Stock Market Time Series Analysis with Pandas

Notebook 08: Multi-Stock Comparison and Correlation Analysis

Python 3.8+ Pandas Latest NumPy Latest License MIT

Part of the comprehensive learning series: Stock Market Time Series Analysis with Pandas

Learning Objectives:

- Fetch and manage multi-stock datasets efficiently
- Perform price normalization for fair performance comparison
- Calculate and visualize correlation matrices
- Understand portfolio diversification through correlation analysis
- Master multi-asset time series analysis techniques

- Real-world finance rarely involves just one stock.
- Portfolio management requires comparing the performance and risk of multiple assets simultaneously.
- This demands special techniques to make an 'apples-to-apples' comparison.

This notebook focuses on:

1. **Multi-Stock Data Fetching:** Downloading multiple tickers into a single, combined DataFrame.
2. **Normalization:** Adjusting prices so they all start at the same base value (e.g., 100) to compare growth rates.
3. **Performance Comparison:** Visualizing normalized cumulative returns.
4. **Correlation Analysis:** Calculating and visualizing the relationship between the daily returns of different stocks using a correlation heatmap.

In [1]:
```
# Importing necessary libraries
import pandas as pd
import numpy as np
import yfinance as yf
import matplotlib.pyplot as plt
import seaborn as sns

# Setting plot style
sns.set_style('whitegrid')
```

In [2]:
```python
# Suppressing future warnings for cleaner output
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [4]:
```python
# --- Concept: Multiple Ticker Fetching ---
# yfinance can download multiple tickers at once, returning a DataFrame with MultiInc
TICKERS = ['AAPL', 'MSFT', 'GOOGL', 'AMZN', 'TSLA']
START_DATE = '2019-01-01'
END_DATE = '2024-01-01'

# --- Code: Fetch Multiple Stocks ---
data = yf.download(TICKERS, start=START_DATE, end=END_DATE)

print("Shape of the multi-stock DataFrame:", data.shape)
```

```
[*********************100%***********************]  5 of 5 completed
Shape of the multi-stock DataFrame: (1258, 25)
```

In [5]:
```python
print("Column structure (MultiIndex):")
print(data.columns.names)
```

```
Column structure (MultiIndex):
['Price', 'Ticker']
```

In [6]:
```python
# Displaying the first few rows of the DataFrame
data.head()
```

Out[6]:

Price					Close			
Ticker	AAPL	AMZN	GOOGL	MSFT	TSLA	AAPL	AMZN	GOC
Date								
2019-01-02	37.575214	76.956497	52.372784	94.789703	20.674667	37.796499	77.667999	52.676
2019-01-03	33.832439	75.014000	50.922283	91.302567	20.024000	34.672361	76.900002	52.947
2019-01-04	35.276711	78.769501	53.534267	95.548988	21.179333	35.345715	79.699997	53.630
2019-01-07	35.198208	81.475502	53.427509	95.670837	22.330667	35.412354	81.727997	53.764
2019-01-08	35.869194	82.829002	53.896774	96.364510	22.356667	36.123790	83.830498	54.293

5 rows × 25 columns

1. Preparing the Data (Closing Prices)

- To compare stocks, we must isolate the necessary price column (`Close`) for all tickers.
- MultiIndex slicing is essential here.

In [7]:
```python
# --- Concept: MultiIndex Slicing with pd.IndexSlice ---
# We use pd.IndexSlice to cleanly select the 'Close' prices across all tickers.
idx = pd.IndexSlice
```

```python
close_prices = data.loc[:, idx['Close', :]] # Select 'Close' prices for all tickers

