# Beyond The Pandemic: How Small-Cap Stocks Continue to Face Heightened Volatility

# Project 1

## 1.1 Introduction

The COVID-19 pandemic introduced uncertainty into financial markets, increasing volatility for large and small-cap stocks. Prior research has analyzed the effects of the pandemic on overall market fluctuations, investor sentiment, and asset pricing (Baker et al., 2020; Albulescu, 2020). One key measure of market fluctuations is the volatility gap, which measures the difference in market uncertainty between small-cap and large-cap stocks. Small-cap stocks tend to experience more significant swings during financial crises (Chen & Chiang, 2020). This study examines how COVID-19 severity and economic indicators impacted the volatility gap before, during, and after the pandemic.

The first part of the report studies stock market volatility using the Russell 2000 and S&P 500 as proxies for stock market volatility, categorizing the timeline into pre-pandemic, pandemic, and post-pandemic periods (Dong et al., 2020). The unemployment rates, inflation, and interest rates were used to represent economic indicators and were taken from FRED. The COVID-19 case data, particular case fatality ratio, and the 7-day moving average of new cases were used to assess how health and economic shocks affected market fluctuations. Findings indicated that the volatility gap peaked early in the pandemic, largely due to extreme uncertainty, liquidity constraints, and sudden shifts in investor risk perception (Mishra et al., 2020).

The second phase broadened the analysis by utilizing macroeconomic controls and sector-fixed effects, allowing for a deeper exploration. Results demonstrated that certain sectors, particularly energy, and industrials, resulted in significantly higher volatility gaps, portraying their exposure to supply chain disruptions and fluctuating demand (Hudepohl et al., 2021; Zhu et al., 2019). Previous literature suggests that contractionary monetary policies played a stabilizing role in broader markets, but small-cap stocks remained more vulnerable due to limited financial capacity and higher capital constraints (Karnizova & Li, 2014; Onan et al., 2014). By integrating economic and pandemic-related variables, this study expands on existing research on systemic crises and their disproportionate effects across asset classes, providing insights into how the volatility gap evolves in response to external shocks (Baig et al., 2020).

The final phase of the study utlizes machine learning methods to help discover non-linear patterns and interactive effects of the models. By webscrpaing COVID-19 sector news, a sentiment score was created using natural language processing tools and then merging thee data with volatility data across states. Regression trees and random forests were then trained on the same variables in the OLS regression in the secound phase. These models results suggest that market sentiment was the most important predictor, followed by COVID-related deaths and interest rates. The findings provide new insight into how market sentiment and health shocks both jointly influenced market behavior beyond what linear regressions could identify.

## How Did The Pandemic and Economic Indicators Impact the Volatility Gap Between Small-Cap and Large-Cap Stocks?

# 1. 2 Data Cleaning/Loading

In [54]:
```
!pip install --quiet yfinance
!pip install --quiet seaborn
!pip install --quiet ace_tools
!pip install --quiet fredapi
!pip install --quiet pandas requests
!pip install --quiet stargazer
!pip install --quiet GoogleNews newspaper3k pandas
!pip install --quiet vaderSentiment
!pip install --quiet lxml_html_clean
!pip install --quiet lxml --upgrade
!pip install --quiet imgkit



import yfinance as yf
import pandas as pd
from fredapi import Fred
from IPython.display import display, HTML
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
import seaborn as sns
import numpy as np
import re
import os
from io import StringIO
import requests
import imgkit
```

In [44]:
```
#The below code is to gather the daily returns and rolling volatility for an russell 2000 and S&P 500 to calculate
#Volailty gap to depict volatility gap between small cap and large cap.

# Function to fetch daily returns and rolling volatility for an index
def fetch_volatility_data(ticker, name):
    # Download historical data
    df = yf.download(ticker, start="2018-01-01", end="2024-12-31", interval="1d",progress = False)
    # Reset index to keep Date as a column
    df.reset_index(inplace=True)
    # Use "Adj Close" if available, otherwise use "Close"
    price_col = "Adj Close" if "Adj Close" in df.columns else "Close"
    # Compute Daily Returns
    df[f"Daily_Return_{name}"] = df[price_col].pct_change()
    # Compute rolling volatility (Standard Deviation of % Changes)
    df[f"30D_Volatility_{name}"] = df[f"Daily_Return_{name}"].rolling(window=30).std()
    df[f"7D_Volatility_{name}"] = df[f"Daily_Return_{name}"].rolling(window=7).std()

    # Keep only relevant columns
    return df[["Date", f"Daily_Return_{name}", f"30D_Volatility_{name}", f"7D_Volatility_{name}"]]
# Fetch Russell 2000 volatility data
```

```
russell_volatility = fetch_volatility_data("^RUT", "Russell2000")
# Fetch S&P 500 volatility data
sp500_volatility = fetch_volatility_data("^GSPC", "SP500")
# Merge both datasets on Date
combined_volatility = russell_volatility.merge(sp500_volatility, on="Date", how="inner")
# Calculate Absolute Daily Volatility Gap
combined_volatility["Daily_Volatility_Gap"] = (
    combined_volatility["7D_Volatility_Russell2000"] - combined_volatility["7D_Volatility_SP500"]
)
# Keep only required columns
final_data = combined_volatility[["Date",
                                  "Daily_Volatility_Gap", "30D_Volatility_Russell2000", "7D_Volatility_Russell2000"]]
# Save final dataset
csv_filename = "russell_volatility_daily_gap.csv"
final_data.to_csv(csv_filename, index=False)
```

In [8]:

```
#The Code below is to grab the data set from FRED(Federal Reserve Economic Data) to get
#Interest Rate, Unemployment, Inflation data

fred = Fred(api_key="d9ec7a0936c1955f1169751b3adecdef")  # This is my Fred API Key to access from jyputerhub

# ✅ Define economic indicators to fetch (2018-2024)
fred_series = {
    "Interest Rate (Fed Funds)": "FEDFUNDS",  # Federal Funds Rate (Monthly)
    "Unemployment Rate": "UNRATE",  # Unemployment Rate (Monthly)
    "Inflation (CPI)": "CPIAUCSL"  # Consumer Price Index (Monthly)
}
# ✅ Fetch data from FRED (LIMITED to 2018-2024)
df_fred = pd.DataFrame()
for name, series in fred_series.items():
    df_fred[name] = fred.get_series(series, start_date="2018-01-01", end_date="2024-12-31")
# ✅ Convert index to datetime format
df_fred.index = pd.to_datetime(df_fred.index)
df_fred.reset_index(inplace=True)
df_fred.rename(columns={"index": "Date"}, inplace=True)  # Standardize date column

# ✅ Categorize periods into Pre-Pandemic, Pandemic, and Post-Pandemic
def categorize_period(date):
    if date < pd.Timestamp("2020-03-11"):
        return "Pre-Pandemic"
    elif pd.Timestamp("2020-03-11") <= date <= pd.Timestamp("2023-05-05"):
        return "Pandemic"
    else:
        return "Post-Pandemic"

df_fred["Period"] = df_fred["Date"].apply(categorize_period)
# ✅ Filter data only from 2018 onward (double-check)
df_fred = df_fred[df_fred["Date"] >= "2018-01-01"]
# ✅ Save the dataset locally
fred_file_path = "fred_interest_unemployment_inflation_2018_2024.csv"
df_fred.to_csv(fred_file_path, index=False)
```

In [10]:

```
# Define the GitHub raw data URL
base_url = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_daily_reports_us/'

# Define the start date (April 12, 2020, as the first available dataset)
start_date = pd.Timestamp('2020-04-12')
end_date = pd.Timestamp.today()  # Get data up to today (latest available)

# Create an empty DataFrame to store state-level data
state_data = pd.DataFrame()

# Create an empty list to store daily aggregated data
national_data = []

# Iterate through each date and download available data
date = start_date
while date <= end_date:
    file_name = date.strftime('%m-%d-%Y') + '.csv'  # Format: MM-DD-YYYY
    file_url = base_url + file_name
    try:
        # Fetch CSV from GitHub
        response = requests.get(file_url)
        if response.status_code == 200:
            df = pd.read_csv(StringIO(response.text))

            # Save the state-level data to a separate CSV file
            state_filename = f'state_data_{date.strftime("%Y_%m_%d")}.csv'
            df.to_csv(state_filename, index=False)

            # Ensure columns exist before selecting
            selected_columns = ['Confirmed', 'Deaths', 'Case_Fatality_Ratio']
            df = df[[col for col in selected_columns if col in df.columns]]

            # Aggregate state-level data to national-level
            confirmed_total = df['Confirmed'].sum()
            deaths_total = df['Deaths'].sum()

            # Store aggregated results
            df_agg = pd.DataFrame({
                'Date': [date],
                'Confirmed': [confirmed_total],
                'Deaths': [deaths_total]
            })

            # Append to list
            national_data.append(df_agg)
    except Exception as e:
        print(f'❌ Data not available for {file_name}, skipping...')

    # Move to the next day
    date += pd.Timedelta(days=1)

# Combine all downloaded daily data into a national DataFrame
```

```
covid_data_national = pd.concat(national_data, ignore_index=True)

# Calculate the 7-day rolling average for new cases
covid_data_national['New_Cases'] = covid_data_national['Confirmed'].diff().fillna(0)
covid_data_national['New_Cases_7D_Rolling'] = covid_data_national['New_Cases'].rolling(window=7).mean()


# Calculate national fatality ratio
covid_data_national['Case_Fatality_Ratio'] = covid_data_national['Deaths'] / covid_data_national['Confirmed'] * 100

# Save the aggregated national data to CSV
csv_filename = 'covid_aggregated_cleaned.csv'
covid_data_national.to_csv(csv_filename, index=False)

# Print out a message indicating completion
print('State-level data saved to individual CSV files.')
print(f'Aggregated national data saved to {csv_filename}.')
```

```
State-level data saved to individual CSV files.
Aggregated national data saved to covid_aggregated_cleaned.csv.
```

In [64]:

```
# This merges dataset, THe time period of study includes pre-pandemic, pandemic and post pandemic. So need to extend
#The Data set but have null values for all the Covid-19 data columns as there is no data on this prior to the pandemic
#as well as after the pandemic.

# ✅ Load datasets -> just using csv makes it quicker
covid_data = pd.read_csv("covid_aggregated_cleaned.csv", parse_dates=["Date"])  # Primary dataset
volatility_data = pd.read_csv("russell_volatility_daily_gap.csv", parse_dates=["Date"])  # Volatility data
fred_data = pd.read_csv("fred_interest_unemployment_inflation_2018_2024.csv", parse_dates=["Date"])  # FRED data

# ✅ Define the full date range (from volatility data start to end of 2024)
full_date_range = pd.date_range(start=volatility_data["Date"].min(), end="2024-12-31", freq="D")
full_date_df = pd.DataFrame({"Date": full_date_range})

# ✅ Merge full date range with COVID-19 data (fill missing COVID values with 0)
covid_data = full_date_df.merge(covid_data, on="Date", how="left").fillna(0)

# ✅ Merge COVID-19 and volatility data (Ensure all volatility data remains intact)
merged_data = covid_data.merge(volatility_data, on="Date", how="right")  # Right join keeps all volatility data
# ✅ Ensure no missing dates in merged_data
merged_data["Date"] = merged_data["Date"].fillna(pd.NaT)  # Fill missing dates with NaT
merged_data = merged_data.dropna(subset=["Date"])  # Drop any rows where Date is NaT
merged_data = merged_data.sort_values("Date")  # Ensure sorting
# ✅ Forward-fill FRED data to match daily frequency

#Fred Data is only monhtly so need to forward fill so other days within a month have the same value in order to deal
#with null values
fred_data = fred_data.set_index("Date").asfreq("D").ffill().reset_index()
# ✅ Merge FRED data (Backward fill ensures each row has the latest known value)
merged_data = pd.merge_asof(merged_data, fred_data, on="Date", direction="backward")
# ✅ Keep only the relevant columns (Drop all unnecessary ones)
columns_to_keep = [
    "Date", "New_Cases_7D_Rolling","Confirmed","Case_Fatality_Ratio",  # COVID-19 Data
```

```
    "30D_Volatility_Russell2000","7D_Volatility_Russell2000", "Daily_Volatility_Gap",  # Market Volatility Data
    "Interest Rate (Fed Funds)", "Unemployment Rate", "Inflation (CPI)","Deaths",  # Macroeconomic Data
    "Period"  # Pandemic Phase
]
merged_data = merged_data[columns_to_keep]

# ✅ Ensure all missing numeric values are filled with zeros
numeric_columns = ["New_Cases_7D_Rolling", "30D_Volatility_Russell2000", "Daily_Volatility_Gap"]
merged_data[numeric_columns] = merged_data[numeric_columns].fillna(0)

# ✅ Define the pandemic indicator variable
def categorize_period(date):
    if date < pd.Timestamp("2020-03-11"):
        return "Pre-Pandemic"
    elif pd.Timestamp("2020-03-11") <= date <= pd.Timestamp("2023-05-05"):
        return "Pandemic"
    else:
        return "Post-Pandemic"
merged_data["Period"] = merged_data["Date"].apply(categorize_period)
# ✅ Save the final cleaned dataset
csv_filename = "cleaned_merged_data.csv"
merged_data.to_csv(csv_filename, index=False)
```

## New Datasets

In [ ]:

```
#This Dataset Breaks down small cap volatilty gap by sector in order to further explore volatilty and what casued it.

# Define the ticker symbols for the ETFs representing each sector for large-cap and small-cap
sector_tickers = {
    "Technology": {"Large": "XLK", "Small": "PSCT"},
    "Health Care": {"Large": "XLV", "Small": "IJR"},
    "Financials": {"Large": "XLF", "Small": "SLYG"},
    "Consumer Discretionary": {"Large": "XLY", "Small": "PSCD"},
    "Consumer Staples": {"Large": "XLP", "Small": "PSCC"},
    "Energy": {"Large": "XLE", "Small": "PSCE"},
    "Utilities": {"Large": "XLU", "Small": "PSCU"},
    "Industrials": {"Large": "XLI", "Small": "PSCI"},
    "Materials": {"Large": "XLB", "Small": "PSCM"},
    "Real Estate": {"Large": "XLRE", "Small": "PSR"},
    "Communication Services": {"Large": "XLC", "Small": "VOX"}
}

# Define the start and end dates for the historical data as strings
start_date = '2018-01-01'
end_date = '2024-01-31'

# DataFrame to hold daily volatility gap data
daily_volatility_gap_df = pd.DataFrame()

# Fetch historical data and calculate daily volatility gap for each sector ETF
for sector, tickers in sector_tickers.items():
```

```
    large_cap_data = yf.download(tickers["Large"], start=start_date, end=end_date)
    small_cap_data = yf.download(tickers["Small"], start=start_date, end=end_date)

    large_cap_volatility_daily = large_cap_data['Close'].pct_change().rolling(window=7).std()
    small_cap_volatility_daily = small_cap_data['Close'].pct_change().rolling(window=7).std()

    # Ensure the result is a Series
    if isinstance(large_cap_volatility_daily, pd.DataFrame):
        large_cap_volatility_daily = large_cap_volatility_daily.iloc[:, 0]
    if isinstance(small_cap_volatility_daily, pd.DataFrame):
        small_cap_volatility_daily = small_cap_volatility_daily.iloc[:, 0]

    daily_volatility_gap = small_cap_volatility_daily - large_cap_volatility_daily

    # The sector name is concatenated with the string ' Daily Volatility Gap' to create a unique column name
    daily_volatility_gap_df[sector + ' Daily Volatility Gap'] = daily_volatility_gap

daily_volatility_gap_df.dropna(how='all', inplace=True)
daily_volatility_gap_csv = 'daily_volatility_gap.csv'
daily_volatility_gap_df.to_csv(daily_volatility_gap_csv, index=True)
print(f"Daily volatility gap data saved to {daily_volatility_gap_csv}")
```

In [ ]:

```
#This Dataset Below looks into every small cap stock in the Russell 2000 index and finds the address so we can map it
# Load Russell 2000 company list
file_path = "russell_2000.xlsx"
df = pd.read_excel(file_path, sheet_name="Characteristics")

# Ensure column names are correctly referenced
ticker_col = "Ticker"
name_col = "Name"
companies = df[[ticker_col, name_col]].dropna()

def get_yahoo_address(ticker):
    """Fetch headquarters address from Yahoo Finance."""
    try:
        stock = yf.Ticker(ticker)
        info = stock.info
        address = info.get("address1", "") + ", " + info.get("city", "") + ", " + info.get("state", "") + ", " + info.get("zip", "")
        return address if address.strip() else None
    except:
        return None

def extract_state(address):
    """Extract state abbreviation from the address."""
    match = re.search(r',\s([A-Z]{2}),\s\d{5}', address)
    return match.group(1) if match else None

# Fetch addresses and extract states
company_data = []
for index, row in companies.iterrows():
    ticker, name = row[ticker_col], row[name_col]
    address = get_yahoo_address(ticker)
    state = extract_state(address) if address else None
```

```
    company_data.append([ticker, name, address, state])
    time.sleep(1)  # Avoid rate limits

# Convert to DataFrame
addresses_df = pd.DataFrame(company_data, columns=[ticker_col, name_col, "Headquarters Address", "State"])

# Merge with original dataframe
df = df.merge(addresses_df, on=[ticker_col, name_col], how="left")

# Save updated data
output_path = "russell_2000_with_states.xlsx"
df.to_excel(output_path, index=False)
```

In [3]:
```
#Code to get unempolyment by state

# ✅ Set up FRED API Key
fred = Fred(api_key="d9ec7a0936c1955f1169751b3adecdef")  # Use your valid FRED API Key

# ✅ Define state-level unemployment series (FRED codes for each state)
state_unemployment_series = {
    "Alabama": "ALUR", "Alaska": "AKUR", "Arizona": "AZUR", "Arkansas": "ARUR", "California": "CAUR",
    "Colorado": "COUR", "Connecticut": "CTUR", "Delaware": "DEUR", "Florida": "FLUR", "Georgia": "GAUR",
    "Hawaii": "HIUR", "Idaho": "IDUR", "Illinois": "ILUR", "Indiana": "INUR", "Iowa": "IAUR",
    "Kansas": "KSUR", "Kentucky": "KYUR", "Louisiana": "LAUR", "Maine": "MEUR", "Maryland": "MDUR",
    "Massachusetts": "MAUR", "Michigan": "MIUR", "Minnesota": "MNUR", "Mississippi": "MSUR", "Missouri": "MOUR",
    "Montana": "MTUR", "Nebraska": "NEUR", "Nevada": "NVUR", "New Hampshire": "NHUR", "New Jersey": "NJUR",
    "New Mexico": "NMUR", "New York": "NYUR", "North Carolina": "NCUR", "North Dakota": "NDUR", "Ohio": "OHUR",
    "Oklahoma": "OKUR", "Oregon": "ORUR", "Pennsylvania": "PAUR", "Rhode Island": "RIUR", "South Carolina": "SCUR",
    "South Dakota": "SDUR", "Tennessee": "TNUR", "Texas": "TXUR", "Utah": "UTUR", "Vermont": "VTUR",
    "Virginia": "VAUR", "Washington": "WAUR", "West Virginia": "WVUR", "Wisconsin": "WIUR", "Wyoming": "WYUR"
}

# ✅ Fetch most recent unemployment rate for each state (latest available in 2024)
state_unemployment = {}
for state, series_id in state_unemployment_series.items():
    try:
        # Fetch data starting in 2024 to get the latest figures
        data = fred.get_series(series_id, start_date="2024-01-01")
        latest_value = data.dropna().iloc[-1]  # Get the most recent non-NA value
        state_unemployment[state] = latest_value
    except Exception as e:
        print(f"❌ Could not fetch data for {state}: {e}")

# ✅ Convert to DataFrame
df_unemployment = pd.DataFrame(list(state_unemployment.items()), columns=["State", "Unemployment Rate"])

# ✅ Save the data to a CSV file for later use
df_unemployment.to_csv("state_unemployment_rates.csv", index=False)
```

The new datasets were needed in order to find an approximate for the volatility gap by state. The first data set finds the average volatility for every market sector using ETFs as proxies. The second dataset finds every company listed on the Russell 2000 which includes the top small-cap stocks. The third data set finds each state's unemployment rate.

## Variables and Their Relevance to Market Volatility

### COVID-19 Market Sentiment and Volatility

This study investigates the impact of COVID-19-related market sentiment and economic policies on the difference in volatility between small-cap and large-cap stocks. The dependent variable is the daily volatility gap between the small-, and large-cap stocks.

The first part of the study focuses on pandemic severity using Confirmed Cases, Case Fatality Ratio, and Deaths from the Jhon Hopkin Dataset. As the pandemic started, uncertainty rose, leading to higher volatility. Fear-driven market reactions to worsening outbreaks and fatality rates likely contributed to short-term spikes in volatility.

### Economic Indicators and Market Stability

The second part of the study looks into the role of economic indicators from the Federal Reserve Economic Data. The variables used are the Interest Rate (Fed Funds), Unemployment Rate, and Inflation (CPI). Interest rate changes influencing risk-taking behavior, and rising unemployment reflect economic distress, which both have the potential of amplifying volatility. Inflation levels impact monetary policy and investor expectations, further shaping market stability.

Together, these variables help explain how both the pandemic and policy responses influenced financial market fluctuations.

# 1. 3 Summary Statistics Tables

In [7]:
```
#Code below is for the summary Statistics
# ✅ Load the dataset
file_path = "cleaned_merged_data.csv"  # Ensure this is the correct file path
df_cleaned = pd.read_csv(file_path, parse_dates=["Date"])

# ✅ Identify columns to convert to percentage
percentage_columns = [
    "30D_Volatility_Russell2000", "7D_Volatility_Russell2000", "Daily_Volatility_Gap"
]

# ✅ Convert percentage columns from decimal format to percentage
df_cleaned[percentage_columns] *= 100

# ✅ Filter data for the COVID-19 pandemic period (March 11, 2020 - May 5, 2023)
df_covid_period = df_cleaned[(df_cleaned["Date"] >= "2020-03-11") & (df_cleaned["Date"] <= "2023-05-05")]

# ✅ Exclude non-numeric columns from summary statistics
exclude_columns = ["Date", "Period"]
numeric_cols = df_covid_period.select_dtypes(include=["number"]).columns
numeric_cols = [col for col in numeric_cols if col not in exclude_columns]

# ✅ Generate Summary Statistics for the COVID-19 period (Transposed)
summary_stats_covid = df_covid_period[numeric_cols].describe().T

# ✅ Format Columns (Apply Percentage Formatting)
styled_summary_covid = summary_stats_covid.style.format(
```

```
    {col: "{:,.2f}%" for col in percentage_columns},  # Format as percentages
    precision=2
).set_table_styles([
    {'selector': 'th', 'props': [('font-size', '14px'), ('text-align', 'center'), ('font-weight', 'bold'), ('background-color', '#E3F2FD'), ('colo
    {'selector': 'td', 'props': [('font-size', '12px'), ('text-align', 'center')]},
    {'selector': 'caption', 'props': [('caption-side', 'top'), ('font-size', '16px'), ('font-weight', 'bold'), ('text-align', 'center')]},
])

# ✅ Create a Styled Title for COVID-19 period
title_html_covid = """
    <h1 style="text-align: left; font-size: 26px; font-weight: bold; color: #2F4F4F; margin-bottom: 20px;">
    COVID-19 Period Summary Statistics (March 11, 2020 - May 5, 2023)
    </h1>
    <p style="text-align: left; font-size: 16px; color: #555;">
        Summary statistics for key COVID-19 indicators, economic conditions, and stock market volatility during the pandemic.
    </p>
"""
# ✅ Display the Title and Table in Jupyter Notebook
display(HTML(title_html_covid))
display(styled_summary_covid)
with open("summary_statistics_covid.html", "w") as f:
    f.write(title_html_covid + styled_summary_covid.to_html())
```

# COVID-19 Period Summary Statistics (March 11, 2020 - May 5, 2023)

Summary statistics for key COVID-19 indicators, economic conditions, and stock market volatility during the pandemic.

