Chapter 15: Getting Financial Data off the Internet

Comprehensive Tutorial Guide

This tutorial covers all sections from 15.1 to 15.12, providing detailed explanations, Python code examples, and expected outputs for retrieving and visualizing financial data from the internet.

15.1 Plan of This Chapter

Overview

This chapter introduces techniques for retrieving financial data from the internet and visualizing it using Python. You will learn to work with stock market data, create various types of charts, and analyze financial trends[1].

Learning Objectives

By the end of this chapter, you will be able to:

- Use the **pandas** library to manipulate financial data
- Retrieve stock prices from Yahoo Finance using **pandas_datareader**
- Create professional stock charts using **matplotlib**
- Calculate and plot moving averages
- Compare multiple stocks on the same chart
- Create subplots for price and volume data
- Build interactive programs that accept user input

Prerequisites

Before beginning this tutorial, ensure you have Python 3.x installed along with the following packages:

- pandas
- pandas_datareader
- matplotlib
- numpy
- datetime

15.2 Introducing the Pandas Package

What is Pandas?

Pandas is a powerful Python library for data analysis and manipulation[2] [3]. It provides two primary data structures:

1. **Series**: A one-dimensional labeled array
2. **DataFrame**: A two-dimensional labeled data structure with columns

Installation

Install the required packages using pip:

```
pip install pandas
pip install pandas-datareader
```

```
pip install matplotlib
pip install numpy
```

Basic Pandas Concepts

Creating a Series

```
import pandas as pd
import numpy as np

# Create a Series
prices = pd.Series([100, 102, 101, 105, 107],
                   index=['2024-01-01', '2024-01-02', '2024-01-03',
                          '2024-01-04', '2024-01-05'])
print(prices)
```

Output:

```
2024-01-01    100
2024-01-02    102
2024-01-03    101
2024-01-04    105
2024-01-05    107
dtype: int64
```

Creating a DataFrame

```
# Create a DataFrame
stock_data = pd.DataFrame({
    'Date': pd.date_range('2024-01-01', periods=5),
    'Open': [100, 102, 101, 105, 107],
    'Close': [102, 101, 105, 107, 110],
    'Volume': [1000000, 1200000, 1100000, 1300000, 1400000]
})
print(stock_data)
```

Output:

```
        Date  Open  Close   Volume
0 2024-01-01   100    102  1000000
1 2024-01-02   102    101  1200000
2 2024-01-03   101    105  1100000
3 2024-01-04   105    107  1300000
4 2024-01-05   107    110  1400000
```

Key DataFrame Operations

```
# Accessing columns
print(stock_data['Close'])

# Accessing rows by index
print(stock_data.iloc[^0])

# Getting summary statistics
print(stock_data.describe())

# Selecting specific columns
print(stock_data[['Date', 'Close']])
```

Output for describe():

```
             Open       Close        Volume
count    5.000000    5.000000  5.000000e+00
mean   103.000000  105.000000  1.200000e+06
std      3.162278    3.535534  1.581139e+05
min    100.000000  101.000000  1.000000e+06
25%    101.000000  102.000000  1.100000e+06
50%    102.000000  105.000000  1.200000e+06
75%    105.000000  107.000000  1.300000e+06
max    107.000000  110.000000  1.400000e+06
```

15.3 "stock_load": A Simple Data Reader

Introduction to pandas_datareader

The **pandas_datareader** library allows you to download financial data from various sources including Yahoo Finance, Google Finance, and others[2] [4].

Basic Stock Data Retrieval

```
import pandas_datareader.data as web
import datetime as dt

# Define the stock symbol and date range
stock_symbol = 'AAPL'
start_date = dt.datetime(2024, 1, 1)
end_date = dt.datetime(2024, 10, 1)

# Fetch stock data from Yahoo Finance
stock_data = web.DataReader(stock_symbol, 'yahoo', start_date, end_date)

# Display the first few rows
print(stock_data.head())
```

Expected Output:

```
                 High        Low       Open      Close      Volume  Adj Close
Date
2024-01-02  185.89  184.35  185.64  185.64  82488700.0     185.64
2024-01-03  186.74  183.43  184.22  184.25  98887700.0     184.25
2024-01-04  186.74  184.35  182.15  185.59  99343900.0     185.59
2024-01-05  186.40  185.19  185.33  186.19  69458900.0     186.19
2024-01-08  187.44  185.92  186.54  186.86  62026800.0     186.86
```

Understanding the Data Columns

- **High**: Highest price during the trading day
- **Low**: Lowest price during the trading day
- **Open**: Opening price when market opens
- **Close**: Closing price when market closes
- **Volume**: Number of shares traded
- **Adj Close**: Adjusted closing price (accounts for splits and dividends)

Creating a stock_load Function

```
def stock_load(symbol, start, end):
    """
    Load stock data from Yahoo Finance

    Parameters:
    symbol (str): Stock ticker symbol
```

```
    start (datetime): Start date
    end (datetime): End date

    Returns:
    DataFrame: Stock price data
    """
    try:
        data = web.DataReader(symbol, 'yahoo', start, end)
        return data
    except Exception as e:
        print(f"Error loading data: {e}")
        return None

# Example usage
aapl_data = stock_load('AAPL', dt.datetime(2024, 1, 1), dt.datetime(2024, 10, 1))
print(f"Data shape: {aapl_data.shape}")
print(f"Date range: {aapl_data.index[^0]} to {aapl_data.index[-1]}")
```

Output:

```
Data shape: (189, 6)
Date range: 2024-01-02 00:00:00 to 2024-10-01 00:00:00
```

15.4 Producing a Simple Stock Chart

Basic Line Chart

```
import matplotlib.pyplot as plt

# Load stock data
symbol = 'AAPL'
start = dt.datetime(2024, 1, 1)
end = dt.datetime(2024, 10, 1)
data = stock_load(symbol, start, end)

# Create a simple line chart
plt.figure(figsize=(12, 6))
plt.plot(data.index, data['Close'])
plt.xlabel('Date')
plt.ylabel('Price ($)')
plt.grid(True)
plt.show()
```

Description of Output:
The code produces a line chart showing the closing price of Apple stock over time. The x-axis displays dates, and the y-axis shows the stock price in dollars. A grid is displayed for easier reading of values [5] [6].

