Economics and Trading Indicators Research

September 9, 2024

1 Connor O'Keefe - Wealth Asset Management Intern

2 09/09/2024

Regarding all the API keys, just run the code in chronological order or create new keys to replicate these results. The aggregated_data excel sheet has been uploaded under my intern folder on the drive.

3 All Libraries Used Throughout

[120]:
```
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
import numpy as np
import matplotlib.colors as mcolors
import matplotlib.cm as cm
import pyfredapi as fred
from fredapi import Fred
from sklearn.preprocessing import MinMaxScaler
```

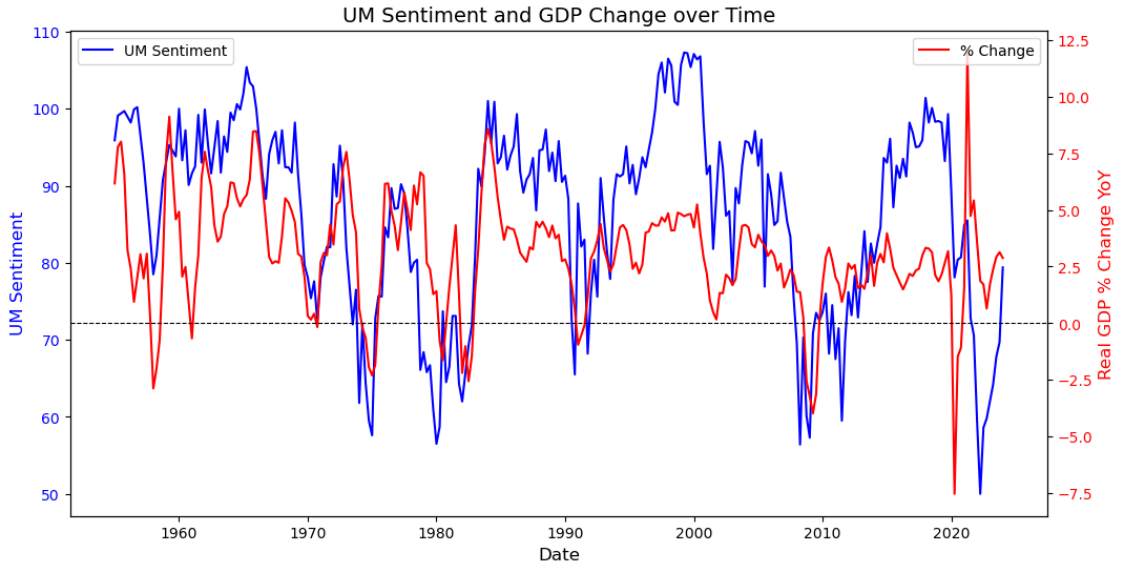
[121]:
```
df = pd.read_excel('aggregated data.xlsx')
df
```

[121]:

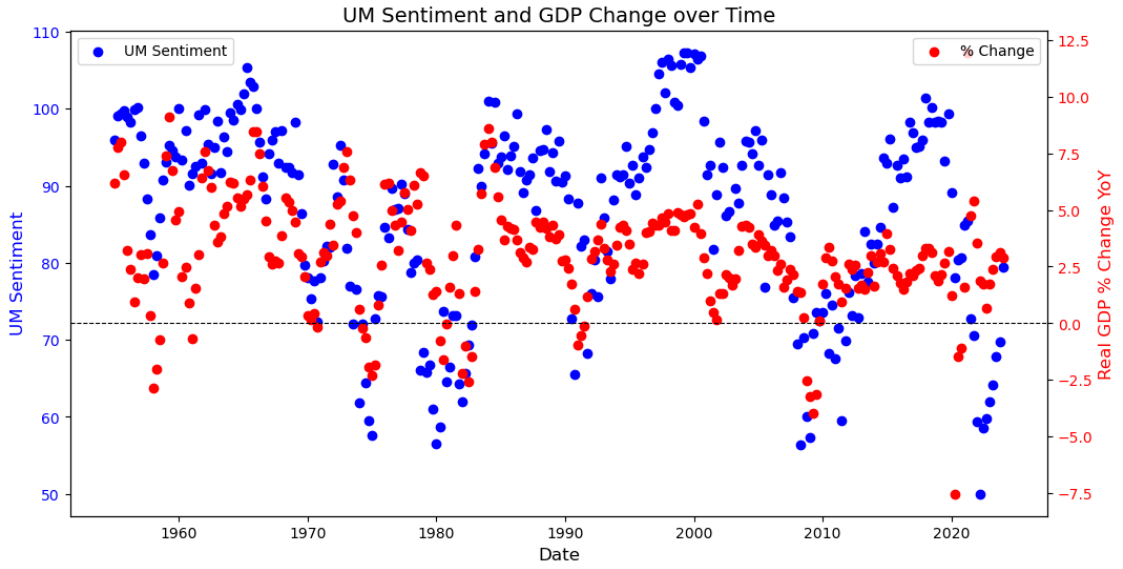
	DATE	UM Sentiment	DATE.1	% Change	DATE.2	Fed Funds Rate
0	1955-01-01	95.90	1955-01-01	6.17020	1955-01-01	1.35
1	1955-04-01	99.10	1955-04-01	7.78024	1955-04-01	1.64
2	1955-07-01	99.40	1955-07-01	8.01592	1955-07-01	2.18
3	1955-10-01	99.70	1955-10-01	6.57902	1955-10-01	2.48
4	1956-01-01	98.95	1956-01-01	3.21695	1956-01-01	2.50
..	…	…	…	…	…	…
272	2023-01-01	62.00	2023-01-01	1.71793	2023-01-01	4.65
273	2023-04-01	64.20	2023-04-01	2.38247	2023-04-01	5.08
274	2023-07-01	67.80	2023-07-01	2.92689	2023-07-01	5.33
275	2023-10-01	69.70	2023-10-01	3.13449	2023-10-01	5.33
276	2024-01-01	79.40	2024-01-01	2.88308	2024-01-01	5.33

[277 rows x 6 columns]

[122]:
```
# Convert DATE columns to datetime
df['DATE'] = pd.to_datetime(df['DATE'])
df['DATE.1'] = pd.to_datetime(df['DATE.1'])

# Drop rows with NaNs in either set of columns
df1 = df[['DATE', 'UM Sentiment']].dropna()
df2 = df[['DATE.1', '% Change']].dropna()

# Plotting
fig, ax1 = plt.subplots(figsize=(12, 6))

# Plot the first data series
ax1.plot(df1['DATE'], df1['UM Sentiment'], 'b-', label='UM Sentiment')
ax1.set_xlabel('Date', fontsize = 12)
ax1.set_ylabel('UM Sentiment', color='b', fontsize = 12)
ax1.tick_params(axis='y', labelcolor='b')

# Create a second y-axis
ax2 = ax1.twinx()
ax2.plot(df2['DATE.1'], df2['% Change'], 'r-', label='% Change')
ax2.set_ylabel('Real GDP % Change YoY', color='r', fontsize = 12)
ax2.tick_params(axis='y', labelcolor='r')
ax2.axhline(0, color='black', linestyle='--', linewidth=0.8)

# Add legends
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

# Title
plt.title('UM Sentiment and GDP Change over Time', fontsize = 14)

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

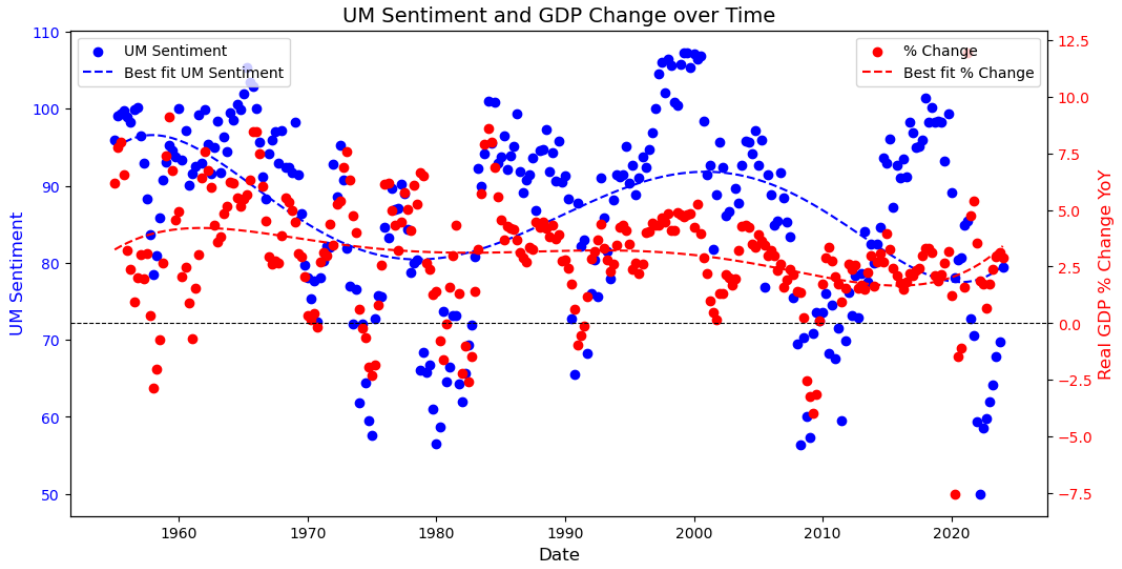


[123]:
```
# Plotting
fig, ax1 = plt.subplots(figsize=(12, 6))

# Plot the first data series as scatter plot
ax1.scatter(df1['DATE'], df1['UM Sentiment'], color='b', label='UM Sentiment')
ax1.set_xlabel('Date', fontsize = 12)
ax1.set_ylabel('UM Sentiment', color='b', fontsize = 12)
ax1.tick_params(axis='y', labelcolor='b')

# Create a second y-axis
ax2 = ax1.twinx()
ax2.scatter(df2['DATE.1'], df2['% Change'], color='r', label='% Change')
ax2.set_ylabel('Real GDP % Change YoY', color='r', fontsize = 12)
ax2.tick_params(axis='y', labelcolor='r')
ax2.axhline(0, color='black', linestyle='--', linewidth=0.8)

# Add legends
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

# Title
plt.title('UM Sentiment and GDP Change over Time', fontsize = 14)

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



[124]:
```
# Fit a higher order polynomial (degree 5) to the first data series
coeffs1 = np.polyfit(df1['DATE'].astype(np.int64) // 10**9, df1['UM␣
 ↪Sentiment'], 5)
poly1 = np.poly1d(coeffs1)
best_fit1 = poly1(df1['DATE'].astype(np.int64) // 10**9)

# Fit a higher order polynomial (degree 5) to the second data series
coeffs2 = np.polyfit(df2['DATE.1'].astype(np.int64) // 10**9, df2['% Change'],␣
 ↪5)
poly2 = np.poly1d(coeffs2)
best_fit2 = poly2(df2['DATE.1'].astype(np.int64) // 10**9)

# Plotting
fig, ax1 = plt.subplots(figsize=(12, 6))

# Plot the first data series as scatter plot
ax1.scatter(df1['DATE'], df1['UM Sentiment'], color='b', label='UM Sentiment')
ax1.plot(df1['DATE'], best_fit1, color='b', linestyle='--', label='Best fit UM␣
 ↪Sentiment')
ax1.set_xlabel('Date', fontsize = 12)
ax1.set_ylabel('UM Sentiment', color='b', fontsize = 12)
ax1.tick_params(axis='y', labelcolor='b')

# Create a second y-axis
ax2 = ax1.twinx()
ax2.scatter(df2['DATE.1'], df2['% Change'], color='r', label='% Change')
```

```
ax2.plot(df2['DATE.1'], best_fit2, color='r', linestyle='--', label='Best fit %␣
 ↪Change')
ax2.set_ylabel('Real GDP % Change YoY', color='r', fontsize = 12)
ax2.tick_params(axis='y', labelcolor='r')
ax2.axhline(0, color='black', linestyle='--', linewidth=0.8)

# Add legends
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

# Title
plt.title('UM Sentiment and GDP Change over Time', fontsize = 14)

# Show the plot
plt.show()

# Output the polynomial functions
um_sentiment_poly = ' + '.join([f'{coef:.6e}*x^{i}' for i, coef in␣
 ↪enumerate(coeffs1[::-1])])
gdp_change_poly = ' + '.join([f'{coef:.6e}*x^{i}' for i, coef in␣
 ↪enumerate(coeffs2[::-1])])

print(f'UM Sentiment best fit polynomial: \n\n{um_sentiment_poly}')
print('\n')
print(f'Real GDP % Change YoY best fit polynomial: \n\n{gdp_change_poly}')
```



UM Sentiment best fit polynomial:

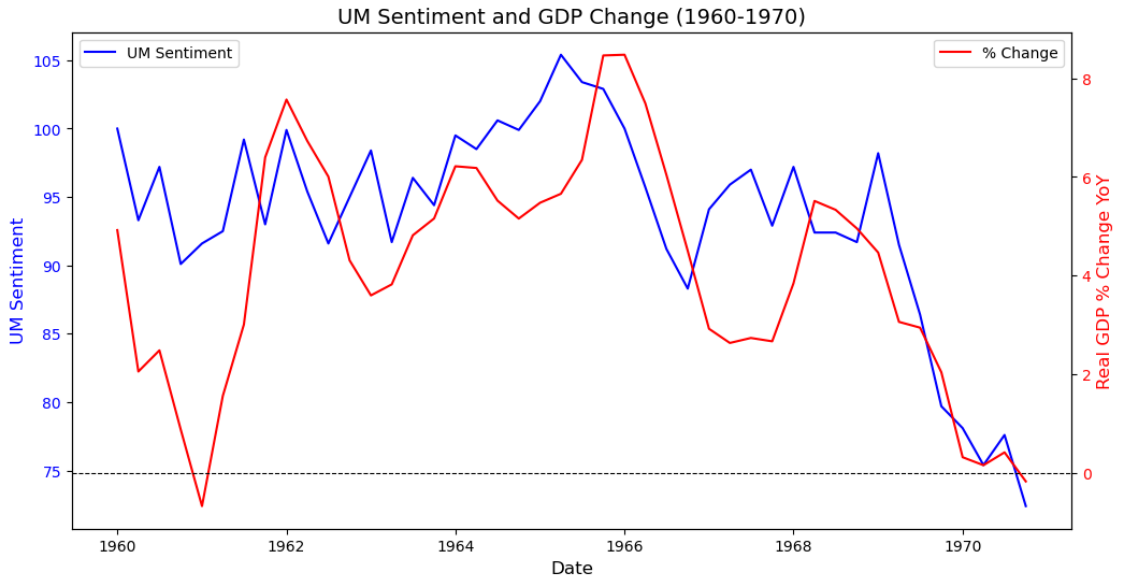
```
8.502466e+01*x^0 + -3.258624e-08*x^1 + 4.462626e-17*x^2 + 7.920849e-26*x^3 +
-1.251015e-34*x^4 + 4.060658e-44*x^5


Real GDP % Change YoY best fit polynomial:

3.684929e+00*x^0 + -2.711412e-09*x^1 + 1.054177e-18*x^2 + 1.037906e-26*x^3 +
-1.458246e-35*x^4 + 5.072347e-45*x^5
```

More work can be done here. I think fitting a function to the whole time series might be a stretch, but setting up thresholds for functions could be interesting to research more trading indicators.

4 1960 to 1970 - How is GDP calculated?

[125]:
```python
# Convert DATE columns to datetime
df['DATE'] = pd.to_datetime(df['DATE'])
df['DATE.1'] = pd.to_datetime(df['DATE.1'])

# Filter data for the years 1960 to 1970
df1_6070 = df[(df['DATE'].dt.year >= 1960) & (df['DATE'].dt.year <= 1970)]
df2_6070 = df[(df['DATE.1'].dt.year >= 1960) & (df['DATE.1'].dt.year <= 1970)]

# Drop rows with NaNs in either set of columns
df1_6070 = df1_6070[['DATE', 'UM Sentiment']].dropna()
df2_6070 = df2_6070[['DATE.1', '% Change']].dropna()

# Plotting
fig, ax1 = plt.subplots(figsize=(12, 6))

# Plot the first data series
ax1.plot(df1_6070['DATE'], df1_6070['UM Sentiment'], 'b-', label='UM Sentiment')
ax1.set_xlabel('Date', fontsize = 12)
ax1.set_ylabel('UM Sentiment', color='b', fontsize = 12)
ax1.tick_params(axis='y', labelcolor='b')

# Create a second y-axis
ax2 = ax1.twinx()
ax2.plot(df2_6070['DATE.1'], df2_6070['% Change'], 'r-', label='% Change')
ax2.set_ylabel('Real GDP % Change YoY', color='r', fontsize = 12)
ax2.tick_params(axis='y', labelcolor='r')
ax2.axhline(0, color='black', linestyle='--', linewidth=0.8)

# Add legends
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

# Title
```

```
plt.title('UM Sentiment and GDP Change (1960-1970)', fontsize = 14)

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



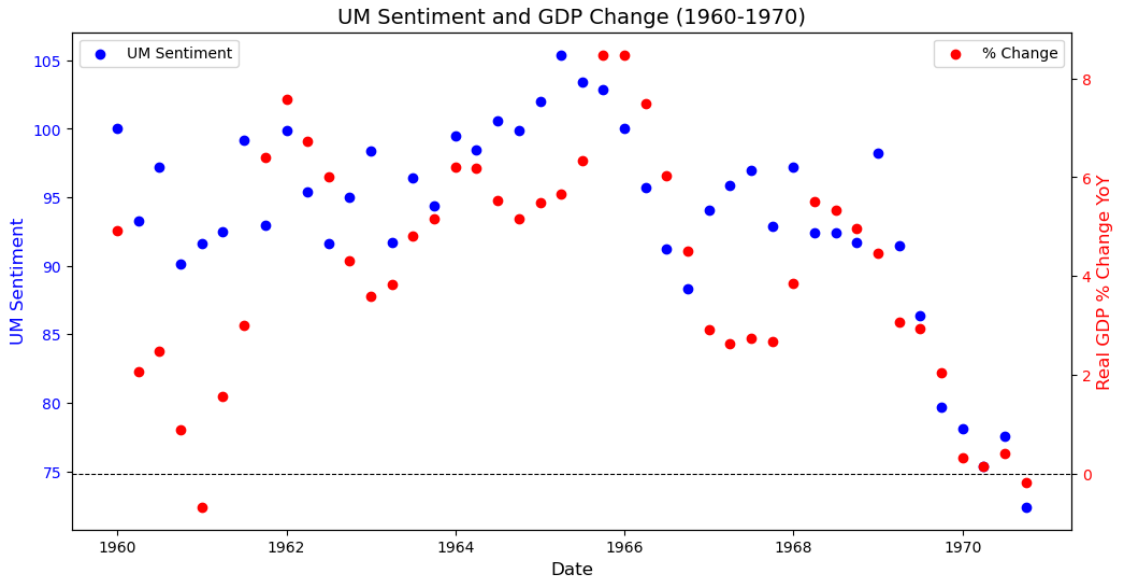
5 Measuring Correlation

[126]:
```
np.corrcoef(df1_6070['UM Sentiment'],df2_6070['% Change'])
```

[126]:
```
array([[1.        , 0.6799574],
       [0.6799574, 1.        ]])
```

[131]:
```
# Plotting
fig, ax1 = plt.subplots(figsize=(12, 6))

# Plot the first data series as scatter plot
ax1.scatter(df1_6070['DATE'], df1_6070['UM Sentiment'], color='b', label='UM␣
 ↪Sentiment')
ax1.set_xlabel('Date', fontsize = 12)
ax1.set_ylabel('UM Sentiment', color='b', fontsize = 12)
ax1.tick_params(axis='y', labelcolor='b')

