Neural network and Granger causality applied to Presidential Odds

Riccardo Pondini

18 ottobre 2024

This study leverages the predictive power of financial market dynamics, exemplified by the S&P 500 index, alongside presidential odds to forecast the 2024 US Presidential Election outcomes. By employing advanced data analytics techniques and machine learning models, this research aims to uncover underlying patterns that might not be apparent through traditional polling methods.

Our research begins with a systematic collection of real-time data by using Selenium for web scraping. While traditional polls have been the standard in predicting political outcomes, betting markets offer a compelling alternative. Betting odds are determined by stakeholders who have a financial incentive to forecast outcomes accurately, thus they often incorporate a wider range of information.

A crucial aspect of our methodology is examining the potential causal relationships between financial markets and political outcomes. To this end, we employ Granger causality tests to determine whether historical data from the S&P 500 can predict fluctuations in the presidential election probabilities. Our analysis indicates that, with a lag of five periods (equivalent to five days), movements in the S&P 500 significantly precede and may predict changes in Donald Trump's election odds. Since the series are not cointegrated, this allows for the application of the Granger causality test on the differences of the data, focusing on short-term dynamics, as there is no long-term equilibrium requiring an error correction mechanism. This suggests that market performance could serve as a short-term leading indicator of shifts in political sentiment. However, while the Granger causality test establishes a temporal link, the modest correlation observed (0.12) between the lagged market changes and probability shifts indicates that this predictive relationship, while present, is relatively weak. Therefore, while financial markets provide some indication of political sentiment, other factors likely play a significant role in shaping electoral outcomes.

We employ Long Short-Term Memory (LSTM) networks to model the time-series data. LSTMs are particularly well-suited for processing sequential data over extended periods, as they can retain important time-dependent features that simpler models might overlook. In our model, we use a two-layer LSTM architecture with 50 units per layer, combined with dropout regularization to prevent overfitting. For simplicity, key parameters such as the number of units in each LSTM layer and the dropout rates are kept fixed and not optimized, allowing us to focus on capturing the core temporal dynamics in the data without unnecessary complexity.

The model's approach involves iterative forecasting, where predictions for each day are fed back into the model to predict the next day, continually extending the forecasting horizon until election day. This method enhances the accuracy and relevance of the forecasts by leveraging the cumulative knowledge of all preceding days, ensuring that the predictions are both responsive and precise as the election approaches. On the final days leading to the election, the trend indicates a declining probability for Donald Trump and an increasing probability for Kamala Harris as the election nears.

[201]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
import os
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
import chromedriver_autoinstaller
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By
import logging
import yfinance as yf
import matplotlib.dates as mdates
from matplotlib.dates import MO
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, LSTM
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from statsmodels.tsa.stattools import coint, grangercausalitytests
from datetime import timedelta
from datetime import datetime
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.layers import Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,␣
↪ReduceLROnPlateau
```

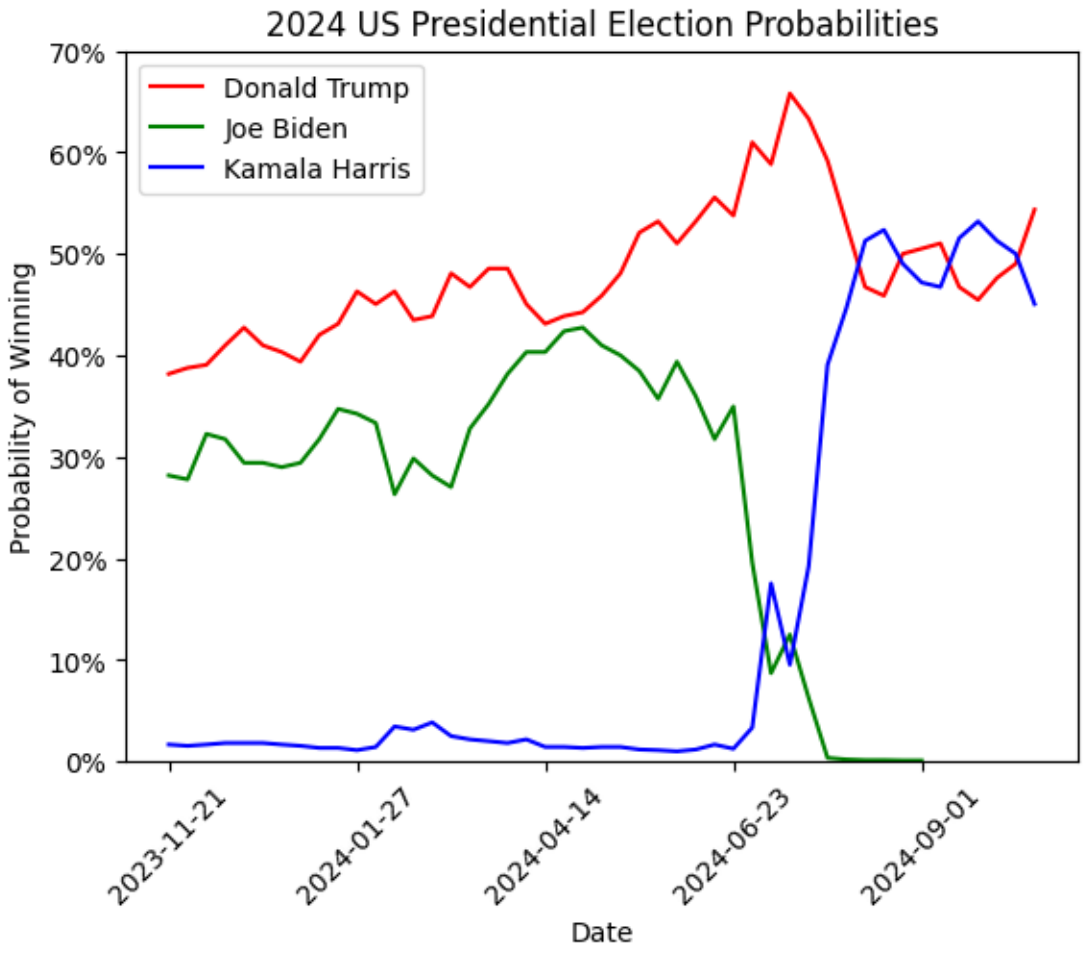
[190]:
```
# Initialize logging
logging.basicConfig(level=logging.CRITICAL)

