BUSN 32120

COVID-19 U.S. Data Analysis

Final Project

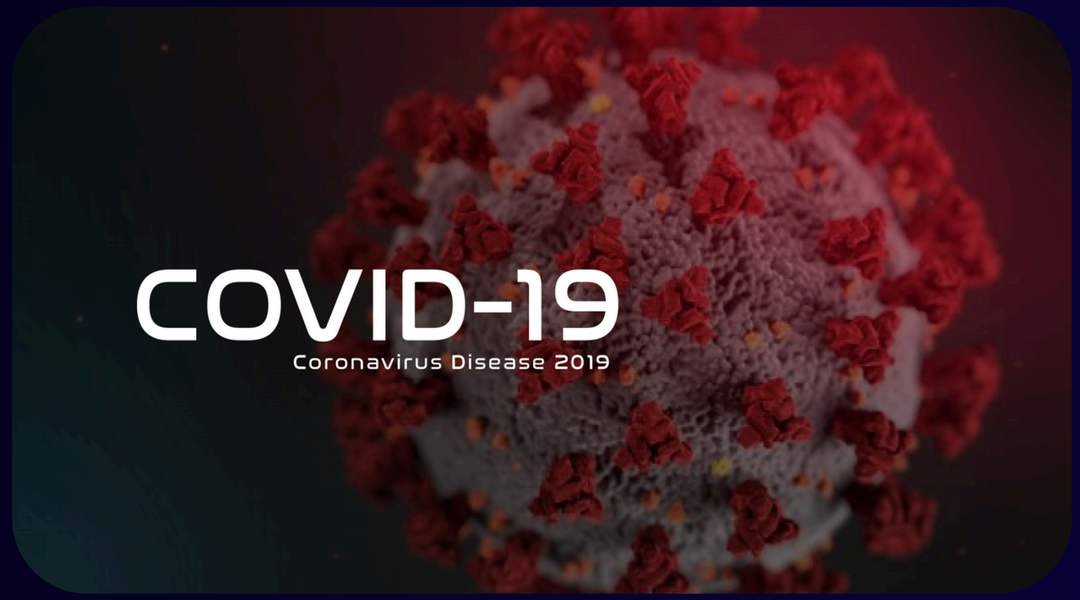
By: Bisma Rana

-

Table of Contents

Introduction

This analysis explores COVID-19 case numbers, deaths, and vaccination rates across U.S. states. The objective is to understand how vaccination coverage influenced COVID-19 fatality outcomes and to potentially uncover patterns that highlight disparities in healthcare access across different regions. To do this, I used two datasets from CDC public APIs.



Target Audience

- Public Health Officials
- Healthcare Policymakers
- Healthcare Practitioners and Administrators

Data Cleaning & Preprocessing

DC API used for fetching two datasets of COVID-19

```python
# Define Function
def pull_api_data(api_url):
    response = requests.get(api_url)
    if response.status_code == 200:
        data = response.json()
        df = pd.DataFrame(data)
        print(f"Successfully pulled {len(df)} records from API")
        return df
    else:
        print(f"Failed to pull data. Status code: {response.status_code}")
        return None

# Define API Endpoints

# ~COVID-19 case and death data by state~
cases_deaths_api = 'https://data.cdc.gov/resource/pwn4-m3yp.json?$limit=50000'
# ~COVID-19 vaccination data by state~
vaccinations_api = 'https://data.cdc.gov/resource/unsk-b7fc.json?$limit=50000'

# Pull Data Into DataFrames
df_cases_deaths = pull_api_data(cases_deaths_api)
df_vaccinations = pull_api_data(vaccinations_api)
```

- Created a function **pull_api_data()** to fetch data from CDC APIs using **requests.get()** and convert the response to a DataFrame.

Check for Null Values

```python
# Print Header to Label Output Section
print("\nMissing Values in Cases & Deaths - ", end="")

# Count Missing Values for Each Column in Cases & Deaths Dataset
missing = df_cases_deaths.isnull().sum()

# Print Number of Columns with At Least One Missing Value
print(len(missing[missing > 0]), "Columns with missing values: \n")

# Show Actual Columns and Number of Values Missing in Each
print(missing[missing > 0])
```

- Uses **df.isnull().sum()** to count missing values in each column of the cases & deaths dataset.
- Prints the number of columns that have missing data.

Relevant Columns Only

```python
# Keep Relevant Columns Only
df_cases_deaths = df_cases_deaths[['state', 'start_date' , 'end_date', 'tot_cases', 'tot_deaths', 'new_cases', 'new_deaths']]

# Rename Columns for Clarity
df_cases_deaths = df_cases_deaths.rename(columns={
    'end_date': 'date',
    'tot_cases': 'total_cases',
    'tot_deaths': 'total_deaths'
})

# Convert Date to Datetime
df_cases_deaths['date'] = pd.to_datetime(df_cases_deaths['date'], errors='coerce')
```

- Selected relevant and rename using **df.rename()** for clearer names.
- Convert dates using **pd.to_datetime()** for proper date handling (errors='coerce').

```python
# Convert Vaccination Columns to Numeric, Force Errors to NaN
cols_to_convert = [
    'fully_vaccinated_pct',
    'one_dose_pct',
    'booster_pct',
    'fully_vaccinated_65plus_pct',
    'booster_65plus_pct'
]

for col in cols_to_convert:
    df_vaccinations[col] = pd.to_numeric(df_vaccinations[col], errors='coerce')

# Create 'Week' Column
df_vaccinations['week'] = df_vaccinations['date'] - pd.to_timedelta(df_vaccinations['date'].dt.weekday, unit='d') + pd.Timedelta(days=2) #Realign to Wednesday

# Group by 'State' and 'Week' and Take Mean
df_vaccinations_weekly = df_vaccinations.groupby(['state', 'week']).agg({
    'fully_vaccinated_pct': 'mean',
    'one_dose_pct': 'mean',
    'booster_pct': 'mean',
    'fully_vaccinated_65plus_pct': 'mean',
    'booster_65plus_pct': 'mean'
}).reset_index()

# Rename 'Week' Column Back to Date
df_vaccinations_weekly = df_vaccinations_weekly.rename(columns={'week': 'date'})
```

- Converts vaccination columns to numeric with **pd.to_numeric()** and creates a Wednesday-aligned weekly date using **pd.to_timedelta()** and **pd.Timedelta()**.

- Groups by state and week with **.groupby()** and **.agg('mean')**, then renames the grouped week column to 'date' using **.rename()**

Aggregate Vaccination Data to Weekly Level

Check for Outliers

```python
numeric_cols = ['total_cases', 'total_deaths', 'new_cases', 'new_deaths']

# Convert Columns to Numeric
for col in numeric_cols:
    df_cases_deaths[col] = pd.to_numeric(df_cases_deaths[col], errors='coerce')

# Create Boxplot
for col in numeric_cols:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x=df_cases_deaths[col], color='orange')
    plt.title(f'Boxplot of {col}')
    plt.xlabel(col)
    plt.show()
```

- Converts specified columns to numeric using **pd.to_numeric()** with error coercion to handle invalid data
- Generates boxplots for each numeric column with **sns.boxplot()** to visually identify outliers and distribution.

Handle Outliers

```python
# Filter to Check Rows with 'Negative' New Deaths
df_cases_deaths[df_cases_deaths['new_deaths'] < 0]

# Manually Inspect Random Rows
df_cases_deaths.sample(10)

# Convert Negative Values in New Deaths to Positive (Absolute Value)
df_cases_deaths['new_deaths'] = abs(df_cases_deaths['new_deaths'])

# Recheck to Confirm
df_cases_deaths[df_cases_deaths['new_deaths'] < 0]

# View Sample
df_cases_deaths.sample(5)
```

- Uses filtering and sampling **df_cases_deaths.sample()** to identify and inspect negative new_deaths values as outliers.
- Corrects outliers by applying the absolute value function **abs()** to new_deaths, then verifies the fix by re-filtering and sampling the cleaned data.

- Connects to an SQLite database (or creates it if not exists) using **sqlite3.connect()**.
- Saves dataframes **df_cases_deaths** and **df_vaccinations_weekly** into tables **(cases_deaths and vaccinations_weekly)** with **.to_sql()**, replacing existing tables if needed.
- Closes the connection with **conn.close()** and confirms successful storage with a print statement.

```python
# Store Data into SQLite Database
import sqlite3

# Define Database File Name
db_filename = "covid_analysis.db"

# Connect to SQLite Database
conn = sqlite3.connect(db_filename)

# Store Cases & Deaths Dataframe as Table
df_cases_deaths.to_sql("cases_deaths", conn, if_exists="replace", index=False)

# Store Weekly Aggregated Vaccinations Dataframe as Table
df_vaccinations_weekly.to_sql("vaccinations_weekly", conn, if_exists="replace", index=False)

# Close Database Connection
conn.close()

# Confirm Storage Completion
print(f"Data stored in {db_filename} (tables: cases_deaths, vaccinations_weekly)")
```

Store Cleaned Datasets into SQLite Database

Merging Datasets on State and Date

```python
# Merging on State and Date Using an Inner Join
df_merged = pd.merge(df_cases_deaths, df_vaccinations_weekly, on=['state', 'date'], how='inner')

# View Shape and Sample of Merged Dataset
print("Merged dataset shape:", df_merged.shape)
print("\nSample of merged data:")
display(df_merged.head())

# Check for Missing Values After Merging
print("\nMissing values after merging:")
print(df_merged.isnull().sum())
```

- **Supports deeper insights into how vaccination coverage relates to COVID-19 case and death trends over time and across different states.**

Key Findings

- Merged cases, deaths, and vaccination data on state and date — resulting in a clean, unified dataset with **7,127** records across key public health metrics.
- Vaccination metrics are consistently populated; missing values in **booster_pc**t appear only in early periods, reflecting the later rollout of booster campaigns.
- Final dataset is complete and analysis-ready — ideal for exploring relationships between vaccination coverage and COVID-19 outcomes over time.

Feature Engineering

Case Fatality Rate

Calculated as (total_deaths / total_cases) * 100, this metric indicates the severity of COVID-19 by showing the proportion of confirmed cases that resulted in death.

```
# Convert 'total_cases' and 'total_deaths' to Numeric
df_merged['total_cases'] = pd.to_numeric(df_merged['total_cases'], errors='coerce')
df_merged['total_deaths'] = pd.to_numeric(df_merged['total_deaths'], errors='coerce')

df_merged['case_fatality_rate'] = (df_merged['total_deaths'] / df_merged['total_cases']) * 100
```

Feature Engineering

Vaccination Coverage Category

astype(float) converts values to float type, and **apply(categorize_vaccination)** runs the function on each value to categorize vaccination coverage into 'High', 'Medium', or 'Low'.

```python
def categorize_vaccination(coverage_pct):
    if coverage_pct >= 70:
        return 'High'
    elif coverage_pct >= 50:
        return 'Medium'
    else:
        return 'Low'

df_merged['vaccination_category'] = df_merged['fully_vaccinated_pct'].astype(float).apply(categorize_vaccination)
```

Feature Engineering

Elderly Protection Index

```
df_merged['elderly_protection_index'] = (
    df_merged['fully_vaccinated_65plus_pct'].astype(float) + df_merged['booster_65plus_pct'].astype(float)
) / 2
```

Calculates the elderly protection index as the average of **fully_vaccinated_65plus_pct** and **booster_65plus_pct** (converted to **float**), indicating overall vaccine coverage among seniors.

