GWP1. Group Number 4175 Group Member : Yhasreen Ebenezer Oratile

In [1]:
```
import pandas_datareader as pdr
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
yf.pdr_override()

# Using code from FRED API: Get US Economic Data using Python


def get_fred_data(param_list, start_date, end_date):
    df = pdr.DataReader(param_list, "fred", start_date, end_date)
    return df.reset_index()
```

Scenario 1

Individual Credit and Financial Data are private and sensetive, therefor concernt and authorization is needed so as to access those. However publicly available aggregate financial/credit data can be easily obtained.

In [1]:
```
#pip install fredapi
```

https://fred.stlouisfed.org/series/

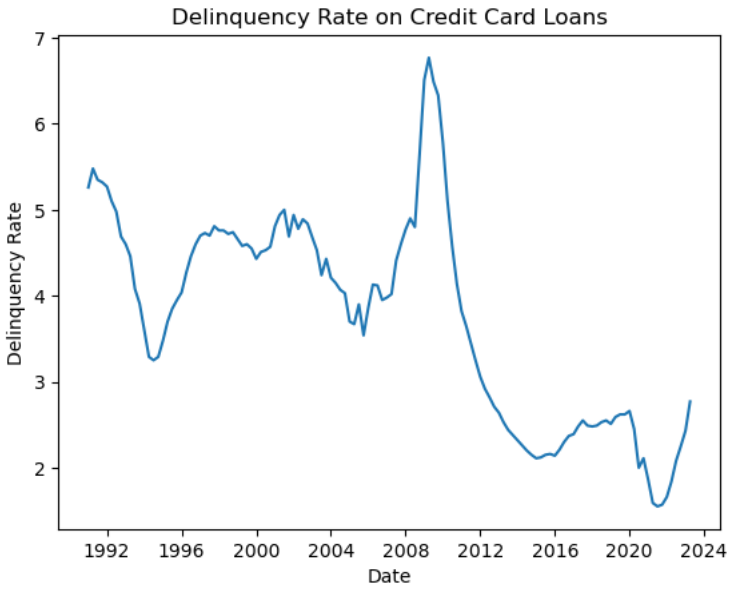
In [3]:
```
from fredapi import Fred

fred = Fred(api_key='7490f681c7eb037113b1a75963b074db')
```

Credit Card Delinquency Rates: These rates provide insights into the percentage of credit card borrowers who are late in making their payments. It can help assess the credit risk associated with lending to individuals.

In [4]:
```
# Delinquency Rate on Credit Card Loans, All Commercial Banks
delinquency_rates = fred.get_series('DRCCLACBS')
```

In [5]:
```
import pandas as pd
df_delinquency = pd.DataFrame(delinquency_rates)
```

In [6]:
```
import matplotlib.pyplot as plt

# Plotting the delinquency rates
plt.plot(df_delinquency.index, df_delinquency[0])

# Adding labels and title
plt.xlabel('Date')
plt.ylabel('Delinquency Rate')
plt.title('Delinquency Rate on Credit Card Loans')

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



The line plot envisions the pattern of wrongdoing rates on charge card advances after some time. It gives a succinct portrayal of the verifiable misconduct rate variances, permitting loan specialists to survey the steadiness and hazard related with loaning Visas. By noticing the progressions in wrongdoing rates, loan specialists can pursue informed choices with respect to charge card issuance, financing costs, credit cutoff points, and hazard the executives systems to moderate likely misfortunes and improve their loaning rehearses for situation 1.

Credit Risk Assesment: The delinquency rate on a Visa loan gives the bank a percentage of the credit card loan amount that is delinquent. Following this process allows the bank to assess the borrower's trustworthiness and the overall riskiness of the Visa portfolio. Higher levels of violations can increase the likelihood of default and higher credit risk. Portfolio Management: Banks can use breach indicators to review the exposure of their payment card credit portfolio. By analyzing changes in the number of delinquencies over time, banks can identify patterns and examples of borrowers' payment behavior and take appropriate action. This may include changing credit limits, applying blended selection techniques, or adjusting the support model to manage credit risk. Early Intervention and Collections: Fraud rates help loan officers recognize borrowers who are at risk of falling behind on their credit card payments. By monitoring the level of misconduct, lenders can proactively intervene, communicate with delinquent borrowers, and implement matching methodologies to limit misfortune and further improve recovery rates.

Credit Card Interest Rates: Obtaining data on credit card interest rates allows you to analyze the cost of borrowing for cardholders. It helps in understanding the financial impact of interest charges on borrowers.

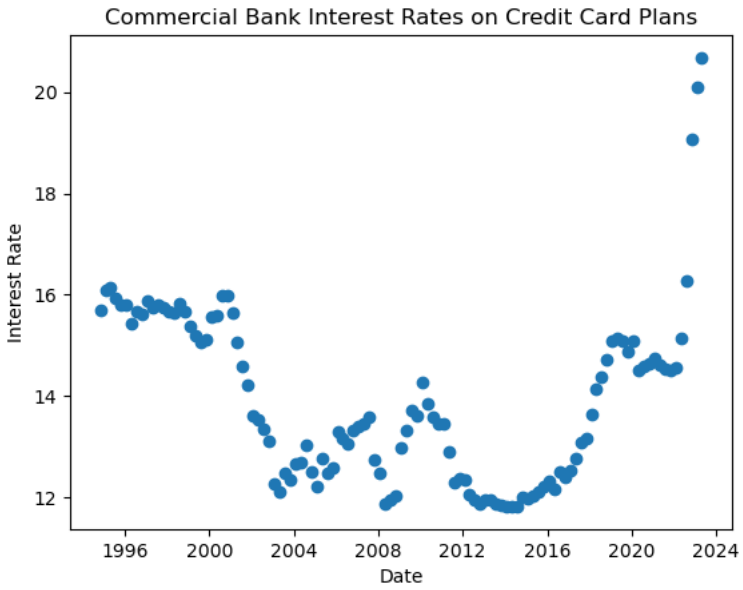
In [7]:
```
# Commercial Bank Interest Rate on Credit Card Plans, All Accounts
creditCard_interest_rates = fred.get_series('TERMCBCCALLNS')
```

In [8]:
```
df_interest = pd.DataFrame(creditCard_interest_rates)
```

In [9]:
```
df_interest.info() # There are 226 null values. Scatter plot will be more suitable for this data
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 343 entries, 1994-11-01 to 2023-05-01
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       115 non-null    float64
dtypes: float64(1)
memory usage: 5.4 KB
```

In [10]:
```
# Create the line plot
plt.scatter(df_interest.index, df_interest[0])

# Set the x-axis label, y-axis label, and title
plt.xlabel('Date')
plt.ylabel('Interest Rate')
plt.title('Commercial Bank Interest Rates on Credit Card Plans')

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

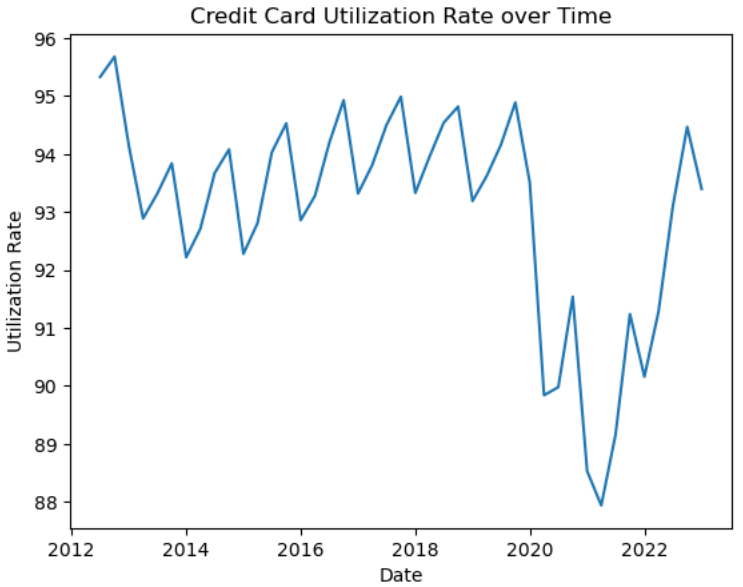


The scatter plot clearly indicates/shows the data points of the commercial bank and its interest rates on credit the card plans over time. Each data point represents a specific date and its corresponding interest rate.

In [11]:
```
# Large Bank Consumer Credit Card Balances: Utilization: Active Accounts Only: 90th Percentile
credit_utilization_rate = fred.get_series('RCCCBACTIVEUTILPCT90')
```

In [12]:
```
df_utilization = pd.DataFrame(credit_utilization_rate)
```

In [13]:
```
import matplotlib.pyplot as plt

# Line plot
plt.plot(df_utilization.index, df_utilization[0])
plt.xlabel('Date')
plt.ylabel('Utilization Rate')
plt.title('Credit Card Utilization Rate over Time')
plt.show()
```



Credit use rate alludes to the level of accessible credit that a borrower is at present utilizing. It is a significant consider surveying a singular's reliability and monetary wellbeing.

In situation 1, where the emphasis is using a loan related gambles and the evaluation of borrowers' capacity to reimburse, the 90th percentile credit use rate can give important experiences. By breaking down this information, you can comprehend the dissemination of credit use rates among Visa accounts and recognize borrowers who have moderately high credit usage.

A high credit use rate recommends that a borrower is using a critical piece of their accessible credit, which might show a higher gamble of monetary strain or trouble in overseeing obligation. Checking the 90th percentile credit use rate can assist with recognizing borrowers who might be at a higher

gamble of default or monetary precariousness. Banks can consider this data while assessing the reliability and reimbursement limit of likely borrowers

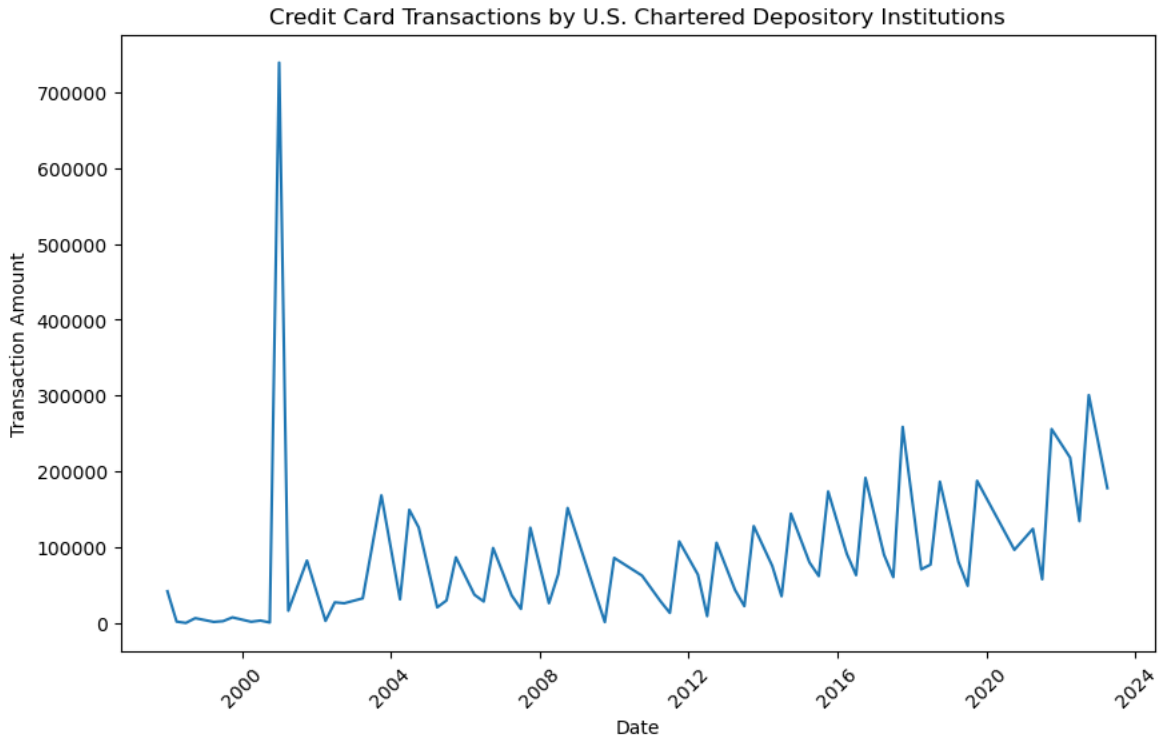
In [14]:
```
# U.S.-Chartered Depository Institutions; Consumer Credit, Credit Cards; Asset, Transactions
depository_institutions_creditcard_transactions = fred.get_series('BOGZ1FA763066113Q')
```

In [15]:
```
df_depository_credit_card_trans = pd.DataFrame(depository_institutions_creditcard_transactions)
```

In [16]:
```
df_depository_credit_card_trans.dropna(inplace=True)
```

In [17]:
```
df_depository_credit_card_trans = df_depository_credit_card_trans[df_depository_credit_card_trans[0]>0]
```

In [18]:
```
# Plotting the line chart
plt.figure(figsize=(10, 6))
plt.plot(df_depository_credit_card_trans.index, df_depository_credit_card_trans[0])

# Customizing the chart
plt.title('Credit Card Transactions by U.S. Chartered Depository Institutions')
plt.xlabel('Date')
plt.ylabel('Transaction Amount')
plt.xticks(rotation=45)

# Displaying the chart
plt.show()
```



The line plot addresses the pattern of Visa exchanges by U.S. sanctioned storehouse organizations over the long run. The x-pivot addresses the dates, while the y-hub addresses the exchange sums.

The chart shows the change in Visa exchanges throughout the given time span. It gives a visual portrayal of the progressions in exchange sums, permitting you to notice any examples, patterns, or irregularities in the information.

The line interfaces the data of interest, showing the overall course of the exchange sums over the long run. Assuming the line is by and large vertical, it recommends a rising pattern in charge card exchanges. On the other hand, on the off chance that the line is for the most part descending, it demonstrates a diminishing pattern. The steepness of the line addresses the pace of progress in exchange sums.

By dissecting this line plot, you can acquire experiences into the general movement and development of Visa exchanges by U.S. contracted vault organizations.

Following patterns in Visa resource development: Observing the development of Mastercard resources can assist with distinguishing the general extension or constriction of Visa loaning by U.S. storehouse establishments. This data can be significant for surveying the accessibility and volume of Mastercard credits.

Dissecting Mastercard obligation levels: The information can give experiences into the aggregate sum of Visa obligation held by purchasers, considering examination of obligation levels and potential dangers related with elevated degrees of customer obligation.

Recognizing shifts in charge card use designs: By looking at the exchange information, you can recognize changes in purchaser ways of managing money and examples of charge card use. This can be useful for understanding customer conduct and distinguishing potential dangers related with changes in spending designs.

In [ ]:

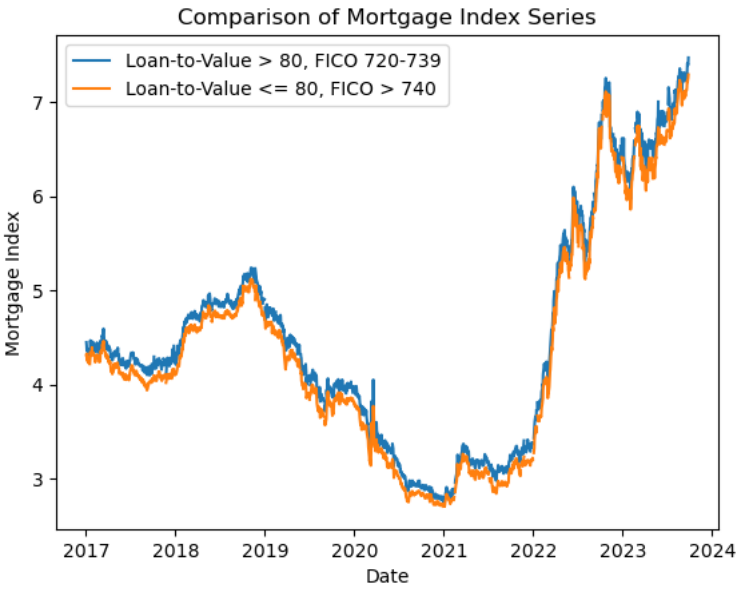
Scenario 2

Individual mortgage loan data

Individual mortgage loan data: It refers to specific information related to each mortgage loan granted to individual borrowers. It includes details such as mortgage rates, default rates, loan terms, borrower information, and loan-to-value ratio, providing a comprehensive picture of the characteristics and performance of individual mortgage loans.