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **New_Cases_7D_Rolling** | 795.00 | 88934.65 | 113465.76 | 0.00 | 35979.86 | 58729.43 | 106949.43 | 805866.00 |
| **Confirmed** | 795.00 | 46600255.00 | 36579603.22 | 0.00 | 8653092.50 | 36198612.00 | 83580596.50 | 103802702.00 |
| **Case_Fatality_Ratio** | 795.00 | 1.81 | 1.31 | 0.00 | 1.11 | 1.60 | 1.79 | 6.12 |
| **30D_Volatility_Russell2000** | 795.00 | 1.77 | 0.92 | 0.95 | 1.30 | 1.53 | 1.83 | 6.20 |
| **7D_Volatility_Russell2000** | 795.00 | 1.69 | 1.07 | 0.44 | 1.14 | 1.45 | 1.91 | 9.72 |
| **Daily_Volatility_Gap** | 795.00 | 0.44 | 0.38 | -0.55 | 0.19 | 0.40 | 0.64 | 1.95 |
| **Interest Rate (Fed Funds)** | 795.00 | 1.09 | 1.63 | 0.05 | 0.08 | 0.09 | 1.68 | 5.06 |
| **Unemployment Rate** | 795.00 | 5.59 | 2.74 | 3.40 | 3.60 | 4.50 | 6.40 | 14.80 |
| **Inflation (CPI)** | 795.00 | 278.05 | 16.13 | 255.80 | 262.05 | 276.53 | 295.07 | 303.32 |
| **Deaths** | 795.00 | 618226.74 | 379773.75 | 0.00 | 224198.00 | 614633.00 | 1004463.00 | 1123836.00 |

## Interpretation of Summary Statistics

The average Case Fatality Ratio indicates that roughly 1.81% of reported cases resulted in death, with a peak of 6.12% during severe outbreaks. The Deaths column, reaching a maximum of 1.12 million, underscores the pandemic's human toll. The Daily Volatility Gap (0.44%) suggests small-cap stocks were more volatile compared to large-cap stocks. The Standard deivation(0.38) of the volatility GAP is almost equivalent to the mean, meaning that the volailty gap was still very volatile. The average interest rate was 1.09% but at its highest was 5.06%, reflecting the government monetary policy responses to the market downturn. The average Inflation (CPI: 278.08) and the wide range of almost 100 units between the min and max show rising prices. The summary statistics prove that volatility gaps jumped around during crises, aligning with concerns about highthened volatilty. Economic policies impacting interest rates and unemployment seem to be associated with volatility gap changes, reinforcing the study's goal of understanding COVID-19's impact on financial markets.

In [18]:
```
#Subset for 2020
# ✅ Load the dataset
file_path = "cleaned_merged_data.csv"  # Update if needed
df_cleaned = pd.read_csv(file_path, parse_dates=["Date"])

# ✅ Identify columns to convert to percentage
percentage_columns = [
    "30D_Volatility_Russell2000", "7D_Volatility_Russell2000", "Daily_Volatility_Gap"
]

# ✅ Convert relevant columns to percentage
df_cleaned[percentage_columns] *= 100

# ✅ Exclude non-numeric columns from summary statistics
exclude_columns = ["Date", "Period"]
numeric_cols = df_cleaned.select_dtypes(include=["number"]).columns
numeric_cols = [col for col in numeric_cols if col not in exclude_columns]

# ✅ Subset Data for 2020
df_2020 = df_cleaned[df_cleaned["Date"].dt.year == 2020]

# ✅ Generate Summary Statistics for 2020
summary_stats_2020 = df_2020[numeric_cols].describe().T

# ✅ Format Table for Display (Convert Percent Columns)
styled_summary_2020 = summary_stats_2020.style.format(
    {col: "{:,.2f}%" for col in percentage_columns},
    precision=2
).set_table_styles([
    {'selector': 'th', 'props': [('font-size', '14px'), ('text-align', 'center'), ('font-weight', 'bold'), ('background-color', '#E3F2FD'), ('colo
    {'selector': 'td', 'props': [('font-size', '12px'), ('text-align', 'center')]},
    {'selector': 'caption', 'props': [('caption-side', 'top'), ('font-size', '16px'), ('font-weight', 'bold'), ('text-align', 'center')]},
])

# ✅ Display Title & Summary Table for 2020
title_html_2020 = """
    <h1 style="text-align: left; font-size: 26px; font-weight: bold; color: #2F4F4F; margin-bottom: 20px;">
     COVID-19, Economic Indicators & Market Volatility Summary (2020)
    </h1>
    <p style="text-align: left; font-size: 16px; color: #555;">
        Summary statistics for COVID-19 cases, economic conditions, and stock market volatility during 2020.
    </p>
```

```
"""

display(HTML(title_html_2020))
display(styled_summary_2020)
```

# COVID-19, Economic Indicators & Market Volatility Summary (2020)

Summary statistics for COVID-19 cases, economic conditions, and stock market volatility during 2020.

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **New_Cases_7D_Rolling** | 253.00 | 51815.96 | 61022.01 | 0.00 | 0.00 | 35992.71 | 62083.29 | 224389.43 |
| **Confirmed** | 253.00 | 4734935.38 | 5180488.28 | 0.00 | 0.00 | 2756013.00 | 7314795.00 | 20219873.00 |
| **Case_Fatality_Ratio** | 253.00 | 2.66 | 2.05 | 0.00 | 0.00 | 2.76 | 4.01 | 6.12 |
| **30D_Volatility_Russell2000** | 253.00 | 2.19 | 1.52 | 0.42 | 1.21 | 1.60 | 2.70 | 6.20 |
| **7D_Volatility_Russell2000** | 253.00 | 2.08 | 1.73 | 0.26 | 1.05 | 1.54 | 2.35 | 9.72 |
| **Daily_Volatility_Gap** | 253.00 | 0.47 | 0.54 | -0.86 | 0.09 | 0.39 | 0.76 | 1.95 |
| **Interest Rate (Fed Funds)** | 253.00 | 0.37 | 0.54 | 0.05 | 0.09 | 0.09 | 0.10 | 1.58 |
| **Unemployment Rate** | 253.00 | 8.12 | 3.45 | 3.50 | 6.70 | 7.80 | 10.20 | 14.80 |
| **Inflation (CPI)** | 253.00 | 258.86 | 1.82 | 255.80 | 258.08 | 259.13 | 260.32 | 262.05 |
| **Deaths** | 253.00 | 127392.70 | 101928.07 | 0.00 | 0.00 | 128477.00 | 206785.00 | 350604.00 |

2020 was the pandemic's initial shock, with an average new case 7 day average of around 52000 cases and a lower case fatality ratio (0.36% vs. 1.14%) due to early underreporting. Market volatility was higher, with 30-day volatility (1.51%) and 7-day volatility (1.44%) peaking early. The daily volatility gap (0.47%) was also elevated. 2020 set the stage for ongoing volatility and policy shifts. Please note there are only 252 Trading days in a year.

In [19]:
```
# ✅ Subset Data for 2022
df_2022 = df_cleaned[df_cleaned["Date"].dt.year == 2022]

# ✅ Generate Summary Statistics for 2022
summary_stats_2022 = df_2022[numeric_cols].describe().T

# ✅ Format Table for Display (Convert Percent Columns)
styled_summary_2022 = summary_stats_2022.style.format(
    {col: "{:,.2f}%" for col in percentage_columns},
    precision=2
).set_table_styles([
    {'selector': 'th', 'props': [('font-size', '14px'), ('text-align', 'center'), ('font-weight', 'bold'), ('background-color', '#E3F2FD'), ('colo
    {'selector': 'td', 'props': [('font-size', '12px'), ('text-align', 'center')]},
    {'selector': 'caption', 'props': [('caption-side', 'top'), ('font-size', '16px'), ('font-weight', 'bold'), ('text-align', 'center')]},
])
```

```
# ✅ Display Title & Summary Table for 2022
title_html_2022 = """
    <h1 style="text-align: left; font-size: 24px; font-weight: bold; color: #2F4F4F; margin-bottom: 20px;">
        COVID-19, Economic Indicators & Market Volatility Summary (2022)
    </h1>
    <p style="text-align: left; font-size: 16px; color: #555;">
        Summary statistics for COVID-19 cases, economic conditions, and stock market volatility during 2022.
    </p>
"""

display(HTML(title_html_2022))
display(styled_summary_2022)
```

# COVID-19, Economic Indicators & Market Volatility Summary (2022)

Summary statistics for COVID-19 cases, economic conditions, and stock market volatility during 2022.

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **New_Cases_7D_Rolling** | 251.00 | 127879.76 | 173008.24 | 27443.43 | 41484.93 | 69691.00 | 110307.86 | 805866.00 |
| **Confirmed** | 251.00 | 87388947.78 | 9810246.73 | 56439072.00 | 80280509.50 | 88079382.00 | 96389318.50 | 100757410.00 |
| **Case_Fatality_Ratio** | 251.00 | 1.16 | 0.07 | 1.08 | 1.10 | 1.16 | 1.21 | 1.47 |
| **30D_Volatility_Russell2000** | 251.00 | 1.77 | 0.24 | 1.30 | 1.57 | 1.78 | 1.95 | 2.34 |
| **7D_Volatility_Russell2000** | 251.00 | 1.72 | 0.48 | 0.61 | 1.40 | 1.67 | 2.02 | 3.10 |
| **Daily_Volatility_Gap** | 251.00 | 0.26 | 0.24 | -0.31 | 0.10 | 0.28 | 0.41 | 0.90 |
| **Interest Rate (Fed Funds)** | 251.00 | 1.70 | 1.40 | 0.08 | 0.33 | 1.68 | 2.82 | 4.10 |
| **Unemployment Rate** | 251.00 | 3.64 | 0.14 | 3.50 | 3.60 | 3.60 | 3.70 | 4.00 |
| **Inflation (CPI)** | 251.00 | 292.72 | 5.34 | 282.54 | 288.58 | 295.07 | 297.20 | 298.81 |
| **Deaths** | 251.00 | 1009319.17 | 64006.94 | 828235.00 | 984375.00 | 1018479.00 | 1059679.00 | 1092738.00 |

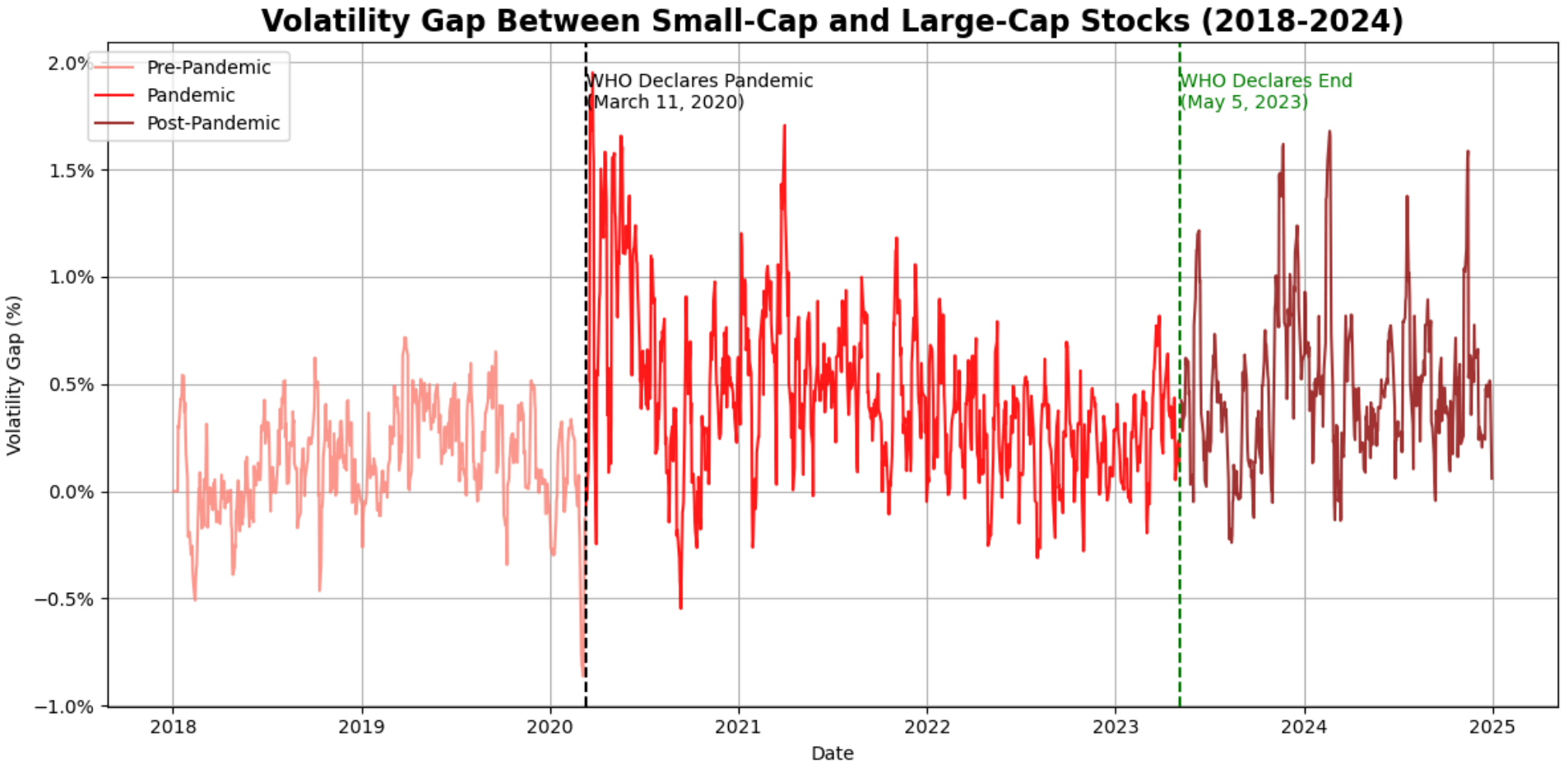
The summary statistics for 2022 display that the number of daily new cases increased tremendously. Signaling that the sprad of Covid-19 was at its highest. The case fatality ratio (1.21%) remained slightly higher than in 2020. The difference in volatility between small-cap and large-cap stocks decreased (0.26%) dropping compared to 2020's spikes. Some key independent variables that are worth highlighting: Unemployment (3.64%) became more stable reaching near pre-pandemic levels. Inflation was at its highest, reflecting lingering economic disruptions.

# 1.4 Plots, Histograms, Figures

## Volatility Gap Over Time

In [9]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker

# Function to plot Volatility Gap Trend
def plot_volatility_gap_trend():
    # Load the cleaned merged dataset
    file_path = 'cleaned_merged_data.csv'  # Update if needed
    df_cleaned = pd.read_csv(file_path, parse_dates=['Date'])

    # Convert volatility gap to percentage format
    df_cleaned['Daily_Volatility_Gap'] *= 100

    # Define pandemic period ranges
    pandemic_start = pd.Timestamp('2020-03-11')
    pandemic_end = pd.Timestamp('2023-05-05')

    # Separate data into different time periods
    pre_pandemic = df_cleaned[df_cleaned['Date'] < pandemic_start]
    pandemic = df_cleaned[(df_cleaned['Date'] >= pandemic_start) & (df_cleaned['Date'] <= pandemic_end)]
    post_pandemic = df_cleaned[df_cleaned['Date'] > pandemic_end]

    # Plot: Volatility Gap Trend (2018-2024) with meaningful colors
    plt.figure(figsize=(12, 6))

    # Use a color gradient based on volatility significance
    plt.plot(pre_pandemic['Date'], pre_pandemic['Daily_Volatility_Gap'], label='Pre-Pandemic', color='#FA8072', alpha=0.8)  # Soft Pink
    plt.plot(pandemic['Date'], pandemic['Daily_Volatility_Gap'], label='Pandemic', color='#FF0000', alpha=0.9)  # Intense Red
    plt.plot(post_pandemic['Date'], post_pandemic['Daily_Volatility_Gap'], label='Post-Pandemic', color='#8B0000', alpha=0.8)  # Dark Rose

    # Mark key pandemic periods with vertical lines (but remove from legend)
    plt.axvline(pandemic_start, color='black', linestyle='--')
    plt.axvline(pandemic_end, color='green', linestyle='--')

    # Add annotations instead of legend entries
    plt.text(pandemic_start, df_cleaned['Daily_Volatility_Gap'].max(), 'WHO Declares Pandemic\n(March 11, 2020)', color='black', fontsize=10, verti
    plt.text(pandemic_end, df_cleaned['Daily_Volatility_Gap'].max(), 'WHO Declares End\n(May 5, 2023)', color='green', fontsize=10, verticalalignme

    # Formatting the plot
    plt.title('Volatility Gap Between Small-Cap and Large-Cap Stocks (2018-2024)', fontsize=16, fontweight='bold')
    plt.xlabel('Date')
    plt.ylabel('Volatility Gap (%)')

    # Format y-axis to display percentages
    plt.gca().yaxis.set_major_formatter(mticker.PercentFormatter(decimals=1))
```

```
    # Move legend to the left outside the plot
    plt.legend(loc='upper left', bbox_to_anchor=(-0.02, 1))  # Moves legend to the left of the plot
    plt.grid(True)

    # Show the plot
    plt.tight_layout()
    plt.savefig("volatility_gap_trend.png", dpi=300)  # You can change filename or format (e.g., .jpg)

    # Show the plot
    plt.show()

# Call the function whenever you want to display the plot
plot_volatility_gap_trend()
```




## Interpretation of Volatility Gap Trend (2018-2024)

This visualization displays the difference in volatility between small-cap and large-cap stocks over time, segmented into pre-, during, and post-pandemic periods. It's apparent that the difference in volatility between small-cap and large-cap stocks has changed drastically since the start of the pandemic. It has remained elevated

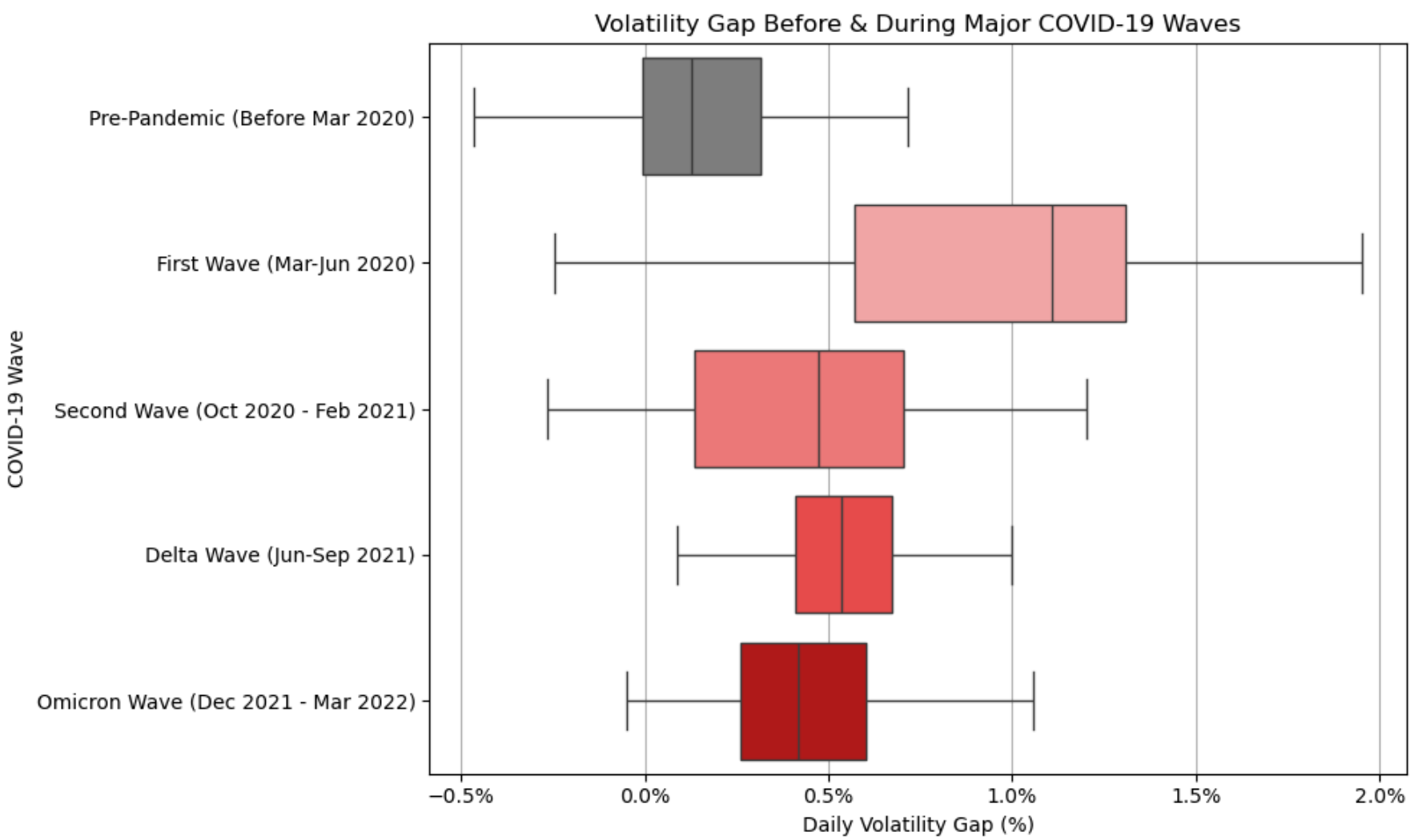
even after the pandemic was declared over by the World Health Organization. The volatility gap on average has become much higher and has remained above pre-pandemic levels, suggesting there is persistent uncertainty that specifically impacts small-cap stocks.

## COVID-19 Impact on Market Volatility

### Volatility Gap Before & During Major COVID-19 Waves

In [11]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker

# Load dataset
df = pd.read_csv('cleaned_merged_data.csv', parse_dates=['Date'])

# Convert volatility gap to percentage format
df['Daily_Volatility_Gap'] *= 100

# Define major COVID waves + pre-pandemic
def categorize_wave(date):
    if date < pd.Timestamp('2020-03-11'):
        return 'Pre-Pandemic (Before Mar 2020)'
    elif pd.Timestamp('2020-03-11') <= date <= pd.Timestamp('2020-06-30'):
        return 'First Wave (Mar-Jun 2020)'
    elif pd.Timestamp('2020-10-01') <= date <= pd.Timestamp('2021-02-28'):
        return 'Second Wave (Oct 2020 - Feb 2021)'
    elif pd.Timestamp('2021-06-01') <= date <= pd.Timestamp('2021-09-30'):
        return 'Delta Wave (Jun-Sep 2021)'
    elif pd.Timestamp('2021-12-01') <= date <= pd.Timestamp('2022-03-31'):
        return 'Omicron Wave (Dec 2021 - Mar 2022)'
    else:
        return 'Other'

df['COVID_Wave'] = df['Date'].apply(categorize_wave)

# Filter out the 'Other' category
df = df[df['COVID_Wave'] != 'Other']

# Define colors manually: Grey for Pre-Pandemic, Reds for others
custom_colors = {
    'Pre-Pandemic (Before Mar 2020)': 'grey',
    'First Wave (Mar-Jun 2020)': '#FF9999',
    'Second Wave (Oct 2020 - Feb 2021)': '#FF6666',
    'Delta Wave (Jun-Sep 2021)': '#FF3333',
    'Omicron Wave (Dec 2021 - Mar 2022)': '#CC0000'
}

# Plot: Box plot for Daily Volatility Gap without outliers
plt.figure(figsize=(10, 6))
sns.boxplot(x='Daily_Volatility_Gap', y='COVID_Wave', data=df,
            order=[
```

```
                'Pre-Pandemic (Before Mar 2020)',
                'First Wave (Mar-Jun 2020)',
                'Second Wave (Oct 2020 - Feb 2021)',
                'Delta Wave (Jun-Sep 2021)',
                'Omicron Wave (Dec 2021 - Mar 2022)'
            ],
            palette=custom_colors,
            showfliers=False,
            hue='COVID_Wave', # Assigning hue to avoid FutureWarning
            dodge=False) # Ensures that the boxplot does not separate the hues

# Formatting
plt.xlabel('Daily Volatility Gap (%)')
plt.ylabel('COVID-19 Wave')
plt.title('Volatility Gap Before & During Major COVID-19 Waves')
plt.grid(axis='x')

# Format x-axis as percentage (instead of decimal)
plt.gca().xaxis.set_major_formatter(mticker.PercentFormatter(decimals=1))

# Hide the legend as it is not needed
plt.legend([],[], frameon=False)
plt.tight_layout()
plt.savefig("Volatility Gap Before & During Major COVID-19 Waves", dpi=300)

# Show Plot
plt.show()
```

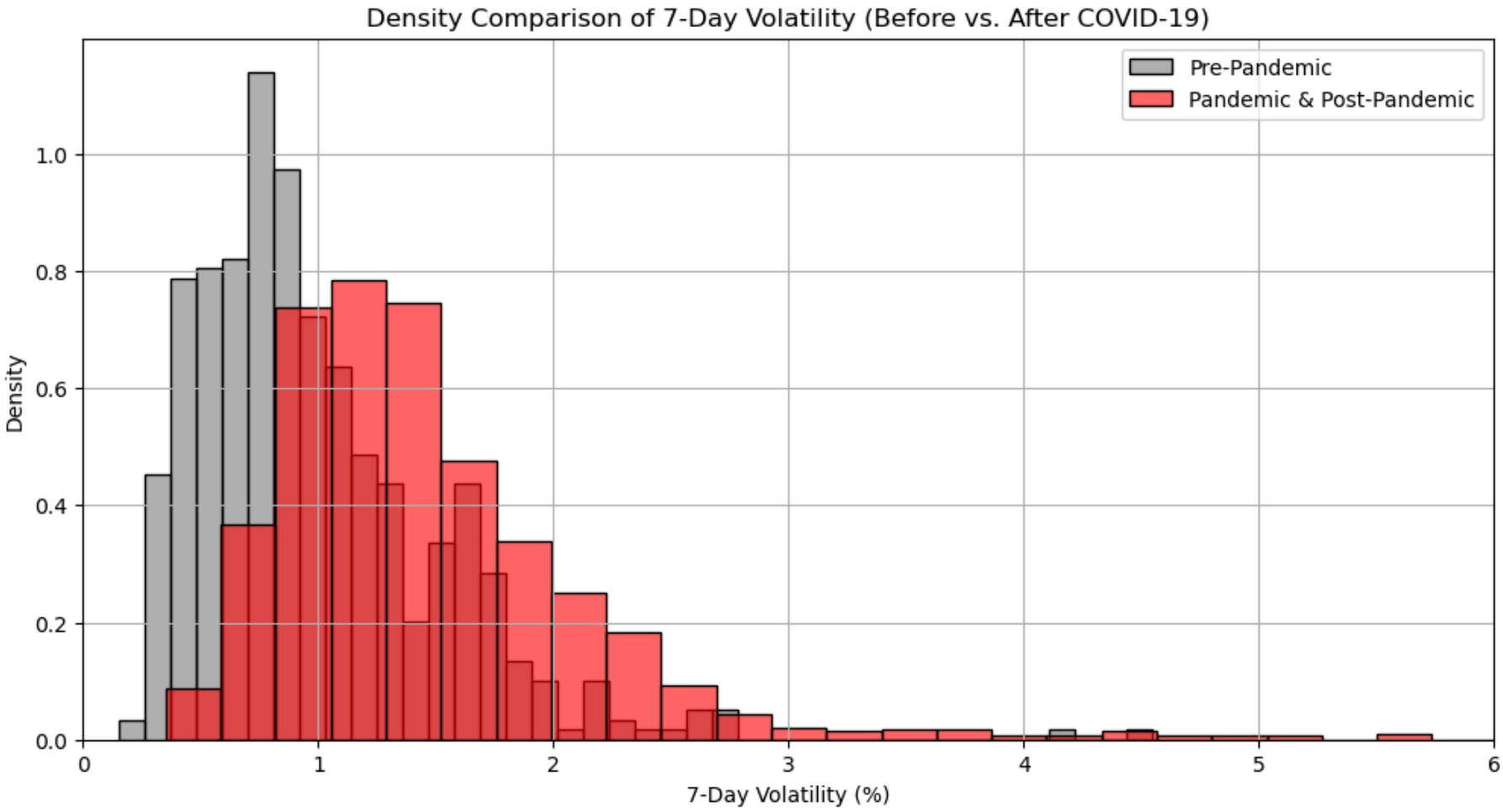



The box chart displays the distribution of the volatility gap between large cap and small cap during Key waves in COVID-19. The average volatility gap was the highest during the first wave, but not as high in the subsequent waves but still above pre-pandemic levels. This aligns with the research question, showing how COVID-19-driven uncertainty persisted beyond the initial crisis, affecting investor sentiment and risk pricing.

