# Correlation between stocks and sentiment analysis:

Analysis by month from TRUMP's Nomination for the elections till the election results.

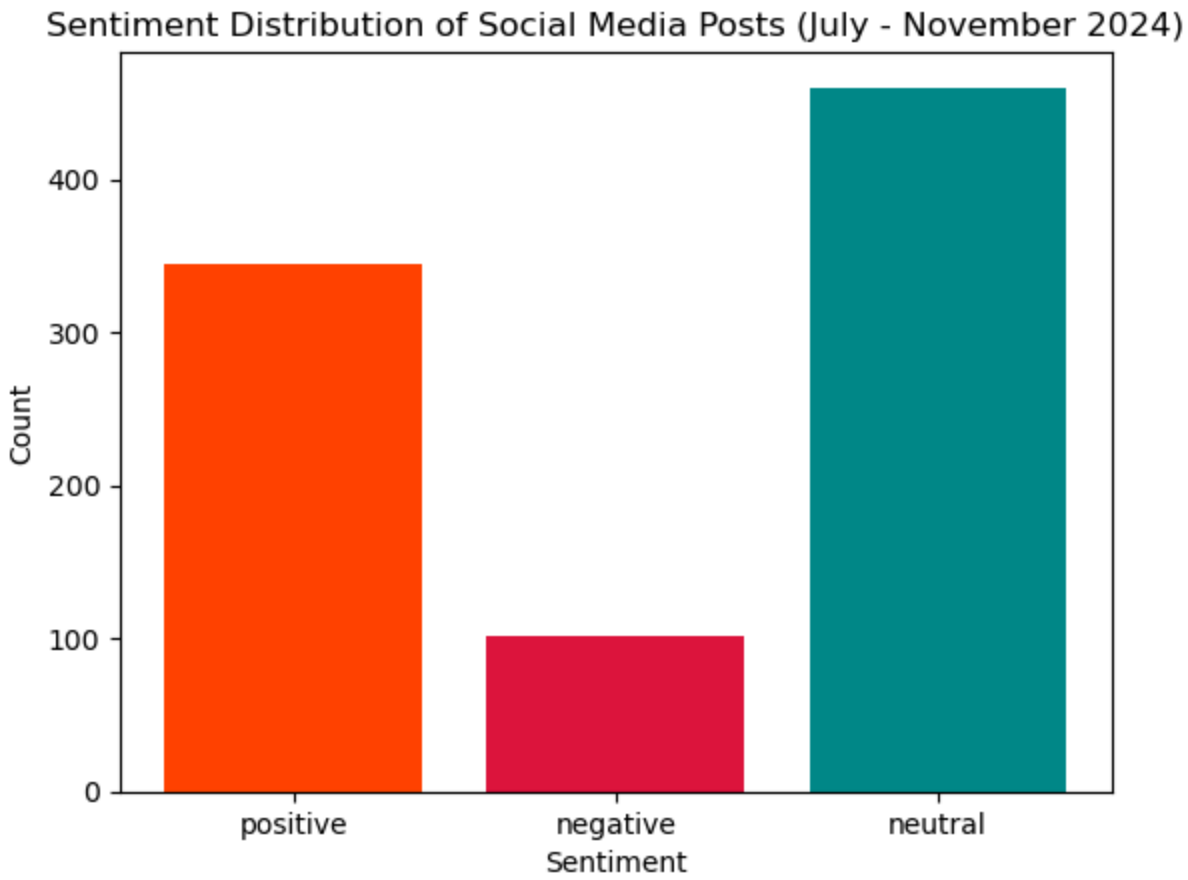
In [33]:

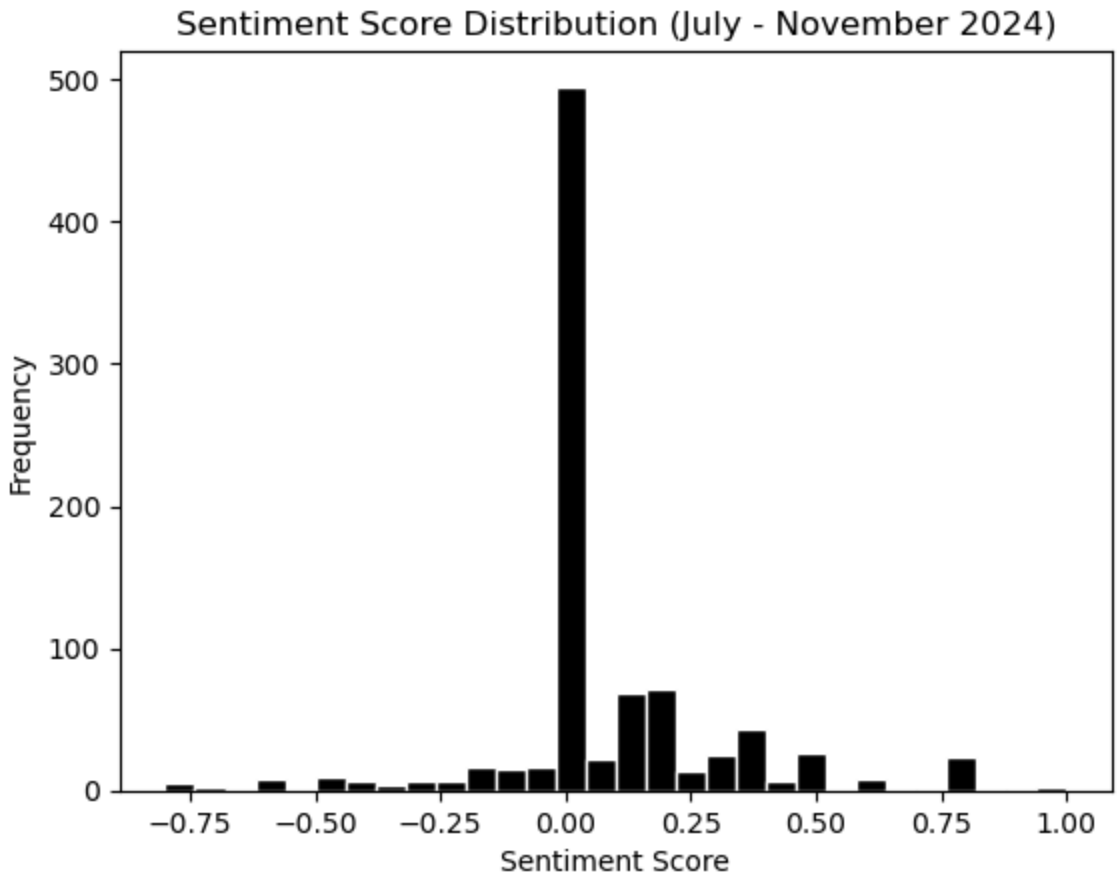
```python
import pandas as pd
import json
from datetime import datetime
import matplotlib.pyplot as plt

# Load  data
with open("cleaned_finance_API_Reddit_with_dates.json") as file:
    reddit_data = json.load(file)

# Function to parse and convert date to the required format
def parse_date(date_str):
    # Check if the date is in ISO format with timezone offset
    try:
        return datetime.strptime(date_str, '%Y-%m-%dT%H:%M:%S+0000').strftime('%Y-%m-%d %H:%M:%S')  # ISO format to simp
    except ValueError:
        pass

    # Check if the date has a 'Z' at the end (UTC)
    try:
        return datetime.strptime(date_str, '%Y-%m-%dT%H:%M:%SZ').strftime('%Y-%m-%d %H:%M:%S')  # Handle 'Z' in ISO form
    except ValueError:
        pass

    # If it's not in ISO format, check if it's in the simple datetime format
    try:
        return datetime.strptime(date_str, '%Y-%m-%d %H:%M:%S').strftime('%Y-%m-%d %H:%M:%S')
    except ValueError:
        raise ValueError(f"Date format not recognized: {date_str}")

# Convert all Reddit data dates to the required format
for post in reddit_data:
    post['date'] = parse_date(post['date'])



# Now proceed with sentiment analysis :

start_date = datetime(2024, 7, 1).strftime('%Y-%m-%d %H:%M:%S')  # Month of Trump's Nomination for Presidential Run
end_date = datetime(2024, 11, 30).strftime('%Y-%m-%d %H:%M:%S')

# Now filter Reddit posts for the specified date range
filtered_reddit_data = [
    post for post in reddit_data
    if start_date <= post['date'] <= end_date
]

# Sentiment analysis function (if you want to continue with TextBlob)
```

```python
def analyze_sentiment(text):
    from textblob import TextBlob
    blob = TextBlob(text)
    sentiment_score = blob.sentiment.polarity  # Returns a sentiment score between -1 and 1
    if sentiment_score > 0:
        return 'positive', sentiment_score
    elif sentiment_score < 0:
        return 'negative', sentiment_score
    else:
        return 'neutral', sentiment_score

# Apply sentiment analysis to Reddit posts using the 'title' key
for post in filtered_reddit_data:
    sentiment, score = analyze_sentiment(post['title'])
    post['sentiment'] = sentiment
    post['sentiment_score'] = score

# Count sentiment types
sentiment_counts = {"positive": 0, "negative": 0, "neutral": 0}
sentiment_scores = []

for post in filtered_reddit_data:
    sentiment_counts[post['sentiment']] += 1
    sentiment_scores.append(post['sentiment_score'])

# Calculate average sentiment score
average_sentiment_score = sum(sentiment_scores) / len(sentiment_scores) if sentiment_scores else 0

# Display sentiment analysis results
print(f"Sentiment Analysis on Social Media Posts from July 1 to November 30, 2024: {sentiment_counts}")
print(f"Average Sentiment Score: {average_sentiment_score:.2f}")

# Visualizing the sentiment distribution
sentiments = list(sentiment_counts.keys())
counts = list(sentiment_counts.values())

# Bar plot of sentiment distribution
plt.bar(sentiments, counts, color=['orangered', 'crimson', 'darkcyan'])
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.title('Sentiment Distribution of Social Media Posts (July - November 2024)')
plt.show()

# Sentiment score distribution (Optional - This will help you understand the intensity)
plt.hist(sentiment_scores, bins=30, color='black', edgecolor='white')
plt.title('Sentiment Score Distribution (July - November 2024)')
plt.xlabel('Sentiment Score')
plt.ylabel('Frequency')
```

```
plt.show()
```

```
Sentiment Analysis on Social Media Posts from July 1 to November 30, 2024: {'positive': 345, 'negative': 102, 'neutra
l': 460}
Average Sentiment Score: 0.07
```

