Stock Market Time Series Analysis with Pandas

Notebook 09: Implementing Technical Indicators with Pandas

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

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

Learning Objectives:

- Master Exponential Moving Averages (EMA) using `.ewm()`
- Implement Bollinger Bands with rolling statistics
- Calculate Relative Strength Index (RSI) from scratch
- Understand advanced technical analysis concepts
- Build complex indicators using pure Pandas operations

- Technical Analysis (TA) relies on mathematical indicators derived from price, volume, and time.
- While many libraries exist for TA, building them manually using Pandas' `.rolling()` and `.ewm()` (Exponential Weighted Moving) functions is the best way to master time series manipulation.

This notebook focuses on the **Pandas Mastery** of TA:

1. **Exponential Moving Average (EMA):** Using `.ewm()` for weighted averages.
2. **Bollinger Bands (BB):** Combining SMA and Rolling Standard Deviation.
3. **Relative Strength Index (RSI):** A powerful momentum oscillator, showcasing complex multi-step calculations.

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

In [3]:
```python
# Reloading the data
TICKER = 'AAPL'
START_DATE = '2019-01-01'
END_DATE = '2024-01-01'
PERIOD = 20 # Standard period for Bollinger Bands and some EMAs

df = yf.download(TICKER, start=START_DATE, end=END_DATE)
df.columns = df.columns.get_level_values(0) # Clean up MultiIndex

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

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

Out[3]:
```
Date
2019-01-02    37.575207
2019-01-03    33.832436
2019-01-04    35.276722
2019-01-07    35.198204
2019-01-08    35.869194
Name: Close, dtype: float64
```

1. Exponential Moving Average (EMA)

- Unlike the Simple Moving Average (SMA) which gives equal weight to all data points, the **EMA** gives more weight to recent prices, making it more responsive to new information.
- Pandas' `.ewm()` is the dedicated function for this.

In [6]:
```python
# --- Concept: Pandas .ewm() ---
# .ewm(span=N) calculates the Exponential Weighted Moving average based on a peric

# --- Code: Calculate 20-day EMA ---
df['EMA_20'] = df['Close'].ewm(span=PERIOD, adjust=False).mean()

print(f"{PERIOD}-Day EMA (first few values):")
print(df['EMA_20'].head(PERIOD+1))
```

```
20-Day EMA (first few values):
Date
2019-01-02    37.575207
2019-01-03    37.218752
2019-01-04    37.033797
2019-01-07    36.858979
2019-01-08    36.764713
2019-01-09    36.737437
2019-01-10    36.723863
2019-01-11    36.677363
2019-01-14    36.583399
2019-01-15    36.567952
2019-01-16    36.596351
2019-01-17    36.642894
2019-01-18    36.706758
2019-01-22    36.684775
2019-01-23    36.678935
2019-01-24    36.646005
2019-01-25    36.730874
2019-01-28    36.774576
2019-01-29    36.777406
2019-01-30    37.019491
2019-01-31    37.265486
Name: EMA_20, dtype: float64
```

In [7]:

```python
# Compare with the SMA (from Notebook 06)
df['SMA_20'] = df['Close'].rolling(window=PERIOD).mean()

print("\nLast 5 values (EMA vs. SMA):")
print(df[['Close', 'SMA_20', 'EMA_20']].tail())
```