# Create a second y-axis
ax2 = ax1.twinx()
ax2.scatter(df2_6070['DATE.1'], df2_6070['% Change'], color='r', label='%␣
 ↪Change')
```

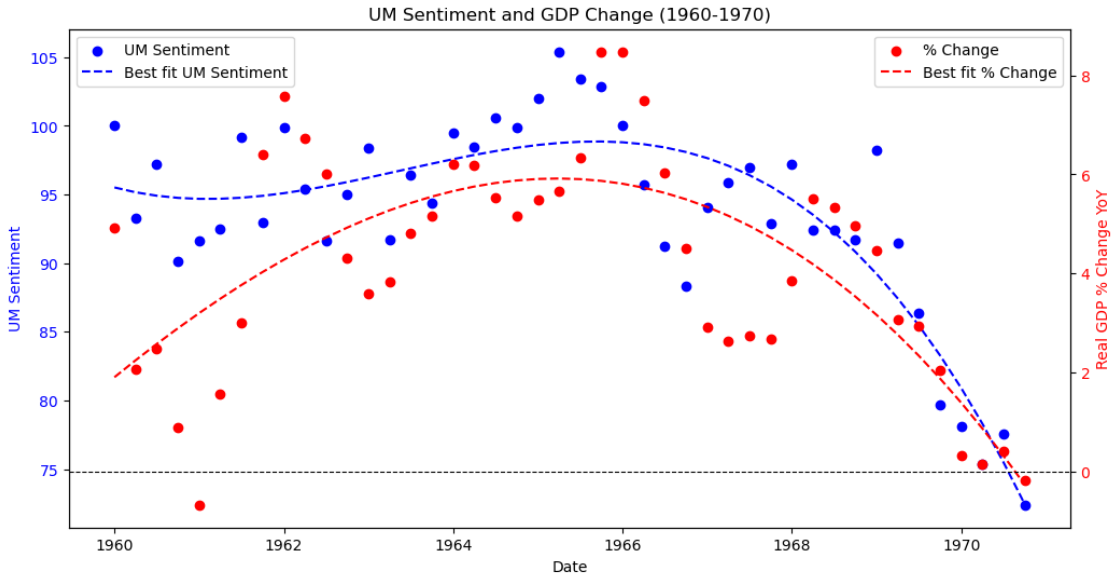
```
ax2.set_ylabel('Real GDP % Change YoY', color='r', fontsize = 12)
ax2.tick_params(axis='y', labelcolor='r')
ax2.axhline(0, color='black', linestyle='--', linewidth=0.8)

# Add legends
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

# Title
plt.title('UM Sentiment and GDP Change (1960-1970)', fontsize = 14)

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



6 Fitting Functions

[134]:
```
# Fit a polynomial (degree 3) to the first data series
coeffs1 = np.polyfit(df1_6070['DATE'].astype(np.int64) // 10**9, df1_6070['UM␣
 ↪Sentiment'], 3)
poly1 = np.poly1d(coeffs1)
best_fit1 = poly1(df1_6070['DATE'].astype(np.int64) // 10**9)

# Fit a polynomial (degree 3) to the second data series
coeffs2 = np.polyfit(df2_6070['DATE.1'].astype(np.int64) // 10**9, df2_6070['%␣
 ↪Change'], 3)
poly2 = np.poly1d(coeffs2)
```

```
best_fit2 = poly2(df2_6070['DATE.1'].astype(np.int64) // 10**9)

# Plotting
fig, ax1 = plt.subplots(figsize=(12, 6))

# Plot the first data series as scatter plot
ax1.scatter(df1_6070['DATE'], df1_6070['UM Sentiment'], color='b', label='UM␣
 ↪Sentiment')
ax1.plot(df1_6070['DATE'], best_fit1, color='b', linestyle='--', label='Best␣
 ↪fit UM Sentiment')
ax1.set_xlabel('Date')
ax1.set_ylabel('UM Sentiment', color='b')
ax1.tick_params(axis='y', labelcolor='b')

# Create a second y-axis
ax2 = ax1.twinx()
ax2.scatter(df2_6070['DATE.1'], df2_6070['% Change'], color='r', label='%␣
 ↪Change')
ax2.plot(df2_6070['DATE.1'], best_fit2, color='r', linestyle='--', label='Best␣
 ↪fit % Change')
ax2.set_ylabel('Real GDP % Change YoY', color='r')
ax2.tick_params(axis='y', labelcolor='r')
ax2.axhline(0, color='black', linestyle='--', linewidth=0.8)

# Add legends
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

# Title
plt.title('UM Sentiment and GDP Change (1960-1970)')

# Show the plot
plt.show()

# Output the polynomial functions
um_sentiment_poly = ' + '.join([f'{coef:.6e}*x^{i}' for i, coef in␣
 ↪enumerate(coeffs1[::-1])])
gdp_change_poly = ' + '.join([f'{coef:.6e}*x^{i}' for i, coef in␣
 ↪enumerate(coeffs2[::-1])])

print(f'UM Sentiment best fit polynomial: \n\n{um_sentiment_poly}')
print('\n')
print(f'Real GDP % Change YoY best fit polynomial: \n\n{gdp_change_poly}')
```

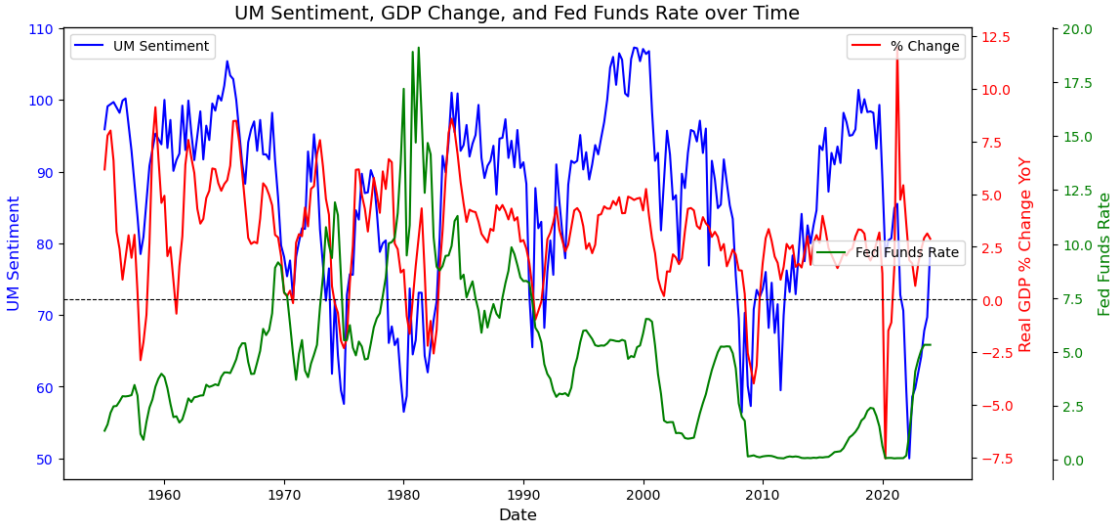



```
UM Sentiment best fit polynomial:

8.088374e+01*x^0 + -3.153550e-07*x^1 + -1.720943e-15*x^2 + -2.752281e-24*x^3


Real GDP % Change YoY best fit polynomial:

1.388919e+00*x^0 + -6.394078e-08*x^1 + -2.499365e-16*x^2 + -1.665509e-25*x^3
```

7 Further Analysis

[135]:
```
# Convert DATE columns to datetime
df['DATE'] = pd.to_datetime(df['DATE'])
df['DATE.1'] = pd.to_datetime(df['DATE.1'])
df['DATE.2'] = pd.to_datetime(df['DATE.2'])

# Drop rows with NaNs in either set of columns
df1 = df[['DATE', 'UM Sentiment']].dropna()
df2 = df[['DATE.1', '% Change']].dropna()
df3 = df[['DATE.2', 'Fed Funds Rate']].dropna()

# Plotting
fig, ax1 = plt.subplots(figsize=(12, 6))

# Plot the first data series
ax1.plot(df1['DATE'], df1['UM Sentiment'], 'b-', label='UM Sentiment')
ax1.set_xlabel('Date', fontsize = 12)
```

```
ax1.set_ylabel('UM Sentiment', color='b', fontsize = 12)
ax1.tick_params(axis='y', labelcolor='b')

# Create a second y-axis
ax2 = ax1.twinx()
ax2.plot(df2['DATE.1'], df2['% Change'], 'r-', label='% Change')
ax2.set_ylabel('Real GDP % Change YoY', color='r', fontsize = 12)
ax2.tick_params(axis='y', labelcolor='r')
ax2.axhline(0, color='black', linestyle='--', linewidth=0.8)

# Create a third y-axis
ax3 = ax1.twinx()
ax3.spines['right'].set_position(('outward', 60))  # Offset the third axis
ax3.plot(df3['DATE.2'], df3['Fed Funds Rate'], 'g-', label='Fed Funds Rate')
ax3.set_ylabel('Fed Funds Rate', color='g', fontsize = 12)
ax3.tick_params(axis='y', labelcolor='g')

# Add legends
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')
ax3.legend(loc='center right')

# Title
plt.title('UM Sentiment, GDP Change, and Fed Funds Rate over Time', fontsize =␣
 ↪14)

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



8 Sector Data Analysis

8.1 Technology: XLK

8.2 Healthcare: XLV

8.3 Financials: XLF

8.4 Consumer Discretionary: XLY

8.5 Industrials: XLI

8.6 Energy: XLE

8.7 Utilities: XLU

8.8 Consumer Staples: XLP

8.9 Materials: XLB

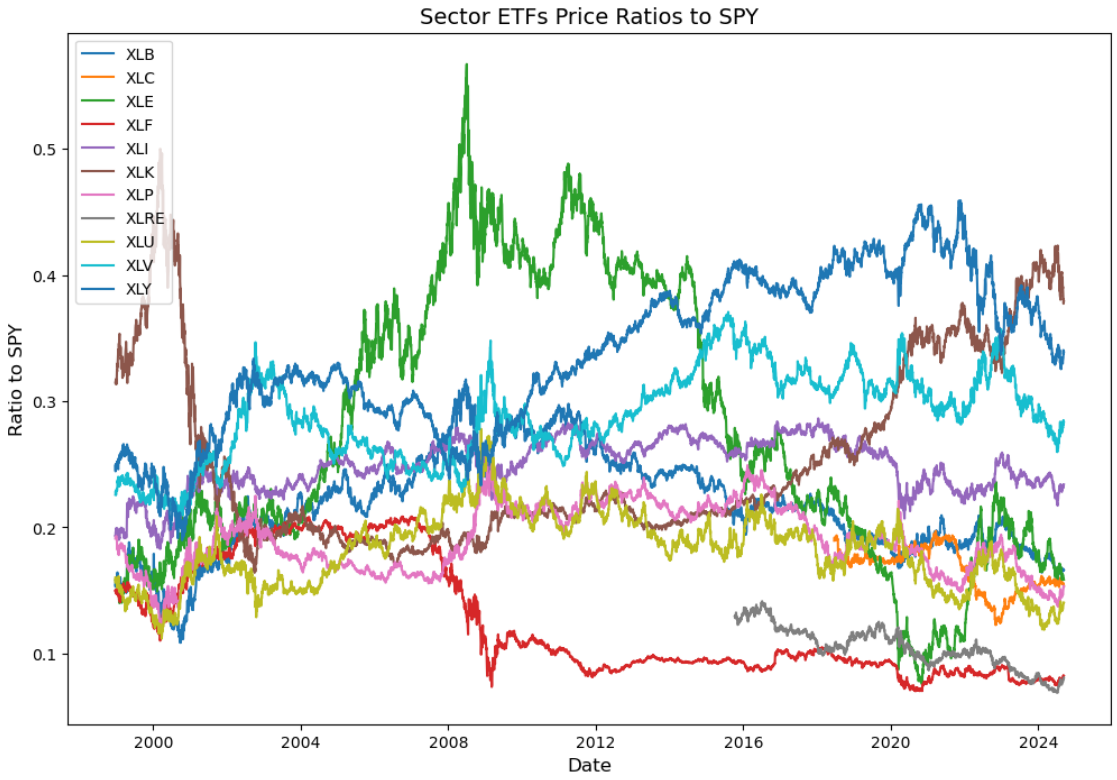
8.10 Real Estate: XLRE

8.11 Communication Services: XLC

[40]:
```
# list of sector ETFs
sector_etfs = {
    'Technology': 'XLK',
    'Healthcare': 'XLV',
    'Financials': 'XLF',
    'Consumer Discretionary': 'XLY',
    'Industrials': 'XLI',
    'Energy': 'XLE',
    'Utilities': 'XLU',
    'Consumer Staples': 'XLP',
    'Materials': 'XLB',
    'Real Estate': 'XLRE',
    'Communication Services': 'XLC'
}

# download SPY price
spy = yf.download('SPY', period='max')['Adj Close']

# download all sector ETFs prices
sector_data = yf.download(list(sector_etfs.values()), period='max')['Adj Close']

# divide each sector ETF price by SPY price
sector_ratio = sector_data.divide(spy, axis=0)

# plot all data
plt.figure(figsize=(12, 8))
for etf in sector_ratio.columns:
    plt.plot(sector_ratio.index, sector_ratio[etf], label=etf)
```

```
plt.title('Sector ETFs Price Ratios to SPY', fontsize = 14)
plt.xlabel('Date', fontsize = 12)
plt.ylabel('Ratio to SPY', fontsize = 12)
plt.legend(loc='upper left')
plt.show()
```

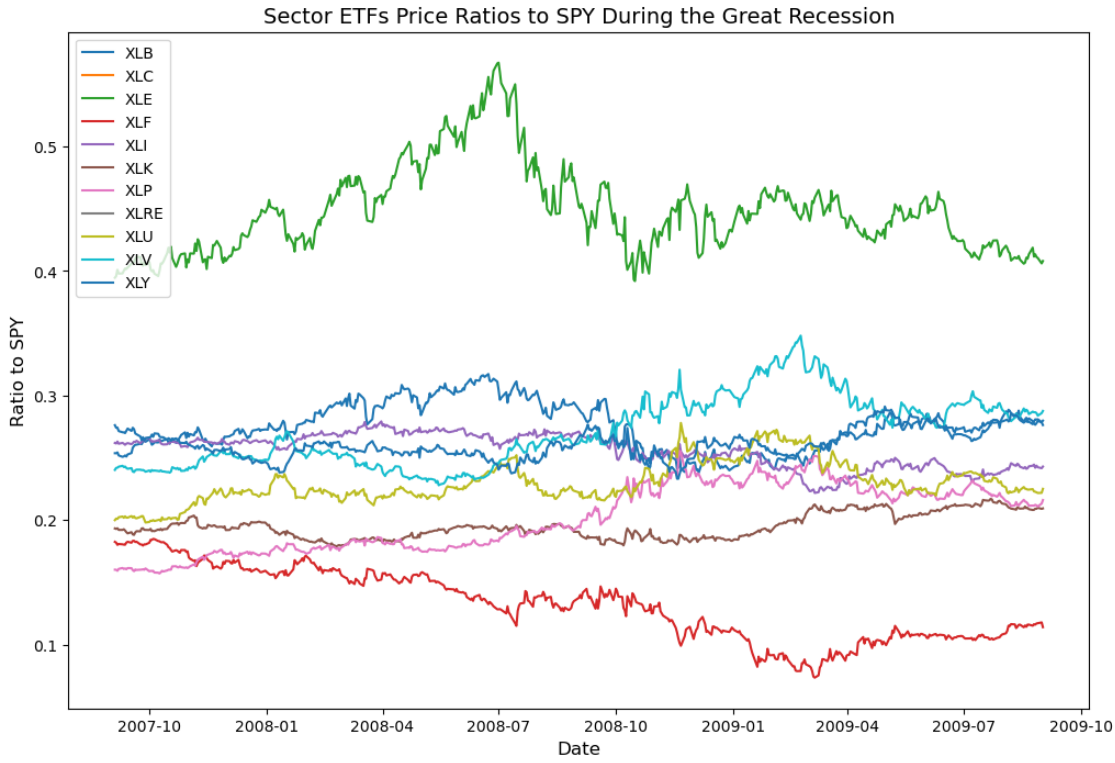
```
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  11 of 11 completed
```



9 Subset to Great Recession

[41]:
```
start_date = '2007-09-01'
end_date = '2009-09-01'
sector_ratio_recession = sector_ratio.loc[start_date:end_date]

# plot the data
plt.figure(figsize=(12, 8))
for etf in sector_ratio_recession.columns:
    plt.plot(sector_ratio_recession.index, sector_ratio_recession[etf],␣
 ↪label=etf)
```

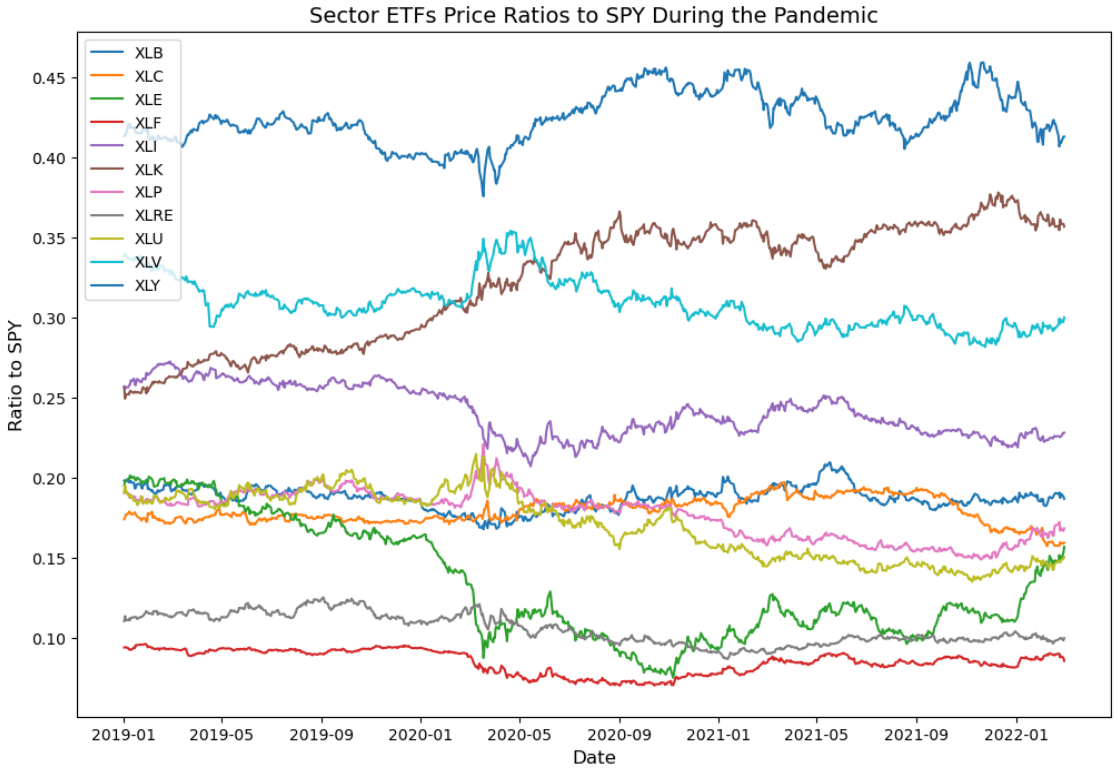
```
plt.title('Sector ETFs Price Ratios to SPY During the Great Recession',␣
 ↪fontsize = 14)
plt.xlabel('Date', fontsize = 12)
plt.ylabel('Ratio to SPY', fontsize = 12)
plt.legend(loc='upper left')
plt.show()
```



10 Subset to Pandemic

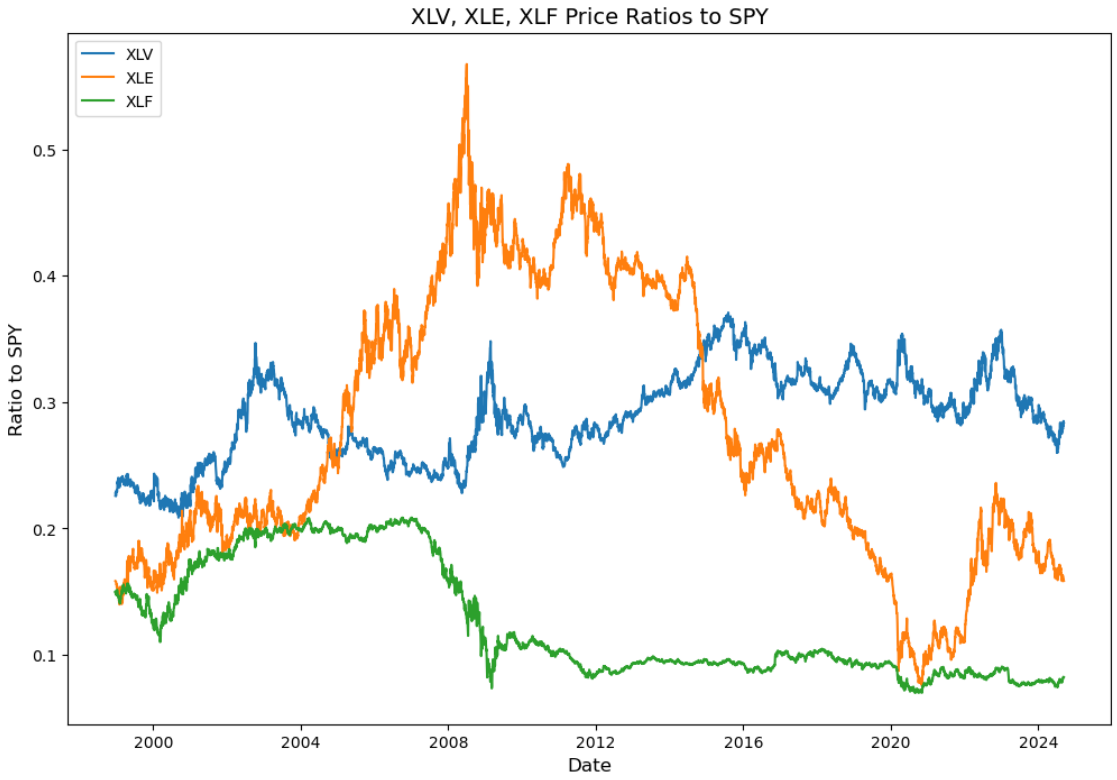
[42]:
```
start_date = '2019-01-01'
end_date = '2022-03-01'
sector_ratio_recession = sector_ratio.loc[start_date:end_date]