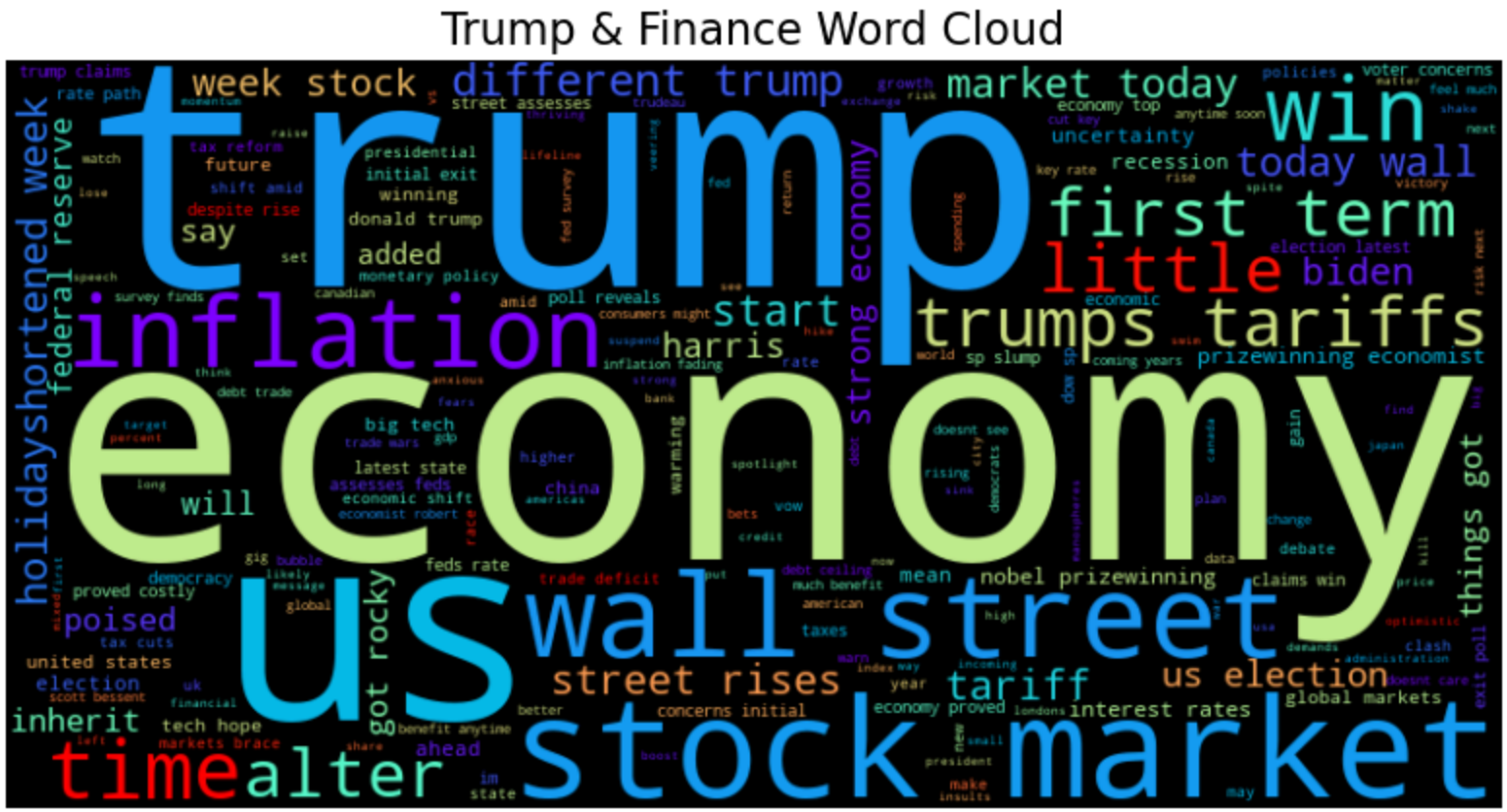





## Sentiment Word Cloud

In [35]:
```
import json
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import re

# Load JSON file (Make sure the path is correct)
file_path = 'cleaned_finance_API_Reddit_with_dates.json'
with open(file_path, 'r') as f:
    data = json.load(f)

# Define finance-related keywords related to Trump and the USA
finance_keywords = [
    'economy', 'u.s. economy', 'gdp', 'stock market', 'wall street', 'inflation',
    'unemployment rate', 'recession', 'federal reserve', 'interest rates',
    'economic growth', 'fiscal policy', 'monetary policy', 'debt ceiling',
    'national debt', 'budget deficit', 'trade deficit', 'treasury bonds',
    'tax reform', 'taxes', 'tax cuts', 'tax policy', 'trump economy',
    'trump tax cuts', 'trump trade policy', 'trump administration economy',
    'america first trade policy', 'trump tariffs', 'trump budget',
    'trump’s economic plan', 'trump deregulation', 'trump’s financial policy',
    'trump stock market', 'trump inflation', 'trump unemployment', 'business growth',
    'corporate taxes', 'corporate regulation', 'financial markets', 'investor sentiment',
    'private sector', 'small business', 'big tech', 'startups', 'venture capital',
    'mergers and acquisitions', 'economic stimulus', 'government spending',
    'infrastructure plan', 'trade deals', 'china trade deal', 'nafta', 'usmca',
    'foreign investment', 'global markets', 'economic sanctions', 'united states',
    'american workers', 'american jobs', 'middle class', 'american consumers',
    'consumer confidence', 'american manufacturing', 'american businesses',
    'us trade', 'us exports', 'us imports'
]

# Extract and filter text based on keywords
def filter_relevant_text(entries, keywords):
    """
    Extracts and filters text data based on given keywords.
    """
    filtered_texts = []
    for entry in entries:
        text = entry.get('title', '').lower()  # Assuming 'title' holds the text
        if any(keyword in text for keyword in keywords):  # Check if any keyword exists in text
            filtered_texts.append(text)
    return filtered_texts

# Filter data
filtered_texts = filter_relevant_text(data, finance_keywords)

# Combine all filtered text into a single string
```

```
filtered_text = " ".join(filtered_texts)

# Optional: Clean the text (remove URLs, special characters, etc.)
def clean_text(text):
    text = re.sub(r'http\S+', '', text)  # Remove URLs
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove special characters
    return text.lower()  # Convert to lowercase

cleaned_text = clean_text(filtered_text)

# Generate the word cloud
wordcloud = WordCloud(width=800, height=400, background_color='black', colormap='rainbow').generate(cleaned_text)

# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title("Trump & Finance Word Cloud", fontsize=16)
plt.show()
```