# Install ChromeDriver
path_to_chromedriver = chromedriver_autoinstaller.install()

# Configure download folder
home_directory = os.path.expanduser("~")
desktop_path = os.path.join(home_directory, 'Desktop')
new_folder_name = 'Raw dataset'
download_folder = os.path.join(desktop_path, new_folder_name)
os.makedirs(download_folder, exist_ok=True)

# Configure WebDriver
options = webdriver.ChromeOptions()
options.add_experimental_option("prefs", {
    "download.default_directory": download_folder,
```

```
    "download.prompt_for_download": False,
    "download.directory_upgrade": True,
    "safebrowsing.enabled": True
})
service = Service(path_to_chromedriver)
driver = webdriver.Chrome(service=service, options=options)

try:
    driver.get("https://www.kaggle.com/code/jdaustralia/
↪2024-us-presidential-race-probabilities/input")
    download_button = WebDriverWait(driver, 10).until(
        EC.element_to_be_clickable((By.XPATH, "//span[@aria-hidden='true' and␣
↪text()='get_app']"))
    )
    driver.execute_script("arguments[0].scrollIntoView();", download_button)
    download_button.click()

    sign_in_button = WebDriverWait(driver, 10).until(
        EC.element_to_be_clickable((By.XPATH, "//span[contains(text(), 'Sign in␣
↪with Email')]"))
    )
    sign_in_button.click()

    # Wait for the input fields to be ready
    email_field = WebDriverWait(driver, 20).until(
        EC.presence_of_element_located((By.ID, ":r0:"))
    )
    email_field.clear()
    email_field.send_keys("Replace with your Kaggle email")
    password_field = WebDriverWait(driver, 20).until(
        EC.presence_of_element_located((By.ID, ":r1:"))
    )
    password_field.clear()
    password_field.send_keys("Replace with your Kaggle password")

    # Submit the form
    submit_button = driver.find_element(By.CSS_SELECTOR, "button[type='submit']")
    submit_button.click()

    # Wait for download button to be clickable again
    download_button = WebDriverWait(driver, 10).until(
        EC.element_to_be_clickable((By.XPATH, "//span[@aria-hidden='true' and␣
↪text()='get_app']"))
    )
    driver.execute_script("arguments[0].scrollIntoView({behavior: 'smooth',␣
↪block: 'center'});", download_button)
```

```
    time.sleep(5)
    download_button.click()

    # Wait for the download to complete
    time.sleep(5)  # Increase time if necessary
finally:
    driver.quit()
```

[192]:
```
file_path = os.path.join(download_folder, '2024ElectionOdds.csv')

