Facebook Metrics Data Analysis - Oral Exam Preparation Guide

Code Explanation

Data Loading and Initial Exploration

```python
import pandas as pd
df = pd.read_csv('dataset_Facebook.csv', delimiter=';')
df
df.describe()
```

- The first line imports the pandas library with the standard alias `pd`
- The second line reads a CSV file with semicolon delimiter into a DataFrame called `df`
- The third line displays the entire DataFrame to see its contents
- The fourth line generates descriptive statistics (count, mean, std, min, percentiles, max) of the numerical columns

Creating Data Subsets

```python
# First subset: Like and Share
df_subset_1 = df[['like','share']]
df_subset_1

# Second subset: Comment and Type
df_subset_2 = df[['comment','Type']]
df_subset_2
```

- The first subset extracts only the 'like' and 'share' columns
- The second subset extracts only the 'comment' and 'Type' columns
- This demonstrates column-based subsetting using double square brackets

Merging Data

```python
merged_data = pd.merge(df_subset_2, df_subset_1, left_on='comment', right_on='like')
merged_data
```

- Merges the two previously created subsets
- Uses the 'comment' column from `df_subset_2` and 'like' column from `df_subset_1` as the joining keys
- Records are joined where the values in these columns match

Sorting Data

```python
# Sorting merged_data in descending order wrt 'Like'
merged_data.sort_values(by=['like'], ascending=False)
```

- Sorts the merged DataFrame based on the 'like' column
- Uses `ascending=False` to sort in descending order (highest values first)

Transposing Data

```python
# Method 1
merged_data.transpose()
# Method 2
merged_data.T
```

- Both methods convert rows to columns and columns to rows
- `.transpose()` is the full method name, while `.T` is a shorthand property

Shape and Reshape Data

```python
df.Type.unique()
# Reshape
# Comment is id_vars and Type is value_vars
pd.melt(df, id_vars=['Type'], value_vars=['comment'])
```

- `df.Type.unique()` returns the unique values in the 'Type' column
- `pd.melt()` transforms the data from wide to long format
- 'Type' column is kept as an identifier variable
- 'comment' column is "melted" into rows

Potential Oral Exam Questions and Answers

1. What does `df.describe()` calculate and why is it useful?

Answer: `df.describe()` generates descriptive statistics for numerical columns in the DataFrame, including count, mean, standard deviation, minimum, 25th percentile, median (50th percentile), 75th percentile, and maximum. It's useful for getting a quick statistical summary of your data, identifying outliers, understanding data distribution, and checking for missing values.

2. What is the difference between single square brackets and double square brackets when subsetting data in pandas?

Answer: Single square brackets (`df['column']`) return a pandas Series object for a single column. Double square brackets (`df[['column1', 'column2']]`) return a pandas DataFrame containing the specified columns. Double brackets are used when you want to select multiple columns or ensure the result remains a DataFrame even when selecting a single column.

3. Explain the `pd.merge()` function used in your code. What parameters did you use and why?

Answer: The `pd.merge()` function combines two DataFrames based on common keys. In this code, I used:

- `df_subset_2` and `df_subset_1` as the DataFrames to merge
- `left_on='comment'` specifies the key column in the left DataFrame (df_subset_2)
- `right_on='like'` specifies the key column in the right DataFrame (df_subset_1)

This performs an inner join by default, keeping only rows where the 'comment' values from df_subset_2 match the 'like' values from df_subset_1.

4. What are other join types available in pandas merge, and when would you use them?

Answer: Pandas supports several join types:

- Inner join (default): Returns only matching rows from both DataFrames
- Left join (`how='left'`): Returns all rows from the left DataFrame and matching rows from the right

- Right join (`how='right'`): Returns all rows from the right DataFrame and matching rows from the left
- Outer join (`how='outer'`): Returns all rows from both DataFrames

You would choose based on what data you need to preserve. Left joins are common when you want to keep all records from a primary table while adding information from a secondary table.

5. How does `sort_values()` work, and what other parameters could you use with it?

Answer: `sort_values()` sorts a DataFrame by specified columns. In the code, we sorted by the 'like' column in descending order. Other useful parameters include:

- `by`: Columns to sort by (can be a list for multi-level sorting)
- `ascending`: Whether to sort ascending (True) or descending (False), can be a list for different directions per column
- `inplace`: Whether to modify the original DataFrame (True) or return a new one (False)
- `na_position`: Where to place NaN values ('first' or 'last')
- `ignore_index`: Whether to reset the index after sorting

6. What's the difference between `transpose()` and `.T` in pandas?

Answer: There is no functional difference. `.T` is simply a shorthand property that calls the `transpose()` method. Both convert rows to columns and columns to rows. The choice between them is stylistic; `.T` is more concise while `transpose()` is more explicit.

7. What does `df.Type.unique()` return and why is it useful?

Answer: `df.Type.unique()` returns an array containing all unique values in the 'Type' column without duplicates. It's useful for understanding the categorical values in a column, checking data quality, preparing for one-hot encoding, or planning how to process different categories.

8. Explain the `pd.melt()` function. When and why would you use it?

Answer: `pd.melt()` reshapes data from wide format to long format. It takes columns and converts them into rows. In the code, we kept 'Type' as an identifier column and "melted" the 'comment' column.

