Stock Market Time Series Analysis with Pandas

Notebook 04: Calculating and Analyzing Returns and Volatility

Python 3.8+ | Pandas Latest | NumPy Latest | License MIT

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

Learning Objectives:

- Calculate daily percent returns using `.pct_change()`
- Master log returns for statistical modeling
- Compute cumulative returns for performance analysis
- Analyze return distributions with statistical measures
- Understand volatility, skewness, and kurtosis in financial data

---

- Raw price plots only tell half the story.
- To properly assess a stock's performance and risk, we must look at its **returns** — the percentage change in value over a period.
- Returns are the core measurement in finance.

In this notebook, we master the calculation and interpretation of returns using Pandas:

1. **Daily Percent Returns:** Calculating simple period-over-period changes using `.pct_change()`.
2. **Log Returns:** Calculating continuously compounded returns using `numpy.log()`.
3. **Cumulative Returns:** Measuring total investment performance over time.
4. **Statistical Analysis:** Visualizing the distribution of returns (volatility, skewness, kurtosis).

In [1]:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import yfinance as yf
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew, kurtosis

sns.set_style('whitegrid')
```

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

In [4]:
```python
# Reloading the data
TICKER = 'AAPL'
START_DATE = '2019-01-01'
END_DATE = '2025-01-01'

df = yf.download(TICKER, start=START_DATE, end=END_DATE)

# Clean up MultiIndex columns if present
if isinstance(df.columns, pd.MultiIndex):
    df.columns = df.columns.get_level_values(0)

print("\nInitial DataFrame head:")
df.head()
```

```
[*********************100%***********************]  1 of 1 completed
Initial DataFrame head:
```

Out[4]:

Price	Close	High	Low	Open	Volume
Date					
2019-01-02	37.575207	37.796491	36.697214	36.854254	148158800
2019-01-03	33.832436	34.672357	33.787227	34.258343	365248800
2019-01-04	35.276718	35.345722	34.215516	34.389210	234428400
2019-01-07	35.198204	35.412351	34.715190	35.381418	219111200
2019-01-08	35.869183	36.123778	35.338581	35.586036	164101200

1. Daily Percent Returns (Simple Returns)

- **Simple Returns** (or arithmetic returns) measure the profit or loss relative to the previous day's closing price.
- This is the most intuitive and commonly quoted return.

In [6]:
```python
# --- Concept: Pandas .pct_change() ---
# The .pct_change() method calculates the percentage difference between
# the current element and a prior element (default is the previous day).
df['Daily_Return'] = df['Close'].pct_change()

# Display the first few rows to verify
print("Daily Returns (first few rows):")

df[['Close', 'Daily_Return']].head()
```

```
Daily Returns (first few rows):
```

Out[6]:

Price	Close	Daily_Return
Date		
2019-01-02	37.575207	NaN
2019-01-03	33.832436	-0.099607
2019-01-04	35.276718	0.042689
2019-01-07	35.198204	-0.002226
2019-01-08	35.869183	0.019063

Insights on `Daily_Return`

1. **First NaN:** The first row of the `Daily_Return` column is always **NaN** because there is no previous day to calculate the change from.
2. **Volatility:** Unlike raw price, daily returns hover around zero. The magnitude of these values (e.g., 0.02 or -0.05) represents the **daily volatility** of the stock.

2. Log Returns (Continuously Compounded)

- **Log Returns** (or continuously compounded returns) are preferred in academic and quantitative finance for statistical modeling because they are additive over time and possess more desirable statistical properties (closer to a Normal distribution).
- The formula used is:

$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right)$$

In [7]:

```
# --- Concept: Calculating Log Returns ---
# Log Return formula: $r_t = ln(P_t / P_{t-1})$.
# We use NumPy's log() and Pandas' .shift(1) to get the previous price.

df['Log_Return'] = np.log(df['Close'] / df['Close'].shift(1))

# Display the first few rows to verify
print("Log Returns (first few rows):")
df[['Close', 'Daily_Return', 'Log_Return']].head()
```