# Converting from odds to probabilities
odds = pd.read_csv(file_path, index_col=0)
probs = 1 / odds

# Define a maximum limit for the sum of probabilities
max_sum = 1

# Function to recalibrate probabilities if their sum exceeds max_sum
def recalibrate_probabilities(row, max_sum):
    total = row.sum()
    if total > max_sum:
        # If the sum of probabilities exceeds max_sum, scale the probabilities
        row = row * (max_sum / total)
    return row

# Apply recalibration for each row that exceeds a sum limit of 1.1
probs[['Donald Trump', 'Joe Biden', 'Kamala Harris']] = probs[['Donald Trump',␣
↪'Joe Biden', 'Kamala Harris']].apply(lambda row:␣
↪recalibrate_probabilities(row, max_sum), axis=1)

# Plotting the top three probabilities
top_three = probs[['Donald Trump', 'Joe Biden','Kamala Harris']]
top_three.index = pd.to_datetime(top_three.index, dayfirst=True)
top_three.index = top_three.index.strftime('%Y-%m-%d')

# Ensure the 'date' column is the correct type
print(top_three.describe())
top_three.plot(kind='line', color=['red', 'green', 'blue'])
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter(1))
plt.ylim(0, 0.7)

print("")
plt.title('2024 US Presidential Election Probabilities')
plt.xticks(rotation=45)
plt.xlabel('Date')
plt.ylabel('Probability of Winning')
plt.show()
```