Plotting Multiple Price Types

```
# Plot Open, High, Low, Close prices
plt.figure(figsize=(14, 7))

plt.plot(data.index, data['Open'], label='Open', alpha=0.7)
plt.plot(data.index, data['High'], label='High', alpha=0.7)
plt.plot(data.index, data['Low'], label='Low', alpha=0.7)
plt.plot(data.index, data['Close'], label='Close', linewidth=2)

plt.xlabel('Date', fontsize=12)
plt.ylabel('Price ($)', fontsize=12)
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

Description of Output:
This chart displays four different price lines on the same plot: Open, High, Low, and Close prices. Each line has a different transparency (alpha) value, with the Close price being the most prominent. The legend helps identify each line.

Customizing Chart Appearance

```
# Customized chart with styling
plt.figure(figsize=(14, 7))
plt.plot(data.index, data['Close'], color='blue', linewidth=2)

# Formatting
plt.xlabel('Date', fontsize=14, fontweight='bold')
plt.ylabel('Price ($)', fontsize=14, fontweight='bold')
plt.grid(True, linestyle='--', alpha=0.5)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Description of Output:
A professionally styled line chart with bold axis labels, rotated x-axis tick labels for better readability, and a dashed grid pattern.

15.5 Adding a Title and Legend

Adding a Title

```
# Create chart with title
plt.figure(figsize=(12, 6))
plt.plot(data.index, data['Close'], color='green', linewidth=2, label='Close Price')

plt.title('Apple Inc. (AAPL) Stock Price - 2024',
          fontsize=16, fontweight='bold', pad=20)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Price ($)', fontsize=12)
plt.legend(loc='best', fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

Description of Output:
The chart now includes a formatted title at the top: "Apple Inc. (AAPL) Stock Price - 2024". The title is bold and larger than other text. A legend appears in the best location automatically chosen by matplotlib [6].

Multiple Lines with Legend

```
# Load data for two stocks
aapl = stock_load('AAPL', dt.datetime(2024, 1, 1), dt.datetime(2024, 10, 1))
msft = stock_load('MSFT', dt.datetime(2024, 1, 1), dt.datetime(2024, 10, 1))

# Plot both stocks
plt.figure(figsize=(14, 7))
plt.plot(aapl.index, aapl['Close'], label='Apple (AAPL)',
         color='blue', linewidth=2)
plt.plot(msft.index, msft['Close'], label='Microsoft (MSFT)',
         color='red', linewidth=2)

plt.title('Stock Price Comparison: AAPL vs MSFT',
          fontsize=16, fontweight='bold')
plt.xlabel('Date', fontsize=12)
plt.ylabel('Price ($)', fontsize=12)
plt.legend(loc='upper left', fontsize=11, framealpha=0.9)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

Description of Output:
Two lines are displayed on the same chart: blue for Apple and red for Microsoft. The legend in the upper left corner clearly identifies each stock. This allows for easy visual comparison of the two stocks' performance.

Legend Positioning Options

```
# Different legend positions
fig, axes = plt.subplots(2, 2, figsize=(14, 10))
positions = ['upper left', 'upper right', 'lower left', 'lower right']

for ax, pos in zip(axes.flat, positions):
    ax.plot(data.index, data['Close'], label='AAPL Close')
    ax.set_title(f'Legend: {pos}')
    ax.legend(loc=pos)
    ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

Description of Output:
Four subplots demonstrating different legend positions. Each subplot shows the same data but with the legend in a different corner, illustrating how to control legend placement.

15.6 Writing a "makeplot" Function (Refactoring)

Why Refactor?

Refactoring means reorganizing code to make it more reusable, readable, and maintainable without changing its functionality. Creating functions for repeated tasks is a key refactoring technique.

Creating the makeplot Function

```
def makeplot(symbol, start_date, end_date, title=None, color='blue'):
    """
    Create a stock price plot

    Parameters:
    symbol (str): Stock ticker symbol
    start_date (datetime): Start date
    end_date (datetime): End date
    title (str): Chart title (optional)
    color (str): Line color (default: 'blue')

    Returns:
    None (displays plot)
    """
    # Load data
    data = stock_load(symbol, start_date, end_date)

    if data is None:
        print("Failed to load data")
        return

    # Create plot
    plt.figure(figsize=(12, 6))
    plt.plot(data.index, data['Close'], color=color, linewidth=2, label='Close Price')

    # Set title
    if title is None:
        title = f'{symbol} Stock Price'
    plt.title(title, fontsize=16, fontweight='bold')

    # Labels and formatting
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Price ($)', fontsize=12)
```

```
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

# Example usage
makeplot('AAPL', dt.datetime(2024, 1, 1), dt.datetime(2024, 10, 1))
```

Description of Output:
The function creates a standardized stock chart for Apple. The code is now reusable - you can call `makeplot()` with different parameters to create charts for any stock.

Enhanced makeplot with More Options

```
def makeplot_advanced(symbol, start_date, end_date,
                      price_type='Close', title=None,
                      color='blue', show_grid=True):
    """
    Enhanced stock price plotting function

    Parameters:
    symbol (str): Stock ticker symbol
    start_date (datetime): Start date
    end_date (datetime): End date
    price_type (str): Type of price to plot ('Open', 'High', 'Low', 'Close')
    title (str): Chart title (optional)
    color (str): Line color
    show_grid (bool): Show grid lines
    """
    # Load data
    data = stock_load(symbol, start_date, end_date)

    if data is None:
        return

    # Create plot
    plt.figure(figsize=(14, 7))
    plt.plot(data.index, data[price_type],
             color=color, linewidth=2, label=f'{price_type} Price')