# plot the data
plt.figure(figsize=(12, 8))
for etf in sector_ratio_recession.columns:
    plt.plot(sector_ratio_recession.index, sector_ratio_recession[etf],␣
 ↪label=etf)

plt.title('Sector ETFs Price Ratios to SPY During the Pandemic', fontsize = 14)
plt.xlabel('Date', fontsize = 12)
plt.ylabel('Ratio to SPY', fontsize = 12)
```

```
plt.legend(loc='upper left')
plt.show()
```



From looking at both of the graphs, XLV (Healthcare), XLE (Energy), and XLF (Financials) are the most interesting and tend to have significant reactions to economic events.

[43]:
```
# plot only XLV, XLE, and XLF
plt.figure(figsize=(12, 8))
for etf in ['XLV', 'XLE', 'XLF']:
    plt.plot(sector_ratio.index, sector_ratio[etf], label=etf)

plt.title('XLV, XLE, XLF Price Ratios to SPY', fontsize = 14)
plt.xlabel('Date', fontsize = 12)
plt.ylabel('Ratio to SPY', fontsize = 12)
plt.legend(loc='upper left')
plt.show()
```



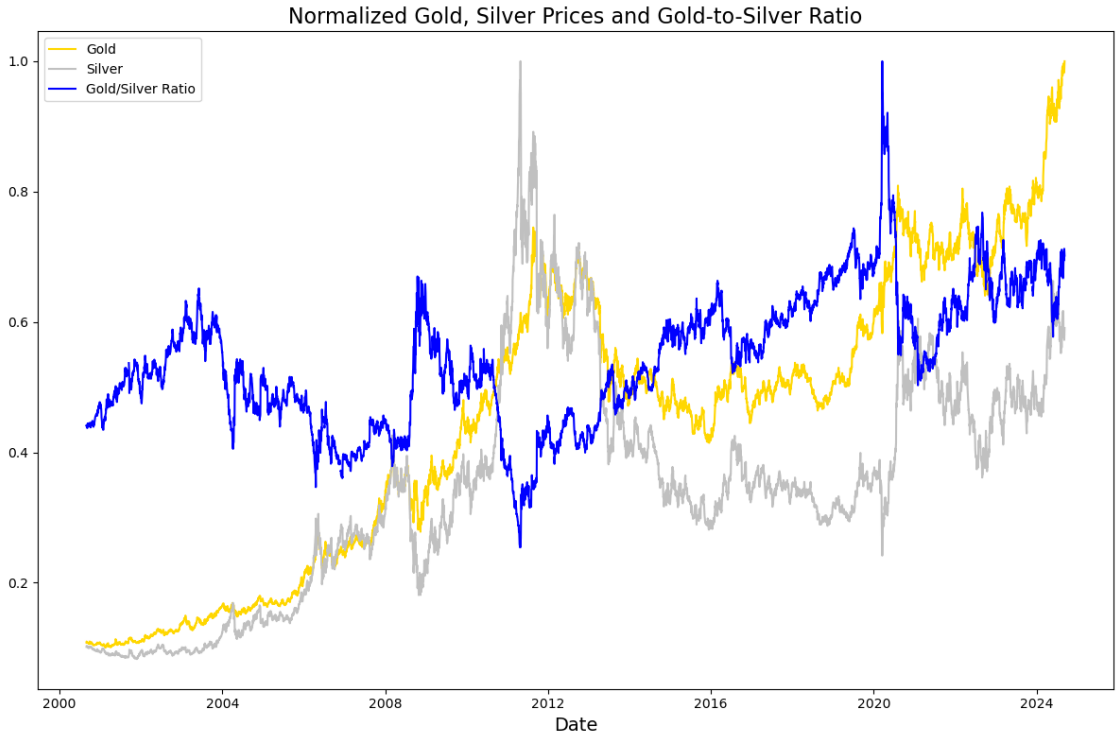
The above is interesting because XLF dropped a lot during the Great Recession (makes sense) but little to no reaction relative to the S&P during the pandemic. Healthcare seems pretty insignificant during both periods, but energy has always stuck out to me.

11 Gold/Silver Ratio

[74]:
```
# Define the tickers for gold and silver
gold_ticker = 'GC=F'
silver_ticker = 'SI=F'

# Download historical data for gold and silver
gold_data = yf.download(gold_ticker, start='1900-01-01')
silver_data = yf.download(silver_ticker, start='1900-01-01')

# Calculate the gold-to-silver ratio
ratio = gold_data['Adj Close'] / silver_data['Adj Close']

# Normalize the prices
gold_norm = gold_data['Adj Close'] / gold_data['Adj Close'].max()
silver_norm = silver_data['Adj Close'] / silver_data['Adj Close'].max()
ratio_norm = ratio / ratio.max()
```

```
# Plot the normalized prices and the gold-to-silver ratio
plt.figure(figsize=(12, 8))
plt.plot(gold_norm, label='Gold', color='gold')
plt.plot(silver_norm, label='Silver', color='silver')
plt.plot(ratio_norm, label='Gold/Silver Ratio', color='blue')
plt.title('Normalized Gold, Silver Prices and Gold-to-Silver Ratio', fontsize =␣
↪16)
plt.xlabel('Date', fontsize = 14)
plt.legend()

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

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



[137]:
```
# Define the tickers for gold and silver
gold_ticker = 'GC=F'
silver_ticker = 'SI=F'

# Download historical data for gold and silver
gold_data = yf.download(gold_ticker, start='1900-01-01')
```

```
silver_data = yf.download(silver_ticker, start='1900-01-01')

# Align the indices of gold and silver data
aligned_data = gold_data['Adj Close'].align(silver_data['Adj Close'],␣
 ↪join='inner')

# Calculate the gold-to-silver ratio
ratio = aligned_data[0] / aligned_data[1]

# Normalize the ratio values for color mapping
norm = mcolors.Normalize(vmin=ratio.min(), vmax=ratio.max())

# use a different color map that is more pronounced
cmap = plt.get_cmap('viridis')

# Create a color array based on the ratio values
colors = cmap(norm(ratio.values))

# Plot the prices of gold and silver
fig, ax1 = plt.subplots(figsize=(14, 7))

# Plot gold prices on the primary y-axis
ax1.plot(gold_data['Adj Close'], label='Gold', color='orange')
ax1.set_xlabel('Date', fontsize = 12)
ax1.set_ylabel('Gold Price', color='orange', fontsize = 12)
ax1.tick_params(axis='y', labelcolor='orange')

# Create a secondary y-axis for silver prices
ax2 = ax1.twinx()
ax2.plot(silver_data['Adj Close'], label='Silver', color='black')
ax2.set_ylabel('Silver Price', color='black', fontsize = 12)
ax2.tick_params(axis='y', labelcolor='black')

# Create a third axis for the gold-to-silver ratio using the same x-axis
ax3 = ax1.twinx()
ax3.spines['right'].set_position(('outward', 60))  # Offset the third axis
ax3.set_frame_on(True)
ax3.patch.set_visible(False)
ax3.set_yticks([])

# Plot the gold-to-silver ratio with color coding
for i in range(len(ratio) - 1):
    ax3.plot(ratio.index[i:i+2], ratio.values[i:i+2], color=colors[i])

# Create a scalar mappable for the color bar
sm = cm.ScalarMappable(cmap=cmap, norm=norm)
sm.set_array([])
```

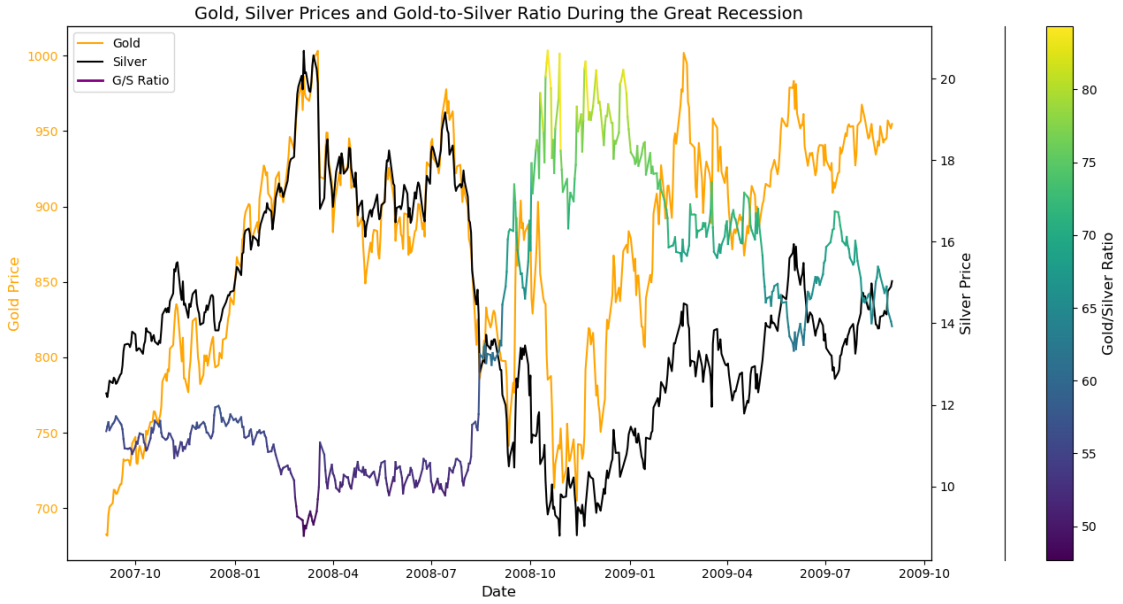
```
# Add a color bar for the gold-to-silver ratio heatmap
cbar = fig.colorbar(sm, ax=ax3, orientation='vertical', pad=0.1)
cbar.set_label('Gold/Silver Ratio', fontsize = 12)

# Add legends
lines_1, labels_1 = ax1.get_legend_handles_labels()
lines_2, labels_2 = ax2.get_legend_handles_labels()
legend_lines = lines_1 + lines_2
legend_labels = labels_1 + labels_2
legend_lines.append(plt.Line2D([0], [0], color='purple', lw=2))
legend_labels.append('G/S Ratio')
ax1.legend(legend_lines, legend_labels, loc='upper left')

# Display the plot
plt.title('Gold, Silver Prices and Gold-to-Silver Ratio with Heatmap', fontsize␣
  ↪= 14)
plt.tight_layout()
plt.show()
```

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

12 Great Recession

[138]:
```
start_date = '2007-09-01'
end_date = '2009-09-01'

# subset the data
gold_data_subset = gold_data.loc[start_date:end_date]
silver_data_subset = silver_data.loc[start_date:end_date]

# align the indices of gold and silver data
aligned_data_subset = gold_data_subset['Adj Close'].
 ↪align(silver_data_subset['Adj Close'], join='inner')

# calculate the gold-to-silver ratio
ratio_subset = aligned_data_subset[0] / aligned_data_subset[1]

# normalize the ratio values for color mapping
norm_subset = mcolors.Normalize(vmin=ratio_subset.min(), vmax=ratio_subset.
 ↪max())

# use a different color map that is more pronounced
cmap = plt.get_cmap('viridis')

# create a color array based on the ratio values
colors_subset = cmap(norm_subset(ratio_subset.values))

# plot the prices of gold and silver
fig, ax1 = plt.subplots(figsize=(14, 7))

# plot gold prices on the primary y-axis
ax1.plot(gold_data_subset['Adj Close'], label='Gold', color='orange')
ax1.set_xlabel('Date', fontsize = 12)
ax1.set_ylabel('Gold Price', color='orange', fontsize = 12)
ax1.tick_params(axis='y', labelcolor='orange')

# create a secondary y-axis for silver prices
ax2 = ax1.twinx()
ax2.plot(silver_data_subset['Adj Close'], label='Silver', color='black')
ax2.set_ylabel('Silver Price', color='black', fontsize = 12)
ax2.tick_params(axis='y', labelcolor='black')

# create a third axis for the gold-to-silver ratio using the same x-axis
ax3 = ax1.twinx()
ax3.spines['right'].set_position(('outward', 60))  # offset the third axis
ax3.set_frame_on(True)
ax3.patch.set_visible(False)
ax3.set_yticks([])
```

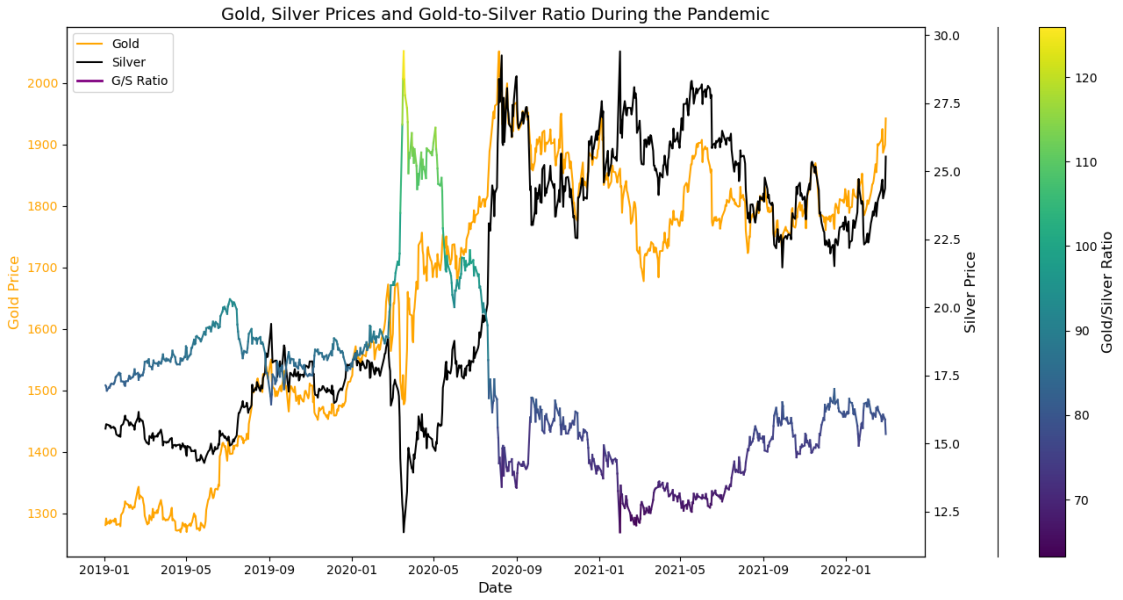
```
# plot the gold-to-silver ratio with color coding
for i in range(len(ratio_subset) - 1):
    ax3.plot(ratio_subset.index[i:i+2], ratio_subset.values[i:i+2],␣
 ↪color=colors_subset[i])

# create a scalar mappable for the color bar
sm_subset = cm.ScalarMappable(cmap=cmap, norm=norm_subset)
sm_subset.set_array([])

# add a color bar for the gold-to-silver ratio heatmap
cbar = fig.colorbar(sm_subset, ax=ax3, orientation='vertical', pad=0.1)
cbar.set_label('Gold/Silver Ratio', fontsize = 12)

# add legends
lines_1, labels_1 = ax1.get_legend_handles_labels()
lines_2, labels_2 = ax2.get_legend_handles_labels()
legend_lines = lines_1 + lines_2
legend_labels = labels_1 + labels_2
legend_lines.append(plt.Line2D([0], [0], color='purple', lw=2))
legend_labels.append('G/S Ratio')
ax1.legend(legend_lines, legend_labels, loc='upper left')

# display the plot
plt.title('Gold, Silver Prices and Gold-to-Silver Ratio During the Great␣
 ↪Recession', fontsize = 14)
plt.tight_layout()
plt.show()
```