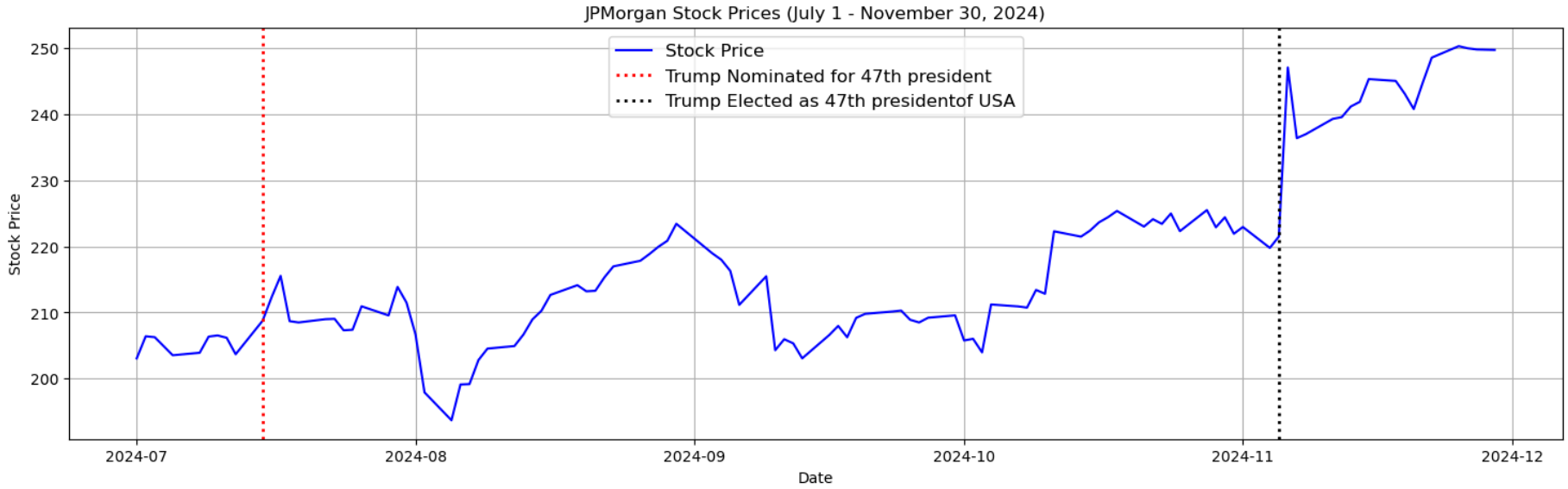


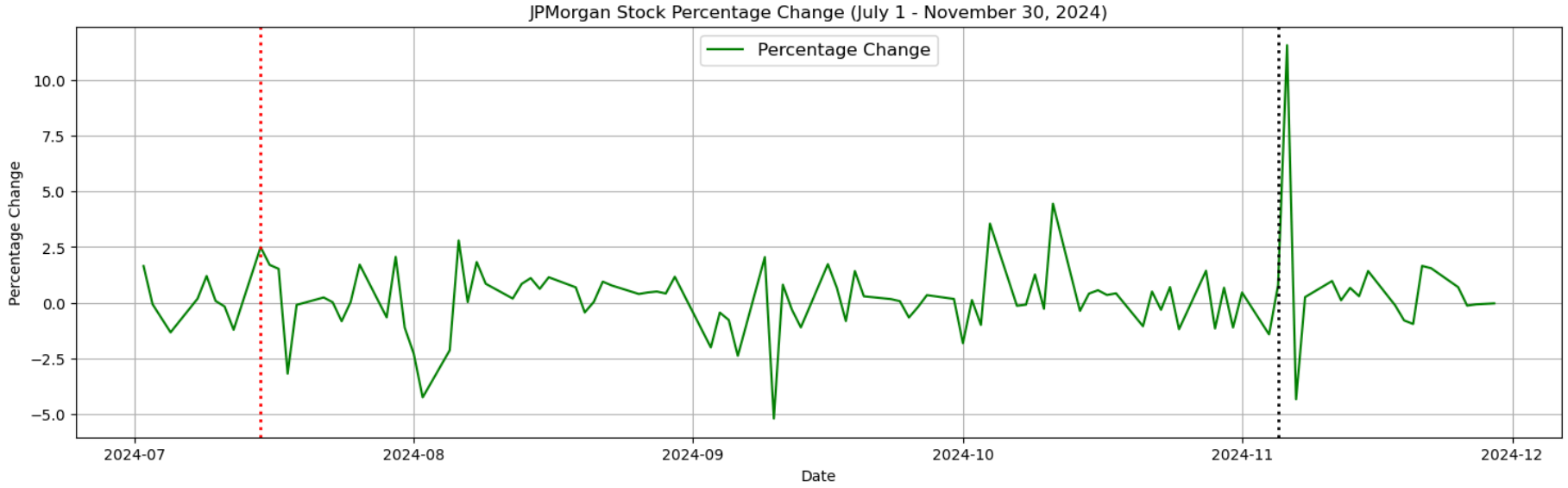
In [26]:

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

# Load stock data for JPMorgan (Assuming you have already loaded this)
stock_data = pd.read_csv("jp_morgan_stock_data.csv")  # Replace with your actual file path

# Ensure Date column is properly converted to datetime
stock_data['Date'] = pd.to_datetime(stock_data['Date'], errors='coerce')

# Check for invalid dates and drop them
if stock_data['Date'].isnull().any():
    print("Some rows have invalid dates and will be dropped.")
stock_data = stock_data.dropna(subset=['Date'])

# Remove timezone information to make the Date column timezone-naive
stock_data['Date'] = stock_data['Date'].apply(lambda x: x.replace(tzinfo=None) if x.tzinfo else x)

# Define the date range for filtering
start_date = datetime(2024, 7, 1)
end_date = datetime(2024, 11, 30)

# Filter stock data within the date range
stock_data_filtered = stock_data[
    (stock_data['Date'] >= start_date) & (stock_data['Date'] <= end_date)
]

# Calculate percentage change in stock prices
stock_data_filtered['Pct_Change'] = stock_data_filtered['Close'].pct_change() * 100

# Plot stock prices
plt.figure(figsize=(18, 5))
plt.plot(stock_data_filtered['Date'], stock_data_filtered['Close'], label='Stock Price', color='blue')
plt.title('JPMorgan Stock Prices (July 1 - November 30, 2024)')
plt.xlabel('Date')
plt.ylabel('Stock Price')

# Add dotted vertical lines on July 15 and November 5
plt.axvline(datetime(2024, 7, 15), color='red', linestyle=':', linewidth=2, label='Trump Nominated for 47th president')
plt.axvline(datetime(2024, 11, 5), color='black', linestyle=':', linewidth=2, label='Trump Elected as 47th presidentof U



# Add legends for the main plot and the vertical lines
plt.legend(loc='upper center', fontsize=12, frameon=True)

# Show the grid and the plot
```

```
plt.grid(True)
plt.show()

# Plot percentage change in stock price
plt.figure(figsize=(18, 5))
plt.plot(stock_data_filtered['Date'], stock_data_filtered['Pct_Change'], label='Percentage Change', color='green')
plt.title('JPMorgan Stock Percentage Change (July 1 - November 30, 2024)')
plt.xlabel('Date')
plt.ylabel('Percentage Change')

# Add dotted vertical lines on July 15 and November 5
plt.axvline(datetime(2024, 7, 15), color='red', linestyle=':', linewidth=2,)
plt.axvline(datetime(2024, 11, 5), color='black', linestyle=':', linewidth=2)

plt.legend(loc='upper center', fontsize=12)
plt.grid(True)
plt.show()

# Display stock price statistics
print(f"JPMorgan Stock Price Statistics (July 1 - November 30, 2024):")
print(stock_data_filtered[['Date', 'Close']].describe())
```