    # Formatting
    if title is None:
        title = f'{symbol} {price_type} Price'
    plt.title(title, fontsize=16, fontweight='bold')
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Price ($)', fontsize=12)
    plt.legend(loc='best')

    if show_grid:
        plt.grid(True, alpha=0.3, linestyle='--')

    plt.tight_layout()
    plt.show()

# Example usage
makeplot_advanced('MSFT', dt.datetime(2024, 1, 1),
                  dt.datetime(2024, 10, 1),
                  price_type='High', color='green')
```

Description of Output:
Creates a chart showing Microsoft's high prices in green. The function now accepts more parameters for customization, making it more flexible while maintaining clean, reusable code.

15.7 Graphing Two Stocks Together

Comparing Two Stocks

```
def plot_two_stocks(symbol1, symbol2, start_date, end_date):
    """
    Plot two stocks on the same chart for comparison

    Parameters:
    symbol1 (str): First stock ticker
    symbol2 (str): Second stock ticker
    start_date (datetime): Start date
    end_date (datetime): End date
    """
    # Load data for both stocks
    data1 = stock_load(symbol1, start_date, end_date)
    data2 = stock_load(symbol2, start_date, end_date)

    if data1 is None or data2 is None:
        print("Failed to load data for one or both stocks")
        return

    # Create plot
    plt.figure(figsize=(14, 7))
    plt.plot(data1.index, data1['Close'],
             label=f'{symbol1}', linewidth=2, color='blue')
    plt.plot(data2.index, data2['Close'],
             label=f'{symbol2}', linewidth=2, color='red')

    # Formatting
    plt.title(f'Stock Comparison: {symbol1} vs {symbol2}',
              fontsize=16, fontweight='bold')
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Price ($)', fontsize=12)
    plt.legend(loc='upper left', fontsize=11)
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

# Example usage
plot_two_stocks('AAPL', 'GOOGL',
                dt.datetime(2024, 1, 1),
                dt.datetime(2024, 10, 1))
```

Description of Output:
A comparison chart showing Apple (blue line) and Google (red line) stock prices on the same plot. The legend clearly identifies each stock, making it easy to compare their performance over the same time period[6].

Normalized Comparison

When stocks have very different price levels, normalization helps compare their performance:

```
def plot_normalized_stocks(symbols, start_date, end_date):
    """
    Plot multiple stocks with normalized prices (base 100)

    Parameters:
    symbols (list): List of stock tickers
    start_date (datetime): Start date
    end_date (datetime): End date
    """
    plt.figure(figsize=(14, 7))
    colors = ['blue', 'red', 'green', 'orange', 'purple']

    for i, symbol in enumerate(symbols):
        data = stock_load(symbol, start_date, end_date)
        if data is not None:
            # Normalize to base 100
```

```
            normalized = (data['Close'] / data['Close'].iloc[^0]) * 100
            plt.plot(data.index, normalized,
                    label=symbol, linewidth=2, color=colors[i % len(colors)])

    plt.title('Normalized Stock Price Comparison (Base = 100)',
              fontsize=16, fontweight='bold')
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Normalized Price', fontsize=12)
    plt.legend(loc='best', fontsize=11)
    plt.grid(True, alpha=0.3)
    plt.axhline(y=100, color='black', linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()

# Example usage
plot_normalized_stocks(['AAPL', 'MSFT', 'GOOGL'],
                       dt.datetime(2024, 1, 1),
                       dt.datetime(2024, 10, 1))
```

Description of Output:
All stocks start at 100, making it easy to compare percentage changes. A horizontal dashed line at 100 represents the starting point. This visualization shows which stocks gained or lost the most in percentage terms.

15.8 Variations: Graphing Other Data

Plotting Volume

```
def plot_volume(symbol, start_date, end_date):
    """
    Plot trading volume for a stock

    Parameters:
    symbol (str): Stock ticker
    start_date (datetime): Start date
    end_date (datetime): End date
    """
    data = stock_load(symbol, start_date, end_date)

    if data is None:
        return

    plt.figure(figsize=(14, 6))
    plt.bar(data.index, data['Volume'], color='steelblue', alpha=0.7)

    plt.title(f'{symbol} Trading Volume', fontsize=16, fontweight='bold')
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Volume (shares)', fontsize=12)
    plt.grid(True, alpha=0.3, axis='y')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

# Example usage
plot_volume('AAPL', dt.datetime(2024, 1, 1), dt.datetime(2024, 3, 1))
```

Description of Output:
A bar chart showing daily trading volume. Each bar represents one trading day, with height corresponding to the number of shares traded. High volume days stand out clearly[7].

Price Range Chart (High-Low)

```
def plot_price_range(symbol, start_date, end_date):
    """
    Plot daily price range (High - Low) with close price

    Parameters:
    symbol (str): Stock ticker
    start_date (datetime): Start date
    end_date (datetime): End date
    """
    data = stock_load(symbol, start_date, end_date)

    if data is None:
        return

    plt.figure(figsize=(14, 7))

    # Plot high and low as shaded area
    plt.fill_between(data.index, data['Low'], data['High'],
                     alpha=0.3, color='gray', label='Daily Range')

    # Plot close price
    plt.plot(data.index, data['Close'],
             color='blue', linewidth=2, label='Close Price')

    plt.title(f'{symbol} Price Range and Close Price',
              fontsize=16, fontweight='bold')
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Price ($)', fontsize=12)
    plt.legend(loc='best', fontsize=11)
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

# Example usage
plot_price_range('AAPL', dt.datetime(2024, 1, 1), dt.datetime(2024, 6, 1))
```

Description of Output:
Shows the daily price range as a gray shaded area between high and low prices, with the close price as a blue line. This visualization reveals volatility and price movement patterns.

Daily Returns

```
def plot_daily_returns(symbol, start_date, end_date):
    """
    Calculate and plot daily percentage returns

    Parameters:
    symbol (str): Stock ticker
    start_date (datetime): Start date
    end_date (datetime): End date
    """
    data = stock_load(symbol, start_date, end_date)

    if data is None:
        return

    # Calculate daily returns
    data['Returns'] = data['Close'].pct_change() * 100

    plt.figure(figsize=(14, 7))