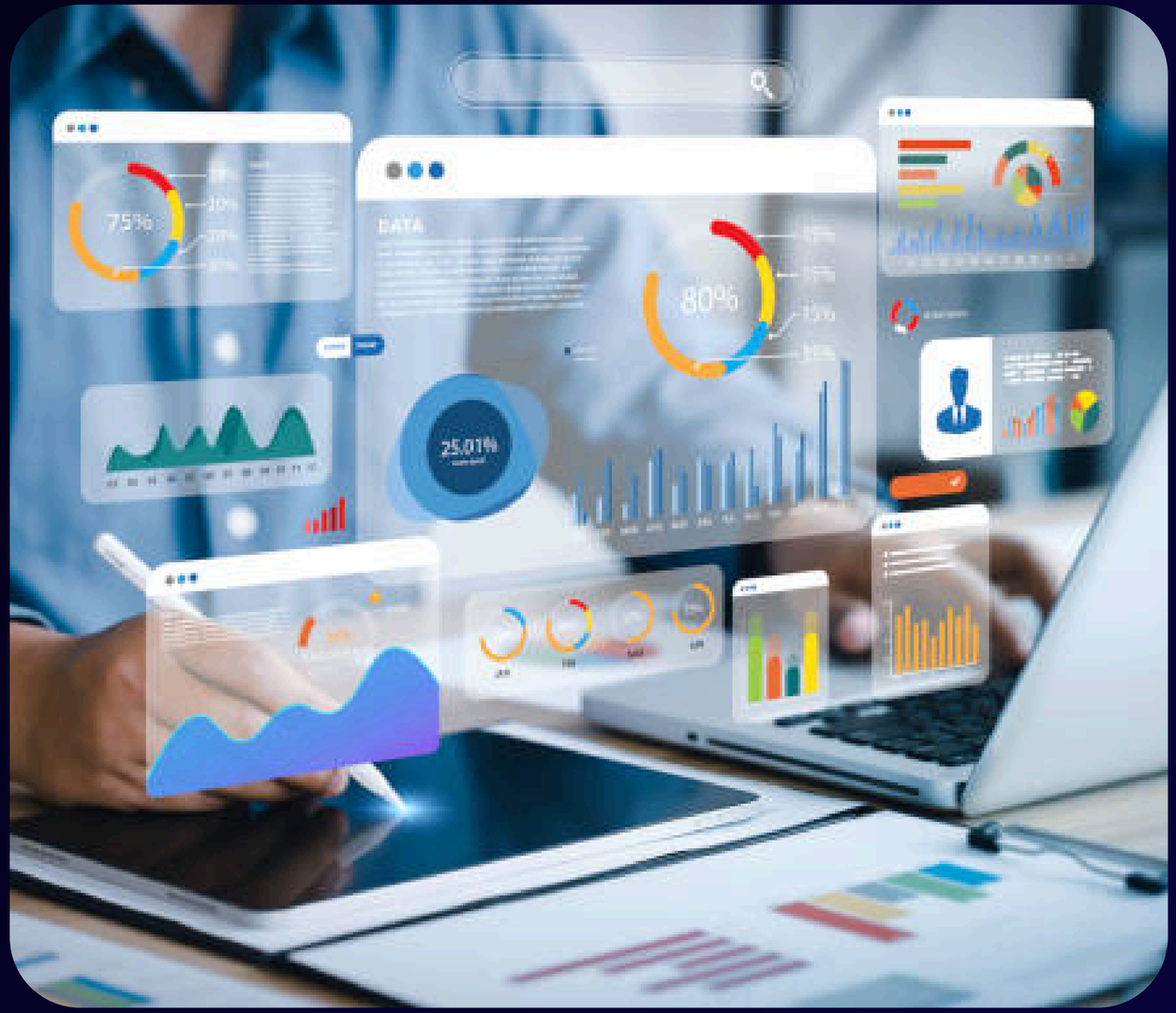
Feature Engineering

Fatality Risk Classification

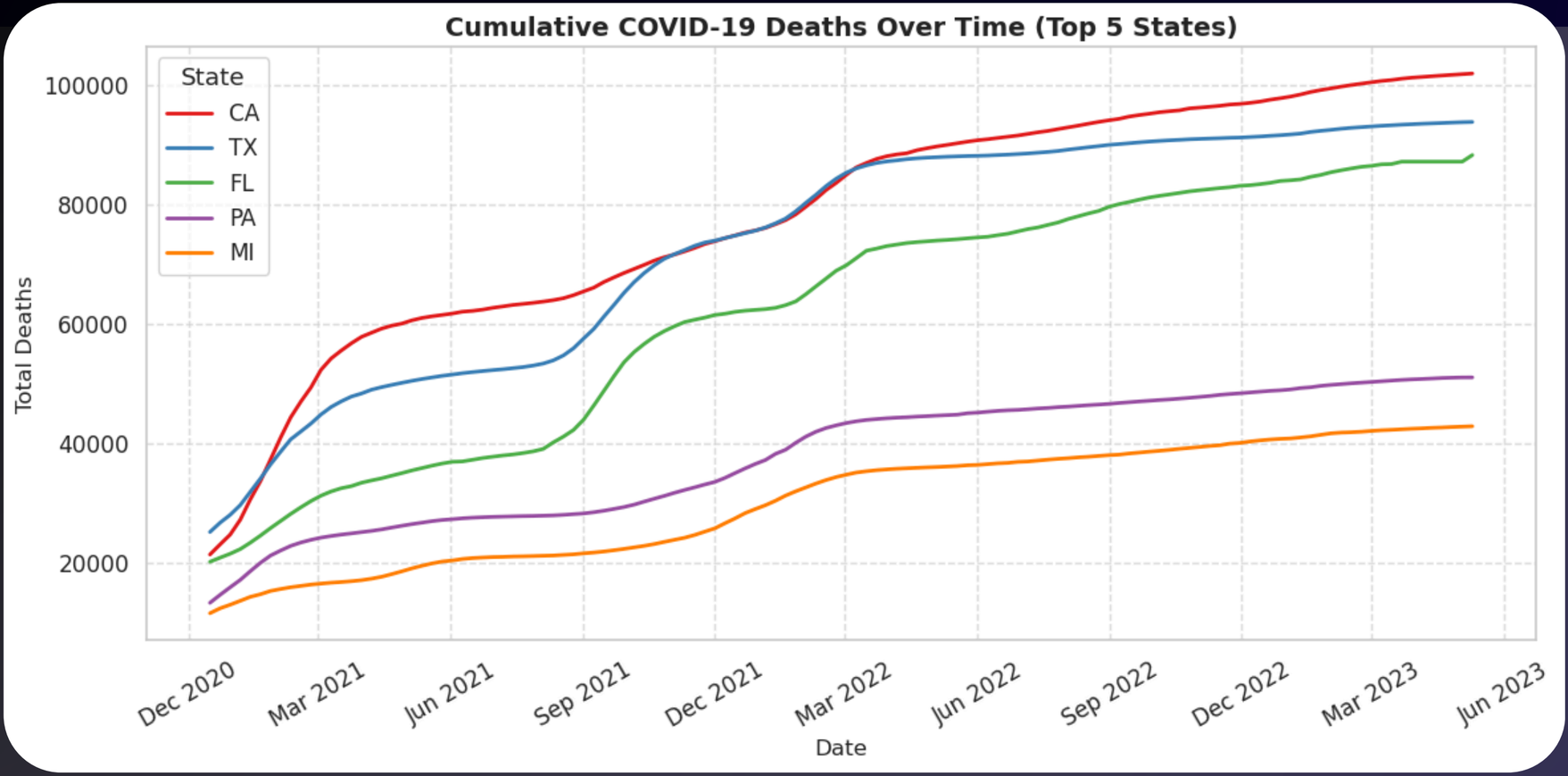
```python
def classify_fatality(r):
    if r < 0.9:
        return 'Low'
    elif r < 1.52:
        return 'Moderate'
    else:
        return 'High'

df_merged["fatality_risk"] = df_merged["case_fatality_rate"].apply(classify_fatality)
```

This code creates a new column **fatality_risk** by applying the **classify_fatality()** function to each row's case_fatality_rate, classifying it as 'Low', 'Moderate', or 'High' based on thresholds.

Exploratory Data Analysis



Most Important Insights



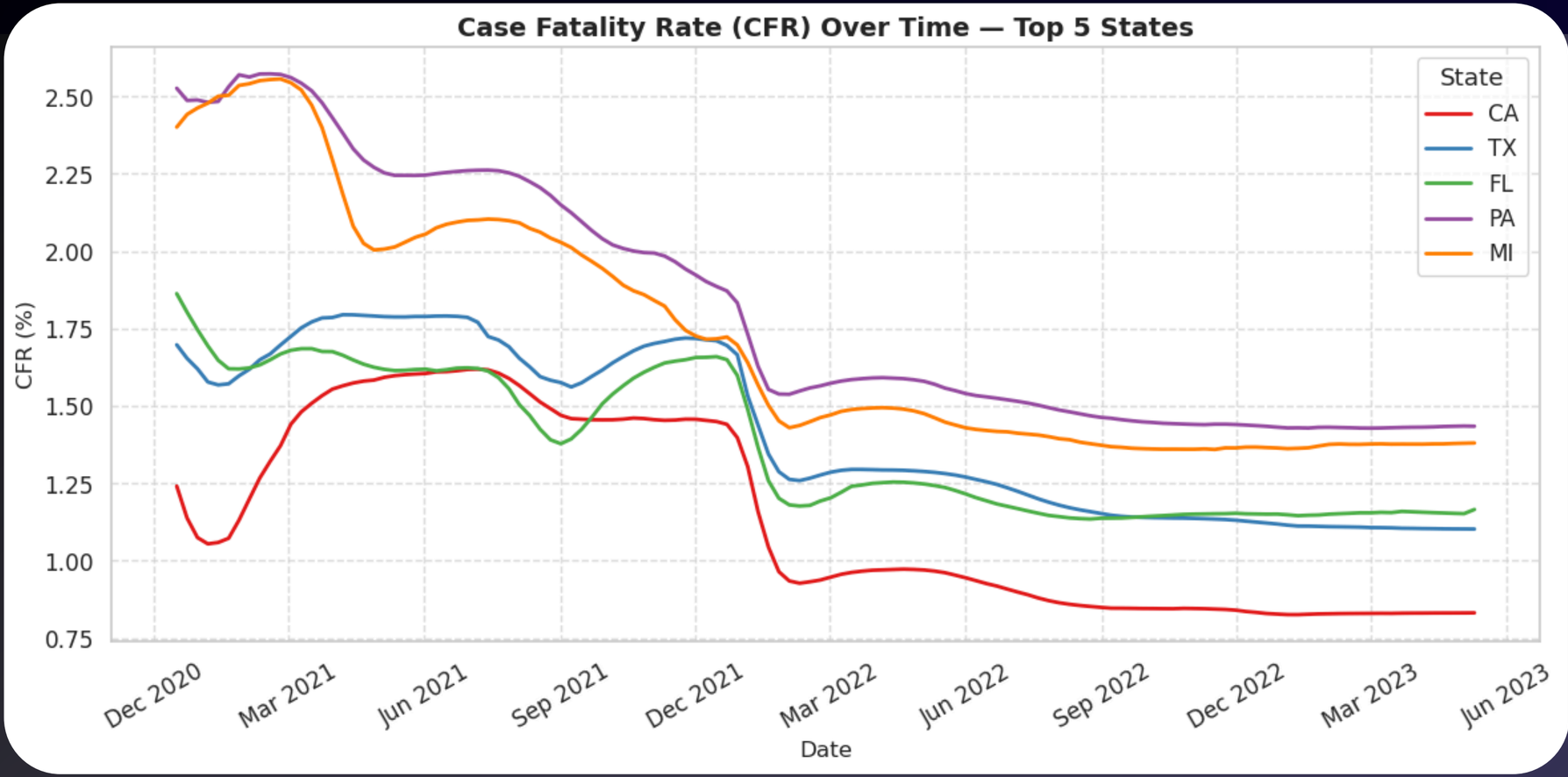
- California and Texas had the highest total COVID-19 deaths, reflecting large population sizes and overall burden.
- Cumulative death totals don't fully capture outbreak severity relative to case counts.
- All states showed steady increases in deaths over time, highlighting persistent vulnerability.

```python
import matplotlib.dates as mdates
plt.figure(figsize=(12, 6))

# Define Top 5 States
top5_states = ['CA', 'TX', 'FL', 'PA', 'MI']

# Plot Total Deaths Over Time
for state in top5_states:
    subset = df_merged[df_merged['state'] == state]
    plt.plot(
        subset['date'],
        subset['total_deaths'],
        linewidth=2,
        label=state
    )

# Add Title and Labels
plt.title('Cumulative COVID-19 Deaths Over Time (Top 5 States)', fontsize=14, fontweight='bold')
plt.xlabel('Date', fontsize=12)
plt.ylabel('Total Deaths', fontsize=12)

# Add Legend, Grid, and Layout
plt.legend(title='State')
plt.grid(True, linestyle='--', alpha=0.6)
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=3))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
plt.xticks(rotation=30)
plt.tight_layout()

# Show Plot
plt.show()
```