### Density Comparison of 7-Day Volatility (Before vs. After COVID-19)

In [22]:
```
#Third Visuzliation
# ✅ Load dataset
df = pd.read_csv("cleaned_merged_data.csv", parse_dates=["Date"])

# ✅ Convert volatility to percentage format
df["7D_Volatility_Russell2000"] *= 100

# ✅ Define two categories
pre_covid = df[df["Period"] == "Pre-Pandemic"]
```

```python
after_covid = df[df["Period"] != "Pre-Pandemic"]  # Combines Pandemic & Post-Pandemic

# ✅ Plot Density Histogram for 7-Day Volatility (Two Series)
plt.figure(figsize=(12, 6))

sns.histplot(pre_covid["7D_Volatility_Russell2000"], bins=40, color="gray", label="Pre-Pandemic", alpha=0.6, stat="density")
sns.histplot(after_covid["7D_Volatility_Russell2000"], bins=40, color="red", label="Pandemic & Post-Pandemic", alpha=0.6, stat="density")

# ✅ Set X-Axis Limit (0% - 6%)
plt.xlim(0, 6)

# ✅ Formatting
plt.title("Density Comparison of 7-Day Volatility (Before vs. After COVID-19)")
plt.xlabel("7-Day Volatility (%)")
plt.ylabel("Density")  # Instead of frequency
plt.legend()
plt.grid(True)

# ✅ Show Plot
plt.show()
```

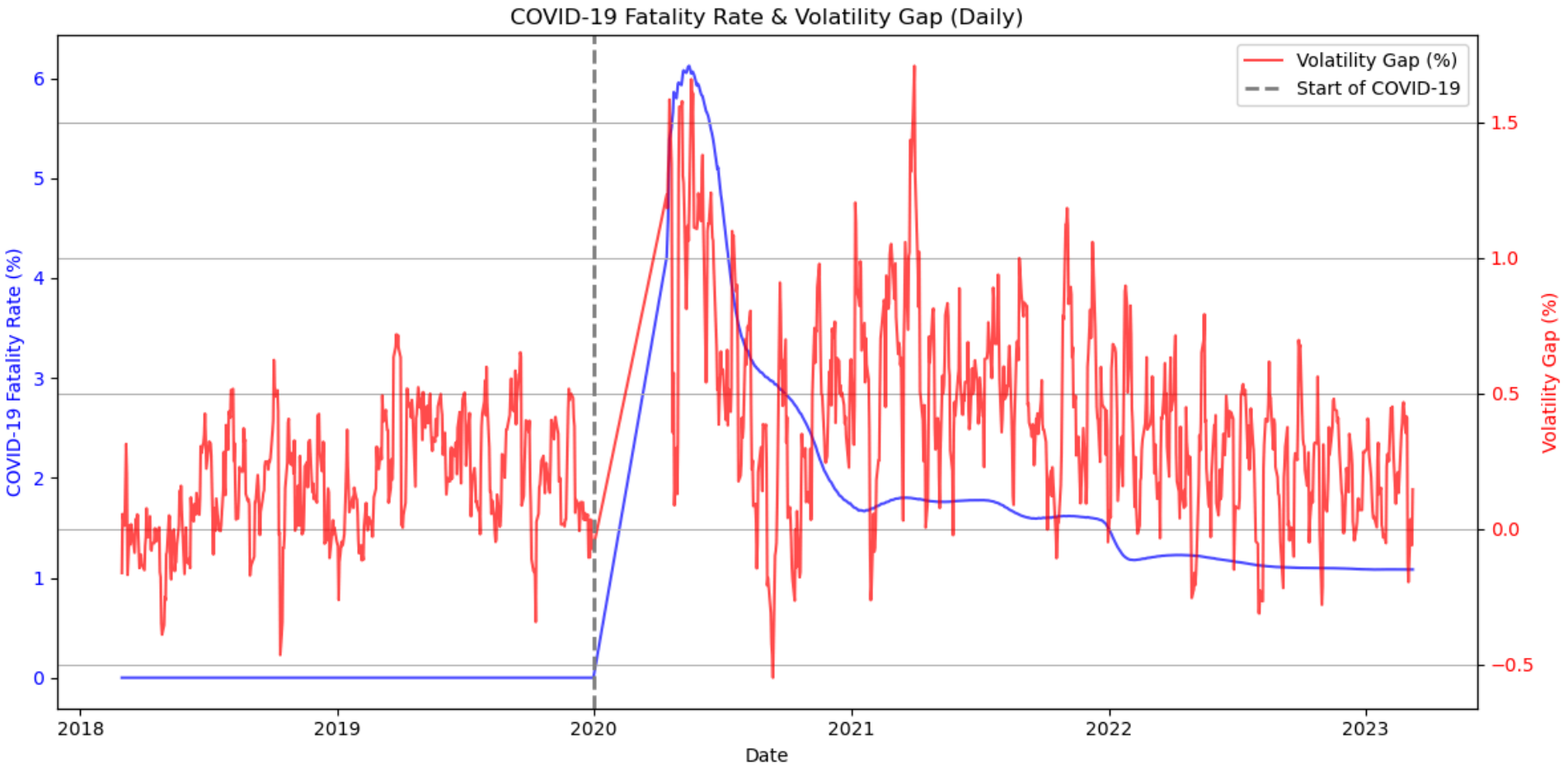


This Density histogram displays the 7-day volatility gap of small-cap stocks pre and post covid-19. We see that the volatility of the small-cap has increased as compared to pre-pandemic, with a wider distribution. This supports the idea that COVID-19 had a lasting impact on market stability, particularly for small-cap stocks.

## COVID-19 Fatality Rate & Volatility Gap (Weekly Averages)

In [17]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv('cleaned_merged_data.csv', parse_dates=['Date'])

# Convert volatility gap to percentage format
df['Daily_Volatility_Gap'] *= 100

# Calculate COVID-19 Fatality Rate (%)
df['Fatality_Rate'] = (df['Deaths'] / df['Confirmed']) * 100

# Ensure columns are numeric (convert & handle errors)
df['Fatality_Rate'] = pd.to_numeric(df['Fatality_Rate'], errors='coerce')
df['Daily_Volatility_Gap'] = pd.to_numeric(df['Daily_Volatility_Gap'], errors='coerce')

# Convert 'Date' to datetime format
df['Date'] = pd.to_datetime(df['Date'], errors='coerce')

# Set 'Date' as Index
df.set_index('Date', inplace=True)

# Define the start date for pre-COVID data
pre_covid_start = pd.to_datetime('2018-03-01')

# Define the transition date to COVID-19 period
covid_start = pd.to_datetime('2020-01-01')

# Include data starting from pre-COVID start date
df = df[df.index >= pre_covid_start]

# Set Fatality Rate to zero for pre-COVID dates
df.loc[df.index < covid_start, 'Fatality_Rate'] = 0

# Drop NA values if any
df.dropna(inplace=True)

# Line Plot: Fatality Rate vs. Volatility Gap Over Time
fig, ax1 = plt.subplots(figsize=(12, 6))

# Plot Fatality Rate
ax1.plot(df.index, df['Fatality_Rate'], label='Fatality Rate (%)', color='blue', alpha=0.7)
ax1.set_xlabel('Date')
ax1.set_ylabel('COVID-19 Fatality Rate (%)', color='blue')
ax1.tick_params(axis='y', labelcolor='blue')
```

```
# Create a secondary y-axis for Volatility Gap
ax2 = ax1.twinx()
ax2.plot(df.index, df['Daily_Volatility_Gap'], label='Volatility Gap (%)', color='red', alpha=0.7)
ax2.set_ylabel('Volatility Gap (%)', color='red')
ax2.tick_params(axis='y', labelcolor='red')

# Title and Formatting
plt.title('COVID-19 Fatality Rate & Volatility Gap (Daily)')

# Add a dotted line to indicate the start of COVID-19 period
plt.axvline(covid_start, color='grey', linestyle='--', linewidth=2, label='Start of COVID-19')

fig.tight_layout()
plt.grid(True)

# Show Plot
plt.legend(loc='best')
plt.savefig("COVID-19 Fatality Rate & Volatility Gap (Daily)", dpi=300)
plt.show()
```



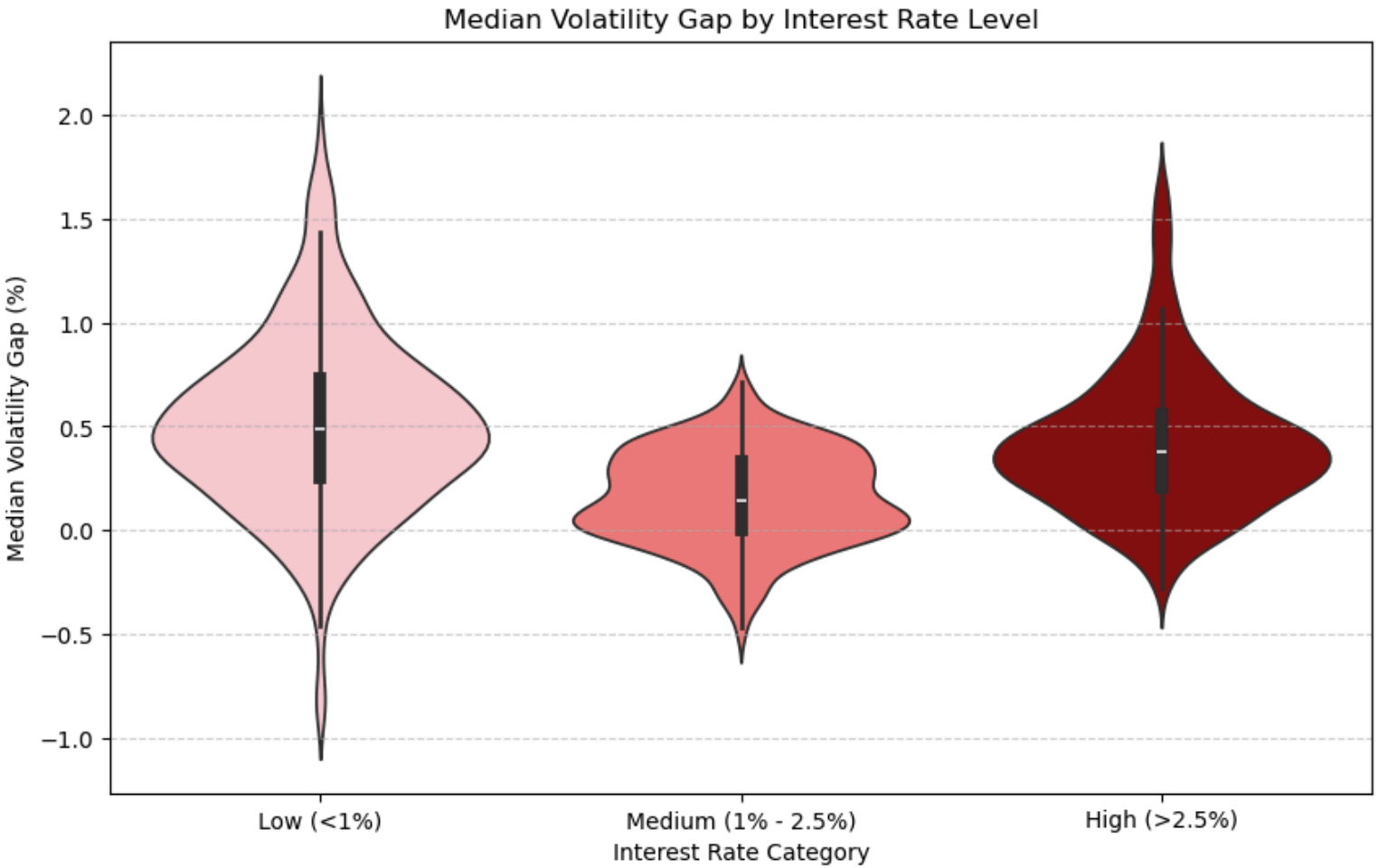
The line graph above displays the relationship between COVID-19 fatality rates and the volatility gap. When the pandemic started both the volatility gap and the fatality rate moved upward and downward together, however after 2021 there does not seem to be a strong relationship between the two.

## Economic Indicators & Market Volatility

### Median Volatility Gap by Interest Rate Level

In [18]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv('cleaned_merged_data.csv', parse_dates=['Date'])

# Convert volatility gap to percentage format
df['Daily_Volatility_Gap'] *= 100

# Define Interest Rate Categories
def categorize_interest_rate(rate):
    if rate < 1.0:
        return 'Low (<1%)'
    elif 1.0 <= rate <= 2.5:
        return 'Medium (1% - 2.5%)'
    else:
        return 'High (>2.5%)'

df['Interest_Rate_Category'] = df['Interest Rate (Fed Funds)'].apply(categorize_interest_rate)

# Sort categories properly
category_order = ['Low (<1%)', 'Medium (1% - 2.5%)', 'High (>2.5%)']
df['Interest_Rate_Category'] = pd.Categorical(df['Interest_Rate_Category'], categories=category_order, ordered=True)

# Define custom colors
custom_palette = {
    'Low (<1%)': '#FFC0CB',  # Light Pink
    'Medium (1% - 2.5%)': '#FF6666',  # Red
    'High (>2.5%)': '#990000'  # Dark Red
}

# Plot Violin Plot with Hue set to 'Interest_Rate_Category' and legend removed
plt.figure(figsize=(10, 6))
sns.violinplot(x='Interest_Rate_Category', y='Daily_Volatility_Gap', data=df, hue='Interest_Rate_Category', palette=custom_palette)
plt.legend([],[], frameon=False)

# Formatting
plt.title('Median Volatility Gap by Interest Rate Level')
plt.xlabel('Interest Rate Category')
plt.ylabel('Median Volatility Gap (%)')
plt.grid(axis='y', linestyle='--', alpha=0.6)  # Lighter grid lines
```

```
# Show Plot
plt.show()
```

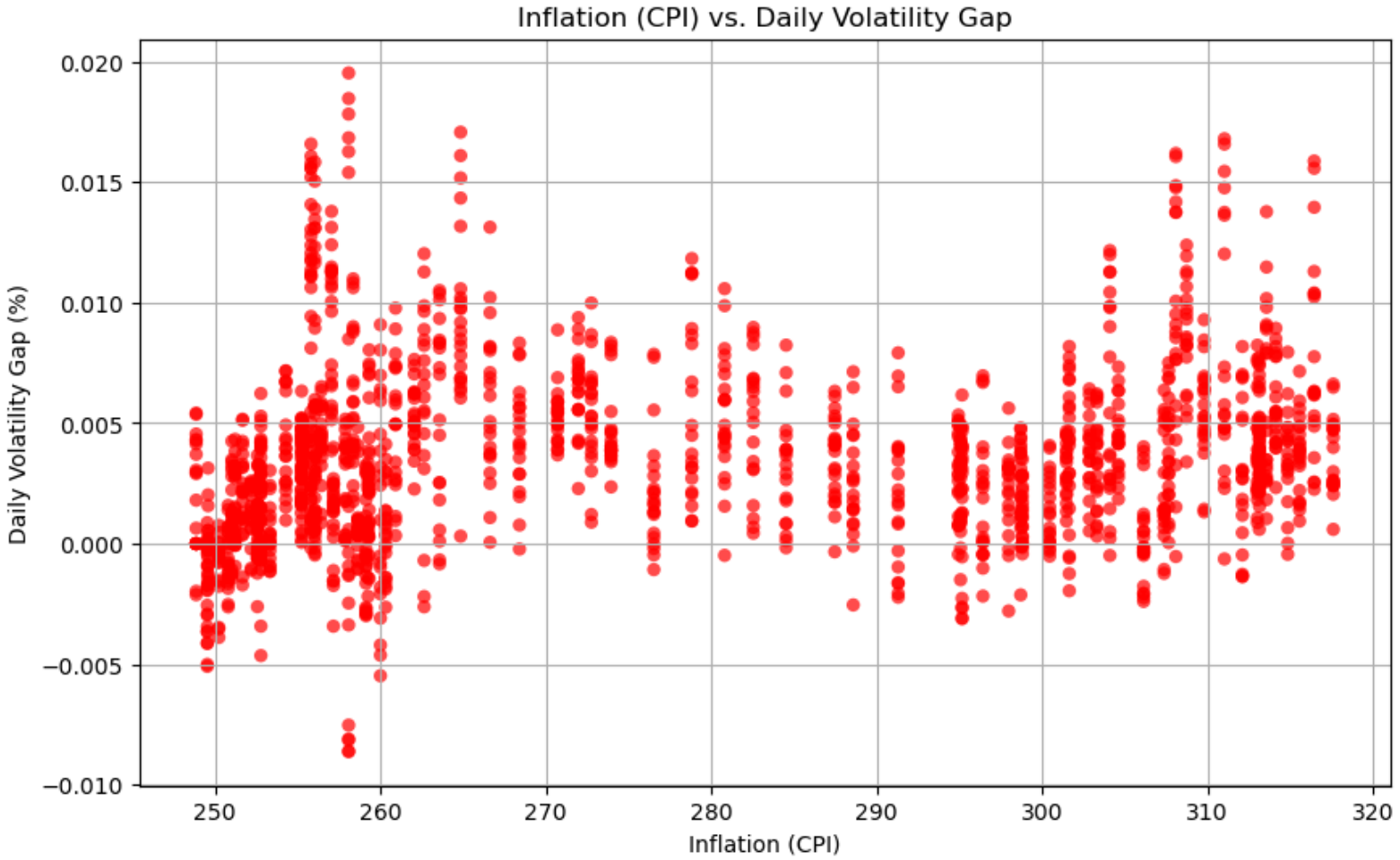



The violin chart above displays the distribution and density relationship between the overnight reserve rate and market volatility. There are 3 bins to categorize low, medium, and high interest rates. When interest is low, we see the largest median volatility gap and also the largest range of distributions. This is very interesting because when interest rates are low, the volatility gap is at its largest. Lower interest rates mean a lower cost of borrowing, so there should be market stability however this was not the case.

### Inflation (CPI) vs. Volatility Gap

In [19]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
df = pd.read_csv('cleaned_merged_data.csv', parse_dates=['Date'])

# Ensure Inflation and Volatility Gap are Numeric
```

```
df['Inflation (CPI)'] = pd.to_numeric(df['Inflation (CPI)'], errors='coerce')
df['Daily_Volatility_Gap'] = pd.to_numeric(df['Daily_Volatility_Gap'], errors='coerce')

# Keep Only Necessary Columns Before Processing
df_numeric = df[['Date', 'Inflation (CPI)', 'Daily_Volatility_Gap']].dropna()

# Scatter Plot: Inflation vs. Daily Volatility Gap
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df_numeric['Inflation (CPI)'], y=df_numeric['Daily_Volatility_Gap'], alpha=0.7, color='red', edgecolor='none')

# Formatting
plt.title('Inflation (CPI) vs. Daily Volatility Gap')
plt.xlabel('Inflation (CPI)')
plt.ylabel('Daily Volatility Gap (%)')
plt.grid(True)

# Show Plot
plt.show()
```

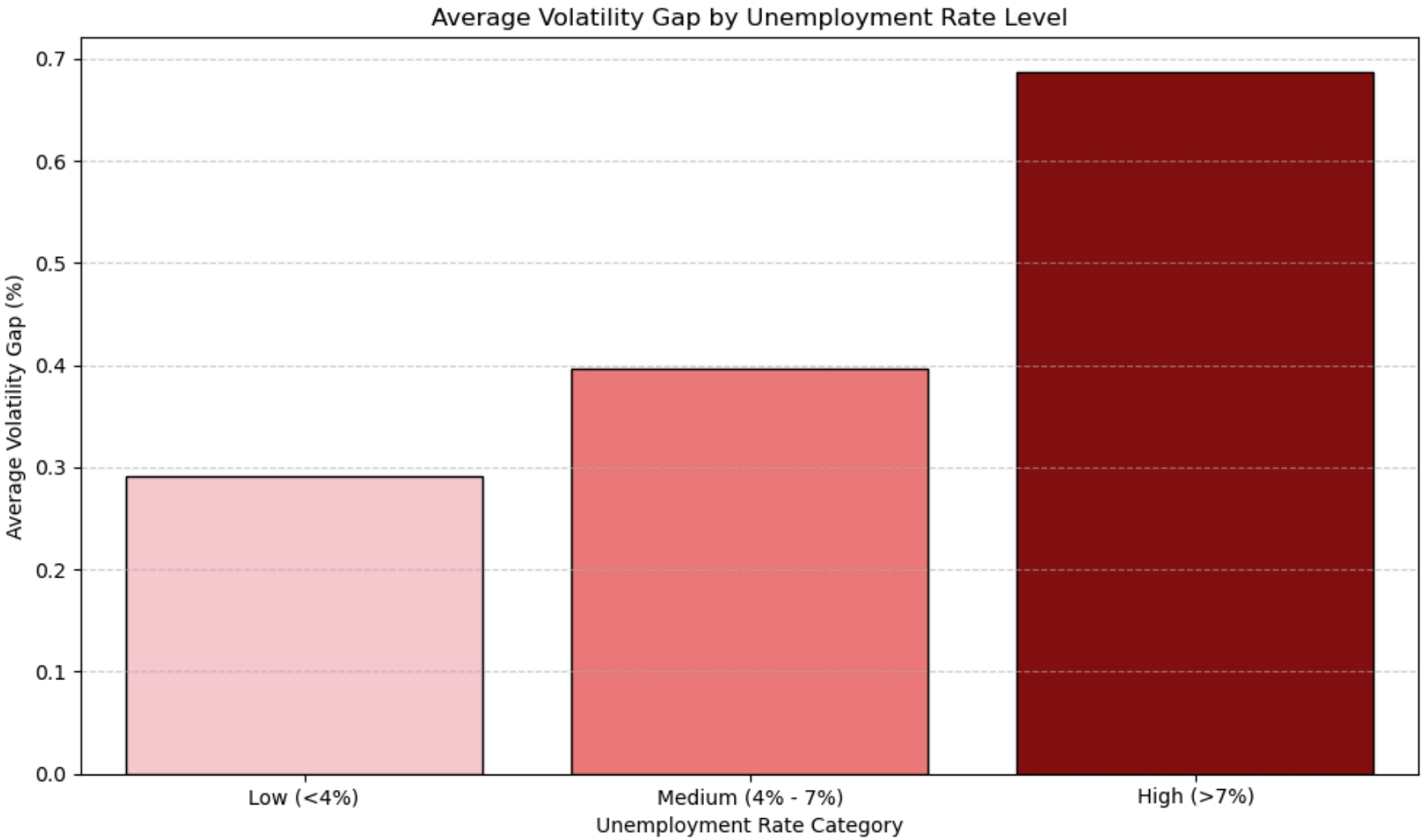


The scatter plot presents the relationship between the volatility gap and inflation. The chart shows that there is no apparent relationship between inflation and CPI.

### Average Volatility Gap by Unemployment Rate Level

In [14]:
```
#7th Visulization
# ✅ Load dataset
df = pd.read_csv("cleaned_merged_data.csv", parse_dates=["Date"])