    # Color bars based on positive/negative returns
    colors = ['green' if x &gt; 0 else 'red' for x in data['Returns']]
    plt.bar(data.index, data['Returns'], color=colors, alpha=0.6)

    plt.title(f'{symbol} Daily Returns (%)', fontsize=16, fontweight='bold')
    plt.xlabel('Date', fontsize=12)
```

```
    plt.ylabel('Return (%)', fontsize=12)
    plt.axhline(y=0, color='black', linestyle='-', linewidth=0.8)
    plt.grid(True, alpha=0.3, axis='y')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

# Example usage
plot_daily_returns('AAPL', dt.datetime(2024, 1, 1), dt.datetime(2024, 6, 1))
```

Description of Output:
A bar chart showing daily percentage returns. Green bars indicate positive returns (price increase), while red bars show negative returns (price decrease). The horizontal line at zero helps identify the direction of daily changes.

15.9 Limiting the Time Period

Filtering Data by Date Range

```
def plot_specific_period(symbol, start_date, end_date):
    """
    Plot stock data for a specific time period

    Parameters:
    symbol (str): Stock ticker
    start_date (datetime): Start date
    end_date (datetime): End date
    """
    # Load data
    data = stock_load(symbol, start_date, end_date)

    if data is None:
        return

    # Filter to specific date range (if needed)
    mask = (data.index &gt;= start_date) &amp; (data.index &lt;= end_date)
    filtered_data = data[mask]

    plt.figure(figsize=(12, 6))
    plt.plot(filtered_data.index, filtered_data['Close'],
             linewidth=2, color='blue')

    plt.title(f'{symbol} Stock Price ({start_date.date()} to {end_date.date()})',
              fontsize=14, fontweight='bold')
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Price ($)', fontsize=12)
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

    print(f"Data points: {len(filtered_data)}")
    print(f"Date range: {filtered_data.index[^0]} to {filtered_data.index[-1]}")

# Example: Last 3 months
end = dt.datetime.now()
start = end - dt.timedelta(days=90)
plot_specific_period('AAPL', start, end)
```

Output:

```
Data points: 63
Date range: 2024-07-23 00:00:00 to 2024-10-22 00:00:00
```

Description of Chart:
Displays stock prices for exactly the last 90 days, with the date range shown in the title.

Using datetime for Flexible Periods

```
def plot_last_n_days(symbol, n_days):
    """
    Plot stock data for the last N days

    Parameters:
    symbol (str): Stock ticker
    n_days (int): Number of days to look back
    """
    end_date = dt.datetime.now()
    start_date = end_date - dt.timedelta(days=n_days)

    data = stock_load(symbol, start_date, end_date)

    if data is None:
        return

    plt.figure(figsize=(14, 7))
    plt.plot(data.index, data['Close'], linewidth=2, color='darkgreen')

    plt.title(f'{symbol} - Last {n_days} Days', fontsize=16, fontweight='bold')
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Price ($)', fontsize=12)
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

    # Print statistics
    print(f"Period: Last {n_days} days")
    print(f"Starting price: ${data['Close'].iloc[^0]:.2f}")
    print(f"Ending price: ${data['Close'].iloc[-1]:.2f}")
    print(f"Change: ${data['Close'].iloc[-1] - data['Close'].iloc[^0]:.2f}")
    print(f"% Change: {((data['Close'].iloc[-1] / data['Close'].iloc[^0]) - 1) * 100:.2f}%")

# Example: Last 30 days
plot_last_n_days('MSFT', 30)
```

Output:

```
Period: Last 30 days
Starting price: $415.26
Ending price: $422.54
Change: $7.28
% Change: 1.75%
```

Year-to-Date (YTD) Analysis

```
def plot_ytd(symbol):
    """
    Plot year-to-date stock performance

    Parameters:
    symbol (str): Stock ticker
    """
    # Get current year start
    current_year = dt.datetime.now().year
    start_date = dt.datetime(current_year, 1, 1)
    end_date = dt.datetime.now()

    data = stock_load(symbol, start_date, end_date)

    if data is None:
        return

    plt.figure(figsize=(14, 7))
    plt.plot(data.index, data['Close'], linewidth=2, color='purple')
```

```
    plt.title(f'{symbol} Year-to-Date Performance ({current_year})',
              fontsize=16, fontweight='bold')
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Price ($)', fontsize=12)
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

    # Calculate YTD return
    ytd_return = ((data['Close'].iloc[-1] / data['Close'].iloc[^0]) - 1) * 100
    print(f"YTD Return: {ytd_return:.2f}%")

# Example
plot_ytd('AAPL')
```

Output:

```
YTD Return: 12.35%
```

15.10 Split Charts: Subplot the Volume

Price and Volume Subplots

```
def plot_price_and_volume(symbol, start_date, end_date):
    """
    Create subplots showing price and volume

    Parameters:
    symbol (str): Stock ticker
    start_date (datetime): Start date
    end_date (datetime): End date
    """
    data = stock_load(symbol, start_date, end_date)

    if data is None:
        return

    # Create figure with 2 subplots
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(14, 10),
                                   sharex=True,
                                   gridspec_kw={'height_ratios': [2, 1]})

    # Plot 1: Price
    ax1.plot(data.index, data['Close'], color='blue', linewidth=2)
    ax1.set_ylabel('Price ($)', fontsize=12, fontweight='bold')
    ax1.set_title(f'{symbol} Stock Price and Volume',
                  fontsize=16, fontweight='bold')
    ax1.grid(True, alpha=0.3)

    # Plot 2: Volume
    ax2.bar(data.index, data['Volume'], color='steelblue', alpha=0.6)
    ax2.set_xlabel('Date', fontsize=12, fontweight='bold')
    ax2.set_ylabel('Volume', fontsize=12, fontweight='bold')
    ax2.grid(True, alpha=0.3, axis='y')