In [19]:
```
# 30-Year Fixed Rate Conforming Mortgage Index: Loan-to-Value Greater Than 80, FICO Score Between 720 and 739
mortgage_index_720_739 = fred.get_series('OBMMIC30YFLVGT80FB720A739')
df_mortgage_index_720_739 = pd.DataFrame(mortgage_index_720_739)

# 30-Year Fixed Rate Conforming Mortgage Index: Loan-to-Value Less Than or Equal to 80, FICO Score Greater Than 740
mortgage_index_740 = fred.get_series('OBMMIC30YFLVLE80FGE740')
df_mortgage_index_740 = pd.DataFrame(mortgage_index_740)
```

In [20]:
```
import matplotlib.pyplot as plt

# Plotting the first mortgage index series
plt.plot(df_mortgage_index_720_739.index, df_mortgage_index_720_739[0], label='Loan-to-Value > 80, FICO 720-739')

# Plotting the second mortgage index series
plt.plot(df_mortgage_index_740.index, df_mortgage_index_740[0], label='Loan-to-Value <= 80, FICO > 740')

# Adding labels and title
plt.xlabel('Date')
plt.ylabel('Mortgage Index')
plt.title('Comparison of Mortgage Index Series')
plt.legend()

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



The chart shows two home loan file series: " Credit to-Esteem More prominent Than 80, credit rating Somewhere in the range of 720 and 739" and "Advance to-Esteem Not exactly or Equivalent to 80, credit rating More noteworthy Than 740." These series address various fragments of home loan borrowers in view of their financial soundness and advance to-esteem proportion.

The x-pivot addresses the date, demonstrating the course of events of the information. The y-hub addresses the home loan record, which is a proportion of home loan financing costs.

The chart permits us to outwardly analyze the two home loan file series after some time. It gives experiences into the patterns and developments of home loan financing costs for various borrower portions. By noticing the progressions in the record values, banks can survey the loan fee climate and pursue informed choices connected with contract loaning.

For Senario 2, where we are dissecting monetary and land information, the diagram of home loan file series can be useful in the accompanying ways:

Market Investigation: The chart assists banks with understanding the general loan costs proposed to borrowers in various financial soundness classifications. It permits them to assess the seriousness of their own loan fees contrasted with the market.

Risk Appraisal: By observing the patterns in contract financing costs for various borrower sections, moneylenders can survey the potential dangers related with explicit loaning classifications. For instance, assuming financing costs for borrowers with lower FICO assessments and higher advance to-esteem proportions are altogether not quite the same as different portions, it can show higher gamble levels for that classification.

Valuing Procedures: The diagram can direct banks in creating evaluating techniques for contract items. By breaking down the developments in contract file series, moneylenders can change their loan cost contributions to line up with market patterns and actually position themselves in the serious scene.

Macroeconomic factors

In [21]:
```
unemployment_rate = fred.get_series('UNRATE')
gdp = fred.get_series('GDP')
inflation_rate = fred.get_series('CPIAUCSL')
interest_rate = fred.get_series('FEDFUNDS')
labor_force_participation = fred.get_series('CIVPART')
consumer_sentiment = fred.get_series('UMCSENT')
financial_index = fred.get_series('SP500')
```

In [22]:
```
unemployment_rate
```

Out[22]:
```
1948-01-01    3.4
1948-02-01    3.8
1948-03-01    4.0
1948-04-01    3.9
1948-05-01    3.5
             ...
2023-04-01    3.4
2023-05-01    3.7
2023-06-01    3.6
2023-07-01    3.5
2023-08-01    3.8
Length: 908, dtype: float64
```

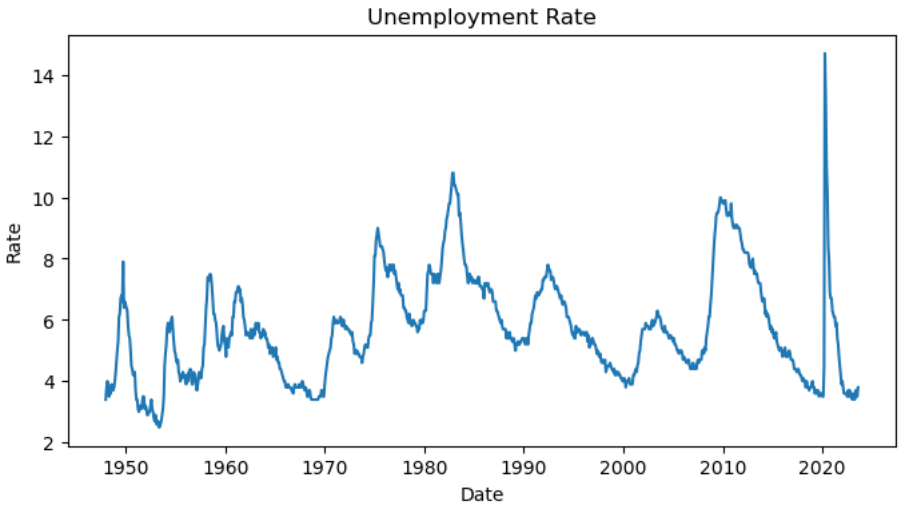
In [23]:
```
info = fred.get_series_info('UNRATE')
```

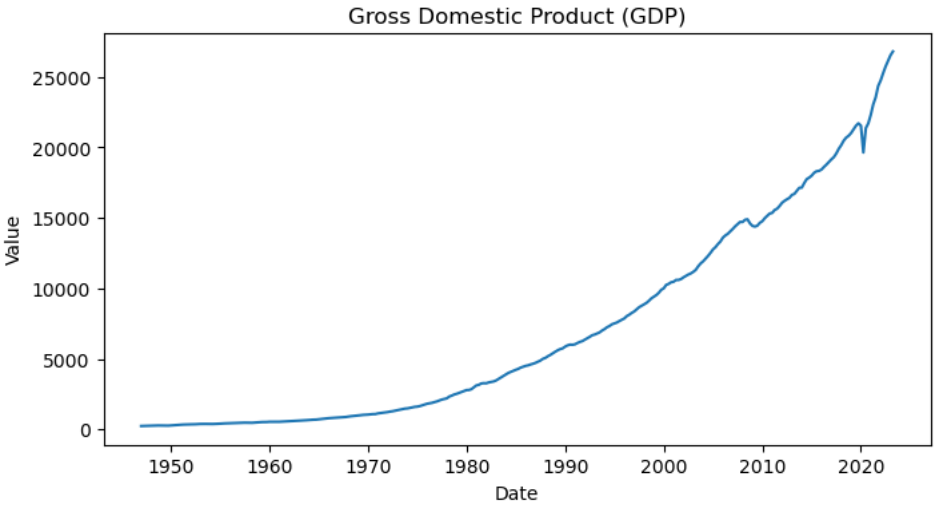
In [24]:
```
info
```

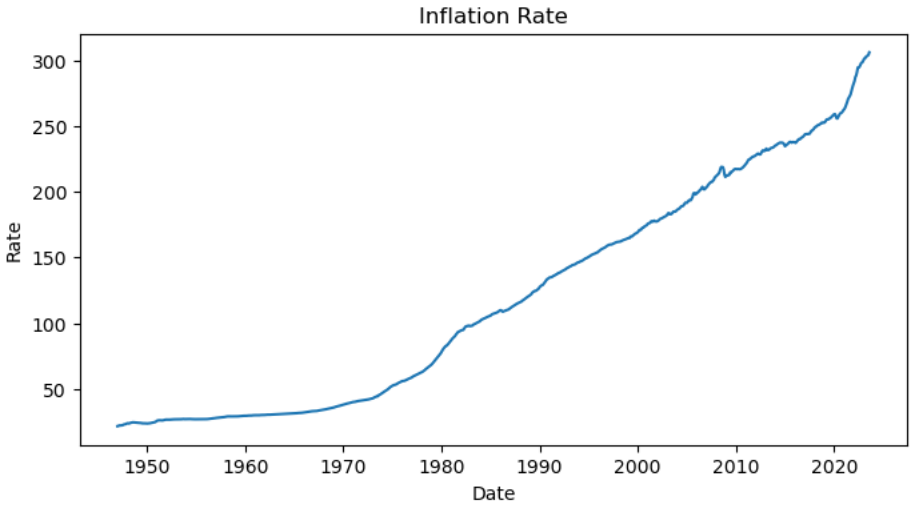
Out[24]:
```
id                                                        UNRATE
realtime_start                                        2023-09-28
realtime_end                                          2023-09-28
title                                          Unemployment Rate
observation_start                                     1948-01-01
observation_end                                       2023-08-01
frequency                                            Monthly
frequency_short                                            M
units                                                Percent
units_short                                                %
seasonal_adjustment                          Seasonally Adjusted
seasonal_adjustment_short                                 SA
last_updated                           2023-09-01 07:45:02-05
popularity                                                94
notes                     The unemployment rate represents the number of...
dtype: object
```

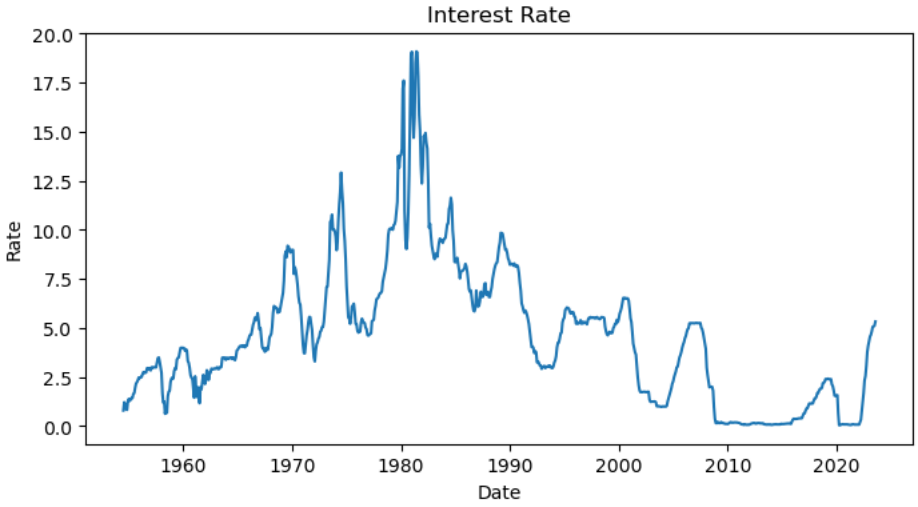
In [25]:
```
import matplotlib.pyplot as plt

# Unemployment Rate
plt.figure(figsize=(8, 4))
plt.plot(unemployment_rate)
plt.title("Unemployment Rate")
plt.xlabel("Date")
plt.ylabel("Rate")
plt.show()

# GDP
plt.figure(figsize=(8, 4))
plt.plot(gdp)
plt.title("Gross Domestic Product (GDP)")
plt.xlabel("Date")
plt.ylabel("Value")
plt.show()
```

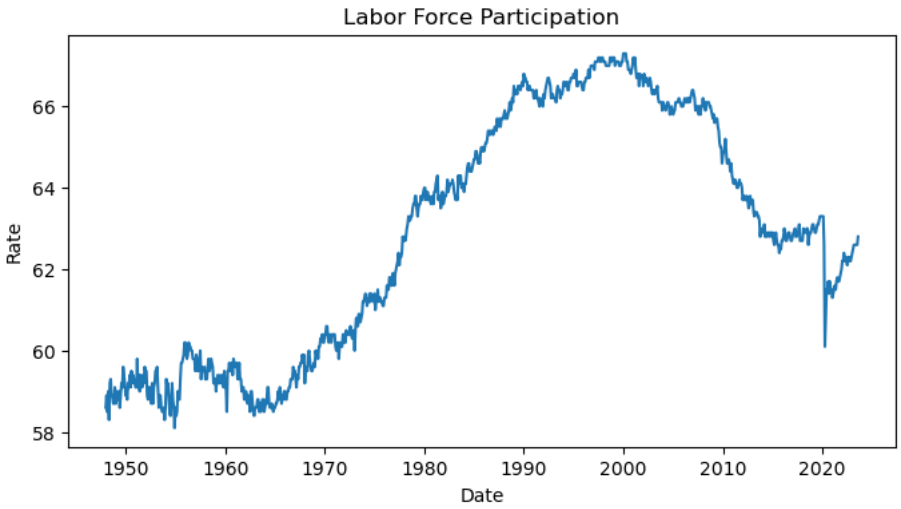
```
# Inflation Rate
plt.figure(figsize=(8, 4))
plt.plot(inflation_rate)
plt.title("Inflation Rate")
plt.xlabel("Date")
plt.ylabel("Rate")
plt.show()

# Interest Rate
plt.figure(figsize=(8, 4))
plt.plot(interest_rate)
plt.title("Interest Rate")
plt.xlabel("Date")
plt.ylabel("Rate")
plt.show()

# Labor Force Participation
plt.figure(figsize=(8, 4))
plt.plot(labor_force_participation)
plt.title("Labor Force Participation")
plt.xlabel("Date")
plt.ylabel("Rate")
plt.show()

# Consumer Sentiment
plt.figure(figsize=(8, 4))
plt.plot(consumer_sentiment)
plt.title("Consumer Sentiment")
plt.xlabel("Date")
plt.ylabel("Index")
plt.show()