# ✅ Convert volatility gap to percentage format
df["Daily_Volatility_Gap"] *= 100

# ✅ Define Unemployment Rate Categories
def categorize_unemployment(rate):
    if rate < 4.0:
        return "Low (<4%)"
    elif 4.0 <= rate <= 7.0:
        return "Medium (4% - 7%)"
    else:
        return "High (>7%)"

df["Unemployment_Category"] = df["Unemployment Rate"].apply(categorize_unemployment)

# ✅ Calculate Average Volatility Gap for Each Unemployment Category
volatility_by_unemployment = df.groupby("Unemployment_Category")["Daily_Volatility_Gap"].mean().reset_index()

# ✅ Sort categories properly
category_order = ["Low (<4%)", "Medium (4% - 7%)", "High (>7%)"]
volatility_by_unemployment["Unemployment_Category"] = pd.Categorical(volatility_by_unemployment["Unemployment_Category"],
                                                                     categories=category_order, ordered=True)
volatility_by_unemployment = volatility_by_unemployment.sort_values("Unemployment_Category")

# ✅ Define custom colors
custom_palette = {
    "Low (<4%)": "#FFC0CB",  # Light Pink
    "Medium (4% - 7%)": "#FF6666",  # Red
    "High (>7%)": "#990000"  # Dark Red
}

# ✅ Plot Bar Chart with Explicit Hue and No Legend
plt.figure(figsize=(10, 6))
sns.barplot(x="Unemployment_Category", y="Daily_Volatility_Gap",
            data=volatility_by_unemployment, hue="Unemployment_Category",
            palette=custom_palette, legend=False, edgecolor="black")  # Explicit hue prevents warning

# ✅ Formatting
plt.title("Average Volatility Gap by Unemployment Rate Level")
plt.xlabel("Unemployment Rate Category")
plt.ylabel("Average Volatility Gap (%)")
plt.grid(axis="y", linestyle="--", alpha=0.6)  # Lighter grid lines
plt.tight_layout()
plt.savefig("Average Volatility Gap by Unemployment Rate Level", dpi=300)
```

```
# ✅ Show Plot
plt.show()
```

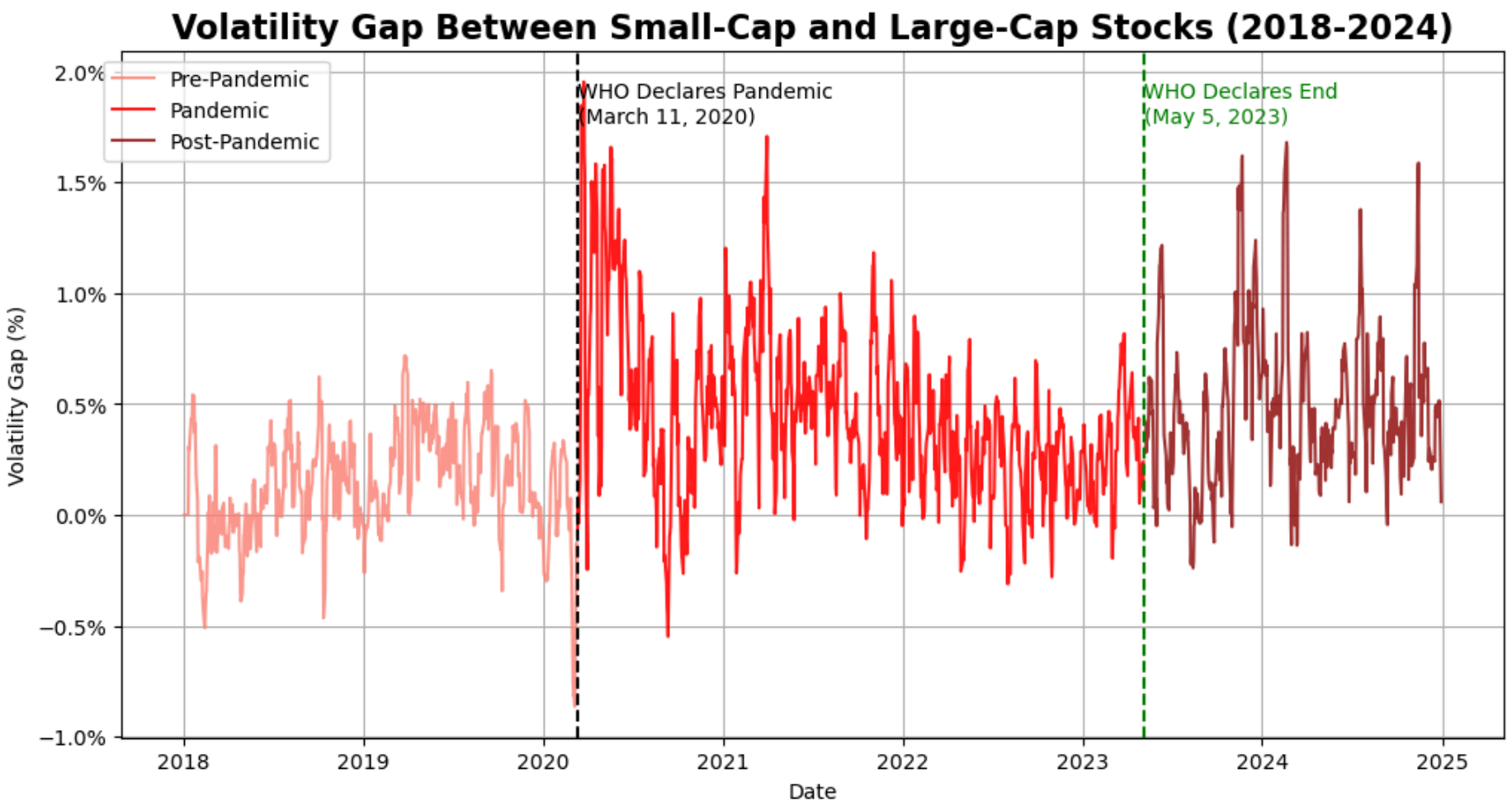



The chart shows that as the unemployment rate increases, the average volatility gap between small-cap and large-cap stocks also rises. During periods of low unemployment, the volatility gap remains smaller in size, suggesting that economic stability and investor confidence can help reduce the volatility gap. However, as unemployment moves into the medium and high range, the volatility difference between small-cap and large-cap expands significantly. This indicates that small-cap stocks face heightened uncertainty during economic downturns.

# Project 2

## The Message

In [27]:
```
plot_volatility_gap_trend()
```



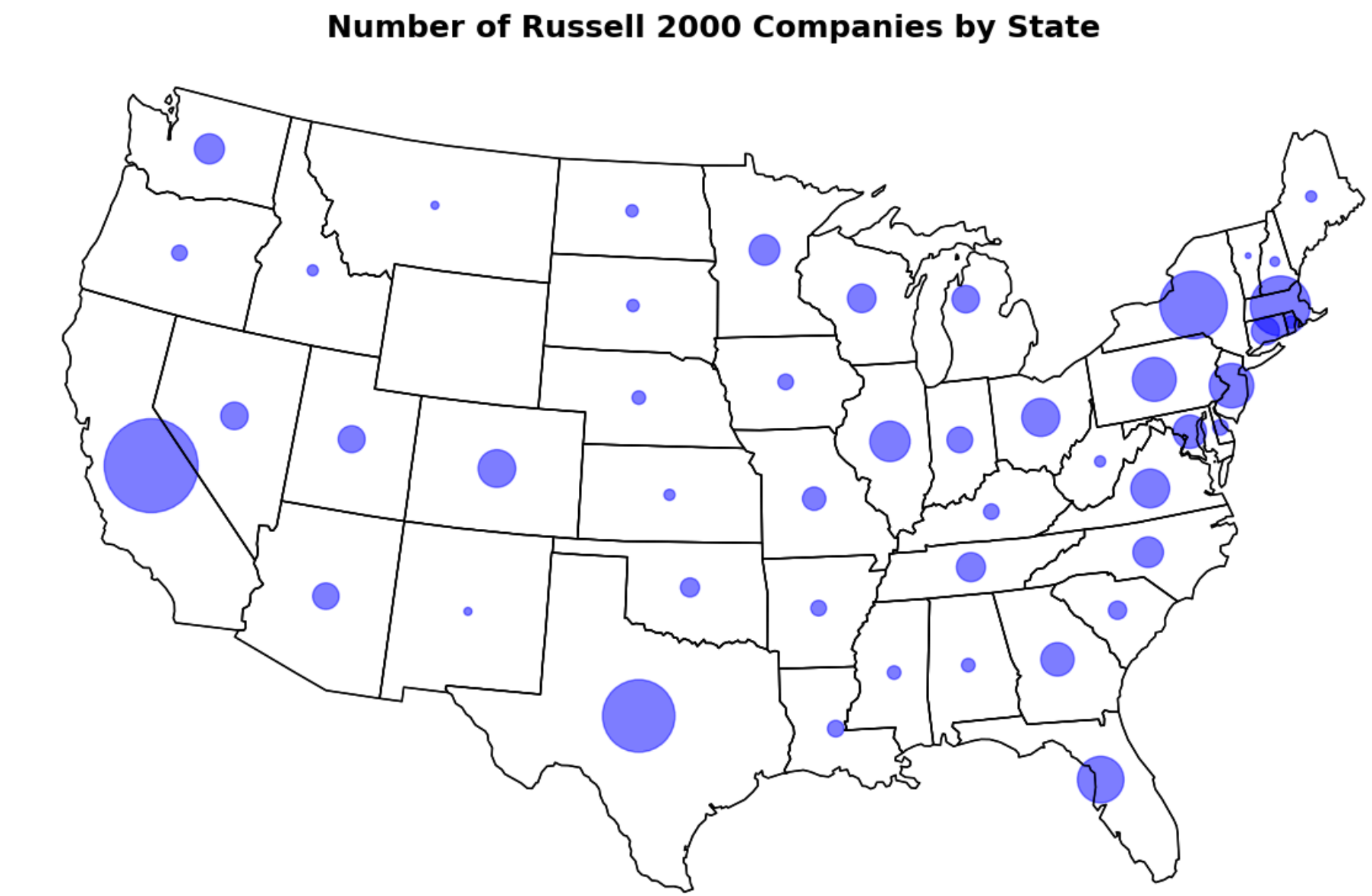

The COVID-19 pandemic dramatically increased the volatility gap between small-cap and large-cap stocks. The dispariry has persisted well after the pandemic, suggesting that small-cap stocks are inherently more vulnerable to economic shocks.

## Maps and Interpretation

In [22]:
```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
import numpy as np
# Fix PROJ error by setting environment variable
os.environ['PROJ_LIB'] = '/usr/share/proj'

# Load Russell 2000 dataset
file_path = 'russell_2000_with_states.xlsx'  # Update with your file path
df = pd.read_excel(file_path)
```

```python
df = df.dropna(subset=['State'])

# Mapping state abbreviations to full names
state_abbrev_to_name = {
    'AL': 'Alabama', 'AK': 'Alaska', 'AZ': 'Arizona', 'AR': 'Arkansas', 'CA': 'California', 'CO': 'Colorado',
    'CT': 'Connecticut', 'DE': 'Delaware', 'FL': 'Florida', 'GA': 'Georgia', 'HI': 'Hawaii', 'ID': 'Idaho',
    'IL': 'Illinois', 'IN': 'Indiana', 'IA': 'Iowa', 'KS': 'Kansas', 'KY': 'Kentucky', 'LA': 'Louisiana',
    'ME': 'Maine', 'MD': 'Maryland', 'MA': 'Massachusetts', 'MI': 'Michigan', 'MN': 'Minnesota', 'MS': 'Mississippi',
    'MO': 'Missouri', 'MT': 'Montana', 'NE': 'Nebraska', 'NV': 'Nevada', 'NH': 'New Hampshire', 'NJ': 'New Jersey',
    'NM': 'New Mexico', 'NY': 'New York', 'NC': 'North Carolina', 'ND': 'North Dakota', 'OH': 'Ohio', 'OK': 'Oklahoma',
    'OR': 'Oregon', 'PA': 'Pennsylvania', 'RI': 'Rhode Island', 'SC': 'South Carolina', 'SD': 'South Dakota',
    'TN': 'Tennessee', 'TX': 'Texas', 'UT': 'Utah', 'VT': 'Vermont', 'VA': 'Virginia', 'WA': 'Washington',
    'WV': 'West Virginia', 'WI': 'Wisconsin', 'WY': 'Wyoming'
}
df['State'] = df['State'].map(state_abbrev_to_name)

# Aggregate company count per state
state_counts = df['State'].value_counts().reset_index()
state_counts.columns = ['State', 'Company Count']

# Load US states geometry
gdf = gpd.read_file('https://raw.githubusercontent.com/PublicaMundi/MappingAPI/master/data/geojson/us-states.json')

# Ensure consistent state naming
gdf['name'] = gdf['name'].str.strip()
state_counts['State'] = state_counts['State'].str.strip()

# Merge company count data with US map
gdf = gdf.merge(state_counts, left_on='name', right_on='State', how='left')
gdf['Company Count'] = gdf['Company Count'].fillna(0)

# Exclude non-contiguous states and small islands
gdf = gdf[~gdf['name'].isin(['Hawaii', 'Alaska', 'Puerto Rico', 'Guam', 'American Samoa',
                             'Northern Mariana Islands', 'U.S. Virgin Islands'])]

# Reproject to a projected CRS to avoid centroid warnings (using EPSG:5070 - US National Atlas Equal Area)
gdf = gdf.to_crs(epsg=5070)

# Compute centroids in the projected CRS
centroids = gdf.centroid

# Plot the map with proportional symbols
fig, ax = plt.subplots(figsize=(15, 10))
gdf.boundary.plot(ax=ax, linewidth=1, edgecolor='black')
gdf.plot(color='white', edgecolor='black', ax=ax)

# Normalize company count for plotting
company_count_normalized = gdf['Company Count'] / gdf['Company Count'].max()

# Add circles at the centroids with size proportional to the number of companies
# Increase the scaling factor (e.g., 3000) to make the circles larger
centroids.plot(ax=ax, markersize=company_count_normalized * 3000, color='blue', alpha=0.5)
```

```
# Improve visualization: title, remove ticks and axis
ax.set_title('Number of Russell 2000 Companies by State', fontsize=18, fontweight='bold')
ax.set_xticks([])
ax.set_yticks([])
plt.axis('off')

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



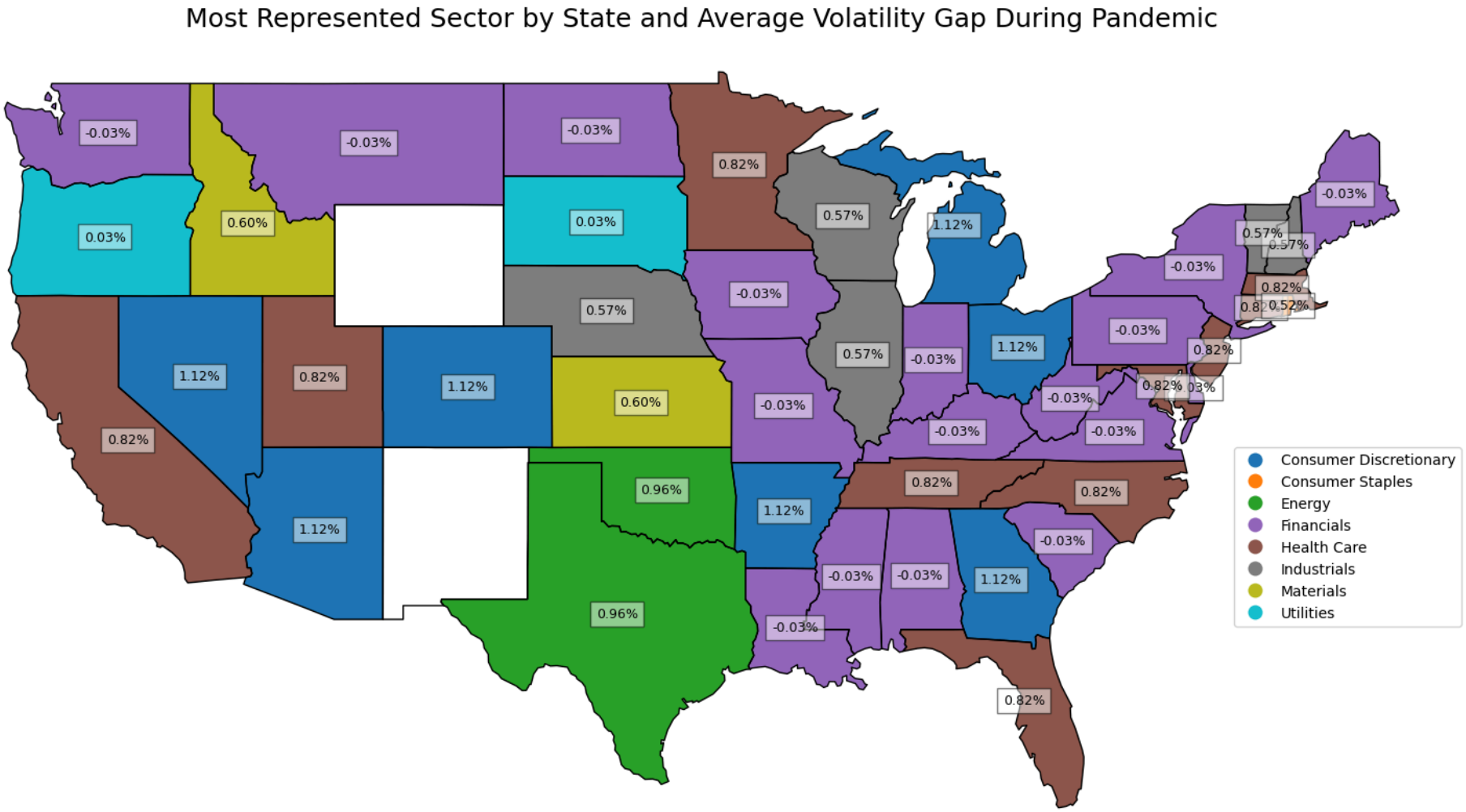

This map is intended to help display where the majority of the listed firms on the Russell 2000 reside in the United States. For example, Texas, Maine, New York, and Florida have a large amount of listed small-cap firms. The purpose of this map is to help show how the maps following were created.

In [23]:

```
# Load Russell 2000 dataset
file_path = 'russell_2000_with_states.xlsx'  # Update with your file path
df = pd.read_excel(file_path)

# Ensure required columns are present
df = df.dropna(subset=['State', 'Sector'])

# Mapping state abbreviations to full names
state_abbrev_to_name = {
    'AL': 'Alabama', 'AK': 'Alaska', 'AZ': 'Arizona', 'AR': 'Arkansas', 'CA': 'California', 'CO': 'Colorado',
    'CT': 'Connecticut', 'DE': 'Delaware', 'FL': 'Florida', 'GA': 'Georgia', 'HI': 'Hawaii', 'ID': 'Idaho',
    'IL': 'Illinois', 'IN': 'Indiana', 'IA': 'Iowa', 'KS': 'Kansas', 'KY': 'Kentucky', 'LA': 'Louisiana',
    'ME': 'Maine', 'MD': 'Maryland', 'MA': 'Massachusetts', 'MI': 'Michigan', 'MN': 'Minnesota', 'MS': 'Mississippi',
    'MO': 'Missouri', 'MT': 'Montana', 'NE': 'Nebraska', 'NV': 'Nevada', 'NH': 'New Hampshire', 'NJ': 'New Jersey',
    'NM': 'New Mexico', 'NY': 'New York', 'NC': 'North Carolina', 'ND': 'North Dakota', 'OH': 'Ohio', 'OK': 'Oklahoma',
    'OR': 'Oregon', 'PA': 'Pennsylvania', 'RI': 'Rhode Island', 'SC': 'South Carolina', 'SD': 'South Dakota',
    'TN': 'Tennessee', 'TX': 'Texas', 'UT': 'Utah', 'VT': 'Vermont', 'VA': 'Virginia', 'WA': 'Washington',
    'WV': 'West Virginia', 'WI': 'Wisconsin', 'WY': 'Wyoming'
}
df['State'] = df['State'].map(state_abbrev_to_name)

# Group by State and Sector and count companies
sector_counts = df.groupby(['State', 'Sector']).size().reset_index(name='Company Count')

# Identify the sector with the most companies for each state
top_sector_by_state = sector_counts.loc[sector_counts.groupby(['State'])['Company Count'].idxmax()]

# Load US states geometry
gdf = gpd.read_file('https://raw.githubusercontent.com/PublicaMundi/MappingAPI/master/data/geojson/us-states.json')
gdf = gdf[~gdf["name"].isin(["Hawaii", "Alaska", "Puerto Rico", "Guam", "American Samoa", "Northern Mariana Islands", "U.S. Virgin Islands"])]

# Rename 'name' to 'State' in gdf for merging
gdf = gdf.rename(columns={'name': 'State'})



# Load the daily volatility gap data
daily_volatility_gap_csv = 'daily_volatility_gap.csv'  # Update with your file path
volatility_df = pd.read_csv(daily_volatility_gap_csv)

# Filter for dates within the pandemic
start_pandemic = '2020-03-11'
end_pandemic = '2021-05-05'
mask = (volatility_df['Date'] >= start_pandemic) & (volatility_df['Date'] <= end_pandemic)
volatility_df_during_pandemic = volatility_df.loc[mask]

# Calculate the average volatility gap for each sector during the pandemic
volatility_df_during_pandemic = volatility_df_during_pandemic.drop(columns=['Date'])
avg_volatility_gap = volatility_df_during_pandemic.mean().reset_index()
avg_volatility_gap.columns = ['Sector', 'Average Volatility Gap']

# Clean sector names in avg_volatility_gap
```

```
avg_volatility_gap['Sector'] = avg_volatility_gap['Sector'].str.replace(' Daily Volatility Gap', '', regex=True)

# Multiply the average volatility gap by 100 to get percentages
avg_volatility_gap['Average Volatility Gap'] *= 100

# Fix sector mismatches by mapping them correctly
sector_mapping = {
    'Financial Services': 'Financials',
    'Consumer Cyclical': 'Consumer Discretionary',
    'Consumer Defensive': 'Consumer Staples',
    'Basic Materials': 'Materials',
    'Healthcare': 'Health Care',
    'Industrials': 'Industrials',
    'Energy': 'Energy',
    'Utilities': 'Utilities',
    'Real Estate': 'Real Estate',
    'Technology': 'Technology',
    'Communication Services': 'Communication Services'
}

# Apply the sector mapping
top_sector_by_state['Sector'] = top_sector_by_state['Sector'].replace(sector_mapping)

# Merge the average volatility gap data with the top sector data
top_sector_with_avg_gap = top_sector_by_state.merge(avg_volatility_gap, on='Sector', how='left')

# Ensure every state has a volatility gap value by filling missing ones with sector average
sector_avg_gaps = top_sector_with_avg_gap.groupby('Sector')['Average Volatility Gap'].mean().to_dict()
top_sector_with_avg_gap['Average Volatility Gap'] = top_sector_with_avg_gap.apply(
    lambda row: sector_avg_gaps[row['Sector']] if pd.isna(row['Average Volatility Gap']) else row['Average Volatility Gap'],
    axis=1
)

# Standardize state and sector names before merging
gdf['State'] = gdf['State'].astype(str).str.strip()
top_sector_with_avg_gap['State'] = top_sector_with_avg_gap['State'].astype(str).str.strip()
top_sector_with_avg_gap['Sector'] = top_sector_with_avg_gap['Sector'].astype(str).str.strip()

# Check for missing states before merging
missing_states = set(gdf['State']) - set(top_sector_with_avg_gap['State'])


# Merge with US states map
gdff = gdf.merge(top_sector_with_avg_gap, on=['State', 'State'], how='left')

# Ensure no states are missing
missing_after_merge = gdff[gdff['Average Volatility Gap'].isna()][['State', 'Sector']]


# Plot the map
fig, ax = plt.subplots(1, 1, figsize=(15, 10))
gdff.boundary.plot(ax=ax, linewidth=1, edgecolor='black')
gdff.plot(ax=ax, column='Sector', categorical=True, legend=True, legend_kwds={'bbox_to_anchor': (1, 0.5)})
```

```
# Annotate the average volatility gap for each state
for idx, row in gdff.iterrows():
    if pd.notnull(row['Sector']) and pd.notnull(row['Average Volatility Gap']):
        plt.text(row.geometry.centroid.x, row.geometry.centroid.y,
                 f"{row['Average Volatility Gap']:.2f}%",
                 fontsize=9, ha='center', va='center', color='black', bbox=dict(facecolor='white', alpha=0.5))

ax.set_title('Most Represented Sector by State and Average Volatility Gap During Pandemic', fontsize=18)
ax.set_axis_off()
plt.tight_layout()
plt.savefig("A map", dpi=300)
plt.show()
```




This map shows the average volatility gap for each state during the pandemic, derived from the dominant sector of small-cap companies in that state. The states without a color do not have a majority in any one sector. There are specific sectors that have a large average difference in volatility, namely energy, healthcare, and consumer discretionary. This suggests that certain industries had a larger reaction in small-cap volatility compared to others.