    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

# Example usage
plot_price_and_volume('AAPL', dt.datetime(2024, 1, 1), dt.datetime(2024, 6, 1))
```

Description of Output:
Two vertically stacked charts sharing the same x-axis (dates). The top chart (twice the height of the bottom) shows the closing price, while the bottom chart displays trading volume as bars. This layout is standard in financial analysis[7].

Enhanced Subplot with Color-Coded Volume

```
def plot_price_volume_enhanced(symbol, start_date, end_date):
    """
    Enhanced price and volume chart with color-coded volume bars

    Parameters:
    symbol (str): Stock ticker
    start_date (datetime): Start date
    end_date (datetime): End date
    """
    data = stock_load(symbol, start_date, end_date)

    if data is None:
        return

    # Calculate daily price change
    data['Price_Change'] = data['Close'].diff()

    # Create figure
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(14, 10),
                                   sharex=True,
                                   gridspec_kw={'height_ratios': [2, 1]})

    # Plot 1: Price with fill
    ax1.plot(data.index, data['Close'], color='blue', linewidth=2, label='Close')
    ax1.fill_between(data.index, data['Close'], alpha=0.1, color='blue')
    ax1.set_ylabel('Price ($)', fontsize=12, fontweight='bold')
    ax1.set_title(f'{symbol} Stock Analysis', fontsize=16, fontweight='bold')
    ax1.legend(loc='upper left')
    ax1.grid(True, alpha=0.3)

    # Plot 2: Volume (green for up days, red for down days)
    colors = ['green' if x &gt; 0 else 'red' for x in data['Price_Change']]
    ax2.bar(data.index, data['Volume'], color=colors, alpha=0.5)
    ax2.set_xlabel('Date', fontsize=12, fontweight='bold')
    ax2.set_ylabel('Volume', fontsize=12, fontweight='bold')
    ax2.grid(True, alpha=0.3, axis='y')

    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

# Example usage
plot_price_volume_enhanced('GOOGL', dt.datetime(2024, 1, 1), dt.datetime(2024, 6, 1))
```

Description of Output:
Similar to the previous subplot, but with enhancements: the price chart has a light blue fill beneath the line, and volume bars are color-coded (green for days when price increased, red for days when price decreased). This helps identify volume patterns related to price movements[7].

Three-Panel Dashboard

```
def plot_dashboard(symbol, start_date, end_date):
    """
    Create a three-panel dashboard: Price, Volume, and Returns

    Parameters:
    symbol (str): Stock ticker
    start_date (datetime): Start date
    end_date (datetime): End date
    """
    data = stock_load(symbol, start_date, end_date)

    if data is None:
        return

    # Calculate returns
    data['Returns'] = data['Close'].pct_change() * 100
```

```
    # Create figure with 3 subplots
    fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(14, 12),
                                        sharex=True,
                                        gridspec_kw={'height_ratios': [2, 1, 1]})

    # Panel 1: Price
    ax1.plot(data.index, data['Close'], color='blue', linewidth=2)
    ax1.set_ylabel('Price ($)', fontsize=11, fontweight='bold')
    ax1.set_title(f'{symbol} Stock Dashboard', fontsize=16, fontweight='bold')
    ax1.grid(True, alpha=0.3)

    # Panel 2: Volume
    ax2.bar(data.index, data['Volume'], color='steelblue', alpha=0.6)
    ax2.set_ylabel('Volume', fontsize=11, fontweight='bold')
    ax2.grid(True, alpha=0.3, axis='y')

    # Panel 3: Daily Returns
    colors = ['green' if x &gt; 0 else 'red' for x in data['Returns']]
    ax3.bar(data.index, data['Returns'], color=colors, alpha=0.6)
    ax3.set_ylabel('Return (%)', fontsize=11, fontweight='bold')
    ax3.set_xlabel('Date', fontsize=12, fontweight='bold')
    ax3.axhline(y=0, color='black', linestyle='-', linewidth=0.8)
    ax3.grid(True, alpha=0.3, axis='y')

    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

# Example usage
plot_dashboard('MSFT', dt.datetime(2024, 1, 1), dt.datetime(2024, 6, 1))
```

Description of Output:
A comprehensive three-panel dashboard showing price (top), volume (middle), and daily returns (bottom). This provides a complete view of the stock's performance in a single visualization.

15.11 Adding a Moving-Average Line

What is a Moving Average?

A **moving average** smooths out price data by creating a constantly updated average price over a specific time period. It helps identify trends by filtering out short-term fluctuations[3] [8] [9].

Simple Moving Average (SMA)

```
def plot_with_sma(symbol, start_date, end_date, window=20):
    """
    Plot stock price with Simple Moving Average

    Parameters:
    symbol (str): Stock ticker
    start_date (datetime): Start date
    end_date (datetime): End date
    window (int): Moving average window size (default: 20 days)
    """
    data = stock_load(symbol, start_date, end_date)

    if data is None:
        return

    # Calculate Simple Moving Average
    data['SMA'] = data['Close'].rolling(window=window).mean()

    # Create plot
    plt.figure(figsize=(14, 7))
    plt.plot(data.index, data['Close'],
             label='Close Price', linewidth=2, color='blue')
```

```
    plt.plot(data.index, data['SMA'],
             label=f'{window}-Day SMA', linewidth=2,
             color='red', linestyle='--')

    plt.title(f'{symbol} with {window}-Day Moving Average',
              fontsize=16, fontweight='bold')
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Price ($)', fontsize=12)
    plt.legend(loc='best', fontsize=11)
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

# Example usage
plot_with_sma('AAPL', dt.datetime(2024, 1, 1), dt.datetime(2024, 10, 1), window=20)
```

Description of Output:

Shows the stock price as a blue solid line and the 20-day moving average as a red dashed line. The moving average line is smoother and helps identify the overall trend direction[8] [9].

Multiple Moving Averages

```
def plot_multiple_sma(symbol, start_date, end_date, windows=[20, 50, 200]):
    """
    Plot stock price with multiple moving averages