```
C:\Users\admin\AppData\Local\Temp\ipykernel_8796\1917637602.py:29: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  stock_data_filtered['Pct_Change'] = stock_data_filtered['Close'].pct_change() * 100
```




JPMorgan Stock Price Statistics (July 1 - November 30, 2024):

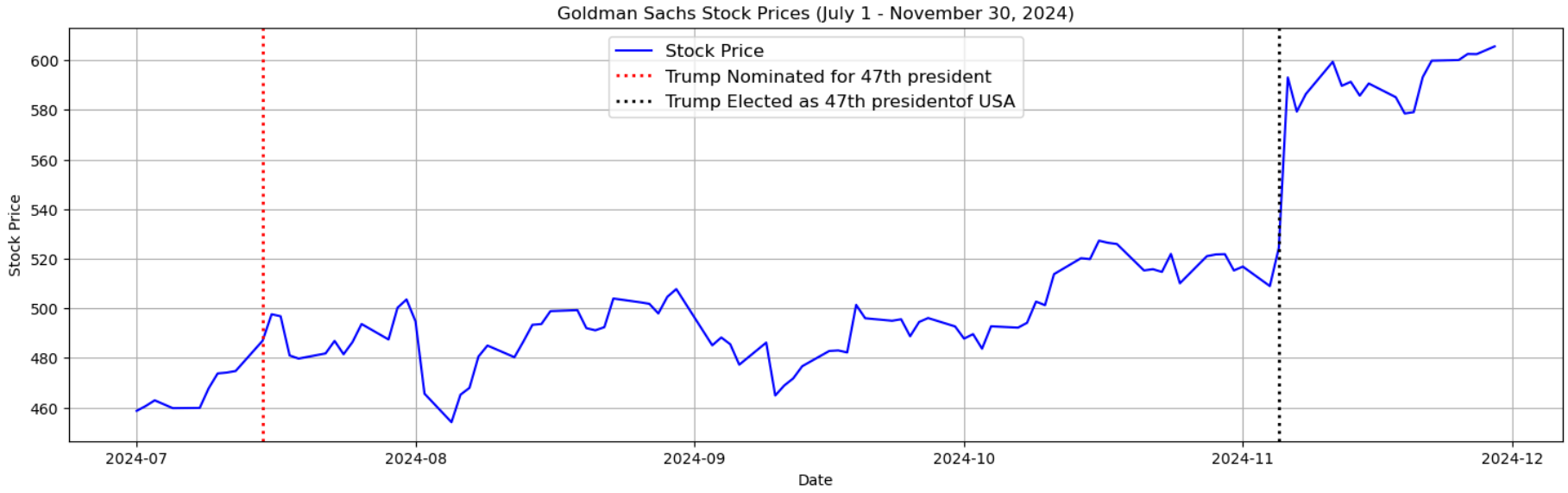
| | Date | Close |
|---|---|---|
| count | 107 | 107.000000 |
| mean | 2024-09-14 14:21:18.504673024 | 217.157280 |
| min | 2024-07-01 00:00:00 | 193.712921 |
| 25% | 2024-08-07 12:00:00 | 207.006462 |
| 50% | 2024-09-16 00:00:00 | 212.666779 |
| 75% | 2024-10-22 12:00:00 | 222.970001 |
| max | 2024-11-29 00:00:00 | 250.289993 |
| std | NaN | 13.727205 |

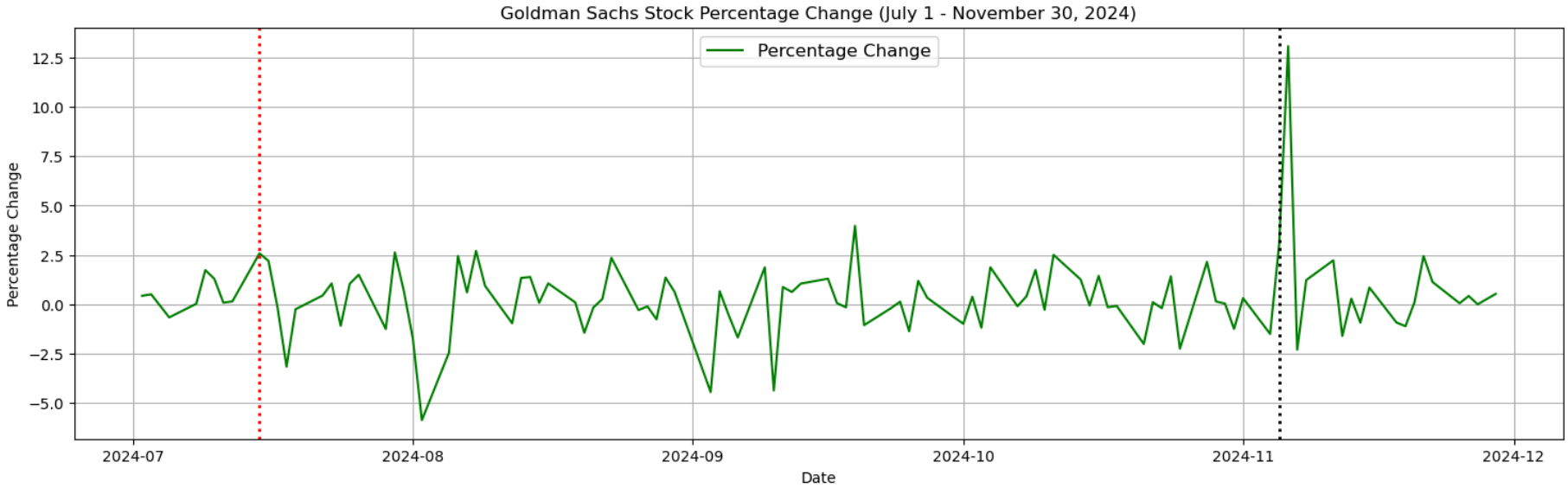
In [118]:

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

# Load stock data for JPMorgan (Assuming you have already loaded this)
stock_data = pd.read_csv("goldmansachs_stock_data.csv")  # Replace with your actual file path

# Ensure Date column is properly converted to datetime
stock_data['Date'] = pd.to_datetime(stock_data['Date'], errors='coerce')

# Check for invalid dates and drop them
if stock_data['Date'].isnull().any():
    print("Some rows have invalid dates and will be dropped.")
stock_data = stock_data.dropna(subset=['Date'])

# Remove timezone information to make the Date column timezone-naive
stock_data['Date'] = stock_data['Date'].apply(lambda x: x.replace(tzinfo=None) if x.tzinfo else x)

# Define the date range for filtering
start_date = datetime(2024, 7, 1)
end_date = datetime(2024, 11, 30)

# Filter stock data within the date range
stock_data_filtered = stock_data[
    (stock_data['Date'] >= start_date) & (stock_data['Date'] <= end_date)
]

# Calculate percentage change in stock prices
stock_data_filtered['Pct_Change'] = stock_data_filtered['Close'].pct_change() * 100

# Plot stock prices
plt.figure(figsize=(18, 5))
plt.plot(stock_data_filtered['Date'], stock_data_filtered['Close'], label='Stock Price', color='blue')
plt.title('Goldman Sachs Stock Prices (July 1 - November 30, 2024)')
plt.xlabel('Date')
plt.ylabel('Stock Price')

# Add dotted vertical lines on July 15 and November 5
plt.axvline(datetime(2024, 7, 15), color='red', linestyle=':', linewidth=2, label='Trump Nominated for 47th president')
plt.axvline(datetime(2024, 11, 5), color='black', linestyle=':', linewidth=2, label='Trump Elected as 47th presidentof U



# Add legends for the main plot and the vertical lines
plt.legend(loc='upper center', fontsize=12, frameon=True)

# Show the grid and the plot
```

```
plt.grid(True)
plt.show()

# Plot percentage change in stock price
plt.figure(figsize=(18, 5))
plt.plot(stock_data_filtered['Date'], stock_data_filtered['Pct_Change'], label='Percentage Change', color='green')
plt.title('Goldman Sachs Stock Percentage Change (July 1 - November 30, 2024)')
plt.xlabel('Date')
plt.ylabel('Percentage Change')

# Add dotted vertical lines on July 15 and November 5
plt.axvline(datetime(2024, 7, 15), color='red', linestyle=':', linewidth=2,)
plt.axvline(datetime(2024, 11, 5), color='black', linestyle=':', linewidth=2)

plt.legend(loc='upper center', fontsize=12)
plt.grid(True)
plt.show()

# Display stock price statistics
print(f"Goldman Sachs Stock Price Statistics (July 1 - November 30, 2024):")
print(stock_data_filtered[['Date', 'Close']].describe())
```

```
C:\Users\admin\AppData\Local\Temp\ipykernel_12840\2669022812.py:29: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  stock_data_filtered['Pct_Change'] = stock_data_filtered['Close'].pct_change() * 100
```






Goldman Sachs Stock Price Statistics (July 1 - November 30, 2024):

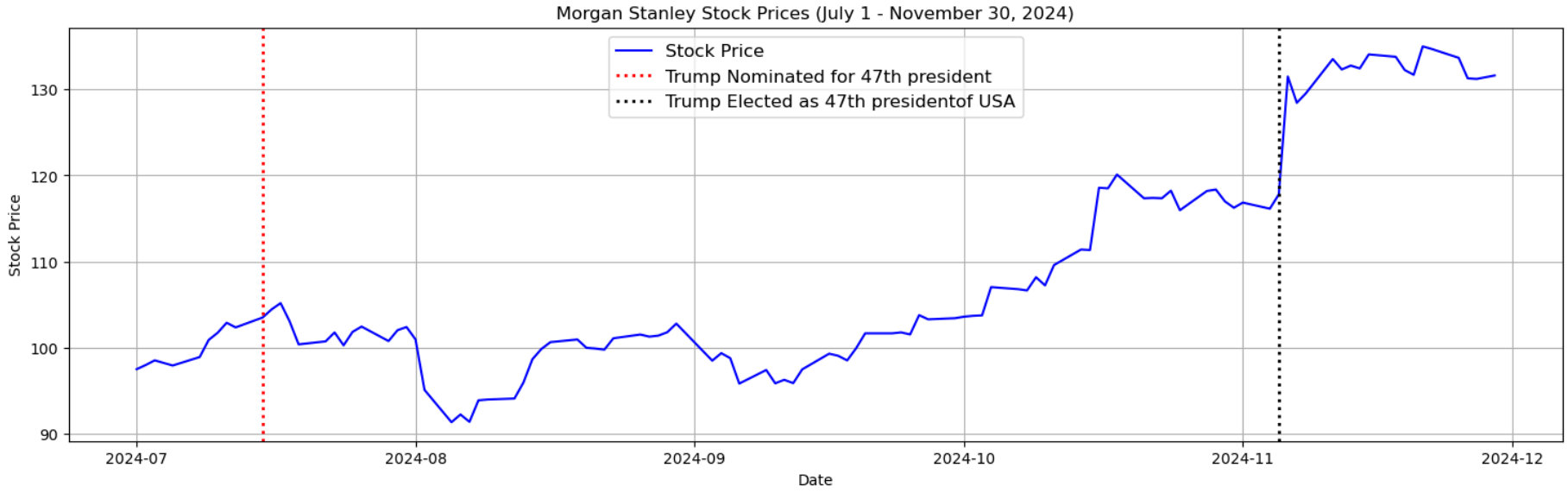
| | Date | Close |
|---|---|---|
| count | 107 | 107.000000 |
| mean | 2024-09-14 14:21:18.504673024 | 508.149918 |
| min | 2024-07-01 00:00:00 | 454.070923 |
| 25% | 2024-08-07 12:00:00 | 483.350479 |
| 50% | 2024-09-16 00:00:00 | 494.957977 |
| 75% | 2024-10-22 12:00:00 | 518.297333 |
| max | 2024-11-29 00:00:00 | 605.570007 |
| std | NaN | 40.221899 |