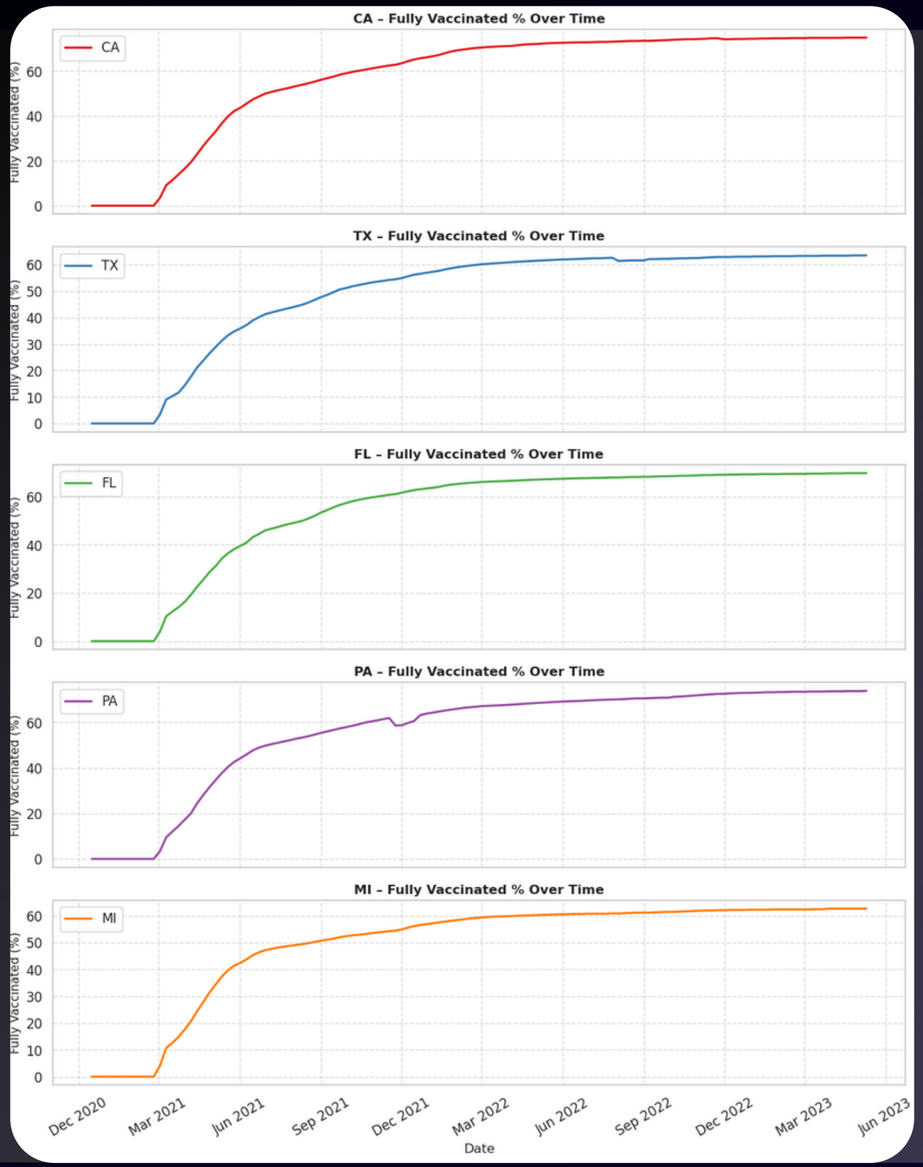
- Uses a for loop to go through each state in top5_states and plots a line for its total_deaths over time with plt.plot().

- Uses **mdates.MonthLocator()** and **mdates.DateFormatter('%b %Y')** to format the x-axis dates in clean 3-month intervals (like "Jan 2021").

- Improves readability with **plt.title()**, **plt.legend()**, **plt.grid()**, and **plt.tight_layout()** so the chart looks clean and easy to follow.

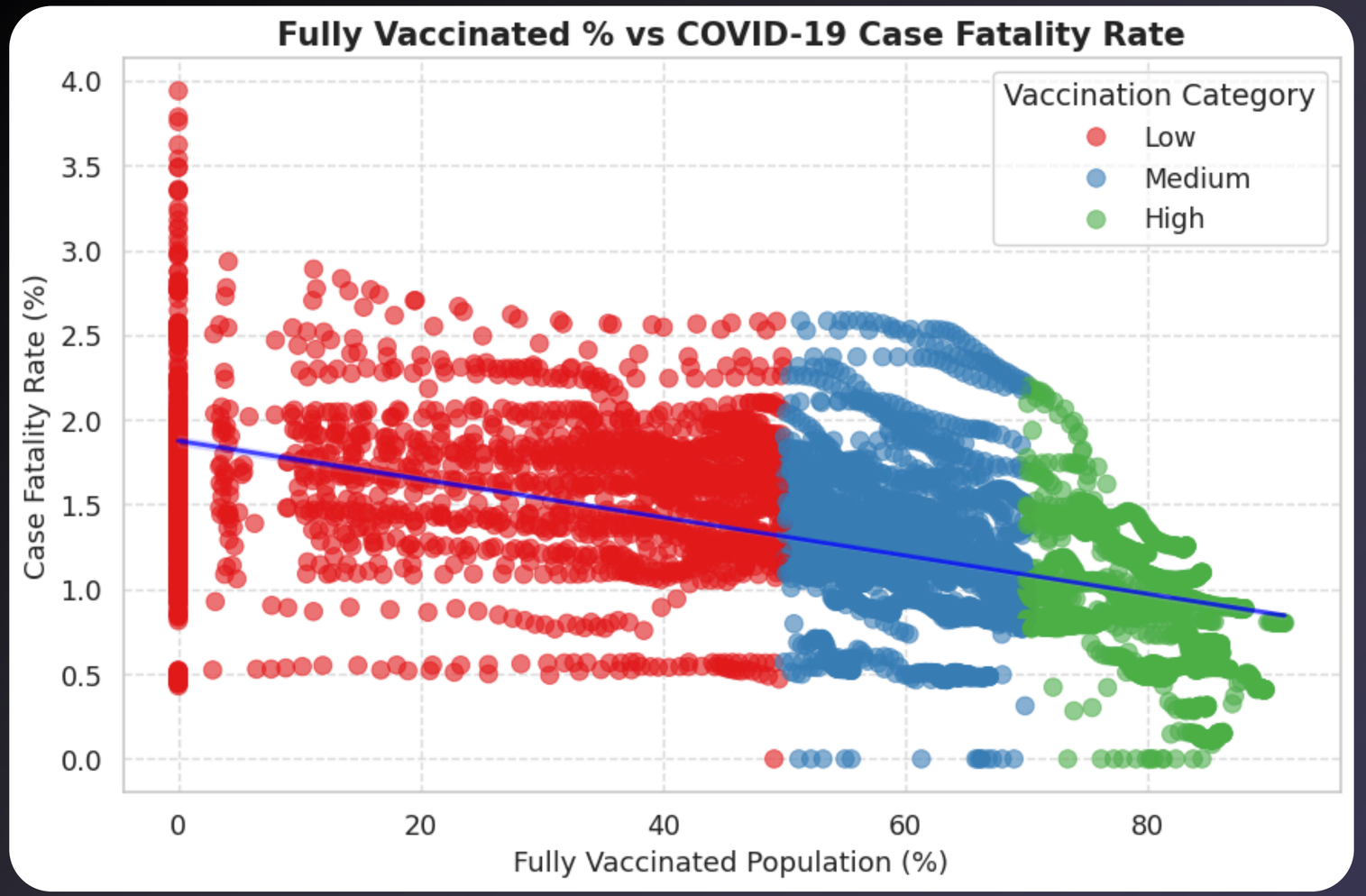


- CFR is a better measure than death counts alone, showing how deadly the virus was relative to infections.
- CFR decreased over time in all states, likely due to better treatments and healthcare response.
- Pennsylvania and Michigan started with higher CFRs (>2.5%), while California maintained the lowest fatality rate.



Fully Vaccinated % Over Time

- All five states show a clear increase in vaccination rates starting early 2021, marking the mass vaccine rollout.
- **CA** and **PA** achieved the highest vaccination rates (**~75%**), while **TX**, **FL**, and MI plateaued around **60–65%.**
- Higher vaccination rates, especially in **CA** and **PA**, correlate with lower case fatality rates (CFR), indicating vaccines helped reduce fatality risk.