# Financial Index
plt.figure(figsize=(8, 4))
plt.plot(financial_index)
plt.title("Financial Index (S&P 500)")
plt.xlabel("Date")
plt.ylabel("Value")
plt.show()
```

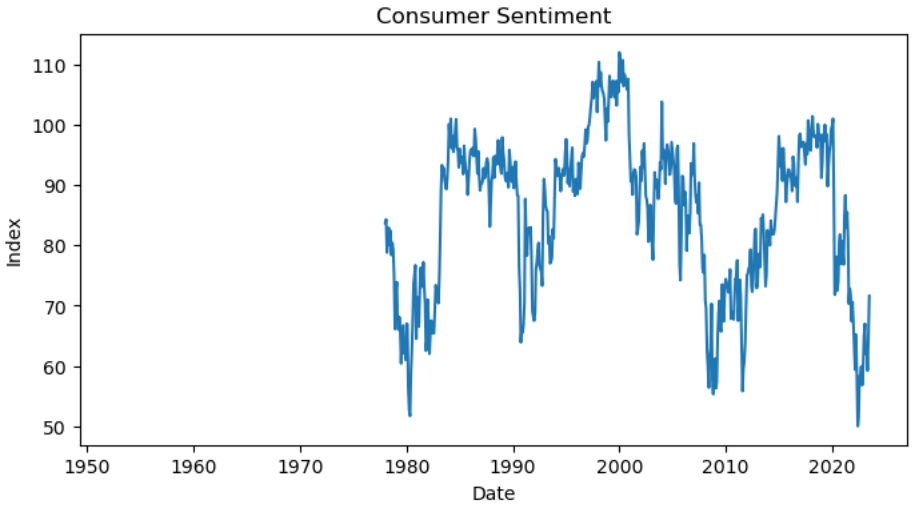


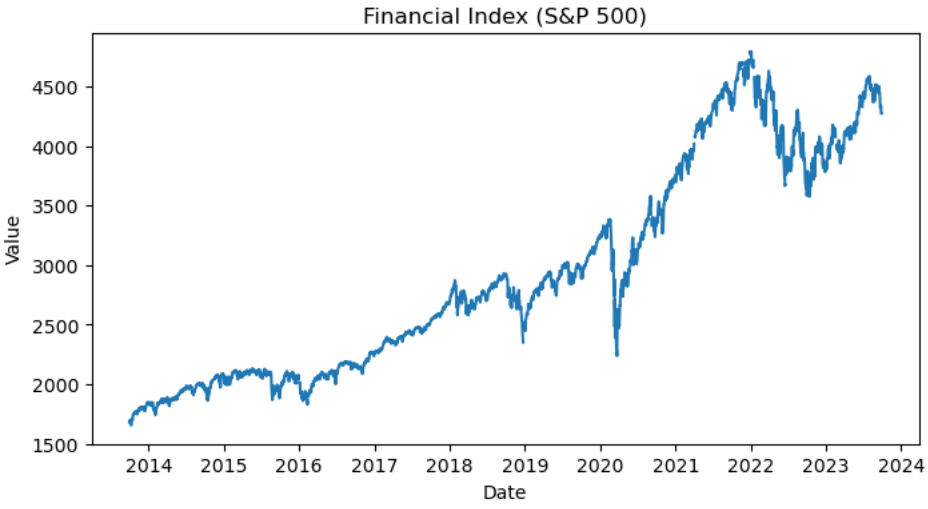

Gross Domestic Product (GDP)
Value
Date
0
5000
10000
15000
20000
25000
1950
1960
1970
1980
1990
2000
2010
2020


Inflation Rate
Rate
Date
50
100
150
200
250
300
1950
1960
1970
1980
1990
2000
2010
2020


Interest Rate
Rate
Date
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
1960
1970
1980
1990
2000
2010
2020







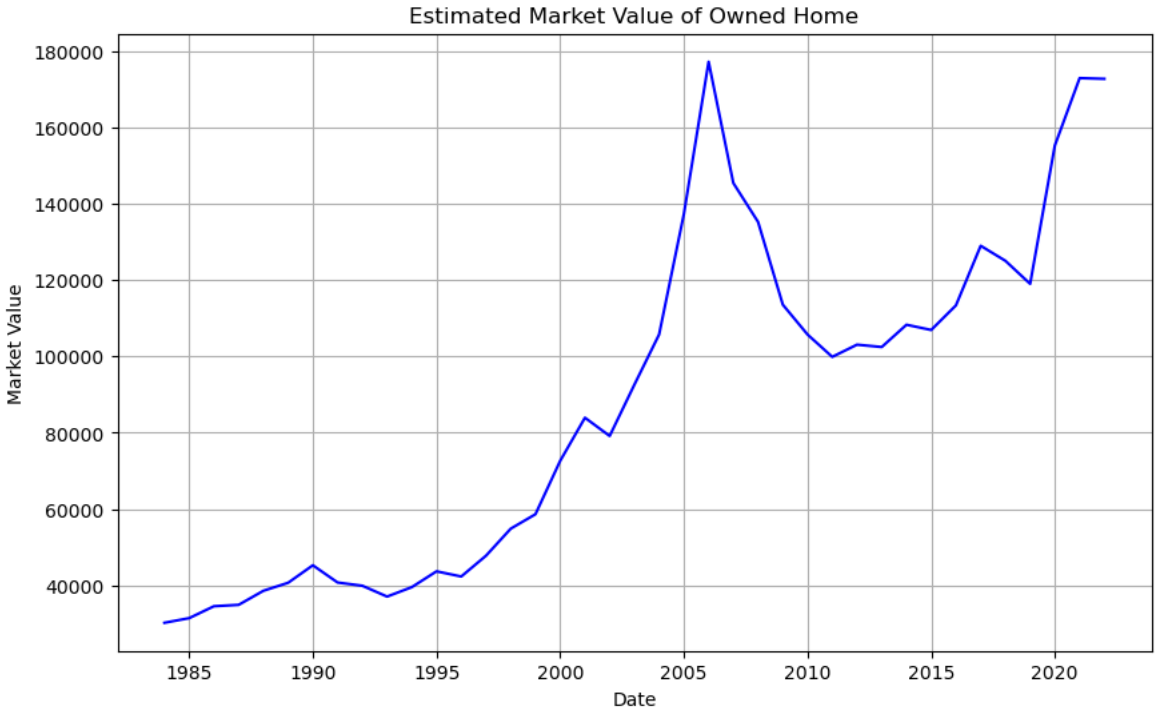
These macroeconomic elements assist with Senario2 by giving significant data to surveying the monetary climate in which loaning choices are made. Checking these elements assists loan specialists with measuring the in general financial circumstances, evaluate the soundness and likely dangers in the economy, and pursue informed choices in regards to reliability, risk the board, and market patterns. For instance, a high joblessness rate or languid Gross domestic product development might show a higher gamble of credit defaults, while low expansion and stable loan costs might make great getting conditions. Purchaser opinion and monetary market execution can likewise influence buyer conduct and credit interest. By integrating these macroeconomic elements into their examination, loan specialists can all the more likely assess the reliability of borrowers and deal with their loaning portfolios successfully.

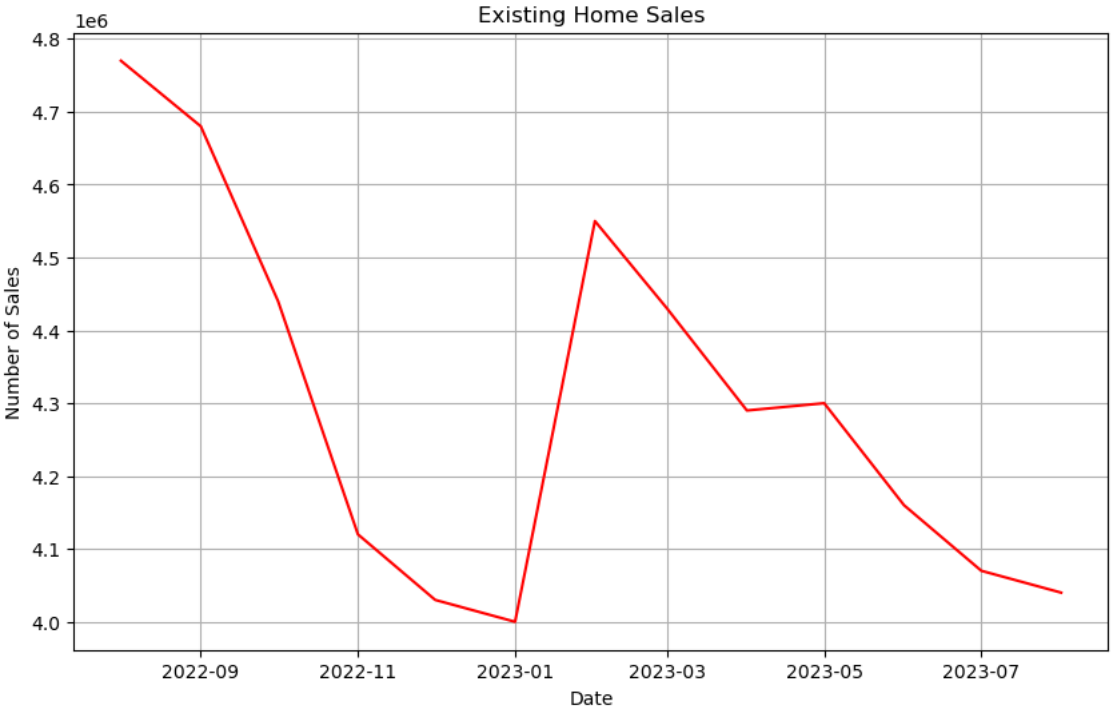
Housing Market Indicators

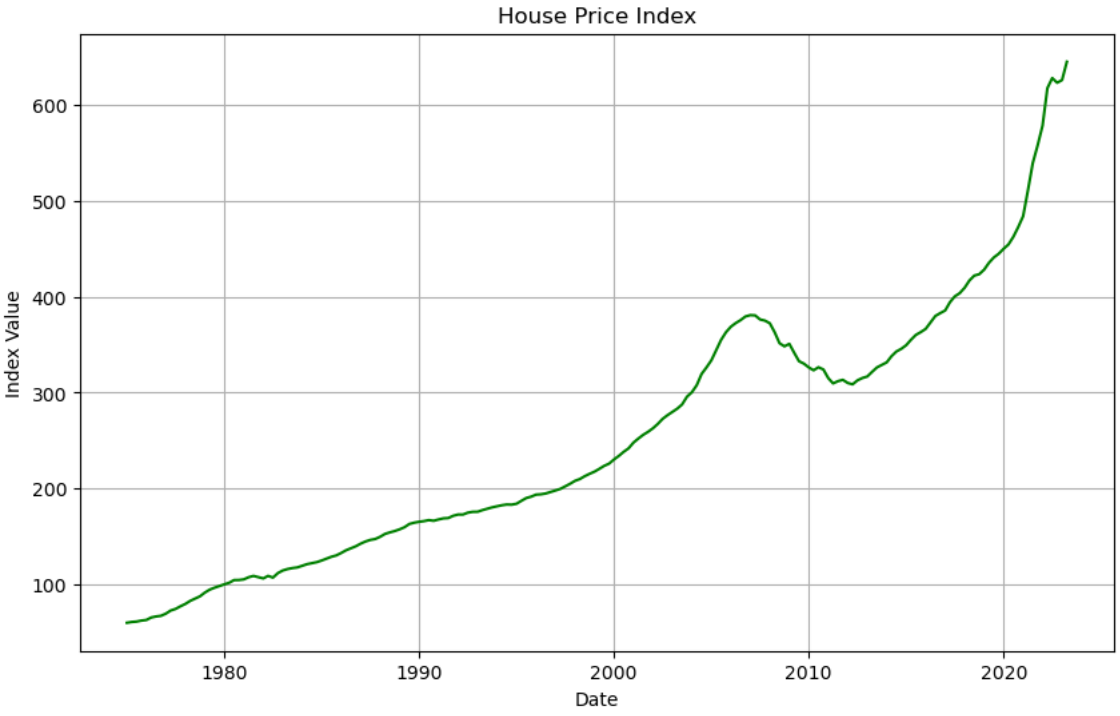
https://fred.stlouisfed.org/series/

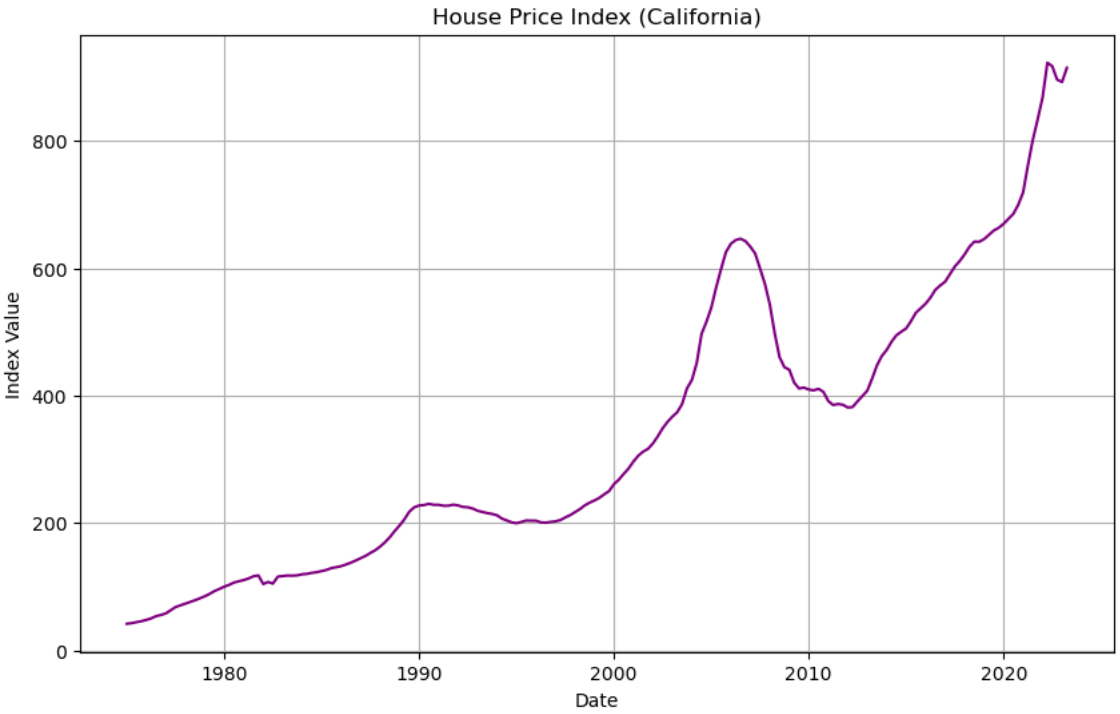
In [26]:
```
estimated_market_value_of_owned_home = fred.get_series('CXU800721LB1210M')
existing_home_sales = fred.get_series('EXHOSLUSM495S')
house_price_index = fred.get_series('USSTHPI')
house_price_index_california = fred.get_series('CASTHPI')
```

In [27]:
```
import matplotlib.pyplot as plt

# Plotting Estimated Market Value of Owned Home
plt.figure(figsize=(10, 6))
plt.plot(estimated_market_value_of_owned_home, color='blue')
plt.title('Estimated Market Value of Owned Home')
plt.xlabel('Date')
plt.ylabel('Market Value')
plt.grid(True)
plt.show()

# Plotting Existing Home Sales
plt.figure(figsize=(10, 6))
plt.plot(existing_home_sales, color='red')
plt.title('Existing Home Sales')
plt.xlabel('Date')
plt.ylabel('Number of Sales')
plt.grid(True)
plt.show()

# Plotting House Price Index
plt.figure(figsize=(10, 6))
plt.plot(house_price_index, color='green')
plt.title('House Price Index')
plt.xlabel('Date')
plt.ylabel('Index Value')
plt.grid(True)
plt.show()

# Plotting House Price Index (California)
plt.figure(figsize=(10, 6))
plt.plot(house_price_index_california, color='purple')
plt.title('House Price Index (California)')
plt.xlabel('Date')
plt.ylabel('Index Value')
plt.grid(True)
plt.show()
```




Existing Home Sales
1e6
4.8
4.7
4.6
4.5
4.4
4.3
4.2
4.1
4.0
Number of Sales
2022-09
2022-11
2023-01
2023-03
2023-05
2023-07
Date


House Price Index
600
500
400
300
200
100
Index Value
1980
1990
2000
2010
2020
Date



Assessed Market Worth of Home owners: This diagram shows the assessed market worth of possessed homes. It gives experiences into the worth of private land resources and can be utilized to survey property holders' value and by and large lodging riches.

Existing Home Deals: This chart addresses the quantity of existing homes sold on the lookout. It demonstrates the degree of movement in the real estate market and reflects purchaser interest and dealer conduct. Following existing home deals can assist with understanding business sector patterns and conditions.

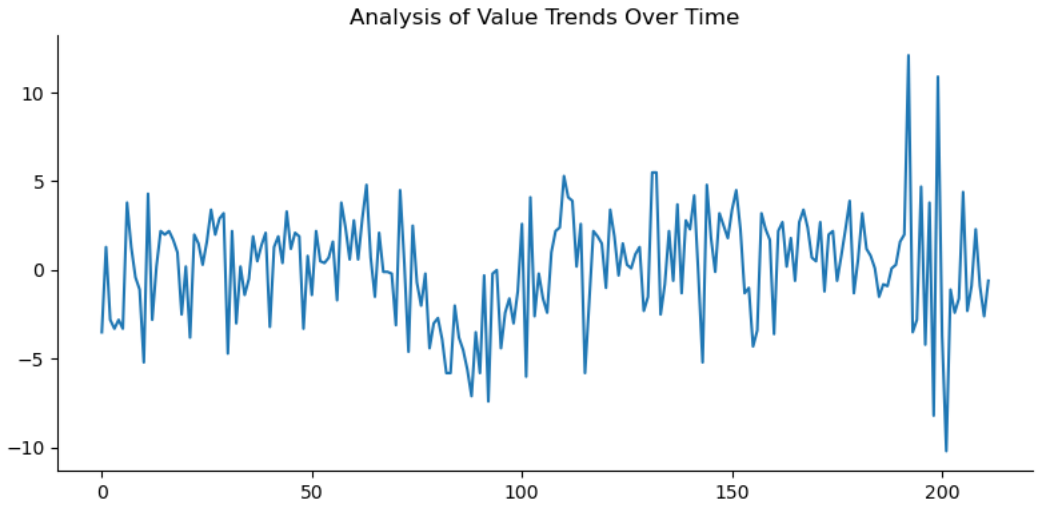
House Value index: This diagram shows the house cost file, which estimates the progressions in the costs of private properties over the long haul. It gives an extensive perspective on in general real estate market drifts and can be utilized to survey cost developments, economic situations, and lodging moderateness.

These real estate market pointers are vital for Situation 2 as they assist moneylenders with evaluating the real estate market's general wellbeing, dependability, and expected chances. By observing these pointers, moneylenders can settle on informed choices with respect to contract loaning, property valuation, risk appraisal, and market patterns. For instance, changes in the assessed market worth of possessed homes can affect borrowers' value positions and their capacity to get to credit. Existing home deals information can mirror the degree of real estate market action, while house cost lists give data on value patterns and likely changes. Banks can use these pointers to assess the financial soundness of borrowers, decide suitable advance to-esteem proportions, and evaluate the general economic situations for contract loaning.

Scenario 3

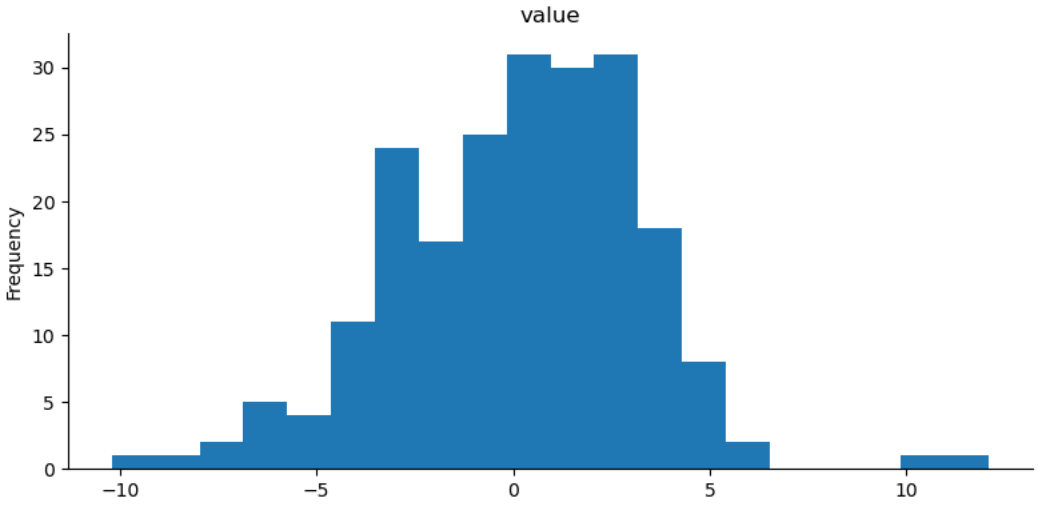
In [35]:
```
data1 = pd.read_csv('MPCT03XXS.csv')
```

In [36]:
```
ata1 = pd.read_csv("MPCT03XXS.csv")

def value_plot(data, y, title, figscale=1):
    plt.figure(figsize=(8 * figscale, 4 * figscale))
    data[y].plot(kind='line', title=title)
    plt.gca().spines[['top', 'right']].set_visible(False)
    plt.tight_layout()
    plt.show()  # Display the plot

value_plot(data1, 'value', title='Analysis of Value Trends Over Time')
```



The line plot depicting changes in the "value" column over time reveals a generally stable trend with a few noticeable fluctuations. While values remained relatively constant, periodic increases and outliers suggest potential external factors impacting the dataset. This analysis provides insights into the temporal dynamics of the "value" variable, allowing for a deeper understanding of its behavior.

In [37]:
```
def histogram(data, colname, num_bins=20, figscale=1):
    data[colname].plot(kind='hist', bins=num_bins, title=colname, figsize=(8*figscale, 4*figscale))
    plt.gca().spines[['top', 'right']].set_visible(False)
    plt.tight_layout()
    plt.show()

data1 = pd.read_csv('MPCT03XXS.csv')

histogram(data1, 'value')
```



The histogram of the "value" column displays a right-skewed distribution, indicating that the majority of values are concentrated at lower levels with a long tail of higher values. The choice of bin size impacts the granularity of the distribution representation. This analysis provides a clear view of the distribution characteristics of the "value" column within the dataset.

In [ ]:
```
```

Library for bond and equity

In [10]:
```
import os
import csv
import nltk, re
import requests
import numpy as np
import pandas as pd
import hvplot.pandas
import seaborn as sns
```

