 -
val_loss: 0.0058 - val_mae: 0.0765
Epoch 19/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 9ms/step - loss: 0.0169 - mae: 0.0919 -
val_loss: 0.0044 - val_mae: 0.0687
Epoch 20/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 10ms/step - loss: 0.0177 - mae: 0.0905 -
val_loss: 0.0153 - val_mae: 0.1070
Epoch 21/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 10ms/step - loss: 0.0214 - mae: 0.0949 -
val_loss: 0.0109 - val_mae: 0.0928
Epoch 22/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0121 - mae: 0.0874 -
val_loss: 0.0365 - val_mae: 0.1260
Epoch 23/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 11ms/step - loss: 0.0176 - mae: 0.0932 -
val_loss: 0.0068 - val_mae: 0.0901
Epoch 24/200
38/38 ━━━━━━━━━━━━━━━━━━━━1s 12ms/step - loss: 0.0148 - mae: 0.0850 -
val_loss: 0.0038 - val_mae: 0.0483
Epoch 25/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0126 - mae: 0.0915 -
val_loss: 0.0056 - val_mae: 0.0856
Epoch 26/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0160 - mae: 0.0875 -
val_loss: 0.0023 - val_mae: 0.0464
Epoch 27/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0162 - mae: 0.0947 -
val_loss: 0.0035 - val_mae: 0.0473
Epoch 28/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 11ms/step - loss: 0.0170 - mae: 0.0892 -
val_loss: 0.0076 - val_mae: 0.0779
```

```
Epoch 29/200
38/38 ━━━━━━━━━━━━━━━━━━━━1s 13ms/step - loss: 0.0226 - mae: 0.0955 -
val_loss: 0.0017 - val_mae: 0.0465
Epoch 30/200
38/38 ━━━━━━━━━━━━━━━━━━━━1s 12ms/step - loss: 0.0183 - mae: 0.0914 -
val_loss: 0.0023 - val_mae: 0.0499
Epoch 31/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 11ms/step - loss: 0.0110 - mae: 0.0834 -
val_loss: 0.0075 - val_mae: 0.0676
Epoch 32/200
38/38 ━━━━━━━━━━━━━━━━━━━━1s 12ms/step - loss: 0.0121 - mae: 0.0782 -
val_loss: 0.0029 - val_mae: 0.0621
Epoch 33/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 10ms/step - loss: 0.0082 - mae: 0.0723 -
val_loss: 0.0021 - val_mae: 0.0508
Epoch 34/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 9ms/step - loss: 0.0328 - mae: 0.1133 -
val_loss: 0.0022 - val_mae: 0.0440
Epoch 35/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 9ms/step - loss: 0.0213 - mae: 0.0854 -
val_loss: 0.0243 - val_mae: 0.1094
Epoch 36/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0217 - mae: 0.0900 -
val_loss: 0.0125 - val_mae: 0.0874
Epoch 37/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 9ms/step - loss: 0.0153 - mae: 0.0837 -
val_loss: 0.0060 - val_mae: 0.0703
Epoch 38/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0101 - mae: 0.0751 -
val_loss: 0.0238 - val_mae: 0.0934
Epoch 39/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 7ms/step - loss: 0.0137 - mae: 0.0828 -
val_loss: 0.0148 - val_mae: 0.0903
Epoch 40/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 7ms/step - loss: 0.0101 - mae: 0.0754 -
val_loss: 0.0082 - val_mae: 0.0760
Epoch 41/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 9ms/step - loss: 0.0111 - mae: 0.0785 -
val_loss: 0.0017 - val_mae: 0.0414
Epoch 42/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 7ms/step - loss: 0.0076 - mae: 0.0714 -
val_loss: 0.0135 - val_mae: 0.0955
Epoch 43/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 7ms/step - loss: 0.0136 - mae: 0.0833 -
val_loss: 0.0028 - val_mae: 0.0608
Epoch 44/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 9ms/step - loss: 0.0207 - mae: 0.0821 -
val_loss: 0.0069 - val_mae: 0.0707
Epoch 45/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0100 - mae: 0.0691 -
```

```
val_loss: 0.0075 - val_mae: 0.0820
Epoch 46/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0205 - mae: 0.0877 -
val_loss: 0.0027 - val_mae: 0.0572
Epoch 47/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 7ms/step - loss: 0.0211 - mae: 0.0860 -
val_loss: 0.0106 - val_mae: 0.0806
Epoch 48/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0099 - mae: 0.0771 -
val_loss: 0.0046 - val_mae: 0.0580
Epoch 49/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 9ms/step - loss: 0.0121 - mae: 0.0686 -
val_loss: 0.0088 - val_mae: 0.0771
Epoch 50/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0057 - mae: 0.0630 -
val_loss: 0.0041 - val_mae: 0.0490
Epoch 51/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 8ms/step - loss: 0.0344 - mae: 0.1013 -
val_loss: 0.0221 - val_mae: 0.1137
Epoch 52/200
38/38 ━━━━━━━━━━━━━━━━━━━━1s 10ms/step - loss: 0.0266 - mae: 0.0959 -
val_loss: 0.0096 - val_mae: 0.0807
Epoch 53/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 9ms/step - loss: 0.0147 - mae: 0.0764 -
val_loss: 0.0060 - val_mae: 0.0721
Epoch 54/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 9ms/step - loss: 0.0154 - mae: 0.0809 -
val_loss: 0.0061 - val_mae: 0.0704
Epoch 55/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 10ms/step - loss: 0.0125 - mae: 0.0771 -
val_loss: 0.0026 - val_mae: 0.0605
Epoch 56/200
38/38 ━━━━━━━━━━━━━━━━━━━━0s 7ms/step - loss: 0.0213 - mae: 0.0866 -
val_loss: 0.0062 - val_mae: 0.0765
10/10 ━━━━━━━━━━━━━━━━━━━━0s 12ms/step
```




```python
# Model Performance Report
test_loss, test_mae = model.evaluate(X_test, y_test, verbose=0)
print(f"\nModel Performance:")
print(f"- Test Huber Loss: {test_loss:.4f}")
print(f"- MAE: {test_mae:.4f} (Scaled),
{scaler_y.inverse_transform([[test_mae]])[0][0]:.1f} (Actual Cases)")
```

```
Model Performance:
- Test Huber Loss: 0.0017
- MAE: 0.0414 (Scaled), 567.5 (Actual Cases)
```

Random forest regressor

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import LabelEncoder


# Load the CSV file
file_path = 'C:\\Users\\Dell\\Desktop\\Machine learnng\\Final proj\\archive
(1)\\COVID-19_Cases__Tests__and_Deaths_by_ZIP_Code_-_Historical.csv'
df = pd.read_csv(file_path)

# Display the first 15 rows
df.head(15)
```

```
   ZIP Code  Week Number  Week Start    Week End  Cases - Weekly  \
0     60622           31  07/26/2020  08/01/2020            28.0
1     60622           32  08/02/2020  08/08/2020            34.0
2     60622           33  08/09/2020  08/15/2020            41.0
3     60622           34  08/16/2020  08/22/2020            42.0
4     60622           35  08/23/2020  08/29/2020            45.0
5     60622           36  08/30/2020  09/05/2020            29.0
6     60622           37  09/06/2020  09/12/2020            46.0
7     60622           39  09/20/2020  09/26/2020            63.0
8     60622           40  09/27/2020  10/03/2020            45.0
9     60622           43  10/18/2020  10/24/2020           166.0
10    60622           44  10/25/2020  10/31/2020           196.0
11    60622           45  11/01/2020  11/07/2020           341.0
12    60622           46  11/08/2020  11/14/2020           310.0
13    60622           47  11/15/2020  11/21/2020           230.0
14    60622           48  11/22/2020  11/28/2020           174.0

    Cases - Cumulative  Case Rate - Weekly  Case Rate - Cumulative  \
0                877.0                53.0                  1661.2
1                911.0                64.0                  1725.6
2                952.0                78.0                  1803.3
3                994.0                80.0                  1882.8
4               1039.0                85.0                  1968.1
5               1068.0                55.0                  2023.0
6               1114.0                87.0                  2110.1
7               1217.0               119.0                  2305.2
8               1262.0                85.0                  2390.5
```

```
9              1617.0              314.0                 3062.9
10             1813.0              371.0                 3434.2
11             2154.0              646.0                 4080.1
12             2464.0              587.0                 4667.3
13             2694.0              436.0                 5102.9
14             2868.0              330.0                 5432.5

    Tests - Weekly  Tests - Cumulative  ...  Test Rate - Cumulative  \
0           1329.0               13148  ...                 24904.8
1           1405.0               14553  ...                 27566.2
2           1542.0               16095  ...                 30487.0
3           1674.0               17769  ...                 33657.9
4           1540.0               19309  ...                 36574.9
5           1547.0               20856  ...                 39505.2
6           1400.0               22256  ...                 42157.1
7           1844.0               25763  ...                 48800.0
8           1705.0               27468  ...                 52029.6
9           2642.0               34626  ...                 65588.2
10          2717.0               37343  ...                 70734.8
11          3067.0               40410  ...                 76544.2
12          3064.0               43474  ...                 82348.0
13          3836.0               47310  ...                 89614.2
14          2761.0               50071  ...                 94844.0

    Percent Tested Positive - Weekly  Percent Tested Positive - Cumulative  \
0                                0.0                                   0.1
1                                0.0                                   0.1
2                                0.0                                   0.1
3                                0.0                                   0.1
4                                0.0                                   0.1
5                                0.0                                   0.1
6                                0.0                                   0.1
7                                0.0                                   0.0
8                                0.0                                   0.0
9                                0.1                                   0.0
10                               0.1                                   0.0
11                               0.1                                   0.1
12                               0.1                                   0.1
13                               0.1                                   0.1
14                               0.1                                   0.1

    Deaths - Weekly  Deaths - Cumulative  Death Rate - Weekly  \
0                 0                   56                  0.0
1                 0                   56                  0.0
2                 0                   56                  0.0
3                 0                   56                  0.0
4                 0                   56                  0.0
5                 0                   56                  0.0
6                 0                   56                  0.0
7                 0                   56                  0.0
```

```
8                 0                   56                   0.0
9                 0                   56                   0.0
10                0                   56                   0.0
11                1                   57                   1.9
12                1                   58                   1.9
13                1                   59                   1.9
14                2                   61                   3.8

    Death Rate - Cumulative  Population        Row ID  \
0                     106.1       52793  60622-2020-31
1                     106.1       52793  60622-2020-32
2                     106.1       52793  60622-2020-33
3                     106.1       52793  60622-2020-34
4                     106.1       52793  60622-2020-35
5                     106.1       52793  60622-2020-36
6                     106.1       52793  60622-2020-37
7                     106.1       52793  60622-2020-39
8                     106.1       52793  60622-2020-40
9                     106.1       52793  60622-2020-43
10                    106.1       52793  60622-2020-44
11                    108.0       52793  60622-2020-45
12                    109.9       52793  60622-2020-46
13                    111.8       52793  60622-2020-47
14                    115.5       52793  60622-2020-48

               ZIP Code Location
0   POINT (-87.681818 41.902762)
1   POINT (-87.681818 41.902762)
2   POINT (-87.681818 41.902762)
3   POINT (-87.681818 41.902762)
4   POINT (-87.681818 41.902762)
5   POINT (-87.681818 41.902762)
6   POINT (-87.681818 41.902762)
7   POINT (-87.681818 41.902762)
8   POINT (-87.681818 41.902762)
9   POINT (-87.681818 41.902762)
10  POINT (-87.681818 41.902762)
11  POINT (-87.681818 41.902762)
12  POINT (-87.681818 41.902762)
13  POINT (-87.681818 41.902762)
14  POINT (-87.681818 41.902762)

[15 rows x 21 columns]
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13132 entries, 0 to 13131
Data columns (total 21 columns):
 #   Column                                Non-Null Count  Dtype
```

```
---  ------                              --------------  -----
 0   ZIP Code                            13132 non-null  object
 1   Week Number                         13132 non-null  int64
 2   Week Start                          13132 non-null  object
 3   Week End                            13132 non-null  object
 4   Cases - Weekly                      12909 non-null  float64
 5   Cases - Cumulative                  12909 non-null  float64
 6   Case Rate - Weekly                  12909 non-null  float64
 7   Case Rate - Cumulative              12909 non-null  float64
 8   Tests - Weekly                      12740 non-null  float64
 9   Tests - Cumulative                  13132 non-null  int64
 10  Test Rate - Weekly                  13132 non-null  int64
 11  Test Rate - Cumulative              13132 non-null  float64
 12  Percent Tested Positive - Weekly    13132 non-null  float64
 13  Percent Tested Positive - Cumulative  13132 non-null  float64
 14  Deaths - Weekly                     13132 non-null  int64
 15  Deaths - Cumulative                 13132 non-null  int64
 16  Death Rate - Weekly                 13132 non-null  float64
 17  Death Rate - Cumulative             13132 non-null  float64
 18  Population                          13132 non-null  int64
 19  Row ID                              13132 non-null  object
 20  ZIP Code Location                   12921 non-null  object
dtypes: float64(10), int64(6), object(5)
memory usage: 2.1+ MB
```

```
df.describe()
```

```
        Week Number  Cases - Weekly  Cases - Cumulative  Case Rate - Weekly  \
count  13132.000000    12909.000000        12909.000000        12909.000000
mean      26.170119       63.458440         8344.924161          136.947401
std       14.871736      121.313518         7516.565007          245.224599
min        1.000000        0.000000            5.000000            0.000000
25%       13.000000       11.000000         1989.000000           32.000000
50%       25.000000       30.000000         6503.000000           76.000000
75%       39.000000       70.000000        12839.000000          150.000000
max       53.000000     2212.000000        36570.000000         6266.000000

       Case Rate - Cumulative  Tests - Weekly  Tests - Cumulative  \
count            12909.000000    12740.000000        13132.000000
mean             17734.813309     1225.955024       129983.026652
std              11955.509645     1400.608932       135184.120574
min                  0.000000        0.000000            0.000000
25%               7127.600000      158.000000        10100.500000
50%              19382.600000      835.500000        86097.000000
75%              27597.600000     1807.250000       223838.750000
max              64450.100000    13173.000000       538868.000000

       Test Rate - Weekly  Test Rate - Cumulative  \
count        13132.000000            1.313200e+04
```

```
mean         2677.341989          2.957631e+05
std          3240.396176          2.931501e+05
min             0.000000          0.000000e+00
25%           369.000000          2.742940e+04
50%          1946.000000          2.331007e+05
75%          3795.250000          4.949480e+05
max         75755.000000          2.037212e+06

       Percent Tested Positive - Weekly  Percent Tested Positive - Cumulative
\
count                      13132.000000                          13132.000000
mean                           0.056298                              0.074147
std                            0.078874                              0.064195
min                            0.000000                              0.000000
25%                            0.000000                              0.000000
50%                            0.000000                              0.100000
75%                            0.100000                              0.100000
max                            1.000000                              0.500000

       Deaths - Weekly  Deaths - Cumulative  Death Rate - Weekly  \
count     13132.000000         13132.000000         13132.000000
mean          0.636689           105.623896             1.218299
std           1.634849            91.039144             3.309388
min           0.000000             0.000000             0.000000
25%           0.000000            19.000000             0.000000
50%           0.000000            90.000000             0.000000
75%           1.000000           168.000000             1.200000
max          25.000000           365.000000            80.400000

       Death Rate - Cumulative     Population
count             13132.000000   13132.000000
mean                199.797525   46258.380064
std                 138.398733   26835.033756
min                   0.000000       0.000000
25%                  81.800000   28804.000000
50%                 192.900000   46024.000000
75%                 309.000000   68096.000000
max                 540.600000  111850.000000
```

```
df.dtypes
```

```
ZIP Code                                  object
Week Number                                int64
Week Start                                object
Week End                                  object
Cases - Weekly                           float64
Cases - Cumulative                       float64
Case Rate - Weekly                       float64
Case Rate - Cumulative                   float64
```

```
Tests - Weekly                            float64
Tests - Cumulative                          int64
Test Rate - Weekly                          int64
Test Rate - Cumulative                    float64
Percent Tested Positive - Weekly          float64
Percent Tested Positive - Cumulative      float64
Deaths - Weekly                             int64
Deaths - Cumulative                         int64
Death Rate - Weekly                       float64
Death Rate - Cumulative                   float64
Population                                  int64
Row ID                                     object
ZIP Code Location                          object
dtype: object
```

```
# Count the number of duplicate rows
num_duplicates = df.duplicated().sum()

print(f"Number of duplicate rows: {num_duplicates}")
```

```
Number of duplicate rows: 0
```

```
# Calculate missing values count and percentage
missing_count = df.isnull().sum()
missing_percentage = (missing_count / len(df)) * 100

# Combine into a DataFrame
missing_data = pd.DataFrame({
    'Missing Values': missing_count,
    'Percentage (%)': missing_percentage
})

# Filter only columns with missing values
missing_data = missing_data[missing_data['Missing Values'] > 0]

# Display the table
missing_data
```