13 Pandemic

[139]:
```
start_date = '2019-01-01'
end_date = '2022-03-01'

# subset the data
gold_data_subset = gold_data.loc[start_date:end_date]
silver_data_subset = silver_data.loc[start_date:end_date]

# align the indices of gold and silver data
aligned_data_subset = gold_data_subset['Adj Close'].
 ↪align(silver_data_subset['Adj Close'], join='inner')

# calculate the gold-to-silver ratio
ratio_subset = aligned_data_subset[0] / aligned_data_subset[1]

# normalize the ratio values for color mapping
norm_subset = mcolors.Normalize(vmin=ratio_subset.min(), vmax=ratio_subset.
 ↪max())

# use a different color map that is more pronounced
cmap = plt.get_cmap('viridis')

# create a color array based on the ratio values
colors_subset = cmap(norm_subset(ratio_subset.values))

# plot the prices of gold and silver
fig, ax1 = plt.subplots(figsize=(14, 7))

# plot gold prices on the primary y-axis
ax1.plot(gold_data_subset['Adj Close'], label='Gold', color='orange')
ax1.set_xlabel('Date', fontsize = 12)
ax1.set_ylabel('Gold Price', color='orange', fontsize = 12)
ax1.tick_params(axis='y', labelcolor='orange')

# create a secondary y-axis for silver prices
ax2 = ax1.twinx()
ax2.plot(silver_data_subset['Adj Close'], label='Silver', color='black')
ax2.set_ylabel('Silver Price', color='black', fontsize = 12)
ax2.tick_params(axis='y', labelcolor='black')

# create a third axis for the gold-to-silver ratio using the same x-axis
ax3 = ax1.twinx()
ax3.spines['right'].set_position(('outward', 60))  # offset the third axis
ax3.set_frame_on(True)
```

```
ax3.patch.set_visible(False)
ax3.set_yticks([])

# plot the gold-to-silver ratio with color coding
for i in range(len(ratio_subset) - 1):
    ax3.plot(ratio_subset.index[i:i+2], ratio_subset.values[i:i+2],␣
 ↪color=colors_subset[i])

# create a scalar mappable for the color bar
sm_subset = cm.ScalarMappable(cmap=cmap, norm=norm_subset)
sm_subset.set_array([])

# add a color bar for the gold-to-silver ratio heatmap
cbar = fig.colorbar(sm_subset, ax=ax3, orientation='vertical', pad=0.1)
cbar.set_label('Gold/Silver Ratio', fontsize = 12)

# add legends
lines_1, labels_1 = ax1.get_legend_handles_labels()
lines_2, labels_2 = ax2.get_legend_handles_labels()
legend_lines = lines_1 + lines_2
legend_labels = labels_1 + labels_2
legend_lines.append(plt.Line2D([0], [0], color='purple', lw=2))
legend_labels.append('G/S Ratio')
ax1.legend(legend_lines, legend_labels, loc='upper left')

# display the plot
plt.title('Gold, Silver Prices and Gold-to-Silver Ratio During the Pandemic',␣
 ↪fontsize = 14)
plt.tight_layout()
plt.show()
```



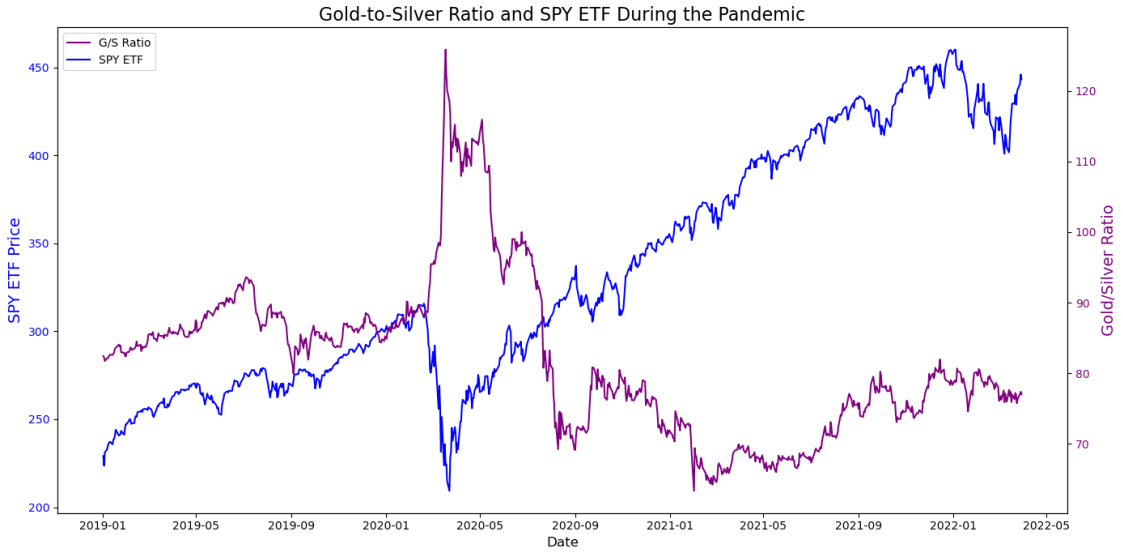
Note that the G/S ratio rose to an all-time high at the beginning of the pandemic.

14 But what did the S&P do at the same time during the pandemic?

[140]:
```
# Define the tickers for gold, silver, and SPY ETF
gold_ticker = 'GC=F'
silver_ticker = 'SI=F'
spy_ticker = 'SPY'

# Download historical data for gold, silver, and SPY ETF
start_date = '2019-01-01'
end_date = '2022-03-31'
gold_data = yf.download(gold_ticker, start=start_date, end=end_date)
silver_data = yf.download(silver_ticker, start=start_date, end=end_date)
spy_data = yf.download(spy_ticker, start=start_date, end=end_date)

# Align the indices of gold and silver data
aligned_data = gold_data['Adj Close'].align(silver_data['Adj Close'],␣
 ↪join='inner')

# Calculate the gold-to-silver ratio
ratio = aligned_data[0] / aligned_data[1]

# Align the indices of the gold-to-silver ratio and SPY data
aligned_data_spy = ratio.align(spy_data['Adj Close'], join='inner')
```

```
# Plot the gold-to-silver ratio and SPY ETF
fig, ax1 = plt.subplots(figsize=(14, 7))

# Plot SPY ETF prices on the primary y-axis
ax1.plot(aligned_data_spy[1], label='SPY ETF', color='blue')
ax1.set_xlabel('Date', fontsize = 12)
ax1.set_ylabel('SPY ETF Price', color='blue', fontsize = 14)
ax1.tick_params(axis='y', labelcolor='blue')

# Create a secondary y-axis for the gold-to-silver ratio
ax2 = ax1.twinx()
ax2.plot(aligned_data_spy[0], label='G/S Ratio', color='purple')
ax2.set_ylabel('Gold/Silver Ratio', color='purple', fontsize = 14)
ax2.tick_params(axis='y', labelcolor='purple')

# Add legends
lines_1, labels_1 = ax1.get_legend_handles_labels()
lines_2, labels_2 = ax2.get_legend_handles_labels()
ax1.legend(lines_2 + lines_1, labels_2 + labels_1, loc='upper left')

# Display the plot
plt.title('Gold-to-Silver Ratio and SPY ETF During the Pandemic', fontsize = 16)
plt.tight_layout()
plt.show()
```

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



15 How much of a lag is there? This is a future trading opportunity

[116]:
```
# Define the tickers for gold, silver, and SPY ETF
gold_ticker = 'GC=F'
silver_ticker = 'SI=F'
spy_ticker = 'SPY'

# Download historical data for gold, silver, and SPY ETF
start_date = '2020-01-01'
end_date = '2020-06-01'
gold_data = yf.download(gold_ticker, start=start_date, end=end_date)
silver_data = yf.download(silver_ticker, start=start_date, end=end_date)
spy_data = yf.download(spy_ticker, start=start_date, end=end_date)

# Align the indices of gold and silver data
aligned_data = gold_data['Adj Close'].align(silver_data['Adj Close'],␣
 ↪join='inner')

# Calculate the gold-to-silver ratio
ratio = aligned_data[0] / aligned_data[1]

# Align the indices of the gold-to-silver ratio and SPY data
aligned_data_spy = ratio.align(spy_data['Adj Close'], join='inner')

# Plot the gold-to-silver ratio and SPY ETF
fig, ax1 = plt.subplots(figsize=(14, 7))

# Plot SPY ETF prices on the primary y-axis
ax1.plot(aligned_data_spy[1], label='SPY ETF', color='blue')
ax1.set_xlabel('Date', fontsize = 14)
ax1.set_ylabel('SPY ETF Price', color='blue', fontsize = 14)
ax1.tick_params(axis='y', labelcolor='blue')

# Create a secondary y-axis for the gold-to-silver ratio
ax2 = ax1.twinx()
ax2.plot(aligned_data_spy[0], label='G/S Ratio', color='purple')
ax2.set_ylabel('Gold/Silver Ratio', color='purple', fontsize = 14)
ax2.tick_params(axis='y', labelcolor='purple')

# Add legends
lines_1, labels_1 = ax1.get_legend_handles_labels()
lines_2, labels_2 = ax2.get_legend_handles_labels()
ax1.legend(lines_2 + lines_1, labels_2 + labels_1, loc='upper left')

# Display the plot
plt.title('Gold-to-Silver Ratio and SPY ETF During 2020', fontsize = 16)
```

```
plt.tight_layout()
fig.savefig('GS Ratio and SPY - covid.png')
plt.show()
```

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

It could be profitable to move into gold during a recession because of the lag.

16 Whole Time Horizon

[67]:
```
# Define the tickers for gold, silver, and SPY ETF
gold_ticker = 'GC=F'
silver_ticker = 'SI=F'
spy_ticker = 'SPY'

# Download historical data for gold, silver, and SPY ETF
start_date = '2000-01-01'
end_date = '2024-06-01'
gold_data = yf.download(gold_ticker, start=start_date, end=end_date)
silver_data = yf.download(silver_ticker, start=start_date, end=end_date)
spy_data = yf.download(spy_ticker, start=start_date, end=end_date)

# Align the indices of gold and silver data
aligned_data = gold_data['Adj Close'].align(silver_data['Adj Close'],␣
 ↪join='inner')

# Calculate the gold-to-silver ratio
```

```
ratio = aligned_data[0] / aligned_data[1]

# Align the indices of the gold-to-silver ratio and SPY data
aligned_data_spy = ratio.align(spy_data['Adj Close'], join='inner')

# Plot the gold-to-silver ratio and SPY ETF
fig, ax1 = plt.subplots(figsize=(14, 7))

# Plot SPY ETF prices on the primary y-axis
ax1.plot(aligned_data_spy[1], label='SPY ETF', color='blue')
ax1.set_xlabel('Date', fontsize = 14)
ax1.set_ylabel('SPY ETF Price', color='blue', fontsize = 14)
ax1.tick_params(axis='y', labelcolor='blue')

# Create a secondary y-axis for the gold-to-silver ratio
ax2 = ax1.twinx()
ax2.plot(aligned_data_spy[0], label='G/S Ratio', color='purple')
ax2.set_ylabel('Gold/Silver Ratio', color='purple', fontsize = 14)
ax2.tick_params(axis='y', labelcolor='purple')

# Add legends
lines_1, labels_1 = ax1.get_legend_handles_labels()
lines_2, labels_2 = ax2.get_legend_handles_labels()
ax1.legend(lines_2 + lines_1, labels_2 + labels_1, loc='upper left')

# Display the plot
plt.title('Gold-to-Silver Ratio and SPY ETF', fontsize = 16)
plt.tight_layout()
fig.savefig('GS Ratio and SPY.png')
plt.show()
```

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

17 Replicating what Greg showed me

[129]:
```
api_key = '876abb970a5e1cba77291fd327ce14e9'
```

[130]:
```
series_id = 'UNRATE'

# Fetch the unemployment data from FRED
unemployment_data = fred.get_series(series_id, api_key=api_key)

# Convert the data to a pandas DataFrame for easier manipulation
unemployment_df = pd.DataFrame(unemployment_data, columns=['date', 'value'])
unemployment_df['date'] = pd.to_datetime(unemployment_df['date'])
unemployment_df.set_index('date', inplace=True)

# Subset the data to start from the year 2000
unemployment_df = unemployment_df[unemployment_df.index >= '2000-01-01']

# Calculate the nominal year-over-year net change in unemployment rate
unemployment_df['YoY Change'] = unemployment_df['value'].diff(periods=12)

# Assume df2 is the DataFrame that contains Real GDP data with columns 'DATE.1'␣
 ↪and '% Change'
# Subset the Real GDP data to start from the year 2000
df2['DATE.1'] = pd.to_datetime(df2['DATE.1'])
df2 = df2[df2['DATE.1'] >= '2000-01-01']

# Create the plot with two y-axes
```

```
fig, ax1 = plt.subplots(figsize=(12, 6))

# Plot the nominal year-over-year net change in unemployment rate
ax1.plot(unemployment_df.index, unemployment_df['YoY Change'],␣
 ↪label='Unemployment Rate', color='blue')
ax1.set_xlabel('Date', fontsize = 14)
ax1.set_ylabel('Unemployment Rate YoY Change', color='blue', fontsize = 14)
ax1.tick_params(axis='y', labelcolor='blue')
ax1.legend(loc='upper left')

# Create a second y-axis for the Real GDP % change
ax2 = ax1.twinx()
ax2.plot(df2['DATE.1'], df2['% Change'], 'r-', label='Real GDP')
ax2.set_ylabel('Real GDP % Change YoY', color='r', fontsize = 14)
ax2.tick_params(axis='y', labelcolor='r')
ax2.axhline(0, color='black', linestyle='--', linewidth=0.8)
ax2.legend(loc='upper right')

# Add a title to the plot
plt.title('Year-over-Year Change in Unemployment Rate and Real GDP % Change␣
 ↪Over Time', fontsize = 16)

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

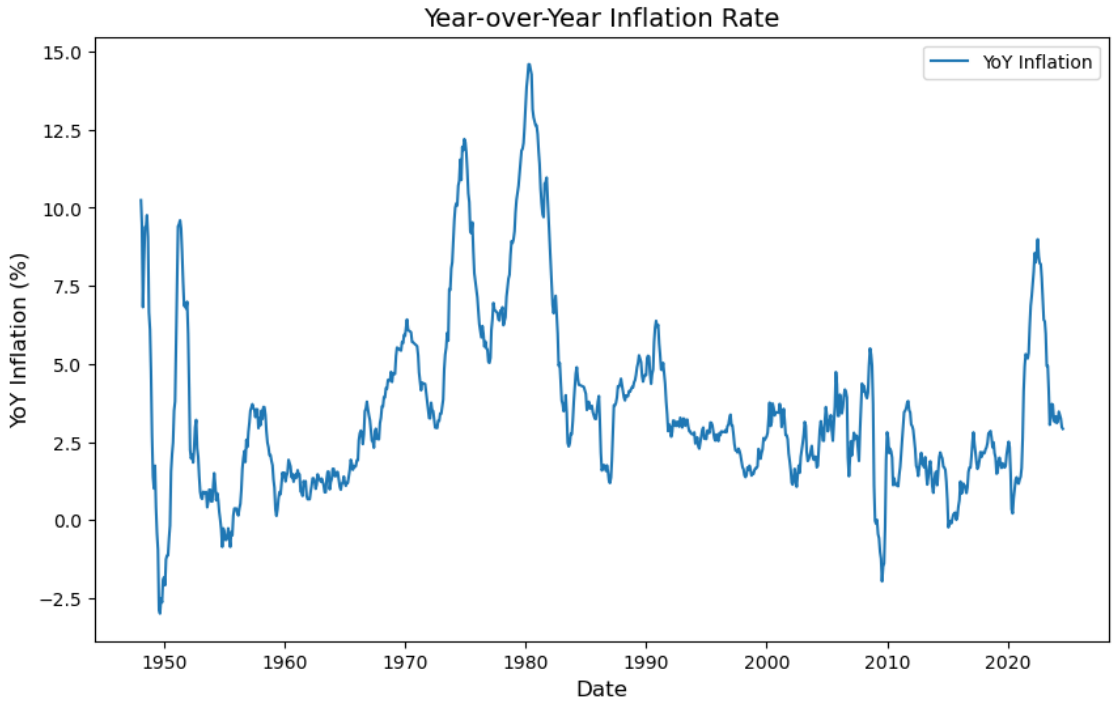
18 Slightly different correlation coefficient than actual because of the join

[32]:
```
# Ensure 'date' columns are in datetime format and set as index
unemployment_df.index = pd.to_datetime(unemployment_df.index)
df2['DATE.1'] = pd.to_datetime(df2['DATE.1'])
df2.set_index('DATE.1', inplace=True)

# Merge the two DataFrames on their indices (date) using an inner join to keep␣
 ↪only matching dates
merged_df = unemployment_df.join(df2['% Change'], how='inner')

# Drop rows with any NaN values
merged_df.dropna(inplace=True)

# Calculate the correlation coefficient
np.corrcoef(merged_df['YoY Change'], merged_df['% Change'])
```

[32]:
```
array([[ 1.       , -0.8428041],
       [-0.8428041,  1.       ]])
```

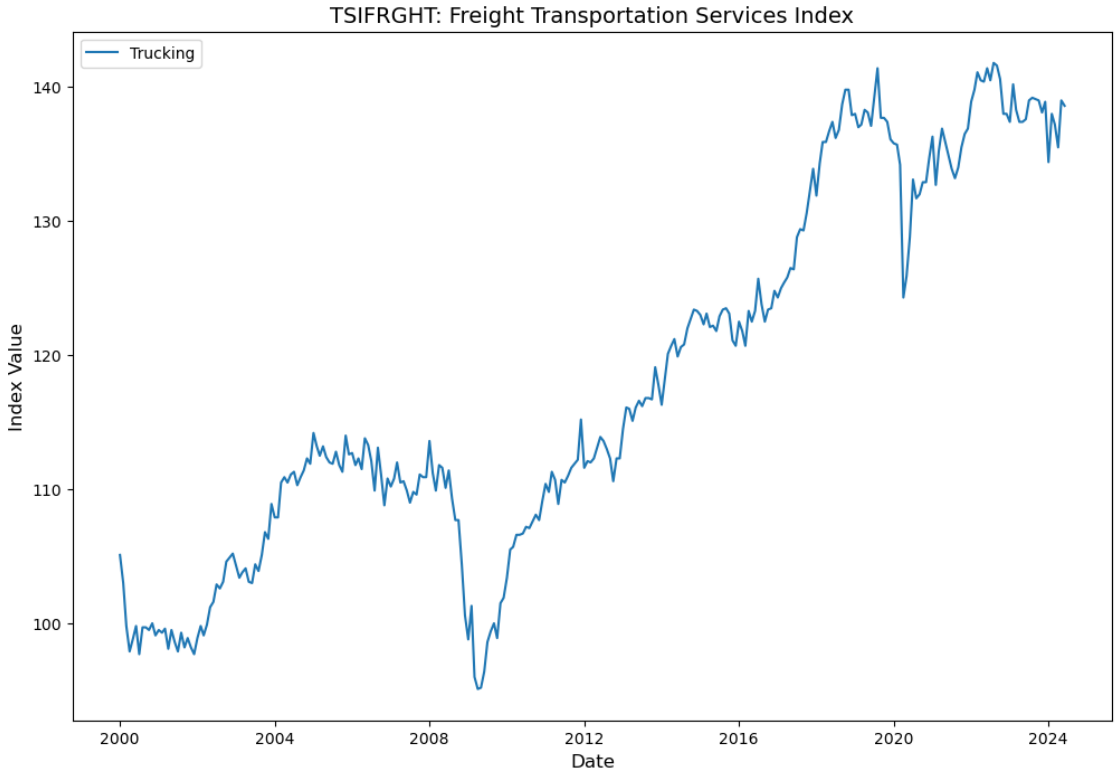
19 Inflation

[69]:
```
fred = Fred(api_key = api_key)

# pull the consumer price index for all urban consumers (CPI-U) for the U.S.
cpi_data = fred.get_series('CPIAUCSL')

# resample the data to monthly frequency and calculate YoY inflation
cpi_monthly = cpi_data.resample('M').last()
cpi_yoy = cpi_monthly.pct_change(periods = 12) * 100

# drop NaN values resulting from the pct_change calculation
cpi_yoy = cpi_yoy.dropna()

# plot the YoY inflation data
plt.figure(figsize = (10, 6))
plt.plot(cpi_yoy.index, cpi_yoy.values, label = 'YoY Inflation')
plt.xlabel('Date', fontsize = 12)
plt.ylabel('YoY Inflation (%)', fontsize = 12)
plt.title('Year-over-Year Inflation Rate', fontsize = 14)
plt.legend()
plt.show()
```



20 Freight Transportation Index

[114]:
```
fred = Fred(api_key = api_key)