    Parameters:
    symbol (str): Stock ticker
    start_date (datetime): Start date
    end_date (datetime): End date
    windows (list): List of MA window sizes
    """
    data = stock_load(symbol, start_date, end_date)

    if data is None:
        return

    # Calculate multiple moving averages
    colors = ['orange', 'green', 'red']

    plt.figure(figsize=(14, 7))
    plt.plot(data.index, data['Close'],
             label='Close Price', linewidth=2, color='blue', alpha=0.7)

    for window, color in zip(windows, colors):
        data[f'SMA_{window}'] = data['Close'].rolling(window=window).mean()
        plt.plot(data.index, data[f'SMA_{window}'],
                 label=f'{window}-Day SMA', linewidth=2,
                 color=color, linestyle='--')

    plt.title(f'{symbol} with Multiple Moving Averages',
              fontsize=16, fontweight='bold')
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Price ($)', fontsize=12)
    plt.legend(loc='best', fontsize=10)
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

# Example usage
plot_multiple_sma('AAPL', dt.datetime(2023, 1, 1), dt.datetime(2024, 10, 1))
```

Description of Output:

Displays the stock price with three moving averages: 20-day (short-term, orange), 50-day (medium-term, green), and 200-day (long-term, red). Traders often use these three periods to identify different trend timeframes[8] [9].

Exponential Moving Average (EMA)

```
def plot_sma_vs_ema(symbol, start_date, end_date, window=20):
    """
    Compare Simple Moving Average with Exponential Moving Average

    Parameters:
    symbol (str): Stock ticker
    start_date (datetime): Start date
    end_date (datetime): End date
    window (int): Window size for both averages
    """
    data = stock_load(symbol, start_date, end_date)

    if data is None:
        return

    # Calculate both types of moving averages
    data['SMA'] = data['Close'].rolling(window=window).mean()
    data['EMA'] = data['Close'].ewm(span=window, adjust=False).mean()

    # Create plot
    plt.figure(figsize=(14, 7))
    plt.plot(data.index, data['Close'],
             label='Close Price', linewidth=1.5, color='blue', alpha=0.7)
    plt.plot(data.index, data['SMA'],
             label=f'{window}-Day SMA', linewidth=2,
             color='red', linestyle='--')
    plt.plot(data.index, data['EMA'],
             label=f'{window}-Day EMA', linewidth=2,
             color='green', linestyle='-.')

    plt.title(f'{symbol}: SMA vs EMA Comparison',
              fontsize=16, fontweight='bold')
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Price ($)', fontsize=12)
    plt.legend(loc='best', fontsize=11)
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

# Example usage
plot_sma_vs_ema('MSFT', dt.datetime(2024, 1, 1), dt.datetime(2024, 10, 1), window=20)
```

Description of Output:
Compares Simple Moving Average (SMA) with Exponential Moving Average (EMA). The EMA gives more weight to recent prices, so it responds more quickly to price changes than the SMA [3] [8].

Moving Average with Price and Volume

```
def plot_ma_price_volume(symbol, start_date, end_date, window=50):
    """
    Plot price with moving average and volume subplot

    Parameters:
    symbol (str): Stock ticker
    start_date (datetime): Start date
    end_date (datetime): End date
    window (int): MA window size
    """
    data = stock_load(symbol, start_date, end_date)

    if data is None:
        return

    # Calculate moving average
    data['SMA'] = data['Close'].rolling(window=window).mean()

    # Create subplots
```

```
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(14, 10),
                                   sharex=True,
                                   gridspec_kw={'height_ratios': [2, 1]})

    # Price and MA plot
    ax1.plot(data.index, data['Close'],
            label='Close Price', linewidth=2, color='blue')
    ax1.plot(data.index, data['SMA'],
            label=f'{window}-Day MA', linewidth=2,
            color='red', linestyle='--')
    ax1.set_ylabel('Price ($)', fontsize=12, fontweight='bold')
    ax1.set_title(f'{symbol} Price with Moving Average',
                  fontsize=16, fontweight='bold')
    ax1.legend(loc='upper left', fontsize=11)
    ax1.grid(True, alpha=0.3)

    # Volume plot
    ax2.bar(data.index, data['Volume'], color='steelblue', alpha=0.6)
    ax2.set_xlabel('Date', fontsize=12, fontweight='bold')
    ax2.set_ylabel('Volume', fontsize=12, fontweight='bold')
    ax2.grid(True, alpha=0.3, axis='y')

    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

# Example usage
plot_ma_price_volume('GOOGL', dt.datetime(2024, 1, 1), dt.datetime(2024, 10, 1), window=50)
```

Description of Output:
Combines moving average analysis with volume information. The top panel shows price and the 50-day moving average, while the bottom shows trading volume, providing a complete technical analysis view.

15.12 Giving Choices to the User

Interactive Stock Selection

```
def interactive_plot():
    """
    Interactive function that accepts user input for stock analysis
    """
    print("=" * 60)
    print("Stock Price Visualization Tool")
    print("=" * 60)

    # Get user input
    symbol = input("Enter stock symbol (e.g., AAPL, MSFT, GOOGL): ").upper()

    # Date range selection
    print("\nSelect time period:")
    print("1. Last 30 days")
    print("2. Last 90 days")
    print("3. Last 6 months")
    print("4. Last 1 year")
    print("5. Custom range")

    choice = input("Enter your choice (1-5): ")

    end_date = dt.datetime.now()

    if choice == '1':
        start_date = end_date - dt.timedelta(days=30)
    elif choice == '2':
        start_date = end_date - dt.timedelta(days=90)
    elif choice == '3':
        start_date = end_date - dt.timedelta(days=180)
    elif choice == '4':
        start_date = end_date - dt.timedelta(days=365)
```