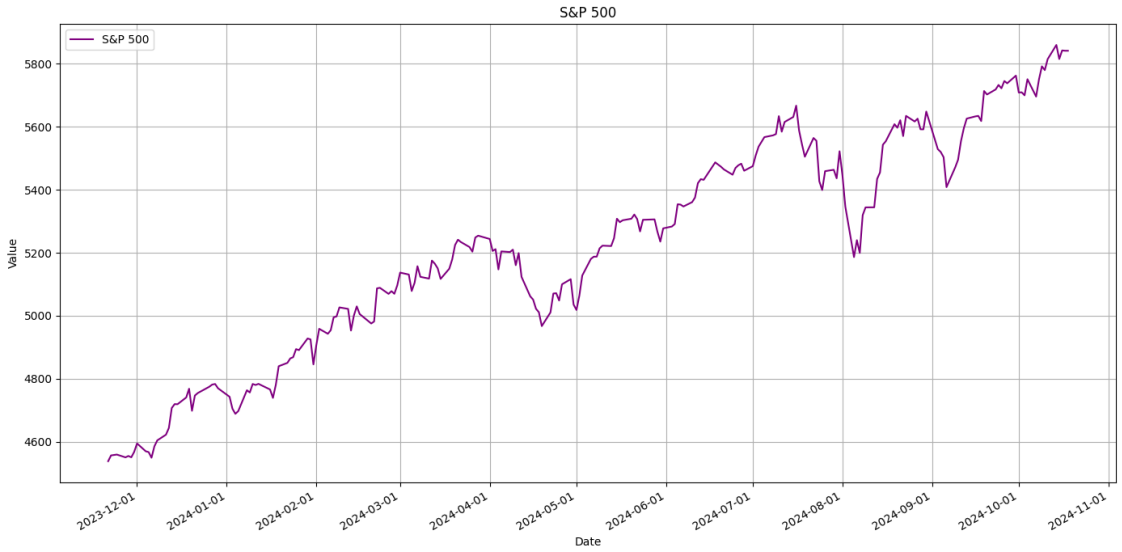
	Donald Trump	Joe Biden	Kamala Harris
count	47.000000	41.000000	47.000000
mean	0.480775	0.268241	0.145789
std	0.064835	0.137500	0.204541
min	0.381679	0.001000	0.010000
25%	0.436690	0.263158	0.014286
50%	0.467290	0.317460	0.018182
75%	0.515519	0.357143	0.291466
max	0.657895	0.427350	0.531915

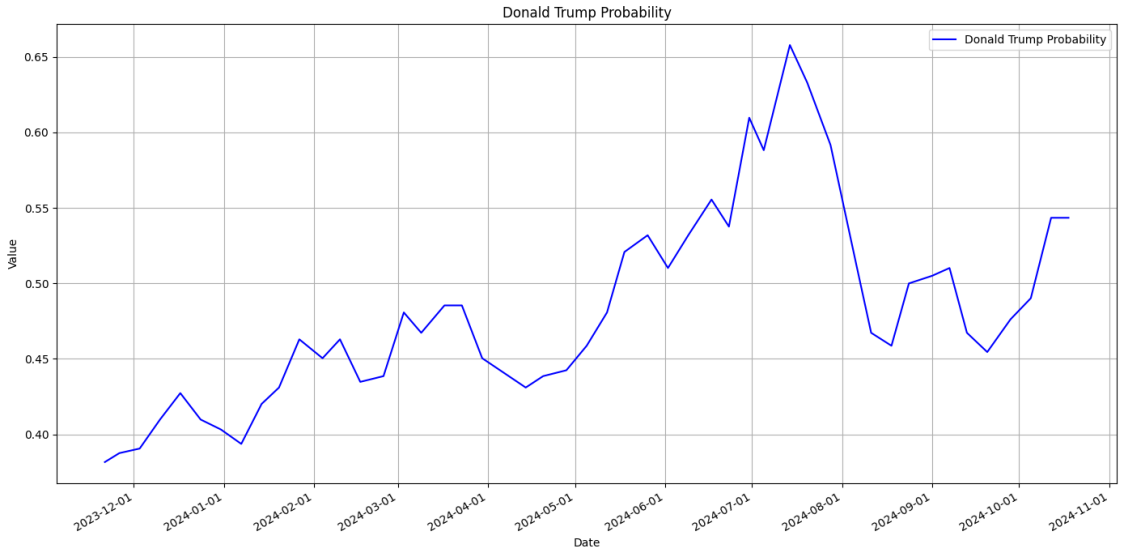


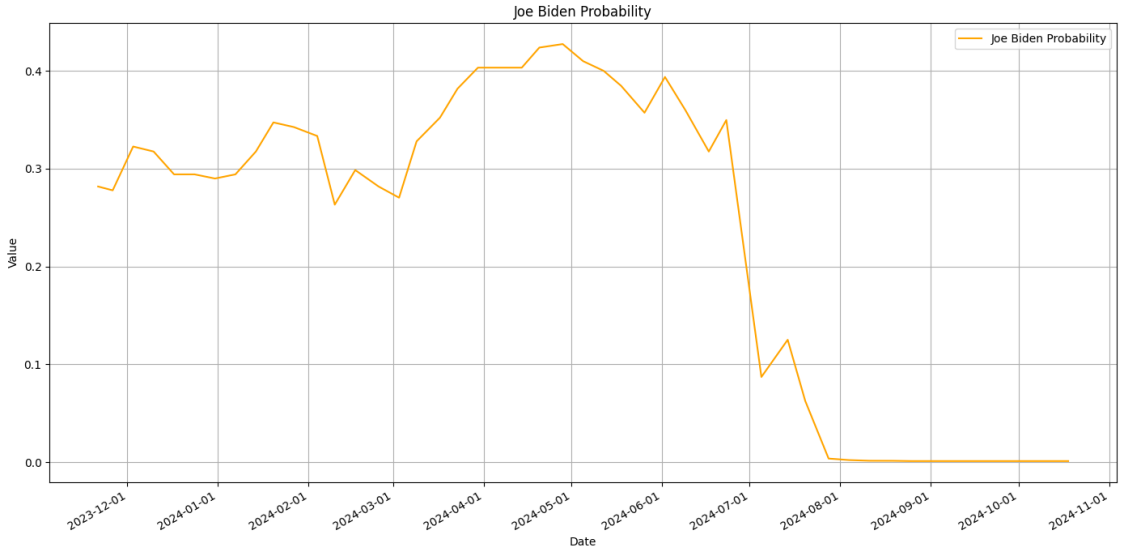
[193]:
```
# Define the start and end dates
start_date = "2023-11-21"
end_date = top_three_full.index[-1].strftime('%Y-%m-%d')

# Download historical S&P 500 data
sp500_data = yf.download("^GSPC", start=start_date, end=end_date, interval="1d")
sp500_data['SP500'] = sp500_data['Adj Close']

# Create a new index that includes every day from start to end date
all_dates = pd.date_range(start=start_date, end=end_date, freq='D')

# Realign the DataFrame to include all dates
sp500_full = sp500_data.reindex(all_dates)

# Fill missing data for S&P 500
sp500_full['SP500'] = sp500_full['SP500'].interpolate()

# Prepare the top_three DataFrame
top_three.index = pd.to_datetime(top_three.index)
top_three_full = top_three.reindex(all_dates)

# Fill missing data for top_three
columns_to_fill = ['Donald Trump', 'Joe Biden', 'Kamala Harris']
for column in columns_to_fill:
    top_three_full[column] = top_three_full[column].interpolate()