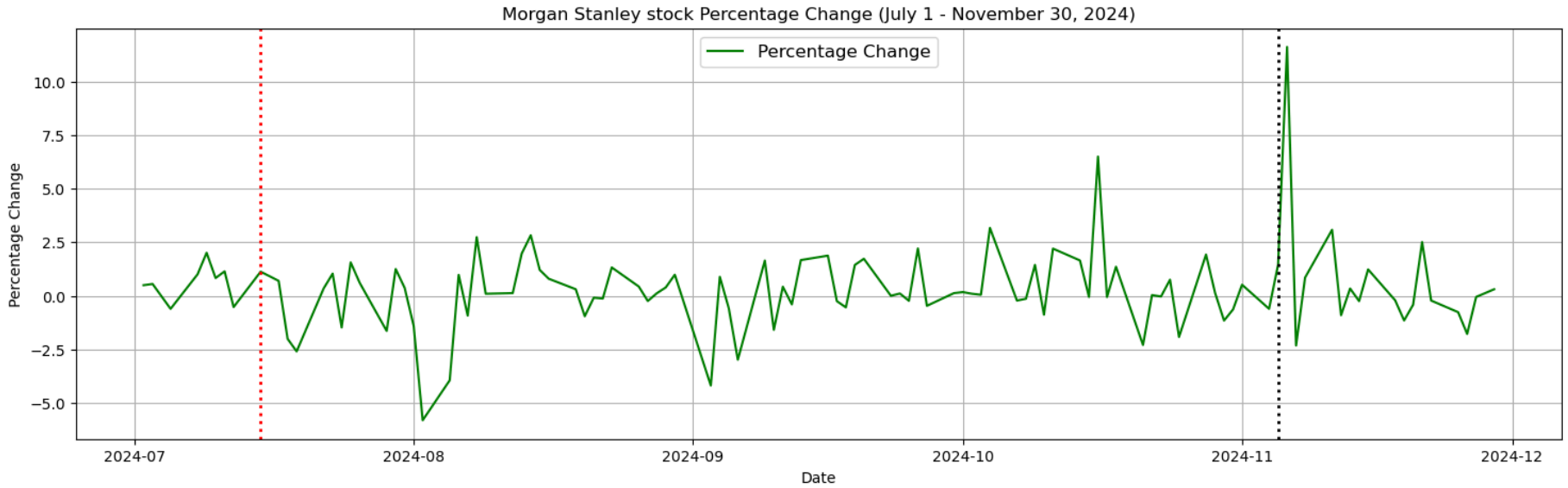
In [117]:

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

# Load stock data for JPMorgan (Assuming you have already loaded this)
stock_data = pd.read_csv("morgan_stanley_stock_data.csv")  # Replace with your actual file path

# Ensure Date column is properly converted to datetime
stock_data['Date'] = pd.to_datetime(stock_data['Date'], errors='coerce')

# Check for invalid dates and drop them
if stock_data['Date'].isnull().any():
    print("Some rows have invalid dates and will be dropped.")
stock_data = stock_data.dropna(subset=['Date'])

# Remove timezone information to make the Date column timezone-naive
stock_data['Date'] = stock_data['Date'].apply(lambda x: x.replace(tzinfo=None) if x.tzinfo else x)

# Define the date range for filtering
start_date = datetime(2024, 7, 1)
end_date = datetime(2024, 11, 30)

# Filter stock data within the date range
stock_data_filtered = stock_data[
    (stock_data['Date'] >= start_date) & (stock_data['Date'] <= end_date)
]

# Calculate percentage change in stock prices
stock_data_filtered['Pct_Change'] = stock_data_filtered['Close'].pct_change() * 100

# Plot stock prices
plt.figure(figsize=(18, 5))
plt.plot(stock_data_filtered['Date'], stock_data_filtered['Close'], label='Stock Price', color='blue')
plt.title('Morgan Stanley Stock Prices (July 1 - November 30, 2024)')
plt.xlabel('Date')
plt.ylabel('Stock Price')

# Add dotted vertical lines on July 15 and November 5
plt.axvline(datetime(2024, 7, 15), color='red', linestyle=':', linewidth=2, label='Trump Nominated for 47th president')
plt.axvline(datetime(2024, 11, 5), color='black', linestyle=':', linewidth=2, label='Trump Elected as 47th presidentof U



# Add legends for the main plot and the vertical lines
plt.legend(loc='upper center', fontsize=12, frameon=True)

# Show the grid and the plot
```

```
plt.grid(True)
plt.show()

# Plot percentage change in stock price
plt.figure(figsize=(18, 5))
plt.plot(stock_data_filtered['Date'], stock_data_filtered['Pct_Change'], label='Percentage Change', color='green')
plt.title('Morgan Stanley stock Percentage Change (July 1 - November 30, 2024)')
plt.xlabel('Date')
plt.ylabel('Percentage Change')

# Add dotted vertical lines on July 15 and November 5
plt.axvline(datetime(2024, 7, 15), color='red', linestyle=':', linewidth=2,)
plt.axvline(datetime(2024, 11, 5), color='black', linestyle=':', linewidth=2)

plt.legend(loc='upper center', fontsize=12)
plt.grid(True)
plt.show()

# Display stock price statistics
print(f"Morgan Stanley Stock Price Statistics (July 1 - November 30, 2024):")
print(stock_data_filtered[['Date', 'Close']].describe())
```