In [42]:

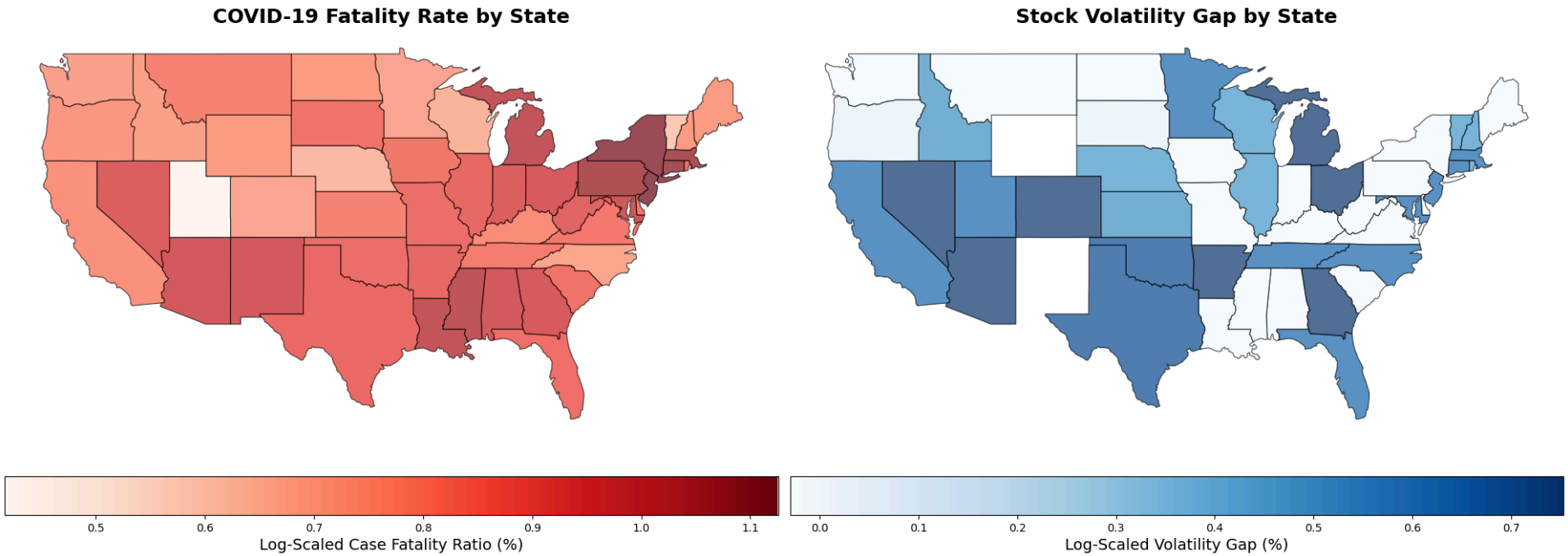
```
# Load Russell 2000 dataset
russell_file = "russell_2000_with_states.xlsx"  # Update with your file path
df = pd.read_excel(russell_file)

# Ensure required columns are present
df = df.dropna(subset=["State", "Sector"])

# Mapping state abbreviations to full names
state_abbrev_to_name = {
    "AL": "Alabama", "AK": "Alaska", "AZ": "Arizona", "AR": "Arkansas", "CA": "California",
    "CO": "Colorado", "CT": "Connecticut", "DE": "Delaware", "FL": "Florida", "GA": "Georgia",
    "HI": "Hawaii", "ID": "Idaho", "IL": "Illinois", "IN": "Indiana", "IA": "Iowa",
    "KS": "Kansas", "KY": "Kentucky", "LA": "Louisiana", "ME": "Maine", "MD": "Maryland",
    "MA": "Massachusetts", "MI": "Michigan", "MN": "Minnesota", "MS": "Mississippi",
    "MO": "Missouri", "MT": "Montana", "NE": "Nebraska", "NV": "Nevada", "NH": "New Hampshire",
    "NJ": "New Jersey", "NM": "New Mexico", "NY": "New York", "NC": "North Carolina",
    "ND": "North Dakota", "OH": "Ohio", "OK": "Oklahoma", "OR": "Oregon", "PA": "Pennsylvania",
    "RI": "Rhode Island", "SC": "South Carolina", "SD": "South Dakota", "TN": "Tennessee",
    "TX": "Texas", "UT": "Utah", "VT": "Vermont", "VA": "Virginia", "WA": "Washington",
    "WV": "West Virginia", "WI": "Wisconsin", "WY": "Wyoming"
}
df["State"] = df["State"].map(state_abbrev_to_name)

# Group by State and Sector and count companies
sector_counts = df.groupby(["State", "Sector"]).size().reset_index(name="Company Count")

# Identify the most represented sector in each state
top_sector_by_state = sector_counts.loc[sector_counts.groupby(["State"])["Company Count"].idxmax()]

# Load US states geometry
gdf = gpd.read_file("https://raw.githubusercontent.com/PublicaMundi/MappingAPI/master/data/geojson/us-states.json")
gdf = gdf[~gdf["name"].isin(["Hawaii", "Alaska", "Puerto Rico", "Guam", "American Samoa", "Northern Mariana Islands", "U.S. Virgin Islands"])]
gdf = gdf.rename(columns={"name": "State"})

# Load the daily volatility gap data
volatility_file = "daily_volatility_gap.csv"  # Update with your file path
volatility_df = pd.read_csv(volatility_file)

# Filter for pandemic period
start_pandemic = "2020-03-11"
end_pandemic = "2021-05-05"
volatility_df["Date"] = pd.to_datetime(volatility_df["Date"])
mask = (volatility_df["Date"] >= start_pandemic) & (volatility_df["Date"] <= end_pandemic)
volatility_df_during_pandemic = volatility_df.loc[mask]

# Calculate the average volatility gap for each sector
volatility_df_during_pandemic = volatility_df_during_pandemic.drop(columns=["Date"])
avg_volatility_gap = volatility_df_during_pandemic.mean().reset_index()
avg_volatility_gap.columns = ["Sector", "Average Volatility Gap"]

# Clean sector names
avg_volatility_gap["Sector"] = avg_volatility_gap["Sector"].str.replace(" Daily Volatility Gap", "", regex=True)
```

```
avg_volatility_gap["Average Volatility Gap"] *= 100  # Convert to percentage

# Map sectors to consistent names
sector_mapping = {
    "Financial Services": "Financials",
    "Consumer Cyclical": "Consumer Discretionary",
    "Consumer Defensive": "Consumer Staples",
    "Basic Materials": "Materials",
    "Healthcare": "Health Care",
    "Industrials": "Industrials",
    "Energy": "Energy",
    "Utilities": "Utilities",
    "Real Estate": "Real Estate",
    "Technology": "Technology",
    "Communication Services": "Communication Services"
}
top_sector_by_state["Sector"] = top_sector_by_state["Sector"].replace(sector_mapping)

# Merge volatility gap data with sector data
top_sector_with_avg_gap = top_sector_by_state.merge(avg_volatility_gap, on="Sector", how="left")

# Ensure each state has a volatility value
sector_avg_gaps = top_sector_with_avg_gap.groupby("Sector")["Average Volatility Gap"].mean().to_dict()
top_sector_with_avg_gap["Average Volatility Gap"] = top_sector_with_avg_gap.apply(
    lambda row: sector_avg_gaps[row["Sector"]] if pd.isna(row["Average Volatility Gap"]) else row["Average Volatility Gap"],
    axis=1
)

# Merge with US map
gdf = gdf.merge(top_sector_with_avg_gap, on="State", how="left")

# Load COVID-19 CFR data
covid_file = "covid_aggregated_by_state.csv"
covid_df = pd.read_csv(covid_file)

# Compute the average case fatality ratio per state
covid_avg = covid_df.groupby("Province_State", as_index=False)["Case_Fatality_Ratio"].mean()
covid_avg.rename(columns={"Province_State": "State", "Case_Fatality_Ratio": "Avg_Fatality_Rate"}, inplace=True)

# Merge CFR data with state map
gdf = gdf.merge(covid_avg, on="State", how="left")

# Log-scale data for better visualization
gdf["Fatality Rate Log"] = np.log1p(gdf["Avg_Fatality_Rate"])
gdf["Volatility Gap Log"] = np.log1p(gdf["Average Volatility Gap"])
```

In [43]:

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
import numpy as np

# Assuming you have already processed your data as per the provided code
```

```python
# Plot the maps
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))  # Set up a subplot grid with 1 row and 2 columns

# **Plot Case Fatality Ratio (CFR) on the first subplot**
gdf.plot(column='Fatality Rate Log', cmap='Reds', linewidth=0.8, edgecolor='black', alpha=0.7, ax=ax1)
ax1.set_title('COVID-19 Fatality Rate by State', fontsize=18, fontweight='bold')
ax1.set_xticks([])
ax1.set_yticks([])
ax1.axis('off')

# **Add Color Bar for Fatality Ratio below the first map**
sm_fatality = plt.cm.ScalarMappable(cmap='Reds', norm=plt.Normalize(vmin=gdf['Fatality Rate Log'].min(), vmax=gdf['Fatality Rate Log'].max()))
cbar_fatality = fig.colorbar(sm_fatality, ax=ax1, orientation='horizontal', pad=0.05)
cbar_fatality.set_label('Log-Scaled Case Fatality Ratio (%)', fontsize=14)  # Increase font size here

# **Plot Volatility Gap on the second subplot**
gdf.plot(column='Volatility Gap Log', cmap='Blues', linewidth=0.8, edgecolor='black', alpha=0.7, ax=ax2)
ax2.set_title('Stock Volatility Gap by State', fontsize=18, fontweight='bold')
ax2.set_xticks([])
ax2.set_yticks([])
ax2.axis('off')

# **Add Color Bar for Volatility Gap below the second map**
sm_volatility = plt.cm.ScalarMappable(cmap='Blues', norm=plt.Normalize(vmin=gdf['Volatility Gap Log'].min(), vmax=gdf['Volatility Gap Log'].max()))
cbar_volatility = fig.colorbar(sm_volatility, ax=ax2, orientation='horizontal', pad=0.05)
cbar_volatility.set_label('Log-Scaled Volatility Gap (%)', fontsize=14)  # Increase font size here

# Show final plots
plt.tight_layout()
plt.savefig("A3 MAp", dpi=300)
plt.show()
```




This two map illustrates the relationship between percentage of COVID-19 case fatality ratios (from the Johns Hopkins dataset) and log-transformed stock volatility gaps across U.S. states using methodolgy from the previous map. The map on the right shades in states with higher volatilty gap, the map on the left shades in based off fatality rates. Based on the previous map and these results, states such as Texas, Maine, and New York, where volatility gaps were high, also exhibit higher case fatality rates, suggesting that there is a potential relationship between the two.

In [45]:
```
# Fix PROJ error by setting environment variable (adjust the path if needed)
os.environ['PROJ_LIB'] = '/usr/share/proj'

# ============================================================
# STEP 1: Load Russell 2000 Data and Map State Abbreviations
# ============================================================
file_path = 'russell_2000_with_states.xlsx'
df = pd.read_excel(file_path)
df = df.dropna(subset=['State', 'Sector'])

state_abbrev_to_name = {
    'AL': 'Alabama', 'AK': 'Alaska', 'AZ': 'Arizona', 'AR': 'Arkansas', 'CA': 'California', 'CO': 'Colorado',
    'CT': 'Connecticut', 'DE': 'Delaware', 'FL': 'Florida', 'GA': 'Georgia', 'HI': 'Hawaii', 'ID': 'Idaho',
    'IL': 'Illinois', 'IN': 'Indiana', 'IA': 'Iowa', 'KS': 'Kansas', 'KY': 'Kentucky', 'LA': 'Louisiana',
    'ME': 'Maine', 'MD': 'Maryland', 'MA': 'Massachusetts', 'MI': 'Michigan', 'MN': 'Minnesota', 'MS': 'Mississippi',
    'MO': 'Missouri', 'MT': 'Montana', 'NE': 'Nebraska', 'NV': 'Nevada', 'NH': 'New Hampshire', 'NJ': 'New Jersey',
    'NM': 'New Mexico', 'NY': 'New York', 'NC': 'North Carolina', 'ND': 'North Dakota', 'OH': 'Ohio', 'OK': 'Oklahoma',
    'OR': 'Oregon', 'PA': 'Pennsylvania', 'RI': 'Rhode Island', 'SC': 'South Carolina', 'SD': 'South Dakota',
    'TN': 'Tennessee', 'TX': 'Texas', 'UT': 'Utah', 'VT': 'Vermont', 'VA': 'Virginia', 'WA': 'Washington',
    'WV': 'West Virginia', 'WI': 'Wisconsin', 'WY': 'Wyoming'
}
df['State'] = df['State'].map(state_abbrev_to_name)
```

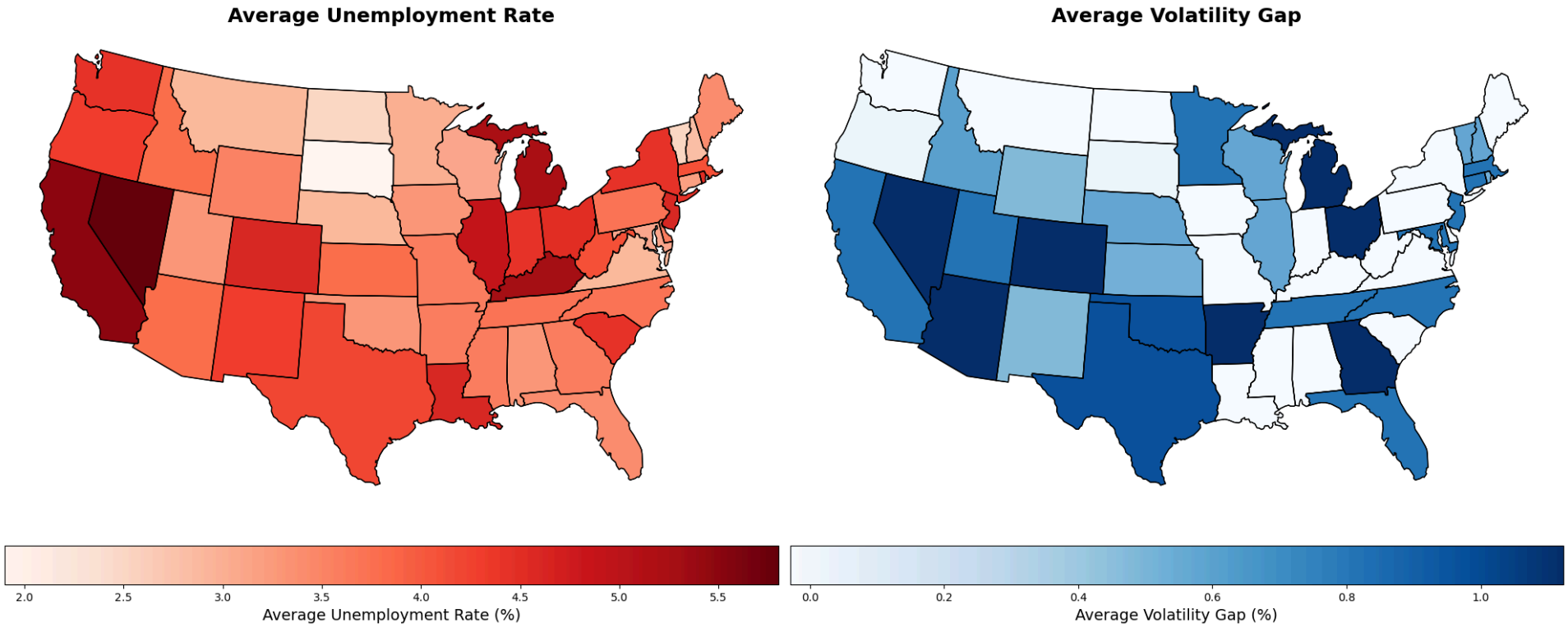
```python
# ============================================================
# STEP 2: Compute Company Count per State
# ============================================================
state_counts = df['State'].value_counts().reset_index()
state_counts.columns = ['State', 'Company Count']

# ============================================================
# STEP 3: Load US States Geometry and Merge Company Count Data
# ============================================================
gdf = gpd.read_file('https://raw.githubusercontent.com/PublicaMundi/MappingAPI/master/data/geojson/us-states.json')
gdf = gdf[~gdf["name"].isin(["Hawaii", "Alaska", "Puerto Rico", "Guam", "American Samoa",
                             "Northern Mariana Islands", "U.S. Virgin Islands"])]
gdf = gdf.rename(columns={'name': 'State'})
gdf['State'] = gdf['State'].str.strip()
state_counts['State'] = state_counts['State'].str.strip()
gdf = gdf.merge(state_counts, on='State', how='left')
gdf['Company Count'] = gdf['Company Count'].fillna(0)

# ============================================================
# STEP 4: Reproject Geometry and Compute Centroids
# ============================================================
gdf = gdf.to_crs(epsg=5070)
gdf["centroid"] = gdf.geometry.centroid

# ============================================================
# STEP 5: Load Unemployment Data and Compute Average Unemployment Rate During the Pandemic
# ============================================================
df_unemployment = pd.read_csv("state_unemployment_rates.csv")
df_unemployment['State'] = df_unemployment['State'].str.strip()
# (Optional: print columns to check the name)
#print("Unemployment Data Columns:", df_unemployment.columns)
# If your unemployment CSV column is named "Unemployment_Rate", rename it to "Unemployment Rate"
df_unemployment = df_unemployment.rename(columns={"Unemployment_Rate": "Unemployment Rate"})

# If there are multiple observations per state (e.g., by month), filter by date if applicable
if "Date" in df_unemployment.columns:
    df_unemployment['Date'] = pd.to_datetime(df_unemployment['Date'])
    mask = (df_unemployment['Date'] >= "2020-03-11") & (df_unemployment['Date'] <= "2021-05-05")
    df_unemployment = df_unemployment.loc[mask]

avg_unemployment = df_unemployment.groupby("State")["Unemployment Rate"].mean().reset_index()
# ============================================================
# Merge average unemployment with the geometry DataFrame
# ============================================================
gdf = gdf.merge(avg_unemployment, on="State", how="left")

# ============================================================
# STEP 6: Load Daily Volatility Gap Data and Compute Average Volatility Gap per Sector
# ============================================================
daily_volatility_gap_csv = 'daily_volatility_gap.csv'
volatility_df = pd.read_csv(daily_volatility_gap_csv)
volatility_df['Date'] = pd.to_datetime(volatility_df['Date'])
start_pandemic, end_pandemic = "2020-03-11", "2021-05-05"
```

```
mask = (volatility_df['Date'] >= start_pandemic) & (volatility_df['Date'] <= end_pandemic)
volatility_df_during_pandemic = volatility_df.loc[mask]
volatility_df_during_pandemic = volatility_df_during_pandemic.drop(columns=['Date'])
avg_volatility_gap = volatility_df_during_pandemic.mean().reset_index()
avg_volatility_gap.columns = ['Sector', 'Average Volatility Gap']
avg_volatility_gap['Sector'] = avg_volatility_gap['Sector'].str.replace(' Daily Volatility Gap', '', regex=True)
avg_volatility_gap['Average Volatility Gap'] *= 100

# ============================================================
# STEP 7: Determine Dominant Sector per State
# ============================================================
sector_mapping = {
    'Financial Services': 'Financials',
    'Consumer Cyclical': 'Consumer Discretionary',
    'Consumer Defensive': 'Consumer Staples',
    'Basic Materials': 'Materials',
    'Healthcare': 'Health Care',
    'Industrials': 'Industrials',
    'Energy': 'Energy',
    'Utilities': 'Utilities',
    'Real Estate': 'Real Estate',
    'Technology': 'Technology',
    'Communication Services': 'Communication Services'
}
df['Sector'] = df['Sector'].replace(sector_mapping)
sector_counts = df.groupby(['State', 'Sector']).size().reset_index(name='Company Count')
top_sector_by_state = sector_counts.loc[sector_counts.groupby('State')['Company Count'].idxmax()]

# ============================================================
# STEP 8: Merge Average Volatility Gap with Dominant Sector Data and Merge with Geometry
# ============================================================
top_sector_with_avg_gap = top_sector_by_state.merge(avg_volatility_gap, on='Sector', how='left')
sector_avg_dict = top_sector_with_avg_gap.groupby('Sector')['Average Volatility Gap'].mean().to_dict()
top_sector_with_avg_gap['Average Volatility Gap'] = top_sector_with_avg_gap.apply(
    lambda row: sector_avg_dict[row['Sector']] if pd.isna(row['Average Volatility Gap']) else row['Average Volatility Gap'],
    axis=1
)
gdf['State'] = gdf['State'].astype(str).str.strip()
top_sector_with_avg_gap['State'] = top_sector_with_avg_gap['State'].astype(str).str.strip()
gdf = gdf.merge(top_sector_with_avg_gap[['State', 'Average Volatility Gap']], on='State', how='left')
gdf['Average Volatility Gap'].fillna(gdf['Average Volatility Gap'].mean(), inplace=True)

# ============================================================
# STEP 9: Plot the Map
# ============================================================
```

In [46]:

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
import os
```

```python
# Fix PROJ error by setting environment variable (adjust the path if needed)
os.environ['PROJ_LIB'] = '/usr/share/proj'

# Assuming all your data processing steps have been completed as per the provided code

# Plot the maps
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))  # Set up a subplot grid with 1 row and 2 columns

# **Plot Average Unemployment Rate on the first subplot using a red color scheme**
gdf.boundary.plot(ax=ax1, linewidth=1, edgecolor='black')
gdf.plot(ax=ax1, column='Unemployment Rate', cmap='Reds', linewidth=0.8, edgecolor='black')
ax1.set_title('Average Unemployment Rate', fontsize=18, fontweight='bold')
ax1.set_xticks([])
ax1.set_yticks([])
ax1.axis('off')

# **Add Color Bar for Unemployment Rate below the first map**
norm_unemployment = mpl.colors.Normalize(vmin=gdf['Unemployment Rate'].min(), vmax=gdf['Unemployment Rate'].max())
sm_unemployment = mpl.cm.ScalarMappable(cmap='Reds', norm=norm_unemployment)
sm_unemployment._A = []  # Workaround for ScalarMappable
cbar_unemployment = fig.colorbar(sm_unemployment, ax=ax1, orientation='horizontal', pad=0.05)
cbar_unemployment.set_label('Average Unemployment Rate (%)', fontsize=14)

# **Plot Average Volatility Gap on the second subplot using a blue color scheme**
gdf.boundary.plot(ax=ax2, linewidth=1, edgecolor='black')
gdf.plot(ax=ax2, column='Average Volatility Gap', cmap='Blues', linewidth=0.8, edgecolor='black')
ax2.set_title('Average Volatility Gap', fontsize=18, fontweight='bold')
ax2.set_xticks([])
ax2.set_yticks([])
ax2.axis('off')

# **Add Color Bar for Volatility Gap below the second map**
norm_volatility = mpl.colors.Normalize(vmin=gdf['Average Volatility Gap'].min(), vmax=gdf['Average Volatility Gap'].max())
sm_volatility = mpl.cm.ScalarMappable(cmap='Blues', norm=norm_volatility)
sm_volatility._A = []  # Workaround for ScalarMappable
cbar_volatility = fig.colorbar(sm_volatility, ax=ax2, orientation='horizontal', pad=0.05)
cbar_volatility.set_label('Average Volatility Gap (%)', fontsize=14)

# Show final plots
plt.tight_layout()
plt.show()
```




The two map above displays the average unemployment rate per state with the average volailty gap per state during the pandemic. The visualizations show us that there is not strong relationship between the unemployment and the average volatilty gap, but still a small correlation nonetheless. As, states with the highest volatility gaps did not consistently have the highest unemployment rates.