```
Last 5 values (EMA vs. SMA):
Price            Close      SMA_20      EMA_20
Date
2023-12-22  191.974670  192.042600  191.695879
2023-12-26  191.429306  192.204234  191.670491
2023-12-27  191.528442  192.340579  191.656963
2023-12-28  191.954849  192.549312  191.685333
2023-12-29  190.913666  192.677229  191.611841
```

Insights on EMA

1. **Responsiveness:** The EMA typically tracks the price closer than the SMA of the same period, confirming that it gives more weight to recent data. Traders use EMAs for faster trend identification.

2. **`adjust=False`:** In finance, `adjust=False` is often used in the `.ewm()` calculation to match the traditional EMA formula, especially when comparing to external trading software.

2. Bollinger Bands (BB)

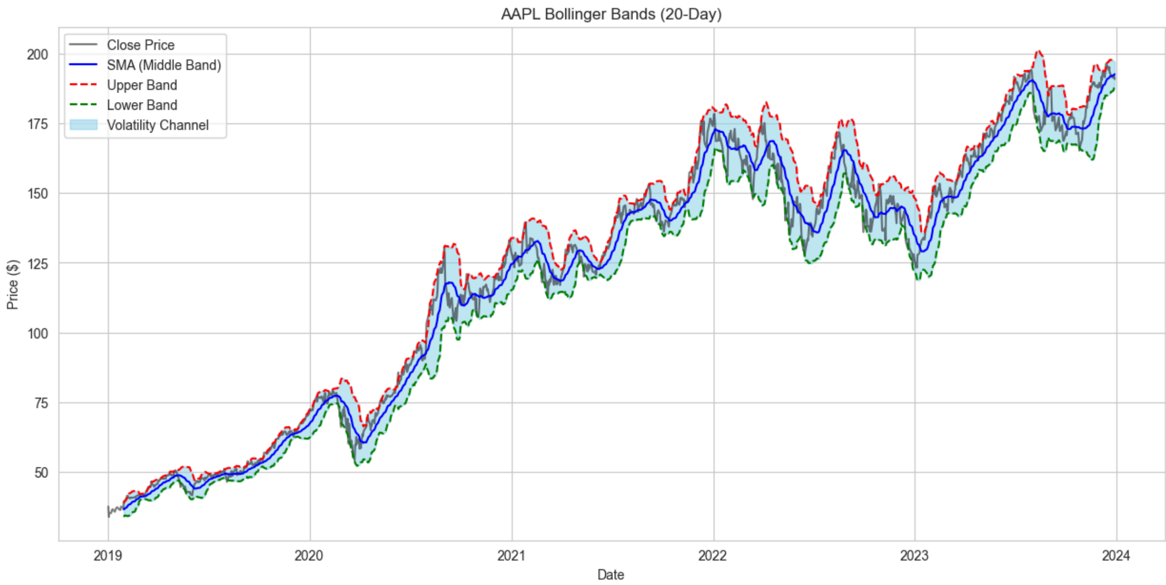
- **Bollinger Bands** are a volatility channel indicator.

- They consist of a central Simple Moving Average (SMA) and upper/lower bands set two standard deviations away from the SMA.

- They show if a price is relatively high or low.

In [8]:
```
# --- Concept: Bollinger Bands (BB) ---
# 1. Calculate N-period SMA (Middle Band)
# 2. Calculate N-period Rolling Standard Deviation (Volatility)
# 3. Upper Band = SMA + (2 * Std Dev)
# 4. Lower Band = SMA - (2 * Std Dev)

# --- Code: Calculate BB Components ---

# 1. Middle Band (20-day SMA)
df['Middle_Band'] = df['Close'].rolling(window=PERIOD).mean()

# 2. Rolling Standard Deviation (Volatility)
df['Rolling_Std'] = df['Close'].rolling(window=PERIOD).std()

# 3. Upper and Lower Bands (2 standard deviations away)
df['Upper_Band'] = df['Middle_Band'] + (df['Rolling_Std'] * 2)
df['Lower_Band'] = df['Middle_Band'] - (df['Rolling_Std'] * 2)
```

In [10]:
```
# --- Code: Plot Bollinger Bands ---
plt.figure(figsize=(15, 7))

# Plot the price and the bands
plt.plot(df['Close'], label='Close Price', color='black', alpha=0.5)
plt.plot(df['Middle_Band'], label='SMA (Middle Band)', color='blue', linewidth=1.5
plt.plot(df['Upper_Band'], label='Upper Band', color='red', linestyle='--')
plt.plot(df['Lower_Band'], label='Lower Band', color='green', linestyle='--')

# Fill the area between the bands
plt.fill_between(
    df.index,
    df['Lower_Band'],
    df['Upper_Band'],
    color='skyblue',
    alpha=0.5,
    label='Volatility Channel'
)