# Flatten the columns by dropping the redundant 'Close' level
close_prices.columns = close_prices.columns.get_level_values(1)

print("Cleaned Close Prices DataFrame (first 5 rows):")
close_prices.head()
```

```
Cleaned Close Prices DataFrame (first 5 rows):
```

Out[7]:

Ticker	AAPL	AMZN	GOOGL	MSFT	TSLA
Date					
2019-01-02	37.575214	76.956497	52.372784	94.789703	20.674667
2019-01-03	33.832439	75.014000	50.922283	91.302567	20.024000
2019-01-04	35.276711	78.769501	53.534267	95.548988	21.179333
2019-01-07	35.198208	81.475502	53.427509	95.670837	22.330667
2019-01-08	35.869194	82.829002	53.896774	96.364510	22.356667

2. Normalization for Performance Comparison

- Normalization aligns the starting price of all stocks to a single point (e.g., 100), regardless of their absolute dollar value.
- This allows us to compare their *rate of growth*.

In [8]:

```python
# --- Concept: Normalization Formula ---
# Normalized Price = (Current Price / First Price) * 100
# Pandas' division and element-wise operations make this simple.

# --- Code: Calculate Normalized Prices ---
# Use .iloc[0] to get the prices on the very first day
normalized_prices = close_prices.div(close_prices.iloc[0]).mul(100)

print("Normalized Prices (showing the start):")
normalized_prices.head(2)
```

```
Normalized Prices (showing the start):
```

Out[8]:

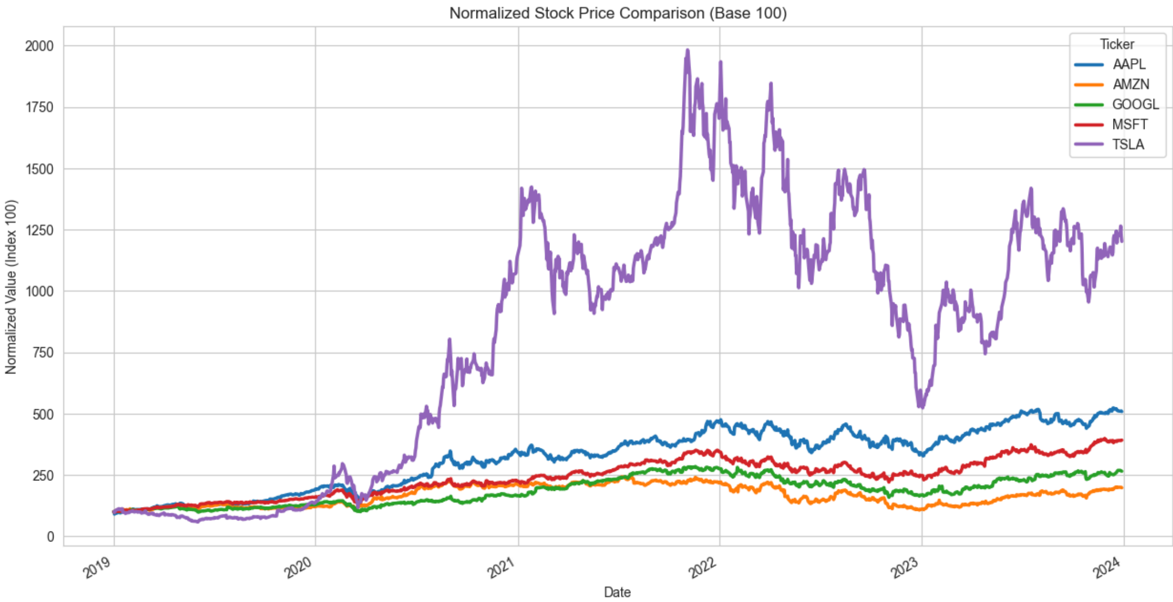
Ticker	AAPL	AMZN	GOOGL	MSFT	TSLA
Date					
2019-01-02	100.000000	100.00000	100.000000	100.000000	100.000000
2019-01-03	90.039245	97.47585	97.230431	96.321186	96.852829

In [ ]:

```python
# --- Code: Plot Normalized Prices ---
plt.figure(figsize=(15, 8))

# Plotting the normalized prices
# ax = plt.gca() gets the current axes, allowing further customization if needed
normalized_prices.plot(
    title='Normalized Stock Price Comparison (Base 100)',
    linewidth=2.5,
    ax=plt.gca()
)
```