```
import yfinance as yf
import dataframe_image as dfi
from wordcloud import WordCloud
import pandas_datareader as pdr
import matplotlib.pyplot as plt
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from pandas_datareader.data import DataReader
from sklearn.preprocessing import MinMaxScaler
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

In [11]:
```
from datetime import timedelta, datetime
import datetime as dt
```

In [12]:
```
yf.pdr_override()
sns.set_style('whitegrid')
plt.style.use("fivethirtyeight")
%matplotlib inline
```

In [22]:
```
import fredpy as fp
fp.api_key = '7490f681c7eb037113b1a75963b074db'
```

In [23]:
```
from fredapi import Fred
fred = Fred(api_key='7490f681c7eb037113b1a75963b074db')
```

In [24]:
```
api_key = 'O2ZMKAVEVDVQ4RWF'
```

In [ ]:
```
```

---

4. Equity Loan By Yhasreen

4a. Financial Statement

In [25]:
```
# Base Alpha Vantage endpoint & parameters
url = "https://www.alphavantage.co/query"
params = {"symbol": "AAPL" ,"apikey": api_key}

# Fetch Annual Earnings per share, AEPS data
params["function"] = "EARNINGS"
response = requests.get(url, params=params)
annual_eps_data = response.json()['annualEarnings']
eps_df = pd.DataFrame(annual_eps_data)
eps_df =eps_df.iloc[:10]

# Fetch Adjusted Income Statement
params["function"] = "INCOME_STATEMENT"
response = requests.get(url, params=params)
income_data = response.json()['annualReports']
income_df = pd.DataFrame(income_data)
income_df =income_df.iloc[:,:5]

# Fetch Balance Sheet
params["function"] = "BALANCE_SHEET"
response = requests.get(url, params=params)
balance_data = response.json()['annualReports']
balance_df = pd.DataFrame(balance_data)
balance_df = balance_df.iloc[:,:5]

# Fetch Cash Flow
params["function"] = "CASH_FLOW"
response = requests.get(url, params=params)
cash_flow_data = response.json()['annualReports']
cash_df = pd.DataFrame(cash_flow_data)
cash_df = cash_df.iloc[:,:5]

eps_df = eps_df.style.set_caption("AAPL Annual Earnings Per Share")
income_df = income_df.style.set_caption("AAPL Adjusted and Truncated Income Statement")
balance_df = balance_df.style.set_caption("AAPL Truncated Balance Sheet")
cash_df = cash_df.style.set_caption("AAPL Truncated Cash Flow")
```

In [26]:
```
eps_df
```

Out[26]:

AAPL Annual Earnings Per Share

	fiscalDateEnding	reportedEPS
0	2023-06-30	4.66
1	2022-09-30	6.11
2	2021-09-30	5.62
3	2020-09-30	3.27
4	2019-09-30	2.98
5	2018-09-30	2.97
6	2017-09-30	2.3
7	2016-09-30	2.0675
8	2015-09-30	2.3
9	2014-09-30	1.6075

In [27]:
```
balance_df
```

Out[27]:

AAPL Truncated Balance Sheet

	fiscalDateEnding	reportedCurrency	totalAssets	totalCurrentAssets	cashAndCashEquivalentsAtCarryingValue
0	2022-09-30	USD	352755000000	135405000000	23646000000
1	2021-09-30	USD	351002000000	134836000000	34940000000
2	2020-09-30	USD	323888000000	143713000000	38016000000
3	2019-09-30	USD	338516000000	162819000000	48844000000
4	2018-09-30	USD	365725000000	131339000000	25913000000

In [28]:
```
cash_df
```

Out[28]:

AAPL Truncated Cash Flow

	fiscalDateEnding	reportedCurrency	operatingCashflow	paymentsForOperatingActivities	proceedsFromOperatingActivities
0	2022-09-30	USD	122151000000	4665000000	None
1	2021-09-30	USD	104038000000	4087000000	None
2	2020-09-30	USD	80674000000	4502000000	None
3	2019-09-30	USD	69391000000	3423000000	None
4	2018-09-30	USD	77434000000	3022000000	None

In [29]:
```
income_df
```

Out[29]:

AAPL Adjusted and Truncated Income Statement

	fiscalDateEnding	reportedCurrency	grossProfit	totalRevenue	costOfRevenue
0	2022-09-30	USD	170782000000	391397000000	248640000000
1	2021-09-30	USD	152836000000	363172000000	234954000000
2	2020-09-30	USD	104956000000	271642000000	189475000000
3	2019-09-30	USD	98392000000	256598000000	180027000000
4	2018-09-30	USD	101839000000	265595000000	163756000000

In [ ]:
```

```

4b. Historical Prices vs Return

In [30]:
```
# The tech stocks we'll use for this analysis
tech_list = ['AAPL', 'GOOG', 'MSFT', 'AMZN']

# Set up End and Start times for data grab
tech_list = ['AAPL', 'GOOG', 'MSFT', 'AMZN']

end = datetime.now()
start = datetime(end.year - 1, end.month, end.day)

for stock in tech_list:
    globals()[stock] = yf.download(stock, start, end ,progress=False)
```

```
company_list = [AAPL, GOOG, MSFT, AMZN]
company_name = ["APPLE", "GOOGLE", "MICROSOFT", "AMAZON"]

for company, com_name in zip(company_list, company_name):
    company["company_name"] = com_name

df = pd.concat(company_list, axis=0)
df.tail(10)
```

Out[30]:

Date	Open	High	Low	Close	Adj Close	Volume	company_name
2023-09-14	145.080002	145.860001	142.949997	144.720001	144.720001	64033600	AMAZON
2023-09-15	142.690002	143.570007	140.089996	140.389999	140.389999	102861700	AMAZON
2023-09-18	140.479996	141.750000	139.220001	139.979996	139.979996	42823500	AMAZON
2023-09-19	138.699997	138.839996	135.559998	137.630005	137.630005	61482500	AMAZON
2023-09-20	138.550003	139.369995	135.199997	135.289993	135.289993	46263700	AMAZON
2023-09-21	131.940002	132.240005	129.309998	129.330002	129.330002	70234800	AMAZON
2023-09-22	131.110001	132.029999	128.520004	129.119995	129.119995	59859500	AMAZON
2023-09-25	129.360001	131.779999	128.770004	131.270004	131.270004	46017800	AMAZON
2023-09-26	130.119995	130.389999	125.279999	125.980003	125.980003	73048200	AMAZON
2023-09-27	125.760002	127.480003	124.129997	125.980003	125.980003	66496800	AMAZON

In [31]:
```
# Summary Stats
AAPL.describe()
```

Out[31]:

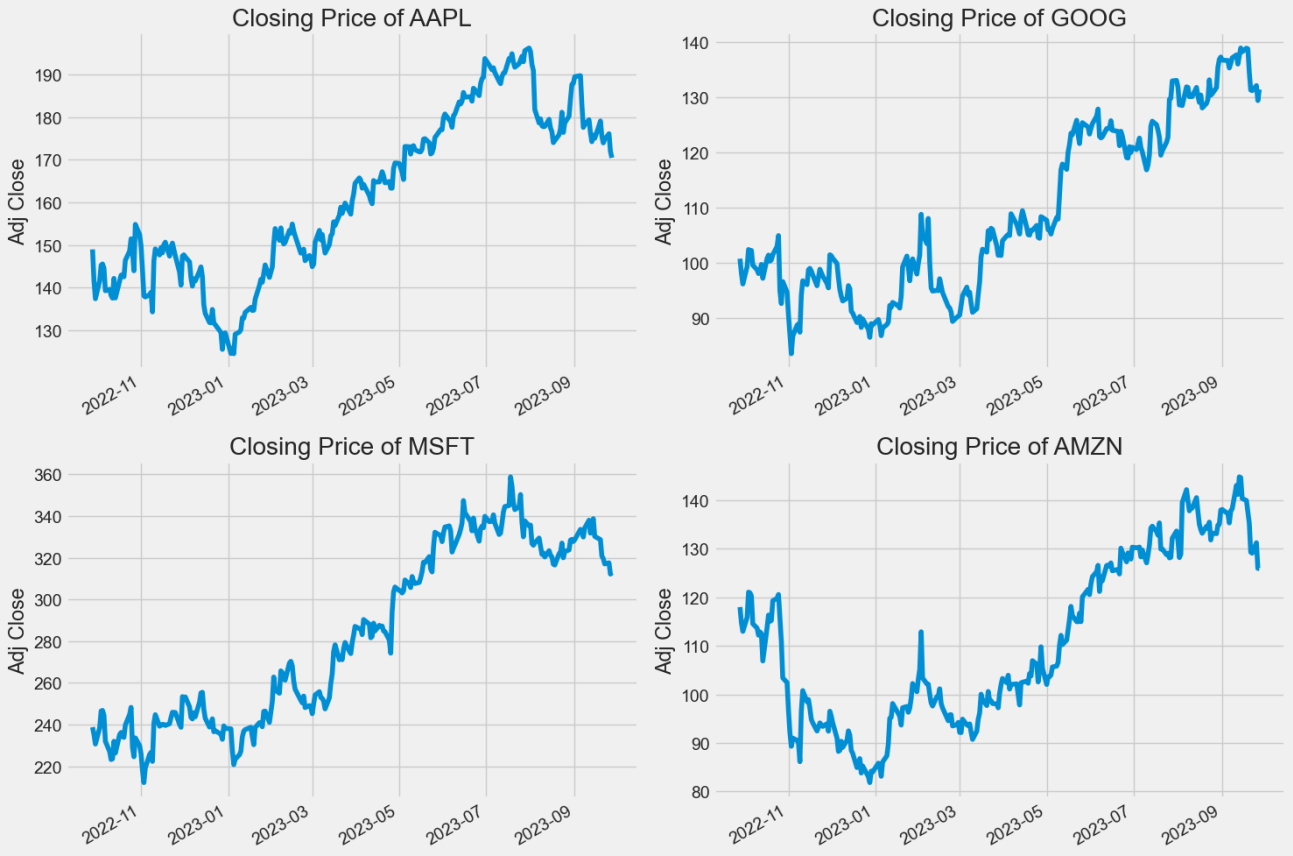
	Open	High	Low	Close	Adj Close	Volume
count	251.000000	251.000000	251.000000	251.000000	251.000000	2.510000e+02
mean	161.597730	163.336056	160.025737	161.740717	161.320722	6.784617e+07
std	19.217619	18.968011	19.451388	19.173090	19.357984	2.279472e+07
min	126.010002	127.769997	124.169998	125.019997	124.488869	3.145820e+07
25%	145.815002	147.340004	144.014999	145.919998	145.274216	5.123105e+07
50%	159.940002	162.029999	159.350006	160.250000	159.812836	6.364660e+07
75%	177.799995	179.660004	176.665001	177.805000	177.655098	7.754820e+07
max	196.240005	198.229996	195.279999	196.449997	196.185074	1.647624e+08

In [32]:
```
# General info
AAPL.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 251 entries, 2022-09-28 to 2023-09-27
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Open          251 non-null    float64
 1   High          251 non-null    float64
 2   Low           251 non-null    float64
 3   Close         251 non-null    float64
 4   Adj Close     251 non-null    float64
 5   Volume        251 non-null    int64
 6   company_name  251 non-null    object
dtypes: float64(5), int64(1), object(1)
memory usage: 15.7+ KB
```

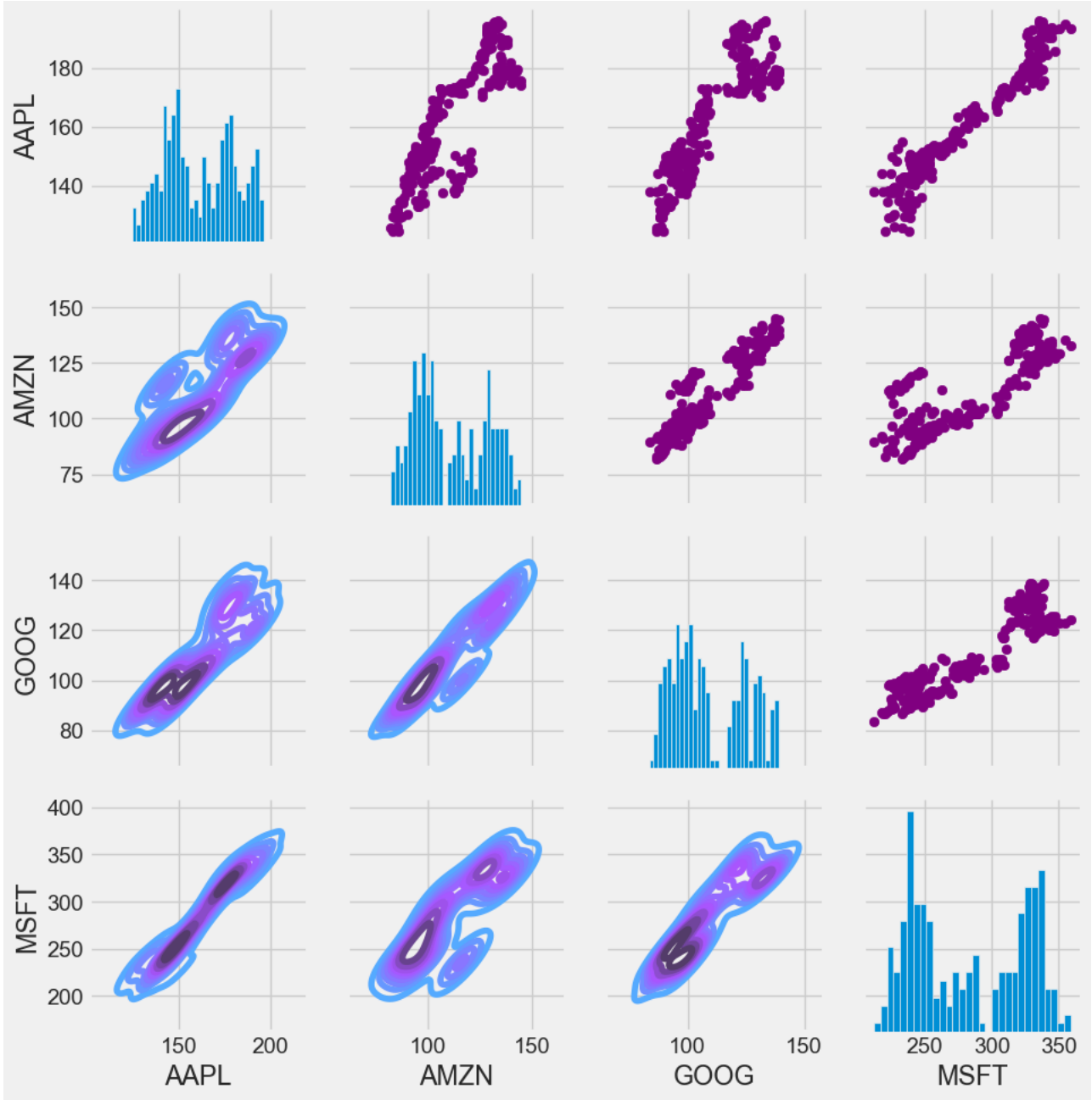
In [33]:
```
# Let's see a historical view of the closing price
plt.figure(figsize=(15, 10))
plt.subplots_adjust(top=1.25, bottom=1.2)

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    company['Adj Close'].plot()
    plt.ylabel('Adj Close')
    plt.xlabel(None)
    plt.title(f"Closing Price of {tech_list[i - 1]}")

plt.tight_layout()
```

```
plt.show()
```