# Final plot adjustments
plt.title(f'{TICKER} Bollinger Bands (20-Day)')
plt.xlabel('Date')
plt.ylabel('Price ($)')
plt.legend()
plt.show()
```



Visualization 1 Insights

1. **Volatility Swings:** The distance between the bands is proportional to the stock's volatility. When the bands are close (contracting), volatility is low; when they widen (expanding), volatility is high.

2. **Oversold/Overbought:** Prices touching or exceeding the bands are often interpreted as potential overbought (Upper Band) or oversold (Lower Band) conditions, indicating a possible reversal.

3. Relative Strength Index (RSI)

- The **RSI** is a momentum oscillator that measures the speed and magnitude of recent price changes to evaluate overbought or oversold conditions.

- It is calculated over a 14-day period and is arguably the most complex indicator to build manually.

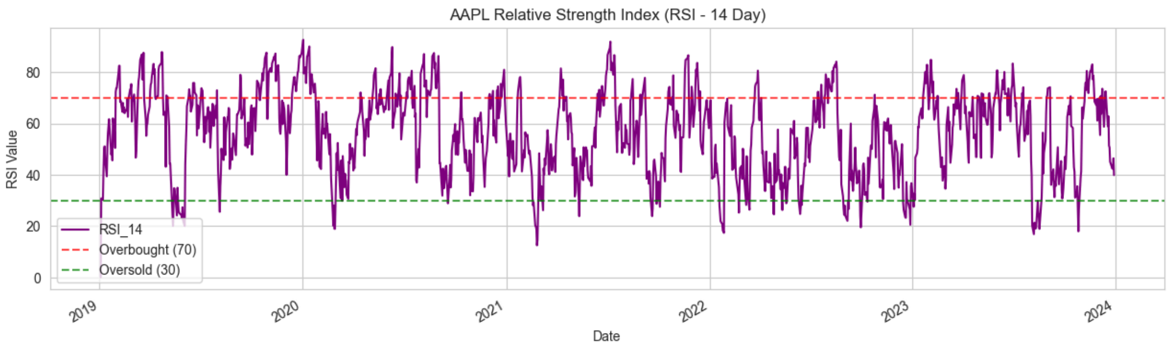
In [11]:

```
# --- Concept: RSI Calculation Steps ---
# 1. Calculate Daily Gains and Losses.
# 2. Calculate the 14-day EMA of Gains and Losses (Average Gain/Loss).
# 3. Calculate Relative Strength (RS) = Avg Gain / Avg Loss.
# 4. Calculate RSI = 100 - (100 / (1 + RS)).

# --- Code: Calculate RSI Components ---
rsi_period = 14

# 1. Calculate Daily Price Changes (Difference)
delta = df['Close'].diff(1)

# 2. Separate Gains (positive changes) and Losses (absolute value of negative char
gain = delta.where(delta > 0, 0) # where delta > 0, keep delta, else 0
loss = -delta.where(delta < 0, 0) # where delta < 0, keep -delta, else 0

# 3. Calculate Exponential Moving Average of Gains and Losses
avg_gain = gain.ewm(span=rsi_period, adjust=False).mean()
```

```
avg_loss = loss.ewm(span=rsi_period, adjust=False).mean()

# 4. Calculate Relative Strength (RS) and RSI
RS = avg_gain / avg_loss
df['RSI_14'] = 100 - (100 / (1 + RS))

print("RSI (first few non-NaN values):")
print(df['RSI_14'].dropna().head())
```

```
RSI (first few non-NaN values):
Date
2019-01-03     0.000000
2019-01-04    30.808016
2019-01-07    30.223928
2019-01-08    41.213634
2019-01-09    49.538402
Name: RSI_14, dtype: float64
```

In [13]:
```
# --- Code: Plot RSI ---
plt.figure(figsize=(15, 4))
df['RSI_14'].plot(
    title=f'{TICKER} Relative Strength Index (RSI - 14 Day)',
    color='purple',
    linewidth=1.5
)

# Add Overbought (70) and Oversold (30) thresholds
# axhline draws horizontal lines across the Axes
plt.axhline(70, color='red', linestyle='--', alpha=0.7, label='Overbought (70)')
plt.axhline(30, color='green', linestyle='--', alpha=0.7, label='Oversold (30)')

# Final plot adjustments
# plt.title(f'{TICKER} Relative Strength Index (RSI - 14 Day)')
plt.xlabel('Date')
plt.ylabel('RSI Value')
plt.legend()
plt.show()
```



Visualization 2 Insights

1. **Overbought/Oversold:** The RSI oscillates between 0 and 100. Values above 70 indicate that the stock may be **overbought** (momentum is too high), and values below 30 indicate it may be **oversold** (momentum is too low).

2. **Momentum Shift:** When the RSI crosses from below 30 back above, it is often considered a strong buy signal, indicating a reversal from the oversold state.

4. Summary and Next Steps

Key Takeaways

- **Advanced Pandas:** We successfully implemented three complex, industry-standard indicators—EMA, Bollinger Bands, and RSI—relying entirely on `.rolling()`, `.ewm()`, and basic NumPy/Pandas logic.
- **Volatility Modeling:** Bollinger Bands provided a visual volatility channel, helping us identify when prices were outside the typical range.
- **Momentum Tracking:** The RSI provided a quantitative measure of momentum, identifying potential overbought and oversold conditions.

Next Notebook Preview

- We have covered every tool in the kit!
- The final piece is the capstone project.
- The next notebook will be the **End-to-End Case Study**, where we combine data fetching, cleaning, resampling, rolling statistics, and all the technical indicators into a single, cohesive analysis to draw a final, business-like conclusion about a stock.

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