```python
plt.xlabel('Date')
plt.ylabel('Normalized Value (Index 100)')
plt.legend(title='Ticker')
plt.show()
```



Visualization 1 Insights

1. **Relative Performance:** The plot immediately shows which stock was the best and worst performer over the period. The line that finishes highest represents the largest percentage gain from the start date.

2. **Risk Exposure:** Stocks with more dramatic peaks and valleys (like TSLA) show higher volatility, while smoother lines (like maybe MSFT) might indicate more stable growth.

3. **Market Alignment:** Notice how all lines tend to move together during major market events (e.g., the 2020 drop), illustrating high market correlation.

3. Correlation Analysis (Daily Returns)

- **Correlation** measures how much the price movements of two assets are related.
- Low correlation is desirable in a portfolio for diversification, as assets don't all crash at the same time.
- We calculate correlation using the **Daily Returns**, not the raw prices.

In [10]:
```python
# --- Concept: Calculating Daily Returns for Multiple Stocks ---
# .pct_change() can be applied directly to the DataFrame, calculating returns column-
daily_returns = close_prices.pct_change().dropna()

# --- Code: Calculate Correlation Matrix ---
correlation_matrix = daily_returns.corr()

print("Correlation Matrix of Daily Returns:")
correlation_matrix
```

```
Correlation Matrix of Daily Returns:
```

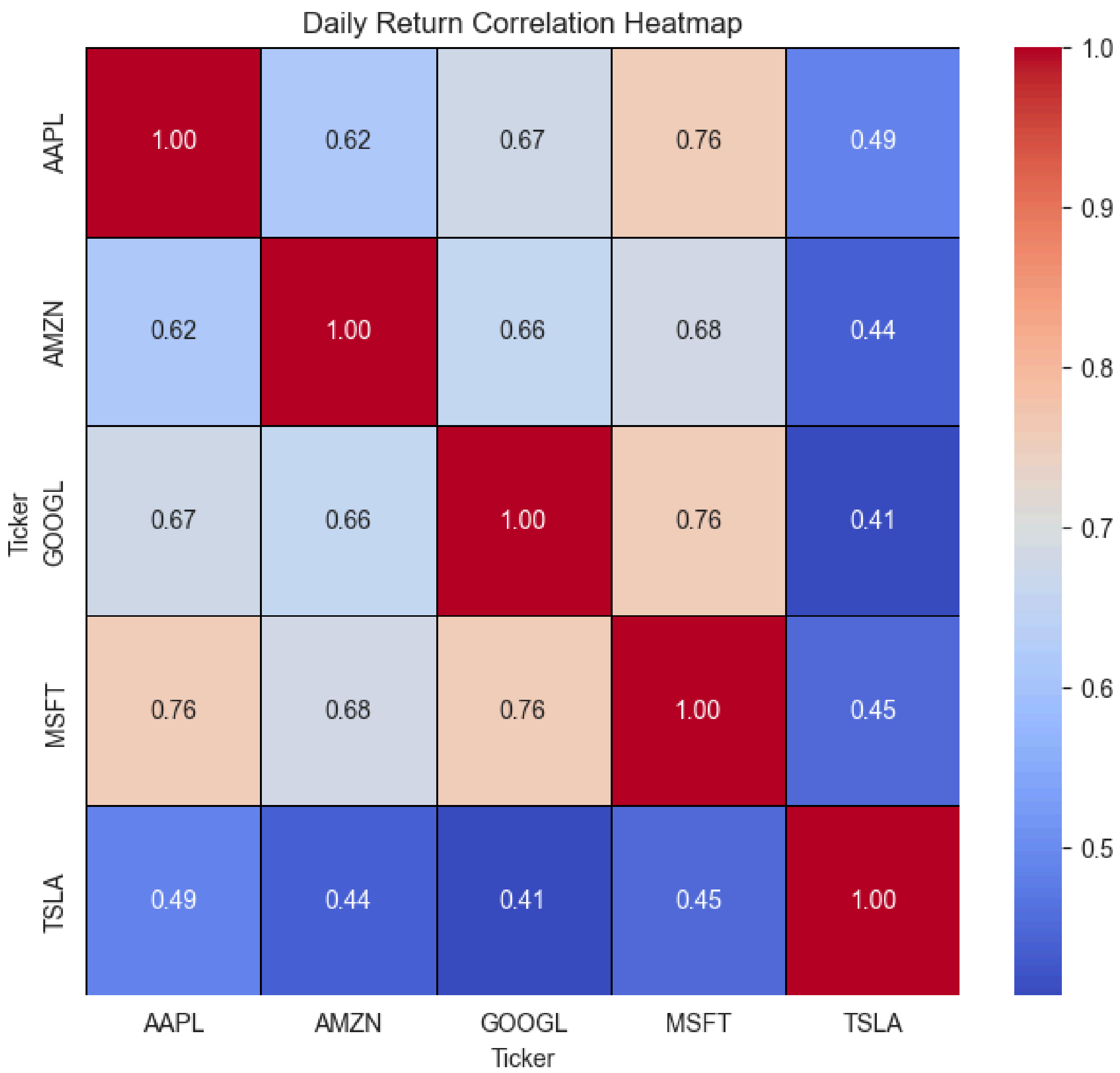
Out[10]:

Ticker	AAPL	AMZN	GOOGL	MSFT	TSLA
Ticker					
AAPL	1.000000	0.619210	0.674939	0.758621	0.488406
AMZN	0.619210	1.000000	0.659608	0.682969	0.438261
GOOGL	0.674939	0.659608	1.000000	0.759000	0.408303
MSFT	0.758621	0.682969	0.759000	1.000000	0.453118
TSLA	0.488406	0.438261	0.408303	0.453118	1.000000

4. Visualization: Correlation Heatmap

- A heatmap is the clearest way to visualize the correlation matrix.

In [11]:
```
# --- Code: Plot Correlation Heatmap ---
plt.figure(figsize=(8, 7))
sns.heatmap(
    correlation_matrix,
    annot=True,         # Show the correlation value on the plot
    cmap='coolwarm',    # Use a diverging color map