# pull data for Freight Transportation (TSIFRGHT)
data = fred.get_series('TSIFRGHT')

# plot data for Trucking
plt.figure(figsize = (12, 8))
plt.plot(data.index, data.values, label = 'Trucking')
plt.xlabel('Date', fontsize = 12)
plt.ylabel('Index Value', fontsize = 12)
plt.title('TSIFRGHT: Freight Transportation Services Index', fontsize = 14)
plt.legend()
plt.show()
```



21 Rewriting Library Imports From Beginning

[35]:
```
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
import numpy as np
import matplotlib.colors as mcolors
import matplotlib.cm as cm
import pyfredapi as fred
from fredapi import Fred
from sklearn.preprocessing import MinMaxScaler
```

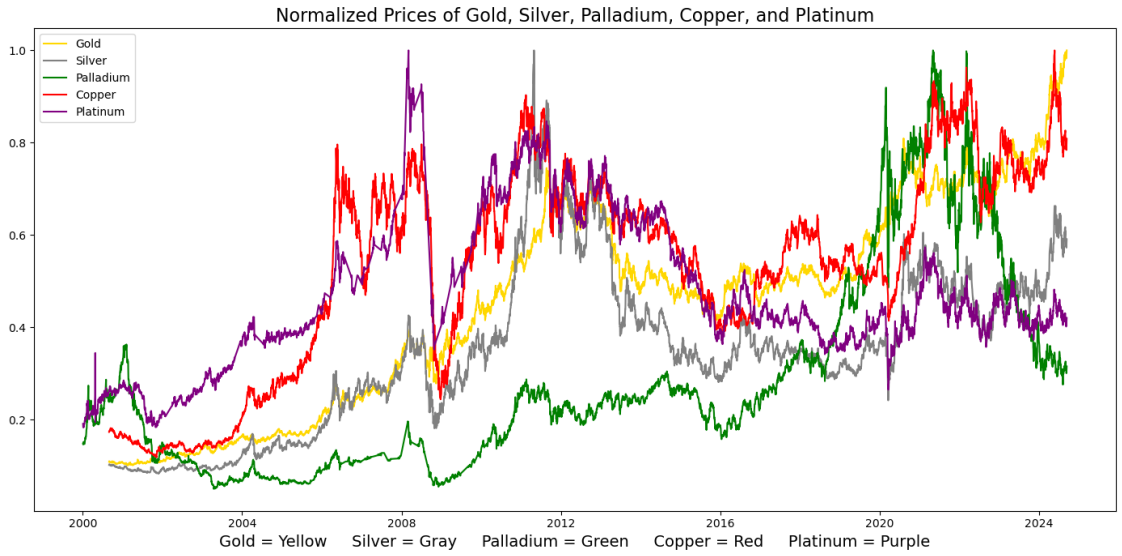
22 Other Precious Metals

adding to earlier plot

[75]:
```
# Define the tickers for gold, silver, palladium, copper, and platinum
gold_ticker = 'GC=F'
silver_ticker = 'SI=F'
palladium_ticker = 'PA=F'
copper_ticker = 'HG=F'
```

```
platinum_ticker = 'PL=F'

# Download historical data for gold, silver, palladium, copper, and platinum
gold_data = yf.download(gold_ticker, start='1900-01-01')
silver_data = yf.download(silver_ticker, start='1900-01-01')
palladium_data = yf.download(palladium_ticker, start='1900-01-01')
copper_data = yf.download(copper_ticker, start='1900-01-01')
platinum_data = yf.download(platinum_ticker, start='1900-01-01')

# Normalize the prices
gold_norm = gold_data['Adj Close'] / gold_data['Adj Close'].max()
silver_norm = silver_data['Adj Close'] / silver_data['Adj Close'].max()
palladium_norm = palladium_data['Adj Close'] / palladium_data['Adj Close'].max()
copper_norm = copper_data['Adj Close'] / copper_data['Adj Close'].max()
platinum_norm = platinum_data['Adj Close'] / platinum_data['Adj Close'].max()

# Plot the normalized prices
plt.figure(figsize=(14, 7))
plt.plot(gold_norm, label='Gold', color='gold')
plt.plot(silver_norm, label='Silver', color='gray')
plt.plot(palladium_norm, label='Palladium', color='green')
plt.plot(copper_norm, label='Copper', color='red')
plt.plot(platinum_norm, label='Platinum', color='purple')
plt.title('Normalized Prices of Gold, Silver, Palladium, Copper, and Platinum',␣
 ↪fontsize = 16)
plt.xlabel('Gold = Yellow     Silver = Gray     Palladium = Green     Copper =␣
 ↪Red     Platinum = Purple', fontsize = 14)
plt.legend()

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

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

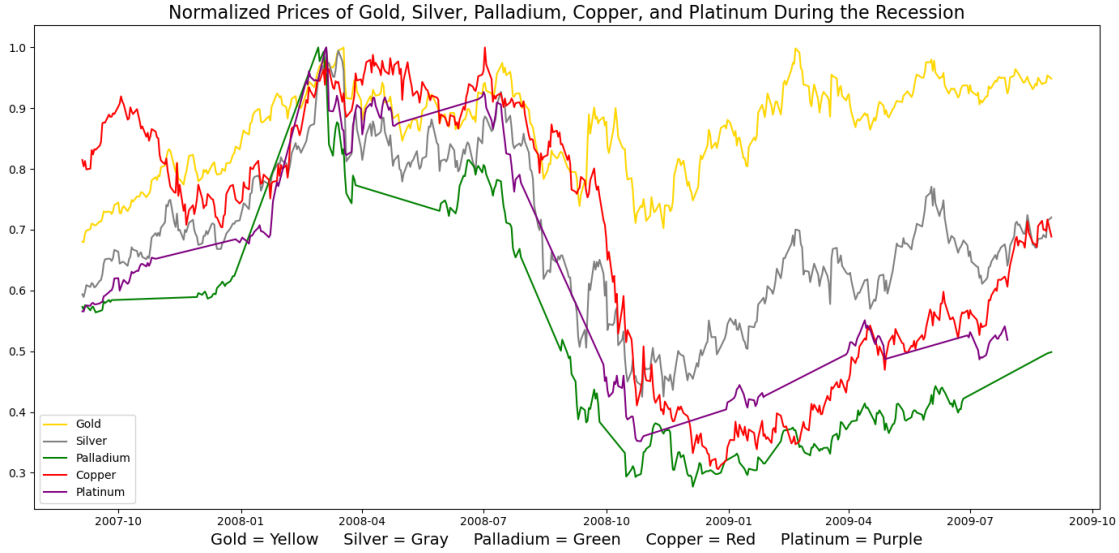


23 Great Recession

[79]:
```
start_date = '2007-09-01'
end_date = '2009-09-01'

# Define the tickers for gold, silver, palladium, copper, and platinum
gold_ticker = 'GC=F'
silver_ticker = 'SI=F'
palladium_ticker = 'PA=F'
copper_ticker = 'HG=F'
platinum_ticker = 'PL=F'

# Download historical data for gold, silver, palladium, copper, and platinum
gold_data = yf.download(gold_ticker, start=start_date, end=end_date)
silver_data = yf.download(silver_ticker, start=start_date, end=end_date)
palladium_data = yf.download(palladium_ticker, start=start_date, end=end_date)
copper_data = yf.download(copper_ticker, start=start_date, end=end_date)
platinum_data = yf.download(platinum_ticker, start=start_date, end=end_date)

# Normalize the prices
gold_norm = gold_data['Adj Close'] / gold_data['Adj Close'].max()
silver_norm = silver_data['Adj Close'] / silver_data['Adj Close'].max()
palladium_norm = palladium_data['Adj Close'] / palladium_data['Adj Close'].max()
copper_norm = copper_data['Adj Close'] / copper_data['Adj Close'].max()
platinum_norm = platinum_data['Adj Close'] / platinum_data['Adj Close'].max()

# Plot the normalized prices
plt.figure(figsize=(14, 7))
```

```
plt.plot(gold_norm, label='Gold', color='gold')
plt.plot(silver_norm, label='Silver', color='gray')
plt.plot(palladium_norm, label='Palladium', color='green')
plt.plot(copper_norm, label='Copper', color='red')
plt.plot(platinum_norm, label='Platinum', color='purple')
plt.title('Normalized Prices of Gold, Silver, Palladium, Copper, and Platinum␣
 ↪During the Recession', fontsize = 16)
plt.xlabel('Gold = Yellow     Silver = Gray     Palladium = Green     Copper =␣
 ↪Red     Platinum = Purple', fontsize = 14)
plt.legend()

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

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

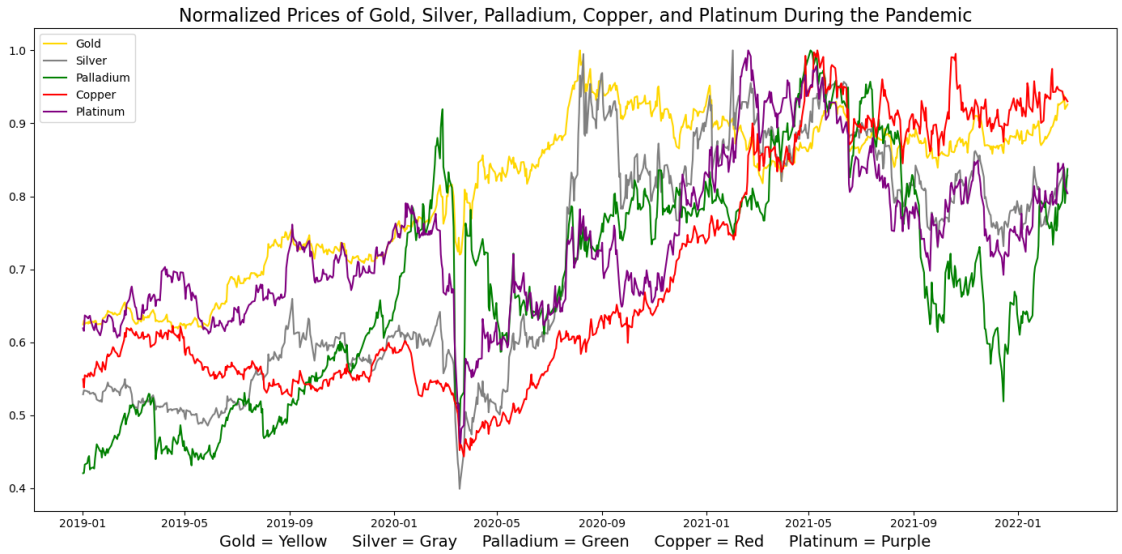


24 Pandemic

[85]:
```
start_date = '2019-01-01'
end_date = '2022-03-01'

# Define the tickers for gold, silver, palladium, copper, and platinum
gold_ticker = 'GC=F'
```

```
silver_ticker = 'SI=F'
palladium_ticker = 'PA=F'
copper_ticker = 'HG=F'
platinum_ticker = 'PL=F'

# Download historical data for gold, silver, palladium, copper, and platinum
gold_data = yf.download(gold_ticker, start=start_date, end=end_date)
silver_data = yf.download(silver_ticker, start=start_date, end=end_date)
palladium_data = yf.download(palladium_ticker, start=start_date, end=end_date)
copper_data = yf.download(copper_ticker, start=start_date, end=end_date)
platinum_data = yf.download(platinum_ticker, start=start_date, end=end_date)

# Normalize the prices
gold_norm = gold_data['Adj Close'] / gold_data['Adj Close'].max()
silver_norm = silver_data['Adj Close'] / silver_data['Adj Close'].max()
palladium_norm = palladium_data['Adj Close'] / palladium_data['Adj Close'].max()
copper_norm = copper_data['Adj Close'] / copper_data['Adj Close'].max()
platinum_norm = platinum_data['Adj Close'] / platinum_data['Adj Close'].max()

# Plot the normalized prices
plt.figure(figsize=(14, 7))
plt.plot(gold_norm, label='Gold', color='gold')
plt.plot(silver_norm, label='Silver', color='gray')
plt.plot(palladium_norm, label='Palladium', color='green')
plt.plot(copper_norm, label='Copper', color='red')
plt.plot(platinum_norm, label='Platinum', color='purple')
plt.title('Normalized Prices of Gold, Silver, Palladium, Copper, and Platinum␣
 ↪During the Pandemic', fontsize = 16)
plt.xlabel('Gold = Yellow     Silver = Gray     Palladium = Green     Copper =␣
 ↪Red     Platinum = Purple', fontsize = 14)
plt.legend()

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

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



25 Want to Look At Ratios

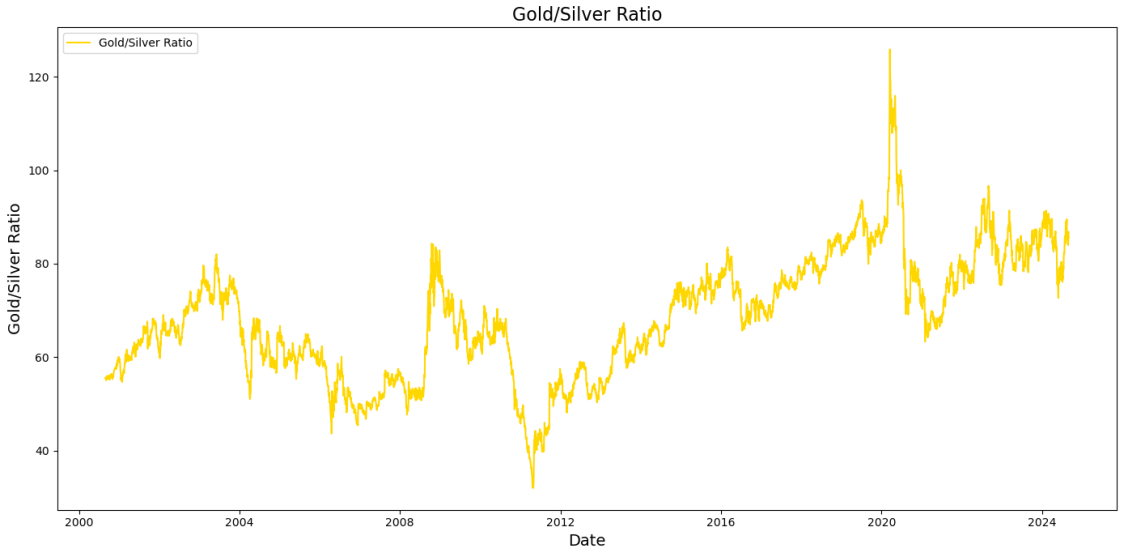
The Palladium and Platinum data is not ideal, probably because of low volume.

26 Gold/Silver Ratio

[84]:
```
# Define the tickers for gold, silver, and SPY ETF
gold_ticker = 'GC=F'
silver_ticker = 'SI=F'

# Download historical data for gold, silver, and SPY ETF
start_date = '2000-01-01'
end_date = '2024-09-01'
gold_data = yf.download(gold_ticker, start=start_date, end=end_date)
silver_data = yf.download(silver_ticker, start=start_date, end=end_date)

# Align the indices of gold and silver data
aligned_data = gold_data['Adj Close'].align(silver_data['Adj Close'],␣
 ↪join='inner')

# Calculate the gold-to-silver ratio
ratio = aligned_data[0] / aligned_data[1]

# plot the gold-to-silver ratio
fig, ax = plt.subplots(figsize=(14, 7))

# plot gold-to-silver ratio
```

```
ax.plot(ratio, label='Gold/Silver Ratio', color='gold')
ax.set_xlabel('Date', fontsize = 14)
ax.set_ylabel('Gold/Silver Ratio', fontsize = 14)

# add legend
ax.legend(loc='upper left')

# display the plot
plt.title('Gold/Silver Ratio', fontsize = 16)
plt.tight_layout()
fig.savefig('Gold-Silver Ratio.png')
plt.show()
```

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

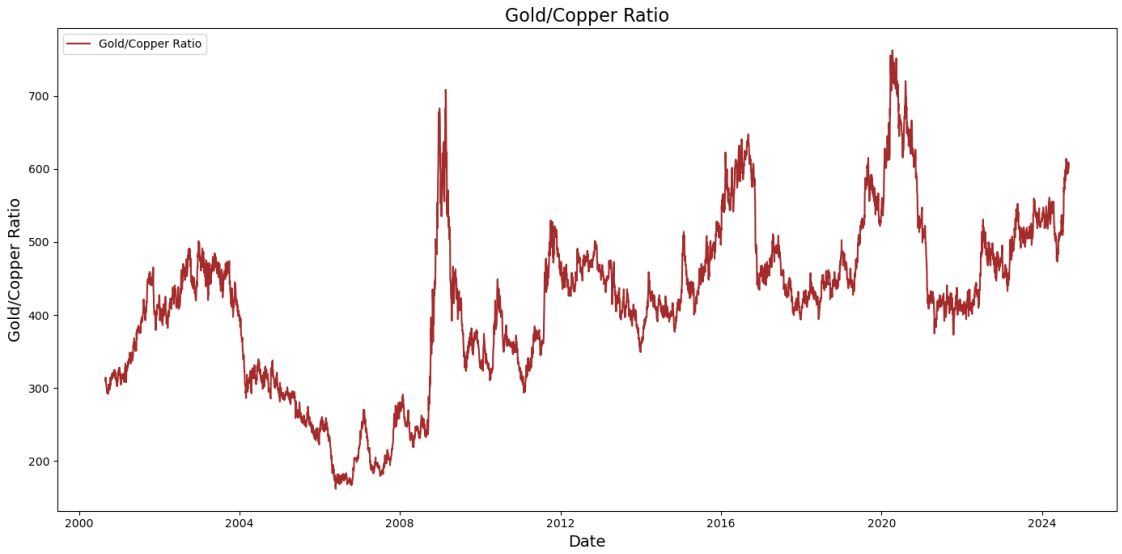


27 Gold/Copper Ratio

[88]:
```
# Define the tickers for gold and copper
gold_ticker = 'GC=F'
copper_ticker = 'HG=F'

# Download historical data for gold and copper
start_date = '2000-01-01'
end_date = '2024-09-01'
gold_data = yf.download(gold_ticker, start=start_date, end=end_date)
copper_data = yf.download(copper_ticker, start=start_date, end=end_date)
```

```
# Align the indices of gold and copper
aligned_data = gold_data['Adj Close'].align(copper_data['Adj Close'],␣
 ↪join='inner')

# Calculate the gold-to-copper ratio
ratio = aligned_data[0] / aligned_data[1]

# plot the gold-to-copper ratio
fig, ax = plt.subplots(figsize=(14, 7))

# plot gold-to-copper ratio
ax.plot(ratio, label='Gold/Copper Ratio', color='brown')
ax.set_xlabel('Date', fontsize = 14)
ax.set_ylabel('Gold/Copper Ratio', fontsize = 14)

# add legend
ax.legend(loc='upper left')

# display the plot
plt.title('Gold/Copper Ratio', fontsize = 16)
plt.tight_layout()
fig.savefig('Gold-Copper Ratio.png')
plt.show()
```