```
    elif choice == '5':
        start_str = input("Enter start date (YYYY-MM-DD): ")
        end_str = input("Enter end date (YYYY-MM-DD): ")
        start_date = dt.datetime.strptime(start_str, '%Y-%m-%d')
        end_date = dt.datetime.strptime(end_str, '%Y-%m-%d')
    else:
        print("Invalid choice. Using last 90 days.")
        start_date = end_date - dt.timedelta(days=90)

    # Moving average option
    use_ma = input("\nAdd moving average? (y/n): ").lower()

    if use_ma == 'y':
        ma_window = int(input("Enter MA window size (e.g., 20, 50, 200): "))
    else:
        ma_window = None

    # Load and plot data
    print(f"\nLoading data for {symbol}...")
    data = stock_load(symbol, start_date, end_date)

    if data is None:
        print("Failed to load data. Please check the symbol and try again.")
        return

    # Create plot
    plt.figure(figsize=(14, 7))
    plt.plot(data.index, data['Close'],
             label='Close Price', linewidth=2, color='blue')

    if ma_window:
        data['MA'] = data['Close'].rolling(window=ma_window).mean()
        plt.plot(data.index, data['MA'],
                 label=f'{ma_window}-Day MA', linewidth=2,
                 color='red', linestyle='--')

    plt.title(f'{symbol} Stock Price Analysis', fontsize=16, fontweight='bold')
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Price ($)', fontsize=12)
    plt.legend(loc='best', fontsize=11)
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

    # Print summary statistics
    print("\n" + "=" * 60)
    print("Summary Statistics")
    print("=" * 60)
    print(f"Symbol: {symbol}")
    print(f"Period: {start_date.date()} to {end_date.date()}")
    print(f"Starting Price: ${data['Close'].iloc[^0]:.2f}")
    print(f"Ending Price: ${data['Close'].iloc[-1]:.2f}")
    print(f"Highest Price: ${data['Close'].max():.2f}")
    print(f"Lowest Price: ${data['Close'].min():.2f}")
    print(f"Average Price: ${data['Close'].mean():.2f}")
    change = data['Close'].iloc[-1] - data['Close'].iloc[^0]
    pct_change = (change / data['Close'].iloc[^0]) * 100
    print(f"Price Change: ${change:.2f} ({pct_change:+.2f}%)")
    print("=" * 60)

# Run the interactive function
# interactive_plot()
```

Example Output:

```
============================================================
Stock Price Visualization Tool
============================================================
Enter stock symbol (e.g., AAPL, MSFT, GOOGL): AAPL

Select time period:
```

```
1. Last 30 days
2. Last 90 days
3. Last 6 months
4. Last 1 year
5. Custom range
Enter your choice (1-5): 2

Add moving average? (y/n): y
Enter MA window size (e.g., 20, 50, 200): 20

Loading data for AAPL...

============================================================
Summary Statistics
============================================================
Symbol: AAPL
Period: 2024-07-23 to 2024-10-22
Starting Price: $218.54
Ending Price: $230.10
Highest Price: $237.23
Lowest Price: $207.23
Average Price: $224.36
Price Change: $11.56 (+5.29%)
============================================================
```

Multi-Stock Comparison Tool

```
def compare_stocks_interactive():
    """
    Interactive tool to compare multiple stocks
    """
    print("=" * 60)
    print("Stock Comparison Tool")
    print("=" * 60)

    # Get stock symbols
    symbols_input = input("Enter stock symbols separated by commas (e.g., AAPL,MSFT,GOOGL): ")
    symbols = [s.strip().upper() for s in symbols_input.split(',')]

    # Time period
    days = int(input("Enter number of days to analyze: "))
    end_date = dt.datetime.now()
    start_date = end_date - dt.timedelta(days=days)

    # Normalization option
    normalize = input("Normalize prices for comparison? (y/n): ").lower() == 'y'

    # Load data and create plot
    plt.figure(figsize=(14, 7))
    colors = ['blue', 'red', 'green', 'orange', 'purple']

    print(f"\nLoading data for {len(symbols)} stocks...")

    for i, symbol in enumerate(symbols):
        data = stock_load(symbol, start_date, end_date)

        if data is not None:
            if normalize:
                # Normalize to base 100
                price = (data['Close'] / data['Close'].iloc[^0]) * 100
                ylabel = 'Normalized Price (Base = 100)'
                title_suffix = '(Normalized)'
            else:
                price = data['Close']
                ylabel = 'Price ($)'
                title_suffix = ''

            plt.plot(data.index, price,
                    label=symbol, linewidth=2,
                    color=colors[i % len(colors)])
```

```
        else:
            print(f"Warning: Could not load data for {symbol}")

    plt.title(f'Stock Comparison {title_suffix}', fontsize=16, fontweight='bold')
    plt.xlabel('Date', fontsize=12)
    plt.ylabel(ylabel, fontsize=12)
    plt.legend(loc='best', fontsize=11)
    plt.grid(True, alpha=0.3)

    if normalize:
        plt.axhline(y=100, color='black', linestyle='--', alpha=0.5)

    plt.tight_layout()
    plt.show()

    print("\nComparison complete!")

# Run the comparison tool
# compare_stocks_interactive()
```

Example Output:

```
============================================================
Stock Comparison Tool
============================================================
Enter stock symbols separated by commas (e.g., AAPL,MSFT,GOOGL): AAPL,MSFT,GOOGL
Enter number of days to analyze: 180
Normalize prices for comparison? (y/n): y

Loading data for 3 stocks...