```
C:\Users\admin\AppData\Local\Temp\ipykernel_12840\4025251808.py:29: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  stock_data_filtered['Pct_Change'] = stock_data_filtered['Close'].pct_change() * 100
```






Morgan Stanley Stock Price Statistics (July 1 - November 30, 2024):

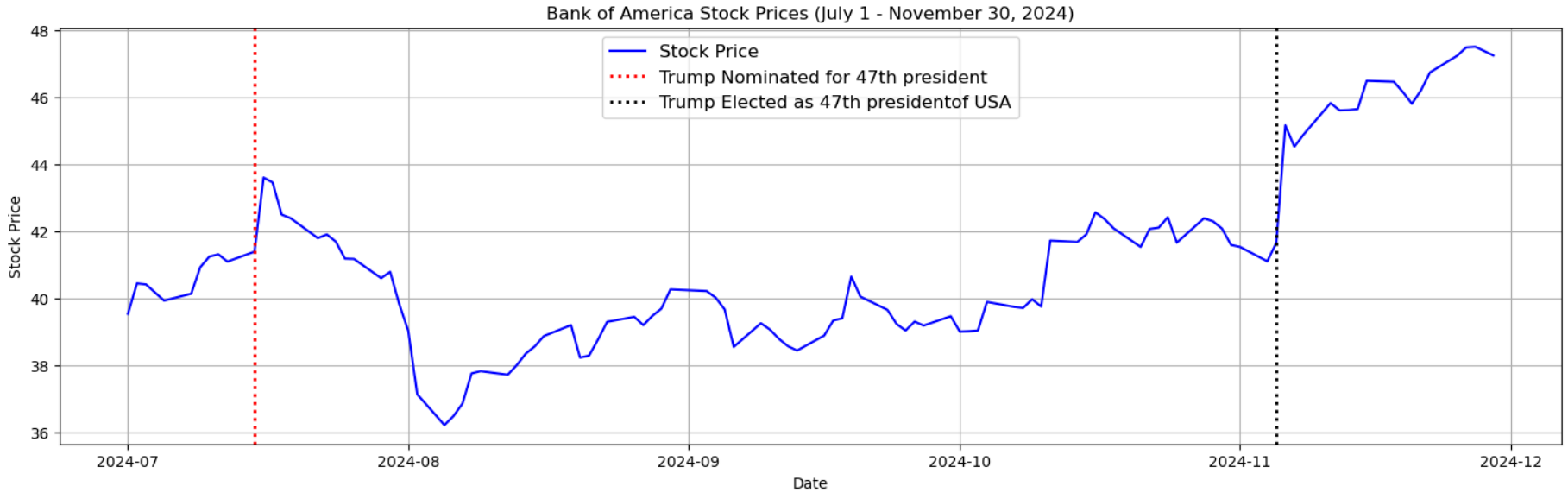
| | Date | Close |
|---|---|---|
| count | 107 | 107.000000 |
| mean | 2024-09-14 14:21:18.504673024 | 108.117266 |
| min | 2024-07-01 00:00:00 | 91.347771 |
| 25% | 2024-08-07 12:00:00 | 99.810860 |
| 50% | 2024-09-16 00:00:00 | 102.400383 |
| 75% | 2024-10-22 12:00:00 | 117.173584 |
| max | 2024-11-29 00:00:00 | 134.990005 |
| std | NaN | 12.563427 |

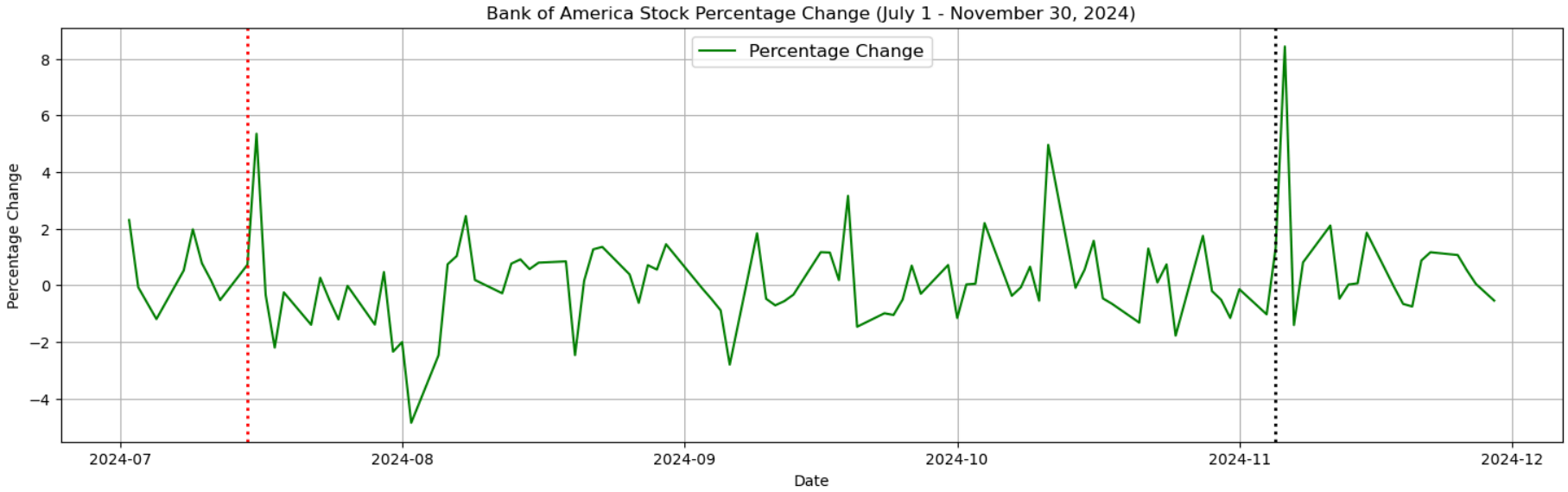
In [119]:

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

# Load stock data for JPMorgan (Assuming you have already loaded this)
stock_data = pd.read_csv("Bank of America_stock_data.csv")  # Replace with your actual file path

# Ensure Date column is properly converted to datetime
stock_data['Date'] = pd.to_datetime(stock_data['Date'], errors='coerce')

# Check for invalid dates and drop them
if stock_data['Date'].isnull().any():
    print("Some rows have invalid dates and will be dropped.")
stock_data = stock_data.dropna(subset=['Date'])

# Remove timezone information to make the Date column timezone-naive
stock_data['Date'] = stock_data['Date'].apply(lambda x: x.replace(tzinfo=None) if x.tzinfo else x)

# Define the date range for filtering
start_date = datetime(2024, 7, 1)
end_date = datetime(2024, 11, 30)

# Filter stock data within the date range
stock_data_filtered = stock_data[
    (stock_data['Date'] >= start_date) & (stock_data['Date'] <= end_date)
]

# Calculate percentage change in stock prices
stock_data_filtered['Pct_Change'] = stock_data_filtered['Close'].pct_change() * 100

# Plot stock prices
plt.figure(figsize=(18, 5))
plt.plot(stock_data_filtered['Date'], stock_data_filtered['Close'], label='Stock Price', color='blue')
plt.title('Bank of America Stock Prices (July 1 - November 30, 2024)')
plt.xlabel('Date')
plt.ylabel('Stock Price')

# Add dotted vertical lines on July 15 and November 5
plt.axvline(datetime(2024, 7, 15), color='red', linestyle=':', linewidth=2, label='Trump Nominated for 47th president')
plt.axvline(datetime(2024, 11, 5), color='black', linestyle=':', linewidth=2, label='Trump Elected as 47th presidentof U



# Add legends for the main plot and the vertical lines
plt.legend(loc='upper center', fontsize=12, frameon=True)

# Show the grid and the plot
```

```
plt.grid(True)
plt.show()

# Plot percentage change in stock price
plt.figure(figsize=(18, 5))
plt.plot(stock_data_filtered['Date'], stock_data_filtered['Pct_Change'], label='Percentage Change', color='green')
plt.title('Bank of America Stock Percentage Change (July 1 - November 30, 2024)')
plt.xlabel('Date')
plt.ylabel('Percentage Change')

# Add dotted vertical lines on July 15 and November 5
plt.axvline(datetime(2024, 7, 15), color='red', linestyle=':', linewidth=2,)
plt.axvline(datetime(2024, 11, 5), color='black', linestyle=':', linewidth=2)

plt.legend(loc='upper center', fontsize=12)
plt.grid(True)
plt.show()

# Display stock price statistics
print(f"Bank of America Stock Price Statistics (July 1 - November 30, 2024):")
print(stock_data_filtered[['Date', 'Close']].describe())
```

```
C:\Users\admin\AppData\Local\Temp\ipykernel_12840\1333240900.py:29: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  stock_data_filtered['Pct_Change'] = stock_data_filtered['Close'].pct_change() * 100
```






Bank of America Stock Price Statistics (July 1 - November 30, 2024):

| | Date | Close |
|---|---|---|
| count | 107 | 107.000000 |
| mean | 2024-09-14 14:21:18.504673024 | 41.065110 |
| min | 2024-07-01 00:00:00 | 36.211174 |
| 25% | 2024-08-07 12:00:00 | 39.195011 |
| 50% | 2024-09-16 00:00:00 | 40.410290 |
| 75% | 2024-10-22 12:00:00 | 42.095833 |
| max | 2024-11-29 00:00:00 | 47.505741 |
| std | NaN | 2.681320 |

# GDP Growth Rate

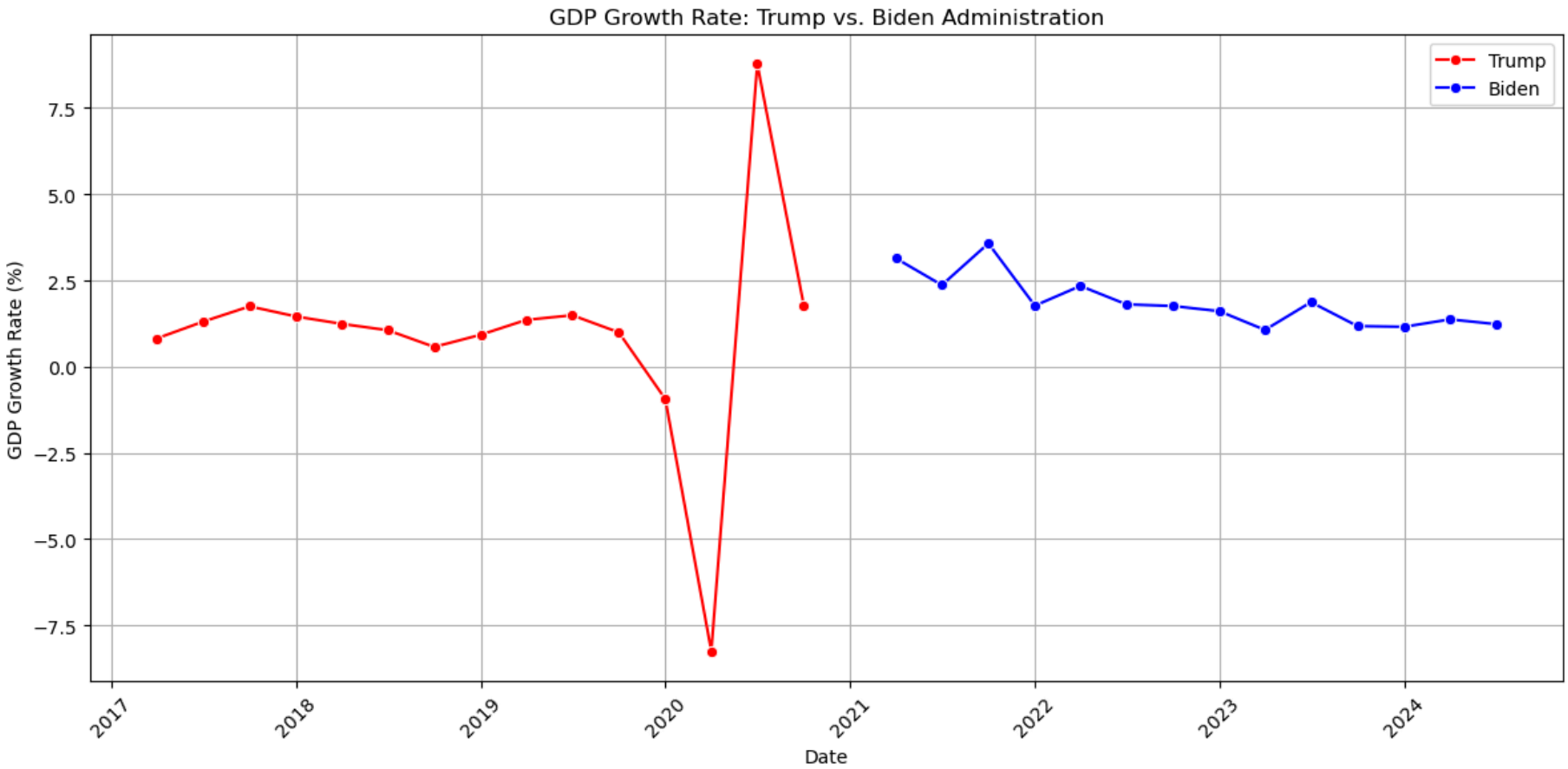
- Reference : https://fred.stlouisfed.org/series/GDP (https://fred.stlouisfed.org/series/GDP)