	Missing Values	Percentage (%)
Cases - Weekly	223	1.698142
Cases - Cumulative	223	1.698142
Case Rate - Weekly	223	1.698142
Case Rate - Cumulative	223	1.698142
Tests - Weekly	392	2.985075
ZIP Code Location	211	1.606762

```
# STEP 1: Drop irrelevant columns if they exist
columns_to_drop = ['Row ID', 'ZIP Code Location']
df.drop(columns=[col for col in columns_to_drop if col in df.columns],
```

```python
inplace=True)

# STEP 2: Remove duplicate rows
df.drop_duplicates(inplace=True)

# STEP 3: Fill missing values with mode or mean
if 'ZIP Code' in df.columns:
    df['ZIP Code'].fillna(df['ZIP Code'].mode()[0], inplace=True)

for col in [
    'Cases - Weekly', 'Cases - Cumulative',
    'Case Rate - Weekly', 'Case Rate - Cumulative',
    'Tests - Weekly'
]:
    if col in df.columns:
        df[col].fillna(df[col].mean(), inplace=True)

# STEP 4: Convert date columns to datetime
for date_col in ['Week Start', 'Week End']:
    if date_col in df.columns:
        df[date_col] = pd.to_datetime(df[date_col], errors='coerce')

# STEP 5: Reset index after cleaning
df.reset_index(drop=True, inplace=True)

# STEP 6: Final check for missing values
df.isnull().sum()
```

```
ZIP Code                                0
Week Number                             0
Week Start                              0
Week End                                0
Cases - Weekly                          0
Cases - Cumulative                      0
Case Rate - Weekly                      0
Case Rate - Cumulative                  0
Tests - Weekly                          0
Tests - Cumulative                      0
Test Rate - Weekly                      0
Test Rate - Cumulative                  0
Percent Tested Positive - Weekly        0
Percent Tested Positive - Cumulative    0
Deaths - Weekly                         0
Deaths - Cumulative                     0
Death Rate - Weekly                     0
Death Rate - Cumulative                 0
Population                              0
dtype: int64
```

```
# Convert 'Week Start' and 'Week End' columns to datetime format (if they
exist)
date_columns = ['Week Start', 'Week End']

for col in date_columns:
    if col in df.columns:
        df[col] = pd.to_datetime(df[col], errors='coerce')  # Invalid dates
will become NaT

# Optional: Check the result
df[date_columns].dtypes
```

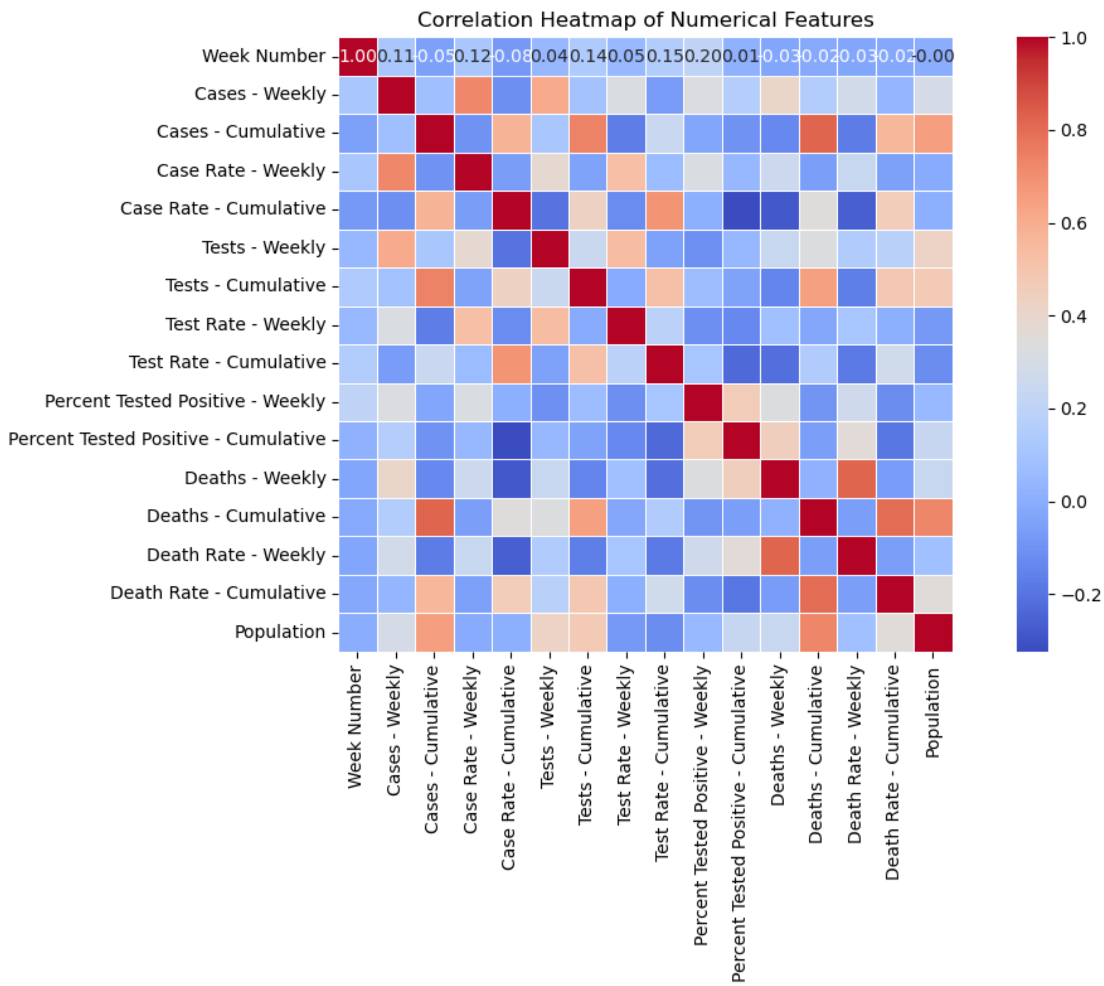
```
Week Start    datetime64[ns]
Week End      datetime64[ns]
dtype: object
```

```
# Identify categorical and numerical columns
categorical_columns = df.select_dtypes(include=['object',
'category']).columns.tolist()
numerical_columns = df.select_dtypes(include=['number']).columns.tolist()

# Print the results
print("Categorical columns:", categorical_columns)
print()
print("Numerical columns:", numerical_columns)
print()
# Identify datetime columns
datetime_columns = df.select_dtypes(include=['datetime64']).columns.tolist()

# Print them too
print("Datetime columns:", datetime_columns)
```

```
Categorical columns: ['ZIP Code']

Numerical columns: ['Week Number', 'Cases - Weekly', 'Cases - Cumulative',
'Case Rate - Weekly', 'Case Rate - Cumulative', 'Tests - Weekly', 'Tests -
Cumulative', 'Test Rate - Weekly', 'Test Rate - Cumulative', 'Percent Tested
Positive - Weekly', 'Percent Tested Positive - Cumulative', 'Deaths -
Weekly', 'Deaths - Cumulative', 'Death Rate - Weekly', 'Death Rate -
Cumulative', 'Population']

Datetime columns: ['Week Start', 'Week End']
```

