Stock Market Time Series Analysis with Pandas

Notebook 06: Rolling Statistics and Moving Averages

Python 3.8+ Pandas Latest Matplotlib Latest License MIT

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

Learning Objectives:

- Master Pandas `.rolling()` method for window-based statistics
- Calculate Simple Moving Averages (SMA) for trend analysis
- Identify moving average crossovers and trading signals
- Compute rolling volatility for dynamic risk assessment
- Understand momentum analysis through rolling windows

- In the last notebook, we used resampling to look at fixed periods (week, month).
- **Rolling statistics** (or window functions) offer a more dynamic view by calculating a metric over a *fixed window size* (e.g., the last 30 days) that moves forward one period at a time.

This technique is essential for smoothing out short-term fluctuations and identifying momentum shifts. We will cover:

1. **Simple Moving Averages (SMA):** Calculating and visualizing 7-day, 30-day, and 90-day SMAs.
2. **Moving Average Crossovers:** Identifying famous trading signals like the 'Golden Cross'.
3. **Rolling Volatility:** Quantifying risk over a fixed period using the rolling standard deviation.

In [2]:
```
# Importing necessary libraries
import pandas as pd
import numpy as np
import yfinance as yf
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style('whitegrid')
```

In [3]:
```
# Suppressing future warnings for cleaner output
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [ ]:

```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)
df.columns = df.columns.get_level_values(0) # Clean up MultiIndex

# Calculate Daily Log Returns (needed for Rolling Volatility later)
# Using natural logarithm for log returns
# shift(1) to get previous day's close price
df['Log_Return'] = np.log(df['Close'] / df['Close'].shift(1))

print("Initial DataFrame head:")
print(df[['Close', 'Log_Return']].head())
```

```
[*********************100%***********************]  1 of 1 completed
Initial DataFrame head:
Price           Close  Log_Return
Date
2019-01-02  37.575207         NaN
2019-01-03  33.832436   -0.104924
2019-01-04  35.276726    0.041803
2019-01-07  35.198212   -0.002228
2019-01-08  35.869183    0.018883
```

1. Simple Moving Averages (SMA)

- The Simple Moving Average is the average of the price over a specified number of periods.
- The longer the window, the smoother the line and the slower it reacts to price changes.

In [5]:

```python
# --- Concept: Pandas .rolling(window=N).mean() ---
# The .rolling() method creates a rolling window object.
# The .mean() function is then applied to all data points within that window.

# --- Code: Calculate Multiple SMAs ---
df['SMA_7'] = df['Close'].rolling(window=7).mean()
df['SMA_30'] = df['Close'].rolling(window=30).mean()
df['SMA_90'] = df['Close'].rolling(window=90).mean()

print("SMAs (first few non-NaN rows):")
# Show rows starting from the first non-NaN SMA (after 90 days)
print(df[['Close', 'SMA_7', 'SMA_30', 'SMA_90']].dropna().head())
```