## Regressions

In [36]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
from stargazer.stargazer import Stargazer
from IPython.display import display, HTML
from bs4 import BeautifulSoup

# ------------------------------
# STEP 1: Load and Prepare Data
# ------------------------------
# Load the merged dataset with daily observations
df = pd.read_csv("cleaned_merged_data.csv", parse_dates=["Date"])

# Multiply Daily_Volatility_Gap by 100
df["Daily_Volatility_Gap"] = df["Daily_Volatility_Gap"] * 100

# Rename columns for simplicity
```

```python
df = df.rename(columns={
    "Interest Rate (Fed Funds)": "Interest_Rate",
    "Unemployment Rate": "Unemp_Rate",
    "Inflation (CPI)": "CPI"
})

# Define a function to categorize periods
def categorize_period(date):
    if date < pd.Timestamp("2020-03-11"):
        return "Pre-Pandemic"
    elif pd.Timestamp("2020-03-11") <= date <= pd.Timestamp("2023-05-05"):
        return "Pandemic"
    else:
        return "Post-Pandemic"

df["Period"] = df["Date"].apply(categorize_period)
df["Period"] = pd.Categorical(df["Period"],
                              categories=["Pre-Pandemic", "Pandemic", "Post-Pandemic"],
                              ordered=True)

# Create separate dataframes for each period
df_pre = df[df["Period"] == "Pre-Pandemic"]
df_pandemic = df[df["Period"] == "Pandemic"]
df_post = df[df["Period"] == "Post-Pandemic"]

# ------------------------------
# STEP 2: Run Period-Specific Regressions
# ------------------------------
# Pre-Pandemic Regression (economic controls only)
model_pre = smf.ols("Daily_Volatility_Gap ~ Interest_Rate + Unemp_Rate + CPI", data=df_pre).fit()

# Pandemic Regression (adds COVID variables)
model_pandemic = smf.ols(
    "Daily_Volatility_Gap ~ Interest_Rate + Unemp_Rate + CPI + New_Cases_7D_Rolling + Case_Fatality_Ratio",
    data=df_pandemic
).fit()

# Post-Pandemic Regression (COVID variables not included)
model_post = smf.ols("Daily_Volatility_Gap ~ Interest_Rate + Unemp_Rate + CPI", data=df_post).fit()

# ------------------------------
# STEP 3: Create and Format the Regression Table with Stargazer
# ------------------------------
# We are using only the period-specific models.
stargazer_period = Stargazer([model_pre, model_pandemic, model_post])
stargazer_period.custom_columns(['Pre-Pandemic', 'Pandemic', 'Post-Pandemic'], [1, 1, 1])
stargazer_period.significant_digits(3)

# Set a custom title (since built-in title styling is limited)
custom_title = """
<h1 style="font-size:22px; font-weight:bold; text-align:left; margin-bottom:20px;">
Baseline Analysis of Daily Volatility Gap Trends Across Economic Conditions
</h1>
```

```
"""
html_output = stargazer_period.render_html()
final_html = custom_title + html_output

# ------------------------------
# STEP 4: Move the Intercept Row to the Bottom Using BeautifulSoup
# ------------------------------
soup = BeautifulSoup(final_html, 'html.parser')
table = soup.find('table')
rows = table.find_all('tr')

intercept_row = None
for row in rows:
    cells = row.find_all('td')
    if cells and 'Intercept' in cells[0].get_text():
        intercept_row = row
        break

if intercept_row:
    intercept_row.extract()   # Remove the intercept row from its original location
    table.append(intercept_row) # Append it at the end of the table

modified_html = str(soup)

# ------------------------------
# STEP 5: Display the Final Regression Table
# ------------------------------
display(HTML(modified_html))
```

# Baseline Analysis of Daily Volatility Gap Trends Across Economic Conditions

| | *Dependent variable: Daily_Volatility_Gap* | | |
|---|---|---|---|
| | Pre-Pandemic | Pandemic | Post-Pandemic |
| | (1) | (2) | (3) |
| CPI | -0.018*** | -0.003 | 0.006 |
| | (0.004) | (0.002) | (0.011) |
| Case_Fatality_Ratio | | -0.022 | |
| | | (0.018) | |
| | (1.314) | (0.572) | (2.911) |
| Interest_Rate | 0.216*** | -0.011 | 0.035 |
| | (0.024) | (0.015) | (0.080) |
| New_Cases_7D_Rolling | | 0.000 | |
| | | (0.000) | |
| Unemp_Rate | -0.681*** | 0.055*** | -0.022 |
| | (0.074) | (0.011) | (0.201) |
| Observations | 550 | 795 | 415 |
| $R^2$ | 0.349 | 0.213 | 0.003 |
| Adjusted $R^2$ | 0.345 | 0.208 | -0.005 |
| Residual Std. Error | 0.197 (df=546) | 0.338 (df=789) | 0.355 (df=411) |
| F Statistic | 97.540*** (df=3; 546) | 42.812*** (df=5; 789) | 0.355 (df=3; 411) |
| Note: | | | *p<0.1; **p<0.05; ***p<0.01 |
| Intercept | 6.807*** | 0.970* | -1.422 |

This regression analysis serves to establish baseline levels of the daily volatility gap across the three periods of study. By comparing these periods, we can observe how the behavior of the volatility gap shifted over time.Prior to the pandemic, a one-point increase in CPI was associated with a 0.018-point decrease in the volatility gap, while a similar increase in interest rates correlated with a 0.216-point increase. Both results were statistically significant at the 1 percent level.During the pandemic,

only the unemployment rate had a statistically significant impact—the case fatality ratio did not. After the pandemic, none of the indicators had a statistically significant effect, suggesting that the factors influencing the volatility gap may have changed permanently since COVID-19.

In [58]:
```
# --- Load and Preprocess ---
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
from stargazer.stargazer import Stargazer
from bs4 import BeautifulSoup
from IPython.display import display, HTML

# ✅ Load Data
df = pd.read_csv("cleaned_merged_data.csv", parse_dates=["Date"])

# ✅ Convert volatility gap to %
df["Daily_Volatility_Gap"] *= 100

# ✅ Rename columns (same as you did earlier)
df = df.rename(columns={
    "Interest Rate (Fed Funds)": "Interest_Rate",
    "Unemployment Rate": "Unemp_Rate",
    "Inflation (CPI)": "CPI"
})

# ✅ Create Period variable
def categorize_period(date):
    if date < pd.Timestamp("2020-03-11"):
        return "Pre-Pandemic"
    elif pd.Timestamp("2020-03-11") <= date <= pd.Timestamp("2023-05-05"):
        return "Pandemic"
    else:
        return "Post-Pandemic"

df["Period"] = df["Date"].apply(categorize_period)
df["Period"] = pd.Categorical(df["Period"],
                              categories=["Pre-Pandemic", "Pandemic", "Post-Pandemic"],
                              ordered=True)

# ✅ Create dummies like you originally did (keeping "Period_PostPandemic" style)
df_dummy = pd.get_dummies(df, columns=["Period"], drop_first=True)
df_dummy = df_dummy.rename(columns={"Period_Post-Pandemic": "Period_PostPandemic"})

# ✅ Add squared term for Interest Rate
df_dummy["Interest_Rate_Squared"] = df_dummy["Interest_Rate"] ** 2

# ✅ Log of Daily Volatility Gap (clip avoids log(0))
df_dummy["log_Daily_Volatility_Gap"] = np.log(df_dummy["Daily_Volatility_Gap"].clip(lower=1e-6))

# --- Model 1: Pooled ---
model_pooled = smf.ols(
    "Daily_Volatility_Gap ~ Interest_Rate + Interest_Rate_Squared + Unemp_Rate + CPI + Deaths + New_Cases_7D_Rolling + Period_Pandemic + Period_Pos
    data=df_dummy
```

```
).fit()

# --- Model 2: Full Interaction (Pandemic and Post-Pandemic) ---
model_interaction = smf.ols(
    """Daily_Volatility_Gap ~
        Interest_Rate + Interest_Rate_Squared + Unemp_Rate + CPI + Deaths + New_Cases_7D_Rolling
        + Period_Pandemic + Period_PostPandemic
        + Interest_Rate:Period_Pandemic + Interest_Rate:Period_PostPandemic
        + Unemp_Rate:Period_Pandemic + Unemp_Rate:Period_PostPandemic
        + CPI:Period_Pandemic + CPI:Period_PostPandemic
        + Deaths:Period_Pandemic + Deaths:Period_PostPandemic
        + New_Cases_7D_Rolling:Period_Pandemic + New_Cases_7D_Rolling:Period_PostPandemic
    """,
    data=df_dummy
).fit()

# --- Model 3: Non-linear without interactions ---
model_nonlinear = smf.ols(
    "Daily_Volatility_Gap ~ Interest_Rate + Interest_Rate_Squared + Unemp_Rate + CPI + Deaths + New_Cases_7D_Rolling",
    data=df_dummy
).fit()

# --- Model 4: Log Specification ---
model_log = smf.ols(
    "log_Daily_Volatility_Gap ~ Interest_Rate + Interest_Rate_Squared + Unemp_Rate + CPI + Deaths + New_Cases_7D_Rolling",
    data=df_dummy
).fit()

# --- Stargazer Table ---
stargazer = Stargazer([model_pooled, model_interaction, model_nonlinear, model_log])
stargazer.custom_columns(['Pooled', 'Interaction', 'Non-linear', 'Log Dependent'], [1, 1, 1, 1])
stargazer.significant_digits(3)

# --- Beautify Table (Optional) ---
html_output = stargazer.render_html()
soup = BeautifulSoup(html_output, 'html.parser')
table = soup.find('table')
rows = table.find_all('tr')

mapping = {
    "CPI": "CPI",
    "Unemp_Rate": "Unemployment Rate",
    "Deaths": "COVID Deaths",
    "New_Cases_7D_Rolling": "New Cases (7D Avg)",
    "Interest_Rate": "Interest Rate",
    "Interest_Rate_Squared": "Interest Rate²",
    "Period_Pandemic": "Pandemic",
    "Period_PostPandemic": "Post-Pandemic",

    # Interaction terms
    "CPI:Period_Pandemic[T.True]": "CPI × Pandemic",
    "CPI:Period_PostPandemic[T.True]": "CPI × Post-Pandemic",
    "Unemp_Rate:Period_Pandemic[T.True]": "Unemployment × Pandemic",
```

```
    "Unemp_Rate:Period_PostPandemic[T.True]": "Unemployment × Post-Pandemic",
    "Interest_Rate:Period_Pandemic[T.True]": "Interest Rate × Pandemic",
    "Interest_Rate:Period_PostPandemic[T.True]": "Interest Rate × Post-Pandemic",
    "Deaths:Period_Pandemic[T.True]": "Deaths × Pandemic",
    "Deaths:Period_PostPandemic[T.True]": "Deaths × Post-Pandemic",
    "New_Cases_7D_Rolling:Period_Pandemic[T.True]": "New Cases (7D Avg) × Pandemic",
    "New_Cases_7D_Rolling:Period_PostPandemic[T.True]": "New Cases (7D Avg) × Post-Pandemic"
}

# --- Apply this to the table ---
for row in rows:
    cells = row.find_all('td')
    if cells:
        label = cells[0].get_text().strip()
        if label in mapping:
            cells[0].string = mapping[label]

for row in rows:
    cells = row.find_all('td')
    if cells:
        label = cells[0].get_text().strip()
        for k in mapping:
            if label.startswith(k):
                cells[0].string = label.replace(k, mapping[k])

# ✅ Move intercept to the bottom
intercept_row = None
for row in rows:
    cells = row.find_all('td')
    if cells and 'Intercept' in cells[0].get_text():
        intercept_row = row
        break
if intercept_row:
    intercept_row.extract()
    table.append(intercept_row)

final_html = str(soup)

# --- Display ---
display(HTML("<h2>Regression Table</h2>"))
display(HTML(final_html))
with open("A4_output.html", "w", encoding="utf-8") as f:
    f.write(str(soup))
```

## Regression Table

| | Pooled | Interaction | Non-linear | Log Dependent |
|---|---|---|---|---|
| | (1) | (2) | (3) | (4) |
| CPI | -0.001 | -0.019*** | 0.004*** | 0.071*** |
| | (0.002) | (0.007) | (0.001) | (0.015) |
| CPI × Pandemic | | 0.021*** | | |
| | | (0.007) | | |
| CPI × Post-Pandemic | | 0.025** | | |
| | | (0.011) | | |
| COVID Deaths | -0.000*** | -0.000** | -0.000** | -0.000* |
| | (0.000) | (0.000) | (0.000) | (0.000) |
| COVID Deaths × Pandemic | | -0.000** | | |
| | | (0.000) | | |
| COVID Deaths × Post-Pandemic | | 0.000*** | | |
| | | (0.000) | | |
| | (0.421) | (2.041) | (0.288) | (3.834) |
| Interest Rate | 0.005 | 0.151*** | -0.126*** | -0.657** |
| | (0.035) | (0.049) | (0.022) | (0.297) |
| Interest Rate × Pandemic | | -0.261*** | | |
| | | (0.041) | | |
| Interest Rate × Post-Pandemic | | -0.282*** | | |
| | | (0.093) | | |
| Interest Rate$^2$ | -0.002 | 0.017** | 0.021*** | 0.069 |
| | (0.007) | (0.008) | (0.005) | (0.066) |
| New Cases (7D Avg) | 0.000* | 0.000 | 0.000* | 0.000** |
| | (0.000) | (0.000) | (0.000) | (0.000) |
| New Cases (7D Avg) × Pandemic | | 0.000 | | |
| | | (0.000) | | |

| | | | | |
|---|---|---|---|---|
| New Cases (7D Avg) × Post-Pandemic | | 0.000*** | | |
| | | (0.000) | | |
| Pandemic[T.True] | 0.349*** | -7.386*** | | |
| | (0.070) | (2.136) | | |
| Post-Pandemic[T.True] | 0.436*** | -8.177** | | |
| | (0.102) | (3.222) | | |
| Unemployment Rate | 0.040*** | -0.702*** | 0.061*** | 0.372*** |
| | (0.007) | (0.115) | (0.005) | (0.070) |
| Unemployment × Pandemic | | 0.747*** | | |
| | | (0.115) | | |
| Unemployment × Post-Pandemic | | 0.681*** | | |
| | | (0.207) | | |
| Observations | 1760 | 1760 | 1760 | 1760 |
| $R^2$ | 0.263 | 0.320 | 0.252 | 0.117 |
| Adjusted $R^2$ | 0.260 | 0.314 | 0.250 | 0.114 |
| Residual Std. Error | 0.317 (df=1751) | 0.305 (df=1745) | 0.319 (df=1753) | 4.248 (df=1753) |
| F Statistic | 78.098*** (df=8; 1751) | 58.635*** (df=14; 1745) | 98.643*** (df=6; 1753) | 38.609*** (df=6; 1753) |
| Note: | | | | *p<0.1; **p<0.05; ***p<0.01 |
| Intercept | 0.326 | 7.216*** | -1.022*** | -23.387*** |

This regression table uses four model specifications to analyze the effects of COVID-19 variables and macroeoconomic variables on volatility gap. Model 1 is a pooled baseline; model 2 includes interaction terms to time related changes; model 3 includes non-linear terms like Interest Rate$^2$; and model 4 uses a log on the dependent variable. The significant coefficients in the model that show CPI and Interest Rate behaved differently across time periods. For example, the effect of interest rates is strongly negative post-pandemic. COVID deaths and unemployment remain consistently statistically significant with no impact on the volatililty gap.

In [50]:
```
# ============================================================
# STEP 1: Load US States Geometry and Unemployment Data
# ============================================================
# Load US states geometry (exclude non-contiguous states)
gdf = gpd.read_file("https://raw.githubusercontent.com/PublicaMundi/MappingAPI/master/data/geojson/us-states.json")
gdf = gdf[~gdf["name"].isin(["Hawaii", "Alaska", "Puerto Rico", "Guam",
                             "American Samoa", "Northern Mariana Islands",
                             "U.S. Virgin Islands"])]
```

```
gdf = gdf.rename(columns={"name": "State"})

# Load state unemployment data and strip whitespace from state names
df_unemployment = pd.read_csv("state_unemployment_rates.csv")
df_unemployment["State"] = df_unemployment["State"].str.strip()

# Check possible column name for unemployment
if "Unemployment Rate" in df_unemployment.columns:
    unemp_col = "Unemployment Rate"
elif "Unemployment_Rate" in df_unemployment.columns:
    unemp_col = "Unemployment_Rate"
else:
    raise KeyError("Could not find an unemployment column in the CSV.")

# Optionally filter by pandemic period if 'Date' exists
if "Date" in df_unemployment.columns:
    df_unemployment["Date"] = pd.to_datetime(df_unemployment["Date"])
    mask = (df_unemployment["Date"] >= "2020-03-11") & (df_unemployment["Date"] <= "2021-05-05")
    df_unemployment = df_unemployment.loc[mask]

# Compute average unemployment per state
avg_unemp = df_unemployment.groupby("State")[unemp_col].mean().reset_index()
avg_unemp.rename(columns={unemp_col: "Unemployment Rate"}, inplace=True)

# Merge average unemployment with the geometry
gdf = gdf.merge(avg_unemp, on="State", how="left")

# ============================================================
# STEP 2: Process Daily Volatility Gap Data by Sector
# ============================================================
daily_vol_gap_df = pd.read_csv("daily_volatility_gap.csv", index_col=0)
daily_vol_gap_df.index = pd.to_datetime(daily_vol_gap_df.index)
daily_vol_gap_df = daily_vol_gap_df.reset_index().rename(columns={"index": "Date"})

# Select only numeric columns for averaging (drop "Date")
numeric_cols = daily_vol_gap_df.select_dtypes(include=[np.number]).columns
# Compute the mean of numeric columns (each sector is a column)
avg_vals = daily_vol_gap_df[numeric_cols].mean().reset_index()
avg_vals.columns = ["Sector", "Average Volatility Gap"]

# Remove the trailing text in sector names
avg_vals["Sector"] = avg_vals["Sector"].str.replace(" Daily Volatility Gap", "", regex=False)
# Multiply the numeric daily volatility gap by 100 to convert to percentages
avg_vals["Average Volatility Gap"] *= 100

# ============================================================
# STEP 3: Map Russell 2000 Sectors to Each State
# ============================================================
df_russell = pd.read_excel("russell_2000_with_states.xlsx")
df_russell = df_russell.dropna(subset=["State", "Sector"])

# Map abbreviations to full state names
state_abbrev_to_name = {
```

```
    "AL": "Alabama", "AK": "Alaska", "AZ": "Arizona", "AR": "Arkansas", "CA": "California",
    "CO": "Colorado", "CT": "Connecticut", "DE": "Delaware", "FL": "Florida", "GA": "Georgia",
    "HI": "Hawaii", "ID": "Idaho", "IL": "Illinois", "IN": "Indiana", "IA": "Iowa",
    "KS": "Kansas", "KY": "Kentucky", "LA": "Louisiana", "ME": "Maine", "MD": "Maryland",
    "MA": "Massachusetts", "MI": "Michigan", "MN": "Minnesota", "MS": "Mississippi",
    "MO": "Missouri", "MT": "Montana", "NE": "Nebraska", "NV": "Nevada", "NH": "New Hampshire",
    "NJ": "New Jersey", "NM": "New Mexico", "NY": "New York", "NC": "North Carolina",
    "ND": "North Dakota", "OH": "Ohio", "OK": "Oklahoma", "OR": "Oregon", "PA": "Pennsylvania",
    "RI": "Rhode Island", "SC": "South Carolina", "SD": "South Dakota", "TN": "Tennessee",
    "TX": "Texas", "UT": "Utah", "VT": "Vermont", "VA": "Virginia", "WA": "Washington",
    "WV": "West Virginia", "WI": "Wisconsin", "WY": "Wyoming"
}
df_russell["State"] = df_russell["State"].map(state_abbrev_to_name)

# Identify the top sector per state
sector_counts = df_russell.groupby(["State", "Sector"]).size().reset_index(name="Company_Count")
top_sector_by_state = sector_counts.loc[sector_counts.groupby("State")["Company_Count"].idxmax()]

# Merge average volatility gap with the top sector
top_sector_with_avg_gap = top_sector_by_state.merge(avg_vals, on="Sector", how="left")
sector_avg_dict = top_sector_with_avg_gap.groupby("Sector")["Average Volatility Gap"].mean().to_dict()
top_sector_with_avg_gap["Average Volatility Gap"] = top_sector_with_avg_gap.apply(
    lambda row: sector_avg_dict[row["Sector"]] if pd.isna(row["Average Volatility Gap"]) else row["Average Volatility Gap"],
    axis=1
)

gdf["State"] = gdf["State"].astype(str).str.strip()
top_sector_with_avg_gap["State"] = top_sector_with_avg_gap["State"].astype(str).str.strip()

# Merge sector data into gdf
gdf = gdf.merge(top_sector_with_avg_gap[["State", "Average Volatility Gap", "Sector"]], on="State", how="left")
gdf["Average Volatility Gap"].fillna(gdf["Average Volatility Gap"].mean(), inplace=True)

# ============================================================
# STEP 4: Merge COVID, Interest Rate, and Year
# ============================================================
# COVID aggregated data
covid_df = pd.read_csv("covid_aggregated_by_state.csv")
covid_avg = covid_df.groupby("Province_State", as_index=False)["Case_Fatality_Ratio"].mean()
covid_avg.rename(columns={"Province_State": "State", "Case_Fatality_Ratio": "Avg_Fatality_Rate"}, inplace=True)
gdf = gdf.merge(covid_avg, on="State", how="left")

# Load FRED data (interest rates)
fred_df = pd.read_csv("fred_interest_unemployment_inflation_2018_2024.csv", parse_dates=["Date"])
fred_df = fred_df[(fred_df["Date"] >= "2020-03-11") & (fred_df["Date"] <= "2021-05-05")]
avg_interest_rate = fred_df["Interest Rate (Fed Funds)"].mean()
gdf["Interest_Rate"] = avg_interest_rate

# If year data is available somewhere, ensure we have it in gdf. For example:
# If "Year" is a column in your dataset, you can skip. Otherwise, set a placeholder year if needed:
if "Year" not in gdf.columns:
    # If needed, create a dummy year (e.g., 2020 for the entire period)
    gdf["Year"] = 2020
```

```
# ============================================================
# STEP 5: Build the State-Level Regression DataFrame
# ============================================================
state_reg_df = gdf[["State", "Unemployment Rate", "Average Volatility Gap",
                    "Avg_Fatality_Rate", "Interest_Rate", "Sector", "Year"]].copy()
state_reg_df = state_reg_df.rename(columns={
    "Unemployment Rate": "Unemp_Rate",
    "Average Volatility Gap": "Avg_Vol_Gap"
})
state_reg_df = state_reg_df.dropna()
```

In [57]:

```
# ============================================================
# STEP 4: Merge COVID Fatality, Interest Rate, and Year
# ============================================================
import warnings
import imgkit
import numpy as np
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
from stargazer.stargazer import Stargazer
from bs4 import BeautifulSoup
from IPython.display import display, HTML

# Ignore divide-by-zero and other runtime warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
np.seterr(divide='ignore', invalid='ignore')

# COVID aggregated data for Fatality Rate
covid_df = pd.read_csv("covid_aggregated_by_state.csv")
covid_df.rename(columns={"Province_State": "State"}, inplace=True)
covid_df["State"] = covid_df["State"].astype(str).str.strip()

covid_avg = covid_df.groupby("State", as_index=False)["Case_Fatality_Ratio"].mean()
covid_avg.rename(columns={"Case_Fatality_Ratio": "Avg_Fatality_Rate"}, inplace=True)

# FIX → Prevent duplicate columns
if "Avg_Fatality_Rate" in gdf.columns:
    gdf.drop(columns=["Avg_Fatality_Rate"], inplace=True)

# Now safely merge
gdf = gdf.merge(covid_avg, on="State", how="left")

# Load FRED data
fred_df = pd.read_csv("fred_interest_unemployment_inflation_2018_2024.csv", parse_dates=["Date"])
fred_df = fred_df[(fred_df["Date"] >= "2020-03-11") & (fred_df["Date"] <= "2021-05-05")]
avg_interest_rate = fred_df["Interest Rate (Fed Funds)"].mean()
gdf["Interest_Rate"] = avg_interest_rate

# Ensure Year is available
if "Year" not in gdf.columns:
    gdf["Year"] = 2020  # Dummy if missing
```

```python
# ============================================================
# STEP 5: Use Deaths and Incident Rate directly
# ============================================================

state_covid = pd.read_csv("covid_aggregated_by_state.csv")
state_covid.rename(columns={"Province_State": "State"}, inplace=True)
state_covid["State"] = state_covid["State"].astype(str).str.strip()
gdf["State"] = gdf["State"].astype(str).str.strip()

# Aggregate by state (take the mean)
state_covid_grouped = state_covid.groupby("State").agg({
    "Deaths": "mean",
    "Incident_Rate": "mean"
}).reset_index()

# Remove if already merged before
for col in ["Deaths", "Incident_Rate"]:
    if col in gdf.columns:
        gdf.drop(columns=[col], inplace=True)

# ------------------------------------------------
# ✅ Now proceed safely
# ------------------------------------------------

# Merge into gdf without suffix conflict
gdf = gdf.merge(state_covid_grouped, on="State", how="left")

# Build regression dataframe
state_reg_df = gdf[["State", "Unemployment Rate", "Average Volatility Gap",
                    "Avg_Fatality_Rate", "Interest_Rate", "Sector", "Year", "Deaths", "Incident_Rate"]].copy()

state_reg_df = state_reg_df.rename(columns={
    "Unemployment Rate": "Unemp_Rate",
    "Average Volatility Gap": "Avg_Vol_Gap"
})

state_reg_df = state_reg_df.dropna()