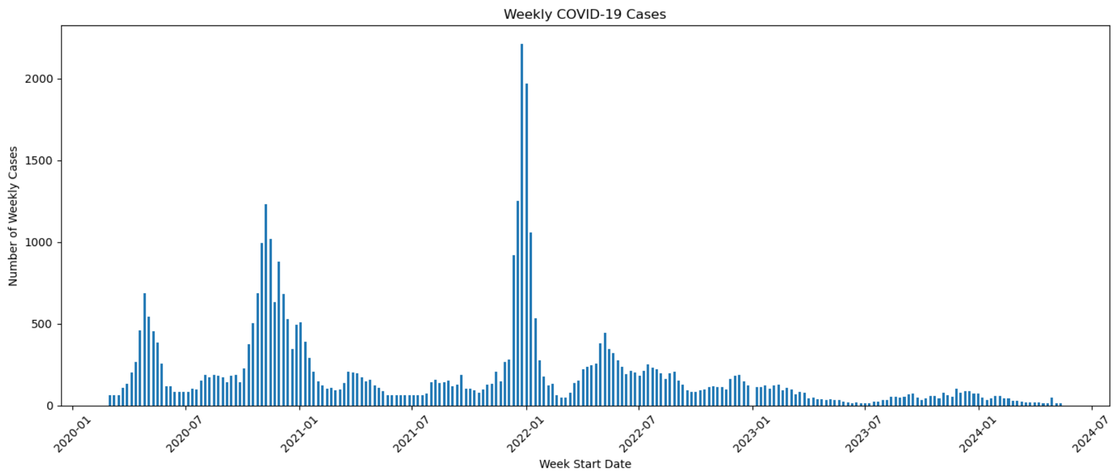
```
# Compute the correlation matrix for numerical features
corr_matrix = df.corr(numeric_only=True)

# Set up the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm", square=True,
linewidths=0.5)
```

```python
# Title and layout
plt.title("Correlation Heatmap of Numerical Features")
plt.tight_layout()
plt.show()
```



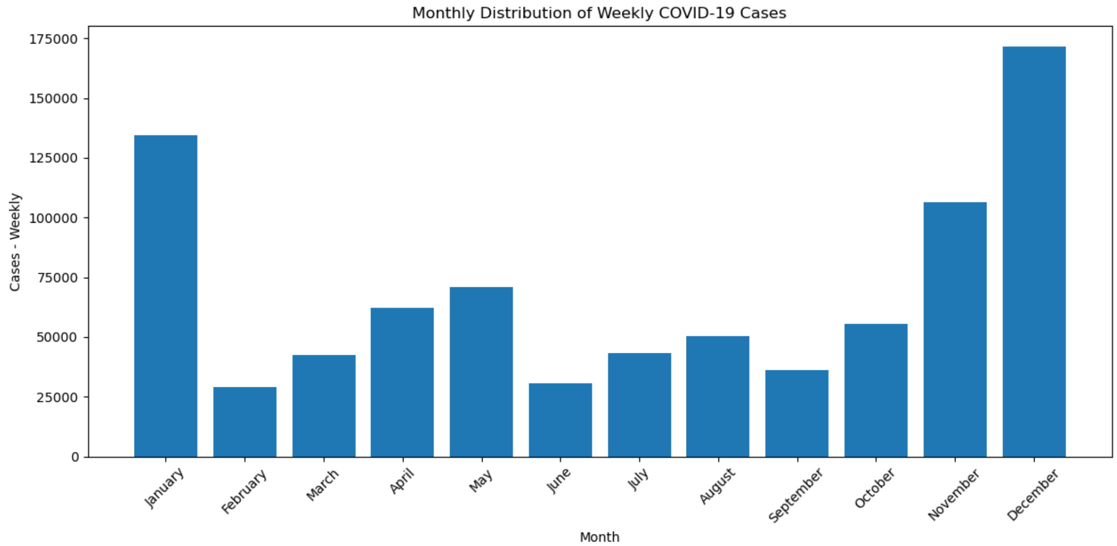

```python
# Check if 'Week Start' exists, otherwise use index
x = df['Week Start'] if 'Week Start' in df.columns else df.index
y = df['Cases - Weekly']

# Plot the bar graph
plt.figure(figsize=(14, 6))
plt.bar(x, y, width=4)  # width=4 works well for weekly data

# Add titles and labels
plt.title("Weekly COVID-19 Cases")
plt.xlabel("Week Start Date")
plt.ylabel("Number of Weekly Cases")
```

```python
# Improve x-axis readability if using dates
if 'Week Start' in df.columns:
    plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```




```python
import matplotlib.pyplot as plt
import pandas as pd

# Ensure 'Week Start' is in datetime format
df['Week Start'] = pd.to_datetime(df['Week Start'], errors='coerce')

# Create a 'Month' column with month names
df['Month'] = df['Week Start'].dt.month_name()

# Define proper month order for x-axis
month_order = [
    'January', 'February', 'March', 'April', 'May', 'June',
    'July', 'August', 'September', 'October', 'November', 'December'
]

# Group by Month and sum 'Cases - Weekly'
monthly_cases = df.groupby('Month')['Cases -
Weekly'].sum().reindex(month_order)

# Plot the bar graph
plt.figure(figsize=(12, 6))
plt.bar(monthly_cases.index, monthly_cases.values)

# Add title and labels
plt.title("Monthly Distribution of Weekly COVID-19 Cases")
plt.xlabel("Month")
plt.ylabel("Cases - Weekly")
```

```python
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




```python
from sklearn.preprocessing import LabelEncoder

# Step 1: Drop rows with missing target or features just in case
df.dropna(subset=['Cases - Weekly'], inplace=True)

# Step 2: Feature selection (drop target and unused date columns)
features = df.drop(columns=['Cases - Weekly', 'Week Start', 'Week End'])
target = df['Cases - Weekly']

# Step 3: Encode categorical columns if any
categorical_cols = features.select_dtypes(include=['object',
'category']).columns

label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    features[col] = le.fit_transform(features[col])
    label_encoders[col] = le

# Step 4: Train/test split
X_train, X_test, y_train, y_test = train_test_split(features, target,
test_size=0.2, random_state=42)

# Step 5: Train the model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```python
# Step 6: Make predictions and evaluate
y_pred = model.predict(X_test)

# Metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
r2_percent = r2 * 100  # Convert to percentage

# Print metrics
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R² Score: {r2:.2f}")
print(f"R² Score (%): {r2_percent:.2f}%")
```

```
Mean Squared Error (MSE): 543.35
Mean Absolute Error (MAE): 2.95
R² Score: 0.97
R² Score (%): 96.83%
```

```python
from sklearn.preprocessing import LabelEncoder

# Step 1: Drop rows with missing target
df.dropna(subset=['Cases - Weekly'], inplace=True)

# Step 2: Feature selection
features = df.drop(columns=['Cases - Weekly', 'Week Start', 'Week End'])
target = df['Cases - Weekly']

# Step 3: Label encode categorical features
categorical_cols = features.select_dtypes(include=['object', 
'category']).columns
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    features[col] = le.fit_transform(features[col])
    label_encoders[col] = le

# Step 4: Train/test split
X_train, X_test, y_train, y_test = train_test_split(features, target, 
test_size=0.2, random_state=42)

# Step 5: Hyperparameter grid
param_grid = {
```

```
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'max_features': ['sqrt', 'log2']
}

# Step 6: GridSearchCV
grid_search = GridSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_grid=param_grid,
    cv=5,
    n_jobs=-1,
    scoring='neg_mean_squared_error',
    verbose=1
)

# Step 7: Fit model with best parameters
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

# Step 8: Predict and evaluate
y_pred = best_model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print metrics
print("Best Parameters:", grid_search.best_params_)
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R² Score: {r2:.2f}")
print(f"R² Score (%): {r2 * 100:.2f}%")
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
Best Parameters: {'max_depth': 20, 'max_features': 'sqrt',
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Mean Squared Error (MSE): 708.94
Mean Absolute Error (MAE): 6.29
R² Score: 0.96
R² Score (%): 95.86%
```

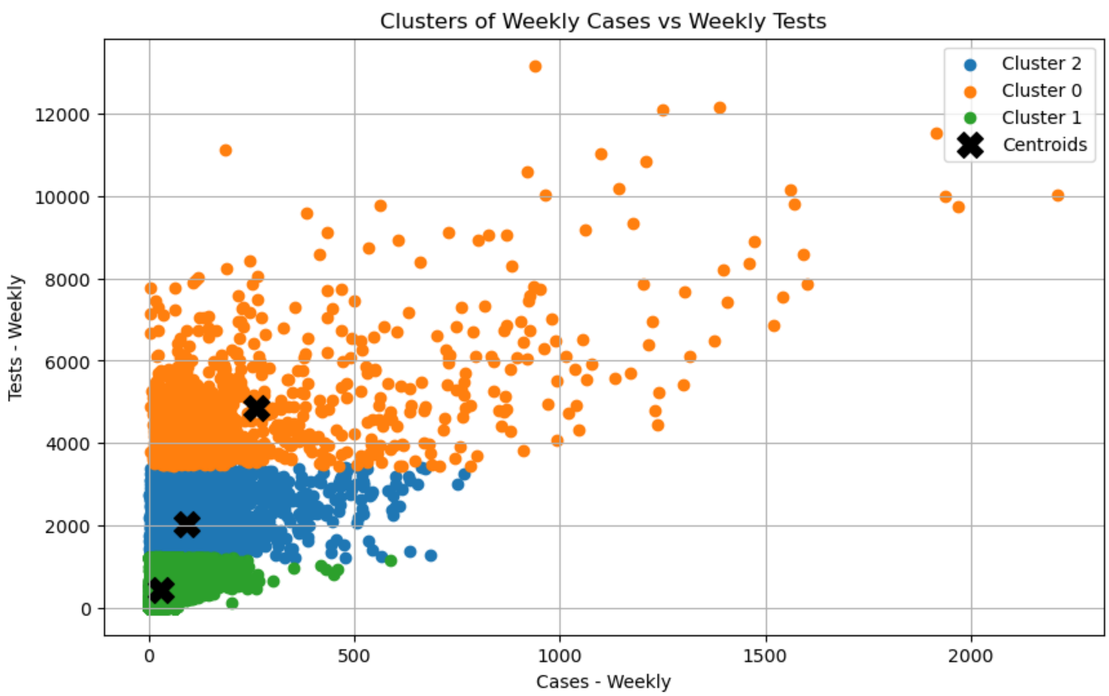
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

param_dist = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30, None],
```

```
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'max_features': ['sqrt', 'log2']
}

rf = RandomForestRegressor(random_state=42)
random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_dist,
    n_iter=20,
    cv=3,
    verbose=1,
    n_jobs=-1,
    random_state=42
)

random_search.fit(X_train, y_train)
best_rf = random_search.best_estimator_

# Evaluation
y_pred = best_rf.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
r2_percent = r2 * 100

print(f"Mean Squared Error: {mse:.2f}")
print(f"Mean Absolute Error: {mae:.2f}")
print(f"R² Score: {r2:.2f}")
print(f"R² Score (%): {r2_percent:.2f}%")
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
Mean Squared Error: 708.12
Mean Absolute Error: 6.28
R² Score: 0.96
R² Score (%): 95.87%
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Load the data (replace with your actual DataFrame if already loaded)
file_path = 'C:\\Users\\Dell\\Desktop\\Machine learnng\\Final proj\\archive
(1)\\COVID-19_Cases__Tests__and_Deaths_by_ZIP_Code_-_Historical.csv'

# Drop rows with missing values in the required columns
df = df.dropna(subset=['Cases - Weekly', 'Tests - Weekly'])

# Extract the features
X = df[['Cases - Weekly', 'Tests - Weekly']]
```

```python
# Fit KMeans with 3 clusters (you can change this number)
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(X)

