Advanced Pandas Coding Questions

1. Given a DataFrame with `Customer_ID`, `Order_Date`, and `Order_Value`, compute the customer lifetime value (CLV) for each `Customer_ID` by summing up all `Order_Value` per customer.

2. Use `merge_asof` to merge two DataFrames `df1` and `df2` on a sorted `Date` column, performing a merge that allows for approximate matches.

3. Create a DataFrame with hierarchical indexes (`Country`, `State`, `City`). Aggregate data at the `Country` level and then at the `State` level using `groupby`.

4. Optimize a DataFrame with 1 million rows by using memory-efficient data types and methods for reducing memory usage.

5. Perform a rolling window calculation to compute the exponential moving average of the `Stock_Price` column with a span of 30 days.

6. Use `pd.DataFrame.query()` to filter rows where the `Amount` column is greater than the median of the column.

7. Create a DataFrame with `Employee_ID`, `Join_Date`, and `Leave_Date`. Calculate the tenure of each employee in months using `pd.DateOffset`.

8. Transform a DataFrame with a `Date` column into multiple DataFrames based on year and month, and then perform a calculation on each subset.

9. Use `applymap` to apply a custom function to every element in a DataFrame and demonstrate its impact.

10. Given a DataFrame with `Transaction_ID` and `Item_Price`, use `pd.cut` to bin `Item_Price` into discrete intervals and analyze the distribution of items in these bins.

11. Implement a function to detect and flag outliers in a DataFrame based on the Interquartile Range (IQR) method and add a new column indicating the outlier status.

12. Create a DataFrame with `User_ID`, `Event`, and `Timestamp`. Calculate the time spent between consecutive events for each user.

13. Use `pd.pivot_table` to create a pivot table with `Department`, `Month`, and `Salary` columns, showing the average salary by department and month.

14. Apply a custom aggregation function to a `groupby` operation that calculates both the mean and standard deviation of the `Sales` column for each `Region`.

15. Perform a time-series decomposition on a DataFrame with `Date` and `Sales` columns to extract trend, seasonal, and residual components.

16. Use `dask.dataframe` to handle and perform operations on a large dataset that cannot fit into memory. Demonstrate a computation like groupby or aggregation.

17. Optimize a DataFrame operation by using `numba` to accelerate a custom function applied to each row or column.

18. Create a DataFrame from a nested JSON file and flatten it into a tabular format with Pandas.

19. Perform an outlier detection using the Isolation Forest algorithm on a DataFrame with multiple features and visualize the results.

20. Implement a multi-index DataFrame with levels `Region`, `Country`, and `City`. Perform a slicing operation to extract data for a specific region and city.