In [34]:
```
from pandas_datareader import data as pdrdata
# # Grab all the closing prices for the tech stock list into one DataFrame
closing_df = pdrdata.get_data_yahoo(tech_list, start=start, end=end ,progress=False )['Adj Close']

# Set up our figure by naming it returns_fig, call PairPLot on the DataFrame
returns_fig = sns.PairGrid(closing_df)

# Using map_upper we can specify what the upper triangle will look like.
returns_fig.map_upper(plt.scatter,color='purple')

# We can also define the lower triangle in the figure, inclufing the plot type (kde) or the color map (BluePurple)
returns_fig.map_lower(sns.kdeplot,cmap='cool_d')

# Finally we'll define the diagonal as a series of histogram plots of the daily return
returns_fig.map_diag(plt.hist,bins=30)
```

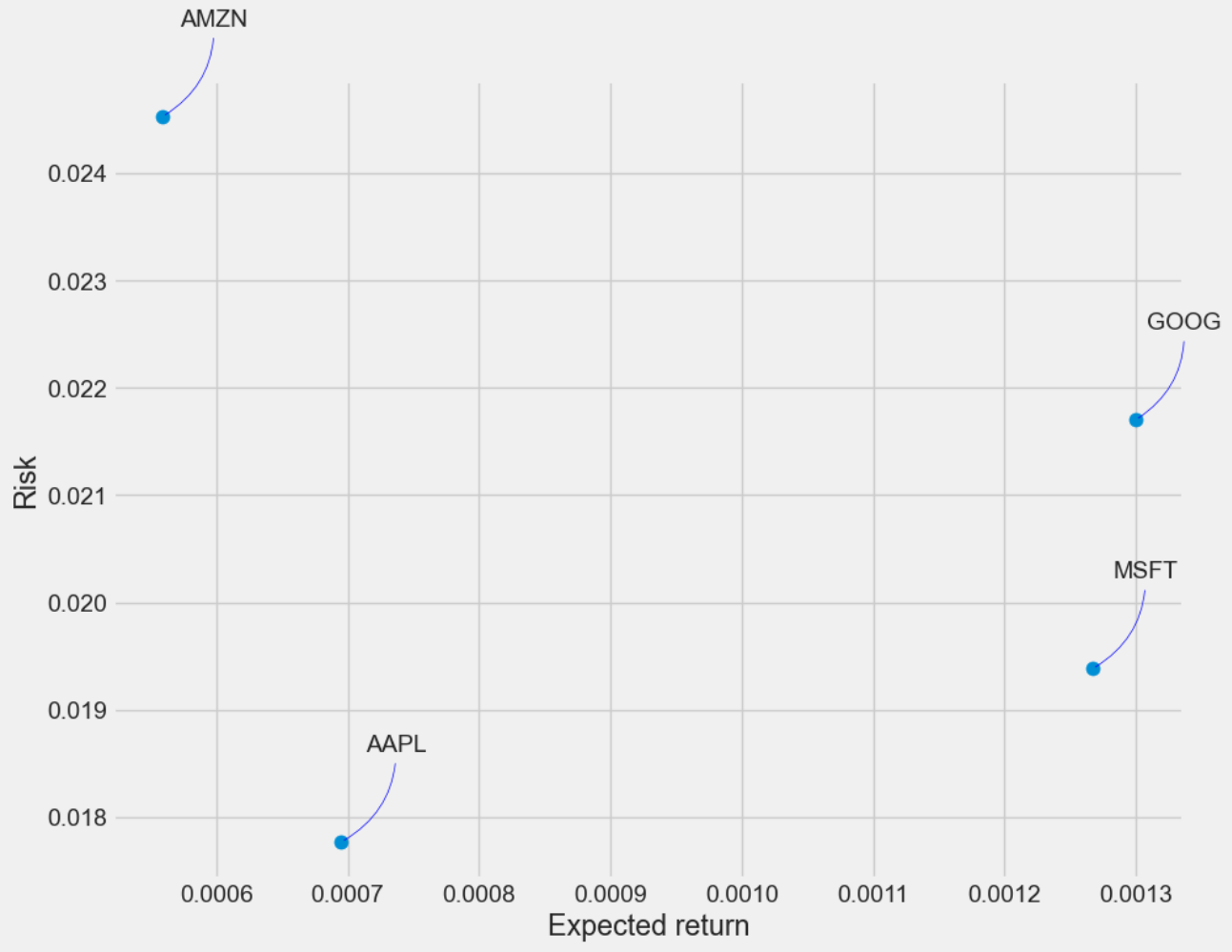
Out[34]:
```
<seaborn.axisgrid.PairGrid at 0x2134513ff90>
```



In [35]:
```
# # Make a new tech returns DataFrame
tech_rets = closing_df.pct_change()

plt.figure(figsize=(10,9))
plt.tight_layout()

plt.subplot(1, 2, 1)
sns.heatmap(tech_rets.corr(), annot=True, cmap='summer')
plt.title('Correlation of stock return')

plt.subplot(1, 2, 2)
sns.heatmap(closing_df.corr(), annot=True, cmap='summer')
plt.title('Correlation of stock closing price')
plt.show()
```

In [36]:

```
rets = tech_rets.dropna()

area = np.pi * 20

plt.figure(figsize=(10, 8))
plt.scatter(rets.mean(), rets.std(), s=area)
plt.xlabel('Expected return')
plt.ylabel('Risk')

for label, x, y in zip(rets.columns, rets.mean(), rets.std()):
    plt.annotate(label, xy=(x, y), xytext=(50, 50), textcoords='offset points', ha='right', va='bottom',
                 arrowprops=dict(arrowstyle='-', color='blue', connectionstyle='arc3,rad=-0.3'))
plt.show()
```



In [ ]:

4c. Maroeconomic Factors

In [37]:
```
def growth_rate(value,df):

    global initial_value
    if df[0] == value:
        initial_value = value
        return 0

    else:
        rate = f'{((value - initial_value) /initial_value) :.2f}'
        initial_value = value
        return float(rate) * 100
```

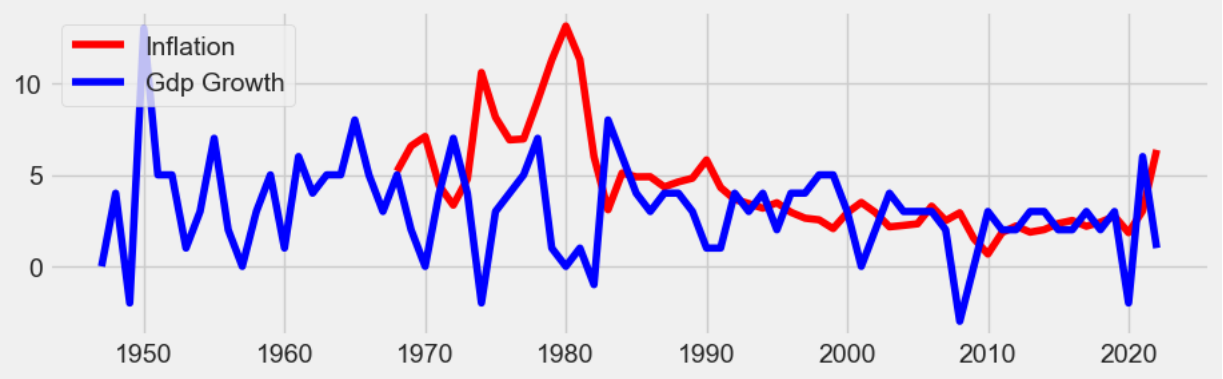
In [38]:
```
def series_to_dataframe(series_object, column_name):

    df = series_object.to_frame(name=column_name)
    df.reset_index(inplace=True)
    df.rename(columns={'index':'date'},inplace=True)

    df['date'] = pd.to_datetime(df['date'])
    df['month'] = df['date'].dt.month
    df['year'] = df['date'].dt.year

    new_df= df.loc[df['month']==10].copy()
    new_df = new_df.loc[df['year']>=1948].copy()
    new_df.reset_index(drop=True,inplace=True)
    new_df.drop(['date','month'],axis=1,inplace=True)

    return new_df
```

In [39]:
```
def plot_many(number,*plots,labels,colors,title=None):

    plt.figure(figsize=(10,3))
    for plotno in range(number):
```

```
        X=plots[0][plotno][0]
        Y=plots[0][plotno][1]

        plt.plot(X,Y,colors[plotno],label=labels[plotno])

    if title:
        plt.title(title)
    plt.legend(loc='upper left')


    plt.show()
```

In [40]:
```
#retrieving the GDP figures

gdp = fred.get_series('GDPC1') #quarterly data
gdp.dropna(axis=0,inplace=True)
gdp_df = gdp.to_frame(name='gdp')
gdp_df['quarter'] = gdp_df.index
gdp_df.index = [index for index in  range(len(gdp_df))]


gdp_df['gdp_growth'] = gdp_df['gdp'].apply(growth_rate, df=gdp_df['gdp'])
gdp_df['quarter'] = pd.to_datetime(gdp_df['quarter'])
gdp_df['month'] = gdp_df['quarter'].dt.month

#gdp growth rate per year
gdp_year_df=gdp_df.loc[gdp_df['month']==10].copy()
gdp_year_df['year'] = gdp_year_df['quarter'].dt.year
gdp_year_df.reset_index(drop=True, inplace=True)
gdp_year_df.drop('month',axis=1,inplace=True)
gdp_year_df.drop('quarter',axis=1,inplace=True)

gdp_year_df['gdp_growth'] = gdp_year_df['gdp'].apply(growth_rate,df=gdp_year_df['gdp'])
```

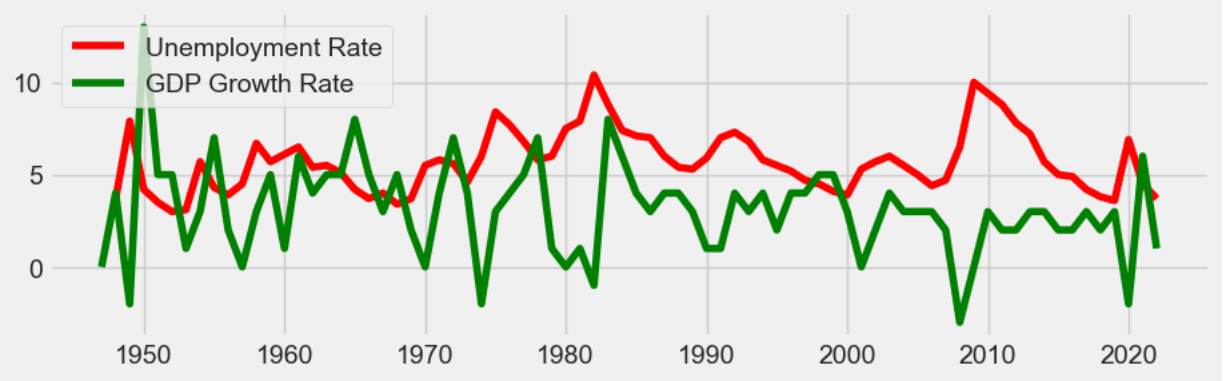
In [41]:
```
cpi = fred.get_series('CORESTICKM159SFRBATL')

cpi_df = series_to_dataframe(cpi,column_name='rate')
plot_many(2,[[cpi_df['year'],cpi_df['rate']],
             [gdp_year_df['year'],gdp_year_df['gdp_growth']]],
             labels=['Inflation','Gdp Growth'],
             colors=['red','blue'])
```



In [42]:
```
def series_to_dataframe(series_object, column_name):

    df = series_object.to_frame(name=column_name)
    df.reset_index(inplace=True)
    df.rename(columns={'index':'date'},inplace=True)

    df['date'] = pd.to_datetime(df['date'])
    df['month'] = df['date'].dt.month
    df['year'] = df['date'].dt.year

    new_df= df.loc[df['month']==10].copy()
    new_df = new_df.loc[df['year']>=1948].copy()
    new_df.reset_index(drop=True,inplace=True)
    new_df.drop(['date','month'],axis=1,inplace=True)

    return new_df
```

In [43]:
```
##Unemployment figures
unef = fred.get_series('UNRATE')
unef_df = series_to_dataframe(unef,column_name='rate')

plt.figure(figsize=(10,3))
plt.plot(unef_df['year'],unef_df['rate'],'r',label='Unemployment Rate')
plt.plot(gdp_year_df['year'],gdp_year_df['gdp_growth'],'g',label='GDP Growth Rate')
```

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

plt.show()
```



In [ ]:

4d. Market Sentiment

In [44]:
```
price_corr_df = pd.read_excel('price_corr_df1.xlsx')
symbol = 'AAPL'
```

In [45]:
```
def create_output_dirs():
    os.makedirs(data_path, exist_ok=True)
    os.makedirs(results_path, exist_ok=True)
```

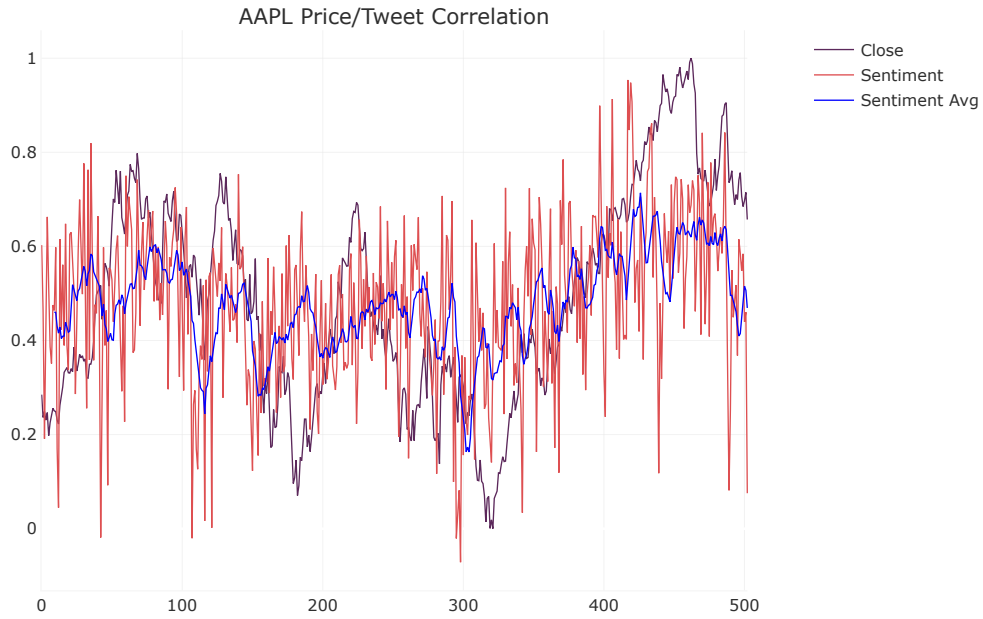
In [46]:
```
def fetch_price_data(symbol):
    #  valid periods: 1d,5d,1mo,3mo,6mo,1y,2y,5y,10y,ytd,max
    period = '2y'
    #  valid intervals: 1m,2m,5m,15m,30m,60m,90m,1h,1d,5d,1wk,1mo,3mo
    interval = '1d'
    data = yf.download(tickers=symbol, period=period, interval=interval, progress=False)
    price_df = pd.DataFrame(data)
    price_df.reset_index(inplace=True)
    price_df.to_excel('price_df1.xlsx')
    return price_df
```

In [47]:
```
def fetch_sentiment(symbol, start_date_str, end_date_str):
    sentiment_file_name = f"{symbol}_tweet_sentiment_{start_date_str}_{end_date_str}.csv"
    sentiment_file_path = os.path.join(data_path, sentiment_file_name)