# Plot the clusters
plt.figure(figsize=(10, 6))
for cluster in df['Cluster'].unique():
    cluster_data = df[df['Cluster'] == cluster]
    plt.scatter(cluster_data['Cases - Weekly'], cluster_data['Tests -
Weekly'], label=f'Cluster {cluster}')

# Plot centroids
centroids = kmeans.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='black', marker='X',
label='Centroids')

plt.title('Clusters of Weekly Cases vs Weekly Tests')
plt.xlabel('Cases - Weekly')
plt.ylabel('Tests - Weekly')
plt.legend()
plt.grid(True)
plt.show()
```



Decision tree regressor

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

#Loading the Dataset
df = pd.read_csv("C:/Users/HP/Downloads/archive (1) (1)/COVID-19_Cases__Tests__and_Deaths_by_ZIP_Code_-_Historical.csv")
df.head(10)
```

```
  ZIP Code  Week Number  Week Start    Week End  Cases - Weekly  \
0    60622           31  07/26/2020  08/01/2020            28.0
1    60622           32  08/02/2020  08/08/2020            34.0
2    60622           33  08/09/2020  08/15/2020            41.0
3    60622           34  08/16/2020  08/22/2020            42.0
4    60622           35  08/23/2020  08/29/2020            45.0
5    60622           36  08/30/2020  09/05/2020            29.0
6    60622           37  09/06/2020  09/12/2020            46.0
7    60622           39  09/20/2020  09/26/2020            63.0
8    60622           40  09/27/2020  10/03/2020            45.0
9    60622           43  10/18/2020  10/24/2020           166.0

   Cases - Cumulative  Case Rate - Weekly  Case Rate - Cumulative  \
0               877.0                53.0                  1661.2
1               911.0                64.0                  1725.6
2               952.0                78.0                  1803.3
3               994.0                80.0                  1882.8
4              1039.0                85.0                  1968.1
5              1068.0                55.0                  2023.0
6              1114.0                87.0                  2110.1
7              1217.0               119.0                  2305.2
8              1262.0                85.0                  2390.5
9              1617.0               314.0                  3062.9

   Tests - Weekly  Tests - Cumulative  ...  Test Rate - Cumulative  \
0          1329.0               13148  ...                 24904.8
1          1405.0               14553  ...                 27566.2
2          1542.0               16095  ...                 30487.0
3          1674.0               17769  ...                 33657.9
4          1540.0               19309  ...                 36574.9
5          1547.0               20856  ...                 39505.2
6          1400.0               22256  ...                 42157.1
7          1844.0               25763  ...                 48800.0
8          1705.0               27468  ...                 52029.6
```

```
9         2642.0               34626  ...                65588.2

   Percent Tested Positive - Weekly  Percent Tested Positive - Cumulative  \
0                               0.0                                   0.1
1                               0.0                                   0.1
2                               0.0                                   0.1
3                               0.0                                   0.1
4                               0.0                                   0.1
5                               0.0                                   0.1
6                               0.0                                   0.1
7                               0.0                                   0.0
8                               0.0                                   0.0
9                               0.1                                   0.0

   Deaths - Weekly  Deaths - Cumulative  Death Rate - Weekly  \
0                0                   56                  0.0
1                0                   56                  0.0
2                0                   56                  0.0
3                0                   56                  0.0
4                0                   56                  0.0
5                0                   56                  0.0
6                0                   56                  0.0
7                0                   56                  0.0
8                0                   56                  0.0
9                0                   56                  0.0

   Death Rate - Cumulative  Population         Row ID  \
0                    106.1       52793  60622-2020-31
1                    106.1       52793  60622-2020-32
2                    106.1       52793  60622-2020-33
3                    106.1       52793  60622-2020-34
4                    106.1       52793  60622-2020-35
5                    106.1       52793  60622-2020-36
6                    106.1       52793  60622-2020-37
7                    106.1       52793  60622-2020-39
8                    106.1       52793  60622-2020-40
9                    106.1       52793  60622-2020-43

           ZIP Code Location
0  POINT (-87.681818 41.902762)
1  POINT (-87.681818 41.902762)
2  POINT (-87.681818 41.902762)
3  POINT (-87.681818 41.902762)
4  POINT (-87.681818 41.902762)
5  POINT (-87.681818 41.902762)
6  POINT (-87.681818 41.902762)
7  POINT (-87.681818 41.902762)
8  POINT (-87.681818 41.902762)
9  POINT (-87.681818 41.902762)
```

```
[10 rows x 21 columns]
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13132 entries, 0 to 13131
Data columns (total 21 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   ZIP Code                              13132 non-null  object 
 1   Week Number                           13132 non-null  int64  
 2   Week Start                            13132 non-null  object 
 3   Week End                              13132 non-null  object 
 4   Cases - Weekly                        12909 non-null  float64
 5   Cases - Cumulative                    12909 non-null  float64
 6   Case Rate - Weekly                    12909 non-null  float64
 7   Case Rate - Cumulative                12909 non-null  float64
 8   Tests - Weekly                        12740 non-null  float64
 9   Tests - Cumulative                    13132 non-null  int64  
 10  Test Rate - Weekly                    13132 non-null  int64  
 11  Test Rate - Cumulative                13132 non-null  float64
 12  Percent Tested Positive - Weekly      13132 non-null  float64
 13  Percent Tested Positive - Cumulative  13132 non-null  float64
 14  Deaths - Weekly                       13132 non-null  int64  
 15  Deaths - Cumulative                   13132 non-null  int64  
 16  Death Rate - Weekly                   13132 non-null  float64
 17  Death Rate - Cumulative               13132 non-null  float64
 18  Population                            13132 non-null  int64  
 19  Row ID                                13132 non-null  object 
 20  ZIP Code Location                     12921 non-null  object 
dtypes: float64(10), int64(6), object(5)
memory usage: 2.1+ MB
```

```
df.shape
```

```
(13132, 21)
```

```
df.describe()
```

```
        Week Number  Cases - Weekly  Cases - Cumulative  Case Rate - Weekly  \
count  13132.000000    12909.000000        12909.000000        12909.000000   
mean      26.170119       63.458440         8344.924161          136.947401   
std       14.871736      121.313518         7516.565007          245.224599   
min        1.000000        0.000000            5.000000            0.000000   
25%       13.000000       11.000000         1989.000000           32.000000   
50%       25.000000       30.000000         6503.000000           76.000000   
75%       39.000000       70.000000        12839.000000          150.000000   
max       53.000000     2212.000000        36570.000000         6266.000000   
```

```
       Case Rate - Cumulative  Tests - Weekly  Tests - Cumulative  \
count            12909.000000    12740.000000        13132.000000
mean             17734.813309     1225.955024       129983.026652
std              11955.509645     1400.608932       135184.120574
min                  0.000000        0.000000            0.000000
25%               7127.600000      158.000000        10100.500000
50%              19382.600000      835.500000        86097.000000
75%              27597.600000     1807.250000       223838.750000
max              64450.100000    13173.000000       538868.000000

       Test Rate - Weekly  Test Rate - Cumulative  \
count        13132.000000            1.313200e+04
mean          2677.341989            2.957631e+05
std           3240.396176            2.931501e+05
min              0.000000            0.000000e+00
25%            369.000000            2.742940e+04
50%           1946.000000            2.331007e+05
75%           3795.250000            4.949480e+05
max          75755.000000            2.037212e+06

       Percent Tested Positive - Weekly  Percent Tested Positive - Cumulative
\
count                      13132.000000                          13132.000000
mean                           0.056298                              0.074147
std                            0.078874                              0.064195
min                            0.000000                              0.000000
25%                            0.000000                              0.000000
50%                            0.000000                              0.100000
75%                            0.100000                              0.100000
max                            1.000000                              0.500000

       Deaths - Weekly  Deaths - Cumulative  Death Rate - Weekly  \
count     13132.000000         13132.000000         13132.000000
mean          0.636689           105.623896             1.218299
std           1.634849            91.039144             3.309388
min           0.000000             0.000000             0.000000
25%           0.000000            19.000000             0.000000
50%           0.000000            90.000000             0.000000
75%           1.000000           168.000000             1.200000
max          25.000000           365.000000            80.400000

       Death Rate - Cumulative     Population
count             13132.000000   13132.000000
mean                199.797525   46258.380064
std                 138.398733   26835.033756
min                   0.000000       0.000000
25%                  81.800000   28804.000000
50%                 192.900000   46024.000000
```

```
75%                  309.000000   68096.000000
max                  540.600000  111850.000000
```

```python
df.dtypes
```

```
ZIP Code                                     object
Week Number                                   int64
Week Start                                   object
Week End                                     object
Cases - Weekly                              float64
Cases - Cumulative                          float64
Case Rate - Weekly                          float64
Case Rate - Cumulative                      float64
Tests - Weekly                              float64
Tests - Cumulative                            int64
Test Rate - Weekly                            int64
Test Rate - Cumulative                      float64
Percent Tested Positive - Weekly            float64
Percent Tested Positive - Cumulative        float64
Deaths - Weekly                               int64
Deaths - Cumulative                           int64
Death Rate - Weekly                         float64
Death Rate - Cumulative                     float64
Population                                    int64
Row ID                                       object
ZIP Code Location                            object
dtype: object
```

```python
df = df.replace('Unknown', np.nan)
```

```python
df.isnull().sum()
```

```
ZIP Code                                  211
Week Number                                 0
Week Start                                  0
Week End                                    0
Cases - Weekly                            223
Cases - Cumulative                        223
Case Rate - Weekly                        223
Case Rate - Cumulative                    223
Tests - Weekly                            392
Tests - Cumulative                          0
Test Rate - Weekly                          0
Test Rate - Cumulative                      0
Percent Tested Positive - Weekly            0
Percent Tested Positive - Cumulative        0
Deaths - Weekly                             0
Deaths - Cumulative                         0
Death Rate - Weekly                         0
Death Rate - Cumulative                     0
Population                                  0
```

```
Row ID                                    0
ZIP Code Location                       211
dtype: int64
```

```
duplicates=df.duplicated()
num_duplicates = duplicates.sum()
print("Number of duplicates: ", num_duplicates)
```

```
Number of duplicates:  0
```

```
df.drop(columns=['Row ID', 'ZIP Code Location'], inplace=True)
```

```
#Data Cleaning and Preprocessing
df['ZIP Code'].fillna(df['ZIP Code'].mode()[0], inplace=True)
df['Cases - Weekly'].fillna(df['Cases - Weekly'].mean(), inplace=True)
df['Cases - Cumulative'].fillna(df['Cases - Cumulative'].mean(),
inplace=True)
df['Case Rate - Weekly'].fillna(df['Case Rate - Weekly'].mean(),
inplace=True)
df['Case Rate - Cumulative'].fillna(df['Case Rate - Cumulative'].mean(),
inplace=True)
df['Tests - Weekly'].fillna(df['Tests - Weekly'].mean(), inplace=True)
df.isnull().sum()
```