```
Log Returns (first few rows):
```

Out[7]:

Price	Close	Daily_Return	Log_Return
Date			
2019-01-02	37.575207	NaN	NaN
2019-01-03	33.832436	-0.099607	-0.104924
2019-01-04	35.276718	0.042689	0.041803
2019-01-07	35.198204	-0.002226	-0.002228
2019-01-08	35.869183	0.019063	0.018883

Log vs. Simple Returns

- For small daily price changes, the Log Return and Simple Return values are very similar (e.g., 0.01 vs. 0.0099).
- They diverge significantly only for very large price changes. For most quantitative work, **Log Returns** are the standard.

3. Cumulative Returns (Total Performance)

- **Cumulative Returns** show the growth of a hypothetical $1 investment over the entire period, making it the best measure of overall portfolio performance.

In [ ]:

```
# --- Concept: Calculating Cumulative Returns ---
# Cumulative return is calculated by taking the cumulative product of (1 + Daily S
# Pandas' .cumprod() is ideal for this.

# --- Code: Calculate Cumulative Returns ---
# Note: We fill the initial NaN with 0 before calculation to start the product at
df['Cumulative_Return'] = (1 + df['Daily_Return'].fillna(0)).cumprod()

print("Total performance (Value of $1 investment):")
print(f"Initial Value: $1.00")
print(f"Final Value: ${df['Cumulative_Return'].iloc[-1]:.2f}")
```

```
Total performance (Value of $1 investment):
Initial Value: $1.00
Final Value: $6.64
```

In [9]:
```
# --- Code: Plot Cumulative Returns ---
# This plot shows how a $1 investment would have grown over the period.
plt.figure(figsize=(14, 6))
df['Cumulative_Return'].plot(
    title=f'{TICKER} Cumulative Return ($1 Investment Growth)',
    color='purple',
    linewidth=2
)
plt.xlabel('Date')
plt.ylabel('Growth Factor (Base 1.0)')
# Adding a horizontal line at y=1 for reference
plt.axhline(1.0, color='red', linestyle='--', alpha=0.6, label='Break-even (1.0)')
plt.legend()
plt.show()
```

Visualization 1 Insights

1. **Performance Visualization:** This chart directly shows the *multiplicative* growth of the investment. A value of 2.5 means a 150% return.
2. **Drawdowns:** The steepness of the drops (like in 2020) and the speed of the recovery are clearly visible, indicating periods of high risk and subsequent reward.
3. **Positive Skew:** The curve shows periods of rapid exponential growth, confirming the long-term bullish trend.

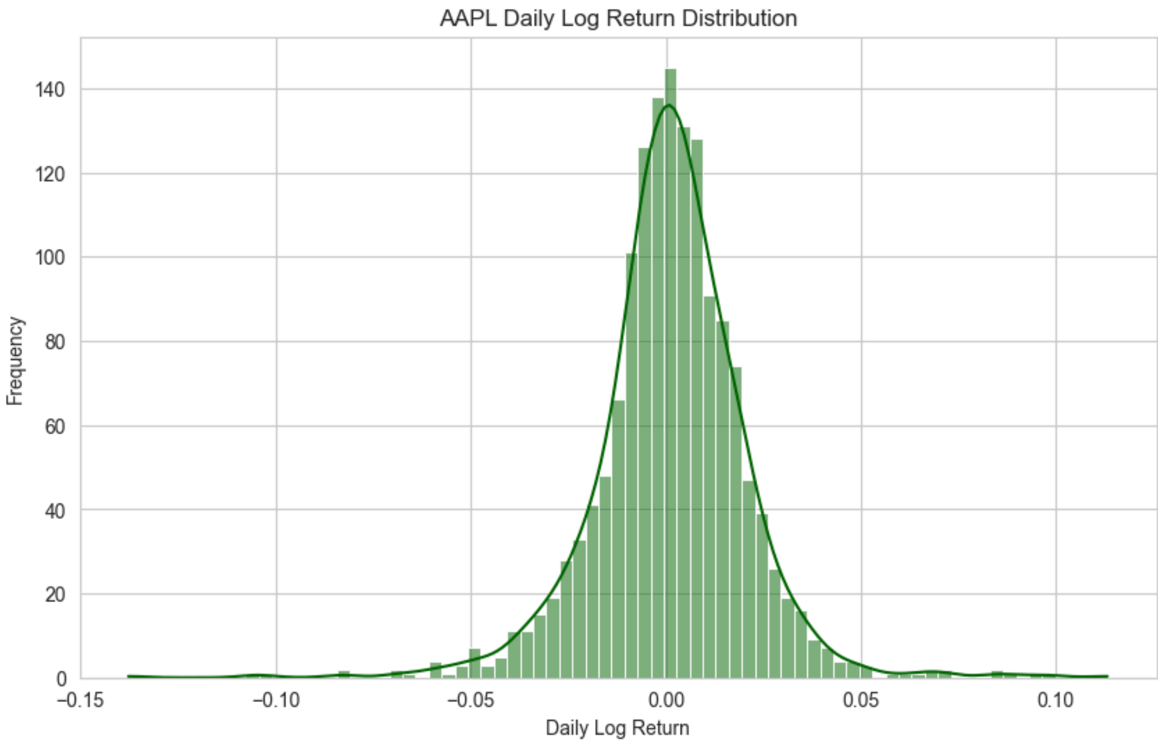
4. Analyzing the Distribution of Returns

- The distribution of daily returns is the key to understanding a stock's risk profile (volatility).
- We use a histogram and statistical measures (Skewness and Kurtosis) for this.

In [10]:
```
# --- Code: Histogram of Log Returns ---
# This histogram visualizes the distribution of daily log returns.
plt.figure(figsize=(10, 6))