    #  Load news file from disk if available
    if os.path.exists(sentiment_file_path):
        news_df = pd.read_csv(sentiment_file_path)
        news_df['date'] = pd.to_datetime(news_df['date'])
        return news_df

    #  Fetch news from EOD Data
    news_base_url = "https://eodhistoricaldata.com/api/sentiments"
    request_url = f"{news_base_url}?api_token={news_api_token}&s={symbol}&from={start_date_str}&to={end_date_str}"
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:101.0) Gecko/20100101 Firefox/101.0',
        'Accept': 'application/json',
        'Accept-Language': 'en-US,en;q=0.5',
    }

    try:
        #  Send GET request
        response = requests.get(request_url, headers=headers)
        response_obj = response.json()
        data_array = response_obj[f"{symbol}.US"]
        #  Format: {"AAPL.US":[{"date":"2022-10-02","count":4,"normalized":0.302},{"date":"2022-10-01","count":8,"normalized":0.617}]}
        sentiment_df = pd.DataFrame({}, columns=['date', 'count', 'normalized'])
        for item in data_array:
            row_df = pd.DataFrame({'date': [item['date']], 'count': [item['count']], 'normalized': [item['normalized']]})
            sentiment_df = pd.concat([sentiment_df, row_df], axis=0, ignore_index=True)

        sentiment_df.to_csv(sentiment_file_path)
        sentiment_df['date'] = pd.to_datetime(sentiment_df['date'])
        sentiment_df.to_excel('sentiment_df1.xlsx')
        return sentiment_df

    except:
        print('Error loading sentiments')
        return None
```

In [48]:
```
def fetch_and_process_sentiment(price_df, symbol):
    start_date = price_df.iloc[0]['Date']
    start_date_str = start_date.strftime("%Y-%m-%d")
    today = dt.datetime.now()
    end_date_str = today.strftime("%Y-%m-%d")

    #  Get Tweet sentiment
    sentiment_df = fetch_sentiment(symbol, start_date_str, end_date_str)

    # Iterate through price date and add tweet sentiment
    for i, row in price_df.iterrows():
        current_date = row['Date']

        sentiment_series = sentiment_df[sentiment_df['date'] == current_date]['normalized']
        if sentiment_series is not None and not sentiment_series.empty:
            price_df.at[i, 'sentiment'] = sentiment_series.values[0]
    price_df.to_excel('price_df2.xlsx')
    return price_df
```

In [49]:
```
def plot_results(symbol, price_corr_df):
    light_palette = {}
    light_palette["bg_color"] = "#ffffff"
    light_palette["plot_bg_color"] = "#ffffff"
    light_palette["grid_color"] = "#e6e6e6"
    light_palette["text_color"] = "#2e2e2e"
    light_palette["border_color"] = "#2e2e2e"
    light_palette["color_1"] = "#5c285b"
    light_palette["color_5"] = "#de4f51"
    palette = light_palette

    #  Create sub plots
    fig = make_subplots(rows=1, cols=1, subplot_titles=[f"{symbol} Price/Tweet Correlation"], \
                        specs=[[{ "secondary_y": True}]])

    #  Plot close price
    fig.add_trace(go.Scatter(x=price_corr_df.index, y=price_corr_df['close_scaled']
                             , line=dict(color=palette['color_1'], width=1)
                             , name="Close"),             row=1, col=1)

    #  Add Correlation
    fig.add_trace(go.Scatter(x=price_corr_df.index, y=price_corr_df['sentiment_scaled']
                             , line=dict(color=palette['color_5'], width=1)
                             , name="Sentiment"),        row=1, col=1)

    #  Moving average of corr
    price_corr_df['sentiment_scaled_avg'] = price_corr_df['sentiment_scaled'].rolling(10).mean()
    fig.add_trace(go.Scatter(x=price_corr_df.index, y=price_corr_df['sentiment_scaled_avg']
                             , line=dict(color="blue", width=1)
                             , name="Sentiment Avg"),   row=1, col=1)

    fig.update_layout(
            title={'text': '', 'x': 0.5},
            font=dict(family="Verdana", size=12, color=palette["text_color"]),
            autosize=True,
            width=800, height=600,
            xaxis={"rangeslider": {"visible": False}},
            plot_bgcolor=palette["plot_bg_color"],
            paper_bgcolor=palette["bg_color"])
    fig.update_yaxes(visible=False, secondary_y=True)

    #  Change grid color
    fig.update_xaxes(showline=True, linewidth=1, linecolor=palette["grid_color"]
                     ,gridcolor=palette["grid_color"])
    fig.update_yaxes(showline=True, linewidth=1, linecolor=palette["grid_color"]
                     ,gridcolor=palette["grid_color"])

    fig.show()
```

In [50]:
```
plot_results(symbol, price_corr_df)
```



In [ ]:

---

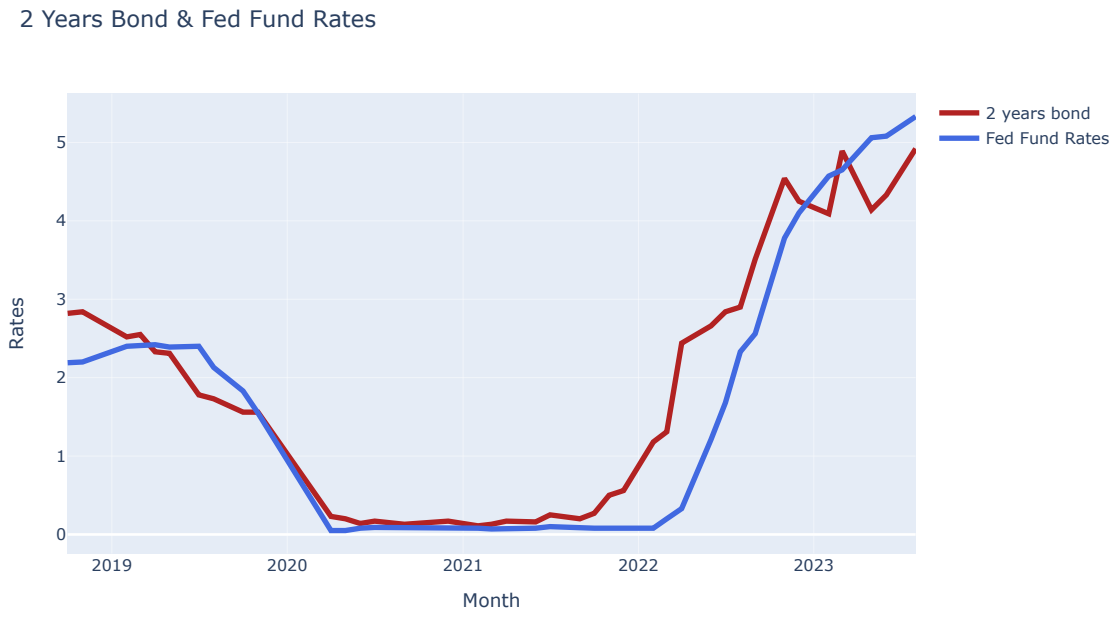
5. Bond By Yhasreen

5a. Interest Rates and Inflation Data

In [51]:
```
def plot_charts(combined_yield_df):
    month = combined_yield_df.index
    bond =combined_yield_df.DGS2
    fedfund = combined_yield_df.FEDFUNDS

    fig = go.Figure()
    fig.add_trace(go.Scatter(x=month, y=bond, name='2 years bond',
                             line=dict(color='firebrick', width=4)))
    fig.add_trace(go.Scatter(x=month, y=fedfund, name = 'Fed Fund Rates',
                             line=dict(color='royalblue', width=4)))

    fig.update_layout(title='2 Years Bond & Fed Fund Rates',
                      xaxis_title='Month',
                      yaxis_title='Rates')


    fig.show()
```

In [52]:
```
if __name__ == '__main__':
    #  Set start date
    today_date = datetime.utcnow()
    start_date = today_date - timedelta(days=5 * 365)
    start_date_str = datetime.strftime(start_date, "%Y-%m-%d")

    #  Federal Reserve Economic Data Service
    data_source = 'fred'
    two_year_treasury_code = 'FEDFUNDS'
    ten_year_treasury_code = 'DGS2'

    #  Fetch data
    two_year_yield_df = pdr.DataReader(two_year_treasury_code, data_source, start_date)
    ten_year_df = pdr.DataReader(ten_year_treasury_code, data_source, start_date)

    #  Merge the two yield dfs
    combined_yield_df = pd.merge(two_year_yield_df, ten_year_df, how='inner'
                                 , left_index=True, right_index=True)
```

```
combined_yield_df.dropna(inplace=True)

#  Plot charts
plot_charts(combined_yield_df)
```

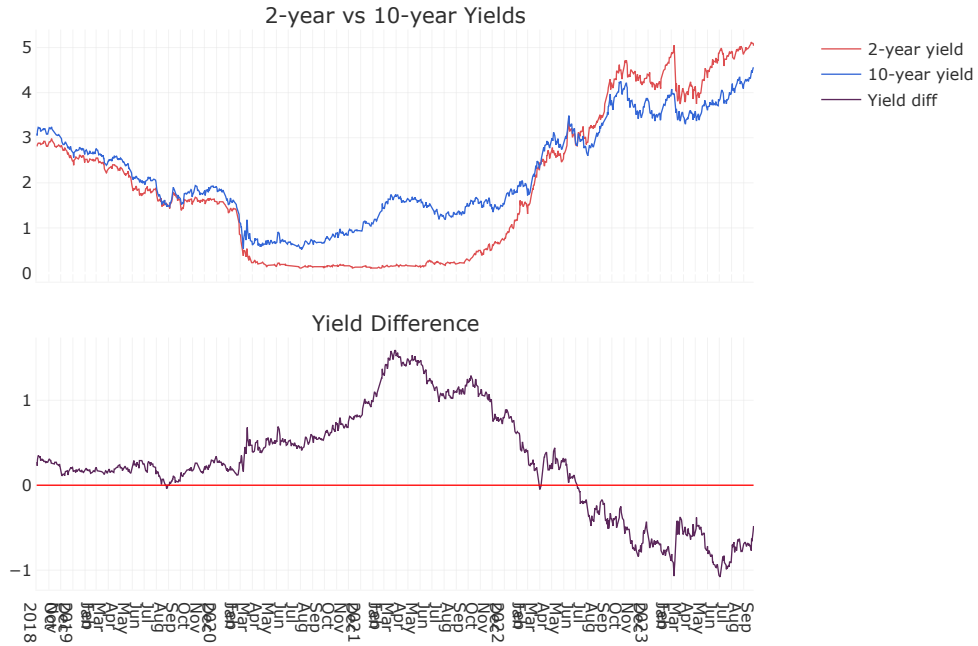


In [53]:

```
def plot_charts(combined_yield_df):
    light_palette = {}
    light_palette["bg_color"] = "#ffffff"
    light_palette["plot_bg_color"] = "#ffffff"
    light_palette["grid_color"] = "#e6e6e6"
    light_palette["text_color"] = "#2e2e2e"
    light_palette["dark_candle"] = "#4d98c4"
    light_palette["light_candle"] = "#b1b7ba"
    light_palette["volume_color"] = "#c74e96"
    light_palette["border_color"] = "#2e2e2e"
    light_palette["color_1"] = "#5c285b"
    light_palette["color_2"] = "#802c62"
    light_palette["color_3"] = "#a33262"
    light_palette["color_4"] = "#c43d5c"
    light_palette["color_5"] = "#de4f51"
    light_palette["color_6"] = "#f26841"
    light_palette["color_7"] = "#fd862b"
    light_palette["color_8"] = "#ffa600"
    light_palette["color_9"] = "#3366d6"
    palette = light_palette

    #  Create sub plots
    fig = make_subplots(rows=2, cols=1, subplot_titles=[f"2-year vs 10-year Yields"
                                                        , 'Yield Difference'], \
                        specs=[[{"secondary_y": True}], [{"secondary_y": True}]], \
                        vertical_spacing=0.1, shared_xaxes=True)

    #  Plot yields
    fig.add_trace(go.Scatter(x=combined_yield_df.index, y=combined_yield_df['DGS2']
                             , line=dict(color=palette['color_5']
                            , width=1), name="2-year yield"), row=1, col=1)
    fig.add_trace(go.Scatter(x=combined_yield_df.index, y=combined_yield_df['DGS10']
                             , line=dict(color=palette['color_9']
                                         , width=1), name="10-year yield"), row=1, col=1)

    # Plot yield curve
    fig.add_trace(go.Scatter(x=combined_yield_df.index, y=combined_yield_df['yield_diff']
                             , line=dict(color=palette['color_1'], width=1), name="Yield diff")
                             , row=2, col=1)

    fig.add_hline(y=0, line=dict(color='red', width=1), row=2, col=1)

    fig.update_layout(
            title={'text': '', 'x': 0.5},
            font=dict(family="Verdana", size=12, color=palette["text_color"]),
            autosize=True,
            width=800, height=600,
```

```
            xaxis={"rangeslider": {"visible": False}},
            plot_bgcolor=palette["plot_bg_color"],
            paper_bgcolor=palette["bg_color"])
    fig.update_layout(yaxis2=dict(range=[230, 300]))

    fig.update_xaxes(
        dtick="M1",
        tickformat="%b\n%Y",
        ticklabelmode="period")

    #  Change grid color
    fig.update_xaxes(showline=True, linewidth=1, linecolor=palette["grid_color"], gridcolor=palette["grid_color"])
    fig.update_yaxes(showline=True, linewidth=1, linecolor=palette["grid_color"], gridcolor=palette["grid_color"])


    fig.show()
```

In [54]:
```
if __name__ == '__main__':
    #  Set start date
    today_date = datetime.utcnow()
    start_date = today_date - timedelta(days=5 * 365)
    start_date_str = datetime.strftime(start_date, "%Y-%m-%d")

    #  Federal Reserve Economic Data Service
    data_source = 'fred'
    two_year_treasury_code = 'DGS2'
    ten_year_treasury_code = 'DGS10'

    #  Fetch data
    two_year_yield_df = pdr.DataReader(two_year_treasury_code, data_source, start_date)
    ten_year_df = pdr.DataReader(ten_year_treasury_code, data_source, start_date)

    #  Merge the two yield dfs
    combined_yield_df = pd.merge(two_year_yield_df, ten_year_df, how='inner'
                                 ,left_index=True,right_index=True)
    combined_yield_df.dropna(inplace=True)

    #  Calculate yield difference
    combined_yield_df['yield_diff'] = combined_yield_df['DGS10'] - combined_yield_df['DGS2']

