$Name : Mukaila\ Rafiu$

$Course : Financial\ Economics$

$Home\ Work\ 1$

```
In [ ]:
```

```
In [195…
import pandas as pd
import numpy as np
from plotnine import *
import tidyfinance as tf
import yfinance as yf
import statsmodels.api as sm
from scipy.stats import pearsonr, spearmanr
```

A

```
In [196…
prices = tf.download_data(
domain="stock_prices",
symbols=["^GSPC","^SP500TR"],
start_date="2000-01-01",
end_date="2025-08-31"
)
prices.head(10).round(10)
```

Out[196...

	date	symbol	volume	open	low	high	close	adjuste
0	2000-01-03	^GSPC	931800000	1469.250000	1438.359985	1478.000000	1455.219971	1455
1	2000-01-04	^GSPC	1009000000	1455.219971	1397.430054	1455.219971	1399.420044	1399
2	2000-01-05	^GSPC	1085500000	1399.420044	1377.680054	1413.270020	1402.109985	1402
3	2000-01-06	^GSPC	1092300000	1402.109985	1392.099976	1411.900024	1403.449951	1403
4	2000-01-07	^GSPC	1225200000	1403.449951	1400.729980	1441.469971	1441.469971	1441
5	2000-01-10	^GSPC	1064800000	1441.469971	1441.469971	1464.359985	1457.599976	1457
6	2000-01-11	^GSPC	1014000000	1457.599976	1434.420044	1458.660034	1438.560059	1438
7	2000-01-12	^GSPC	974600000	1438.560059	1427.079956	1442.599976	1432.250000	1432
8	2000-01-13	^GSPC	1030400000	1432.250000	1432.250000	1454.199951	1449.680054	1449
9	2000-01-14	^GSPC	1085900000	1449.680054	1449.680054	1473.000000	1465.150024	1465

In [197...

```python
#Getting the prices of last day of each month
# Extract data for both symbols and get last day of each month
monthly_prices = (
    prices
    .assign(month_end = lambda x: x['date'] + pd.offsets.MonthEnd(0))
    .groupby(['symbol', 'month_end'])
    .last()
    .reset_index()
    .sort_values(['symbol', 'month_end'])
)
```

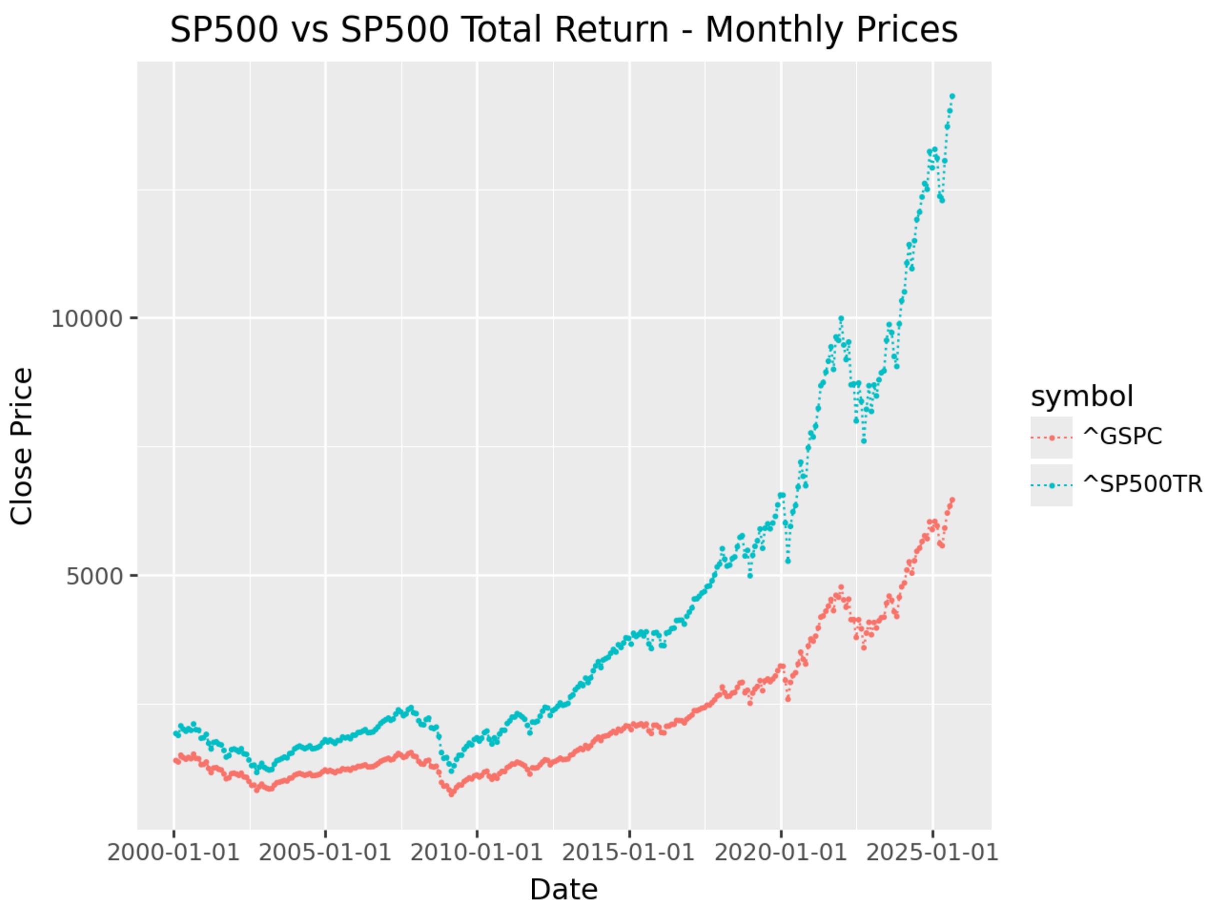
In [198...

```python
# Displaying the monthly prices
print("Monthly prices for both symbols (first 10 rows):")
print(monthly_prices.head(10).round(10))

print("\nMonthly prices for both symbols (last 10 rows):")
print(monthly_prices.tail(10).round(10))
```

```
Monthly prices for both symbols (first 10 rows):
  symbol   month_end       date      volume         open          low  \
0  ^GSPC 2000-01-31 2000-01-31   993800000  1360.160034  1350.140015
1  ^GSPC 2000-02-29 2000-02-29  1204300000  1348.050049  1348.050049
2  ^GSPC 2000-03-31 2000-03-31  1227400000  1487.920044  1484.380005
3  ^GSPC 2000-04-30 2000-04-28   984600000  1464.920044  1448.150024
4  ^GSPC 2000-05-31 2000-05-31   960500000  1422.439941  1415.500000
5  ^GSPC 2000-06-30 2000-06-30  1459700000  1442.390015  1438.709961
6  ^GSPC 2000-07-31 2000-07-31   952600000  1419.890015  1418.709961
7  ^GSPC 2000-08-31 2000-08-31  1056600000  1502.589966  1502.589966
8  ^GSPC 2000-09-30 2000-09-29  1197100000  1458.290039  1436.290039
9  ^GSPC 2000-10-31 2000-10-31  1366400000  1398.660034  1398.660034

          high        close  adjusted_close
0  1394.479980  1394.459961     1394.459961
1  1369.630005  1366.420044     1366.420044
2  1519.810059  1498.579956     1498.579956
3  1473.619995  1452.430054     1452.430054
4  1434.489990  1420.599976     1420.599976
5  1454.680054  1454.599976     1454.599976
6  1437.650024  1430.829956     1430.829956
7  1525.209961  1517.680054     1517.680054
8  1458.290039  1436.510010     1436.510010
9  1432.219971  1429.400024     1429.400024

Monthly prices for both symbols (last 10 rows):
       symbol  month_end       date  volume          open           low  \
606  ^SP500TR 2024-11-30 2024-11-29       0  13164.870117  13164.870117
607  ^SP500TR 2024-12-31 2024-12-31       0  12995.490234  12883.879883
608  ^SP500TR 2025-01-31 2025-01-31       0  13394.990234  13250.629883
609  ^SP500TR 2025-02-28 2025-02-28       0  12883.179688  12841.650391
610  ^SP500TR 2025-03-31 2025-03-31       0  12175.349609  12089.049805
611  ^SP500TR 2025-04-30 2025-04-30       0  12122.919922  11977.389648
612  ^SP500TR 2025-05-31 2025-05-30       0  13031.429688  12900.120117
613  ^SP500TR 2025-06-30 2025-06-30       0  13687.099609  13646.559570
614  ^SP500TR 2025-07-31 2025-07-31       0  14214.240234  13994.509766
615  ^SP500TR 2025-08-31 2025-08-29       0  14368.919922  14269.990234

             high         close  adjusted_close
606  13252.589844  13227.129883    13227.129883
607  13017.179688  12911.820312    12911.820312
608  13447.650391  13271.379883    13271.379883
609  13107.589844  13098.219727    13098.219727
610  12394.179688  12360.209961    12360.209961
611  12303.549805  12276.389648    12276.389648
612  13071.900391  13049.129883    13049.129883
613  13735.009766  13712.709961    13712.709961
614  14214.240234  14020.459961    14020.459961
615  14373.309570  14304.679688    14304.679688
```

In [199…

```
#ploting the prices
(ggplot(monthly_prices)
 + aes(x = 'date', y = 'close', color = 'symbol')
 + geom_point(size = 0.2)
 + geom_line(linetype = "dotted")
 + labs(x = "Date", y = "Close Price", title = "SP500 vs SP500 Total Return - Month
)
```



B

```
In [91]: # Load Shiller data
```

```
In [92]: # Load and clean Shiller data
shiller_data = pd.read_excel(r'C:\Users\gon_c\Downloads\ie_data.xls',
                             sheet_name="Data", skiprows=7, usecols=[0, 1, 2],
                             names=['Date', 'P', 'D'])
shiller_data = shiller_data[pd.to_numeric(shiller_data.P, errors='coerce').notna()]
shiller_data[['P', 'D']] = shiller_data[['P', 'D']].apply(pd.to_numeric)
shiller_data['Date'] = pd.to_datetime(shiller_data.Date.astype(str) + '.01')
shiller_data = shiller_data.query('Date >= "1871-01-01" and Date <= "2025-08-31"')
```

```
In [93]: # Show first 10 rows and check for September 2023
print("First 10 rows, (Prices 'P', and Dividend 'D' in 1871):")
print(shiller_data.head(10).round(4))

print("\nLast 10 rows, (Prices 'P', and Dividend 'D' up to 2023):")
print(shiller_data.tail(10).round(4))
```

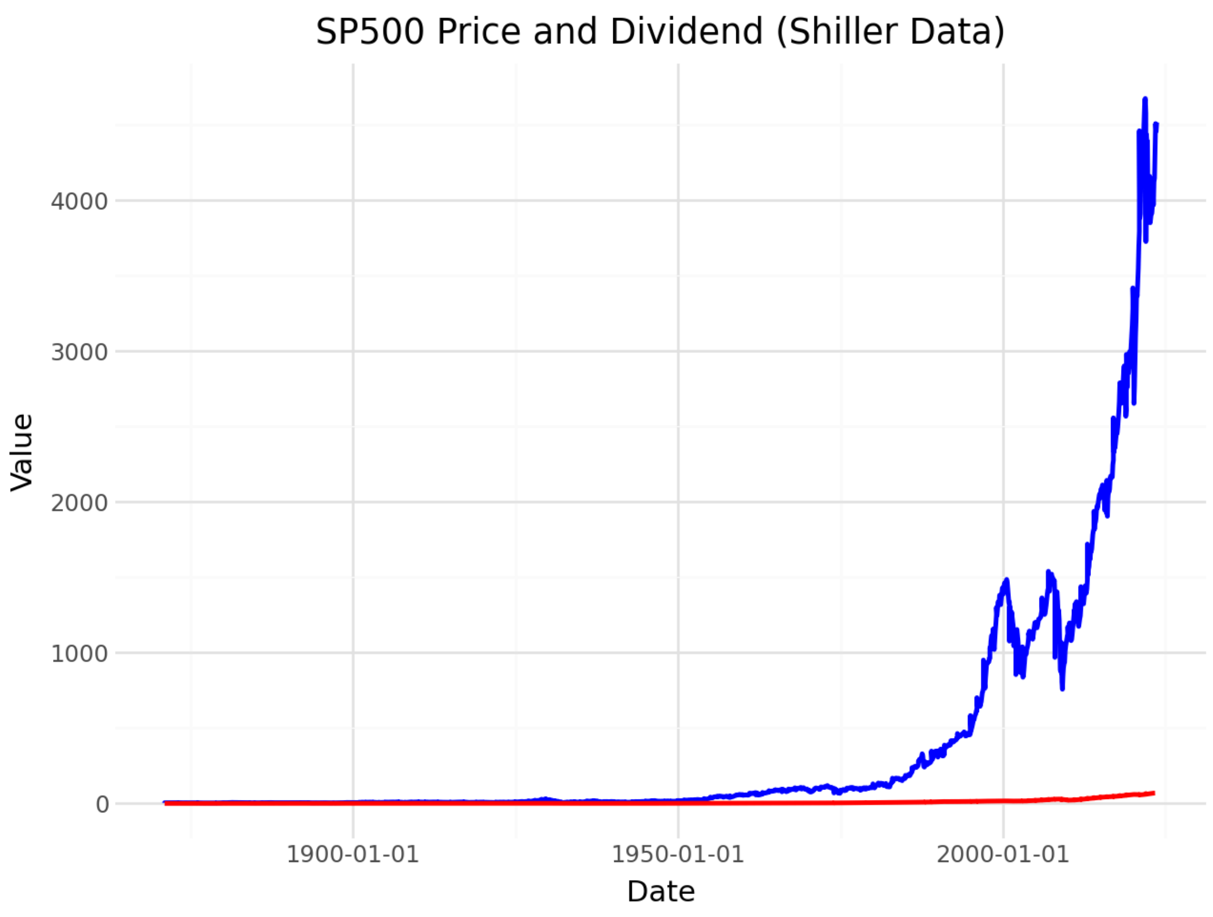
```
First 10 rows, (Prices 'P', and Dividend 'D' in 1871):
        Date     P     D
0 1871-01-01  4.44  0.26
1 1871-02-01  4.50  0.26
2 1871-03-01  4.61  0.26
3 1871-04-01  4.74  0.26
4 1871-05-01  4.86  0.26
5 1871-06-01  4.82  0.26
6 1871-07-01  4.73  0.26
7 1871-08-01  4.79  0.26
8 1871-09-01  4.84  0.26
9 1871-01-01  4.59  0.26

Last 10 rows, (Prices 'P', and Dividend 'D' up to 2023):
           Date          P        D
1823 2022-12-01  3912.3810  66.9200
1824 2023-01-01  3960.6565  67.3500
1825 2023-02-01  4079.6847  67.7800
1826 2023-03-01  3968.5591  68.2100
1827 2023-04-01  4121.4674  68.3767
1828 2023-05-01  4146.1732  68.5433
1829 2023-06-01  4345.3729  68.7100
1830 2023-07-01  4508.0755      NaN
1831 2023-08-01  4457.3587      NaN
1832 2023-09-01  4515.7700      NaN
```

In [94]:
```
# Checking some statistics for P and D only
print("\nBasic statistics for Prices and Dividends:")
print(shiller_data[['P', 'D']].describe().round(4))
```

```
Basic statistics for Prices and Dividends:
               P          D
count  1833.0000  1830.0000
mean    376.9708     7.4818
std     808.2854    13.6461
min       2.7300     0.1800
25%       7.9300     0.4233
50%      18.0700     0.9284
75%     186.2000     7.7067
max    4674.7727    68.7100
```

In [95]:
```
(ggplot(shiller_data)
 + aes(x='Date')
 + geom_line(aes(y='P'), color='blue', size=1)
 + geom_line(aes(y='D'), color='red', size=1)
 + labs(title='SP500 Price and Dividend (Shiller Data)',
        x='Date',
        y='Value')
 + theme_minimal()
)
```

```
C:\Users\gon_c\anaconda3\Lib\site-packages\plotnine\geoms\geom_path.py:100: Plotnine
Warning: geom_path: Removed 3 rows containing missing values.
```



C

First and foremost, Shiller's data uses historical reconstruction and academic sources, while Yahoo Finance provides real-time market data. Shiller's work aims for long-term consistency, even if it means some adjustments to historical figures.

Shiller uses monthly average prices rather than end-of-month closing prices. From the Shiller's data, August 2023 price is 4,457.36, which would be the average of all August trading days, not just the closing price on August 31st.

Shiller's data is often presented in inflation-adjusted terms, though the raw "P" column appears to be nominal prices. There might still be some normalization for consistency across the very long time series.

In the nutshell, Shiller prioritizes academic consistency across 150+ years, while Yahoo Finance prioritizes real-time accuracy for current market participants.

D

In [96]:
```
# Calculate approximate monthly dividends (divide by 12)
shiller_data['D_monthly'] = shiller_data['D'] / 12

# Calculate monthly total returns: ret = (P + D_monthly) / lag(P) - 1
shiller_data['P_lag'] = shiller_data['P'].shift(1)
```

```
shiller_data['ret'] = (shiller_data['P'] + shiller_data['D_monthly']) / shiller_dat
shiller_data_returns = shiller_data.dropna()

print("Shiller data with monthly returns - First 10 rows:")
print(shiller_data_returns[['Date', 'P', 'D', 'D_monthly', 'ret']].head(10).round(6
print(f"\nDate range with returns: {shiller_data_returns['Date'].min()} to {shiller
print(f"Number of months with returns: {len(shiller_data_returns)}")
```

```
Shiller data with monthly returns - First 10 rows:
         Date     P     D  D_monthly       ret
1  1871-02-01  4.50  0.26   0.021667  0.018393
2  1871-03-01  4.61  0.26   0.021667  0.029259
3  1871-04-01  4.74  0.26   0.021667  0.032899
4  1871-05-01  4.86  0.26   0.021667  0.029887
5  1871-06-01  4.82  0.26   0.021667 -0.003772
6  1871-07-01  4.73  0.26   0.021667 -0.014177
7  1871-08-01  4.79  0.26   0.021667  0.017266
8  1871-09-01  4.84  0.26   0.021667  0.014962
9  1871-01-01  4.59  0.26   0.021667 -0.047176
10 1871-11-01  4.64  0.26   0.021667  0.015614

Date range with returns: 1871-01-01 00:00:00 to 2023-06-01 00:00:00
Number of months with returns: 1829
```

In [97]:

```
# Summary statistics for returns
print("\nReturn statistics:")
print(shiller_data_returns['ret'].describe().round(6))

# Show the last rows in 2023
print("\nLast 10 rows in 2023:")
shiller_2023 = shiller_data_returns[shiller_data_returns['Date'] >= '2023-01-01']
print(shiller_2023[['Date', 'P', 'D', 'D_monthly', 'ret']].round(6))
```

```
Return statistics:
count    1829.000000
mean        0.008156
std         0.040604
min        -0.261879
25%        -0.011400
50%         0.009889
75%         0.030863
max         0.513085
Name: ret, dtype: float64

Last 10 rows in 2023:
           Date            P          D  D_monthly       ret
1824 2023-01-01  3960.656500  67.350000   5.612500  0.013774
1825 2023-02-01  4079.684737  67.780000   5.648333  0.031479
1826 2023-03-01  3968.559130  68.210000   5.684167 -0.025845
1827 2023-04-01  4121.467368  68.376667   5.698056  0.039966
1828 2023-05-01  4146.173182  68.543333   5.711944  0.007380
1829 2023-06-01  4345.372857  68.710000   5.725833  0.049425
```

In [ ]:

E

```
In [212…
# Show the full Shiller data with all calculated returns
print("Full Shiller Data with Returns:")
print("=" * 50)
print(shiller_full[['P', 'D', 'Total_Return', 'real_return', 'excess_return']].head

print("\nRecent Shiller Data with Returns:")
print("=" * 50)
print(shiller_full[['P', 'D', 'Total_Return', 'real_return', 'excess_return']].tail
```

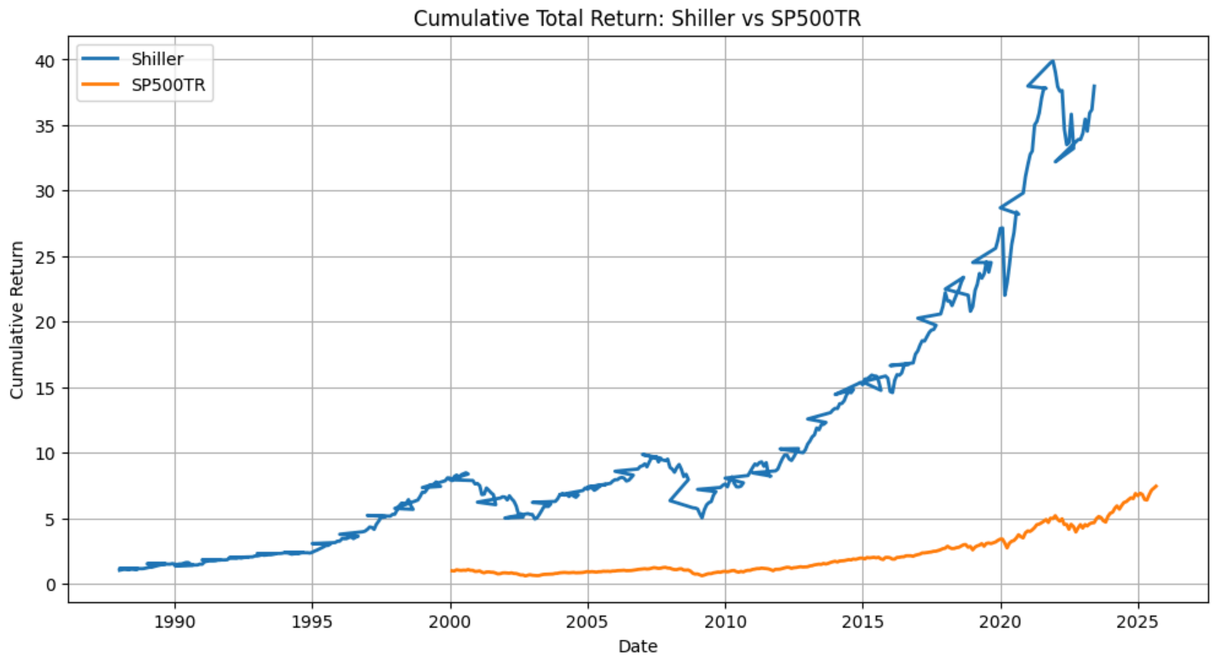
```
Full Shiller Data with Returns:
==================================================
               P     D  Total_Return  real_return  excess_return
Date
1871-01-01  4.44  0.26           NaN          NaN            NaN
1871-01-01  4.50  0.26      0.018393    -0.011781       0.014062
1871-01-01  4.61  0.26      0.029259     0.014230       0.024925
1871-01-01  4.74  0.26      0.032899     0.072026       0.028563
1871-02-01  4.86  0.26      0.029887     0.053836       0.025548
1871-02-01  4.82  0.26     -0.003772     0.011920      -0.008114
1871-02-01  4.73  0.26     -0.014177    -0.014177      -0.018522
1871-02-01  4.79  0.26      0.017266     0.033538       0.012918
1871-02-01  4.84  0.26      0.014962    -0.008825       0.010612
1871-02-01  4.59  0.26     -0.047176    -0.061832      -0.051529

Recent Shiller Data with Returns:
==================================================
                      P          D  Total_Return  real_return  excess_return
Date
2022-02-01  3912.380952  66.920000      0.000120     0.003200      -0.002848
2023-01-01  3960.656500  67.350000      0.013774     0.005733       0.010879
2023-01-01  4079.684737  67.780000      0.031479     0.025753       0.028406
2023-01-01  3968.559130  68.210000     -0.025845    -0.028542      -0.028845
2023-01-01  4121.467368  68.376667      0.039966     0.034179       0.037127
2023-02-01  4146.173182  68.543333      0.007380     0.004850       0.004453
2023-02-01  4345.372857  68.710000      0.049425     0.046048       0.046353
2023-02-01  4508.075500        NaN           NaN          NaN            NaN
2023-02-01  4457.358696        NaN           NaN          NaN            NaN
2023-02-01  4515.770000        NaN           NaN          NaN            NaN
```

```
In [213…
# Calculate gross returns and cumulative returns for Shiller data
shiller_data["R_t"] = shiller_data["ret"] + 1  # Gross return R_t = r_t + 1

# Filter post 1988
shiller88 = shiller_data[shiller_data.index >= '1988-01-01'].copy()
sp500tr_88 = monthly_prices[monthly_prices['symbol'] == '^SP500TR'].copy()
sp500tr_88 = sp500tr_88[sp500tr_88['date'] >= '1988-01-01']

# Cumulative return for Shiller
shiller88["cum_return_shiller"] = shiller88["R_t"].cumprod()
sp500tr_88 = sp500tr_88.sort_values('date')
sp500tr_88["cum_return_sp500tr"] = sp500tr_88["close"] / sp500tr_88["close"].iloc[0

print("Shiller Cumulative Returns (first 5 rows):")
print(shiller88[["P", "D", "ret", "R_t", "cum_return_shiller"]].head().round(3))
print("\nSP500TR Cumulative Returns (first 5 rows):")
print(sp500tr_88[["date", "close", "cum_return_sp500tr"]].head().round(3))
```

```
Shiller Cumulative Returns (first 5 rows):
                P      D    ret    R_t  cum_return_shiller
Date
1988-01-01  250.5  8.857  0.042  1.042               1.042
1988-02-01  258.1  8.903  0.033  1.033               1.077
1988-03-01  265.7  8.950  0.032  1.032               1.112
1988-04-01  262.6  9.043 -0.009  0.991               1.102
1988-05-01  256.1  9.137 -0.022  0.978               1.078

SP500TR Cumulative Returns (first 5 rows):
           date    close  cum_return_sp500tr
308  2000-01-31  1919.84               1.000
309  2000-02-29  1883.50               0.981
310  2000-03-31  2067.76               1.077
311  2000-04-28  2005.55               1.045
312  2000-05-31  1964.40               1.023
```

In [203…

```
# Plot cumulative returns
plt.figure(figsize=(12, 6))
plt.plot(shiller88.index, shiller88["cum_return_shiller"], label="Shiller", linewid
plt.plot(sp500tr_88["date"], sp500tr_88["cum_return_sp500tr"], label="SP500TR", lin
plt.title("Cumulative Total Return: Shiller vs SP500TR")
plt.xlabel("Date")
plt.ylabel("Cumulative Return")
plt.legend()
plt.grid(True)
plt.show()
```



In [217…

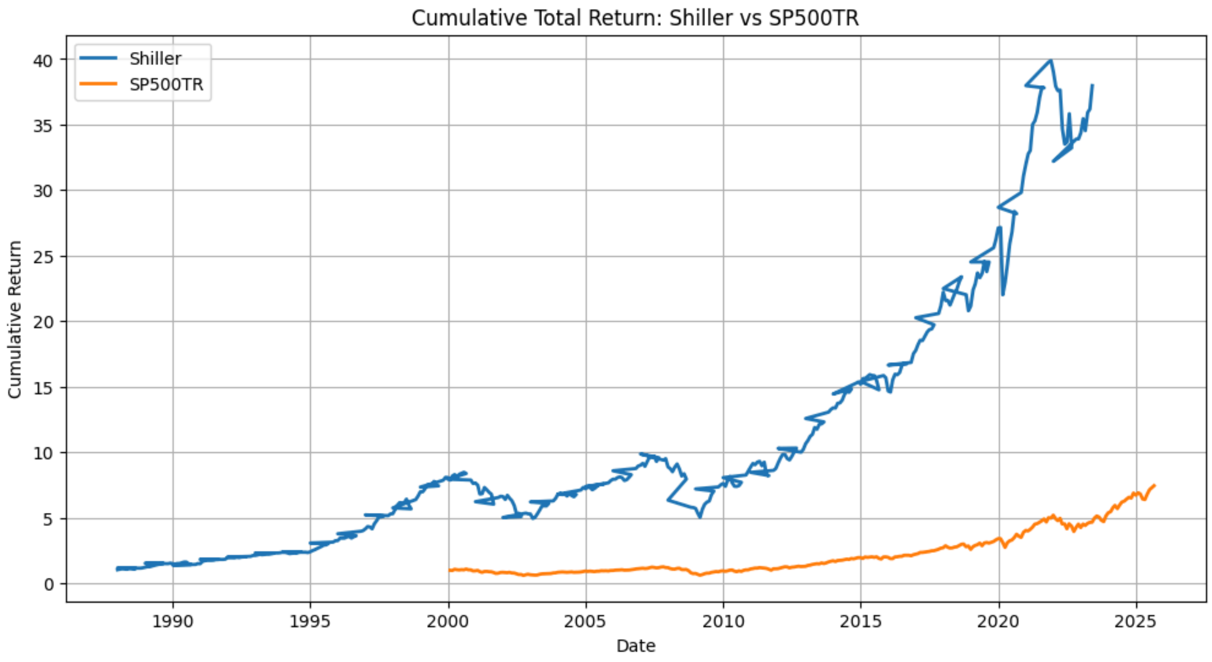
```
# Regression and correlation analysis
common_dates = shiller88.index.intersection(sp500tr_88['date'])
shiller_returns_aligned = shiller88.loc[common_dates, 'ret']
sp500tr_returns_aligned = sp500tr_88.set_index('date').loc[common_dates, 'close'].p
```

```
final_dates = shiller_returns_aligned.index.intersection(sp500tr_returns_aligned.in
shiller_final = shiller_returns_aligned.loc[final_dates]
sp500tr_final = sp500tr_returns_aligned.loc[final_dates]

df_analysis = pd.DataFrame({
    'shiller_return': shiller_final,
    'sp500tr_return': sp500tr_final
}).dropna()
```

In [218…

```
# Calculate gross returns and cumulative returns for Shiller data
shiller_data["R_t"] = shiller_data["ret"] + 1  # Gross return R_t = r_t + 1

# Filter post 1988
shiller88 = shiller_data[shiller_data.index >= '1988-01-01'].copy()
sp500tr_88 = monthly_prices[monthly_prices['symbol'] == '^SP500TR'].copy()
sp500tr_88 = sp500tr_88[sp500tr_88['date'] >= '1988-01-01']

# Cumulative return for Shiller
shiller88["cum_return_shiller"] = shiller88["R_t"].cumprod()

# Cumulative return for SP500TR (already compounded)
sp500tr_88 = sp500tr_88.sort_values('date')
sp500tr_88["cum_return_sp500tr"] = sp500tr_88["close"] / sp500tr_88["close"].iloc[0

print("Shiller Cumulative Returns (first 5 rows):")
print(shiller88[["P", "D", "ret", "R_t", "cum_return_shiller"]].head().round(3))

print("\nSP500TR Cumulative Returns (first 5 rows):")
print(sp500tr_88[["date", "close", "cum_return_sp500tr"]].head().round(3))

# Plot cumulative returns
plt.figure(figsize=(12, 6))
plt.plot(shiller88.index, shiller88["cum_return_shiller"], label="Shiller", linewid
plt.plot(sp500tr_88["date"], sp500tr_88["cum_return_sp500tr"], label="SP500TR", lin
plt.title("Cumulative Total Return: Shiller vs SP500TR")
plt.xlabel("Date")
plt.ylabel("Cumulative Return")
plt.legend()
plt.grid(True)
plt.show()
```

```
Shiller Cumulative Returns (first 5 rows):
                P      D    ret    R_t  cum_return_shiller
Date
1988-01-01  250.5  8.857  0.042  1.042               1.042
1988-02-01  258.1  8.903  0.033  1.033               1.077
1988-03-01  265.7  8.950  0.032  1.032               1.112
1988-04-01  262.6  9.043 -0.009  0.991               1.102
1988-05-01  256.1  9.137 -0.022  0.978               1.078

SP500TR Cumulative Returns (first 5 rows):
          date    close  cum_return_sp500tr
308 2000-01-31  1919.84               1.000
309 2000-02-29  1883.50               0.981
310 2000-03-31  2067.76               1.077
311 2000-04-28  2005.55               1.045
312 2000-05-31  1964.40               1.023
```



In [ ]:

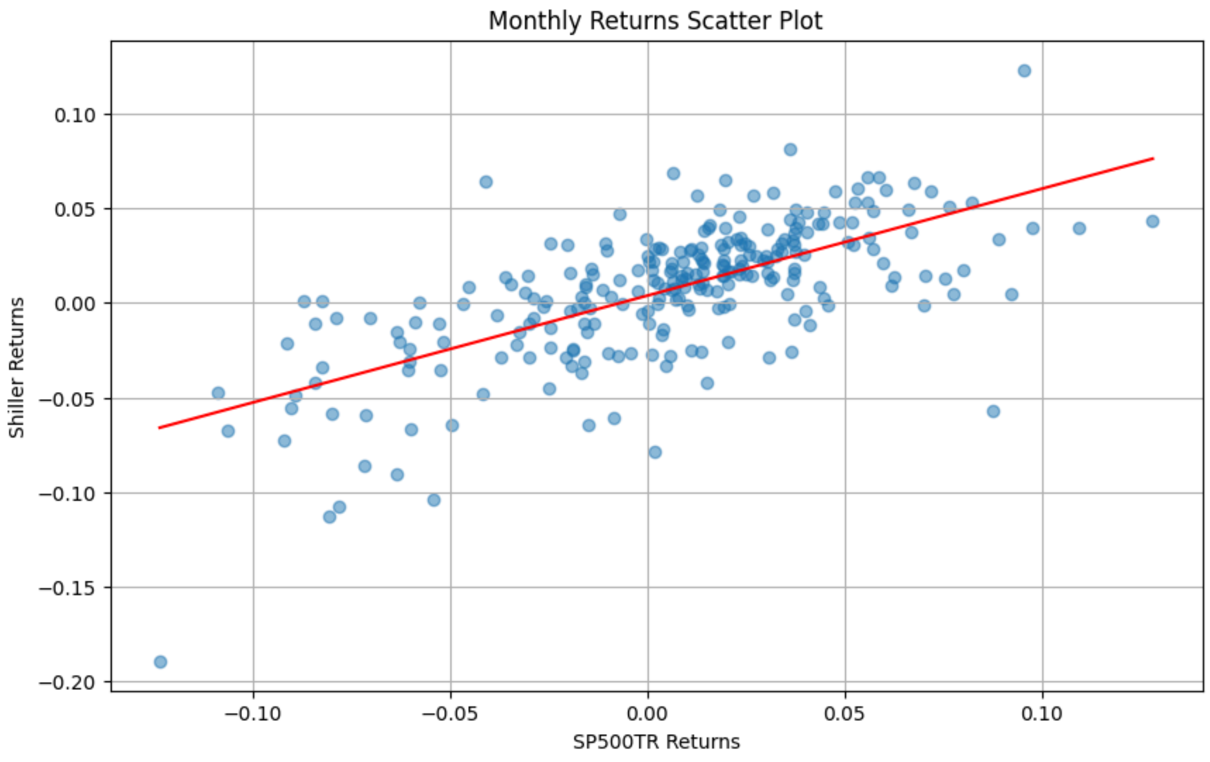
In [219…

```
shiller88['year_month'] = shiller88.index.strftime('%Y-%m')
sp500tr_88['year_month'] = sp500tr_88['date'].dt.strftime('%Y-%m')
sp500tr_88['sp500tr_return'] = sp500tr_88['close'].pct_change()
common_months = set(shiller88['year_month']).intersection(set(sp500tr_88['year_mont
df_analysis = pd.DataFrame([
    {
        'date': shiller88[shiller88['year_month'] == month].index[0],
        'shiller_return': shiller88[shiller88['year_month'] == month]['ret'].iloc[0
        'sp500tr_return': sp500tr_88[sp500tr_88['year_month'] == month]['sp500tr_re
    }
    for month in common_months
]).set_index('date').dropna()

print(f"Aligned data: {len(df_analysis)} months")

# Analysis
X = sm.add_constant(df_analysis["sp500tr_return"])
```

```
y = df_analysis["shiller_return"]
model = sm.OLS(y, X).fit()

pearson_corr, _ = pearsonr(df_analysis["sp500tr_return"], df_analysis["shiller_retu
spearman_corr, _ = spearmanr(df_analysis["sp500tr_return"], df_analysis["shiller_re
print(f"R-squared: {model.rsquared:.4f}")
print(f"Pearson: {pearson_corr:.4f}, Spearman: {spearman_corr:.4f}")
print("\nPost-1988 Stats:")
print(df_analysis.agg(["mean", "std"]).round(4))
print(f"\nFull Sample Stats:")
print(f"Mean: {shiller_data['ret'].mean():.4f}, Std: {shiller_data['ret'].std():.4f
# Plots
plt.figure(figsize=(10, 6))
plt.scatter(df_analysis["sp500tr_return"], df_analysis["shiller_return"], alpha=0.5
x_vals = np.linspace(df_analysis["sp500tr_return"].min(), df_analysis["sp500tr_retu
plt.plot(x_vals, model.params[0] + model.params[1] * x_vals, color='red')
plt.title("Monthly Returns Scatter Plot")
plt.xlabel("SP500TR Returns")
plt.ylabel("Shiller Returns")
plt.grid(True)
plt.show()
```

```
Aligned data: 258 months
R-squared: 0.4474
Pearson: 0.6689, Spearman: 0.6637

Post-1988 Stats:
      shiller_return  sp500tr_return
mean          0.0071          0.0058
std           0.0363          0.0430

Full Sample Stats:
Mean: 0.0082, Std: 0.0406
```

```
C:\Users\gon_c\AppData\Local\Temp\ipykernel_32144\1995821364.py:39: FutureWarning: S
eries.__getitem__ treating keys as positions is deprecated. In a future version, int
eger keys will always be treated as labels (consistent with DataFrame behavior). To
access a value by position, use `ser.iloc[pos]`
```



F

In [208…

```python
shiller_full = pd.read_excel(r'C:\Users\gon_c\Downloads\ie_data.xls',
                            sheet_name="Data", skiprows=7)
print("Working with actual column names...")
print("Date column sample:")
print(shiller_full['Date'].head(10))
shiller_full = shiller_full.dropna(subset=['Date'])
shiller_full = shiller_full[shiller_full['Date'] != '']
def convert_shiller_date(date_val):
    try:
        if isinstance(date_val, (int, float)):
            year = int(date_val)
            month = int(round((date_val - year) * 12 + 1))
            return pd.Timestamp(year=year, month=month, day=1)
        elif isinstance(date_val, str):
            if '.' in date_val:
                year_part, month_part = date_val.split('.')
                year = int(year_part)
                month = int(float('0.' + month_part) * 12 + 1)
                return pd.Timestamp(year=year, month=month, day=1)
    except:
        return pd.NaT
    return pd.NaT

shiller_full['Date'] = shiller_full['Date'].apply(convert_shiller_date)
shiller_full = shiller_full.dropna(subset=['Date'])
shiller_full = shiller_full.set_index('Date')
numeric_cols = ['P', 'D', 'E', 'CPI', 'Rate GS10']
for col in numeric_cols:
    shiller_full[col] = pd.to_numeric(shiller_full[col], errors='coerce')
shiller_full = shiller_full[(shiller_full.index >= '1871-01-01') & (shiller_full.in
print(f"\nFinal dataset: {len(shiller_full)} months from {shiller_full.index.min()}
print("\nKey columns for analysis:")
print(shiller_full[['P', 'D', 'CPI', 'Rate GS10']].head().round(3))
```

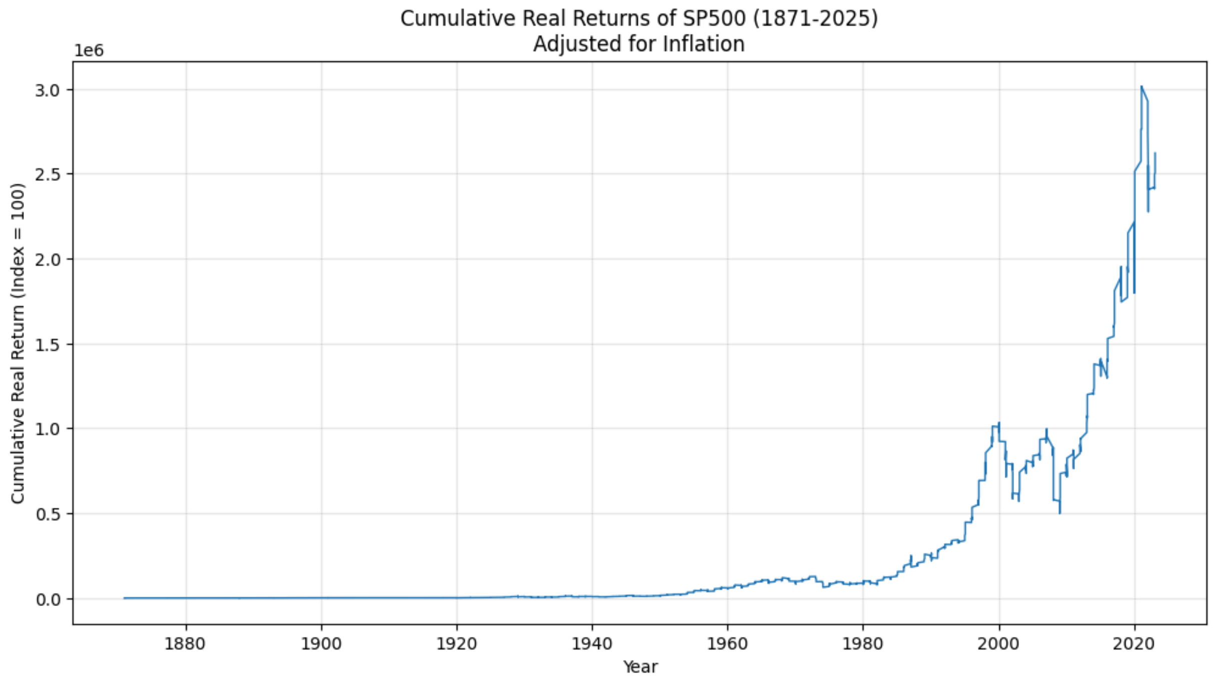
```
Working with actual column names...
Date column sample:
0    1871.01
1    1871.02
2    1871.03
3    1871.04
4    1871.05
5    1871.06
6    1871.07
7    1871.08
8    1871.09
9    1871.10
Name: Date, dtype: float64

Final dataset: 1833 months from 1871-01-01 00:00:00 to 2023-02-01 00:00:00

Key columns for analysis:
               P     D     CPI  Rate GS10
Date
1871-01-01  4.44  0.26  12.464      5.320
1871-01-01  4.50  0.26  12.845      5.323
1871-01-01  4.61  0.26  13.035      5.327
1871-01-01  4.74  0.26  12.559      5.330
1871-02-01  4.86  0.26  12.274      5.333
```

In [ ]:

In [209…

```
# Question F: Calculate Real Returns

# Calculate total returns (price return + dividend yield)
shiller_full['Price_Return'] = shiller_full['P'].pct_change()
shiller_full['Dividend_Yield'] = shiller_full['D'] / shiller_full['P'].shift(1) / 1
shiller_full['Total_Return'] = shiller_full['Price_Return'] + shiller_full['Dividen

# Calculate inflation from CPI
shiller_full['inflation'] = shiller_full['CPI'].pct_change()

# Calculate real returns: (1 + nominal return) / (1 + inflation) - 1
shiller_full['real_return'] = (1 + shiller_full['Total_Return']) / (1 + shiller_ful

# Calculate cumulative real returns
shiller_full['cum_real_return'] = (1 + shiller_full['real_return']).cumprod()

# Plot cumulative real returns
plt.figure(figsize=(12, 6))
plt.plot(shiller_full.index, shiller_full['cum_real_return'] * 100, linewidth=1)
plt.title('Cumulative Real Returns of SP500 (1871-2025)\nAdjusted for Inflation')
plt.ylabel('Cumulative Real Return (Index = 100)')
plt.xlabel('Year')
plt.grid(True, alpha=0.3)
plt.show()
```

```
print("Real returns calculated and plotted successfully")
```



```
Real returns calculated and plotted successfully
```

In [ ]:

G

In [224…

```
#Largest Historical Drawdowns in Real Returns

def analyze_drawdowns(series):
    dates = series.index
    values = series.values
    drawdowns = []

    peak = values[0]
    peak_date = dates[0]
    current_drawdown = None

    for i in range(len(values)):
        current_val = values[i]
        current_date = dates[i]
        if current_val > peak:
            peak = current_val
            peak_date = current_date
            if current_drawdown:
                current_drawdown['recovery_date'] = current_date
                current_drawdown['recovery_months'] = (current_date - current_drawd
                drawdowns.append(current_drawdown)
                current_drawdown = None

        current_dd = (current_val - peak) / peak

        # Start new drawdown if > 5% and not already in one
        if current_dd < -0.05 and current_drawdown is None:
            current_drawdown = {
                'start_date': peak_date,
                'peak_value': peak,
                'min_drawdown': current_dd,
                'trough_date': current_date,
                'trough_value': current_val
            }
        # Update existing drawdown
        elif current_drawdown and current_dd < current_drawdown['min_drawdown']:
            current_drawdown['min_drawdown'] = current_dd
            current_drawdown['trough_date'] = current_date
            current_drawdown['trough_value'] = current_val

    return drawdowns

# Calculate drawdowns for real returns
real_drawdowns = analyze_drawdowns(shiller_full['cum_real_return'].dropna())

# Get 5 largest drawdowns
largest_drawdowns = sorted(real_drawdowns, key=lambda x: x['min_drawdown'])[:5]

print("5 LARGEST HISTORICAL DRAWDOWNS IN REAL RETURNS")
print("=" * 70)
print("(Adjusted for Inflation, Total Returns including Dividends)")
print("=" * 70)

for i, dd in enumerate(largest_drawdowns, 1):
    # Calculate durations
    drawdown_months = (dd['trough_date'] - dd['start_date']).days // 30
    recovery_months = dd['recovery_months'] if 'recovery_months' in dd else "Still

    # Identify major historical events
    event_name = ""
    start_year = dd['start_date'].year

    if 1929 <= start_year <= 1932:
        event_name = " - Great Depression"
    elif 2000 <= start_year <= 2002:
        event_name = " - Dot-com Bubble"
    elif 2007 <= start_year <= 2009:
        event_name = " - Global Financial Crisis"
    elif 1973 <= start_year <= 1974:
        event_name = " - 1973-74 Oil Crisis"
    elif 1937 <= start_year <= 1938:
        event_name = " - 1937-38 Recession"
    elif 1916 <= start_year <= 1920:
        event_name = " - WWI and Post-War Recession"
    elif 1970 <= start_year <= 1970:
        event_name = " - 1970 Recession"

    print(f"{i}. {dd['start_date'].strftime('%B %Y')} to {dd['trough_date'].strftim
    print(f"   Peak: {dd['peak_value']:.2f}, Trough: {dd['trough_value']:.2f}")
    print(f"   Drawdown: {dd['min_drawdown']*100:.1f}%")
    print(f"   Duration to bottom: {drawdown_months} months")
    print(f"   Full recovery: {recovery_months} months")
    print()
print("SUMMARY")
print("=" * 70)
print(f"Analysis period: {shiller_full.index.min().year} to {shiller_full.index.max
print(f"Total months analyzed: {len(shiller_full)}")
print(f"Number of significant drawdowns (>5%): {len(real_drawdowns)}")
```

```
5 LARGEST HISTORICAL DRAWDOWNS IN REAL RETURNS
======================================================================
(Adjusted for Inflation, Total Returns including Dividends)
======================================================================
1. February 1929 to February 1932 - Great Depression
   Peak: 108.33, Trough: 25.14
   Drawdown: -76.8%
   Duration to bottom: 36 months
   Full recovery: 85 months

2. February 2000 to January 2009 - Dot-com Bubble
   Peak: 10332.79, Trough: 4984.68
   Drawdown: -51.8%
   Duration to bottom: 108 months
   Full recovery: 158 months

3. January 1973 to February 1974 - 1973-74 Oil Crisis
   Peak: 1266.25, Trough: 632.37
   Drawdown: -50.1%
   Duration to bottom: 13 months
   Full recovery: 146 months

4. January 1937 to January 1942 - 1937-38 Recession
   Peak: 115.72, Trough: 59.81
   Drawdown: -48.3%
   Duration to bottom: 60 months
   Full recovery: 97 months

5. February 1916 to February 1920 - WWI and Post-War Recession
   Peak: 24.94, Trough: 13.19
   Drawdown: -47.1%
   Duration to bottom: 48 months
   Full recovery: 97 months

SUMMARY
======================================================================
Analysis period: 1871 to 2023
Total months analyzed: 1833
Number of significant drawdowns (>5%): 43
```

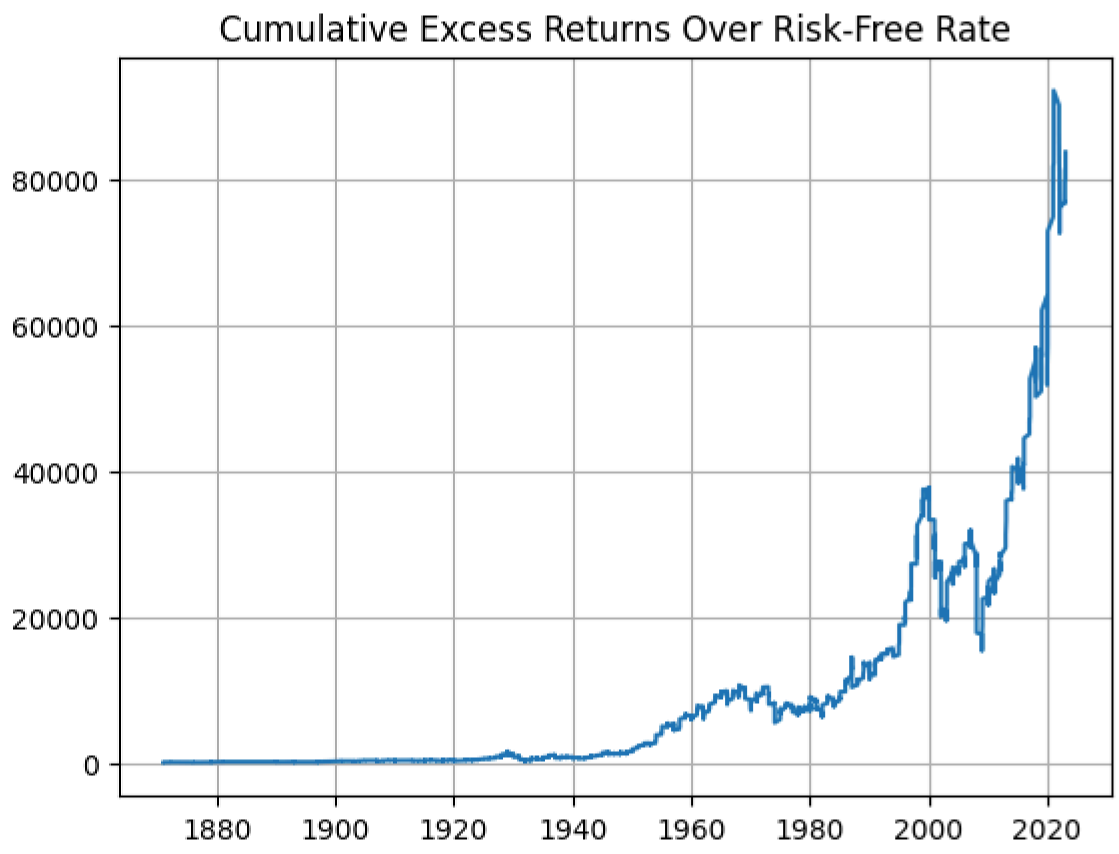
In [ ]:

H

In [211…

```
#Excess Returns over Risk-Free Rate
shiller_full['rf_monthly'] = (1 + shiller_full['Rate GS10']/100) ** (1/12) - 1
shiller_full['excess_return'] = shiller_full['Total_Return'] - shiller_full['rf_mon
shiller_full['cum_excess_return'] = (1 + shiller_full['excess_return']).cumprod()
plt.plot(shiller_full.index, shiller_full['cum_excess_return'] * 100)
```

```python
plt.title('Cumulative Excess Returns Over Risk-Free Rate')
plt.grid(True)
plt.show()

dates = shiller_full.index
values = shiller_full['cum_excess_return'].dropna().values
peak = 0
drawdowns = []

for i, value in enumerate(values):
    if value > peak:
        peak = value
    drawdown = (value - peak) / peak
    drawdowns.append((dates[i], drawdown))

# Get 5 largest drawdowns
largest_dd = sorted(drawdowns, key=lambda x: x[1])[:5]

print("5 LARGEST DRAWDOWNS (Excess Returns)")
for i, (trough_date, dd) in enumerate(largest_dd, 1):
    trough_idx = list(dates).index(trough_date)
    peak_val = max(values[:trough_idx+1])
    peak_idx = list(values[:trough_idx+1]).index(peak_val)
    peak_date = dates[peak_idx]

    # Find recovery
    recovery_idx = None
    for j in range(trough_idx, len(values)):
        if values[j] >= peak_val:
            recovery_idx = j
            break

    dd_months = (trough_date - peak_date).days // 30
    rec_months = (dates[recovery_idx] - peak_date).days // 30 if recovery_idx else

    print(f"{i}. {peak_date.strftime('%b %Y')} to {trough_date.strftime('%b %Y')}")
    print(f"   Drawdown: {dd*100:.1f}%")
    print(f"   Duration: {dd_months} months")
    print(f"   Recovery: {rec_months} months\n")
```



```
5 LARGEST DRAWDOWNS (Excess Returns)
1. Feb 1929 to Feb 1932
   Drawdown: -83.5%
   Duration: 36 months
   Recovery: 254 months

2. Feb 1929 to Feb 1932
   Drawdown: -82.5%
   Duration: 36 months
   Recovery: 254 months

3. Feb 1929 to Jan 1932
   Drawdown: -81.1%
   Duration: 35 months
   Recovery: 254 months

4. Feb 1929 to Jan 1932
   Drawdown: -78.6%
   Duration: 35 months
   Recovery: 254 months

5. Feb 1929 to Jan 1933
   Drawdown: -77.6%
   Duration: 47 months
   Recovery: 254 months
```

```
In [ ]:
```