# We use Log Returns for a statistically cleaner distribution
sns.histplot(df['Log_Return'].dropna(), bins=75, kde=True, color='darkgreen')
```

```python
plt.title(f'{TICKER} Daily Log Return Distribution')
plt.xlabel('Daily Log Return')
plt.ylabel('Frequency')
plt.show()
```



In [11]:
```python
# --- Code: Statistical Measures ---
# Calculating key statistical properties of the Log Returns
# We drop NaN values for accurate statistics
returns_series = df['Log_Return'].dropna()

print("\n--- Statistical Properties of Log Returns ---")
print(f"Mean (Average Daily Return): {returns_series.mean():.6f}")
print(f"Standard Deviation (Volatility): {returns_series.std():.4f}")

# Skew and Kurtosis are calculated using scipy.stats
print(f"Skewness: {skew(returns_series):.4f}")
print(f"Kurtosis (Excess): {kurtosis(returns_series):.4f}")
```

```
--- Statistical Properties of Log Returns ---
Mean (Average Daily Return): 0.001255
Standard Deviation (Volatility): 0.0194
Skewness: -0.2295
Kurtosis (Excess): 5.6739
```

Visualization 2 & Statistical Insights

1. **Volatility (**
$sigma$
): The Standard Deviation is the mathematical measure of volatility. A higher value means the stock has wider daily price swings.

2. **Skewness:** Measures the asymmetry. A positive skew (Skewness > 0) means the distribution has a longer tail of large positive gains, while a negative skew suggests more large losses.

3. **Kurtosis (Fat Tails):** A high **Excess Kurtosis** (Kurtosis > 0) indicates **'fat tails'**. This means extreme events (both large gains and large losses) happen *more often* than a standard Normal distribution would predict, confirming the inherent **risk of outliers** in financial data.

5. Summary and Next Steps

Key Takeaways

- **Return Calculation:** We successfully used `.pct_change()` and `np.log().diff()` to calculate both simple and log returns, the fundamental inputs for all financial models.
- **Performance & Risk:** The cumulative return plot shows the overall performance, while the return distribution and statistics (Std Dev, Kurtosis) quantify the **stock's inherent risk**.
- **Statistical Reality:** We confirmed that stock returns are generally **leptokurtic** (fat tails), meaning extreme market moves are a real and frequent risk.

Next Notebook Preview

- Daily returns show short-term noise.
- The next step in time series analysis is to smooth out this noise and find the underlying trends.
- We will use Pandas' powerful `.resample()` feature to aggregate data over weeks and months, revealing long-term patterns.

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