# Merge the DataFrames
merged_df = pd.merge(top_three_full, sp500_full[['SP500']], left_index=True,␣
 ↪right_index=True, how='inner')

# Graphs
graphs = [
    {'column': 'SP500', 'color': 'purple', 'title': 'S&P 500'},
    {'column': 'Donald Trump', 'color': 'blue', 'title': 'Donald Trump␣
 ↪Probability'},
    {'column': 'Joe Biden', 'color': 'orange', 'title': 'Joe Biden Probability'},
    {'column': 'Kamala Harris', 'color': 'green', 'title': 'Kamala Harris␣
 ↪Probability'}
]

for graph in graphs:
    plt.figure(figsize=(14, 7))
    plt.plot(merged_df.index, merged_df[graph['column']], label=graph['title'],␣
 ↪color=graph['color'])
    plt.title(graph['title'])
    plt.xlabel('Date')
    plt.ylabel('Value')
```

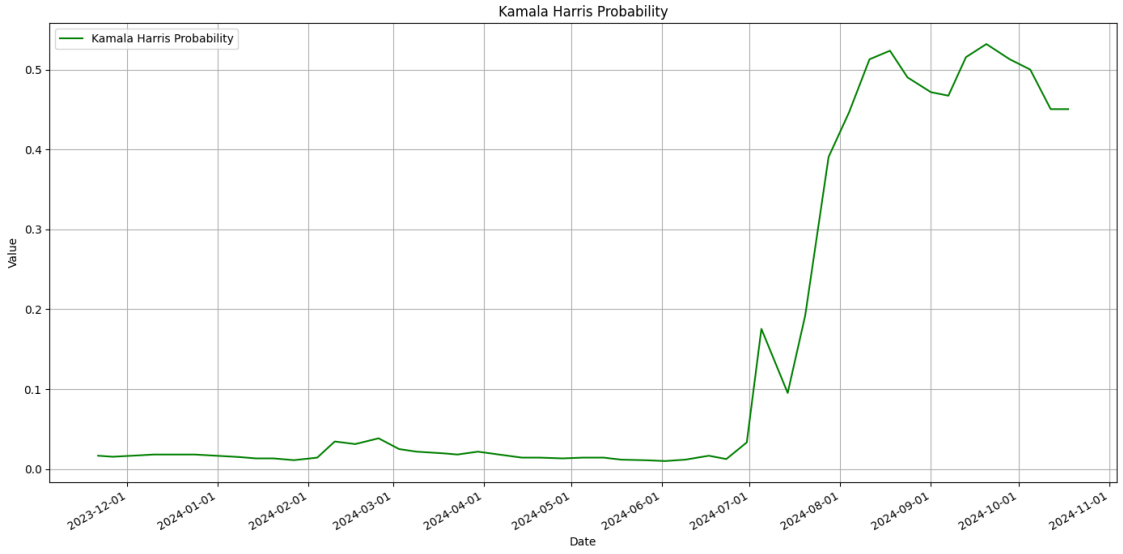
```
    plt.legend()
    plt.grid(True)
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
    plt.gca().xaxis.set_major_locator(mdates.MonthLocator())
    plt.gcf().autofmt_xdate()
    plt.tight_layout()
    plt.show()
```

```
[**********100%%**********]  1 of 1 completed
```









[194]:
```
# Perform cointegration test (Engle-Granger) between SP500 and Trump's␣
↪probabilities
coint_stat, p_value, crit_value = coint(merged_df['SP500'], merged_df['Donald␣
↪Trump'])
print(f"Engle-Granger Cointegration Test P-value: {p_value}")

# If p_value is less than 0.05, the series are cointegrated
if p_value < 0.05:
    print("The series are cointegrated. Proceeding with VECM for Granger␣
↪Causality test.")
```

```
else:
    print("The series are not cointegrated. Proceeding with differencing the␣
 ↪series.")

# Initialize the scaler
scaler = MinMaxScaler()

# Granger Causality Test
# Standardize both series
scaler = MinMaxScaler()
merged_df[['SP500', 'Donald Trump']] = scaler.fit_transform(merged_df[['SP500',␣
 ↪'Donald Trump']])

# Maximum number of lags to test for Granger Causality
max_lag = 5

# If the series are cointegrated, we can proceed with VECM; otherwise, we␣
 ↪perform Granger Causality test directly
if p_value < 0.05:
    # In case of cointegration, a VECM would be more appropriate, but we␣
 ↪directly perform Granger test on levels for simplicity
    grangercausalitytests(merged_df[['SP500', 'Donald Trump']], max_lag)
else:
    # If not cointegrated, difference the series before running Granger␣
 ↪Causality test
    merged_df_diff = merged_df.diff().dropna()