    #  Plot charts
    plot_charts(combined_yield_df)
```



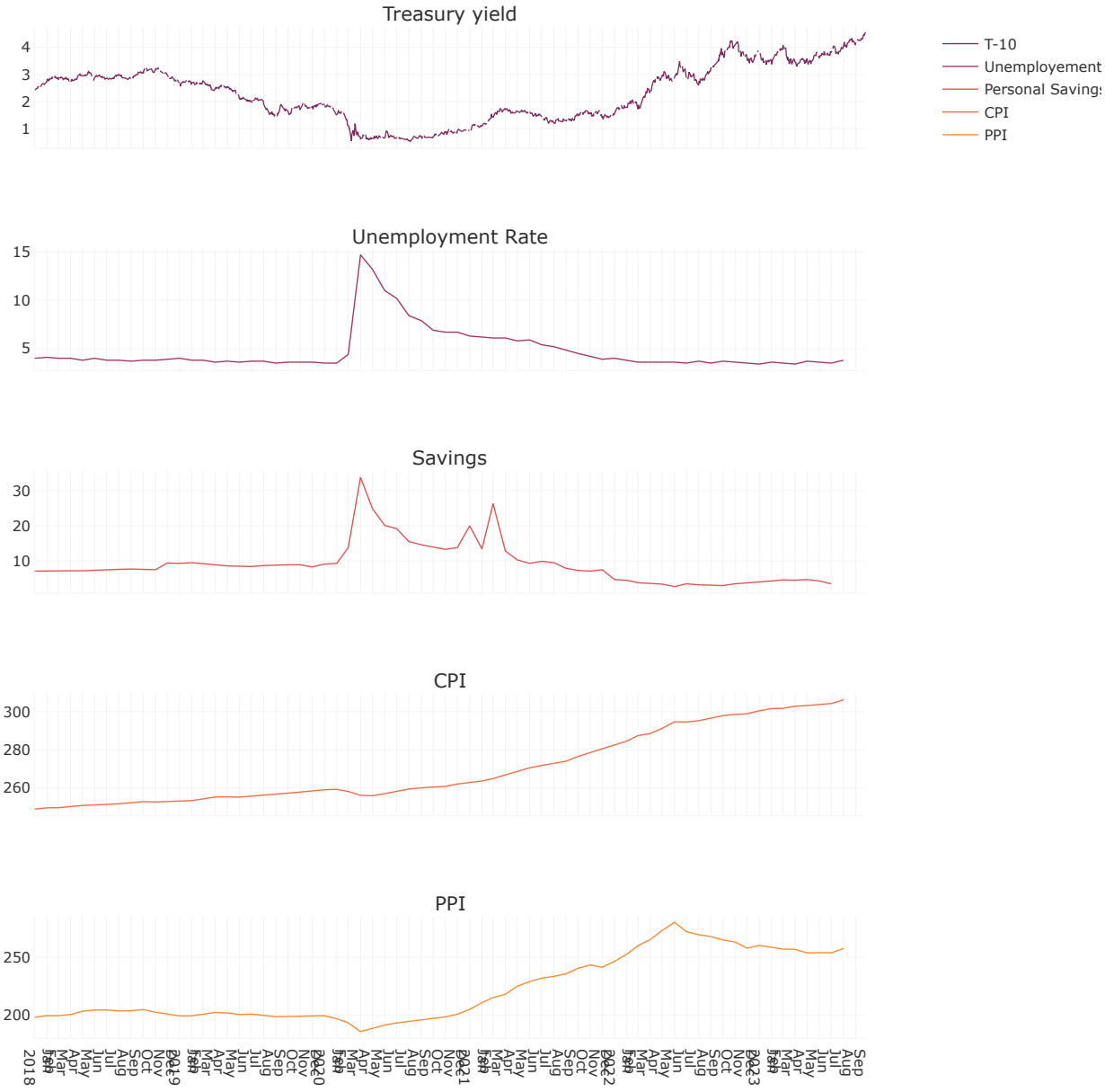
In [ ]:
```
```

5b. Economic Indicator

In [55]:
```
import datetime as dt
def plot_charts(treasury_yield_df, unemployment_rate_df \
                , savings_df, cpi_df ,ppi_df):
    light_palette = {}
    light_palette["bg_color"] = "#ffffff"
    light_palette["plot_bg_color"] = "#ffffff"
    light_palette["grid_color"] = "#e6e6e6"
    light_palette["text_color"] = "#2e2e2e"
    light_palette["dark_candle"] = "#4d98c4"
    light_palette["light_candle"] = "#b1b7ba"
    light_palette["volume_color"] = "#c74e96"
    light_palette["border_color"] = "#2e2e2e"
    light_palette["color_1"] = "#5c285b"
    light_palette["color_2"] = "#802c62"
    light_palette["color_3"] = "#a33262"
    light_palette["color_4"] = "#c43d5c"
    light_palette["color_5"] = "#de4f51"
    light_palette["color_6"] = "#f26841"
    light_palette["color_7"] = "#fd862b"
    light_palette["color_8"] = "#ffa600"
    light_palette["color_9"] = "#3366d6"
    palette = light_palette

    #  Create sub plots
    fig = make_subplots(rows=5, cols=1, subplot_titles=[f'Treasury yield'
                                                        , 'Unemployment Rate' \
                                                        , 'Savings', 'CPI' ,'PPI'] \
                        ,specs=[[{"secondary_y": True}], [{"secondary_y": True}] \


                                , [{"secondary_y": True}] \
                                , [{"secondary_y": True}] , [{"secondary_y": True}] ] \
                        ,vertical_spacing=0.1, shared_xaxes=True)

    #  Plots
    fig.add_trace(go.Scatter(x=treasury_yield_df.index, y=treasury_yield_df['DGS10']
                             , line=dict(color=palette['color_2']
                             , width=1), name="T-10"),row=1, col=1)
    fig.add_trace(go.Scatter(x=unemployment_rate_df.index, y=unemployment_rate_df['UNRATE']
                             , line=dict(color=palette['color_3'], width=1)
                             , name="Unemployement rate"),row=2, col=1)
    fig.add_trace(go.Scatter(x=savings_df.index, y=savings_df['PSAVERT']
                             , line=dict(color=palette['color_5'], width=1)
                             , name="Personal Savings Rate"),  row=3, col=1)
    fig.add_trace(go.Scatter(x=cpi_df.index, y=cpi_df['CPIAUCSL']
                             , line=dict(color=palette['color_6']
                             , width=1), name="CPI"), row=4, col=1)
    fig.add_trace(go.Scatter(x=ppi_df.index, y=ppi_df['PPIACO']
                             , line=dict(color=palette['color_7']
                             , width=1), name="PPI"), row=5, col=1)


    fig.update_layout(
            title={'text': '', 'x': 0.5},
            font=dict(family="Verdana", size=12, color=palette["text_color"]),
            autosize=True,
            width=1024, height=1024,
            xaxis={"rangeslider": {"visible": False}},
            plot_bgcolor=palette["plot_bg_color"],
            paper_bgcolor=palette["bg_color"])
    fig.update_layout(yaxis2=dict(range=[230, 300]))

    fig.update_xaxes(     dtick="M1",   tickformat="%b\n%Y",  ticklabelmode="period")

    #  Change grid color
    fig.update_xaxes(showline=True, linewidth=1
                     , linecolor=palette["grid_color"], gridcolor=palette["grid_color"])
    fig.update_yaxes(showline=True, linewidth=1
                     , linecolor=palette["grid_color"], gridcolor=palette["grid_color"])


    fig.show()
```

In [56]:
```
if __name__ == '__main__':

    #  Set start date
    start_date = dt.date(2018, 1, 1)

    start_date_str = dt.datetime.strftime(start_date, "%Y-%m-%d")
    #end = datetime.datetime(2023, 1, 1)

    #  Federal Reserve Economic Data Service
    data_source = 'fred'
```

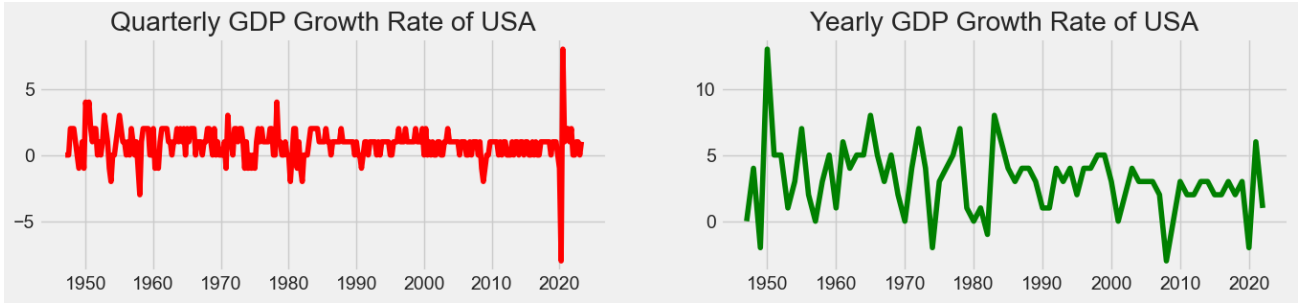
```
treasury_yield_code = 'DGS10' #  10-year Treasury Rate
unemployment_rate_code = 'UNRATE'
gdp_code = 'GDPC1'


personal_savings_rate_code = 'PSAVERT'
cpi_series = 'CPIAUCSL'
ppi_series = 'PPIACO'

#  Fetch data
treasury_yield_df = pdr.DataReader(treasury_yield_code, data_source, start_date)
unemployment_rate_df = pdr.DataReader(unemployment_rate_code, data_source, start_date)

savings_df = pdr.DataReader(personal_savings_rate_code, data_source, start_date)
cpi_df = pdr.DataReader(cpi_series, data_source, start_date)
ppi_df = pdr.DataReader(ppi_series, data_source, start_date)

#  Plot charts
plot_charts(treasury_yield_df, unemployment_rate_df ,savings_df, cpi_df ,ppi_df)
#print('Done.')
```



In [ ]:

In [57]:
```
def growth_rate(value,df):
    global initial_value
    if df[0] == value:
        initial_value = value
        return 0

    else:
        rate = f'{((value - initial_value) /initial_value) :.2f}'
        initial_value = value
        return float(rate) * 100
```

In [58]:
```
#retrieving the GDP figures
gdp = fred.get_series('GDPC1') #quarterly data
gdp.dropna(axis=0,inplace=True)
gdp_df = gdp.to_frame(name='gdp')
gdp_df['quarter'] = gdp_df.index
gdp_df.index = [index for index in  range(len(gdp_df))]

gdp_df['gdp_growth'] = gdp_df['gdp'].apply(growth_rate, df=gdp_df['gdp'])
gdp_df['quarter'] = pd.to_datetime(gdp_df['quarter'])
gdp_df['month'] = gdp_df['quarter'].dt.month

#gdp growth rate per year
gdp_year_df=gdp_df.loc[gdp_df['month']==10].copy()
gdp_year_df['year'] = gdp_year_df['quarter'].dt.year
gdp_year_df.reset_index(drop=True, inplace=True)
gdp_year_df.drop('month',axis=1,inplace=True)
gdp_year_df.drop('quarter',axis=1,inplace=True)

gdp_year_df['gdp_growth'] = gdp_year_df['gdp'].apply(growth_rate,df=gdp_year_df['gdp'])

#plotting gdp_growth_rate
plt.figure(figsize=(15,3))
plt.subplot(121)
plt.plot(gdp_df['quarter'] ,gdp_df['gdp_growth'],'r')
plt.title('Quarterly GDP Growth Rate of USA')
#-----------------------------------------------------
plt.subplot(122)
plt.plot(gdp_year_df['year'],gdp_year_df['gdp_growth'],'g')
plt.title('Yearly GDP Growth Rate of USA')
#-----------------------------------------------------

plt.show()
```



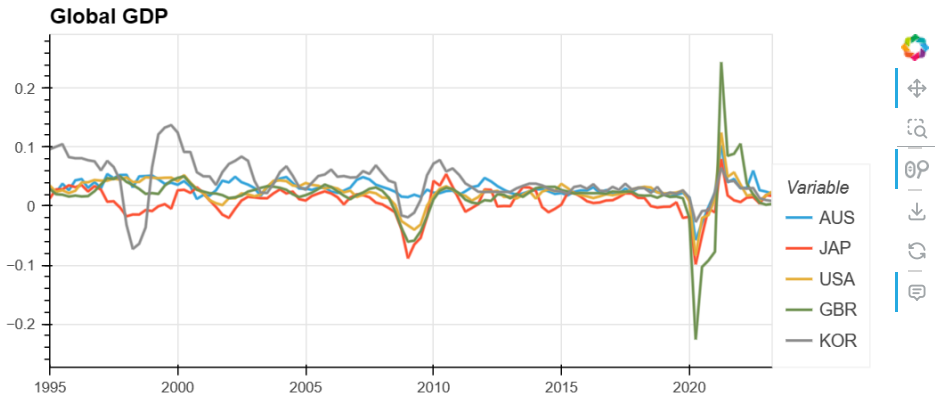
In [ ]:

5c. Global Macroeconomic Factors

In [59]:
```
columns = ['AUS','JAP','USA','GBR','KOR']

aus = fred.get_series('NGDPRSAXDCAUQ')
jap = fred.get_series('JPNRGDPEXP')
us = fred.get_series('GDPC1')
uk = fred.get_series('NGDPRSAXDCGBQ')
korea = fred.get_series('NGDPRSAXDCKRQ')

#Combine all the single series, only take common dates using "inner"
gdp_global = pd.concat([aus, jap, us, uk, korea], join='inner', axis=1)

# Create YoY Real GDP (divided by 4 because we have quarterly data) & Drop NA's
gdp_global = ((gdp_global / gdp_global.shift(4)) - 1).dropna(axis = 0)
gdp_global.columns = columns

# Plot
```

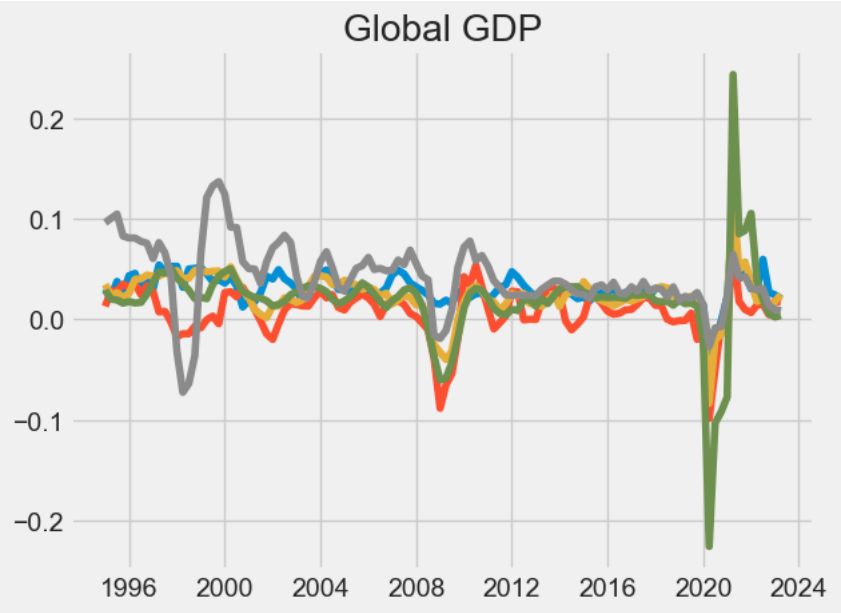
```
plot1=gdp_global.hvplot(title = 'Global GDP', grid = True)
gdp_global.hvplot(title = 'Global GDP', grid = True)
```

Out[59]:



In [60]:

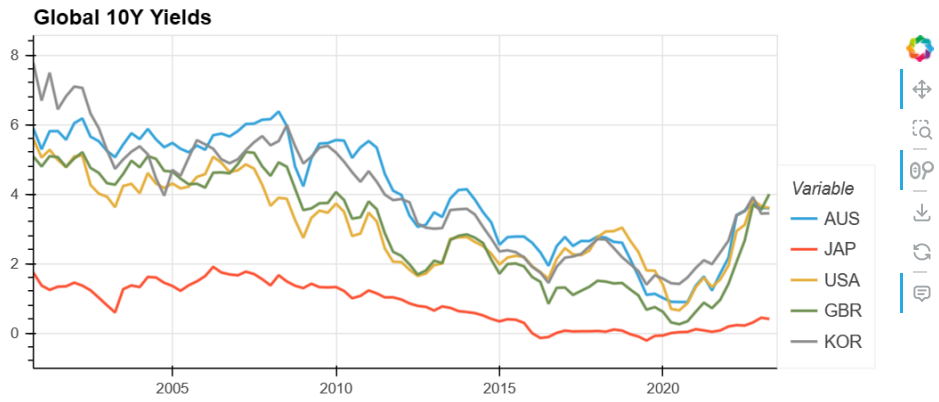
```
plt.plot(gdp_global)
plt.title('Global GDP')
plt.show()
```



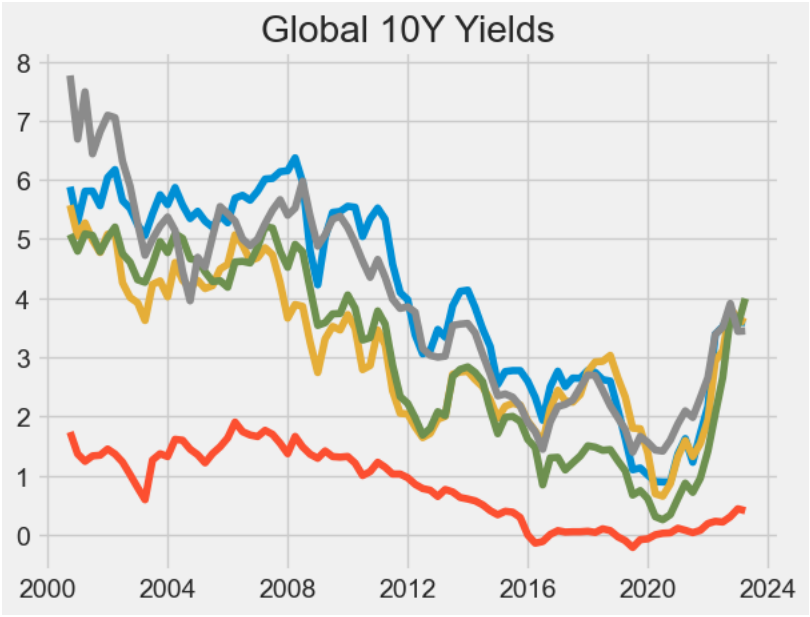
In [61]:

```
# Extract the 10Y Government Bond Yields for each country (make quarterly):
aus_y = fred.get_series('IRLTLT01AUM156N', frequency = 'q')
jap_y = fred.get_series('IRLTLT01JPM156N', frequency = 'q')
us_y = fred.get_series('IRLTLT01USM156N', frequency = 'q')
uk_y = fred.get_series('IRLTLT01GBM156N', frequency = 'q')
korea_y = fred.get_series('IRLTLT01KRM156N', frequency = 'q')