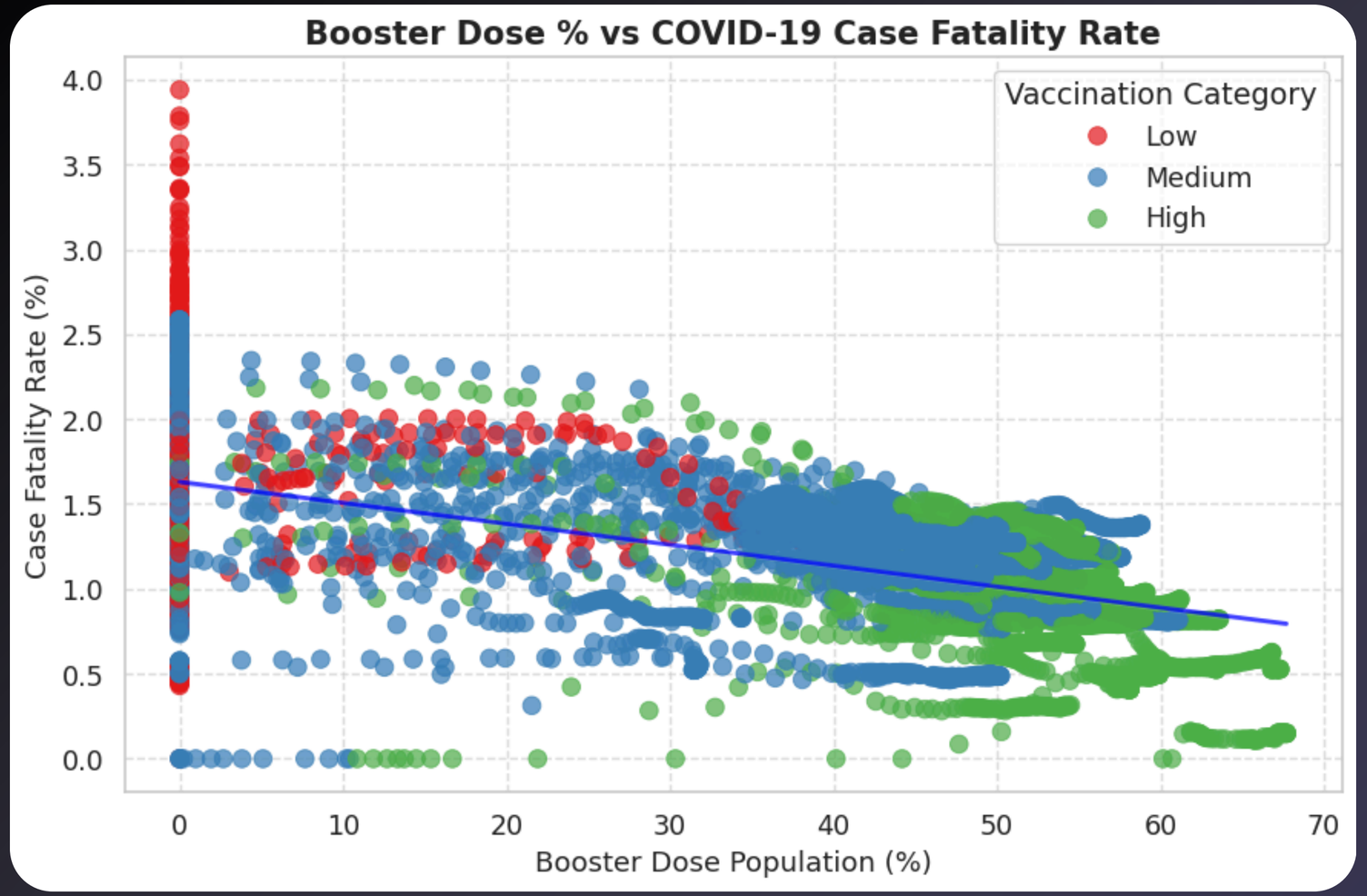


- Increasing vaccination rates are linked to a decrease in case fatality rates (CFR), showing a modest negative correlation.
- Once vaccination surpasses 60–70%, CFR mostly stays below 1%, indicating stronger protection against severe outcomes.

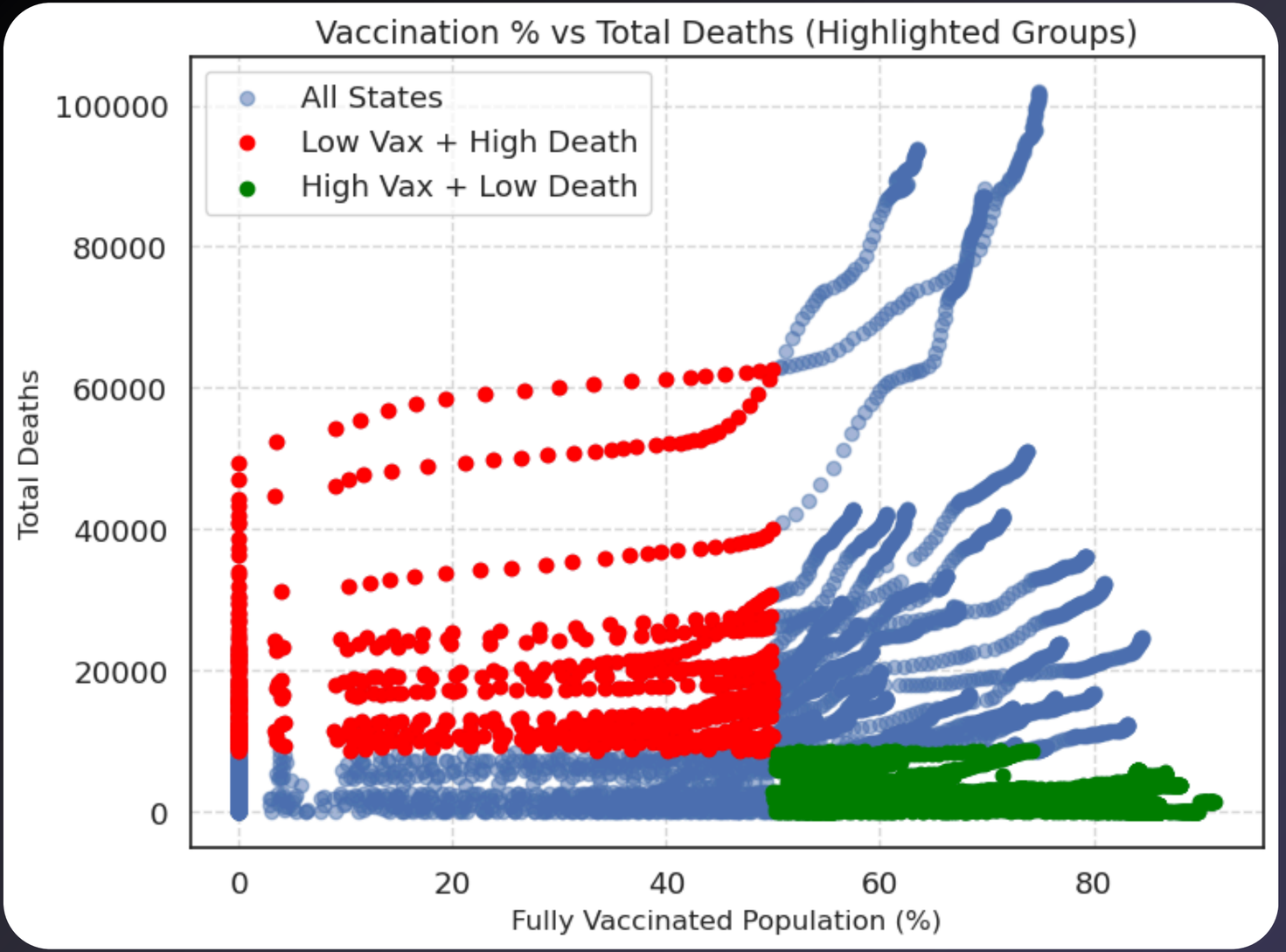
```python
# Set Up Figure and Style
plt.figure(figsize=(9, 6))
sns.set(style="whitegrid", font_scale=1.15)

# Plot Scatter Points by Vaccination Category
sns.scatterplot(
    data=df_merged,
    x='fully_vaccinated_pct',
    y='case_fatality_rate',
    hue='vaccination_category',
    palette='Set1',
    alpha=0.6,
    s=70,
    edgecolor=None
)

# Add Regression Line
sns.regplot(
    data=df_merged,
    x='fully_vaccinated_pct',
    y='case_fatality_rate',
    scatter=False,
    color='blue',
    line_kws={'linewidth': 2, 'alpha': 0.7}
)

# Add Title, Labels, Legend, and Grid
plt.title('Fully Vaccinated % vs COVID-19 Case Fatality Rate', fontsize=15, fontweight='bold')
plt.xlabel('Fully Vaccinated Population (%)', fontsize=13)
plt.ylabel('Case Fatality Rate (%)', fontsize=13)
plt.legend(title='Vaccination Category')
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()

# Show Plot
plt.show()
```

- Use **sns.scatterplot()** to visualize the relationship between vaccination % and fatality rate, with points colored by vaccination category.

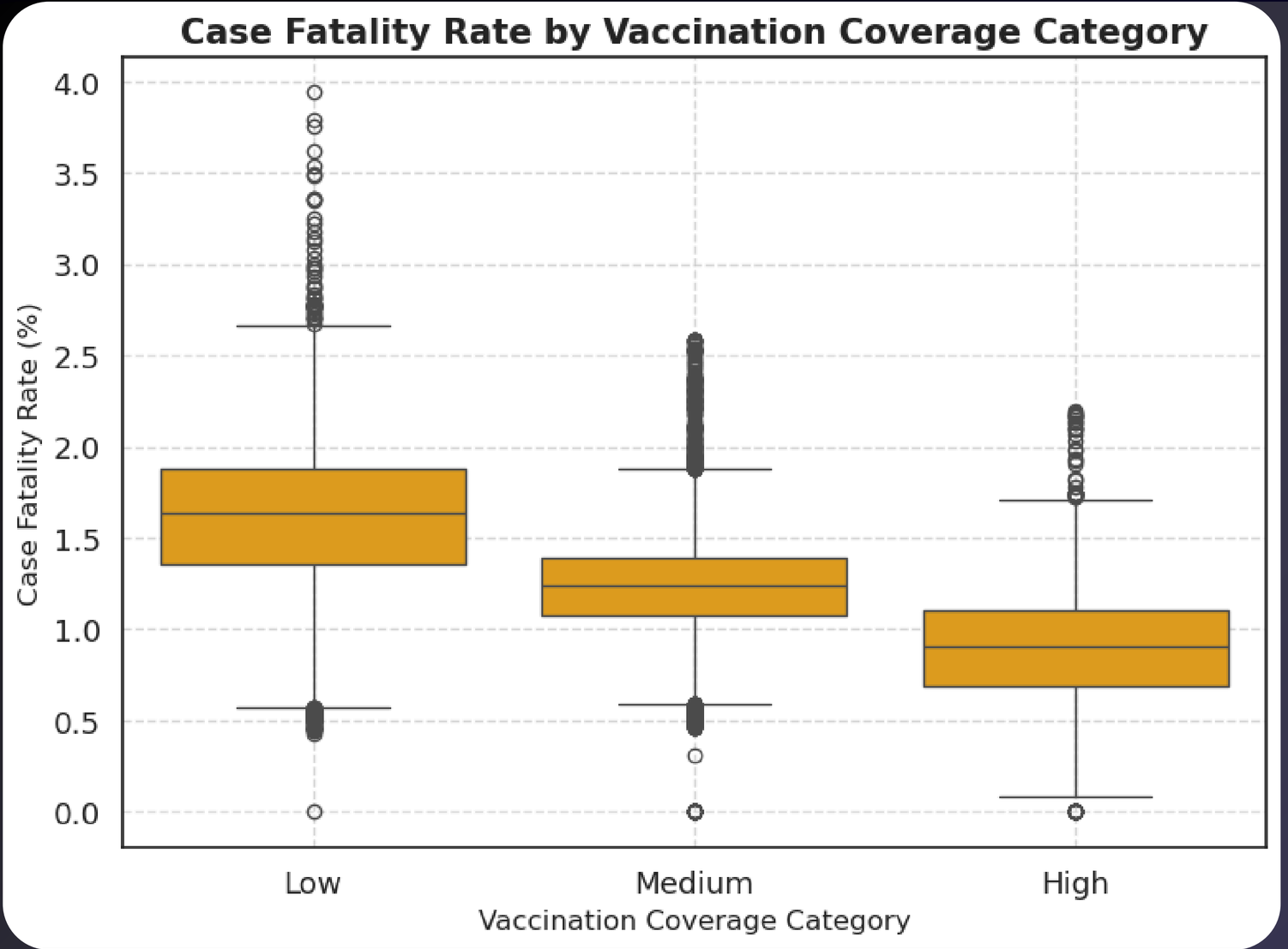
- Add a trend line using **sns.regplot()** to show the overall correlation, and customize the plot with titles, labels, and grid for clarity.