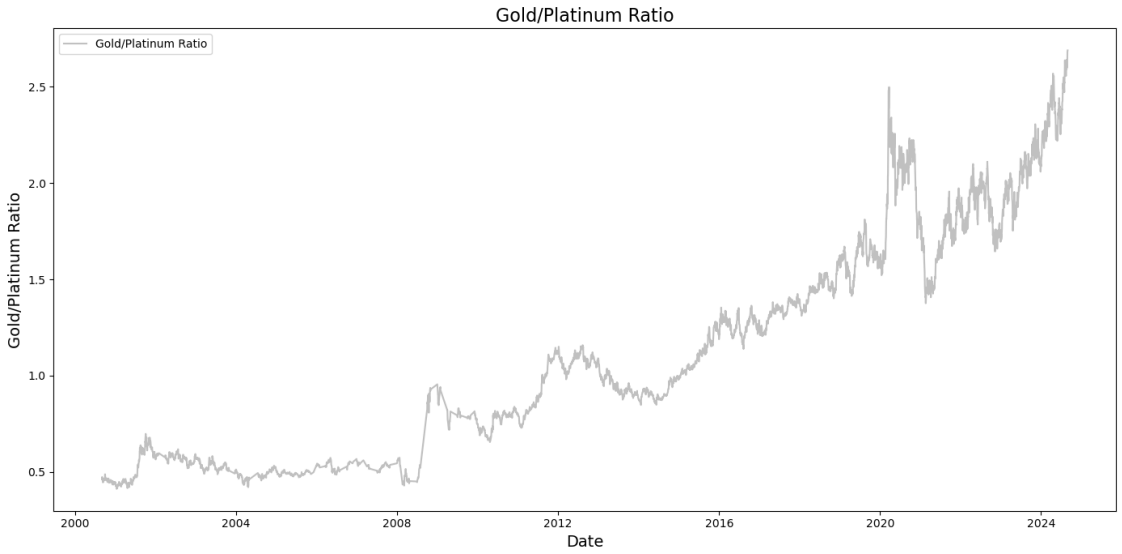
```
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
```



28 Gold/Platinum Ratio

[91]:
```
# Define the tickers for gold and platinum
gold_ticker = 'GC=F'
platinum_ticker = 'PL=F'

# Download historical data for gold and platinum
start_date = '2000-01-01'
end_date = '2024-09-01'
gold_data = yf.download(gold_ticker, start=start_date, end=end_date)
platinum_data = yf.download(platinum_ticker, start=start_date, end=end_date)

# Align the indices of gold and platinum data
aligned_data = gold_data['Adj Close'].align(platinum_data['Adj Close'],␣
 ↪join='inner')

# Calculate the gold-to-platinum ratio
ratio = aligned_data[0] / aligned_data[1]

# plot the gold-to-silver ratio
fig, ax = plt.subplots(figsize=(14, 7))

# plot gold-to-silver ratio
ax.plot(ratio, label='Gold/Platinum Ratio', color='silver')
ax.set_xlabel('Date', fontsize = 14)
ax.set_ylabel('Gold/Platinum Ratio', fontsize = 14)

# add legend
ax.legend(loc='upper left')

# display the plot
plt.title('Gold/Platinum Ratio', fontsize = 16)
plt.tight_layout()
fig.savefig('Gold-Platinum Ratio.png')
plt.show()
```

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

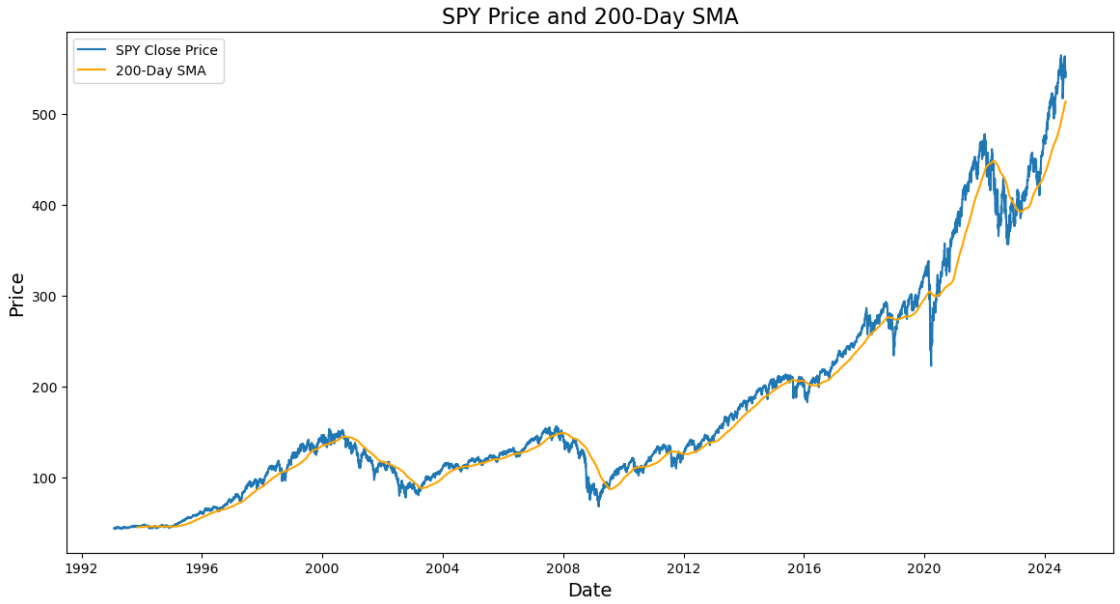


There always seems to be a flock to gold away from silver, copper, and platinum in a recession. The same is also seen in the Gold/BTC ratio.

29 Last Ideas - Technicals

[142]:
```
# fetch SPY data
spy = yf.download('SPY', start='1993-01-29')  # SPY started trading on Jan 29,␣
 ↪1993

# calculate the 200-day SMA
spy['SMA200'] = spy['Close'].rolling(window = 200).mean()

# plot SPY close price and 200-day SMA
plt.figure(figsize = (14, 7))
plt.plot(spy.index, spy['Close'], label = 'SPY Close Price')
plt.plot(spy.index, spy['SMA200'], label = '200-Day SMA', color = 'orange')
plt.title('SPY Price and 200-Day SMA', fontsize = 16)
plt.xlabel('Date', fontsize = 14)
plt.ylabel('Price', fontsize = 14)
plt.legend()
plt.show()
```

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



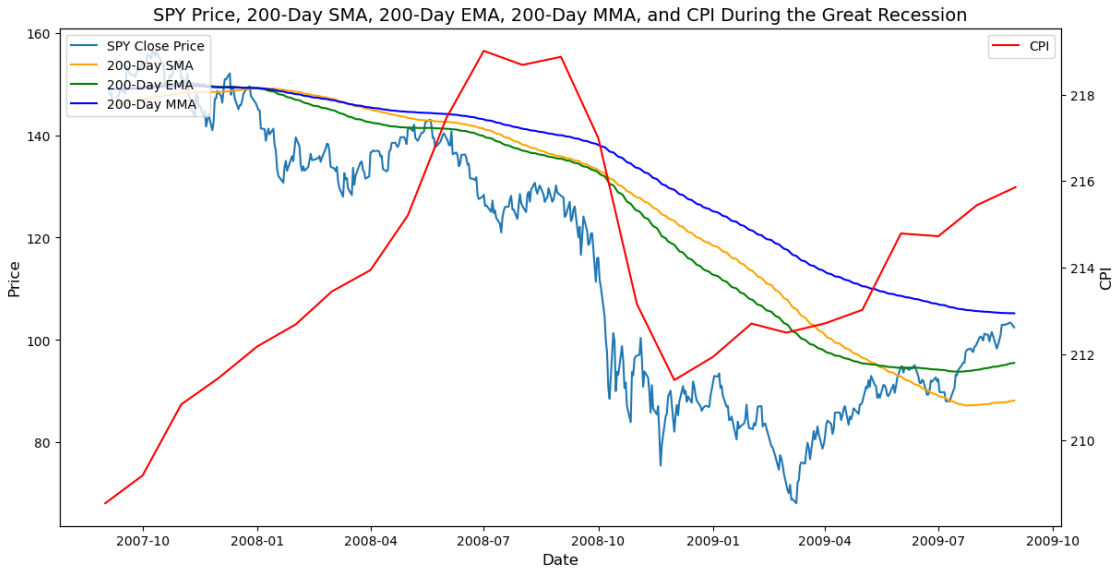
[143]:
```
api_key = '876abb970a5e1cba77291fd327ce14e9' # new
```

30 Great Recession

[149]:
```
# fetch SPY data from 2006 to 2009 to cover the full range for the 200-day SMA␣
 ↪calculation
spy = yf.download('SPY', start='2006-01-01', end='2009-09-01')

# calculate the 200-day SMA
spy['SMA200'] = spy['Close'].rolling(window = 200).mean()

# subset data for the Great Recession period
start_date = '2007-09-01'
end_date = '2009-09-01'
spy_subset = spy.loc[start_date:end_date].copy()

# calculate the 200-day EMA and MMA
spy_subset.loc[:, 'EMA200'] = spy_subset['Close'].ewm(span = 200, adjust =␣
 ↪False).mean()

n = 200
mma = []
for i in range(len(spy_subset)):
    if i == 0:
        mma.append(spy_subset['Close'].iloc[i])  # initial value is the first␣
 ↪price
```

```
    else:
        previous_mma = mma[-1]
        current_price = spy_subset['Close'].iloc[i]
        new_mma = ((n - 1) * previous_mma + current_price) / n
        mma.append(new_mma)
spy_subset.loc[:, 'MMA200'] = mma

# fetch CPI data from FRED
cpi = fred.get_series('CPIAUCSL', start_date, end_date)
cpi = cpi.resample('D').interpolate(method = 'linear')  # resample to daily and␣
 ↪interpolate

# plot SPY close price, 200-day SMA, 200-day EMA, and CPI
fig, ax1 = plt.subplots(figsize = (14, 7))

ax1.plot(spy_subset.index, spy_subset['Close'], label = 'SPY Close Price')
ax1.plot(spy_subset.index, spy_subset['SMA200'], label = '200-Day SMA', color =␣
 ↪'orange')
ax1.plot(spy_subset.index, spy_subset['EMA200'], label = '200-Day EMA', color =␣
 ↪'green')
ax1.plot(spy_subset.index, spy_subset['MMA200'], label = '200-Day MMA', color =␣
 ↪'blue')
ax1.set_xlabel('Date', fontsize = 12)
ax1.set_ylabel('Price', fontsize = 12)
ax1.legend(loc = 'upper left')

# create a secondary y-axis for CPI
ax2 = ax1.twinx()
ax2.plot(cpi.index, cpi, label = 'CPI', color = 'red')
ax2.set_ylabel('CPI', fontsize = 12)
ax2.legend(loc = 'upper right')

plt.title('SPY Price, 200-Day SMA, 200-Day EMA, 200-Day MMA, and CPI During the␣
 ↪Great Recession', fontsize = 14)
plt.show()
```

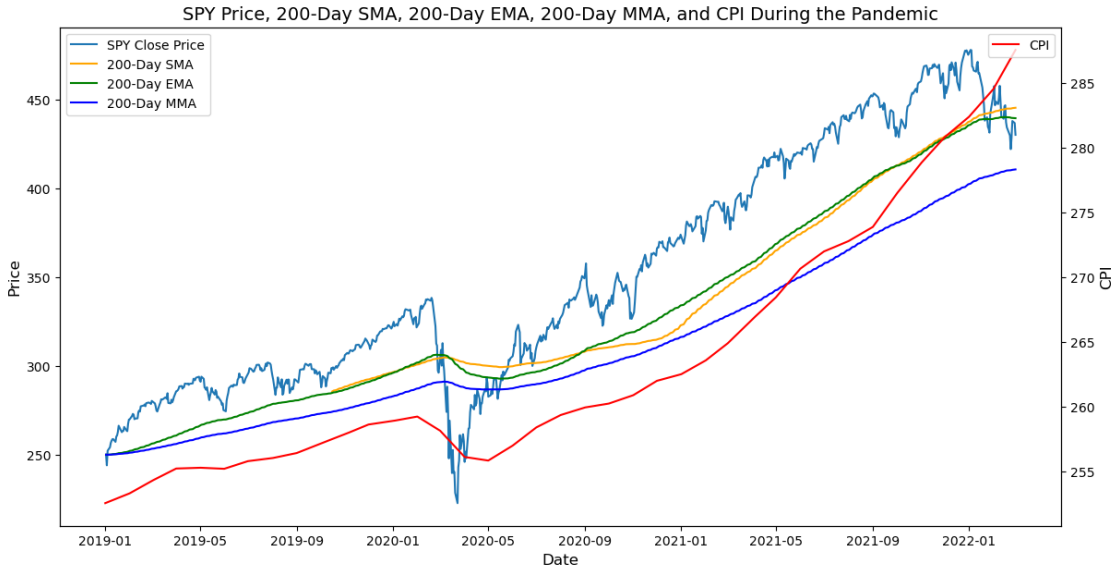
```
[**********100%%**********]  1 of 1 completed
```



31 Pandemic

[147]:
```
start_date = '2019-01-01'
end_date = '2022-03-01'
spy_subset = spy.loc[start_date:end_date].copy()

# calculate 200-day simple moving average (SMA)
spy_subset.loc[:, 'SMA200'] = spy_subset['Close'].rolling(window = 200).mean()

# calculate 200-day exponential moving average (EMA)
spy_subset.loc[:, 'EMA200'] = spy_subset['Close'].ewm(span = 200, adjust =␣
 ↪False).mean()

# calculate 200-day modified moving average (MMA)
n = 200
mma = []
for i in range(len(spy_subset)):
    if i == 0:
        mma.append(spy_subset['Close'].iloc[i])  # initial value is the first␣
 ↪price
    else:
        previous_mma = mma[-1]
        current_price = spy_subset['Close'].iloc[i]
        new_mma = ((n - 1) * previous_mma + current_price) / n
        mma.append(new_mma)
spy_subset.loc[:, 'MMA200'] = mma
```

```
# fetch CPI data from FRED
fred = Fred(api_key = api_key)
cpi = fred.get_series('CPIAUCSL', start_date, end_date)
cpi = cpi.resample('D').interpolate(method = 'linear')  # resample to daily and interpolate

# plot SPY close price, 200-day SMA, 200-day EMA, and CPI
fig, ax1 = plt.subplots(figsize = (14, 7))

ax1.plot(spy_subset.index, spy_subset['Close'], label = 'SPY Close Price')
ax1.plot(spy_subset.index, spy_subset['SMA200'], label = '200-Day SMA', color = 'orange')
ax1.plot(spy_subset.index, spy_subset['EMA200'], label = '200-Day EMA', color = 'green')
ax1.plot(spy_subset.index, spy_subset['MMA200'], label = '200-Day MMA', color = 'blue')
ax1.set_xlabel('Date', fontsize = 12)
ax1.set_ylabel('Price', fontsize = 12)
ax1.legend(loc = 'upper left')

# create a secondary y-axis for CPI
ax2 = ax1.twinx()
ax2.plot(cpi.index, cpi, label = 'CPI', color = 'red')
ax2.set_ylabel('CPI', fontsize = 12)
ax2.legend(loc = 'upper right')

plt.title('SPY Price, 200-Day SMA, 200-Day EMA, 200-Day MMA, and CPI During the Pandemic', fontsize = 14)
plt.show()
```



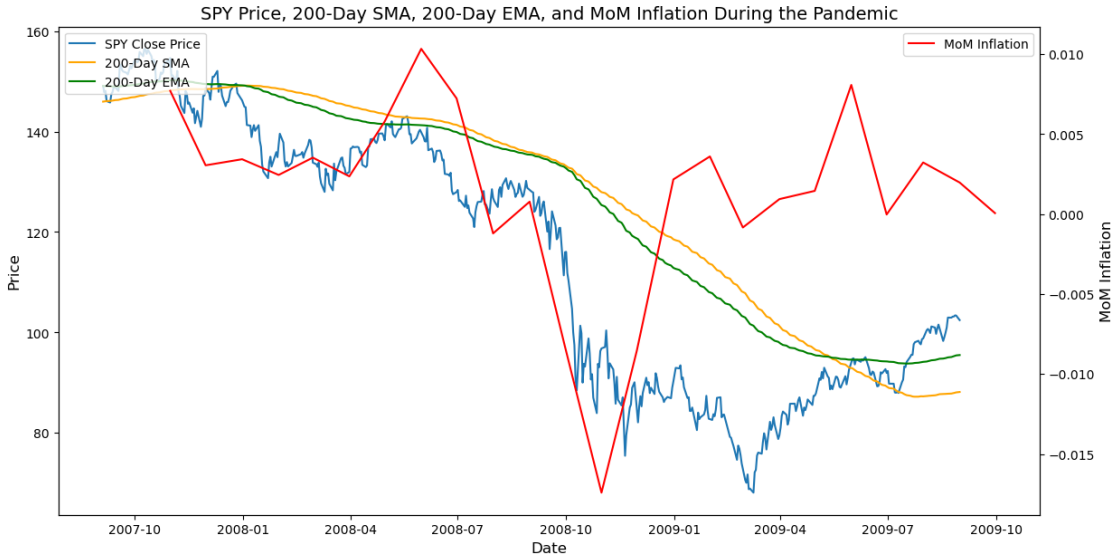
32 Modified Moving Average (MMA)

The Modified Moving Average (MMA) is calculated using the following recursive formula:

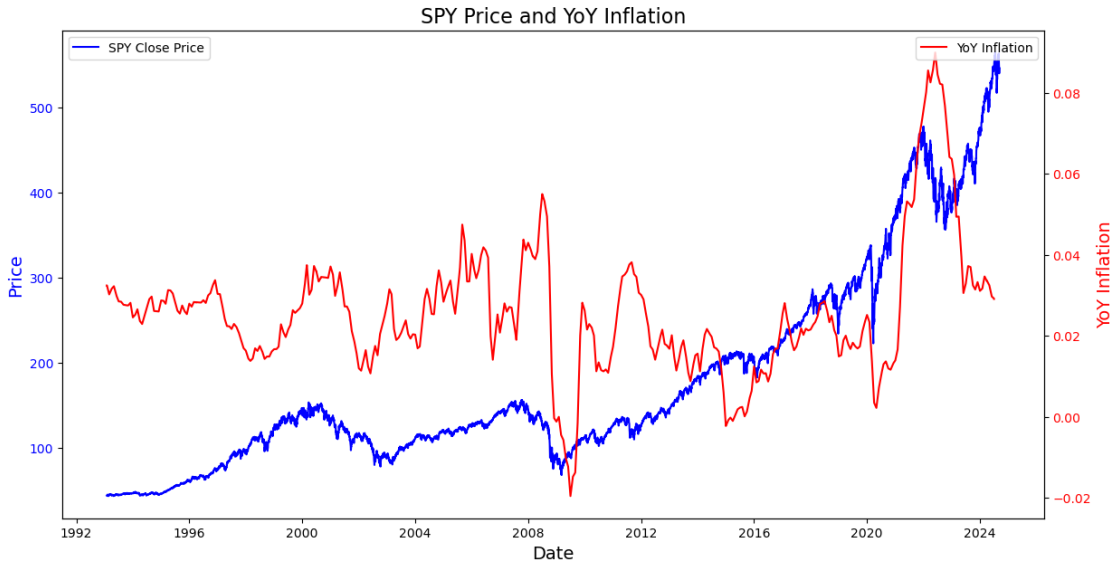
$$\text{MMA}_t = \frac{(n-1) \cdot \text{MMA}_{t-1} + P_t}{n}$$

where: - MMA_t is the modified moving average at time t - MMA_{t-1} is the modified moving average at time t-1 - P_t is the price at time t - n is the period of the moving average

[152]:
```
# fetch CPI data from FRED
fred = Fred(api_key = api_key)
cpi = fred.get_series('CPIAUCSL', start_date, end_date)
cpi = cpi.resample('D').interpolate(method = 'linear')  # resample to daily and␣
 ↪interpolate

# calculate MoM inflation
cpi_monthly = cpi.resample('M').last()
mom_inflation = cpi_monthly.pct_change().dropna()

# plot SPY close price, 200-day SMA, 200-day EMA, and MoM inflation
fig, ax1 = plt.subplots(figsize = (14, 7))

ax1.plot(spy_subset.index, spy_subset['Close'], label = 'SPY Close Price')
ax1.plot(spy_subset.index, spy_subset['SMA200'], label = '200-Day SMA', color =␣
 ↪'orange')
ax1.plot(spy_subset.index, spy_subset['EMA200'], label = '200-Day EMA', color =␣
 ↪'green')
ax1.set_xlabel('Date', fontsize = 12)
ax1.set_ylabel('Price', fontsize = 12)
ax1.legend(loc = 'upper left')

# create a secondary y-axis for MoM inflation
ax2 = ax1.twinx()
ax2.plot(mom_inflation.index, mom_inflation, label = 'MoM Inflation', color =␣
 ↪'red')
ax2.set_ylabel('MoM Inflation', fontsize = 12)
ax2.legend(loc = 'upper right')

plt.title('SPY Price, 200-Day SMA, 200-Day EMA, and MoM Inflation During the␣
 ↪Pandemic', fontsize = 14)
plt.show()
```