This is used when:

- Converting from wide to long format for visualization libraries (like seaborn)
- Preparing data for certain statistical analyses that require long format
- Unpivoting previously pivoted data
- Normalizing denormalized data structures

9. What's the difference between `id_vars` and `value_vars` in `pd.melt()`?

Answer: In `pd.melt()`:

- `id_vars`: These columns are kept as is and not transformed
- `value_vars`: These columns are "melted" into two columns: a 'variable' column (containing the original column names) and a 'value' column (containing the values)

In our example, 'Type' is kept as an identifier, while 'comment' values are transformed into rows.

10. What other data subsetting techniques could you have used besides selecting columns?

Answer: Several other subsetting techniques include:

- Boolean indexing: `df[df['like'] > 100]` to filter rows by condition
- `.loc[]`: Label-based indexing for both rows and columns, e.g., `df.loc[0:5, 'like':'share']`

- `.iloc[]`: Integer position-based indexing, e.g., `df.iloc[0:5, 2:4]`
- `.query()`: SQL-like filtering, e.g., `df.query('like > 100 & share < 50')`
- `.filter()`: Selecting columns based on regex patterns or other criteria

11. What is the purpose of creating data subsets in data analysis?

Answer: Creating data subsets serves several purposes:

- Focusing on relevant variables for specific analyses
- Reducing memory usage when working with large datasets
- Simplifying complex data structures
- Creating separate datasets for different analyses or visualizations
- Extracting training and testing sets for machine learning
- Improving readability and maintainability of code

12. Why would you transpose a DataFrame in data analysis?

Answer: Transposing a DataFrame is useful for:

- Changing the perspective of the data for specific visualizations
- Making wide data more readable when there are many columns
- Preparing data for certain statistical methods that expect features as rows
- Computing correlations between rows instead of columns
- Creating summary tables where categories should be columns
- Restructuring data for export to specific file formats

13. What is the dataset_Facebook.csv likely to contain, and what kind of analysis might you perform with it?

Answer: Based on the column names ('like', 'share', 'comment', 'Type'), the dataset likely contains metrics from Facebook posts or pages. Possible analyses include:

- Engagement analysis: How likes, comments, and shares correlate
- Content performance by post type
- Identifying the most engaging content types
- Time series analysis of engagement metrics
- Anomaly detection for viral posts
- Predicting engagement based on post characteristics
- Comparing performance across different content categories

14. How would you check for and handle missing values in this dataset?

Answer: I could check for missing values using:

```python
df.isnull().sum()  # Count missing values per column
```

To handle missing values, I might:

- Fill with mean/median: `df['like'].fillna(df['like'].mean())`
- Fill with mode: `df['Type'].fillna(df['Type'].mode()[0])`
- Forward/backward fill: `df.fillna(method='ffill')`
- Drop rows: `df.dropna()`
- Interpolate: `df.interpolate()`

The choice depends on the nature of the data and the purpose of the analysis.

15. What other merge strategies could be used with these datasets?

Answer: Other merge strategies include:

- Merging on index: `pd.merge(df1, df2, left_index=True, right_index=True)`
- Merging with suffixes: `pd.merge(df1, df2, on='common_col', suffixes=('_left', '_right'))`
- Using different join types: `pd.merge(df1, df2, on='key', how='outer')`
- Using `pd.concat()` for appending DataFrames: `pd.concat([df1, df2], axis=0)`
- Using `df1.join(df2)` for index-based joining

16. How would you visualize the relationship between 'like' and 'share' columns?

Answer: I could create visualizations using matplotlib or seaborn:

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='like', y='share')
plt.title('Relationship between Likes and Shares')
plt.xlabel('Number of Likes')
plt.ylabel('Number of Shares')
plt.show()

# Or a regression plot
sns.regplot(data=df, x='like', y='share')

# Or a heatmap of correlation
correlation = df[['like', 'share', 'comment']].corr()
sns.heatmap(correlation, annot=True)
```

17. Why did you use a semicolon as a delimiter in the `read_csv()` function?

Answer: The delimiter parameter in `read_csv()` specifies the character used to separate fields in the CSV file. I used a semicolon (`delimiter=';'`) because the dataset_Facebook.csv file apparently uses semicolons to separate values instead of the default comma. This is common in regions where commas are used as decimal separators in numbers.

18. What's the difference between `sort_values()` and `sort_index()` in pandas?

Answer: `sort_values()` sorts the DataFrame based on the values in specified columns, while `sort_index()` sorts the DataFrame based on the index labels. Use `sort_values()` when you want to order data based on column values (like sorting posts by number of likes), and use `sort_index()` when you want to order based on the index (like sorting by post ID or timestamp index).

19. How would you group the data by 'Type' and calculate the average likes, comments, and shares for each type?

Answer: I would use the `groupby()` function followed by `mean()`:

```python
type_summary = df.groupby('Type')[['like', 'comment', 'share']].mean()
type_summary
```

This creates a summary DataFrame showing the average engagement metrics for each content type.

20. What other reshaping operations could be useful for this dataset?

Answer: Other useful reshaping operations include:

- `pivot_table()`: Create a spreadsheet-style pivot table
- `stack()` and `unstack()`: Reshape from wide to long format and vice versa
- `pivot()`: Convert distinct values into columns
- `explode()`: Transform list-like elements to individual rows
- `get_dummies()`: Convert categorical variables to dummy/indicator variables
- `crosstab()`: Compute a cross-tabulation of two or more factors

Each of these operations helps in restructuring data for different analytical needs.