Python Advanced Assignment-25

Q1. What is the distinction between a NumPy array and a pandas DataFrame? Is there a way to convert between the two if there is?

Ans.

- NumPy Array: It is a homogenous collection of elements, meaning all elements must be of the same data type. It's highly efficient for numerical computations.

- pandas DataFrame: A DataFrame is a 2D table-like structure that can hold data of different types (e.g., integers, floats, strings). It provides labeled axes (rows and columns) and supports more complex operations like data manipulation and analysis.

Conversion:

- You can convert a NumPy array to a pandas DataFrame using `pd.DataFrame(array)`.

- You can convert a pandas DataFrame to a NumPy array using `df.to_numpy()` or `df.values`.

Q2. What can go wrong when a user enters a stock-ticker symbol, and how do you handle it?

Ans.

Several issues can arise:

1. Invalid ticker symbol: The user might enter a symbol that doesn't exist or is delisted.

- Handling: Use error handling (try-except) to catch invalid ticker exceptions. You can also validate the symbol using external APIs.

2. Case-sensitivity: Ticker symbols are usually uppercase, and the user might enter lowercase.

- Handling: Convert the input to uppercase before processing it (`ticker.upper()`).

3. Network or API errors: Issues with the data source could arise, leading to failure in fetching data.

- Handling: Implement retries and fallback options, along with informative error messages for the user.

4. Typos or malformed input: Users might enter symbols with invalid characters.

- Handling: Use regular expressions or predefined stock symbol formats to validate input before processing.

Q3. Identify some of the plotting techniques that are used to produce a stock-market chart.

Ans.

Common plotting techniques used in stock market charts include:

1. Line charts: Show the closing price over time, providing a clear trend view.

2. Candlestick charts: Represent the opening, closing, high, and low prices of a stock for each time interval.

3. Bar charts: Similar to candlestick charts but often use solid bars to depict the price range.

4. Moving averages: Plot lines representing different moving averages (e.g., 50-day, 200-day) to smooth out price fluctuations and show trends.

5. Volume bars: Plotted below the main price chart to show trading volume over time.

Q4. Why is it essential to print a legend on a stock market chart?

Ans.

A legend is essential in stock market charts because:

- It clarifies what each line or symbol represents, especially when displaying multiple metrics (e.g., price, moving averages, volume).

- It helps users quickly identify different data points without confusion, enhancing readability and interpretation.

- Legends are particularly important when comparing several stocks or indices on the same chart.

Q5. What is the best way to limit the length of a pandas DataFrame to less than a year?

Ans.

To limit the length of a pandas DataFrame to less than a year, you can filter the DataFrame by a date range. For example:

```
# Assuming 'date' is a column with datetime objects in the DataFrame
df_filtered = df[df['date'] >= '2023-01-01'] & df[df['date'] < '2024-01-01']
```

This will filter the DataFrame to only include data from 2023. You can adjust the dates to fit your specific time frame, such as limiting it to the last 180 days or 6 months.

Q6. What is the definition of a 180-day moving average?

Ans.

A 180-day moving average is a technical indicator that smooths out price data by calculating the average closing price of a stock over the past 180 days. It helps identify long-term trends by reducing the noise from short-term price fluctuations.

The moving average for each day is recalculated as the average of the previous 180 days, and the results are plotted over time. It's often used by traders to gauge overall market direction.

Q7. Did the chapter's final example use "indirect" importing? If so, how exactly do you do it?

Ans.

Yes, the chapter's final example likely used "indirect" importing, which involves importing a module inside a function or a specific part of the code rather than at the top of the file. This is done to avoid circular dependencies or to delay the import until it's needed.

For example:

```
def some_function():
    import module_name  # Module is imported only when this function is called
    module_name.some_method()
```

This approach is often used to improve performance (by importing only when needed) or to avoid issues with circular imports.