```
SMAs (first few non-NaN rows):
Price           Close      SMA_7     SMA_30     SMA_90
Date
2019-05-10  47.299328  48.945546  48.151051  42.978486
2019-05-13  44.550320  48.170133  48.123055  43.055987
2019-05-14  45.255569  47.406714  48.108291  43.182911
2019-05-15  45.797680  46.832367  48.089455  43.299811
2019-05-16  45.596191  46.421086  48.053308  43.415344
```

Insights on SMAs

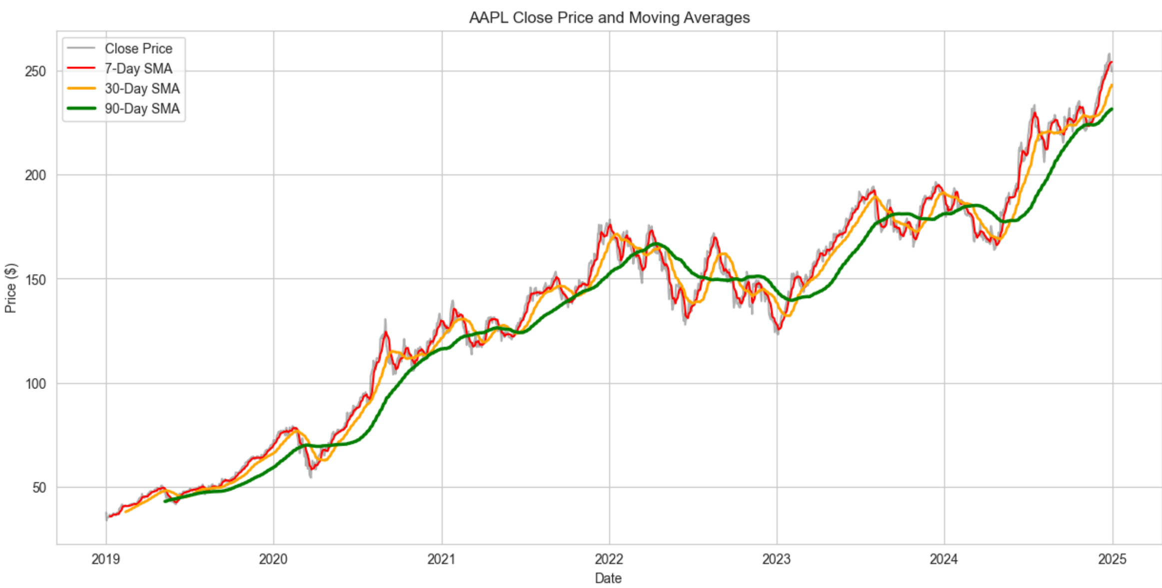
- In this output, all SMAs — SMA_7, SMA_30, and SMA_90 — have valid values. This means the data shown starts after the required window sizes (7, 30, and 90 days) have been met, so there are no NaN values present.
- **Support/Resistance:** The lines often act as price floors or ceilings. SMAs can act as dynamic support or resistance levels. For instance, the SMA_30 and SMA_90 may serve as key areas where the price could bounce or reverse direction.
- **Trend Direction:** When price is above the MA, the trend is generally considered bullish. The short-term SMA_7 is declining and sits below both the SMA_30 and SMA_90, suggesting short-term bearish momentum. Generally, when the price is above a moving average, it indicates a bullish trend — and when it's below, it suggests bearishness.

2. Visualization: Price and Moving Averages

- Overlaying different window lengths on the price provides a clear picture of short-term momentum versus long-term trend.

In [6]:
```
# --- Code: Plot SMAs ---
# --- Visualization: Close Price and Moving Averages ---
plt.figure(figsize=(15, 7))
plt.plot(df['Close'], label='Close Price', color='grey', alpha=0.6, linewidth=1.5)
plt.plot(df['SMA_7'], label='7-Day SMA', color='red', linewidth=1.5)
plt.plot(df['SMA_30'], label='30-Day SMA', color='orange', linewidth=2)
plt.plot(df['SMA_90'], label='90-Day SMA', color='green', linewidth=2.5)

plt.title(f'{TICKER} Close Price and Moving Averages')
plt.xlabel('Date')
plt.ylabel('Price ($)')
plt.legend()
plt.show()
```



Visualization 1 Insights

1. **Lag:** The 90-day SMA is the smoothest and lags the price the most, representing the **long-term trend**. The 7-day SMA follows the price closely, representing **short-term momentum**.
2. **Crossovers:** Notice areas where the fast MA (7-day) crosses the slow MA (90-day). These are potential **buy/sell signals** (trading strategies, discussed next).

3. Moving Average Crossovers (Trading Signals)

- A common trading signal is the **Golden Cross** (short-term MA crosses *above* long-term MA, suggesting a shift to a bullish trend) and the **Death Cross** (short-term MA crosses *below* long-term MA, suggesting a shift to a bearish trend).
- We can use simple Boolean indexing and `np.where()` to flag these signals.

In [7]:
```python
# --- Concept: Using .shift() for Crossovers ---
# A crossover occurs when: (Fast MA > Slow MA) AND (Fast MA.shift(1) < Slow MA.sh

# --- Code: Identify Golden/Death Crosses (using 30-day and 90-day) ---
fast_ma = df['SMA_30']
slow_ma = df['SMA_90']

# Condition 1: Fast MA is now ABOVE Slow MA, AND
# Condition 2: Fast MA was BELOW Slow MA yesterday
golden_cross_condition = (fast_ma > slow_ma) & (fast_ma.shift(1) <= slow_ma.shift(

df['Signal'] = np.where(golden_cross_condition, 'Golden Cross', None)
df['Signal'] = np.where((fast_ma < slow_ma) & (fast_ma.shift(1) >= slow_ma.shift(1

print("Identified trading signals (showing non-NaN signals):")
print(df[df['Signal'].notna()][['Close', 'SMA_30', 'SMA_90', 'Signal']].head(5))
```