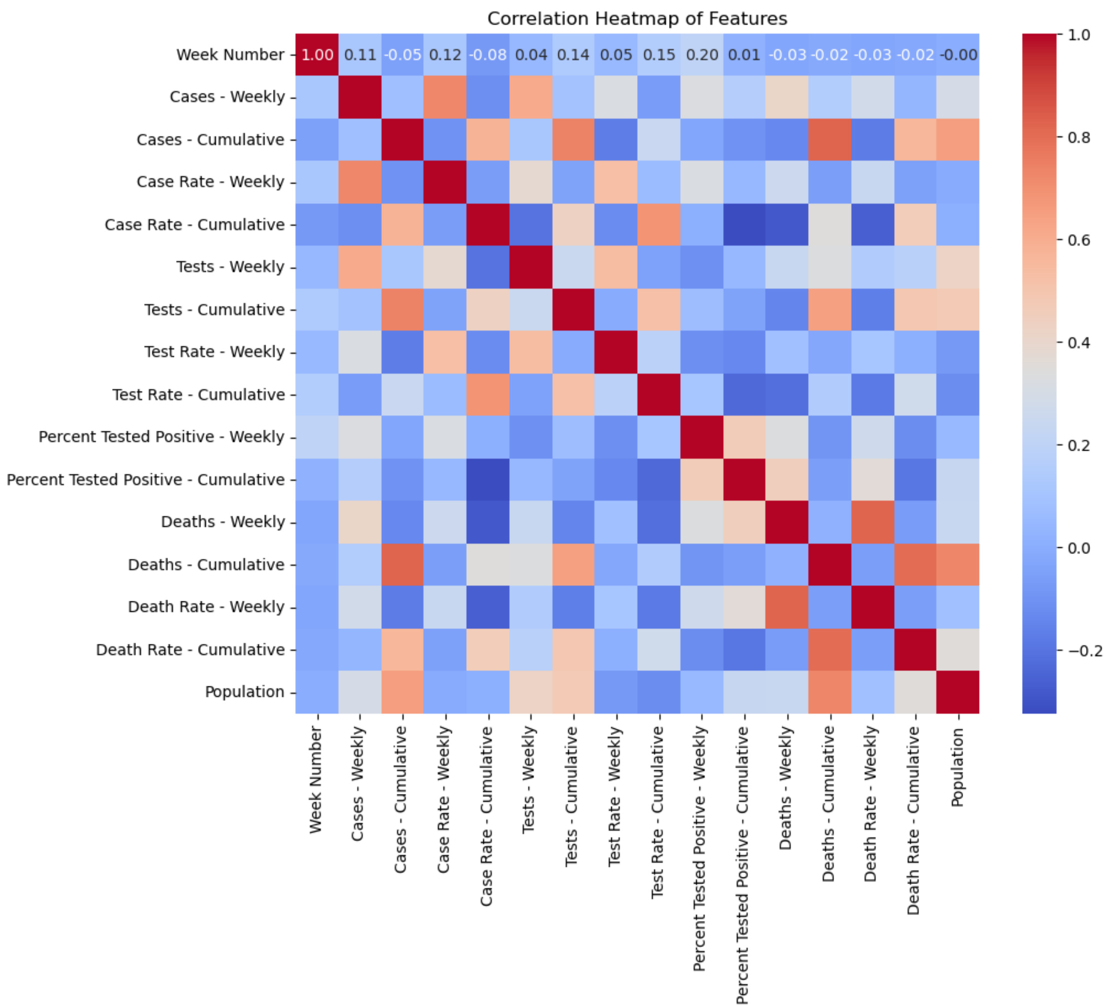
```
ZIP Code                                0
Week Number                             0
Week Start                              0
Week End                                0
Cases - Weekly                          0
Cases - Cumulative                      0
Case Rate - Weekly                      0
Case Rate - Cumulative                  0
Tests - Weekly                          0
Tests - Cumulative                      0
Test Rate - Weekly                      0
Test Rate - Cumulative                  0
Percent Tested Positive - Weekly        0
Percent Tested Positive - Cumulative    0
Deaths - Weekly                         0
Deaths - Cumulative                     0
Death Rate - Weekly                     0
Death Rate - Cumulative                 0
Population                              0
dtype: int64
```

```
df['ZIP Code'] = df['ZIP Code'].astype(str)
# Convert date columns to datetime format
df['Week Start'] = pd.to_datetime(df['Week Start'])
df['Week End'] = pd.to_datetime(df['Week End'])
```

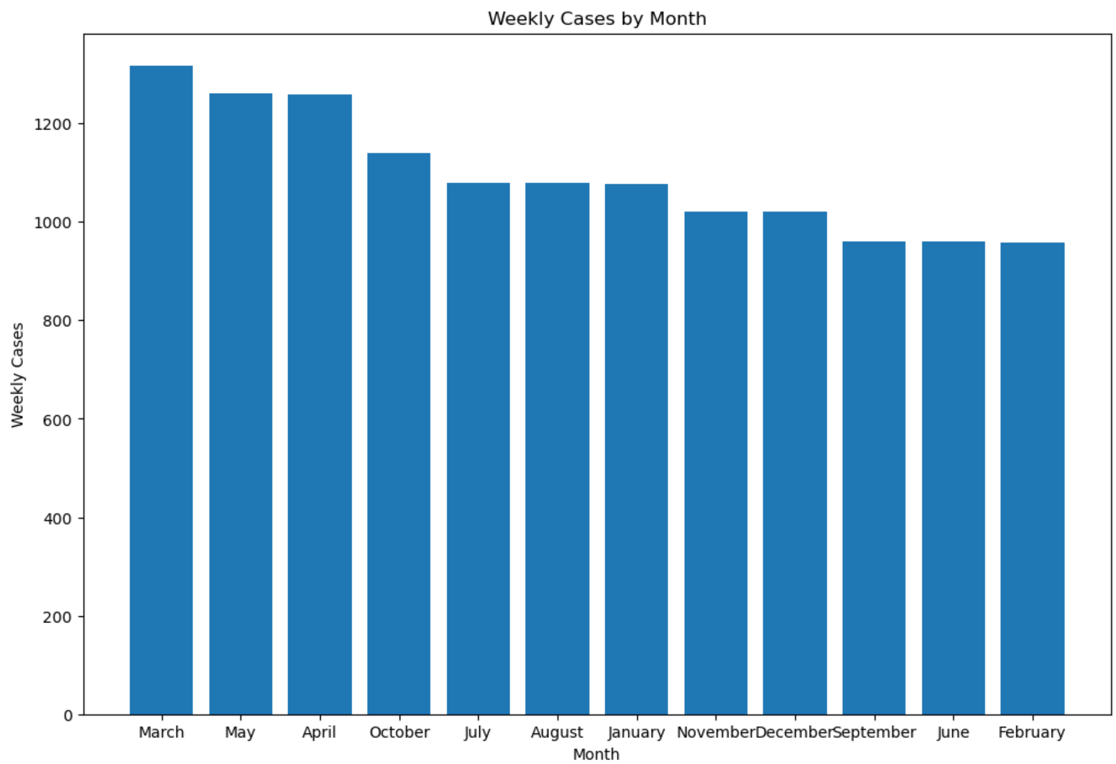
```
numerical_columns = df.select_dtypes(include='number').columns
categorical_columns_1 = df.select_dtypes(include='object').columns
categorical_columns_2 = df.select_dtypes(exclude=np.number).columns
print(numerical_columns)
print("\n")
print(categorical_columns_1)
print("\n")
print(categorical_columns_2)
```

```
Index(['Week Number', 'Cases - Weekly', 'Cases - Cumulative',
       'Case Rate - Weekly', 'Case Rate - Cumulative', 'Tests - Weekly',
       'Tests - Cumulative', 'Test Rate - Weekly', 'Test Rate - Cumulative',
       'Percent Tested Positive - Weekly',
       'Percent Tested Positive - Cumulative', 'Deaths - Weekly',
       'Deaths - Cumulative', 'Death Rate - Weekly', 'Death Rate -
Cumulative',
       'Population'],
      dtype='object')


Index(['ZIP Code'], dtype='object')


Index(['ZIP Code', 'Week Start', 'Week End'], dtype='object')
```

```
plt.figure(figsize=(10, 8))
correlation_matrix = df[numerical_columns].corr()
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Heatmap of Features')
plt.show()
```




```
#Get the name of month
df_new = df.copy()
df_new['Week_Start_Month'] = df['Week Start'].dt.month_name()
df_new['Week Start Year'] = df['Week Start'].dt.year

#Calculate the count
Week_Start = df_new['Week_Start_Month'].value_counts()
plt.figure(figsize=(12, 8))
plt.bar(Week_Start.index, Week_Start.values)
plt.title('Weekly Cases by Month')
plt.xlabel('Month')
plt.ylabel('Weekly Cases')
plt.show()

df.drop(columns=['Week Start', 'Week End'], inplace=True)
```



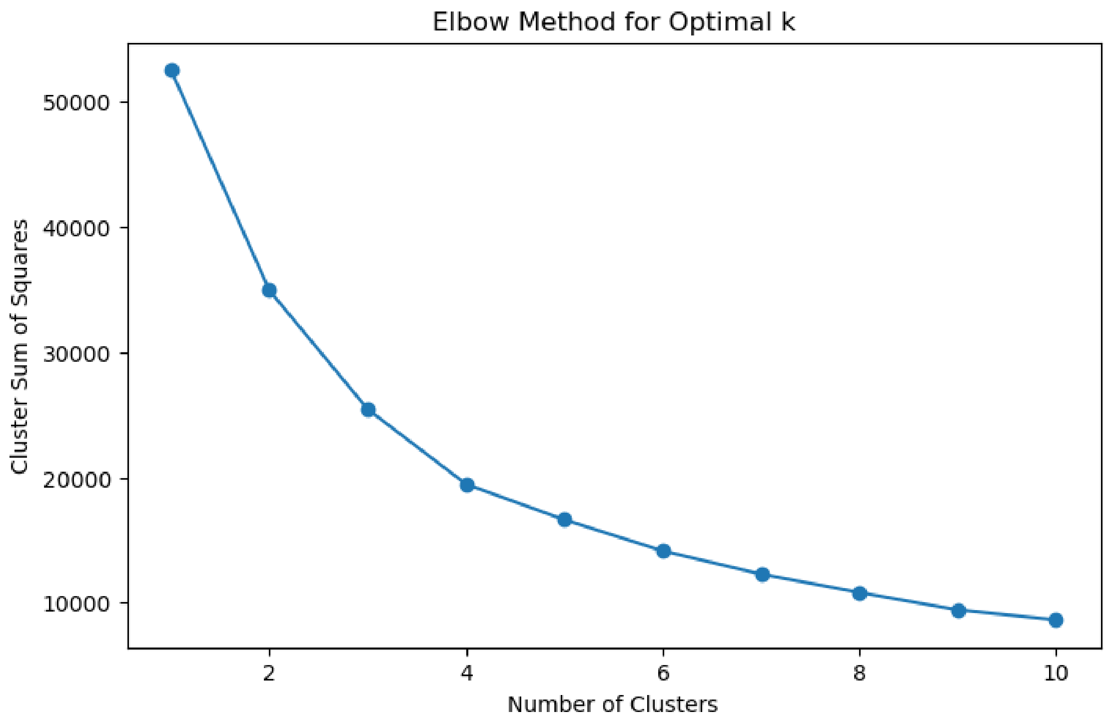

```
df = pd.get_dummies(df, columns=['ZIP Code'], drop_first=True)

X = df.drop(columns=["Cases - Weekly"])
y = df["Cases - Weekly"]

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

model = DecisionTreeRegressor(random_state=42)
model.fit(X_train, y_train)

DecisionTreeRegressor(random_state=42)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"R-squared (R² Score): {r2*100:.4f}%")

Mean Absolute Error (MAE): 4.1203
Mean Squared Error (MSE): 711.8736
R-squared (R² Score): 95.8450%
```

```python
param_grid = {
    'max_depth': [5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 5, 10],
    'max_features': [None, 'sqrt', 'log2']
}

grid_search = GridSearchCV(DecisionTreeRegressor(random_state=42),
param_grid, cv=5, scoring='r2', n_jobs=-1)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
print(f"Best Parameters: {best_params}")

best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)

mse_best = mean_squared_error(y_test, y_pred_best)
r2_best = r2_score(y_test, y_pred_best)
mae_best = mean_absolute_error(y_test, y_pred_best)

print(f"Optimized MSE: {mse_best:.4f}")
print(f"Optimized MAE: {mae_best:.4f}")
print(f"R-squared (R² Score): {r2_best*100:.2f}%")
```

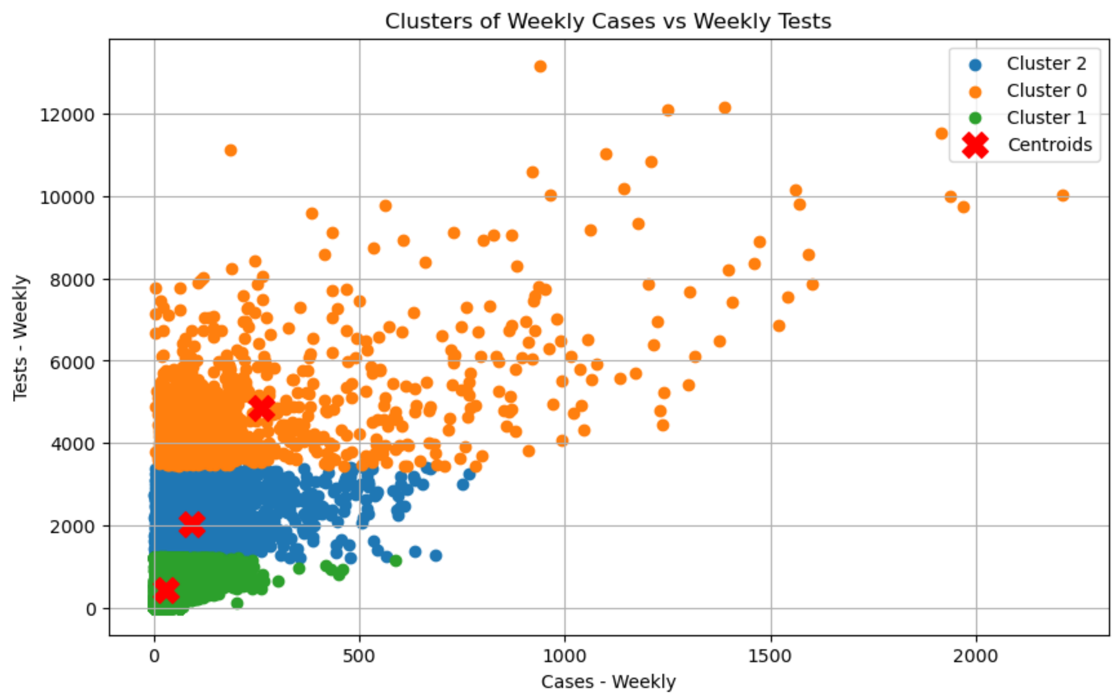
```
Best Parameters: {'max_depth': 20, 'max_features': None, 'min_samples_leaf':
1, 'min_samples_split': 2}
Optimized MSE: 674.2795
Optimized MAE: 4.1368
R-squared (R² Score): 96.06%
```

```python
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

features = ['Cases - Weekly', 'Case Rate - Weekly', 'Tests - Weekly',
'Percent Tested Positive - Weekly']
df_new = df_new[['ZIP Code'] + features]

scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_new[features])

wcss = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(df_scaled)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), wcss, marker='o')
plt.xlabel("Number of Clusters")
```

```
plt.ylabel("Cluster Sum of Squares")
plt.title("Elbow Method for Optimal k")
plt.show()
```

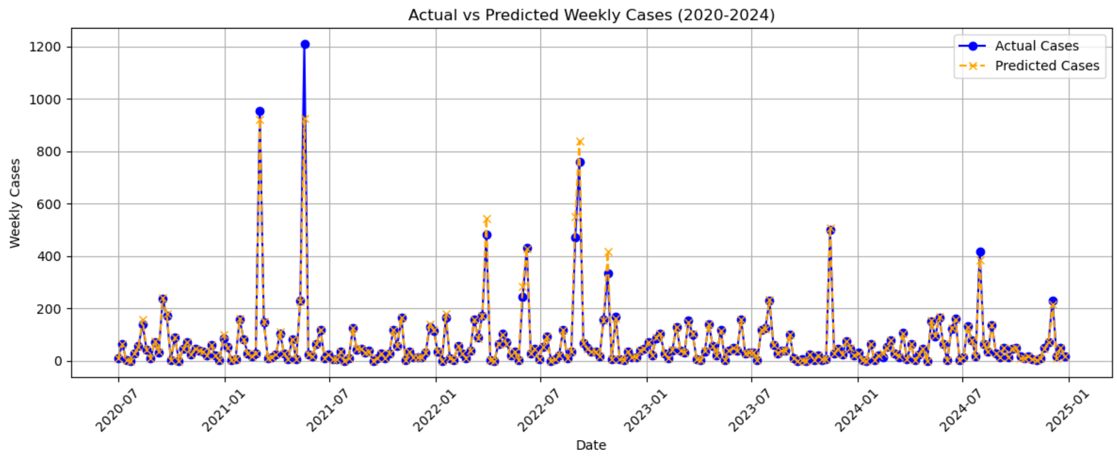



```
X = df_new[['Cases - Weekly', 'Tests - Weekly']]

kmeans = KMeans(n_clusters=3, random_state=42)

df_new['Cluster'] = kmeans.fit_predict(X)

plt.figure(figsize=(10, 6))
for cluster in df_new['Cluster'].unique():
    cluster_data = df_new[df_new['Cluster'] == cluster]
    plt.scatter(cluster_data['Cases - Weekly'], cluster_data['Tests - 
Weekly'], label=f'Cluster {cluster}')

centroids = kmeans.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1], s=200, color='red', marker='X', 
label='Centroids')

plt.title('Clusters of Weekly Cases vs Weekly Tests')
plt.xlabel('Cases - Weekly')
plt.ylabel('Tests - Weekly')
plt.legend()
plt.grid(True)
plt.show()
```

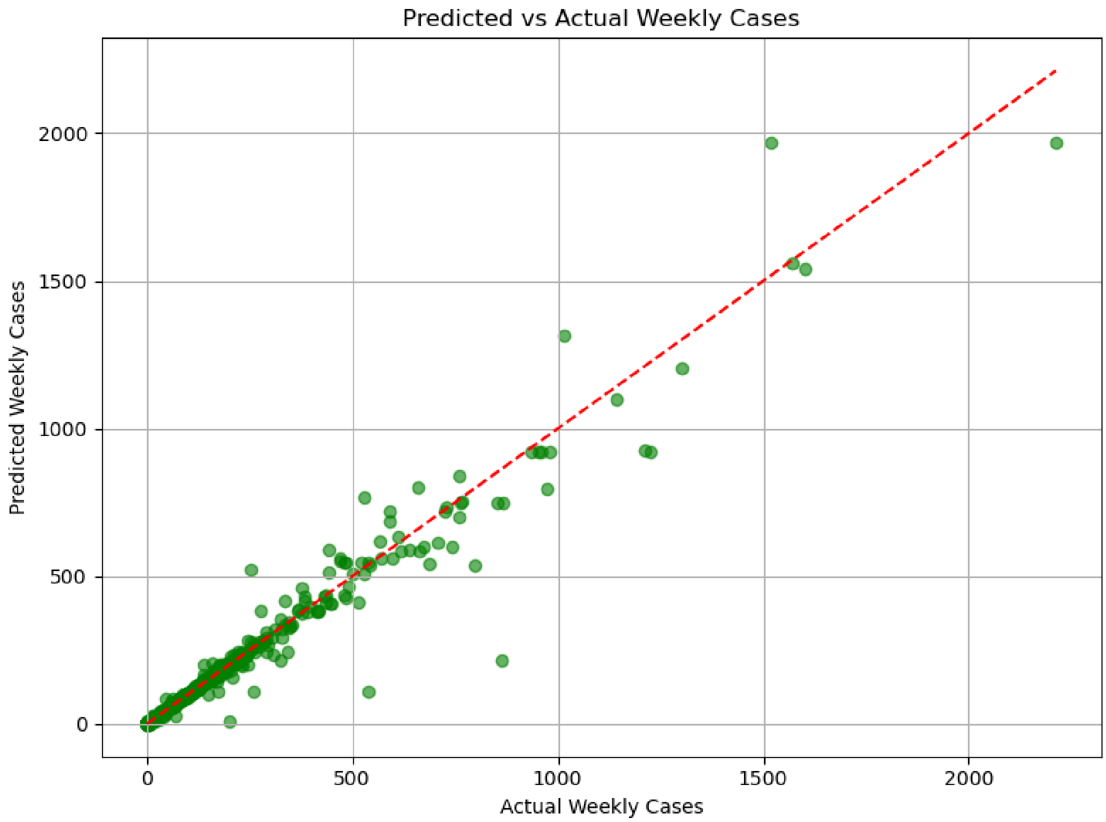



```python
import matplotlib.pyplot as plt
import pandas as pd
import datetime

start_date = datetime.datetime(2020, 1, 1)
dates = [start_date + datetime.timedelta(weeks=i) for i in
range(len(y_test))]

df = pd.DataFrame({
    "Week Start": dates,
    "Actual": y_test,
    "Predicted": y_pred_best
})
df = df[(df["Week Start"] >= '2020-07-01') & (df["Week Start"] <= '2024-12-
31')]

plt.figure(figsize=(12, 5))
plt.plot(df['Week Start'], df['Actual'], 'o-', color='blue', label='Actual
Cases')
plt.plot(df['Week Start'], df['Predicted'], 'x--', color='orange',
label='Predicted Cases')

plt.title("Actual vs Predicted Weekly Cases (2020-2024)")
plt.xlabel("Date")
plt.ylabel("Weekly Cases")
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
```

```
plt.tight_layout()
plt.show()
```




```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred_best, color='green', alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.title("Predicted vs Actual Weekly Cases")
plt.xlabel("Actual Weekly Cases")
plt.ylabel("Predicted Weekly Cases")
plt.grid(True)
plt.tight_layout()
plt.show()
```


Predicted vs Actual Weekly Cases
Predicted Weekly Cases
Actual Weekly Cases
0
500
1000
1500
2000
0
500
1000
1500
2000

Gradient boost regressor

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

from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score,mean_absolute_error
from sklearn.model_selection import RandomizedSearchCV

df = pd.read_csv("D:/PH 3022/Project/COVID-
19_Cases__Tests__and_Deaths_by_ZIP_Code_-_Historical.csv")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13132 entries, 0 to 13131
Data columns (total 21 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   ZIP Code                            13132 non-null  object
 1   Week Number                         13132 non-null  int64
 2   Week Start                          13132 non-null  object
 3   Week End                            13132 non-null  object
 4   Cases - Weekly                      12909 non-null  float64
 5   Cases - Cumulative                  12909 non-null  float64
 6   Case Rate - Weekly                  12909 non-null  float64
 7   Case Rate - Cumulative              12909 non-null  float64
 8   Tests - Weekly                      12740 non-null  float64
 9   Tests - Cumulative                  13132 non-null  int64
 10  Test Rate - Weekly                  13132 non-null  int64
 11  Test Rate - Cumulative              13132 non-null  float64
 12  Percent Tested Positive - Weekly    13132 non-null  float64
 13  Percent Tested Positive - Cumulative  13132 non-null  float64
 14  Deaths - Weekly                     13132 non-null  int64
 15  Deaths - Cumulative                 13132 non-null  int64
 16  Death Rate - Weekly                 13132 non-null  float64
 17  Death Rate - Cumulative             13132 non-null  float64
 18  Population                          13132 non-null  int64
 19  Row ID                              13132 non-null  object
 20  ZIP Code Location                   12921 non-null  object
dtypes: float64(10), int64(6), object(5)
memory usage: 2.1+ MB
```

```
print("Dataset Null Value")
print(df.isnull().sum())
print("Number of duplicates values: ", df.duplicated().sum())
```

```
Dataset Null Value
ZIP Code                                 0
Week Number                              0
Week Start                               0
Week End                                 0
Cases - Weekly                         223
Cases - Cumulative                     223
Case Rate - Weekly                     223
Case Rate - Cumulative                 223
Tests - Weekly                         392
Tests - Cumulative                       0
Test Rate - Weekly                       0
Test Rate - Cumulative                   0
Percent Tested Positive - Weekly         0
Percent Tested Positive - Cumulative     0
Deaths - Weekly                          0
Deaths - Cumulative                      0
Death Rate - Weekly                      0
Death Rate - Cumulative                  0
Population                               0
Row ID                                   0
ZIP Code Location                      211
dtype: int64
Number of duplicates values:  0
```

```
unique = ['ZIP Code', 'Week Number', 'ZIP Code Location']
for col in unique:
    print(df[col].value_counts())
    print()
```

```
ZIP Code
60622      219
60609      219
60615      219
60610      219
60619      219
60607      219
60624      219
60625      219
60626      219
60628      219
60629      219
60612      219
60654      219
60618      219
60636      219
60651      219
60657      219
60660      219
60611      219
```

```
60601      219
60602      219
60641      219
60646      219
60827      219
60661      219
60603      219
60642      219
60630      219
60631      219
60632      219
60633      219
60634      219
60623      219
60614      219
60638      219
60666      219
60604      219
60653      219
60644      219
60616      219
60617      219
60621      219
60637      219
60639      219
60640      219
60643      219
60652      219
60655      219
60656      219
60608      219
60659      219
60605      219
60620      219
60707      219
60613      219
60606      219
60649      219
60645      219
60647      219
Unknown    211
Name: count, dtype: int64

Week Number
20    300
18    300
19    300
17    300
16    300
13    300
```

```
10    300
15    299
11    299
14    299
12    299
43    240
40    240
36    240
37    240
39    240
31    240
32    240
33    240
34    240
35    240
27    240
24    240
52    240
51    240
50    240
49    240
48    240
47    240
46    240
45    240
44    240
41    240
22    240
8     240
5     240
21    240
30    240
29    240
23    240
38    240
42    240
25    240
6     240
28    240
26    240
3     240
9     239
2     239
4     239
7     239
1     180
53     60
Name: count, dtype: int64

ZIP Code Location
```

POINT (-87.681818 41.902762)	219
POINT (-87.653382 41.812017)	219
POINT (-87.602725 41.801993)	219
POINT (-87.63581 41.90455)	219
POINT (-87.60569 41.744737)	219
POINT (-87.652727 41.876104)	219
POINT (-87.722735 41.879417)	219
POINT (-87.701816 41.971155)	219
POINT (-87.669834 42.009469)	219
POINT (-87.621537 41.694192)	219
POINT (-87.711565 41.777061)	219
POINT (-87.687011 41.88004)	219
POINT (-87.636354 41.892485)	219
POINT (-87.703343 41.946699)	219
POINT (-87.668597 41.77599)	219
POINT (-87.741017 41.901964)	219
POINT (-87.658216 41.939715)	219
POINT (-87.666362 41.991062)	219
POINT (-87.620291 41.894734)	219
POINT (-87.622844 41.886262)	219
POINT (-87.628309 41.883136)	219
POINT (-87.746791 41.946682)	219
POINT (-87.761826 41.993931)	219
POINT (-87.633087 41.650765)	219
POINT (-87.644283 41.882786)	219
POINT (-87.625473 41.880112)	219
POINT (-87.657821 41.899935)	219
POINT (-87.759611 41.971261)	219
POINT (-87.813371 41.995019)	219
POINT (-87.711251 41.810038)	219
POINT (-87.556037 41.653147)	219
POINT (-87.797373 41.944967)	219
POINT (-87.717446 41.850321)	219
POINT (-87.652064 41.922605)	219
POINT (-87.771902 41.787032)	219
POINT (-87.896371 41.979511)	219
POINT (-87.629029 41.878153)	219
POINT (-87.611244 41.819261)	219
POINT (-87.756863 41.881113)	219
POINT (-87.629531 41.844869)	219
POINT (-87.556897 41.721257)	219
POINT (-87.638812 41.776931)	219
POINT (-87.604053 41.780991)	219
POINT (-87.75531 41.920609)	219
POINT (-87.662232 41.971888)	219
POINT (-87.662381 41.700445)	219
POINT (-87.714238 41.745398)	219
POINT (-87.701434 41.696456)	219
POINT (-87.817934 41.974566)	219
POINT (-87.670366 41.849879)	219

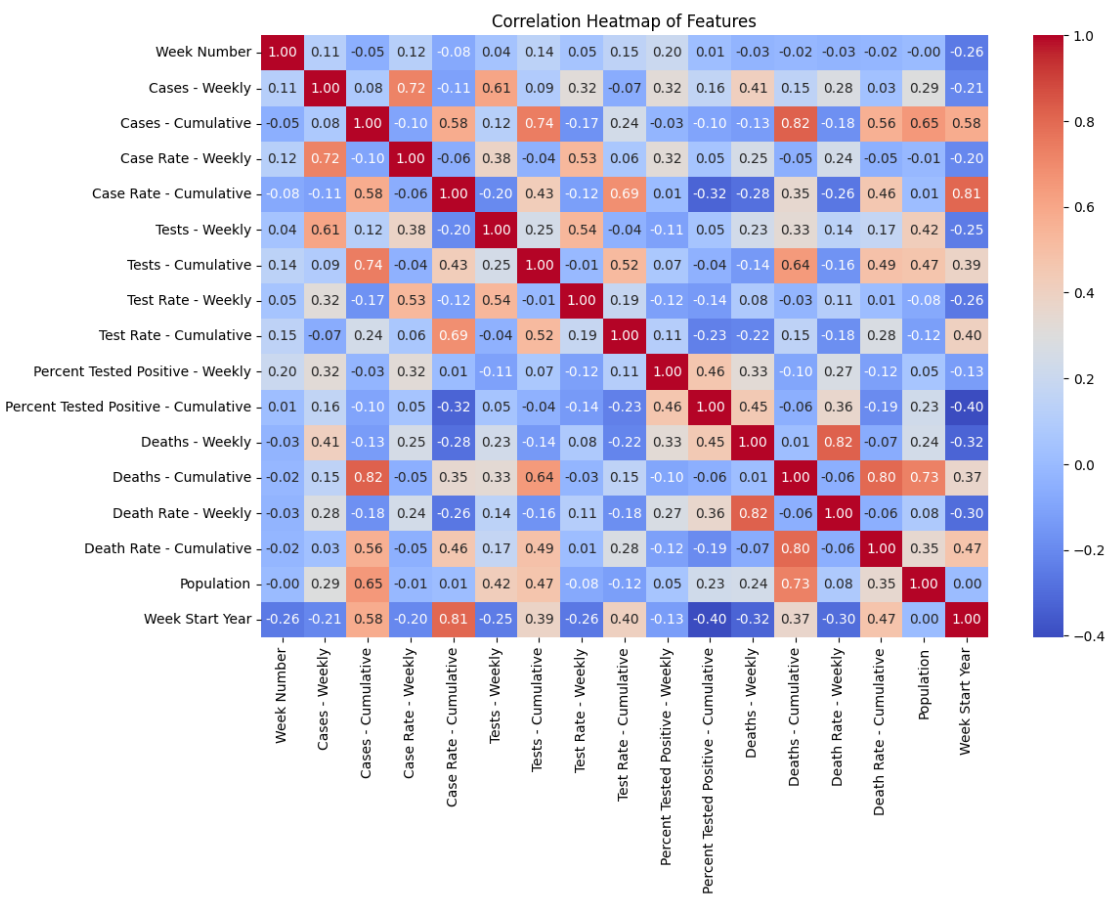
```
POINT (-87.703266 41.990803)    219
POINT (-87.623449 41.867824)    219
POINT (-87.651656 41.740873)    219
POINT (-87.808283 41.921777)    219
POINT (-87.661343 41.953742)    219
POINT (-87.63676 41.882634)     219
POINT (-87.695049 42.008927)    219
POINT (-87.571522 41.762202)    219
POINT (-87.701101 41.921058)    219
Name: count, dtype: int64
```

```
df['Cases - Weekly'] = df['Cases - Weekly'].fillna(df['Cases - Weekly'].mean())
df['Cases - Cumulative'] = df['Cases - Cumulative'].fillna(df['Cases - Cumulative'].mean())
df['Case Rate - Weekly'] = df['Case Rate - Weekly'].fillna(df['Case Rate - Weekly'].mean())
df['Case Rate - Cumulative'] = df['Case Rate - Cumulative'].fillna(df['Case Rate - Cumulative'].mean())
df['Tests - Weekly'] = df['Tests - Weekly'].fillna(df['Tests - Weekly'].mean())

df['ZIP Code Location'] = df['ZIP Code Location'].fillna(df['ZIP Code Location'].mode()[0])

df['Week Start'] = pd.to_datetime(df['Week Start'], errors='coerce')
df['Week End'] = pd.to_datetime(df['Week End'], errors='coerce')

print("Dataset Null Value")
print(df.isnull().sum())
```

```
Dataset Null Value
ZIP Code                              0
Week Number                           0
Week Start                            0
Week End                              0
Cases - Weekly                        0
Cases - Cumulative                    0
Case Rate - Weekly                    0
Case Rate - Cumulative                0
Tests - Weekly                        0
Tests - Cumulative                    0
Test Rate - Weekly                    0
Test Rate - Cumulative                0
Percent Tested Positive - Weekly      0
Percent Tested Positive - Cumulative  0
Deaths - Weekly                       0
Deaths - Cumulative                   0
```

```
Death Rate - Weekly                  0
Death Rate - Cumulative              0
Population                           0
Row ID                               0
ZIP Code Location                    0
dtype: int64
```

```python
df['Week_Start_Month'] = df['Week Start'].dt.month_name()
df['Week Start Year'] = df['Week Start'].dt.year

num_cols = df.select_dtypes(include=np.number).columns
cat_cols = df.select_dtypes(exclude=np.number).columns

plt.figure(figsize=(12, 8))
correlation_matrix = df[num_cols].corr()
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Heatmap of Features')
plt.show()
```

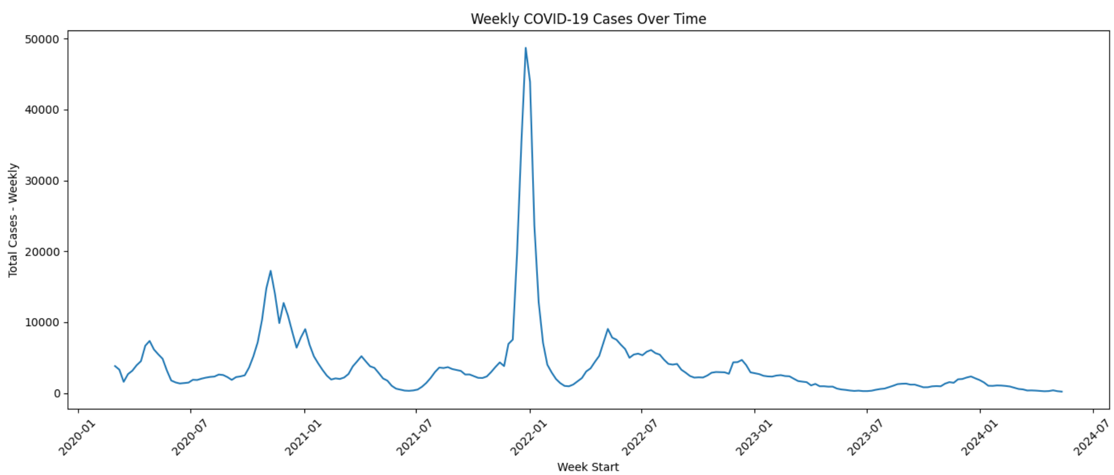



```python
threshold = 0.05

correlation_matrix = df.corr(numeric_only=True)
high_corr_features = correlation_matrix.index[abs(correlation_matrix["Cases -
```

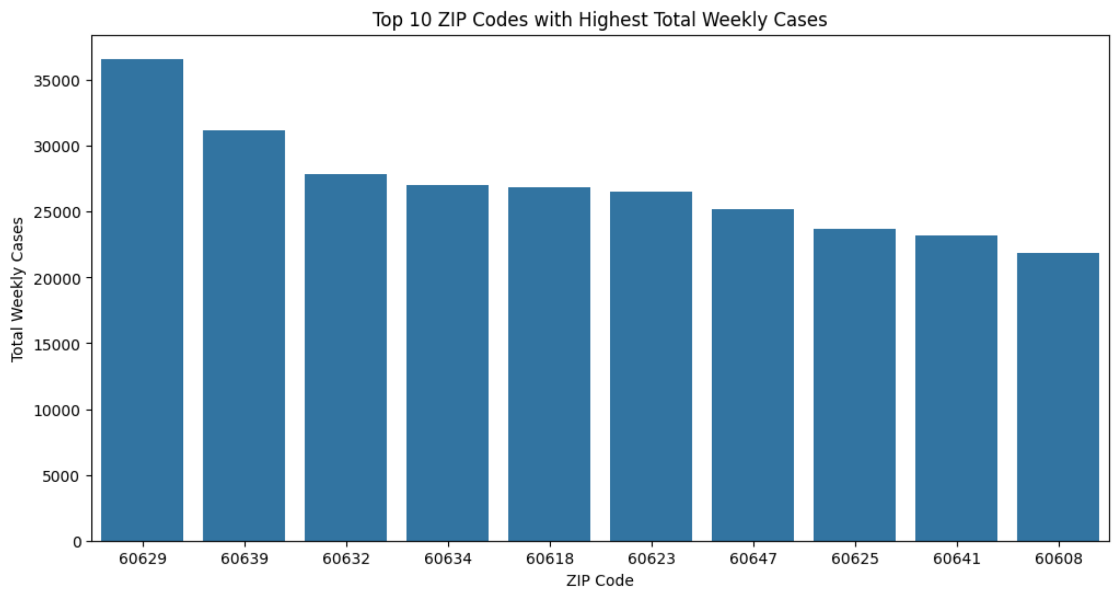
```
Weekly"]) > threshold].tolist()
high_corr_features.remove("Cases - Weekly")
print(high_corr_features)
```

```
['Week Number', 'Cases - Cumulative', 'Case Rate - Weekly', 'Case Rate -
Cumulative', 'Tests - Weekly', 'Tests - Cumulative', 'Test Rate - Weekly',
'Test Rate - Cumulative', 'Percent Tested Positive - Weekly', 'Percent Tested
Positive - Cumulative', 'Deaths - Weekly', 'Deaths - Cumulative', 'Death Rate
- Weekly', 'Population', 'Week Start Year']
```

```
# Cases over time
cases_over_time = df.groupby('Week Start')['Cases -
Weekly'].sum().reset_index()

plt.figure(figsize=(14, 6))
sns.lineplot(data=cases_over_time, x='Week Start', y='Cases - Weekly')
plt.title("Weekly COVID-19 Cases Over Time")
plt.xlabel("Week Start")
plt.ylabel("Total Cases - Weekly")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




```
top_zip = df.groupby("ZIP Code")["Cases -
Weekly"].sum().sort_values(ascending=False).head(10)

plt.figure(figsize=(12, 6))
sns.barplot(x=top_zip.index.astype(str), y=top_zip.values)
plt.title("Top 10 ZIP Codes with Highest Total Weekly Cases")
plt.xlabel("ZIP Code")
plt.ylabel("Total Weekly Cases")
plt.show()
```




```
X_selected = df[high_corr_features]
Y = df['Cases - Weekly']

X_train, X_test, y_train, y_test = train_test_split(X_selected, Y,
test_size=0.2, random_state=42)

model1 = GradientBoostingRegressor(learning_rate=0.1, max_depth=3,
min_samples_leaf=1, min_samples_split=2, n_estimators=100)
model1.fit(X_train, y_train)

y_pred = model1.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print(f"{model1.__class__.__name__}")
print(f"MSE: {mse}")
print(f"R2 Score: {r2 * 100:.2f}%"+'\n')
```

```
GradientBoostingRegressor
MSE: 516.9064574625049
MAE: 6.29

R2 Score: 96.98%
```

```
param_dist = {
    "n_estimators": [50, 100, 200, 500],
    "learning_rate": [0.01, 0.05, 0.1, 0.2],
    "max_depth": [3, 4, 5, 6, 7],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
```

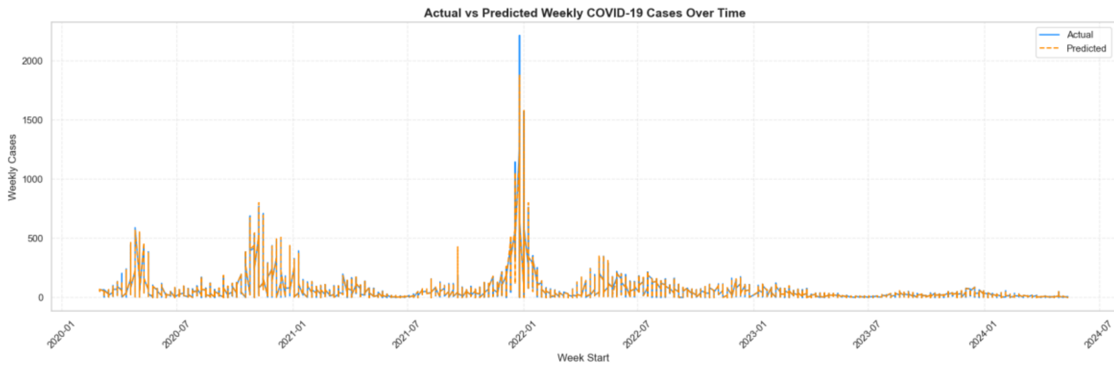
```
    "subsample": [0.7, 0.8, 0.9, 1.0],
    "max_features": ["sqrt", "log2", None]  # Removed 'auto'
}

model = GradientBoostingRegressor(random_state=42)

#RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=model,
    param_distributions=param_dist,
    n_iter=50,
    cv=5,
    scoring='r2',  # Optimize for R2 score
    n_jobs=-1,  # Use all available CPU cores
    verbose=2,
    random_state=42
)

random_search.fit(X_train, y_train)

best_params = random_search.best_params_
print(f"Best Hyperparameters: {best_params}")

best_model = GradientBoostingRegressor(**best_params)
best_model.fit(X_train, y_train)

# Evaluate
y_pred_best = best_model.predict(X_test)
mse_best = mean_squared_error(y_test, y_pred_best)
r2_best = r2_score(y_test, y_pred_best)
mae_best = mean_absolute_error(y_test, y_pred_best)


print("Tuned Gradient Boosting Regressor")
print(f"MSE: {mse_best:.2f}")
print(f"R2 Score: {r2_best * 100:.2f}%")
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 500,
'min_samples_split': 5, 'min_samples_leaf': 4, 'max_features': None,
'max_depth': 4, 'learning_rate': 0.05}
Tuned Gradient Boosting Regressor
MSE: 476.47751213733517
MAE: 3.70

R2 Score: 97.22%
```

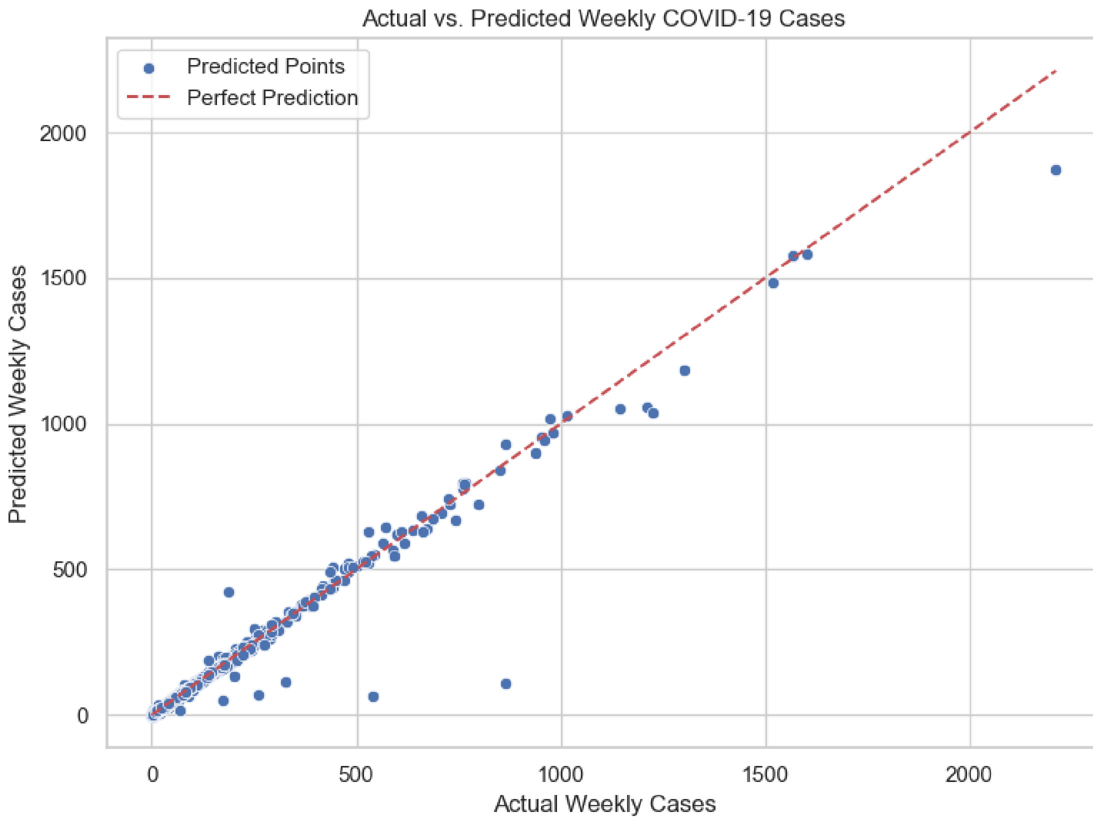
```
y_test_reset = y_test.reset_index()
```

```
y_pred_series = pd.Series(y_pred_best, index=y_test_reset.index,
name='Predicted')

# Combine Week Start with actual and predicted values
comparison_df = pd.DataFrame({
    'Week Start': df.loc[y_test_reset['index'], 'Week Start'].values,
    'Actual': y_test.values,
    'Predicted': y_pred_series.values
})

# Sort by date to
comparison_df = comparison_df.sort_values(by='Week Start')

sns.set(style='whitegrid')

plt.figure(figsize=(18, 6))

# Smoother lines using alpha
plt.plot(comparison_df['Week Start'], comparison_df['Actual'],
label='Actual', color='dodgerblue', linewidth=1.5)
plt.plot(comparison_df['Week Start'], comparison_df['Predicted'],
label='Predicted', color='darkorange', linewidth=1.5, linestyle='--')

plt.title(' Actual vs Predicted Weekly COVID-19 Cases Over Time',
fontsize=14, fontweight='bold')
plt.xlabel('Week Start', fontsize=12)
plt.ylabel('Weekly Cases', fontsize=12)

# Rotate and limit ticks for clarity
plt.xticks(rotation=45)
plt.locator_params(axis='x', nbins=12)

plt.grid(True, linestyle='--', alpha=0.4)
plt.legend()
plt.tight_layout()
plt.show()
```

```
C:\Users\Dinid\AppData\Local\Temp\ipykernel_16052\3504365313.py:28:
UserWarning: 'set_params()' not defined for locator of type <class
'matplotlib.dates.AutoDateLocator'>
  plt.locator_params(axis='x', nbins=12)
```




```python
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred_best, label='Predicted Points')
plt.plot(
    [y_test.min(), y_test.max()],
    [y_test.min(), y_test.max()],
    'r--',
    label='Perfect Prediction'
)
plt.xlabel('Actual Weekly Cases')
plt.ylabel('Predicted Weekly Cases')
plt.title('Actual vs. Predicted Weekly COVID-19 Cases')
plt.legend()
plt.tight_layout()
plt.show()
```


Actual vs. Predicted Weekly COVID-19 Cases
Predicted Points
Perfect Prediction
Predicted Weekly Cases
Actual Weekly Cases
0
500
1000
1500
2000
0
500
1000
1500
2000