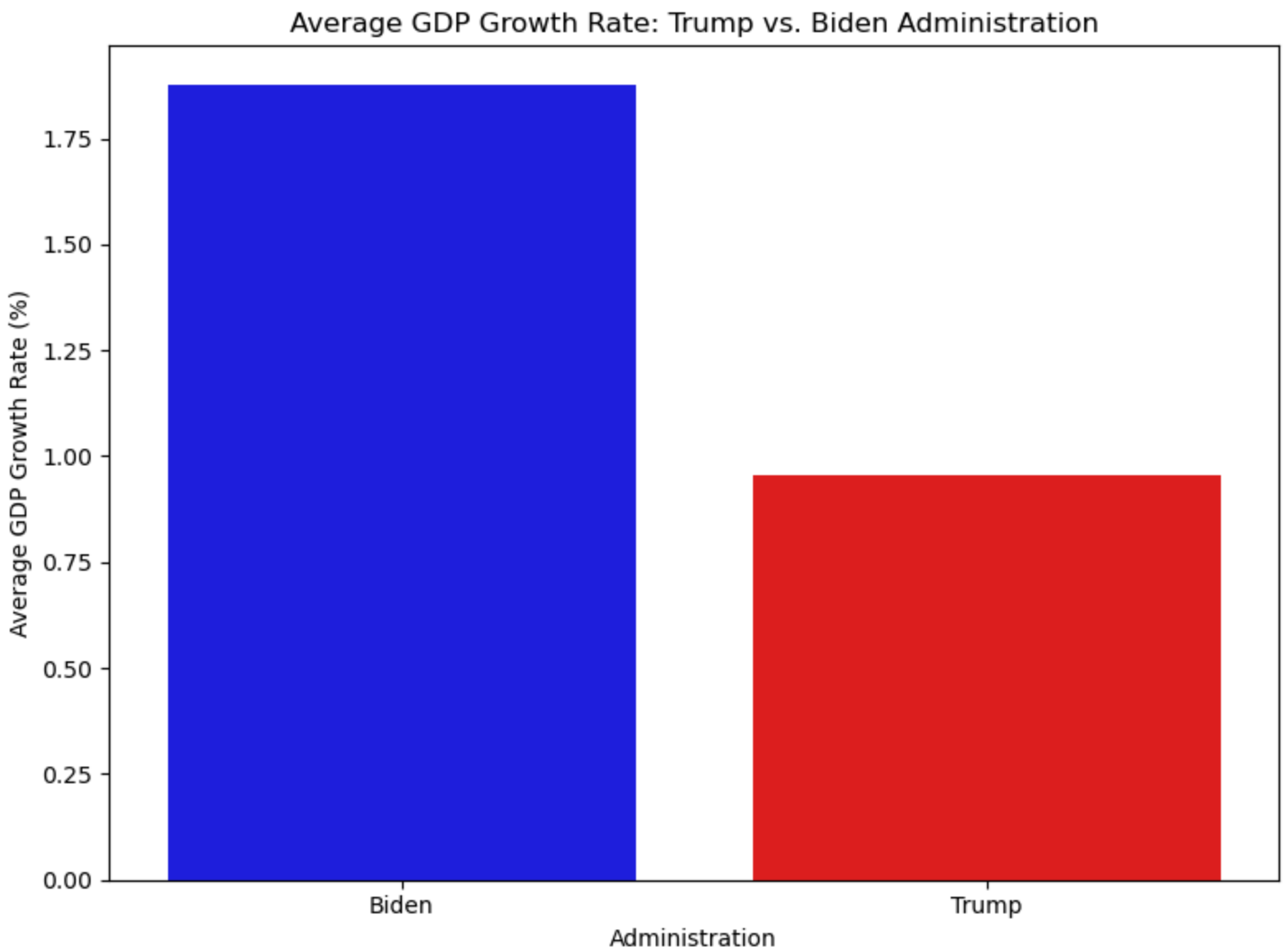
In [6]:

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

# Load the CSV files
trump_data = pd.read_csv('GDP_Trump_Administration.csv')
biden_data = pd.read_csv('GDP_Biden_Administration.csv')

# Display the first few rows to ensure data is loaded correctly
print("Trump Data:")
print(trump_data.head())
print("\nBiden Data:")
print(biden_data.head())

# Convert the 'Date' column to datetime for proper sorting
trump_data['Date'] = pd.to_datetime(trump_data['Date'])
biden_data['Date'] = pd.to_datetime(biden_data['Date'])

# Sort the data by date
trump_data.sort_values(by='Date', inplace=True)
biden_data.sort_values(by='Date', inplace=True)

# Calculate GDP Growth Rate
trump_data['GDP Growth Rate'] = trump_data['GDP'].pct_change() * 100
biden_data['GDP Growth Rate'] = biden_data['GDP'].pct_change() * 100

# Add a column to label the administration
trump_data['Administration'] = 'Trump'
biden_data['Administration'] = 'Biden'

# Combine the two datasets
combined_data = pd.concat([trump_data, biden_data])

# Drop rows with NaN GDP Growth Rate (first row of each dataset)
combined_data.dropna(subset=['GDP Growth Rate'], inplace=True)

# Plot the line chart for GDP growth over time
plt.figure(figsize=(12, 6))
sns.lineplot(data=combined_data[combined_data['Administration'] == 'Trump'],
             x='Date', y='GDP Growth Rate', label='Trump', marker='o', color='red')
sns.lineplot(data=combined_data[combined_data['Administration'] == 'Biden'],
             x='Date', y='GDP Growth Rate', label='Biden', marker='o', color='blue')
plt.title('GDP Growth Rate: Trump vs. Biden Administration')
plt.xlabel('Date')
plt.ylabel('GDP Growth Rate (%)')
plt.xticks(rotation=45)
plt.legend()
```

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

# Custom colors: Red for Trump, Blue for Biden
custom_colors = ['blue', 'red']

# Plot the bar chart with the specified colors
plt.figure(figsize=(8, 6))
sns.barplot(data=avg_growth, x='Administration', y='GDP Growth Rate', palette=custom_colors)
plt.title('Average GDP Growth Rate: Trump vs. Biden Administration')
plt.ylabel('Average GDP Growth Rate (%)')
plt.xlabel('Administration')
plt.tight_layout()
plt.show()
```

```
Trump Data:
         Date        GDP
0  2017-01-01  19280.084
1  2017-04-01  19438.643
2  2017-07-01  19692.595
3  2017-10-01  20037.088
4  2018-01-01  20328.553

Biden Data:
         Date        GDP
0  2021-01-01  22656.793
1  2021-04-01  23368.861
2  2021-07-01  23921.991
3  2021-10-01  24777.038
4  2022-01-01  25215.491
```


GDP Growth Rate: Trump vs. Biden Administration
Trump
Biden
GDP Growth Rate (%)
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
2017
2018
2019
2020
2021
2022
2023
2024
Date