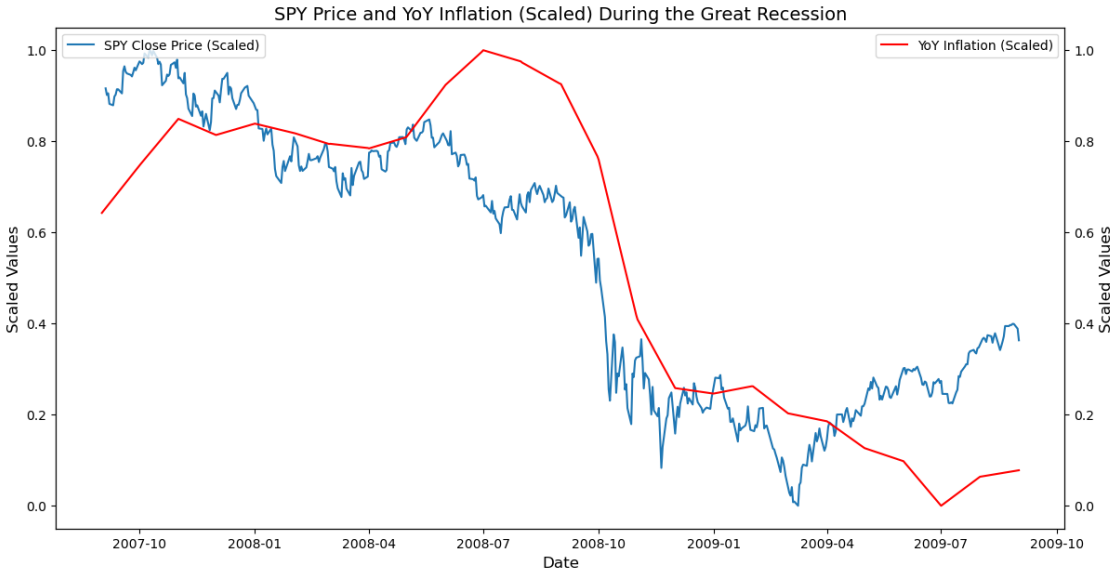
[156]:
```
# plot SPY close price and YoY inflation with color-coded axes
fig, ax1 = plt.subplots(figsize = (14, 7))

# plot SPY close price on the first axis (blue)
ax1.plot(spy.index, spy['Close'], label = 'SPY Close Price', color = 'blue')
ax1.set_xlabel('Date', fontsize = 14)
ax1.set_ylabel('Price', fontsize = 14, color = 'blue')
ax1.tick_params(axis = 'y', labelcolor = 'blue')
ax1.legend(loc = 'upper left')

# create a secondary y-axis for YoY inflation (red)
ax2 = ax1.twinx()
ax2.plot(yoy_inflation.index, yoy_inflation, label = 'YoY Inflation', color =␣
↪'red')
ax2.set_ylabel('YoY Inflation', fontsize = 14, color = 'red')
ax2.tick_params(axis = 'y', labelcolor = 'red')
ax2.legend(loc = 'upper right')

plt.title('SPY Price and YoY Inflation', fontsize = 16)
plt.show()
```



33 Min-Max Transormation During the Great Recession

[97]:
```
# define the Great Recession period
recession_start = '2007-09-01'
recession_end = '2009-09-01'

# subset data to the Great Recession period
spy_recession = spy.loc[recession_start:recession_end]
yoy_inflation_recession = yoy_inflation.loc[recession_start:recession_end]

# perform min-max transformation
scaler = MinMaxScaler()
spy_recession_scaled = scaler.fit_transform(spy_recession[['Close']])
yoy_inflation_recession_scaled = scaler.fit_transform(yoy_inflation_recession.
 ↪values.reshape(-1, 1))

# plot scaled SPY close price and YoY inflation during the Great Recession
fig, ax1 = plt.subplots(figsize = (14, 7))

ax1.plot(spy_recession.index, spy_recession_scaled, label = 'SPY Close Price␣
 ↪(Scaled)')
ax1.set_xlabel('Date', fontsize = 12)
ax1.set_ylabel('Scaled Values', fontsize = 12)
ax1.legend(loc = 'upper left')

# create a secondary y-axis for scaled YoY inflation
ax2 = ax1.twinx()
```

```
ax2.plot(yoy_inflation_recession.index, yoy_inflation_recession_scaled, label =␣
 ↪'YoY Inflation (Scaled)', color = 'red')
ax2.set_ylabel('Scaled Values', fontsize = 12)
ax2.legend(loc = 'upper right')

plt.title('SPY Price and YoY Inflation (Scaled) During the Great Recession',␣
 ↪fontsize = 14)
plt.show()
```

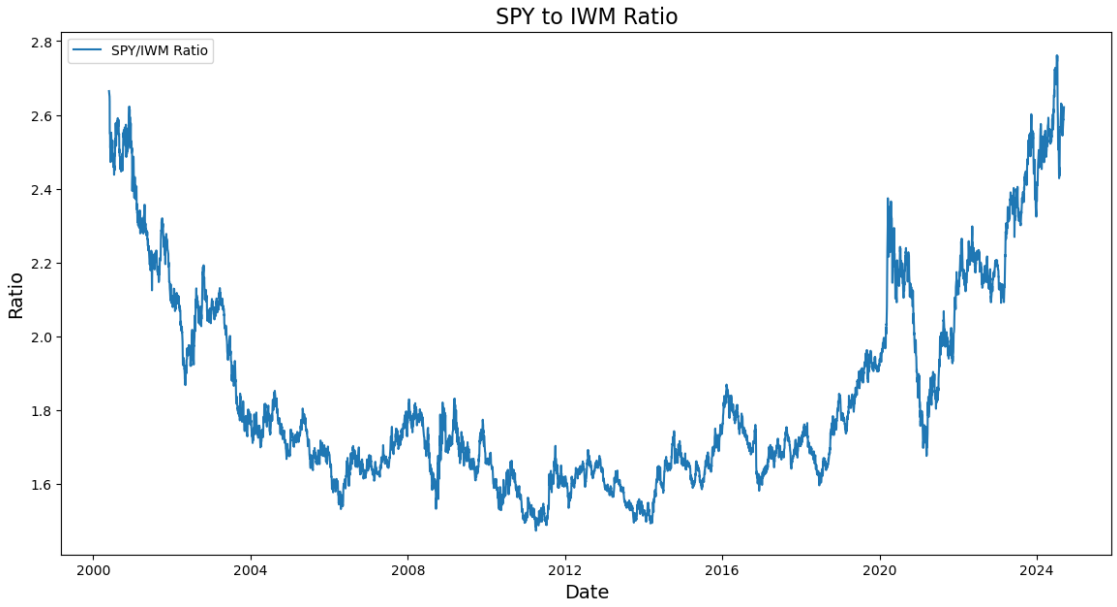


34 The Rotation from Large to Small (Recently)

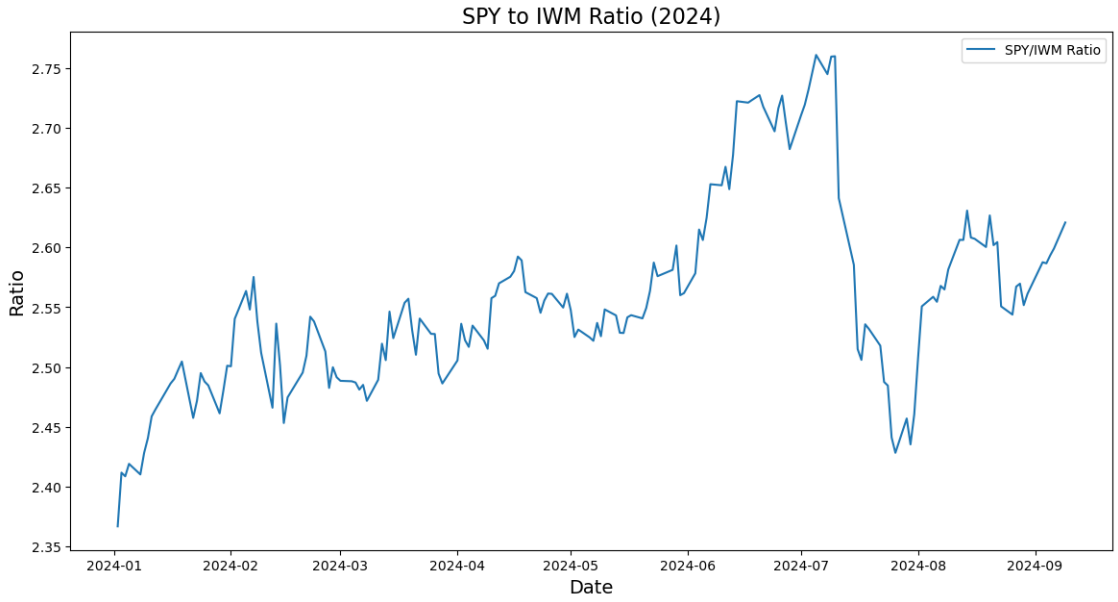
[101]:
```
# fetch SPY and IWM data
spy = yf.download('SPY', start='1993-01-29')['Adj Close']
iwm = yf.download('IWM', start='2000-05-26')['Adj Close']  # IWM started␣
 ↪trading on May 26, 2000

# align the data to the same date range
data = pd.DataFrame({'SPY': spy, 'IWM': iwm}).dropna()

# calculate the ratio of SPY to IWM
data['SPY/IWM'] = data['SPY'] / data['IWM']

# plot the ratio
plt.figure(figsize=(14, 7))
plt.plot(data.index, data['SPY/IWM'], label='SPY/IWM Ratio')
plt.xlabel('Date', fontsize = 14)
```

```
plt.ylabel('Ratio', fontsize = 14)
plt.title('SPY to IWM Ratio', fontsize = 16)
plt.legend()
plt.show()
```

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



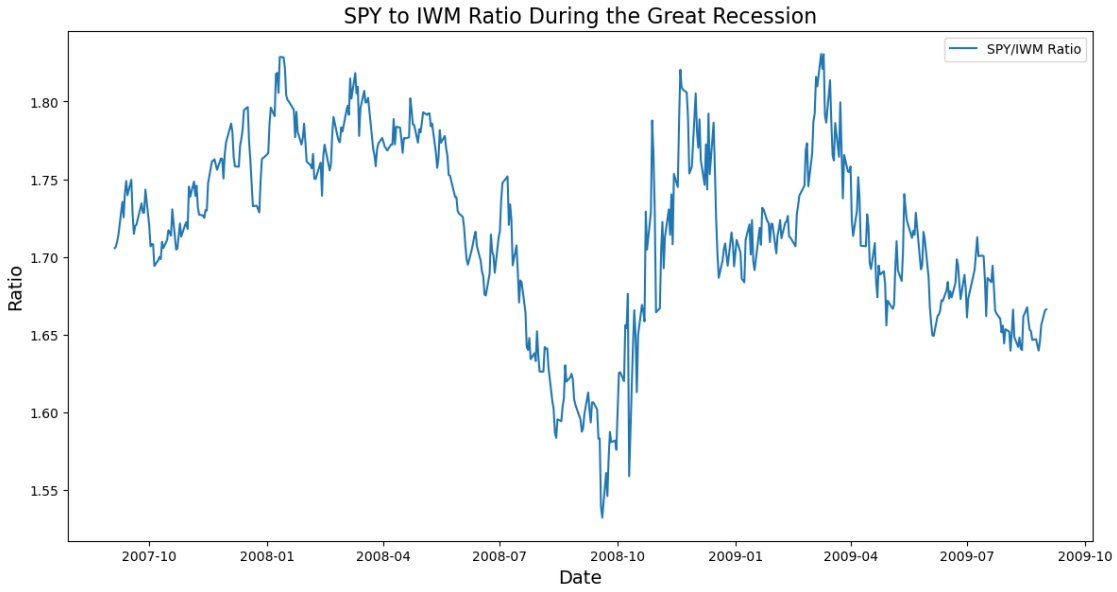
[102]:
```
# define the date range for subsetting
start = '2024-01-01'
end = '2024-09-09'

# subset the data to the specified date range
rotation = data.loc[start:end, 'SPY/IWM']

# plot the ratio
plt.figure(figsize = (14, 7))
plt.plot(rotation.index, rotation, label = 'SPY/IWM Ratio')
plt.xlabel('Date', fontsize = 14)
plt.ylabel('Ratio', fontsize = 14)
plt.title('SPY to IWM Ratio (2024)', fontsize = 16)
plt.legend()
plt.show()
```



We are pushing back up towards all-time highs.

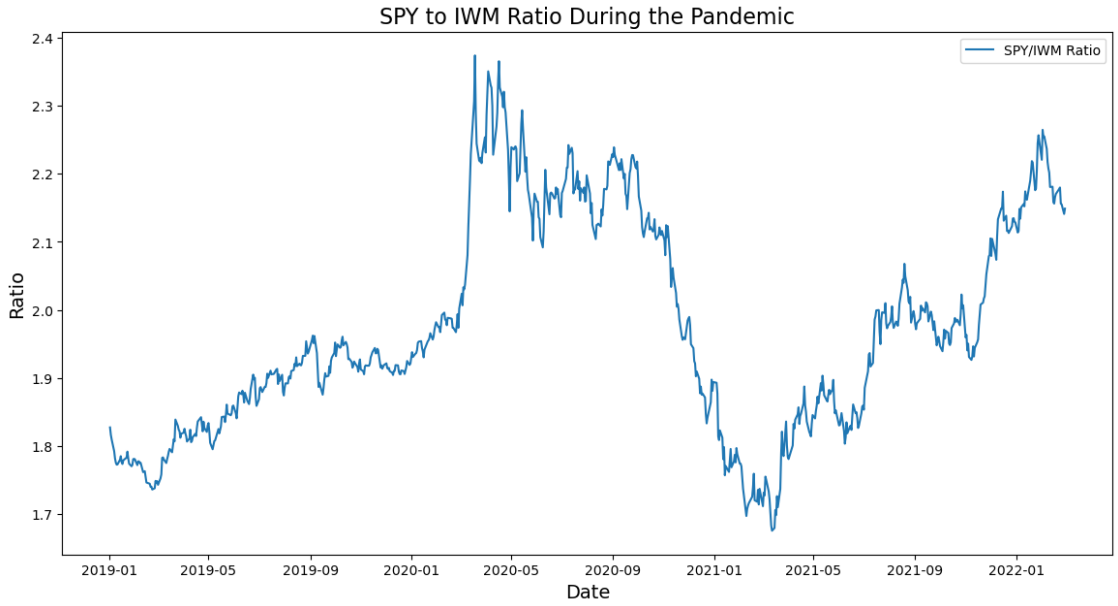
35 Great Recession

[103]:
```
recession_start = '2007-09-01'
recession_end = '2009-09-01'

# subset the data to the specified date range
rotation = data.loc[recession_start:recession_end, 'SPY/IWM']

# plot the ratio
plt.figure(figsize = (14, 7))
plt.plot(rotation.index, rotation, label = 'SPY/IWM Ratio')
plt.xlabel('Date', fontsize = 14)
plt.ylabel('Ratio', fontsize = 14)
plt.title('SPY to IWM Ratio During the Great Recession', fontsize = 16)
plt.legend()
plt.show()
```



36 Pandemic

[104]:
```
pandemic_start = '2019-01-01'
pandemic_end = '2022-03-01'

# subset the data to the specified date range
rotation = data.loc[pandemic_start:pandemic_end, 'SPY/IWM']

# plot the ratio
plt.figure(figsize = (14, 7))
plt.plot(rotation.index, rotation, label = 'SPY/IWM Ratio')
plt.xlabel('Date', fontsize = 14)
plt.ylabel('Ratio', fontsize = 14)
plt.title('SPY to IWM Ratio During the Pandemic', fontsize = 16)
plt.legend()
plt.show()
```



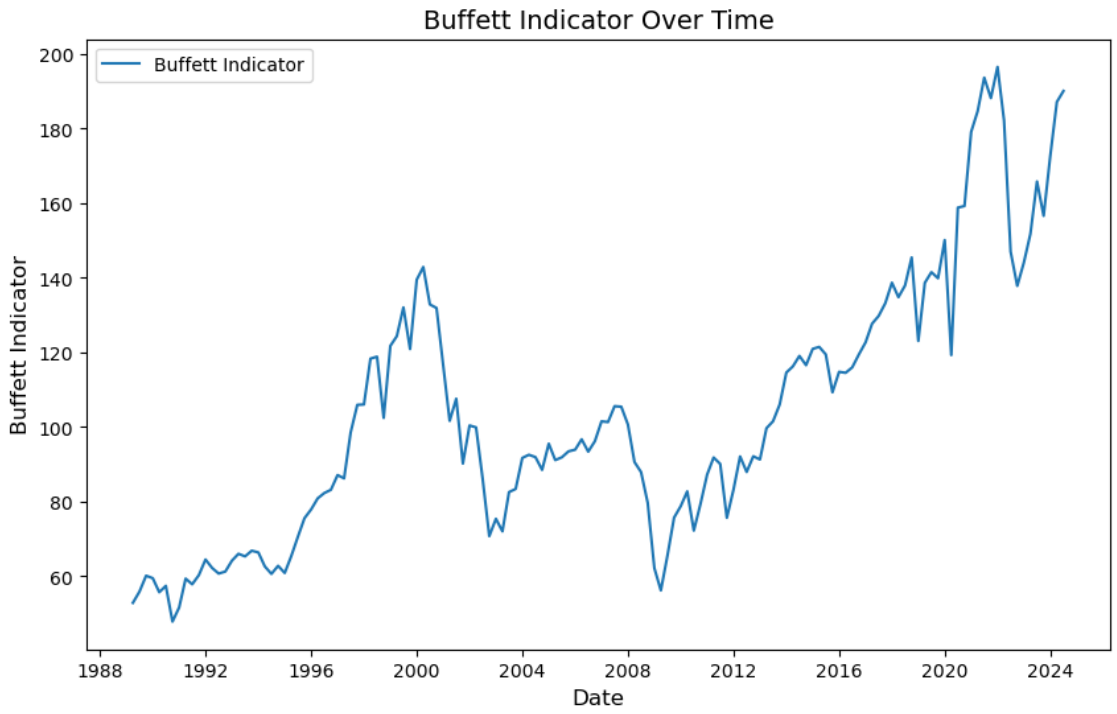
37 Buffet Indicator

[9]:
```
api_key = '876abb970a5e1cba77291fd327ce14e9' # new
```

[100]:
```
fred = Fred(api_key=api_key)

# download data for the Wilshire 5000 index from the earliest available date
wilshire5000 = yf.download('^W5000', start='1970-01-01', end='2024-09-01',␣
 ↪interval='1mo')

# download GDP data from FRED starting as early as possible
gdp = fred.get_series('GDP', observation_start='1970-01-01')

# resample GDP data to quarterly frequency and forward fill
gdp = gdp.resample('Q').ffill()

# align Wilshire 5000 data to the quarterly GDP data
wilshire5000 = wilshire5000.resample('Q').ffill()

# trim the data to the overlapping date range
common_index = wilshire5000.index.intersection(gdp.index)
wilshire5000 = wilshire5000.loc[common_index]
gdp = gdp.loc[common_index]

# calculate the Buffett Indicator
buffett_indicator = (wilshire5000['Adj Close'] / gdp) * 100
```

```
# plot the Buffett Indicator
plt.figure(figsize=(10, 6))
plt.plot(buffett_indicator, label='Buffett Indicator')
plt.title('Buffett Indicator Over Time', fontsize = 14)
plt.xlabel('Date', fontsize = 12)
plt.ylabel('Buffett Indicator', fontsize = 12)
plt.legend()
plt.show()
```