Comparison complete!
```

Advanced Analysis Menu

```
def stock_analysis_menu():
    """
    Comprehensive menu-driven stock analysis tool
    """
    while True:
        print("\n" + "=" * 60)
        print("Stock Analysis Menu")
        print("=" * 60)
        print("1. Plot stock price")
        print("2. Plot with moving average")
        print("3. Plot price and volume")
        print("4. Compare multiple stocks")
        print("5. Show daily returns")
        print("6. Full dashboard (price, volume, returns)")
        print("7. Exit")
        print("=" * 60)

        choice = input("Select option (1-7): ")

        if choice == '7':
            print("Exiting... Thank you!")
            break

        # Get common inputs
        symbol = input("Enter stock symbol: ").upper()
        days = int(input("Enter number of days: "))

        end_date = dt.datetime.now()
        start_date = end_date - dt.timedelta(days=days)

        # Execute based on choice
        if choice == '1':
            makeplot(symbol, start_date, end_date)
```

```
        elif choice == '2':
            window = int(input("Enter MA window (e.g., 20, 50): "))
            plot_with_sma(symbol, start_date, end_date, window)

        elif choice == '3':
            plot_price_and_volume(symbol, start_date, end_date)

        elif choice == '4':
            symbol2 = input("Enter second stock symbol: ").upper()
            plot_two_stocks(symbol, symbol2, start_date, end_date)

        elif choice == '5':
            plot_daily_returns(symbol, start_date, end_date)

        elif choice == '6':
            plot_dashboard(symbol, start_date, end_date)

        else:
            print("Invalid choice. Please try again.")

# Run the menu system
# stock_analysis_menu()
```

Example Interaction:

```
============================================================
Stock Analysis Menu
============================================================
1. Plot stock price
2. Plot with moving average
3. Plot price and volume
4. Compare multiple stocks
5. Show daily returns
6. Full dashboard (price, volume, returns)
7. Exit
============================================================
Select option (1-7): 2
Enter stock symbol: MSFT
Enter number of days: 90
Enter MA window (e.g., 20, 50): 20
```

Chapter Summary

This comprehensive tutorial has covered all aspects of retrieving and visualizing financial data using Python:

Key Concepts Learned

1. **Pandas Fundamentals**: Understanding Series and DataFrames for data manipulation [2] [3]
2. **Data Retrieval**: Using pandas_datareader to fetch stock data from Yahoo Finance [2] [4] [10]
3. **Visualization**: Creating professional charts with matplotlib including:
    - Line charts for price data
    - Bar charts for volume
    - Subplots for multiple data series
    - Color-coded visualizations
4. **Technical Analysis**: Implementing moving averages and other indicators [3] [8] [9]
5. **User Interaction**: Building interactive tools with input validation

Best Practices

- **Error Handling**: Always check if data loads successfully before plotting
- **Code Reusability**: Create functions for repeated tasks
- **Clear Visualization**: Use titles, labels, legends, and appropriate colors
- **Date Management**: Use datetime objects for flexible date handling
- **Normalization**: When comparing stocks with different price levels

Next Steps

To extend your learning:

1. Explore other technical indicators (RSI, MACD, Bollinger Bands)
2. Implement candlestick charts for detailed price analysis [11]
3. Add statistical analysis (correlation, volatility)
4. Create automated reporting systems
5. Build web-based dashboards using libraries like Dash or Streamlit

Additional Resources

- **pandas documentation**: https://pandas.pydata.org/
- **matplotlib gallery**: https://matplotlib.org/stable/gallery/
- **pandas-datareader**: https://pandas-datareader.readthedocs.io/
- **Yahoo Finance**: Free historical stock data source

Practice Exercises

Exercise 1: Basic Plotting

Create a program that plots the closing price of Tesla (TSLA) for the last 6 months with a 30-day moving average.

Exercise 2: Multi-Stock Analysis

Write a function that takes a list of tech stocks (AAPL, MSFT, GOOGL, AMZN) and creates a normalized comparison chart.

Exercise 3: Volume Analysis

Create a visualization that identifies and highlights the top 5 highest volume trading days in a given period.

Exercise 4: Return Calculator

Build a tool that calculates and displays:

- Daily returns
- Weekly returns
- Monthly returns
- Cumulative returns

Exercise 5: Interactive Dashboard

Create a comprehensive interactive program that combines all features learned in this chapter with a user-friendly menu system.

Conclusion

You have now completed a comprehensive tutorial on retrieving and visualizing financial data using Python. The skills learned in this chapter form the foundation for financial data analysis, algorithmic trading, and investment research. Practice these techniques with different stocks and time periods to become proficient in financial data visualization.

Remember: The key to mastery is practice. Experiment with the code examples, modify them, and create your own variations. Happy coding!

End of Tutorial
[12] [13] [14] [15] [16] [17] [18] [19] [20] [21]

⁂

1. CHAPTER-15.pdf
2. https://dev.to/shahstavan/how-to-fetch-stock-data-using-api-in-python-5e15
3. https://www.geeksforgeeks.org/pandas/how-to-calculate-moving-average-in-a-pandas-dataframe/
4. https://stackoverflow.com/questions/49604224/pulling-stock-information-using-pandas-datareader
5. https://dev.to/dm8ry/python-code-that-generates-a-stock-price-chart-for-the-last-n-days-g8a
6. https://www.youtube.com/watch?v=GZRozGYOp1o
7. https://llego.dev/posts/matplotlib-financial-data-visualization/
8. https://favtutor.com/articles/moving-average-pandas/
9. https://aleksandarhaber.com/4-simple-moving-averages-of-stock-time-series-in-pandas-and-python/
10. https://pythonandvba.com/blog/how-to-get-stock-data-export-it-to-excel-using-python/
11. https://www.geeksforgeeks.org/python/how-to-create-a-candlestick-chart-in-matplotlib/
12. https://www.reddit.com/r/Python/comments/ut6xxu/using_pandas_datareader_to_collect_free/
13. https://www.youtube.com/watch?v=FYji-5S8D2s
14. https://www.youtube.com/watch?v=t_vZDyQDUkk
15. https://pythonprogramming.net/last-price-stock-chart-matplotlib-tutorial/
16. https://stackoverflow.com/questions/40060842/moving-average-pandas
17. https://pandas-datareader.readthedocs.io/en/latest/remote_data.html
18. https://www.datacamp.com/tutorial/matplotlib-tutorial-python
19. https://www.datacamp.com/tutorial/moving-averages-in-pandas
20. https://www.geeksforgeeks.org/python/how-to-download-historical-stock-prices-in-python/
21. https://matplotlib.org/stable/gallery/showcase/stock_prices.html