- Higher booster coverage tends to link with lower case fatality rates, but the relationship is weak and scattered.
- When booster coverage exceeds **~30–40%**, CFR values often cluster below 1%, indicating improved protection.
- Low booster coverage (**<20%**) shows high CFR variability, suggesting other factors also impact fatality rates.



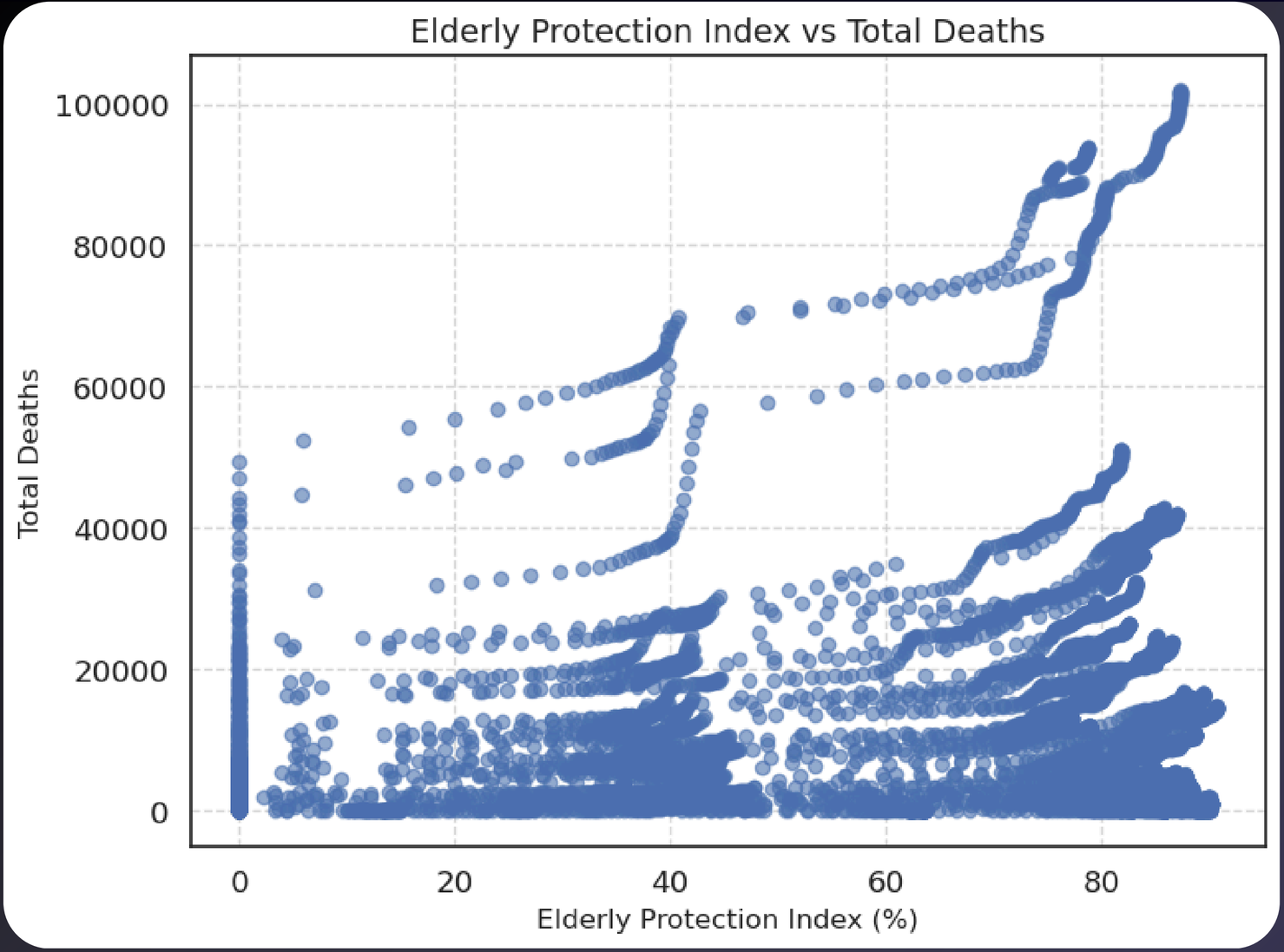
- Low vaccination + high death states (red) cluster at lower vaccination rates and higher total deaths.
- High vaccination + low death states (green) cluster at higher vaccination rates and lower death counts.
- All other states (blue) scatter between the two extremes, indicating that vaccination was important but other factors (population size, health access) also influenced deaths.



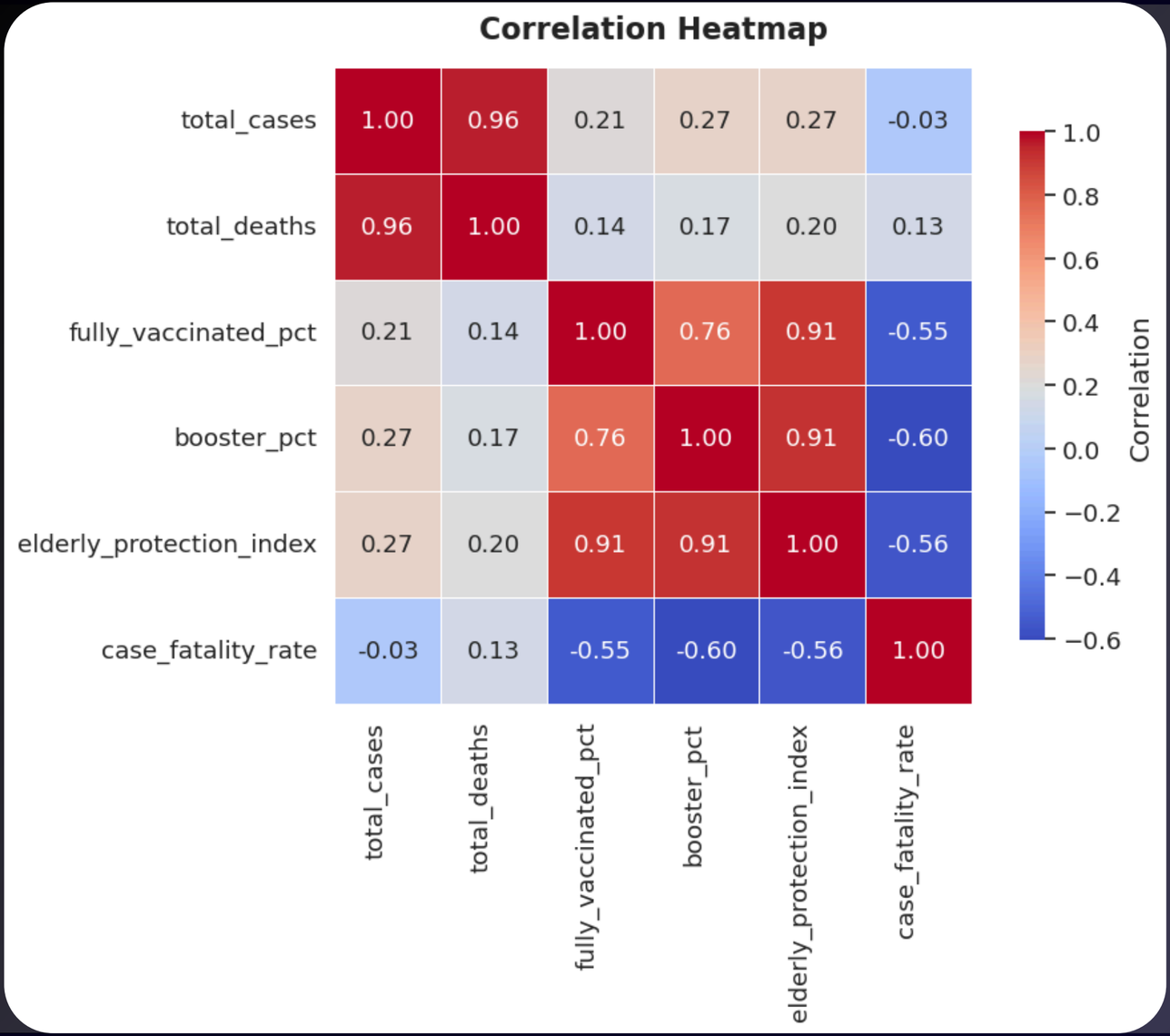
- States with **high** vaccination coverage show lower median CFRs and fewer extreme outliers.
- **Low** coverage states have higher median CFRs and more variability, reflecting worse outcomes.
- Overall, CFR tends to **decrease** as vaccination coverage **improves** from Low to High.

```python
# Create Figure
plt.figure(figsize=(8,6))

# Draw Boxplot
sns.boxplot(
    x='vaccination_category',
    y='case_fatality_rate',
    data=df_merged,
    order=['Low', 'Medium', 'High'],
    color='orange'  # <-- manually specifying order
)

# Add Title and Labels
plt.title('Case Fatality Rate by Vaccination Coverage Category', fontsize=15, fontweight='bold')
plt.xlabel('Vaccination Coverage Category', fontsize=12)
plt.ylabel('Case Fatality Rate (%)', fontsize=12)

# Add Grid and Layout
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()

# Show Plot
plt.show()
```

- Use **sns.boxplot()** to compare the distribution of case fatality rates across vaccination coverage levels (Low, Medium, High).
- The **order** parameter ensures consistent category sorting on the x-axis.
- Add titles, labels, and gridlines to clearly present the relationship between vaccination and fatality outcomes.



- States with higher elderly protection (**60%–90%**) still experienced a spread in total deaths, implying that other factors like outbreak size, timing, or healthcare access mattered.



