Stock Market Time Series Analysis with Pandas

Notebook 02: Working with the Datetime Index and Time Slicing

Python 3.8+ Pandas Latest License MIT

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

Learning Objectives:

- Master Pandas DatetimeIndex for time series analysis
- Learn powerful time slicing techniques using string references
- Extract time-based features from datetime indices
- Create visualizations with highlighted time periods
- Understand seasonal and cyclical pattern analysis

- In the previous notebook, we loaded our financial data.
- The **Datetime Index** is the single most powerful feature of a Pandas time series DataFrame.
- It allows us to treat the data's index not just as a label, but as a filter for time.

This notebook focuses on **Pandas Mastery** of the time index:

1. **Ensuring Correct Index Type:** Verifying and converting the date column to a proper `DatetimeIndex`.
2. **Time Slicing:** Efficiently filtering the DataFrame by year, month, or a range of dates.
3. **Feature Extraction:** Generating new columns from the index (e.g., year, month, weekday) for deeper analysis.
4. **Visualization:** Highlighting specific periods to visualize market events.

In [1]:

```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import yfinance as yf
import seaborn as sns

# Setting up visualization styles
sns.set_style('whitegrid')
```

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

In [4]:
```python
# Reloading the data (using the same logic as before)
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 (as established in previous notebooks)
df.columns = df.columns.get_level_values(0)

print("Initial 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	33.832443	34.672365	33.787234	34.258351	365248800
2019-01-04	35.276730	35.345734	34.215527	34.389221	234428400
2019-01-07	35.198204	35.412351	34.715190	35.381418	219111200
2019-01-08	35.869190	36.123786	35.338589	35.586043	164101200

1. Ensuring the Datetime Index

- When fetching data with `yfinance`, the index is usually correct.
- However, if loading from a CSV, we must manually ensure the index is a Pandas `DatetimeIndex` for time slicing to work.

In [5]:
```python
# --- Concept: Checking the Index ---
# If the index were a standard column named 'Date', the steps to fix it would be:
# 1. df['Date'] = pd.to_datetime(df['Date'])
# 2. df = df.set_index('Date')

# --- Code: Verification ---
print("Type of the DataFrame's index:")
print(type(df.index))
```

```
Type of the DataFrame's index:
<class 'pandas.core.indexes.datetimes.DatetimeIndex'>
```

In [6]:
```python
# Check if the index is a DatetimeIndex
if isinstance(df.index, pd.DatetimeIndex):
    print("\nIndex is correctly formatted as a DatetimeIndex. Proceeding...")
else:
```

```python
    # If loading from CSV, this block would typically run
    print("\nIndex needs conversion! This step is critical.")
```

```
Index is correctly formatted as a DatetimeIndex. Proceeding...
```

2. Powerful Time Slicing

- One of the greatest benefits of the `DatetimeIndex` is the ability to filter data using simple string references.

A. Slicing by Year

In [7]:
```python
# --- Concept: Filtering by Year ---
# Pass a simple year string to retrieve all rows for that year.

# --- Code: Slice 2022 ---
df_2022 = df.loc['2022']

print("Shape of the entire DataFrame:", df.shape)
print("Shape of the 2022 slice:", df_2022.shape)
print("\nFirst day of 2022 slice:", df_2022.index.min())
print("Last day of 2022 slice:", df_2022.index.max())
```

```
Shape of the entire DataFrame: (1510, 5)
Shape of the 2022 slice: (251, 5)

First day of 2022 slice: 2022-01-03 00:00:00
Last day of 2022 slice: 2022-12-30 00:00:00
```

B. Slicing by Month and Range

In [8]:
```python
# --- Concept: Filtering by Month/Range ---
# You can specify the month ('YYYY-MM') or a continuous range of dates ('YYYY-MM-L

# --- Code: Slice March 2022 ---
df_mar_2022 = df.loc['2022-03']
print("\nData for March 2022 (first 5 rows):")
print(df_mar_2022['Close'].head())
```

```
Data for March 2022 (first 5 rows):
Date
2022-03-01    160.205627
2022-03-02    163.503983
2022-03-03    163.179993
2022-03-04    160.176147
2022-03-07    156.377136
Name: Close, dtype: float64
```

In [9]:
```python
# --- Code: Slice 6-Month Range ---
# Select the first half of 2021
df_H1_2021 = df.loc['2021-01-01':'2021-06-30']
print("\nData for H1 2021 (total trading days):", df_H1_2021.shape[0])
```