    # Granger Causality Test for Donald Trump and the S&P 500
    print("\nGranger Causality Test for Trump and S&P 500:")
    grangercausalitytests(merged_df_diff[['SP500', 'Donald Trump']], max_lag)

    # Reverse Granger Causality Test: S&P 500 influencing Trump
    print("\nGranger Causality Test: S&P 500 causing Trump:")
    grangercausalitytests(merged_df_diff[['Donald Trump', 'SP500']], max_lag)

# Create a lagged version of S&P 500 with a lag of 5 days
merged_df_diff['SP500_lagged'] = merged_df_diff['SP500'].shift(5)

# Drop rows with NaN values due to the lag
lagged_df = merged_df_diff.dropna()

# Calculate the correlation between lagged S&P 500 and Trump's probability
correlation = lagged_df['SP500_lagged'].corr(lagged_df['Donald Trump'])
print(f"\nCorrelation between lagged Delta S&P 500 (5 days) and Delta Trump's␣
 ↪election probability: {correlation:.4f}")
```

```python
# Fisher transformation
z = 0.5 * np.log((1 + correlation) / (1 - correlation))

# Number of observations
n = len(lagged_df)
# Standard error of Z
se_z = 1 / np.sqrt(n - 3)

# Calculate the 95% confidence interval for Z
z_conf_interval = (z - 1.96 * se_z, z + 1.96 * se_z)

# Reverse the Fisher transformation to get the confidence interval for r
r_conf_interval = ((np.exp(2 * z_conf_interval[0]) - 1) / (np.exp(2 *␣
 ↪z_conf_interval[0]) + 1),
                   (np.exp(2 * z_conf_interval[1]) - 1) / (np.exp(2 *␣
 ↪z_conf_interval[1]) + 1))

# Print the results
print(f"\nCorrelation between lagged Delta S&P 500 (5 days) and Delta Trump's␣
 ↪election probability: {correlation:.4f}")
print(f"95% Confidence Interval: {r_conf_interval}")
```

```
Engle-Granger Cointegration Test P-value: 0.7688803433830244
The series are not cointegrated. Proceeding with differencing the series.

Granger Causality Test for Trump and S&P 500:

Granger Causality
number of lags (no zero) 1
ssr based F test:         F=3.1896  , p=0.0750  , df_denom=323, df_num=1
ssr based chi2 test:   chi2=3.2192  , p=0.0728  , df=1
likelihood ratio test: chi2=3.2034  , p=0.0735  , df=1
parameter F test:         F=3.1896  , p=0.0750  , df_denom=323, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=1.6765  , p=0.1887  , df_denom=320, df_num=2
ssr based chi2 test:   chi2=3.4054  , p=0.1822  , df=2
likelihood ratio test: chi2=3.3877  , p=0.1838  , df=2
parameter F test:         F=1.6765  , p=0.1887  , df_denom=320, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=2.1749  , p=0.0909  , df_denom=317, df_num=3
ssr based chi2 test:   chi2=6.6687  , p=0.0832  , df=3
```

```
likelihood ratio test: chi2=6.6010  , p=0.0858  , df=3
parameter F test:         F=2.1749  , p=0.0909  , df_denom=317, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=1.7372  , p=0.1415  , df_denom=314, df_num=4
ssr based chi2 test:   chi2=7.1480  , p=0.1283  , df=4
likelihood ratio test: chi2=7.0701  , p=0.1322  , df=4
parameter F test:         F=1.7372  , p=0.1415  , df_denom=314, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test:         F=1.3693  , p=0.2356  , df_denom=311, df_num=5
ssr based chi2 test:   chi2=7.0889  , p=0.2141  , df=5
likelihood ratio test: chi2=7.0119  , p=0.2198  , df=5
parameter F test:         F=1.3693  , p=0.2356  , df_denom=311, df_num=5



Granger Causality Test: S&P 500 causing Trump:

Granger Causality
number of lags (no zero) 1
ssr based F test:         F=0.0169  , p=0.8967  , df_denom=323, df_num=1
ssr based chi2 test:   chi2=0.0170  , p=0.8961  , df=1
likelihood ratio test: chi2=0.0170  , p=0.8961  , df=1
parameter F test:         F=0.0169  , p=0.8967  , df_denom=323, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=0.6991  , p=0.4978  , df_denom=320, df_num=2
ssr based chi2 test:   chi2=1.4200  , p=0.4916  , df=2
likelihood ratio test: chi2=1.4169  , p=0.4924  , df=2
parameter F test:         F=0.6991  , p=0.4978  , df_denom=320, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=1.4626  , p=0.2247  , df_denom=317, df_num=3
ssr based chi2 test:   chi2=4.4845  , p=0.2137  , df=3
likelihood ratio test: chi2=4.4538  , p=0.2164  , df=3
parameter F test:         F=1.4626  , p=0.2247  , df_denom=317, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=3.6079  , p=0.0068  , df_denom=314, df_num=4
ssr based chi2 test:   chi2=14.8454 , p=0.0050  , df=4
likelihood ratio test: chi2=14.5143 , p=0.0058  , df=4
parameter F test:         F=3.6079  , p=0.0068  , df_denom=314, df_num=4
```

```
Granger Causality
number of lags (no zero) 5
ssr based F test:         F=3.6120  , p=0.0034  , df_denom=311, df_num=5
ssr based chi2 test:   chi2=18.6987 , p=0.0022  , df=5
likelihood ratio test: chi2=18.1759 , p=0.0027  , df=5
parameter F test:         F=3.6120  , p=0.0034  , df_denom=311, df_num=5


Correlation between lagged Delta S&P 500 (5 days) and Delta Trump's election
probability: 0.1234
95% Confidence Interval: (0.014289457288261247, 0.22960106892737753)
```

[197]:
```python
# Function to create a dataset with time series sequences
def create_dataset(data, time_step=1):
    dataX, dataY = [], []
    # Loop to create sequences of data with the specified time steps
    for i in range(len(data) - time_step - 1):
        # Extract a sequence of 'time_step' length
        a = data[i:(i + time_step)]
        dataX.append(a)
        # The target is the value right after the current sequence
        dataY.append(data[i + time_step])
    return np.array(dataX), np.array(dataY)

# Preprocessing and separate normalization for each time series
def prepare_data(series, time_step):
    # Initialize the MinMaxScaler to scale the data between 0 and 1
    scaler = MinMaxScaler(feature_range=(0, 1))

    # Reshape the data to fit the scaler, then scale the data
    scaled_data = scaler.fit_transform(series.values.reshape(-1, 1))

    # Create sequences of data and corresponding target values
    X, y = create_dataset(scaled_data, time_step)

    # Reshape X to fit the LSTM model's expected input format (samples, time␣
 ↪steps, features)
    X = X.reshape(X.shape[0], X.shape[1], 1)

    return X, y, scaler

# Number of time steps for sequence generation
time_step = 30

# Prepare data for Donald Trump
X_trump, y_trump, scaler_trump = prepare_data(probs['Donald Trump'], time_step)
```

```
# Prepare data for Kamala Harris
X_harris, y_harris, scaler_harris = prepare_data(probs['Kamala Harris'],␣
↪time_step)

# Function to split data into training and test sets
def split_data(X, y, test_size=0.2):
    # Use train_test_split to split the dataset, with 20% of data for testing␣
↪and 80% for training
    return train_test_split(X, y, test_size=test_size, random_state=42)

# Split data for Donald Trump into training and test sets
X_train_trump, X_test_trump, y_train_trump, y_test_trump = split_data(X_trump,␣
↪y_trump)

# Split data for Kamala Harris into training and test sets
X_train_harris, X_test_harris, y_train_harris, y_test_harris =␣
↪split_data(X_harris, y_harris)

# Function to build an LSTM model for each series
def create_lstm_model(time_step):
    model = Sequential()
    model.add(Input(shape=(time_step, 1)))  # Input shape: (time steps, 1␣
↪feature)
    model.add(LSTM(50, return_sequences=True))  # First LSTM layer with 50␣
↪units, returns sequences
    model.add(Dropout(0.2))  # Dropout for regularization
    model.add(LSTM(50))  # Second LSTM layer with 50 units, returns the final␣
↪state
    model.add(Dropout(0.2))  # Dropout for regularization
    model.add(Dense(1))  # Output layer with 1 unit, predicting a single value
    model.compile(optimizer='adam', loss='mean_squared_error')  # Compile the␣
↪model with Adam optimizer and MSE loss
    return model

# Create the LSTM models for Donald Trump and Kamala Harris
time_step = 30
model_trump = create_lstm_model(time_step)
model_harris = create_lstm_model(time_step)
```

[203]:
```
# Define callbacks for each candidate
callbacks_trump = [
    EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    ModelCheckpoint('model_trump.keras', monitor='val_loss',␣
↪save_best_only=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5)
]
```