# Combine series
global_10y_yields = pd.concat([aus_y, jap_y, us_y, uk_y, korea_y], join='inner', axis=1)
global_10y_yields.columns = columns

# Plot
plot2=global_10y_yields.hvplot(title = 'Global 10Y Yields', grid = True)
global_10y_yields.hvplot(title = 'Global 10Y Yields', grid = True)
```

Out[61]:



In [62]:
```
plt.plot(global_10y_yields)
plt.title('Global 10Y Yields')
plt.show()
```



In [ ]:
```
```

5d. Market Sentiment

In [64]:
```
df = pd.read_csv('tweetdata.csv')
df = df.drop(['Unnamed: 0','Date', 'User','likeCount','replyCount','retweetCount','location'],axis=1)
#df.isnull().sum()
```

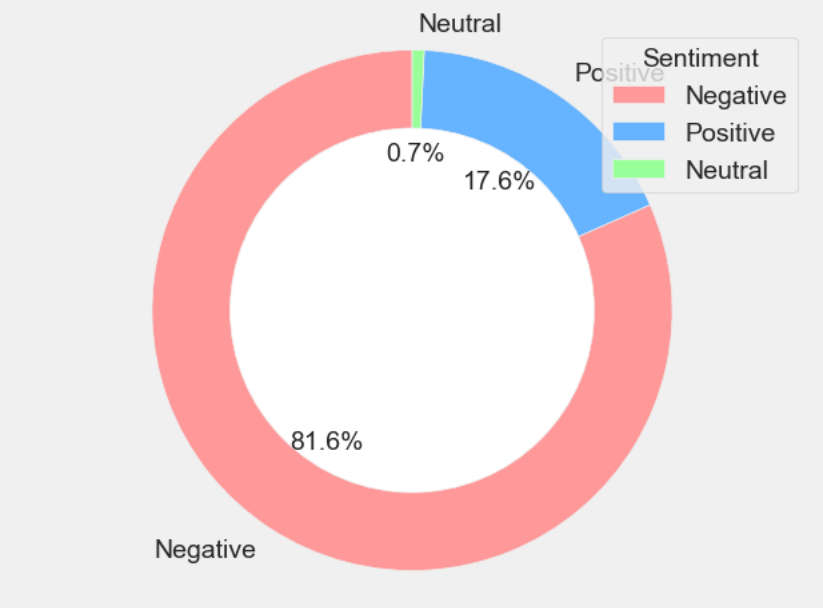
In [65]:
```
def clean(text):
    text = str(text).lower()
    text = re.sub(r'https?:\/\/\S+', '', text)
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    return text

df['Tweet'] = df['Tweet'].apply(clean)
```

In [66]:
```
#nltk.download('vader_lexicon')
for index, row in df['Tweet'].iteritems():
  score = SentimentIntensityAnalyzer().polarity_scores(row)
  neg = score['neg']
  neu = score['neu']
  pos = score['pos']
  if neg > pos:
    df.loc[index, 'Sentiment'] = 'Negative'
  elif pos > neg:
    df.loc[index, 'Sentiment'] = 'Positive'
  else:
    df.loc[index, 'Sentiment'] = 'Neutral'
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_22720\4173848573.py:2: FutureWarning:

iteritems is deprecated and will be removed in a future version. Use .items instead.
```

In [67]:
```
def count_tp_in_column(data,feature):
  total = data.loc[:,feature].value_counts(dropna=False)
  percentage = data.loc[:,feature].value_counts(dropna=False, normalize=True)*100
  return pd.concat([total, round(percentage,2)], axis=1, keys=['Total', 'Percentage'])

tp = count_tp_in_column(df,'Sentiment')
```

In [68]:
```
labels = tp.index
sizes = tp['Percentage']
colors = ['#ff9999','#66b3ff','#99ff99']

fig1, ax1 = plt.subplots()
ax1.pie(sizes,colors=colors,labels=labels ,autopct='%1.1f%%' ,startangle=90)

centre_circle = plt.Circle((0,0),0.70,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

ax1.axis('equal')
plt.tight_layout()
plt.legend(title='Sentiment')

plt.show()
```



In [69]:
```
all_tweets = " ".join(tweet for tweet in df['Tweet'])
wordcloud = WordCloud(width=1024 ,height=512 ,random_state=21 ,max_font_size=110
                      ,background_color='white').generate(all_tweets)

plt.figure(figsize=(20, 10))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')

plt.show()
```

In [ ]:

Scenario 6

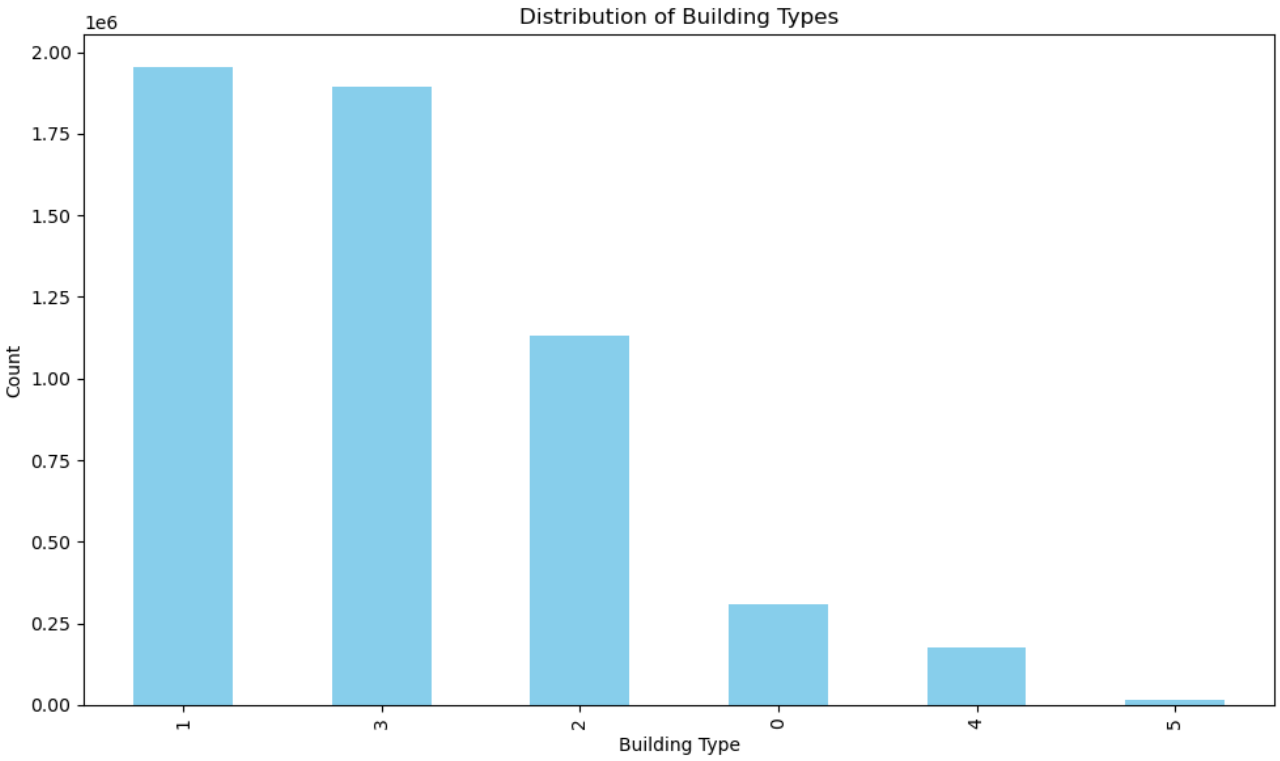
In [3]:
```
#Import necessary libraries
import datetime
from datetime import date
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import yfinance as yf
import os
```

In [7]:
```
data = pd.read_csv('all_v2.csv')
data.head(3)
```

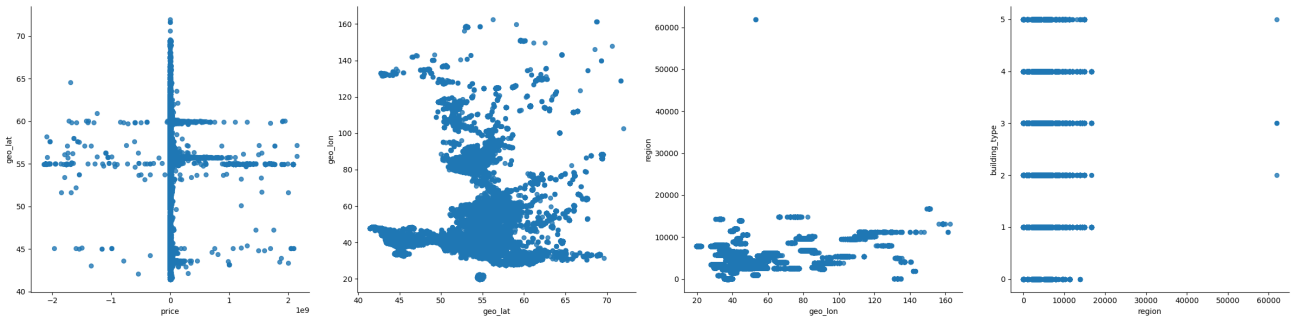
Out[7]:

	price	date	time	geo_lat	geo_lon	region	building_type	level	levels	rooms	area	kitchen_area	object_type
0	6050000	2018-02-19	20:00:21	59.805808	30.376141	2661	1	8	10	3	82.6	10.8	1
1	8650000	2018-02-27	12:04:54	55.683807	37.297405	81	3	5	24	2	69.1	12.0	1
2	4000000	2018-02-28	15:44:00	56.295250	44.061637	2871	1	5	9	3	66.0	10.0	1

In [5]:
```
from pandas.core.base import value_counts
Building_type_column = 'building_type'
building_type_counts = data[Building_type_column].value_counts()

plt.figure(figsize=(10,6))
building_type_counts.plot(kind = 'bar', color = 'skyblue')
plt.xlabel('Building Type')
plt.ylabel('Count')
plt.title('Distribution of Building Types')

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



In [6]:
```
#csv_file_path = "C:\\Users\\EBEN\\Downloads\\GWP1\\all_v2.csv"
#data = pd.read_csv(csv_file_path)

def scatter_plots(df, colname_pairs, figscale=1, alpha=0.8):
    plt.figure(figsize=(len(colname_pairs) * 6 * figscale, 6 * figscale))

    for plot_i, (x_colname, y_colname) in enumerate(colname_pairs, start=1):
        ax = plt.subplot(1, len(colname_pairs), plot_i)
        df.plot(kind='scatter', x=x_colname, y=y_colname, s=(32 * figscale), alpha=alpha, ax=ax)
        ax.spines[['top', 'right']].set_visible(False)

    plt.tight_layout()
    plt.show()

colname_pairs = [['price', 'geo_lat'], ['geo_lat', 'geo_lon'], ['geo_lon', 'region'], ['region', 'building_type']]

scatter_plots(data, colname_pairs)
```



Scatter Plot 1: This scatter plot shows the relationship between property prices and geographic latitude. It appears that there is no clear linear relationship between price and latitude, but there might be some clustering of data points.

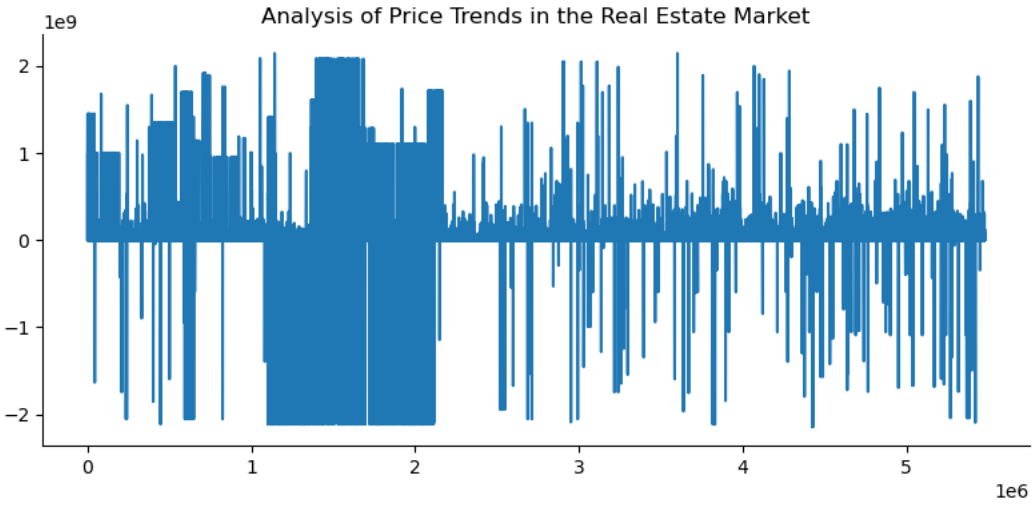
Scatter Plot 2: This scatter plot displays the geographic distribution of data points based on latitude and longitude. It helps visualize the spatial distribution of properties in the dataset.

Scatter Plot 3: This scatter plot shows how geographic longitude is related to different regions. It may reveal whether certain regions are associated with specific longitudes.

Scatter Plot 4: This scatter plot visualizes the distribution of building types across different regions. It can help identify which building types are common in specific regions.

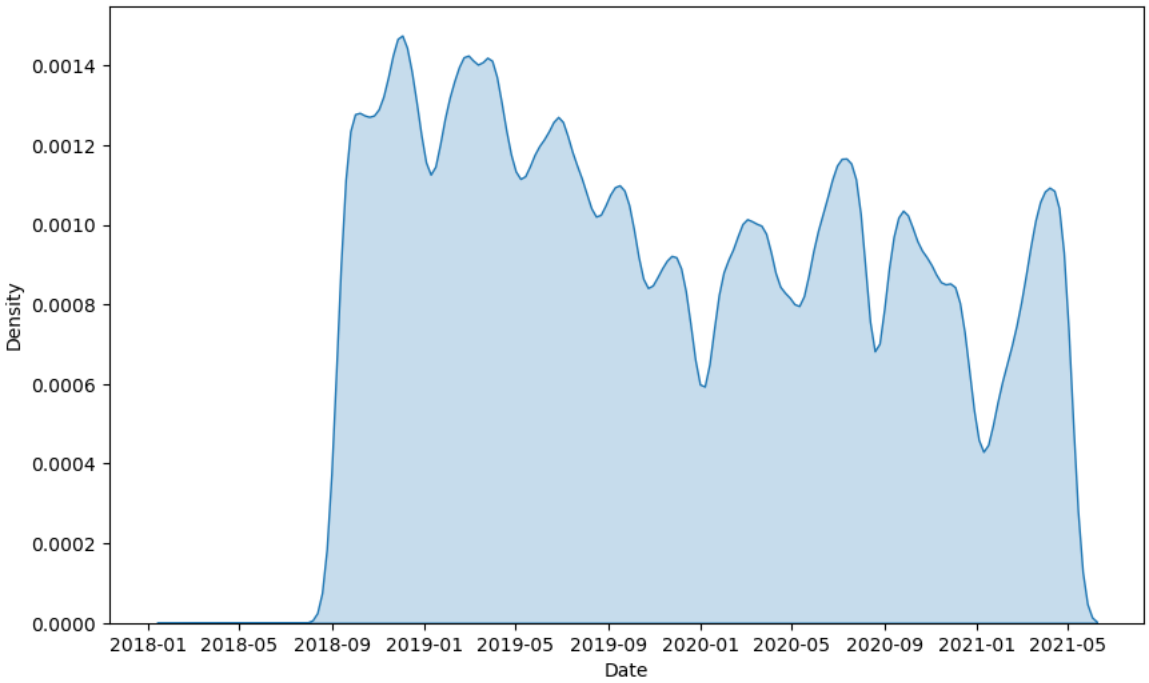
In [8]:
```
def value_plot(data, y, figscale=1):
    data[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale), title=y)
```

```
    plt.gca().spines[['top', 'right']].set_visible(False)
    plt.tight_layout()
    plt.title('Analysis of Price Trends in the Real Estate Market')

# Replace 'price' with the column name from your CSV file that you want to plot
chart = value_plot(data, 'price', figscale=1)
plt.show()
```



The histogram illustrates the distribution of property prices, showing whether they are concentrated in certain price ranges or spread out uniformly. Observe the shape of the distribution. If it is skewed to the left (negatively skewed), it indicates a concentration of lower-priced properties. If it is skewed to the right (positively skewed), it indicates a concentration of higher-priced properties.

In [9]:
```
data['date'] = pd.to_datetime(data['date'])


plt.figure(figsize=(10, 6))
sns.kdeplot(data=data['date'], fill=True)
plt.xlabel('Date')
plt.ylabel('Density')
plt.show()
```



End of Project

In [ ]:
```

```