    fmt=".2f",          # Format to two decimal places
    linewidths=0.5,     # Lines between cells
    linecolor='black'
)
plt.title('Daily Return Correlation Heatmap')
plt.show()
```



Visualization 2 Insights

1. **Interpretation:** The matrix is symmetric and the diagonal is always 1.0 (a stock is perfectly correlated with itself).

2. **High Correlation:** Most large-cap technology stocks will show a high positive correlation (typically 0.60 to 0.80). This means when AAPL has a positive return, MSFT is highly likely to have one too.

3. **Diversification Implication:** Since all these stocks move strongly together, adding another highly correlated stock offers little **diversification benefit**. For true portfolio risk reduction, one would need to add assets with low or even negative correlation (e.g., gold, long-term bonds, or commodities).

5. Summary and Next Steps

Key Takeaways

- **Multi-Series Handling:** We successfully fetched and cleaned multi-ticker data, demonstrating Pandas' ability to manage complex column structures.

- **Normalization:** We performed **price normalization** (Base 100), enabling a fair comparison of growth rates across stocks with different initial prices.
- **Correlation:** We calculated and visualized the **correlation matrix** of daily returns, confirming that these technology stocks are highly related, which is an important insight for portfolio risk.

Next Notebook Preview

- We have relied on basic averages (SMA) so far.
- The next notebook will focus on implementing full, industry-standard **Technical Indicators** like Bollinger Bands and RSI, calculated entirely using Pandas functions like `.rolling()` and the `.ewm()` method for Exponential Moving Averages.

About This Project

This notebook is part of the **Stock Market Time Series Analysis with Pandas** repository - a comprehensive, beginner-to-intermediate friendly guide for mastering financial time series analysis using Python and Pandas.

Repository: `stock-time-series-analysis-with-pandas`

Author

Prakash Ukhalkar
GitHub prakash-ukhalkar

Built with ❤️ for the Python community