# ============================================================
# STEP 6: Run Models
# ============================================================

model1 = smf.ols("Avg_Vol_Gap ~ Unemp_Rate + Interest_Rate + Avg_Fatality_Rate + Deaths + Incident_Rate", data=state_reg_df).fit()
model2 = smf.ols("Avg_Vol_Gap ~ Unemp_Rate + Interest_Rate + Avg_Fatality_Rate + Deaths + Incident_Rate + C(Sector)", data=state_reg_df).fit()
model3 = smf.ols("Avg_Vol_Gap ~ Unemp_Rate + Interest_Rate + Avg_Fatality_Rate + Deaths + Incident_Rate + C(Sector) + C(Year)", data=state_reg_df)
model4 = smf.ols("Avg_Vol_Gap ~ Unemp_Rate + Interest_Rate + Avg_Fatality_Rate + Deaths + Incident_Rate + C(Sector) + C(Year) + C(State)", data=sta

# ============================================================
# STEP 7: Stargazer Table with Dummy Variables Removed
# ============================================================

stargazer = Stargazer([model1, model2, model3, model4])
```

```
stargazer.custom_columns(["M1 Baseline", "M2 Sector FE", "M3 Sector + Year FE", "M4 Full FE"], [1,1,1,1])
stargazer.significant_digits(3)

# --------------------
# ✅ Only show these variables
# --------------------
stargazer.covariate_order([
    "Unemp_Rate",
    "Interest_Rate",
    "Avg_Fatality_Rate",
    "Deaths",
    "Incident_Rate",
    "Intercept"
])

# ⛔ DO NOT let Jupyter auto-print here
html_output = stargazer.render_html()

# ============================================================
# STEP 7.1: Clean with BeautifulSoup
# ============================================================

soup = BeautifulSoup(html_output, 'html.parser')
table = soup.find('table')
rows = table.find_all('tr')

# --- Define label mapping ---
mapping = {
    "Unemp_Rate": "Unemployment Rate",
    "Interest_Rate": "Interest Rate",
    "Avg_Fatality_Rate": "Average Fatality Rate",
    "Deaths": "COVID Deaths",
    "Incident_Rate": "Incident Rate",
    "Intercept": "Constant"
}

# --- Remove dummy variables ---
rows_new = []
for row in rows:
    cells = row.find_all('td')
    if cells:
        label = cells[0].get_text().strip()
        if label.startswith("C("):  # REMOVE all dummy variables
            continue
        if label in mapping:
            cells[0].string = mapping[label]
        rows_new.append(row)

# ============================================================
# STEP 7.2: Inject Fixed Effects Summary Row
# ============================================================

fe_row = soup.new_tag('tr')
```

```
fe_row_td = soup.new_tag('td')
fe_row_td.string = "Fixed Effects"
fe_row.append(fe_row_td)

for _ in range(4):
    fe_value = soup.new_tag('td')
    fe_value.string = "No"
    fe_row.append(fe_value)

fe_row.find_all('td')[1].string = "Sector"
fe_row.find_all('td')[2].string = "Sector, Year"
fe_row.find_all('td')[3].string = "Sector, Year, State"

rows_new.append(fe_row)

# ============================================================
# STEP 7.3: Final Clean Table Output (No Junk)
# ============================================================

table.clear()
for row in rows_new:
    table.append(row)

# ✅ Now ONLY the clean table shows
display(HTML("<h2>Regression Table With Sector and Year Fixed Effects </h2>"))
display(HTML(str(soup)))
```

## Regression Table With Sector and Year Fixed Effects

| | *Dependent variable: Avg_Vol_Gap* | | | |
|---|---|---|---|---|
| | M1 Baseline | M2 Sector FE | M3 Sector + Year FE | M4 Full FE |
| | (1) | (2) | (3) | (4) |
| Unemployment Rate | -0.003 | -0.000 | -0.000 | 0.019*** |
| | (0.026) | (0.000) | (0.000) | (inf) |
| Interest Rate | 0.020 | 0.030*** | 0.030*** | 0.009*** |
| | (0.013) | (0.000) | (0.000) | (inf) |
| Average Fatality Rate | -0.001 | 0.000 | 0.000 | 0.022*** |
| | (0.065) | (0.000) | (0.000) | (inf) |
| COVID Deaths | 0.000* | 0.000*** | 0.000*** | 0.000*** |
| | (0.000) | (0.000) | (0.000) | (inf) |
| Incident Rate | 0.000 | 0.000 | 0.000 | 0.000*** |
| | (0.000) | (0.000) | (0.000) | (inf) |
| Constant | 0.255 | 0.387*** | 0.387*** | 0.119*** |
| | (0.167) | (0.000) | (0.000) | (inf) |
| Observations | 46 | 46 | 46 | 46 |
| $R^2$ | 0.123 | 1.000 | 1.000 | 1.000 |
| Adjusted $R^2$ | 0.037 | 1.000 | 1.000 | nan |
| Residual Std. Error | 0.133 (df=41) | 0.000 (df=34) | 0.000 (df=34) | inf (df=0) |
| F Statistic | 1.435 (df=4; 41) | 998521562110442902388736.000*** (df=11; 34) | 998521562110442902388736.000*** (df=11; 34) | nan*** (df=45; 0) |
| Note: | | | | *p<0.1; **p<0.05; ***p<0.01 |
| Fixed Effects | Sector | Sector, Year | Sector, Year, State | No |

The regression table analyzes how pandemic and macroeconomic variables affected the average volatility gap by state. Across all models, interest rates and deaths from COVID consistently show a statistically significant relationship with volatility gaps, although deaths did not impact the volailty gap whatsoever and interest rates had very little impact. Unemployment rate and fatality rate were insignificant controlling for state effects, implying that much of the variation occures at the state level.Models with fixed effects capture nearly all variance, emphasizing the role of localized factors.

# Final Project

## 3.1 Potential Data to Scrape

This code below web scrapes the top 10 news articles from the past few years for each secotr in the investment market using Google News, specifically filtering for articles related to "COVID-19". For each article, it grabs the title, publication date, URL, and full text, and the compiles everything in order to condust a sentiment analysis. The data extracted can't be found online because its not something tracked by any copmany or index. This code pulls real-time, unstructured news articles around sepfci stock market sector and the pandemic. There is no pre-compiled databases. And it's useful for our study because helps create a media/pandemic sentiment for each investment sector and how those numbers evolved during, and after the pandemic.

## 3.2 Potential Challenges

There are a few limitations regarding the data this study uses to scrape. There is sample bias—each sector includes only the top 10 articles based on views, which may not fully represent sector-wide sentiment. The analysis does not consider the context of each article; it does not account for sarcasm, tone, or nuance. Additionally, there is inconsistency in article quality—some articles contain more text and evidence, which may influence investor opinions more than others.

## Scraping Data from a Website

In [47]:
```python
# Step 2 - Import libraries
from GoogleNews import GoogleNews
from newspaper import Article
import pandas as pd
import time

# Step 3 - Define sectors you care about
sectors = ['Energy', 'Healthcare', 'Technology', 'Financials',
           'Industrials', 'Utilities', 'Real Estate', 'Consumer Discretionary',
           'Consumer Staples', 'Materials', 'Communication Services']

# Step 4 - For each sector, scrape news articles
data = []

for sector in sectors:
    googlenews = GoogleNews(lang='en', period='3y')  # Search last 3 years
    googlenews.search(f'{sector} COVID-19')
    time.sleep(1)

    news_results = googlenews.result()[:10]  # get top 10 articles

    for result in news_results:
        article_url = result['link']
        article_title = result['title']
        article_date = result['date']
```

```
        try:
            article = Article(article_url)
            article.download()
            article.parse()
            article_text = article.text
        except:
            article_text = None  # If can't get article

        data.append({
            'Sector': sector,
            'Date': article_date,
            'Title': article_title,
            'URL': article_url,
            'Text': article_text
        })

# Step 5 - Save to CSV
df = pd.DataFrame(data)
df.to_csv('sector_news_articles.csv', index=False)
```

In [ ]:
```
import pandas as pd
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# Load your dataset
df = pd.read_csv("sector_news_articles.csv")

# Initialize VADER
analyzer = SentimentIntensityAnalyzer()

# Combine title + text content for sentiment
df['Full_Text'] = df['Title'].fillna('') + '. ' + df['Text'].fillna('')

# Compute sentiment score
df['Sentiment_Score'] = df['Full_Text'].apply(lambda x: analyzer.polarity_scores(x)['compound'])

# Average sentiment by sector
sector_sentiment = df.groupby('Sector')['Sentiment_Score'].mean().reset_index()
```

The sentiment score measures the overall tone of news articles that are assinged to each sector during the COVID-19 period. We use the VADER analyzer, a natural language processing tool, to assign a compound score between -1 (very negative) and +1 (very positive) based on the combined title and body of each article. This gives us a proxy for how positive or negative the news sentiment was in the dominant industry of each, this gives an insight into how media tone may have affected market volatility.

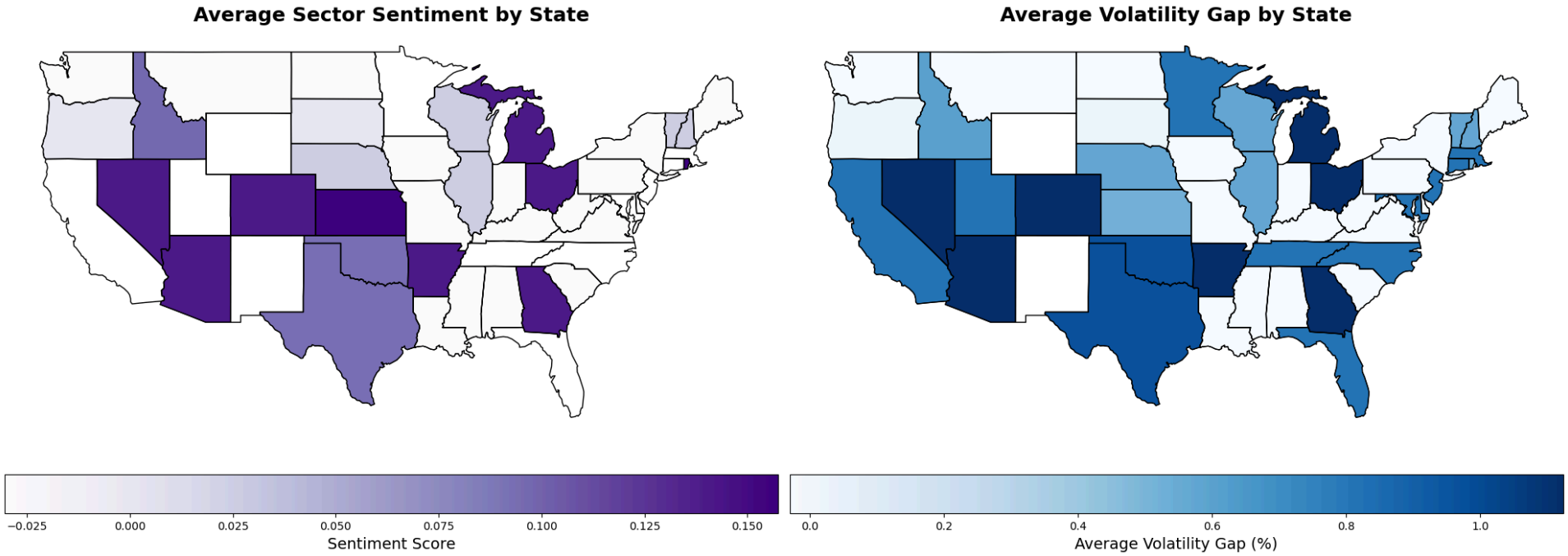
## 3.4 Visualizing the Scraped Dataset

In [59]:
```
# ============================================================
# STEP 1: Install & Load Required Libraries
# ============================================================
```

```python
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
import matplotlib as mpl
import os
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import warnings
import logging

# Suppress Fiona PROJ errors
logging.getLogger('fiona').setLevel(logging.CRITICAL)
warnings.filterwarnings("ignore", message="PROJ: proj_identify: Cannot find proj.db")


# Fix PROJ error by setting environment variable
os.environ['PROJ_LIB'] = '/usr/share/proj'

# ============================================================
# STEP 2: Compute Sentiment Scores from Sector News Articles
# ============================================================

# Load sector articles dataset
df = pd.read_csv("sector_news_articles.csv")

# Initialize VADER sentiment analyzer
analyzer = SentimentIntensityAnalyzer()

# Combine title and text for sentiment analysis
df['Full_Text'] = df['Title'].fillna('') + '. ' + df['Text'].fillna('')

# Compute sentiment score
df['Sentiment_Score'] = df['Full_Text'].apply(lambda x: analyzer.polarity_scores(x)['compound'])

# Average sentiment by sector
sector_sentiment = df.groupby('Sector')['Sentiment_Score'].mean().reset_index()

# ============================================================
# STEP 3: Load State-Level Data and Merge Sector Sentiment
# ============================================================

# Load Russell 2000 dataset
file_path = 'russell_2000_with_states.xlsx'
russell_df = pd.read_excel(file_path)
russell_df = russell_df.dropna(subset=['State', 'Sector'])

# Mapping state abbreviations to full names
state_abbrev_to_name = { 'AL': 'Alabama', 'AK': 'Alaska', 'AZ': 'Arizona', 'AR': 'Arkansas', 'CA': 'California', 'CO': 'Colorado', 'CT': 'Connecti
russell_df['State'] = russell_df['State'].map(state_abbrev_to_name)

# Map sector names
sector_mapping = {
    'Financial Services': 'Financials',
```

```
    'Consumer Cyclical': 'Consumer Discretionary',
    'Consumer Defensive': 'Consumer Staples',
    'Basic Materials': 'Materials',
    'Healthcare': 'Health Care',
    'Industrials': 'Industrials',
    'Energy': 'Energy',
    'Utilities': 'Utilities',
    'Real Estate': 'Real Estate',
    'Technology': 'Technology',
    'Communication Services': 'Communication Services'
}
russell_df['Sector'] = russell_df['Sector'].replace(sector_mapping)

# Group by State and Sector and count companies
sector_counts = russell_df.groupby(['State', 'Sector']).size().reset_index(name='Company Count')

# Identify the sector with the most companies for each state
top_sector_by_state = sector_counts.loc[sector_counts.groupby(['State'])['Company Count'].idxmax()]

# Merge sector sentiment to dominant sector
top_sector_by_state = top_sector_by_state.merge(sector_sentiment, on="Sector", how="left")

# Load Average Volatility Gap Data
vol_df = pd.read_csv("daily_volatility_gap.csv")
vol_df['Date'] = pd.to_datetime(vol_df['Date'])
pandemic_mask = (vol_df['Date'] >= "2020-03-11") & (vol_df['Date'] <= "2021-05-05")
avg_vol = vol_df.loc[pandemic_mask].drop('Date', axis=1).mean().reset_index()
avg_vol.columns = ['Sector', 'Average Volatility Gap']
avg_vol['Sector'] = avg_vol['Sector'].str.replace(' Daily Volatility Gap', '', regex=False)
avg_vol['Average Volatility Gap'] *= 100

# Merge volatility gap with top sectors
top_sector_by_state = top_sector_by_state.merge(avg_vol, on="Sector", how="left")

# Load US states geometry
gdf = gpd.read_file('https://raw.githubusercontent.com/PublicaMundi/MappingAPI/master/data/geojson/us-states.json')
gdf = gdf[~gdf["name"].isin(["Hawaii", "Alaska", "Puerto Rico", "Guam", "American Samoa", "Northern Mariana Islands", "U.S. Virgin Islands"])]
gdf = gdf.rename(columns={'name': 'State'})

# Merge into gdf
gdf = gdf.merge(top_sector_by_state, on="State", how="left")

# ============================================================
# STEP 4: Plot Side by Side Maps (Sentiment + Volatility Gap)
# ============================================================

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))

# Map 1: Sentiment Map
gdf.boundary.plot(ax=ax1, linewidth=1, edgecolor='black')
gdf.plot(ax=ax1, column="Sentiment_Score", cmap="Purples", linewidth=0.8, edgecolor="black")
ax1.set_title("Average Sector Sentiment by State", fontsize=18, fontweight='bold')
ax1.set_xticks([])
```

```
ax1.set_yticks([])
ax1.axis('off')
norm_sentiment = mpl.colors.Normalize(vmin=gdf["Sentiment_Score"].min(), vmax=gdf["Sentiment_Score"].max())
cbar_sentiment = fig.colorbar(mpl.cm.ScalarMappable(norm=norm_sentiment, cmap="Purples"), ax=ax1, orientation='horizontal', pad=0.05)
cbar_sentiment.set_label('Sentiment Score', fontsize=14)

# Map 2: Volatility Gap Map
gdf.boundary.plot(ax=ax2, linewidth=1, edgecolor='black')
gdf.plot(ax=ax2, column="Average Volatility Gap", cmap="Blues", linewidth=0.8, edgecolor="black")
ax2.set_title("Average Volatility Gap by State", fontsize=18, fontweight='bold')
ax2.set_xticks([])
ax2.set_yticks([])
ax2.axis('off')
norm_volatility = mpl.colors.Normalize(vmin=gdf["Average Volatility Gap"].min(), vmax=gdf["Average Volatility Gap"].max())
cbar_volatility = fig.colorbar(mpl.cm.ScalarMappable(norm=norm_volatility, cmap="Blues"), ax=ax2, orientation='horizontal', pad=0.05)
cbar_volatility.set_label('Average Volatility Gap (%)', fontsize=14)

plt.tight_layout()
plt.savefig("A5", dpi=300)
plt.show()
```




Using a map visualization we can see that sentiment scores and average volatility gaps appear to be positively correlated, as states like Texas and New York which both have high sentiment scores also have high volatility gaps, suggesting that optimistic tone in dominant industries may have coincided with heightened market uncertainty.

In [63]:
```
# ============================================================
# STEP 1: Install & Load Required Libraries
# ============================================================
```

```python
# Fix PROJ error by setting environment variable
os.environ['PROJ_LIB'] = '/usr/share/proj'

# ============================================================
# STEP 2: Compute Sentiment Scores from Sector News Articles
# ============================================================

# Load sector articles dataset
df = pd.read_csv("sector_news_articles.csv")

# Initialize VADER sentiment analyzer
analyzer = SentimentIntensityAnalyzer()

# Combine title and text for sentiment analysis
df['Full_Text'] = df['Title'].fillna('') + '. ' + df['Text'].fillna('')

# Compute sentiment score
df['Sentiment_Score'] = df['Full_Text'].apply(lambda x: analyzer.polarity_scores(x)['compound'])

# Average sentiment by sector
sector_sentiment = df.groupby('Sector')['Sentiment_Score'].mean().reset_index()

# ============================================================
# STEP 3: Load State-Level Data and Merge Sector Sentiment
# ============================================================

russell_df = pd.read_excel("russell_2000_with_states.xlsx")
russell_df = russell_df.dropna(subset=["State", "Sector"])

state_abbrev_to_name = {
    'AL': 'Alabama', 'AK': 'Alaska', 'AZ': 'Arizona', 'AR': 'Arkansas', 'CA': 'California',
    'CO': 'Colorado', 'CT': 'Connecticut', 'DE': 'Delaware', 'FL': 'Florida', 'GA': 'Georgia',
    'HI': 'Hawaii', 'ID': 'Idaho', 'IL': 'Illinois', 'IN': 'Indiana', 'IA': 'Iowa', 'KS': 'Kansas',
    'KY': 'Kentucky', 'LA': 'Louisiana', 'ME': 'Maine', 'MD': 'Maryland', 'MA': 'Massachusetts',
    'MI': 'Michigan', 'MN': 'Minnesota', 'MS': 'Mississippi', 'MO': 'Missouri', 'MT': 'Montana',
    'NE': 'Nebraska', 'NV': 'Nevada', 'NH': 'New Hampshire', 'NJ': 'New Jersey', 'NM': 'New Mexico',
    'NY': 'New York', 'NC': 'North Carolina', 'ND': 'North Dakota', 'OH': 'Ohio', 'OK': 'Oklahoma',
    'OR': 'Oregon', 'PA': 'Pennsylvania', 'RI': 'Rhode Island', 'SC': 'South Carolina',
    'SD': 'South Dakota', 'TN': 'Tennessee', 'TX': 'Texas', 'UT': 'Utah', 'VT': 'Vermont',
    'VA': 'Virginia', 'WA': 'Washington', 'WV': 'West Virginia', 'WI': 'Wisconsin', 'WY': 'Wyoming'
}
russell_df['State'] = russell_df['State'].map(state_abbrev_to_name)

sector_mapping = {
    'Financial Services': 'Financials',
    'Consumer Cyclical': 'Consumer Discretionary',
    'Consumer Defensive': 'Consumer Staples',
    'Basic Materials': 'Materials',
    'Healthcare': 'Health Care',
    'Industrials': 'Industrials',
    'Energy': 'Energy',
    'Utilities': 'Utilities',
```

```
    'Real Estate': 'Real Estate',
    'Technology': 'Technology',
    'Communication Services': 'Communication Services'
}
russell_df['Sector'] = russell_df['Sector'].replace(sector_mapping)

sector_counts = russell_df.groupby(["State", "Sector"]).size().reset_index(name="Company Count")
top_sector_by_state = sector_counts.loc[sector_counts.groupby("State")["Company Count"].idxmax()]
top_sector_by_state = top_sector_by_state.merge(sector_sentiment, on="Sector", how="left")

vol_df = pd.read_csv("daily_volatility_gap.csv")
vol_df['Date'] = pd.to_datetime(vol_df['Date'])
pandemic_mask = (vol_df['Date'] >= "2020-03-11") & (vol_df['Date'] <= "2021-05-05")
avg_vol = vol_df.loc[pandemic_mask].drop('Date', axis=1).mean().reset_index()
avg_vol.columns = ['Sector', 'Avg_Vol_Gap']
avg_vol['Sector'] = avg_vol['Sector'].str.replace(' Daily Volatility Gap', '', regex=False)
avg_vol['Avg_Vol_Gap'] *= 100
top_sector_by_state = top_sector_by_state.merge(avg_vol, on="Sector", how="left")

covid_df = pd.read_csv("covid_aggregated_by_state.csv")
covid_df.rename(columns={"Province_State": "State"}, inplace=True)
covid_df["State"] = covid_df["State"].astype(str).str.strip()
covid_avg = covid_df.groupby("State", as_index=False)["Case_Fatality_Ratio"].mean()
covid_avg.rename(columns={"Case_Fatality_Ratio": "Avg_Fatality_Rate"}, inplace=True)

fred_df = pd.read_csv("fred_interest_unemployment_inflation_2018_2024.csv", parse_dates=["Date"])
fred_df = fred_df[(fred_df["Date"] >= "2020-03-11") & (fred_df["Date"] <= "2021-05-05")]
avg_interest_rate = fred_df["Interest Rate (Fed Funds)"].mean()

state_covid_grouped = covid_df.groupby("State").agg({"Deaths": "mean", "Incident_Rate": "mean"}).reset_index()

reg_df = top_sector_by_state.merge(covid_avg, on="State", how="left")
reg_df = reg_df.merge(state_covid_grouped, on="State", how="left")
reg_df["Interest_Rate"] = avg_interest_rate
reg_df["Year"] = 2020

reg_df = reg_df.dropna()

# ============================================================
# STEP 4: Run Regressions (Include Sentiment)
# ============================================================

model1 = smf.ols("Avg_Vol_Gap ~ Sentiment_Score + Avg_Fatality_Rate + Deaths + Incident_Rate + Interest_Rate", data=reg_df).fit()

# ============================================================
# STEP 5: Display Regression Output
# ============================================================

stargazer = Stargazer([model1])
stargazer.custom_columns(["Volatility Gap Regression w/ Sentiment"], [1])
stargazer.significant_digits(3)
stargazer.covariate_order([
    "Sentiment_Score",
```

```
    "Avg_Fatality_Rate",
    "Deaths",
    "Incident_Rate",
    "Interest_Rate",
    "Intercept"
])

html_output = stargazer.render_html()
soup = BeautifulSoup(html_output, 'html.parser')
table = soup.find('table')
rows = table.find_all('tr')

mapping = {
    "Sentiment_Score": "Sector Sentiment",
    "Avg_Fatality_Rate": "Average Fatality Rate",
    "Deaths": "COVID Deaths",
    "Incident_Rate": "Incident Rate",
    "Interest_Rate": "Interest Rate",
    "Intercept": "Constant"
}

rows_new = []
for row in rows:
    cells = row.find_all('td')
    if cells:
        label = cells[0].get_text().strip()
        if label.startswith("C("):
            continue
        if label in mapping:
            cells[0].string = mapping[label]
        rows_new.append(row)