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



38 6-Month Sahm Rule

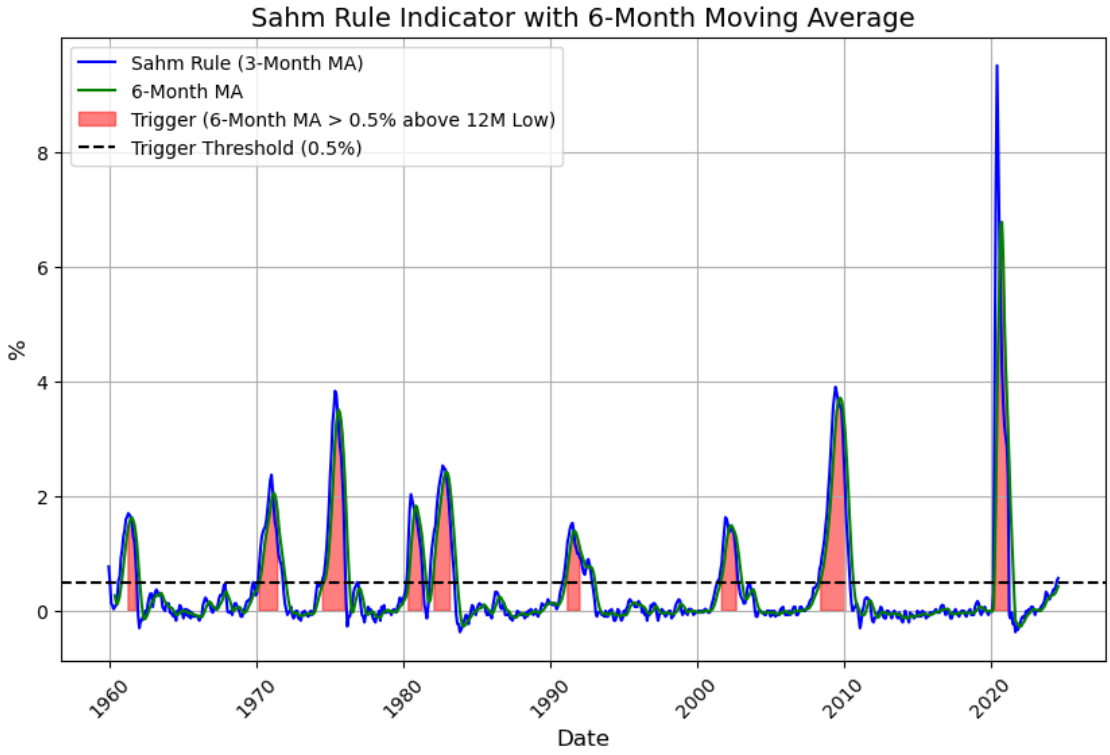
Below you will see a link to a website that shows the % of states where the Sahm rule has triggered with different thresholds. 0.5 is the traditional threshold. 0.75 would mean a more significant uptick in unemployment, so this will naturally include fewer states.

Even though the Sahm rule has triggered, our Strategas research indicates that historically, a different calculation has actually been a validating factor for the recession, and that has not triggered as of August 2024. This unique rule takes into consideration changes in the labor force over time. A big driver over the last 70 years or so has been the male and female labor force composition.

https://en.macromicro.me/charts/102426/US-of-States-Where-the-Sahm-Rule-Has-Been-Triggered

I wanted to do a little research here on my own, so I calculated a 6-month moving average version of the Sahm Rule. Please have Paul check over this. Is there anything significant here?

[110]:
```
fred = Fred(api_key=api_key)

# get the Sahm Rule data (3-month MA unemployment rate)
sahm_rule = fred.get_series('SAHMREALTIME')

# calculate the 6-month moving average based on the 3-month data
sahm_rule_ma6 = sahm_rule.rolling(window=6).mean()

# calculate the rolling 12-month minimum of the 6-month MA unemployment rate
rolling_min_12m = sahm_rule_ma6.rolling(window=12).min()

# calculate when the 6-month MA is 0.5 percentage points above the rolling␣
 ↪12-month minimum
trigger = (sahm_rule_ma6 - rolling_min_12m) >= 0.5

# plot the original Sahm Rule data and the 6-month moving average
plt.figure(figsize=(10, 6))
plt.plot(sahm_rule.index, sahm_rule, label='Sahm Rule (3-Month MA)',␣
 ↪color='blue')
plt.plot(sahm_rule_ma6.index, sahm_rule_ma6, label='6-Month MA', color='green')

# highlight in red where the 6-month moving average exceeds the 12-month␣
 ↪minimum by 0.5 percentage points
plt.fill_between(sahm_rule_ma6.index, sahm_rule_ma6, where=trigger,␣
 ↪color='red', alpha=0.5, label='Trigger (6-Month MA > 0.5% above 12M Low)')

# add labels, title, and grid
plt.title('Sahm Rule Indicator with 6-Month Moving Average', fontsize=14)
plt.xlabel('Date', fontsize=12)
plt.ylabel('%', fontsize=12)
plt.axhline(y=0.5, color='black', linestyle='--', label='Trigger Threshold (0.
 ↪5%)')
plt.legend()

# format the x-axis with dates
plt.xticks(rotation=45)
plt.grid(True)

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

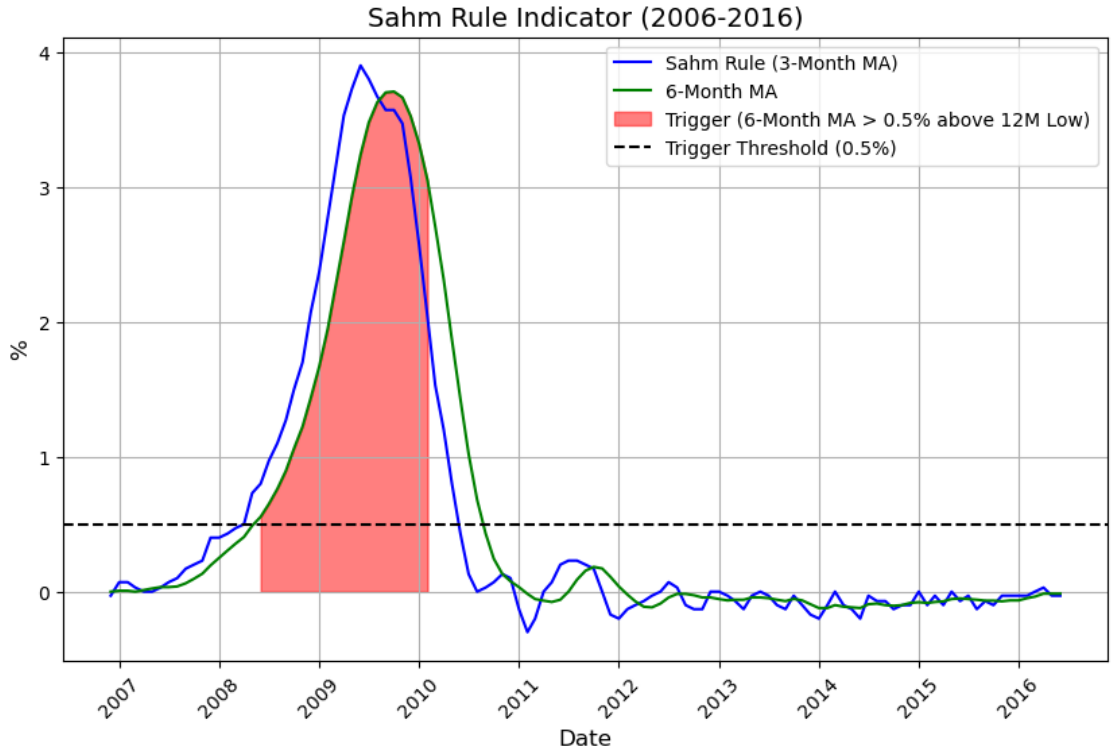


An important note is that this version of the Sahm Rule has also not triggered. We are in unique times, but also, it is a wider moving average, so some people might just say that it is taking more time to show up. Also, the trigger threshold seen in these graphs is there just for an approximation. The technical measure of the Sahm Rule is the change in the moving average from the 12-month minimum, so exceeding the threshold does not necessarily mean the rule has triggered.

39 Zooming In

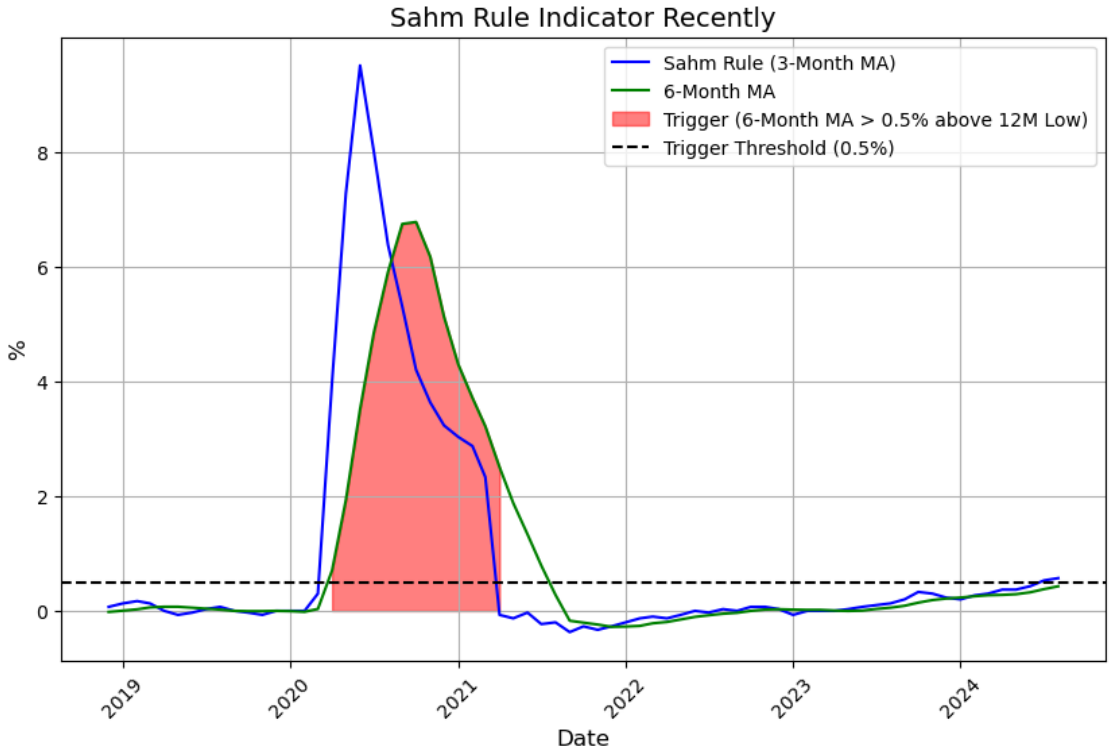
[16]:
```
# want to get a closer look at a narrower time period
recession_period = (sahm_rule.index >= '2006-12-01') & (sahm_rule.index <=␣
 ↪'2016-06-30')

# plot for the 2008 Recession
plt.figure(figsize=(10, 6))
plt.plot(sahm_rule.index[recession_period], sahm_rule[recession_period],␣
 ↪label='Sahm Rule (3-Month MA)', color='blue')
plt.plot(sahm_rule_ma6.index[recession_period],␣
 ↪sahm_rule_ma6[recession_period], label='6-Month MA', color='green')

# highlight in red where the 6-month moving average exceeds the 12-month␣
 ↪minimum by 0.5 percentage points
```

```
plt.fill_between(sahm_rule_ma6.index[recession_period],␣
  ↪sahm_rule_ma6[recession_period], where=trigger[recession_period],␣
  ↪color='red', alpha=0.5, label='Trigger (6-Month MA > 0.5% above 12M Low)')

# add labels, title, and grid
plt.title('Sahm Rule Indicator (2006-2016)', fontsize=14)
plt.xlabel('Date', fontsize=12)
plt.ylabel('%', fontsize=12)
plt.axhline(y=0.5, color='black', linestyle='--', label='Trigger Threshold (0.
  ↪5%)')
plt.legend()

# format the x-axis with dates
plt.xticks(rotation=45)
plt.grid(True)

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



40 Zooming Into The Last Few Years - Chance of It Triggering?

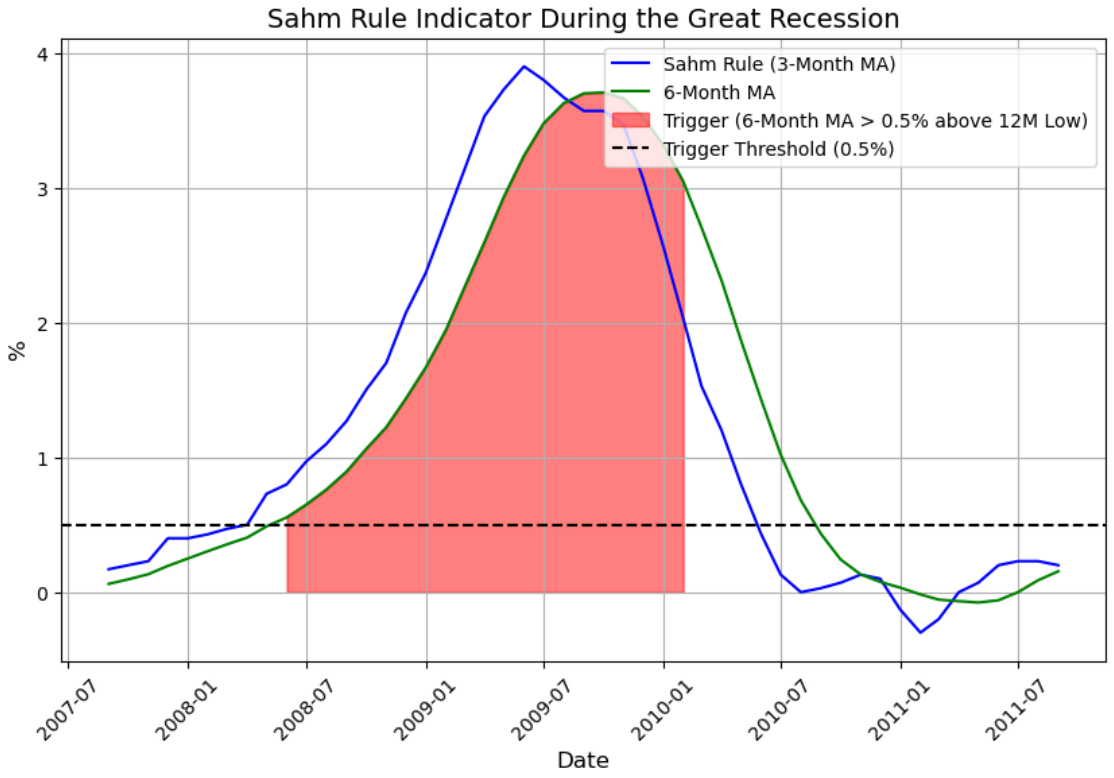
[17]:
```
# recent data
recession_period = (sahm_rule.index >= '2018-12-01') & (sahm_rule.index <=␣
 ↪'2024-09-01')

# plot for the 2008 Recession
plt.figure(figsize=(10, 6))
plt.plot(sahm_rule.index[recession_period], sahm_rule[recession_period],␣
 ↪label='Sahm Rule (3-Month MA)', color='blue')
plt.plot(sahm_rule_ma6.index[recession_period],␣
 ↪sahm_rule_ma6[recession_period], label='6-Month MA', color='green')

# highlight in red where the 6-month moving average exceeds the 12-month␣
 ↪minimum by 0.5 percentage points
plt.fill_between(sahm_rule_ma6.index[recession_period],␣
 ↪sahm_rule_ma6[recession_period], where=trigger[recession_period],␣
 ↪color='red', alpha=0.5, label='Trigger (6-Month MA > 0.5% above 12M Low)')

# add labels, title, and grid
plt.title('Sahm Rule Indicator Recently', fontsize=14)
plt.xlabel('Date', fontsize=12)
plt.ylabel('%', fontsize=12)
plt.axhline(y=0.5, color='black', linestyle='--', label='Trigger Threshold (0.
 ↪5%)')
plt.legend()

# format the x-axis with dates
plt.xticks(rotation=45)
plt.grid(True)

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



41 Great Recession

[109]:
```
recession_period = (sahm_rule.index >= '2007-09-01') & (sahm_rule.index <=␣
 ↪'2011-09-01')

# plot for the 2008 Recession
plt.figure(figsize=(10, 6))
plt.plot(sahm_rule.index[recession_period], sahm_rule[recession_period],␣
 ↪label='Sahm Rule (3-Month MA)', color='blue')
plt.plot(sahm_rule_ma6.index[recession_period],␣
 ↪sahm_rule_ma6[recession_period], label='6-Month MA', color='green')

# highlight in red where the 6-month moving average exceeds the 12-month␣
 ↪minimum by 0.5 percentage points
plt.fill_between(sahm_rule_ma6.index[recession_period],␣
 ↪sahm_rule_ma6[recession_period], where=trigger[recession_period],␣
 ↪color='red', alpha=0.5, label='Trigger (6-Month MA > 0.5% above 12M Low)')

# add labels, title, and grid
plt.title('Sahm Rule Indicator During the Great Recession', fontsize=14)
plt.xlabel('Date', fontsize=12)
```

```
plt.ylabel('%', fontsize=12)
plt.axhline(y=0.5, color='black', linestyle='--', label='Trigger Threshold (0.
  ↪5%)')
plt.legend()

# format the x-axis with dates
plt.xticks(rotation=45)
plt.grid(True)

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



42 Pandemic

[111]:
```
pandemic_period = (sahm_rule.index >= '2019-01-01') & (sahm_rule.index <=␣
  ↪'2022-03-01')

# plot for the COVID-19 Pandemic
plt.figure(figsize=(10, 6))
plt.plot(sahm_rule.index[pandemic_period], sahm_rule[pandemic_period],␣
  ↪label='Sahm Rule (3-Month MA)', color='blue')
```

```
plt.plot(sahm_rule_ma6.index[pandemic_period], sahm_rule_ma6[pandemic_period],␣
 ↪label='6-Month MA', color='green')

# highlight in red where the 6-month moving average exceeds the 12-month␣
 ↪minimum by 0.5 percentage points
plt.fill_between(sahm_rule_ma6.index[pandemic_period],␣
 ↪sahm_rule_ma6[pandemic_period], where=trigger[pandemic_period], color='red',␣
 ↪alpha=0.5, label='Trigger (6-Month MA > 0.5% above 12M Low)')

# add labels, title, and grid
plt.title('Sahm Rule Indicator During the Pandemic', fontsize=14)
plt.xlabel('Date', fontsize=12)
plt.ylabel('%', fontsize=12)
plt.axhline(y=0.5, color='black', linestyle='--', label='Trigger Threshold (0.
 ↪5%)')
plt.legend()

# format the x-axis with dates
plt.xticks(rotation=45)
plt.grid(True)

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