```
Identified trading signals (showing non-NaN signals):
Price             Close      SMA_30      SMA_90        Signal
Date
2019-06-19    47.464836   45.043202   45.118048   Death Cross
2019-07-08    47.980583   46.058917   46.047075  Golden Cross
2020-03-23    54.316940   69.285386   69.769419   Death Cross
2020-05-19    76.012169   70.292540   70.247239  Golden Cross
2021-03-19   117.092484  123.296301  123.740428   Death Cross
```

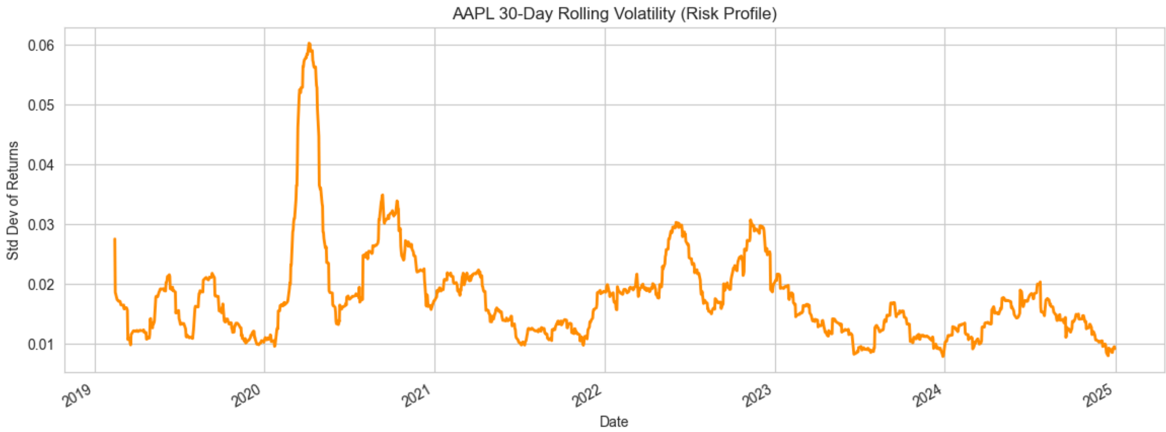
4. Rolling Volatility (Rolling Standard Deviation)

- In Notebook 04, we calculated overall volatility.
- Here, we calculate **Rolling Volatility** (rolling standard deviation of returns) to see how the stock's risk profile changes over time.

In [8]:
```python
# --- Concept: Rolling Volatility ---
# Volatility is the standard deviation of returns.
# We use a 30-day window on the Log Returns to measure short-term risk.

# --- Code: Calculate Rolling Volatility ---
df['Rolling_Vol_30'] = df['Log_Return'].rolling(window=30).std()
```

```python
# --- Visualization: Rolling Volatility ---
plt.figure(figsize=(14, 5))
df['Rolling_Vol_30'].plot(
    title=f'{TICKER} 30-Day Rolling Volatility (Risk Profile)',
    color='darkorange',
    linewidth=2
)
plt.xlabel('Date')
plt.ylabel('Std Dev of Returns')
plt.show()
```



Visualization 2 Insights

1. **Risk Spikes:** The peaks in the chart (especially early 2020) correspond to periods of extreme market turbulence. High rolling volatility means the stock is experiencing larger-than-average daily price movements (risk).

2. **Risk Trending:** We can observe if the stock is trending toward higher or lower risk. The rolling volatility smooths out noise, showing sustained periods of elevated risk.

3. **Low Volatility Traps:** Extremely low volatility periods can sometimes precede sharp price movements (shocks), making them interesting areas for further investigation.

5. Summary and Next Steps

Key Takeaways

- **Rolling Mastery:** We effectively used `.rolling()` to calculate dynamic, window-based statistics, which are central to technical analysis.

- **Trend Identification:** We created and plotted **SMAs** of different lengths, clarifying the relationship between price, short-term momentum, and long-term trends.

- **Risk Quantification:** We quantified dynamic risk using **Rolling Standard Deviation**, showing how the stock's volatility profile changes over time.

Next Notebook Preview

- Price is only one side of the market.
- The next crucial step is to analyze **Volume**, which dictates liquidity and market conviction.
- We will dedicate the next notebook to detailed volume analysis, including the calculation of **Volume-Weighted Average Price (VWAP)**.

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