```
Data for H1 2021 (total trading days): 124
```

Insights on Time Slicing

- Time slicing is faster and cleaner than creating Boolean masks (e.g., `df[(df.index >= start) & (df.index <= end)]`).
- It's essential for comparing performance across different market conditions (e.g., pre-COVID vs. post-COVID).

3. Feature Extraction from the Index

- We can extract attributes like year, month, or weekday directly from the index.
- This allows us to look for **seasonal or cyclical patterns**.

In [10]:
```python
# --- Concept: Using the .dt accessor ---
# The `.dt` accessor is used to access datetime components (year, month, day, week

# --- Code: Creating New Features ---
# Extracting year, month, and weekday from the index
df['Year'] = df.index.year
df['Month'] = df.index.month

# Weekday: Monday=0, Sunday=6
df['Weekday'] = df.index.dayofweek

# Check the new columns
print("DataFrame with new time features:")
df[['Close', 'Year', 'Month', 'Weekday']].tail()
```

```
DataFrame with new time features:
```

Out[10]:

Price	**Close**	**Year**	**Month**	**Weekday**
Date				
2024-12-24	257.286652	2024	12	1
2024-12-26	258.103729	2024	12	3
2024-12-27	254.685883	2024	12	4
2024-12-30	251.307877	2024	12	0
2024-12-31	249.534180	2024	12	1

A. Visualizing Weekly Behavior

- Does the stock perform differently on Mondays (0) versus Fridays (4)?
- We can use the extracted `Weekday` feature to find out.

In [12]:
```python
# Checking random samples of the DataFrame to ensure new columns are added correct
print("\nRandom samples from the DataFrame:")
df.sample(5)
```

```
Random samples from the DataFrame:
```

Out[12]:

Price	Close	High	Low	Open	Volume	Year	Month	Week
Date								
2019-06-21	47.683140	48.179691	47.532015	47.687938	191202400	2019	6	
2024-09-19	227.809769	228.755380	223.589420	223.947753	66781300	2024	9	
2019-04-02	46.362770	46.467912	45.653062	45.662619	91062800	2019	4	
2020-07-09	92.972534	93.521126	91.923888	93.467722	125642800	2020	7	
2021-10-18	143.468643	143.752539	140.149922	140.433818	85589200	2021	10	

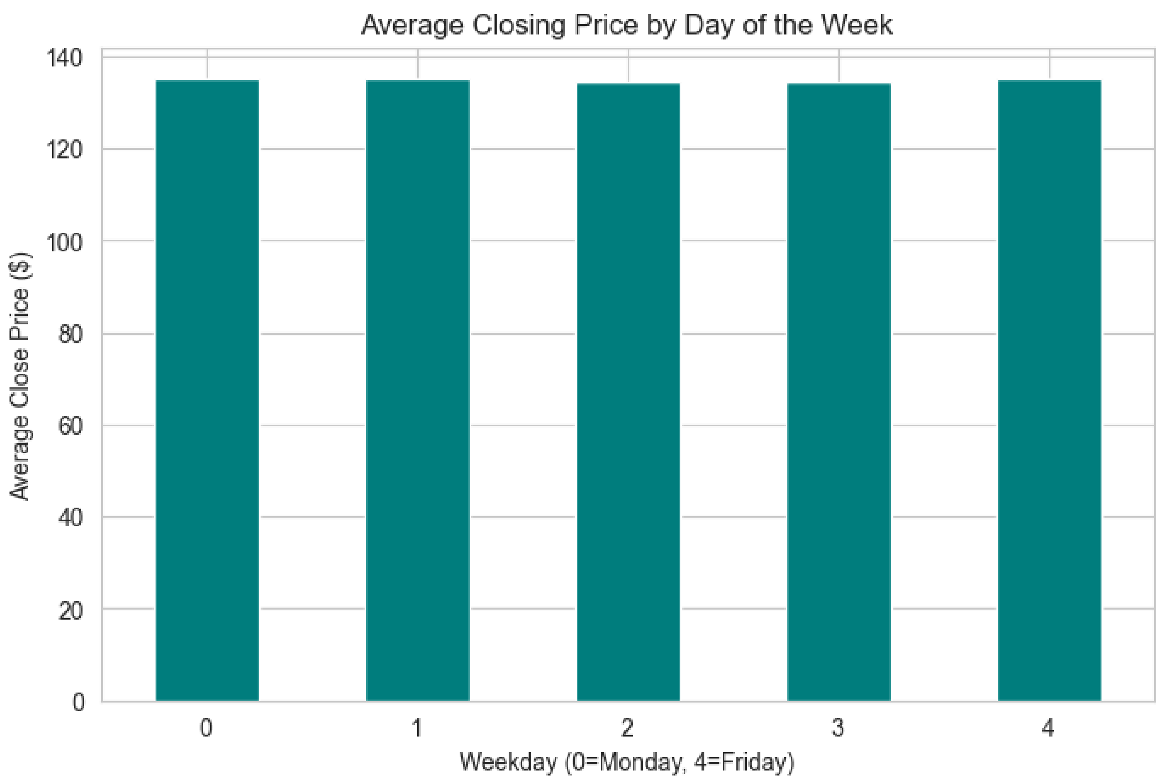
In [11]:

```
# --- Code: Mean Close Price by Weekday ---

# Calculate average closing price by weekday
avg_price_by_weekday = df.groupby('Weekday')['Close'].mean()

# Plotting the average closing price by weekday
plt.figure(figsize=(8, 5))
avg_price_by_weekday.plot(kind='bar', color='teal')

# Adding titles and labels
plt.title('Average Closing Price by Day of the Week')
plt.xlabel('Weekday (0=Monday, 4=Friday)')
plt.ylabel('Average Close Price ($)')
plt.xticks(rotation=0)
plt.show()
```



Insights on Weekly Behavior (Visualization 1)

- **Caveat:** The above chart shows a strong upward trend because the stock price generally increases over time (later years have higher prices, which are equally distributed across all weekdays). *This doesn't show daily returns, only absolute price level.*
- However, we can infer that **all weekdays are equally represented in the price increase**. If one day were consistently lower, it might indicate a systematic bias. We'll properly analyze weekday returns in a later notebook.

4. Visualization: Highlighting Key Market Events

- We can use the `DatetimeIndex` and Matplotlib's spanning features to visually mark significant periods like the 2020 market crash.

In [13]:

```python
# --- Concept: Highlighting a Time Span ---
# We plot the full price history and use plt.axvspan to shade the area of interest

# --- Code: Plot with Highlight ---
# figure and axis setup
# subplots to create a figure and axis
fig, ax = plt.subplots(figsize=(14, 6))

# Plot the closing price with axis, specified line width and color
df['Close'].plot(ax=ax, linewidth=1.5, color='darkblue')

# axvspan to highlight a specific time period in the plot
# Define the span for the 2020 major crash (approx. Feb 2020 to Apr 2020)
ax.axvspan(
    '2020-02-19',
    '2020-04-01',
    color='red',
    alpha=0.2,
    label='2020 Crash Period'
)

ax.set_title(f'{TICKER} Price with Highlighted Market Event')
ax.set_xlabel('Date')
ax.set_ylabel('Close Price ($)')
ax.legend()
plt.show()
```

Visualization 2 Insights

- This plot effectively uses time slicing visualization to tell a story:

    1. **Contextualization:** By highlighting the crash period, we clearly isolate the high-risk, high-volatility window from the rest of the trend.

    2. **Magnitude:** The chart visually demonstrates the **sharpness** of the decline and the equally **aggressive V-shaped recovery** that followed, a key characteristic of the 2020 market.

5. Summary and Next Steps

Key Takeaways

- **Pandas Mastery:** We leveraged the `DatetimeIndex` for powerful, single-line **time slicing** (e.g., `df.loc['2022-03']`) to filter data with precision.

- **Feature Engineering:** We successfully extracted **time-based features** (`Year`, `Month`, `Weekday`) directly from the index, allowing us to group data and look for cyclic patterns.

- **Storytelling:** We used visualization tools like `axvspan` to highlight and contextualize specific market events within the overall trend.

Next Notebook Preview

- The next step is to master the visual side of the data.

- We will move beyond simple line plots to explore advanced financial chart types and powerful visualization techniques to uncover deeper trends and patterns in the stock data.

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