## Unemployment Rate:

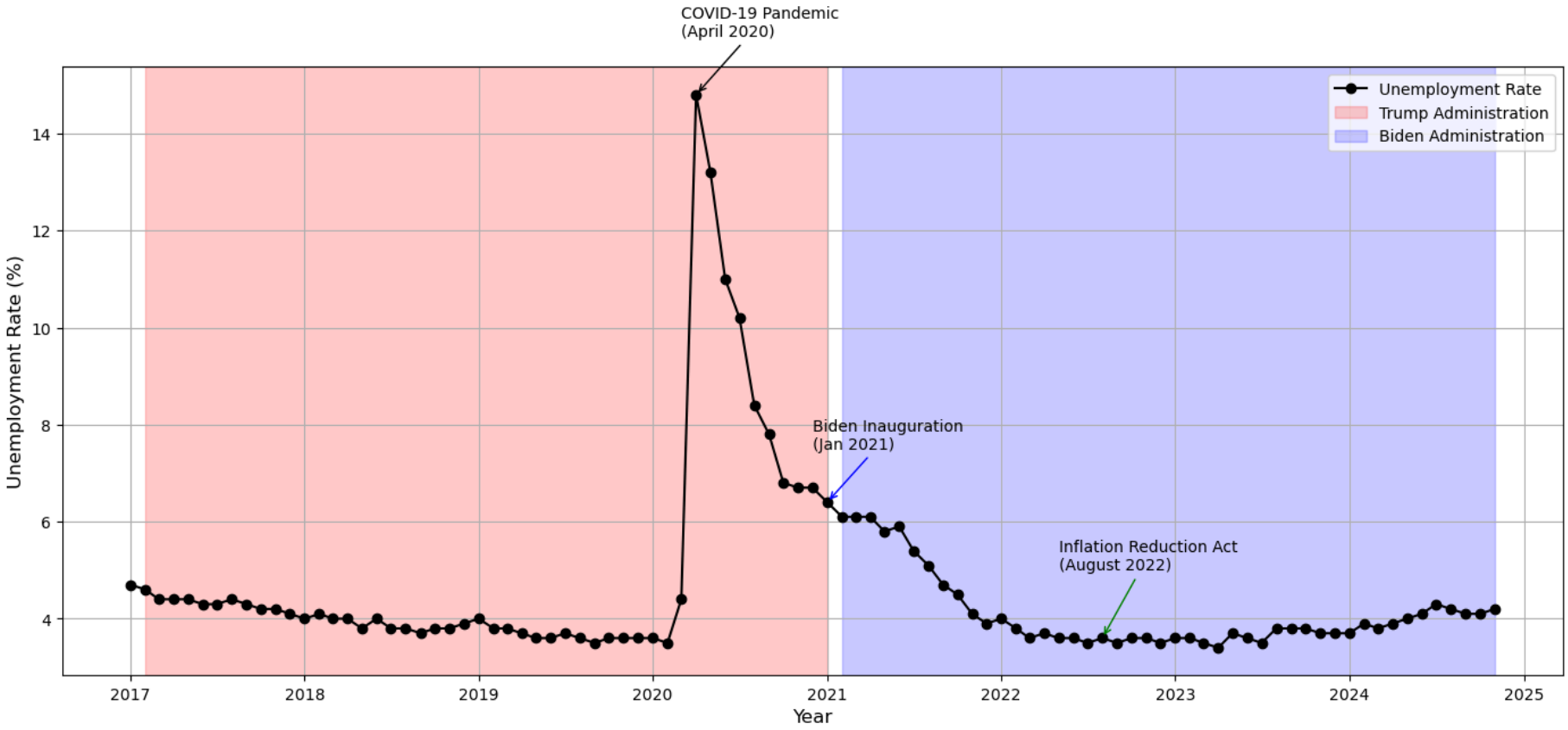
Reference : https://www.bls.gov/cps/ (https://www.bls.gov/cps/)

In [21]:

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

# Data for Unemployment Rates
data = {
    "Year-Month": [
        "2017-01", "2017-02", "2017-03", "2017-04", "2017-05", "2017-06", "2017-07", "2017-08", "2017-09", "2017-10", "
        "2018-01", "2018-02", "2018-03", "2018-04", "2018-05", "2018-06", "2018-07", "2018-08", "2018-09", "2018-10", "
        "2019-01", "2019-02", "2019-03", "2019-04", "2019-05", "2019-06", "2019-07", "2019-08", "2019-09", "2019-10", "
        "2020-01", "2020-02", "2020-03", "2020-04", "2020-05", "2020-06", "2020-07", "2020-08", "2020-09", "2020-10", "
        "2021-01", "2021-02", "2021-03", "2021-04", "2021-05", "2021-06", "2021-07", "2021-08", "2021-09", "2021-10", "
        "2022-01", "2022-02", "2022-03", "2022-04", "2022-05", "2022-06", "2022-07", "2022-08", "2022-09", "2022-10", "
        "2023-01", "2023-02", "2023-03", "2023-04", "2023-05", "2023-06", "2023-07", "2023-08", "2023-09", "2023-10", "
        "2024-01", "2024-02", "2024-03", "2024-04", "2024-05", "2024-06", "2024-07", "2024-08", "2024-09", "2024-10", "
    ],
    "Unemployment Rate (%)": [
        4.7, 4.6, 4.4, 4.4, 4.4, 4.3, 4.3, 4.4, 4.3, 4.2, 4.2, 4.1,
        4.0, 4.1, 4.0, 4.0, 3.8, 4.0, 3.8, 3.8, 3.7, 3.8, 3.8, 3.9,
        4.0, 3.8, 3.8, 3.7, 3.6, 3.6, 3.7, 3.6, 3.5, 3.6, 3.6, 3.6,
        3.6, 3.5, 4.4, 14.8, 13.2, 11.0, 10.2, 8.4, 7.8, 6.8, 6.7, 6.7,
        6.4, 6.1, 6.1, 6.1, 5.8, 5.9, 5.4, 5.1, 4.7, 4.5, 4.1, 3.9,
        4.0, 3.8, 3.6, 3.7, 3.6, 3.6, 3.5, 3.6, 3.5, 3.6, 3.6, 3.5,
        3.6, 3.6, 3.5, 3.4, 3.7, 3.6, 3.5, 3.8, 3.8, 3.8, 3.7, 3.7,
        3.7, 3.9, 3.8, 3.9, 4.0, 4.1, 4.3, 4.2, 4.1, 4.1, 4.2
    ]
}

# Create a DataFrame
df = pd.DataFrame(data)

# Convert Year-Month to datetime
df["Year-Month"] = pd.to_datetime(df["Year-Month"])

# Split data into Trump and Biden administrations
trump = df[(df["Year-Month"] >= "2017-01-20") & (df["Year-Month"] < "2021-01-20")]
biden = df[(df["Year-Month"] >= "2021-01-20")]

# Calculate average unemployment rates
trump_avg = trump["Unemployment Rate (%)"].mean()
biden_avg = biden["Unemployment Rate (%)"].mean()

print(f"Trump Administration Average Unemployment Rate: {trump_avg:.2f}%")
print(f"Biden Administration Average Unemployment Rate: {biden_avg:.2f}%")

# Plot the unemployment rate over time
plt.figure(figsize=(14, 7))
plt.plot(df["Year-Month"], df["Unemployment Rate (%)"], label="Unemployment Rate", color="black", marker='o')
```

```
# Add shaded areas for each administration
plt.axvspan(trump["Year-Month"].min(), trump["Year-Month"].max(), color="red", alpha=0.2, label="Trump Administration")
plt.axvspan(biden["Year-Month"].min(), biden["Year-Month"].max(), color="blue", alpha=0.2, label="Biden Administration"

# Annotations for key events
plt.annotate("COVID-19 Pandemic\n(April 2020)",
             xy=(pd.Timestamp("2020-04"), 14.8), xytext=(pd.Timestamp("2020-03"), 16),
             arrowprops=dict(arrowstyle="->", color="black"), fontsize=10, color="black")

plt.annotate("Biden Inauguration\n(Jan 2021)",
             xy=(pd.Timestamp("2021-01"), 6.4), xytext=(pd.Timestamp("2020-12"), 7.5),
             arrowprops=dict(arrowstyle="->", color="blue"), fontsize=10, color="black")

plt.annotate("Inflation Reduction Act\n(August 2022)",
             xy=(pd.Timestamp("2022-08"), 3.6), xytext=(pd.Timestamp("2022-05"), 5),
             arrowprops=dict(arrowstyle="->", color="green"), fontsize=10, color="black")

# Add the title under the plot
plt.subplots_adjust(bottom=0.15)  # Adjust space at the bottom
plt.suptitle("Unemployment Rate: Trump vs. Biden Administrations (With Key Events)", fontsize=16, y=0.008)

plt.xlabel("Year", fontsize=12)
plt.ylabel("Unemployment Rate (%)", fontsize=12)
plt.legend()
plt.grid()

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

```
Trump Administration Average Unemployment Rate: 5.04%
Biden Administration Average Unemployment Rate: 4.12%
```




Unemployment Rate: Trump vs. Biden Administrations (With Key Events)