```
callbacks_harris = [
    EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    ModelCheckpoint('model_harris.keras', monitor='val_loss',␣
 ↪save_best_only=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5)
]

# Training the model for Donald Trump
model_trump.fit(X_trump, y_trump, epochs=50, batch_size=64, validation_split=0.2,
                callbacks=callbacks_trump)
# Training the model for Kamala Harris
model_harris.fit(X_harris, y_harris, epochs=50, batch_size=64,␣
 ↪validation_split=0.2,
                callbacks=callbacks_harris)
```

[206]:
```
# Define the number of days to predict (up to November 5, 2024)
last_date_in_data = pd.to_datetime(top_three_full.index[-1])
end_date = pd.to_datetime("2024-11-05")

# Check if the last available date is earlier than November 5, 2024
if last_date_in_data >= end_date:
    raise ValueError(f"The last available date ({last_date_in_data}) is on or␣
 ↪after November 5, 2024.")

# Calculate the number of days to predict
days_to_predict = (end_date - last_date_in_data).days

# Iterative forecasting for Donald Trump
predicted_trump = []

# Get the last available sequence for Trump
last_sequence_trump = X_trump[-1].reshape(1, time_step, 1)

for _ in range(days_to_predict):
    # Predict the next day
    next_pred_trump = model_trump.predict(last_sequence_trump)

    # Denormalize the prediction to the original scale
    next_pred_trump_denorm = scaler_trump.
 ↪inverse_transform(next_pred_trump)[0][0]
    predicted_trump.append(next_pred_trump_denorm)

    # Update the sequence: remove the oldest day and add the new prediction
    next_pred_trump_reshaped = np.array(next_pred_trump).reshape(1, 1, 1)  #␣
 ↪Ensure 3D shape
```

```
    last_sequence_trump = np.append(last_sequence_trump[:, 1:, :],␣
↪next_pred_trump_reshaped, axis=1)

# Iterative forecasting for Kamala Harris
predicted_harris = []
last_sequence_harris = X_harris[-1].reshape(1, time_step, 1)

for _ in range(days_to_predict):
    next_pred_harris = model_harris.predict(last_sequence_harris)
    next_pred_harris_denorm = scaler_harris.
↪inverse_transform(next_pred_harris)[0][0]
    predicted_harris.append(next_pred_harris_denorm)
    next_pred_harris_reshaped = np.array(next_pred_harris).reshape(1, 1, 1)  #␣
↪Ensure correct 3D shape
    last_sequence_harris = np.append(last_sequence_harris[:, 1:, :],␣
↪next_pred_harris_reshaped, axis=1)

# Create a time series with the predicted dates and forecasted values
prediction_dates = pd.date_range(start=last_date_in_data + timedelta(days=1),␣
↪periods=days_to_predict)

# Create a DataFrame with the forecasted probabilities for each candidate
predictions_df = pd.DataFrame({
    'Date': prediction_dates,
    'Donald Trump': predicted_trump,
    'Kamala Harris': predicted_harris
})

# Set the 'Date' column as the index
predictions_df.set_index('Date', inplace=True)

# Define a function to recalibrate the probabilities so that they sum to 1
def recalibrate_probabilities(row):
    total = row.sum()
    if total > 0:
        row = row / total  # Normalize probabilities to sum to 1
    return row

# Apply the recalibration function to ensure the sum of probabilities is 1 for␣
↪each day
predictions_df[['Donald Trump', 'Kamala Harris']] = predictions_df[['Donald␣
↪Trump', 'Kamala Harris']].apply(recalibrate_probabilities, axis=1)

# Print the recalibrated predictions
print(predictions_df)
```

Date	Donald Trump	Kamala Harris
2024-10-19	0.424602	0.575398
2024-10-20	0.410687	0.589313
2024-10-21	0.396990	0.603010
2024-10-22	0.383791	0.616209
2024-10-23	0.371337	0.628663
2024-10-24	0.359784	0.640216
2024-10-25	0.349186	0.650814
2024-10-26	0.339545	0.660455
2024-10-27	0.330844	0.669156
2024-10-28	0.323037	0.676963
2024-10-29	0.316069	0.683931
2024-10-30	0.309872	0.690128
2024-10-31	0.304393	0.695607
2024-11-01	0.299582	0.700418
2024-11-02	0.295351	0.704649
2024-11-03	0.291640	0.708360
2024-11-04	0.288407	0.711593
2024-11-05	0.285558	0.714442