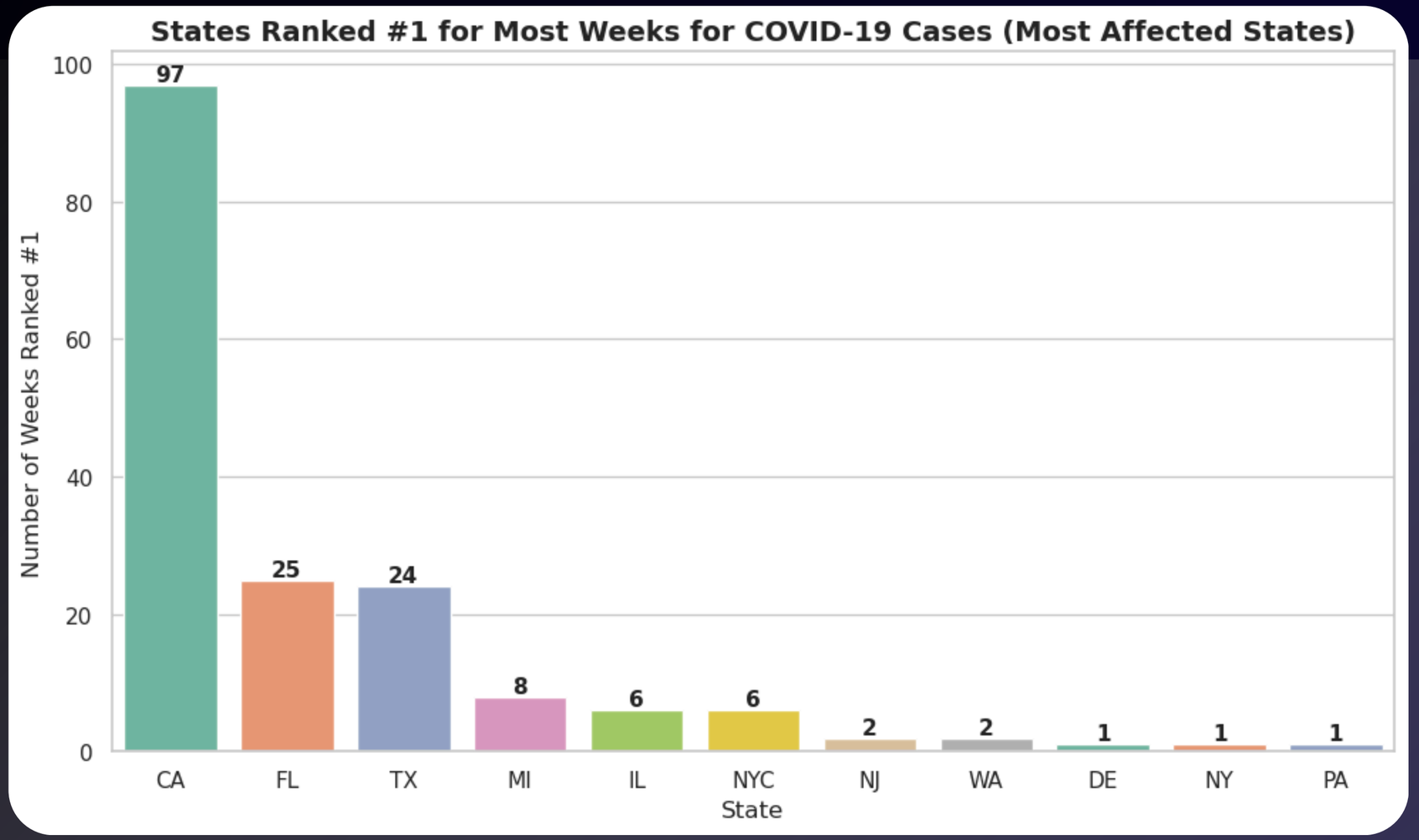
- Vaccination metrics (full vax %, booster %, elderly protection) are strongly correlated (**0.76–0.91**), showing states with strong initial rollout continued strong coverage.
- CFR has moderate negative correlation (~ **-0.55** to **-0.60**) with vaccination, supporting its role in reducing death severity.
- Total cases and total deaths are highly correlated (**0.96**), reflecting the overall impact of widespread infections.

```python
# Pick Numeric Columns Only
corr_columns = [
    'total_cases',
    'total_deaths',
    'fully_vaccinated_pct',
    'booster_pct',
    'elderly_protection_index',
    'case_fatality_rate'
]

# Calculate Correlation Matrix
corr_matrix = df_merged[corr_columns].corr()

# Create Heatmap with Seaborn Styling
plt.figure(figsize=(10, 8))
sns.set(style="white", font_scale=1.2)

ax = sns.heatmap(
    corr_matrix,
    annot=True,
    cmap='coolwarm',
    fmt='.2f',
    linewidths=0.5,
    square=True,
    cbar_kws={'shrink': 0.8, 'label': 'Correlation'},
    annot_kws={"size": 13}
)

# Add Title and Ticks
ax.set_title('Correlation Heatmap', fontsize=16, fontweight='bold', pad=16)
plt.yticks(rotation=0)
plt.xticks(rotation=90, ha='right')
plt.tight_layout()

# Show Plot
plt.show()
```

- Select relevant numeric columns and use **.corr()** to compute the correlation matrix.
- Visualize the matrix with **sns.heatmap()**, showing relationships between variables with color and annotated values.
- Customize the plot (title, labels, color scale) for readability and professional presentation.

SQL Analysis

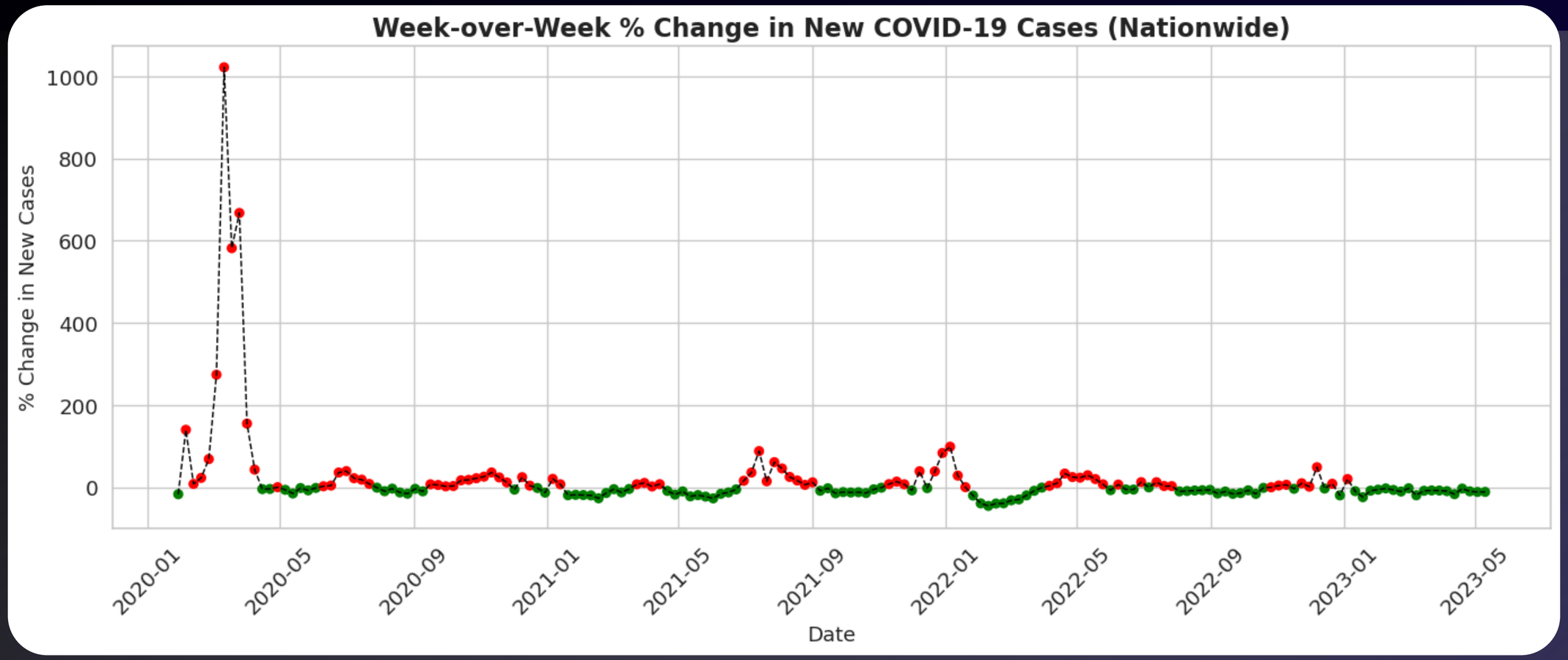
Most Important Insights



- California (CA), Florida (FL), and Texas (TX) were the most affected states.
- They ranked #1 in weekly death counts for 97, 25, and 24 weeks respectively.

```sql
query = """
-- QUERY 4: Rank states by new cases per date using window function, only extract rank 1
SELECT
    state,
    COUNT(date) AS num_weeks_rank1
FROM (
    SELECT
        state,
        date,
        new_cases,
        RANK() OVER (PARTITION BY date ORDER BY new_cases DESC) AS rank_by_cases
    FROM cases_deaths
)
WHERE rank_by_cases = 1
GROUP BY state
ORDER BY num_weeks_rank1 DESC, state;
"""

ranked_cases = pd.read_sql(query, conn) # Check unique states with rank 1
ranked_cases.to_csv("sql_results/query_4.csv", index=False) # Save to CSV for further analysis
ranked_cases
```