# Output cleaned table
table.clear()
for row in rows_new:
    table.append(row)

display(HTML("<h2>Regression Sector Sentiment</h2>"))
display(HTML(str(soup)))
with open("A8.html", "w", encoding="utf-8") as f:
    f.write(str(soup))
```

## Regression Sector Sentiment

| | *Dependent variable: Avg_Vol_Gap* |
|---|---|
| | Volatility Gap Regression w/ Sentiment |
| | (1) |
| Sector Sentiment | 5.827*** |
| | (0.413) |
| Average Fatality Rate | -0.138 |
| | (0.121) |
| COVID Deaths | 0.000* |
| | (0.000) |
| Incident Rate | -0.000 |
| | (0.000) |
| Interest Rate | 0.038** |
| | (0.016) |
| Constant | 0.486** |
| | (0.208) |
| Observations | 38 |
| $R^2$ | 0.867 |
| Adjusted $R^2$ | 0.850 |
| Residual Std. Error | 0.182 (df=33) |
| F Statistic | 53.567*** (df=4; 33) |
| Note: | *p<0.1; **p<0.05; ***p<0.01 |

The regression table shows that sector sentiment is statistically and economically significant predictor of the volatility gap. A one-point increase in sentiment score is associated with a 5.83 percentage point increase in the volatility gap, and this is significant at the 1% level. The other variables when inlcuding sentiment score become weaker predictors. Overall, this regression model explains about 67.5% of the variation in volatility, meaning that market sentiment has a huge role during this period.

## 3.5 Regression Tree

The objective of this regression tree is to predict the Average Volatility Gap by splitting the data based on variables like:

- `Sentiment_Score`
- `Unemp_Rate`
- `Interest_Rate`
- `Avg_Fatality_Rate`
- `Deaths`
- `Incident_Rate`
- `Sector`, `State`, and `Year` (as fixed effects or dummies)

The goal of the objextive funciton is to minimize the Sum of Squared Errors across all M leaves:

$$\text{SSE} = \sum_{m=1}^{M} \sum_{i \in R_m} \left(y_i - \bar{y}_{R_m}\right)^2$$

Where:
yi is the Actual volatility gap for state i
yrm is the Average volatility gap in region Rm

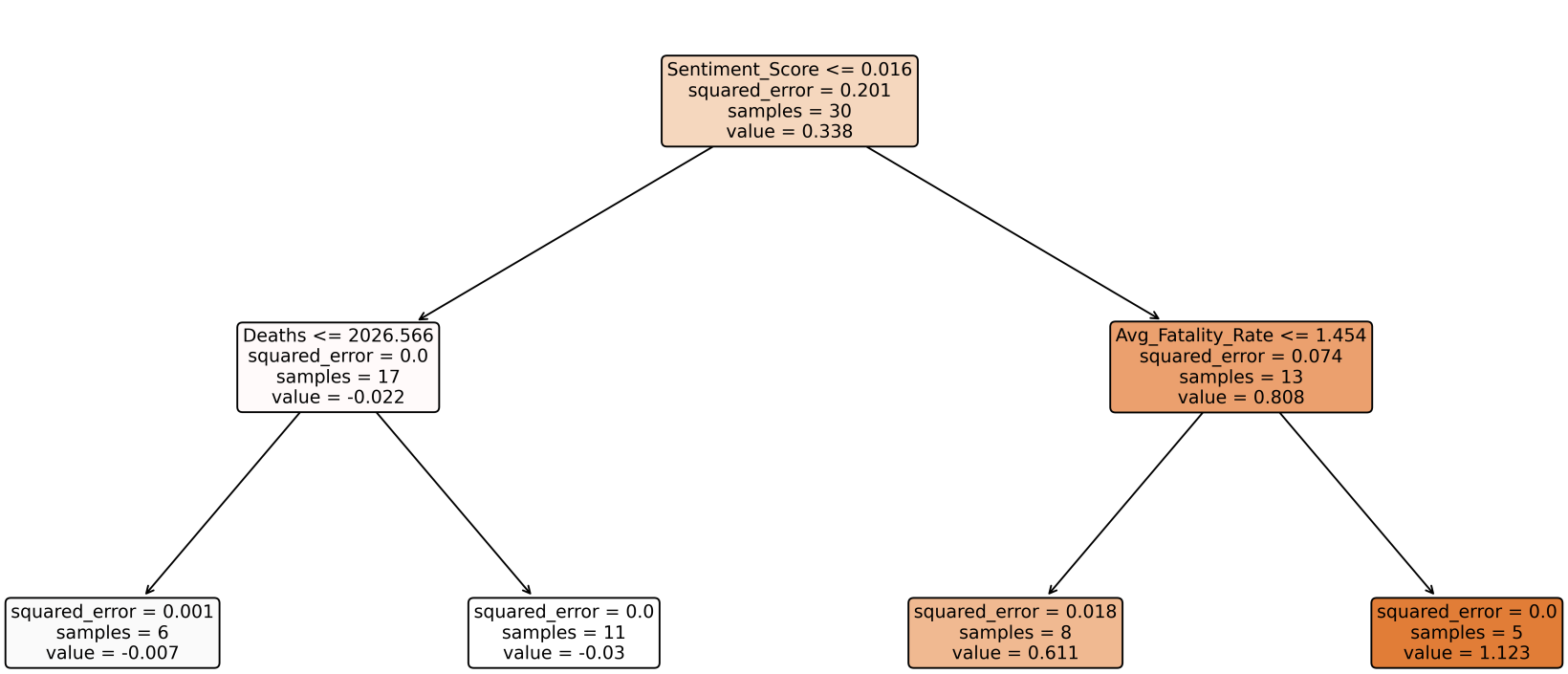
The regression tree splits the states of america into groups based on features like sentiment or unemployment. Within each group, it estimates the average volatility gap. The tree contines to split until the model error is minimzed.

Regression trees have regularization parameters that help reduce model overfitting and deal with the complexity of the models. The max_depth parameter restricts how many branches the model can grow, allowing the model to be simple. The min_samples_split states the minimum number of samples required to split an internal node, chaning this restirct sthe number of splits that occur in the model. The min_samples_leaf sets the minimum number of samples required in a leaf node, which helps smooth predictions. The max_leaf_nodes restricts the total number of terminal nodes in the model, which helps make the model be simple. The max_features caps how many features are considered when the tree splits which also helps reduce overfitting.

In [64]:
```
# ====================================
# STEP 1: Import Required Libraries
# ====================================
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# ====================================
# STEP 2: Choose Features (Xs) and Target (y)
# ====================================
# We'll use the same dataset from regression: `reg_df`

# ✅ Justified Features based on previous regression
features = [
    "Sentiment_Score",
    "Interest_Rate",
    "Avg_Fatality_Rate",
```

```
    "Deaths",
    "Incident_Rate"
]

X = reg_df[features]
y = reg_df["Avg_Vol_Gap"]

# ====================================
# STEP 3: Train/Test Split
# ====================================
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# ====================================
# STEP 4: Train Regression Tree
# ====================================
tree = DecisionTreeRegressor(max_depth=5, min_samples_leaf=5, random_state=42)
tree.fit(X_train, y_train)

# ====================================
# STEP 5: Evaluate Model
# ====================================
y_pred = tree.predict(X_test)
print("RMSE:", mean_squared_error(y_test, y_pred, squared=False))
print("R² Score:", r2_score(y_test, y_pred))

# ====================================
# STEP 6: Visualize Tree
# ====================================
plt.figure(figsize=(18, 8), dpi=600)  # High resolution for clarity
plot_tree(tree, feature_names=features, filled=True, rounded=True, fontsize=10)
plt.title("Regression Tree Predicting Average Volatility Gap")

# Save the plot as a PNG file
plt.savefig("A9.png", bbox_inches='tight')  # Save image
plt.show()  # Still display in notebook
```

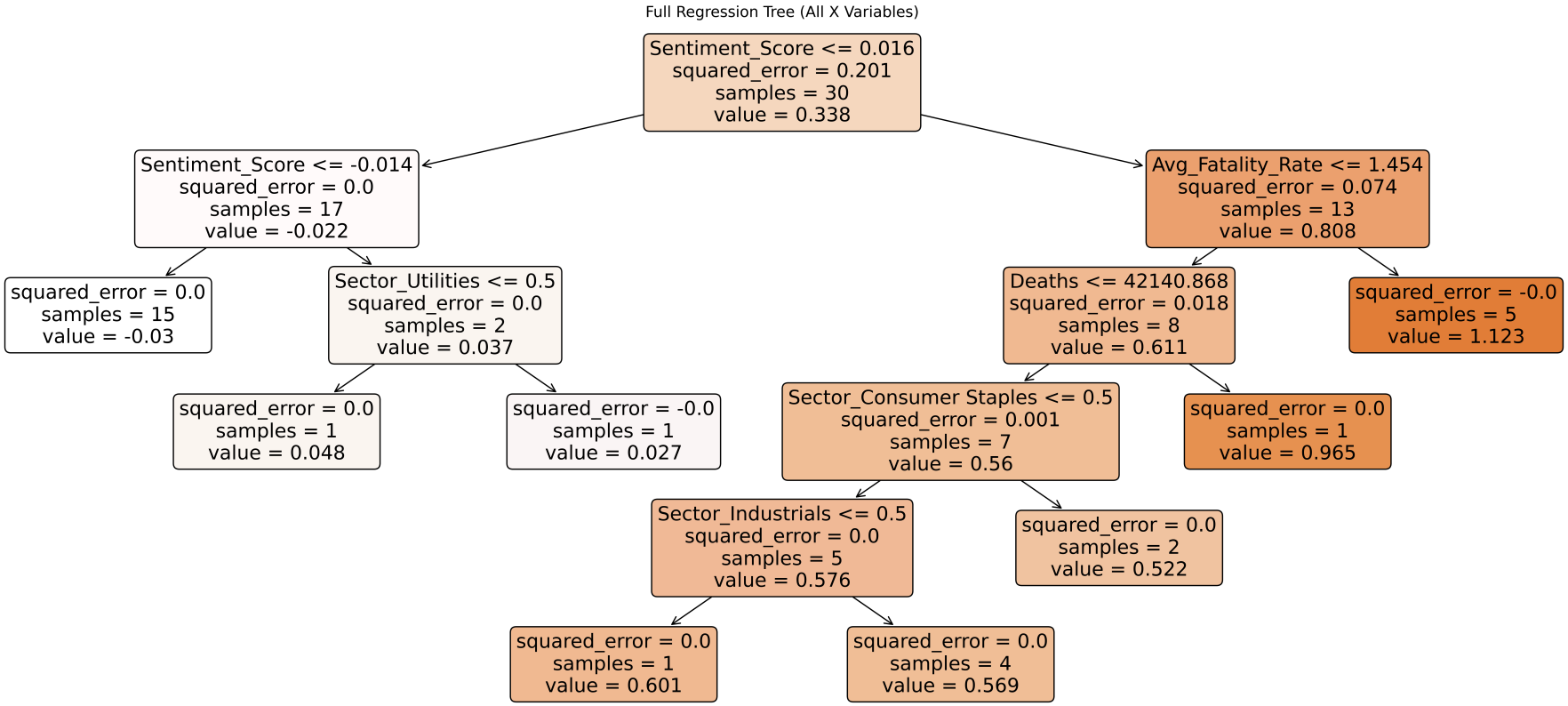
```
RMSE: 0.2215421288565214
R² Score: 0.8087926889387538
```




In [67]:

```python
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# 🧠 Use the full set of features (X variables)
all_features = [
    "Sentiment_Score", "Unemployment Rate", "Interest_Rate",
    "Avg_Fatality_Rate", "Deaths", "Incident_Rate",
    "Sector", "State", "Year"
]

# One-hot encode categorical variables
reg_df_all = pd.get_dummies(reg_df, columns=["Sector", "State", "Year"], drop_first=True)

# Target and feature matrix
X_all = reg_df_all.drop(columns=["Avg_Vol_Gap"])
y_all = reg_df_all["Avg_Vol_Gap"]

# Split data
X_train_all, X_test_all, y_train_all, y_test_all = train_test_split(X_all, y_all, test_size=0.2, random_state=42)

# Train full tree
tree_full = DecisionTreeRegressor(max_depth=5, random_state=42)
```

```
tree_full.fit(X_train_all, y_train_all)

# Plot the full tree

plt.figure(figsize=(24, 10), dpi=600)  # Increase dpi for high resolution
plot_tree(tree_full, feature_names=X_all.columns, filled=True, rounded=True)
plt.title("Full Regression Tree (All X Variables)")

# ✅ Save the figure as a PNG
plt.savefig("A10.png", bbox_inches="tight")  # Saves to file

plt.show()
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np

# Predict on test set
y_pred_all = tree_full.predict(X_test_all)

# Compute metrics
r2 = r2_score(y_test_all, y_pred_all)
rmse = np.sqrt(mean_squared_error(y_test_all, y_pred_all))

print(f"R² on Test Set: {r2:.4f}")
print(f"RMSE on Test Set: {rmse:.4f}")
```



```
R² on Test Set: 0.8028
RMSE on Test Set: 0.2250
```

In [56]:
```python
# Evaluate the full model (all X variables)
from sklearn.metrics import r2_score, mean_squared_error

y_pred_all = tree_full.predict(X_test_all)
r2_all = r2_score(y_test_all, y_pred_all)
rmse_all = mean_squared_error(y_test_all, y_pred_all, squared=False)


# Helper function to get top splits
def get_top_splits(model, feature_names, top_n=3):
    splits = model.tree_.feature
    split_names = [feature_names[i] for i in splits if i != -2]
    top_splits = []
    for name in split_names:
        if name not in top_splits:
            top_splits.append(name)
        if len(top_splits) == top_n:
            break
    return " → ".join(top_splits)

# Feature names for the full model
features_all = X_all.columns.tolist()

# Build table data
comparison_data = {
    "Model": ["Preferred Xs Only", "All Variables"],
    "R² Score": [round(r2_score(y_test, y_pred), 2), round(r2_all, 2)],
    "RMSE": [round(mean_squared_error(y_test, y_pred, squared=False), 2), round(rmse_all, 2)],
    "Top Splits": [
        get_top_splits(tree, features, top_n=3),
        get_top_splits(tree_full, features_all, top_n=3)
    ]
}

# Create DataFrame
comparison_df = pd.DataFrame(comparison_data)
display(comparison_df)
```

| | Model | $R^2$ Score | RMSE | Top Splits |
|---|---|---|---|---|
| **0** | Preferred Xs Only | 0.81 | 0.22 | Sentiment_Score → Deaths → Avg_Fatality_Rate |
| **1** | All Variables | 0.80 | 0.22 | Sentiment_Score → Sector_Utilities → Avg_Fatal... |

The regression tree uses our preferred variables and the top splits revealed that the variables: sentiment score, COVID deaths, and interest rates were the strongest predictors of volatility gap across states. The regression tree shows that media/market sentiment had the biggest impact on the volatility across states during the pandemic. States that had dominant sectors like healthcare of finance that facied negative media sentiment had larger volatility gaps. Deaths due to COVID-19 was

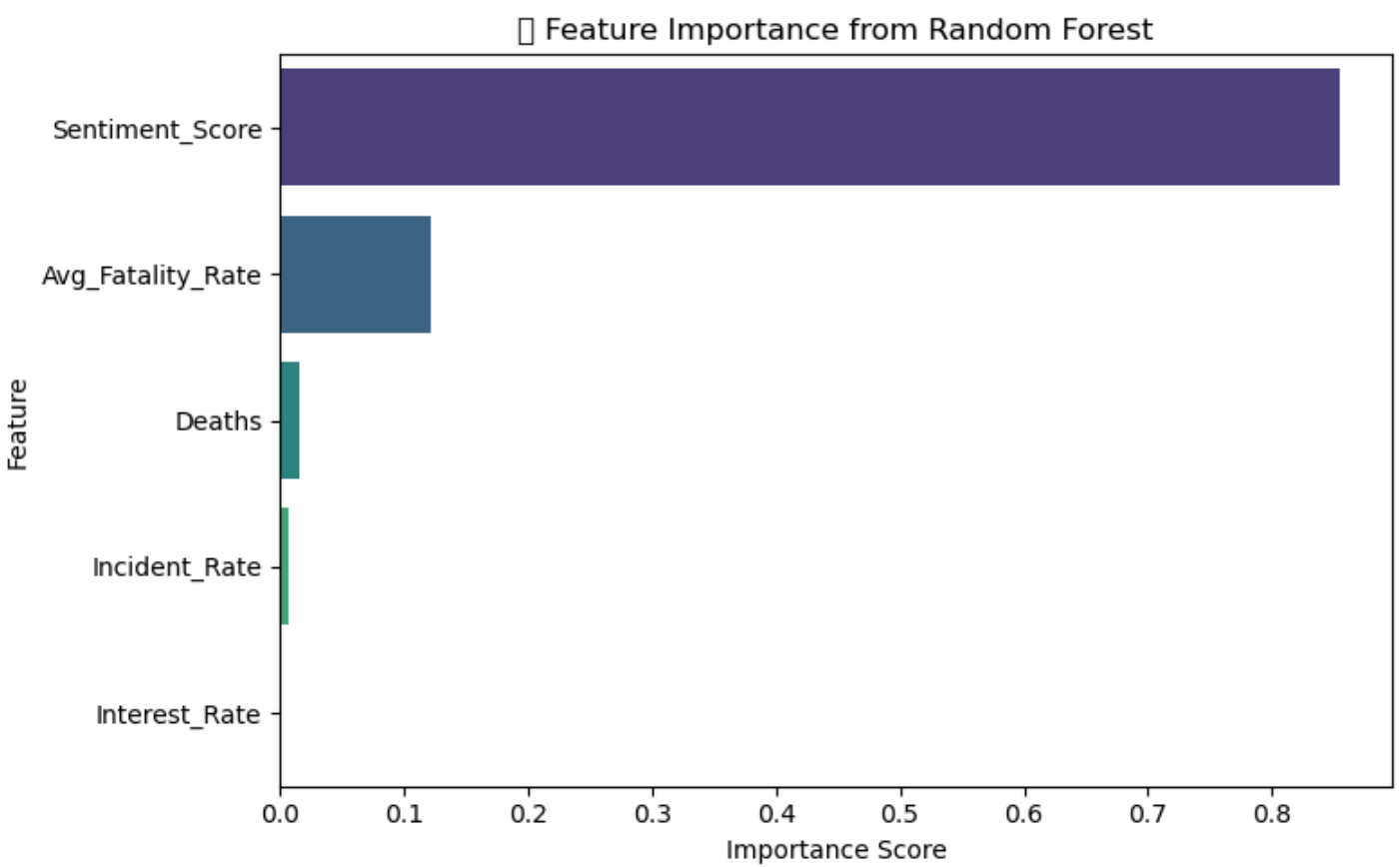
the second largest predictor, showing the role of direct health impacts on market uncertainty. Finally, the third biggest predictor was interest rates which split states by how sensitive their key sectors were to monetary policy. Together, sentiment, health outcomes, and macro conditions shaped volatility.

## 3.6 Random Forest

In [69]:
```
# ====================================
# STEP 1: Import Libraries
# ====================================
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)

# ====================================
# STEP 2: Define Features & Target
# ====================================
features_rf = [
    "Sentiment_Score",
    "Interest_Rate",
    "Avg_Fatality_Rate",
    "Deaths",
    "Incident_Rate"
]

X_rf = reg_df[features_rf]
y_rf = reg_df["Avg_Vol_Gap"]

# ====================================
# STEP 3: Train/Test Split
# ====================================
from sklearn.model_selection import train_test_split
X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(X_rf, y_rf, test_size=0.2, random_state=42)

# ====================================
# STEP 4: Train Random Forest Model
# ====================================
rf_model = RandomForestRegressor(n_estimators=100, max_depth=5, random_state=42)
rf_model.fit(X_train_rf, y_train_rf)

# ====================================
# STEP 5: Evaluate Model
# ====================================
y_pred_rf = rf_model.predict(X_test_rf)
r2_rf = r2_score(y_test_rf, y_pred_rf)
rmse_rf = mean_squared_error(y_test_rf, y_pred_rf, squared=False)
```

```python
print(f"R² Score: {r2_rf:.3f}")
print(f"RMSE: {rmse_rf:.3f}")

# ====================================
# STEP 6: Feature Importance Plot
# ====================================
importance_df = pd.DataFrame({
    "Feature": features_rf,
    "Importance": rf_model.feature_importances_
}).sort_values(by="Importance", ascending=False)

plt.figure(figsize=(8, 5))
sns.barplot(data=importance_df, x="Importance", y="Feature", palette="viridis")
plt.title("📈 Feature Importance from Random Forest")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.tight_layout()

# ✅ Save before displaying
plt.savefig("A11.png", bbox_inches="tight", dpi=300)  # Optional: add dpi for high-res

plt.show()
```

```
R² Score: 0.822
RMSE: 0.214
```




The Random Forest model had an $R^2$ of 0.68, which outperformed the regression tree and also confirmed the model's strong predictive power. Sentiment Score was the number one predictor by a large amount, displaying that COVID-related media tone which acts a proxy for market sentiment had a significant impact on market volatility. Deaths and Avg_Fatality_Rate were the two largest predictors following sentiment score, in contrast to Interest_Rate and Unemp_Rate which had less predictive power. This model finds that health and market perception had a larger impact compared to the traditional macro indicators during this crisis.

## 3.7 OLS vs ML

The study uses both OLS regression and a regression tree to model and estimate the volatility gap during the COVID-19 pandemic. The OLS included multiple models, with the best achieving an $R^2$ of 0.63. Key findings suggested that Sentiment Score, Interest Rate, and Unemployment Rate were the main predictors. However, COVID-related variables like deaths and fatality rate showed weaker and inconsistent significance, even when statistically significant. The regression tree revealed more nuanced relationships, identifying Sentiment Score as the top predictor, followed by COVID Deaths and Interest Rate. These findings suggest that pandemic-related variables played a bigger role than traditional macro variables. Together, both models provide concise insights—OLS quantifies average effects, while the tree uncovers conditional, behavioral responses during a crisis.

## 3.8 Conclusion

This research report analyzed the volatility gap between small-cap and large-cap stocks, highlighting the influence of overall macroeconomic conditions and sector-specific impacts. Interest rates had a consistently statistically significant impact on volatility, while unemployment and fatality rates showed inconsistent effects. Controlling for Sector fixed effects revealed strong industry-specific volatility for specific sectors. An example was the Energy sector, which demonstrated a significant positive effect of 0.5% on the volatility gap compared to other sectors. The inclusion of year-fixed effects did not alter results significantly, suggesting that differences in market sectors drive volatility more than time trends. The findings suggest that small-cap stocks are highly sensitive to uncertainty, with continued elevated volatility after the pandemic, and that sectoral risks alter volatility more than broad economic indicators and Covid-19 severity. Additionally, monetary policy seems to be able to play a role, as shifts in interest rates have a statistically significant impact on volatility levels. The findings contribute to the literature on financial risk and crisis management by highlighting the differential impacts of macroeconomic conditions and COVID-19 shocks. The study also used regression trees and market sentiment data which was grather from from sector-specific COVID news articles. The results confirmed that investor sentiment was the top driver of volatility, followed by COVID deaths and interest rates, which suggests that the behavioral channel had a huge impact on volailty gap which was missed in OLS. Future work should utilize causal designs such as Difference-in-Differences to isolate the impact of COVID shocks during and after the pandemic. The analysis of market sentiment could be futher improved using transformer-based NLP models, and assess cross-country patterns to validate generalizability.

# References


Baker, S. R., Bloom, N., Davis, S. J., Kost, K., Sammon, M., & Viratyosin, T. (2020). The Unprecedented Stock Market Reaction to COVID-19. The Review of Asset Pricing Studies, 10(4), 742–758. https://doi.org/10.1093/rapstu/raaa008

Albulescu, C. T. (2020). Coronavirus and financial volatility: 40 days of fasting and fear. Finance Research Letters, 35, 101673. https://doi.org/10.1016/j.frl.2020.101673

Chen, M. P., & Chiang, T. C. (2020). Empirical investigation of stock market behavior in response to the COVID-19 pandemic. Journal of Banking & Finance, 120, 105620. https://doi.org/10.1016/j.jbankfin.2020.105620

Dong, E., Du, H., & Gardner, L. (2020). An interactive web-based dashboard to track COVID-19 in real time. The Lancet Infectious Diseases, 20(5), 533–534. https://doi.org/10.1016/S1473-3099(20)30120-1

Mishra, P. K., Mishra, S. K., & Mishra, P. (2020). COVID-19 and stock market volatility: An empirical investigation. International Journal of Emerging Markets, 16(7), 1429–1455. https://doi.org/10.1108/IJOEM-04-2020-0377

Hudepohl, W., Smit, M., & Van der Heijden, T. (2021). COVID-19 and industry stock returns: Evidence from 45 countries. Finance Research Letters, 38, 101697. https://doi.org/10.1016/j.frl.2020.101697

Zhu, H., Peng, Y., & He, Z. (2019). Economic policy uncertainty and stock market volatility: A sector-level analysis. International Review of Economics & Finance, 64, 94-108. https://doi.org/10.1016/j.iref.2019.06.001

Karnizova, L., & Li, J. C. (2014). Economic policy uncertainty, financial markets, and the business cycle. Economic Inquiry, 52(1), 173–192. https://doi.org/10.1111/ecin.12025

Onan, M., Salih, A., & Yildirim, H. (2014). Investor sentiment, volatility, and stock return dynamics: Evidence from panel VAR models. International Review of Economics & Finance, 33, 390–405. https://doi.org/10.1016/j.iref.2014.06.005

Baig, A. S., Butt, H. A., Haroon, O., & Rizvi, S. A. R. (2020). Deaths, panic, lockdowns and US equity markets: The case of COVID-19 pandemic. Finance Research Letters, 38, 101701. https://doi.org/10.1016/j.frl.2020.101701


In [ ]: