Part 1: Data Acquisition & Preprocessing

Objective: Acquire and preprocess S&P 500 stock price data for portfolio analysis.

Key Tasks: 1. Scrape top 100 S&P 500 tickers by market capitalization 2. Download 5 years of End-of-Day OHLCV data 3. Clean and align data to uniform trading calendar 4. Compute daily log returns

Deliverables: - `prices`: Adjusted close price matrix (DataFrame) - `log_returns`: Daily log returns matrix (DataFrame) - Complete code workflow for data acquisition and preprocessing

```
# Import required libraries
import yfinance as yf
import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
from datetime import datetime, timedelta

warnings.filterwarnings('ignore')

print("Libraries imported successfully!")
```

```
Libraries imported successfully!
```

Task 1: Scrape Top 100 S&P 500 Tickers by Market Cap

```
# Get S&P 500 tickers from Wikipedia and select top 100 by market cap
def get_sp500_tickers():
    """Scrape S&P 500 tickers from Wikipedia"""
    url = 'https://en.wikipedia.org/wiki/List_of_S%26P_500_companies'
```

```
    tables = pd.read_html(url)
    sp500_table = tables[0]
    tickers = sp500_table['Symbol'].str.replace('.', '-').tolist()
    return tickers

# Get all S&P 500 tickers
sp500_tickers = get_sp500_tickers()
print(f"Found {len(sp500_tickers)} S&P 500 tickers")

# Get market caps to select top 100
print("Getting market capitalizations...")
market_caps = {}

# Process efficiently - check first 150 tickers
for ticker in sp500_tickers[:150]:
    try:
        stock = yf.Ticker(ticker)
        info = stock.info
        market_cap = info.get('marketCap', 0)
        if market_cap > 0:
            market_caps[ticker] = market_cap
    except:
        continue

# Sort by market cap and get top 100
sorted_tickers = sorted(market_caps.items(), key=lambda x: x[1], reverse=True)
top_100_tickers = [ticker for ticker, _ in sorted_tickers[:100]]

print(f"  Selected top 100 tickers by market cap")
print(f"Sample tickers: {top_100_tickers[:5]}..." +
      f"{top_100_tickers[-5:]}")
print(f"Total: {len(top_100_tickers)} tickers")
```

```
Found 503 S&P 500 tickers
Getting market capitalizations...
  Selected top 100 tickers by market cap
Sample tickers: ['AAPL', 'GOOGL', 'GOOG', 'AMZN', 'AVGO']...['DXCM', 'AWK', 'ADM', 'AEE', 'A
Total: 100 tickers
  Selected top 100 tickers by market cap
Sample tickers: ['AAPL', 'GOOGL', 'GOOG', 'AMZN', 'AVGO']...['DXCM', 'AWK', 'ADM', 'AEE', 'A
Total: 100 tickers
```

Task 2: Download EOD OHLCV Data (5 Years)

```
# Download 5 years of EOD OHLCV data

# Define date range
end_date = "2025-08-07"  # Use today's date
start_date = "2020-08-07"

print(f"Downloading data from {start_date} to " +
      f"{end_date} for {len(top_100_tickers)} tickers...")
# Download data for all tickers
raw_data = yf.download(
    tickers=top_100_tickers,
    start=start_date,
    end=end_date,
    group_by='ticker',
    auto_adjust=True,
    prepost=False,
    threads=True
)

print(f"  Download completed!")
print(f"Data shape: {raw_data.shape}")
print(f"Date range: {raw_data.index[0]} to {raw_data.index[-1]}")

# Display first 5 rows of raw data
print("Displaying first 5 rows of raw data:")
print(raw_data.head(5))
```

```
Downloading data from 2020-08-07 to 2025-08-07 for 100 tickers...

[**********************100%**********************]  100 of 100 completed


  Download completed!
Data shape: (1255, 500)
Date range: 2020-08-07 00:00:00 to 2025-08-06 00:00:00
Displaying first 5 rows of raw data:
Ticker             DLR                                                  \
Price             Open        High         Low       Close    Volume
Date
```

```
2020-08-07  131.495502  133.595399  131.218315  133.225815   971700
2020-08-10  133.209032  133.645814  130.672366  132.915054  1520400
2020-08-11  133.167019  133.167019  127.211716  127.362915  1422000
2020-08-12  127.606506  130.084382  127.253725  129.723206   953000
2020-08-13  129.429230  130.269189  127.707312  127.791313   968800

Ticker             BLK                                              ...  \
Price             Open        High         Low       Close  Volume  ...
Date                                                                ...
2020-08-07  516.052317  521.074095  513.173895  520.941223  469900  ...
2020-08-10  519.187621  520.028963  513.395316  513.864746  465800  ...
2020-08-11  518.381662  528.310024  516.636921  519.568481  576800  ...
2020-08-12  524.439566  536.466901  517.584503  522.756775  612200  ...
2020-08-13  519.639280  527.238329  519.639280  522.154602  359700  ...

Ticker             APH                                                GOOGL  \
Price             Open       High        Low      Close   Volume       Open
Date
2020-08-07  25.742188  25.854246  25.544299  25.739803  4357600  75.002191
2020-08-10  25.675430  25.973454  25.656356  25.777950  9071600  74.095621
2020-08-11  25.918617  26.226179  25.882854  25.942459  6656000  74.254662
2020-08-12  26.049747  26.316776  25.854243  26.176109  5422400  73.912728
2020-08-13  26.071203  26.559965  25.942456  26.362076  4139600  74.960936

Ticker
Price             High        Low      Close    Volume
Date
2020-08-07  75.551395  73.870474  74.471870  27718000
2020-08-10  74.908244  73.434084  74.394829  20546000
2020-08-11  75.071768  73.468878  73.585678  31098000
2020-08-12  75.132911  73.807357  74.912727  22512000
2020-08-13  76.390366  74.960936  75.380417  22388000

[5 rows x 500 columns]
```

Task 3: Clean and Align Data to Uniform Calendar

```
# Extract and clean adjusted close prices
print("Extracting and cleaning adjusted close prices...")
```

```
# Extract close prices for all tickers
if len(top_100_tickers) == 1:
    # Single ticker case
    prices = raw_data[['Close']].copy()
    prices.columns = top_100_tickers
else:
    # Multiple tickers case
    prices = pd.DataFrame(index=raw_data.index)
    for ticker in top_100_tickers:
        if ticker in raw_data.columns.get_level_values(0):
            prices[ticker] = raw_data[ticker]['Close']

# Clean the data
print("Cleaning data...")
# Remove columns with all NaN values
prices = prices.dropna(axis=1, how='all')
# Remove rows with all NaN values
prices = prices.dropna(how='all')
# Forward fill missing values
prices = prices.fillna(method='ffill')
# Backward fill remaining NaN values
prices = prices.fillna(method='bfill')

final_tickers = prices.columns.tolist()
print(f"  Clean price data: {prices.shape[0]} dates × " +
      f"{prices.shape[1]} tickers")
print(f"Date range: {prices.index[0]} to {prices.index[-1]}")
print(f"Final tickers count: {len(final_tickers)}")

# Display uniformly formatted output
print("Displaying first 5 rows of the cleaned price data:")
print(prices.head(5))

# Plot the first 5 tickers to visualize the data
print("Plotting first 5 tickers...")
prices.iloc[:, :5].plot(figsize=(14, 7))
plt.title('Adjusted Close Prices of Top 5 S&P 500 Tickers')
plt.xlabel('Date')
plt.ylabel('Adjusted Close Price')
plt.legend(loc='upper left')
plt.grid()
plt.show()
```

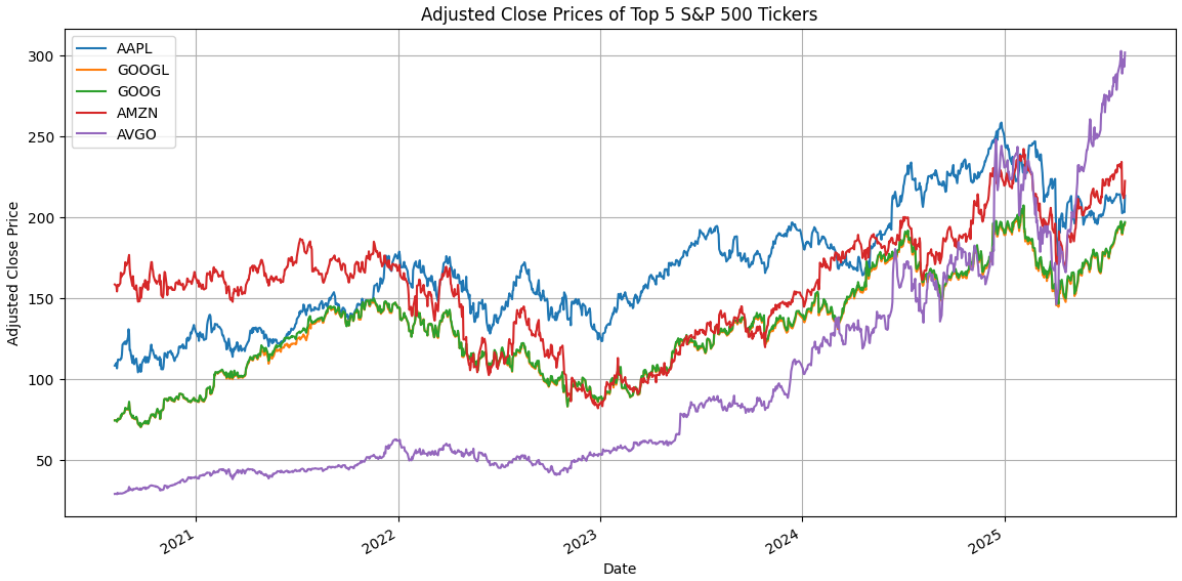
```
Extracting and cleaning adjusted close prices...
Cleaning data...
  Clean price data: 1255 dates × 100 tickers
Date range: 2020-08-07 00:00:00 to 2025-08-06 00:00:00
Final tickers count: 100
Displaying first 5 rows of the cleaned price data:
                 AAPL      GOOGL       GOOG        AMZN       AVGO  \
Date
2020-08-07  108.203766  74.471870  74.282959  158.373001  28.915096
2020-08-10  109.776474  74.394829  74.362984  157.408005  29.041965
2020-08-11  106.511765  73.585678  73.578629  154.033493  28.746536
2020-08-12  110.051575  74.912727  74.885880  158.112000  29.599096
2020-08-13  111.999237  75.380417  75.473877  158.050995  29.224718

                BRK-B        COST       ABBV        BAC        CVX  ...  \
Date                                                                 ...
2020-08-07  209.479996  314.168152  75.978493  23.083563  69.710854  ...
2020-08-10  212.580002  313.329559  75.536949  23.481401  72.064011  ...
2020-08-11  212.660004  306.353302  75.774048  23.799673  71.975662  ...
2020-08-12  213.240005  310.343689  78.096268  23.631695  72.859093  ...
2020-08-13  211.979996  309.366791  77.417610  23.295744  72.136284  ...

                ACGL          A          BR         STZ        BRO  \
Date
2020-08-07  30.894695  94.532814  124.429207  158.571686  44.604271
2020-08-10  31.122908  93.914070  125.671204  158.506989  44.478382
2020-08-11  31.237015  93.101967  130.740372  159.819763  44.216919
2020-08-12  30.999290  94.387810  127.989571  162.251190  44.352482
2020-08-13  30.771076  95.489937  127.731972  163.286652  44.371861

                DXCM         AWK        ADM        AEE         AVB
Date
2020-08-07  110.175003  136.478470  38.364445  71.071014  129.755478
2020-08-10  105.305000  135.858902  38.765259  71.996376  129.704926
2020-08-11  102.787498  130.243851  38.678120  70.188889  127.967628
2020-08-12  106.972504  133.398849  38.730412  70.898048  128.954315
2020-08-13  108.820000  133.984116  38.712982  70.370499  127.090546

[5 rows x 100 columns]
Plotting first 5 tickers...
```



Task 4: Compute Daily Log Returns

Formula: r_t = ln(P_t / P_{t-1}) = ln(P_t) - ln(P_{t-1})

```
# Compute daily log returns: r_t = ln(P_t / P_{t-1})
print("Computing daily log returns...")

log_returns = np.log(prices / prices.shift(1))
log_returns = log_returns.dropna()

# Clean any infinite or NaN values
log_returns.replace([np.inf, -np.inf], np.nan, inplace=True)
log_returns.fillna(method='ffill', inplace=True)
log_returns.fillna(0, inplace=True)

print(f"  Log returns computed successfully!")
print(f"Shape: {log_returns.shape}")
print(f"Date range: {log_returns.index[0]} to {log_returns.index[-1]}")

# Basic statistics
print(f"\nSummary statistics:")
print(f"Mean daily return: {log_returns.mean().mean():.6f}")
print(f"Std daily return: {log_returns.std().mean():.6f}")
print(f"Data quality - NaN values: {log_returns.isnull().sum().sum()}")
```

```python
print(f"Data quality - Infinite values: {np.isinf(log_returns).sum().sum()}")

# Display first 5 rows of log returns
print("Displaying first 5 rows of log returns:")
print(log_returns.head(5))

# Plot log returns: cleaner and more informative visualization
import matplotlib.dates as mdates

plt.figure(figsize=(16, 7))
# Plot only a subset of tickers for clarity (e.g., top 10 by market cap)
top10 = [ticker for ticker in top_100_tickers[:10] if ticker in log_returns.columns]
log_returns[top10].plot(ax=plt.gca(), alpha=0.7, linewidth=1.2)

plt.title('Daily Log Returns of Top 10 S&P 500 Stocks by Market Cap',
          fontsize=18, fontweight='bold')
plt.xlabel('Date', fontsize=14)
plt.ylabel('Log Return', fontsize=14)
plt.legend(top10, loc='upper right', ncol=2, fontsize=10,
           frameon=True)
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.xlim(log_returns.index[0], log_returns.index[-1])
plt.gca().xaxis.set_major_locator(mdates.YearLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.show()
```

```
Computing daily log returns...
 Log returns computed successfully!
Shape: (1254, 100)
Date range: 2020-08-10 00:00:00 to 2025-08-06 00:00:00

Summary statistics:
Mean daily return: 0.000567
Std daily return: 0.019338
Data quality - NaN values: 0
Data quality - Infinite values: 0
Displaying first 5 rows of log returns:
                AAPL     GOOGL      GOOG      AMZN      AVGO     BRK-B  \
Date
2020-08-10  0.014430 -0.001035  0.001077 -0.006112  0.004378  0.014690
2020-08-11 -0.030191 -0.010936 -0.010604 -0.021671 -0.010225  0.000376
```

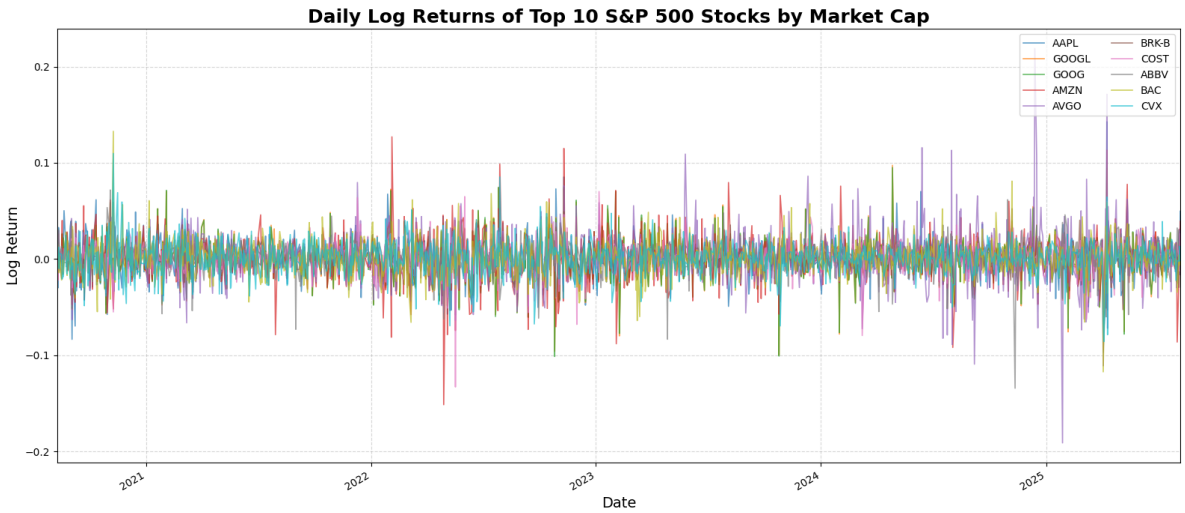
```
2020-08-12  0.032694  0.017873  0.017611  0.026134  0.029227  0.002724
2020-08-13  0.017543  0.006224  0.007821 -0.000386 -0.012729 -0.005926
2020-08-14 -0.000892 -0.007957 -0.007085 -0.004121 -0.004869 -0.004823

                COST      ABBV       BAC       CVX  ...      ACGL         A  \
Date                                                  ...
2020-08-10 -0.002673 -0.005828  0.017088  0.033199  ...  0.007360 -0.006567
2020-08-11 -0.022517  0.003134  0.013463 -0.001227  ...  0.003660 -0.008685
2020-08-12  0.012941  0.030186 -0.007083  0.012199  ... -0.007639  0.013717
2020-08-13 -0.003153 -0.008728 -0.014318 -0.009970  ... -0.007389  0.011609
2020-08-14  0.001726  0.004110  0.004544  0.005884  ... -0.003405 -0.014276

                 BR       STZ       BRO      DXCM       AWK       ADM  \
Date
2020-08-10  0.009932 -0.000408 -0.002826 -0.045209 -0.004550  0.010393
2020-08-11  0.039544  0.008248 -0.005896 -0.024197 -0.042208 -0.002250
2020-08-12 -0.021265  0.015099  0.003061  0.039908  0.023935  0.001351
2020-08-13 -0.002015  0.006362  0.000437  0.017123  0.004378 -0.000450
2020-08-14  0.000288 -0.003346  0.000000 -0.023524 -0.008224  0.003595

                AEE       AVB
Date
2020-08-10  0.012936 -0.000390
2020-08-11 -0.025426 -0.013485
2020-08-12  0.010053  0.007681
2020-08-13 -0.007469 -0.014558
2020-08-14 -0.001230  0.015474

[5 rows x 100 columns]
```



Deliverables

```
# DELIVERABLE 1: prices - Adjusted close price matrix (DataFrame)
print("DELIVERABLE 1: prices")
print("=" * 40)
print(f"Type: {type(prices)}")
print(f"Shape: {prices.shape}")
print(f"Index: {prices.index[0]} to {prices.index[-1]}")
print(f"Columns: {len(prices.columns)} tickers")
print("\nSample data (first 5 rows, first 5 columns):")
display(prices.iloc[:5, :5])

print("\n" + "=" * 40)

# DELIVERABLE 2: log_returns - Daily log returns matrix (DataFrame)
print("DELIVERABLE 2: log_returns")
print("=" * 40)
print(f"Type: {type(log_returns)}")
print(f"Shape: {log_returns.shape}")
print(f"Index: {log_returns.index[0]} to {log_returns.index[-1]}")
print(f"Columns: {len(log_returns.columns)} tickers")
print(f"Formula: r_t = ln(P_t / P_{{t-1}})")
print("\nSample data (first 5 rows, first 5 columns):")
display(log_returns.iloc[:5, :5])
```

```
print("\n" + "=" * 40)

# DELIVERABLE 3: This notebook - Data preprocessing script
print("DELIVERABLE 3: This notebook")
print("=" * 40)
print("Type: Jupyter Notebook (.ipynb)")
print("Contains all code and comments for data acquisition " +
      "and preprocessing")
```

```
DELIVERABLE 1: prices
========================================
Type: <class 'pandas.core.frame.DataFrame'>
Shape: (1255, 100)
Index: 2020-08-07 00:00:00 to 2025-08-06 00:00:00
Columns: 100 tickers

Sample data (first 5 rows, first 5 columns):
```

Date	AAPL	GOOGL	GOOG	AMZN	AVGO
2020-08-07	108.203766	74.471870	74.282959	158.373001	28.915096
2020-08-10	109.776474	74.394829	74.362984	157.408005	29.041965
2020-08-11	106.511765	73.585678	73.578629	154.033493	28.746536
2020-08-12	110.051575	74.912727	74.885880	158.112000	29.599096
2020-08-13	111.999237	75.380417	75.473877	158.050995	29.224718

```
========================================
DELIVERABLE 2: log_returns
========================================
Type: <class 'pandas.core.frame.DataFrame'>
Shape: (1254, 100)
Index: 2020-08-10 00:00:00 to 2025-08-06 00:00:00
Columns: 100 tickers
Formula: r_t = ln(P_t / P_{t-1})

Sample data (first 5 rows, first 5 columns):
```

Date	AAPL	GOOGL	GOOG	AMZN	AVGO
2020-08-10	0.014430	-0.001035	0.001077	-0.006112	0.004378
2020-08-11	-0.030191	-0.010936	-0.010604	-0.021671	-0.010225
2020-08-12	0.032694	0.017873	0.017611	0.026134	0.029227
2020-08-13	0.017543	0.006224	0.007821	-0.000386	-0.012729
2020-08-14	-0.000892	-0.007957	-0.007085	-0.004121	-0.004869

```
========================================
DELIVERABLE 3: This notebook
========================================
Type: Jupyter Notebook (.ipynb)
Contains all code and comments for data acquisition and preprocessing
```