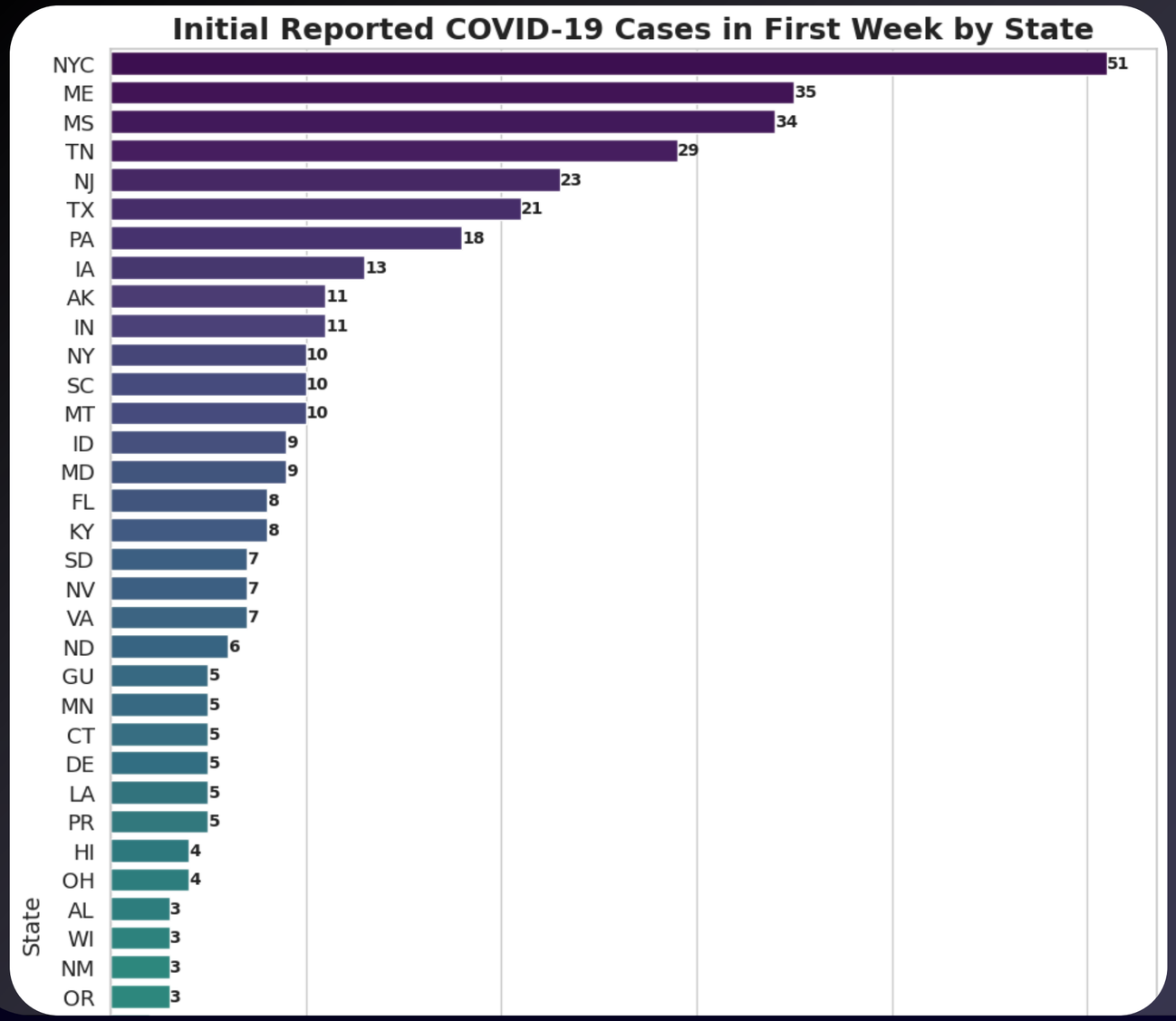
- Use a SQL window function (**RANK() OVER**) to rank states by **new_cases** for each date.
- Filter for **rank = 1** to find which state had the most cases per week.
- Group and count results in Python with **pd.read_sql()** and save to **CSV** for analysis.



- Early 2020 saw extreme spikes in weekly case growth (1000%+), driven by rapid initial spread.
- After mid-2020, % changes stabilized to -30% to +30%, except during variant waves (Delta, Omicron).
- By 2022–23, WoW changes flattened, reflecting controlled spread due to vaccines and prior exposure.

```sql
query = """
-- QUERY 5: National week-over-week % change in total new cases

WITH weekly_totals AS (
    SELECT
        date,
        SUM(new_cases) AS total_new_cases
    FROM cases_deaths
    GROUP BY date
),
weekly_change AS (
    SELECT
        date,
        total_new_cases,
        LAG(total_new_cases) OVER (ORDER BY date) AS previous_total_cases
    FROM weekly_totals
)
SELECT
    date,
    total_new_cases,
    previous_total_cases,
    ROUND(
        CASE
            WHEN previous_total_cases = 0 THEN NULL
            ELSE ((total_new_cases - previous_total_cases) * 100.0 / previous_total_cases)
        END, 2
    ) AS percent_change
FROM weekly_change
ORDER BY date;
"""

national_wow_df = pd.read_sql(query, conn)
national_wow_df.to_csv("sql_results/query_5.csv", index=False)  # Save to CSV for further an
alysis
national_wow_df.head(20)

```

- Use CTEs to first sum new cases by date, creating weekly totals (**weekly_totals**).
- Calculate the previous week's total using SQL's window function **LAG()** to compare week-over-week case numbers.
- Compute the percentage change between weeks, handling division by zero safely, then load the results into pandas and save for further analysis.



- New York City had the highest initial cases (**51**), showing an early intense outbreak.
- Maine, Mississippi, Tennessee, and New Jersey also had high first-week cases, indicating early spread.
- Many states like California, Michigan, Georgia, and District of Columbia started with low cases, while Washington, Arizona, Illinois, Massachusetts, and North Carolina reported just **1** case, suggesting isolated introductions.

```
query = """
-- QUERY 6: First week with > 0 new cases per state


WITH filtered_data AS (
    SELECT *
    FROM cases_deaths
    WHERE new_cases > 0
),
ranked_first_case AS (
    SELECT
        state,
        date,
        new_cases,
        ROW_NUMBER() OVER (PARTITION BY state ORDER BY date) AS rn
    FROM filtered_data
)
SELECT
    state,
    date AS first_reported_week,
    new_cases AS initial_reported_cases
FROM ranked_first_case
WHERE rn = 1
ORDER BY first_reported_week;
"""

first_nonzero_df = pd.read_sql(query, conn)
first_nonzero_df.to_csv("sql_results/query_6.csv", index=False)  # Save
to CSV for further analysis
first_nonzero_df.head()
```

- Filter the dataset to only include weeks with new cases (**new_cases > 0**).
- Use **ROW_NUMBER()** window function, partitioned by state, ordered by date, to find each state's first week with reported cases.
- **Select** and **order** results by the first reported week, then load into **pandas** and save as **CSV** for further analysis.

States with Below-Average Vaccination and Above-Average Deaths

	state	avg_vax	avg_deaths_per_week
0	Texas	48.9%	555
1	Georgia	43.9%	260
2	Ohio	47.9%	258
3	Michigan	49.4%	255
4	Arizona	50.0%	209
5	Tennessee	43.7%	193
6	North Carolina	49.9%	180
7	Indiana	45.4%	152
8	Alabama	40.9%	136
9	Kentucky	47.3%	130
10	Missouri	45.6%	129
11	South Carolina	46.0%	121
12	Oklahoma	46.4%	110

- States like Texas, Georgia, Alabama, and Tennessee show low vaccination rates combined with high death rates.
- Texas has the lowest vaccination rate (49%) and highest weekly deaths (555).
- These areas are high-risk and may need stronger public health interventions.

```sql
query = """
-- QUERY 6: First week with > 0 new cases per state


WITH filtered_data AS (
    SELECT *
    FROM cases_deaths
    WHERE new_cases > 0
),
ranked_first_case AS (
    SELECT
        state,
        date,
        new_cases,
        ROW_NUMBER() OVER (PARTITION BY state ORDER BY date) AS rn
    FROM filtered_data
)
SELECT
    state,
    date AS first_reported_week,
    new_cases AS initial_reported_cases
FROM ranked_first_case
WHERE rn = 1
ORDER BY first_reported_week;
"""

first_nonzero_df = pd.read_sql(query, conn)
first_nonzero_df.to_csv("sql_results/query_6.csv", index=False)  # Save to CSV for further analysis
first_nonzero_df.head()
```

- Join vaccination and case data on state and date, then compute **average vaccination %** and deaths per state using **AVG()**
- Filter states with below-average vaccination and above-average deaths using subqueries.
- Rank by deaths (**ORDER BY**), load with **pd.read_sql()**, and save to CSV for analysis.

Weeks with ≥ 50% Spike in New Cases

	date	total_new_cases	prev_week_cases	percent_change
0	2020-02-05	12	5	140%
1	2020-02-26	27	16	69%
2	2020-03-04	101	27	274%
3	2020-03-11	1133	101	1022%
4	2020-03-18	7725	1133	582%
5	2020-03-25	59286	7725	667%
6	2020-04-01	151264	59286	155%
7	2021-07-14	251109	133430	88%
8	2021-07-28	464744	288157	61%
9	2021-12-29	2141043	1165714	84%
10	2022-01-05	4258203	2141043	99%

- Weeks with over 50% increase in new cases highlight significant COVID-19 transmission rebounds.
- The largest spike occurred on March 11, 2020, with a 1021% increase.

```sql
query = """
-- QUERY 11: COVID rebound alert — spikes of 50%+ in national new cases

WITH daily_totals AS (
    SELECT
        date,
        SUM(new_cases) AS total_new_cases
    FROM cases_deaths
    GROUP BY date
),
with_lagged AS (
    SELECT
        date,
        total_new_cases,
        LAG(total_new_cases) OVER (ORDER BY date) AS prev_week_cases
    FROM daily_totals
),
spike_alerts AS (
    SELECT
        date,
        total_new_cases,
        prev_week_cases,
        ROUND(((total_new_cases - prev_week_cases) * 100.0 / prev_week_cases), 2) AS percent_change
    FROM with_lagged
    WHERE prev_week_cases IS NOT NULL
)

SELECT
    SUBSTR(date, 1, 10) AS date,  -- Only keep YYYY-MM-DD
    total_new_cases,
    prev_week_cases,
    percent_change
FROM spike_alerts
WHERE percent_change >= 50
ORDER BY date;
"""

spike_alerts_df = pd.read_sql(query, conn)
spike_alerts_df.to_csv("sql_results/query_11.csv", index=False)  # Save to CSV for further analysis
spike_alerts_df
```

- Uses **CTEs** to first sum daily new cases (**daily_totals**), then add previous day's cases with the **LAG()** window function (**with_lagged**).
- Calculates percentage change in new cases and filters for spikes of **50%** or more (**spike_alerts**).
- Selects and formats key columns, ordering results by date to identify significant COVID-19 rebounds.

Machine Learning Model

Linear Regression

Predict Weekly New COVID-19 Cases Based on Vaccination Coverage

Goal

To estimate how different levels of vaccination coverage impact the number of new COVID-19 cases reported in a given week across U.S. states.

Importing scikit-learn, Data Preparation and Feature Engineering

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Select Relevant Features and Drop Nulls
model_df = df_merged[
    ["new_cases", "new_deaths", "fully_vaccinated_pct",
     "booster_pct", "one_dose_pct", "date", "elderly_protection_index"]
].dropna()

# Create a 'Week' Feature to Encode Time Progression
model_df["week"] = model_df["date"].dt.isocalendar().week

# Define X and y
X = model_df.drop(columns=["new_cases", "date"])
y = model_df["new_cases"]

# Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran
dom_state=42, shuffle=True)
```

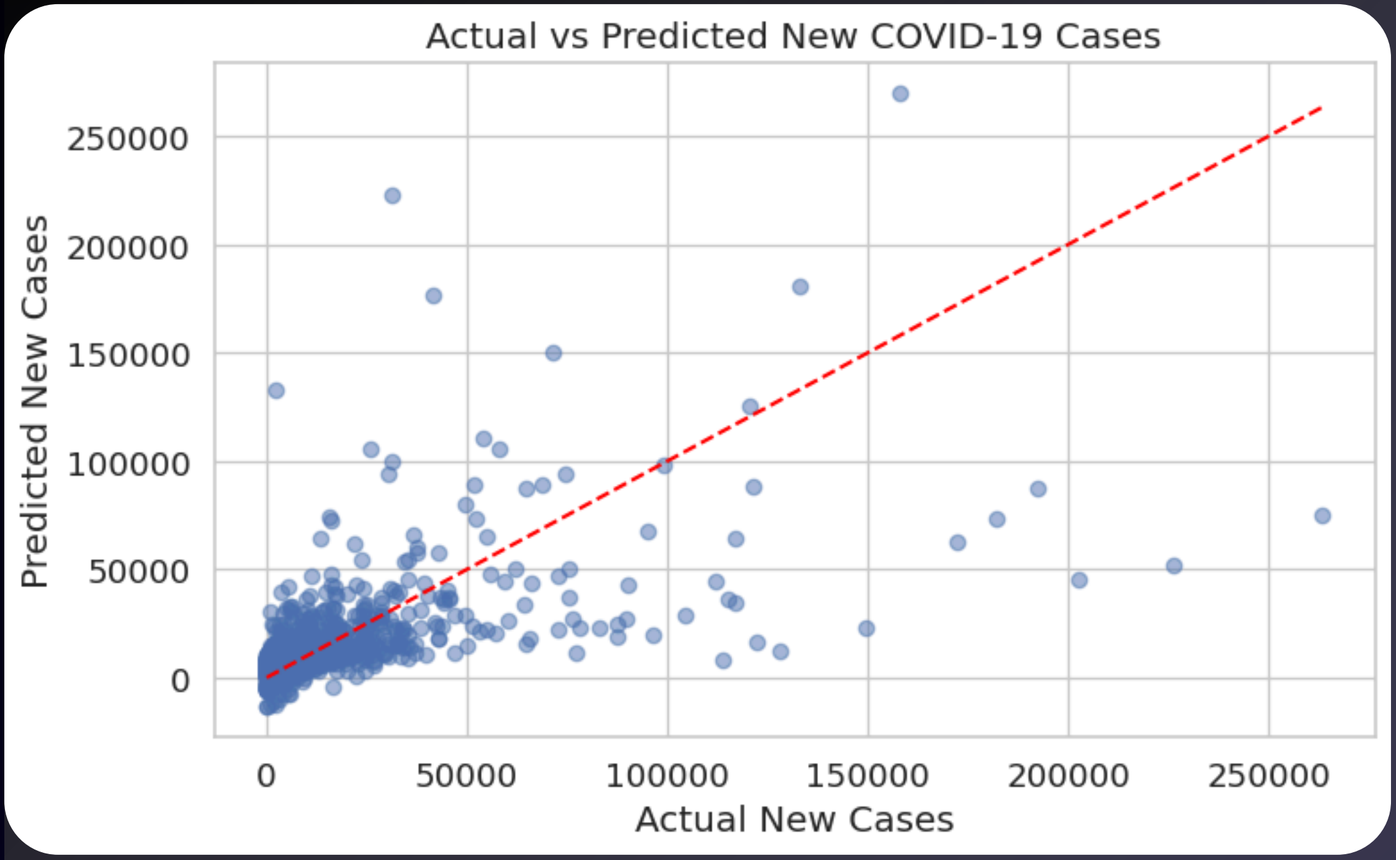
- Selected features include vaccination rates **(fully_vaccinated_pct, booster_pct, one_dose_pct), elderly_protection_index**, **week** (from date), and encoded state data.
- These variables were chosen to reflect vaccine-driven immunity and regional variation over time.
- Defined **X** by dropping new_cases and date; **y** was set as new_cases.
- Used **train_test_split()** to split the data into training and test sets **(80/20)** with **shuffling**.

Model Training, Predictions and Evaluating Metrics

```python
# Instantiate and Train Linear Regression Model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Predict on Test Data
y_pred = lr.predict(X_test)

# Evaluate Model using R^2 and RMSE
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f"R² Score: {r2:.3f}")
print(f"RMSE: {rmse:.2f}")
```

- Used **LinearRegression()** from scikit-learn to model the relationship between vaccine features, time (week), and new COVID-19 cases.
- Trained the model with **.fit()** and made predictions on test data using **.predict()**.
- Evaluated performance using **r2_score** and mean_squared_error (converted to RMSE with **np.sqrt**).
- Output metrics (R^2, RMSE) indicate how accurately the model captures case trends.

Actual vs Predictions Scatter Plot



- The model explains about **29%** of case variation (R^2 = **0.291**) with a moderate average error of **~18,283** cases (RMSE), reflecting limited predictive power due to real-world data variability.

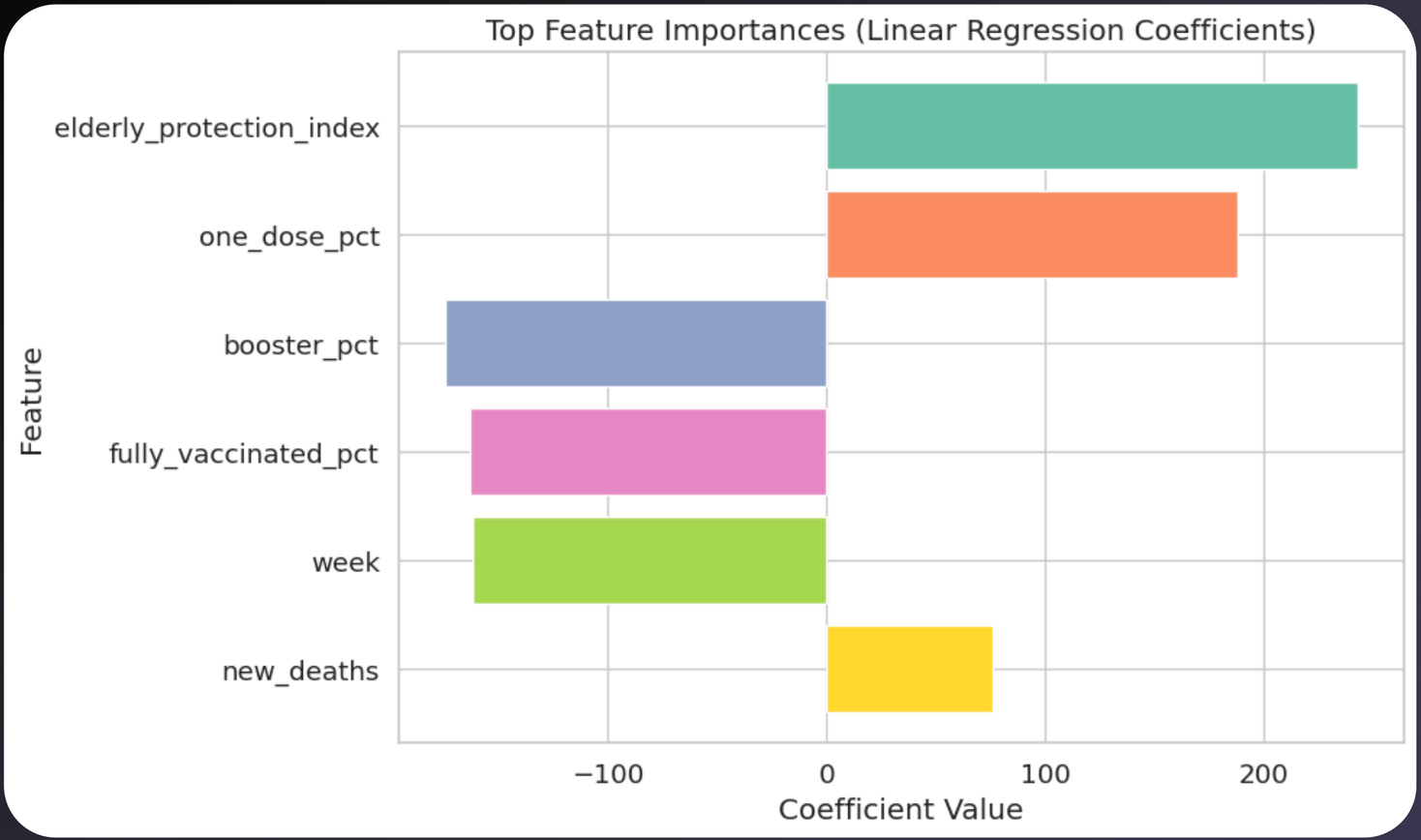
- Most predictions are clustered near the bottom left (**0–50,000 range**), suggesting the model captures low/moderate case weeks better.

- However, it underperforms on high-case weeks, which drags down R^2. This is common in linear models when data has outliers or a nonlinear pattern.

```
plt.figure(figsize=(8, 5))

plt.scatter(y_test, y_pred, alpha=0.5)

plt.plot([0, max(y_test)], [0, max(y_test)], color='red', linestyle='--')

plt.xlabel("Actual New Cases")
plt.ylabel("Predicted New Cases")
plt.title("Actual vs Predicted New COVID-19 Cases")
plt.tight_layout()

plt.show()
```

- **plt.scatter()** plots actual vs predicted cases with transparency (alpha=0.5) for better visibility of overlapping points.
- **plt.plot()** line adds a red dashed line representing perfect predictions where values match exactly.
- plt.xlabel(), plt.ylabel(), and plt.title() add descriptive labels and a title; plt.tight_layout() adjusts spacing; plt.show() displays the final plot.

Feature Importance (Coefficient Plot)



- **Booster** and **full vaccination** rates show **negative coefficients**, indicating they help reduce predicted COVID-19 cases.
- The **negative** week coefficient suggests cases tend to **decline** over time.
- **Positive** coefficients for **elderly protection**, **one-dose percentage**, and new deaths may reflect reactive trends or lagging indicators.

```python
# Create Coefficient DataFrame
coef_df = pd.DataFrame({
    "Feature": X.columns,
    "Coefficient": lr.coef_
}).sort_values(by="Coefficient", key=abs, ascending=False)

# Define Top N
top_n = 10
top_features = coef_df.head(top_n)

# Manually Assign Colors (length = top_n)
bar_colors = sns.color_palette("Set2", n_colors=top_n)

# Plot
plt.figure(figsize=(10, 6))
bars = plt.barh(
    y=top_features["Feature"],
    width=top_features["Coefficient"],
    color=bar_colors  # assign directly to matplotlib's barh
)

plt.title("Top Feature Importances (Linear Regression Coefficients)")
plt.xlabel("Coefficient Value")
plt.ylabel("Feature")
plt.gca().invert_yaxis()  # Highest at top
plt.tight_layout()
plt.show()
```

- Creates a DataFrame **coef_df** to pair features with their linear regression **coefficients**, sorting by absolute value for importance.
- Selects the top features (top_features) for visualization and assigns distinct colors using **Seaborn's "Set2" palette**.
- Uses **plt.barh()** to draw a horizontal bar chart of feature coefficients, inverts the y-axis to show highest importance on top, and adds labels and title for clarity.

Conclusion

- Higher vaccination linked to fewer cases; booster rates varied.
- Elderly protection reduces deaths.
- CA, FL, TX had highest deaths; CA led 97 weeks.
- Early spikes included 1000% surge in March 2020.
- NYC saw early outbreaks; some states had minimal initial cases.
- TX, GA, AL, TN showed low vaccination and high deaths.

- Model predicts low/moderate case weeks well but misses large surges. Booster and full vaccination rates lower predicted cases. Cases decline over time.
- Vaccination and time features alone aren't sufficient for a robust model, as many other factors (e.g., variants, mobility, policies) influence case trends.

The analysis used state-level data without population normalization or socioeconomic factors, limiting depth; future work could include classification, time series, or causal methods.

Thank You!