Editorial-W5A2: Comprehensive Pandas Exercises - Data Management, Merging, and Missing Values

Question 1 (MSQ)

You're a data analyst at a multinational company that operates in various countries. The company collects different types of data:

1. Customer demographics from different regions (cross-sectional)
2. Daily sales figures for the past 5 years (time series)
3. Employee performance ratings across departments over time (panel data)
4. Customer reviews in text format (unstructured text data)

You're considering using Pandas for data analysis. **Which of these data types can Pandas efficiently handle?**

- A) Customer demographics from different regions
- B) Daily sales figures for the past 5 years
- C) Employee performance ratings across departments over time
- D) Customer reviews in text format

Correct Answers: A, B, C

Explanation: Pandas is designed to efficiently handle cross-sectional data, time series data, and panel data. While it can store text data, it's not optimized for processing unstructured text data, which typically requires specialized natural language processing tools.

Question 2 (MCQ)

You're working on a project to analyze customer data. You have information about each customer's name, age, purchase history, and loyalty points. You want to organize this data in a tabular format where each row represents a customer and each column represents a different attribute.

Which Pandas data structure would be most suitable for this task?

- A) Array
- B) List
- C) DataFrame
- D) Dictionary

Correct Answer: C) DataFrame

Explanation: A DataFrame is the ideal choice for this scenario. It allows you to organize data in a table-like structure with labeled axes (rows and columns), where each column can have a different data type. This matches perfectly with the customer data scenario where you

have different types of information (names, ages, purchase history, loyalty points) for each customer.

Question 3 (MSQ)

You're analyzing a dataset of products sold by an e-commerce company. The dataset includes the following information for each product:

- Product name (text)
- Price (decimal number)
- Quantity sold (integer)
- Date of last sale (date)

You decide to use a Pandas DataFrame to store this data. **Which of the following statements about this DataFrame are true?**

- A) It can hold all these different types of data (text, numbers, dates) in a single structure
- B) All columns in the DataFrame must have the same data type
- C) The structure is similar to what you might see in a spreadsheet application
- D) The DataFrame can only store the numerical data (price and quantity) from this dataset

Correct Answers: A, C

Explanation: Pandas DataFrames can hold heterogeneous data, meaning different columns can have different data types (text for names, numbers for price and quantity, dates for last sale). They are indeed similar to spreadsheets, making data manipulation easier. Each column can have its own data type, and DataFrames are not limited to just numerical data.

Question 4 (MCQ)

You're working with a DataFrame df that contains information about employees in a company. The DataFrame has columns for 'Name', 'Age', 'Department', and 'Salary'. You want to check the data type of the 'Age' column.

What will be the output of the following code?

```
import pandas as pd

data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 28]}

df = pd.DataFrame(data)

print(df.dtypes['Age'])
```

- A) object
- B) int64

- C) float64
- D) str

Correct Answer: B) int64

Explanation: In this scenario, the 'Age' column contains integer values. Pandas will infer the data type of this column as int64, which is a 64-bit integer type. This is appropriate for storing whole numbers like ages.

Question 5 (MSQ)

You're analyzing sales data stored in a Pandas DataFrame df. The DataFrame has columns for 'Date', 'Product', 'Quantity', and 'Revenue'. You need to perform various operations to extract specific data for your analysis.

Which of the following methods can you use to select columns in this DataFrame?

- A) df['Revenue']
- B) df[['Product', 'Quantity']]
- C) df.loc[:, 'Date']
- D) df.iloc[:, 0]

Correct Answers: A, B, C, D

Explanation: All these methods can be used to select columns in a DataFrame:

- A) selects the 'Revenue' column
- B) selects both 'Product' and 'Quantity' columns
- C) selects the 'Date' column using label-based indexing
- D) selects the first column (index 0) using integer-based indexing

These different methods provide flexibility in how you can access and analyze your sales data.

Question 6 (MCQ)

You're analyzing data from a cricket tournament. You have a Pandas DataFrame with columns 'Team' and 'Score'. Some teams haven't played yet, so their scores are recorded as None.

import pandas as pd

data = {'Team': ['Ind', 'Aus', 'Eng'], 'Score': [4, None, None]}

df = pd.DataFrame(data)

Now, following python code is run. print(df.dropna()) What will be the output?

- A) An empty DataFrame

- B) The original DataFrame with all three teams
- C) A DataFrame with only the 'Ind' row
- D) A DataFrame with 'Aus' and 'Eng' rows

Correct Answer: C) A DataFrame with only the 'Ind' row

Explanation: The dropna() method removes any row that contains at least one missing value. In this cricket tournament scenario, only India ('Ind') has a recorded score, so it's the only row that remains after dropping rows with NaN values.

Question 7 (MSQ)

Using the same pandas dataframe as above. Now, for your analysis, you must first replace the None scores with a score of 0, and then, find sum of all scores of all teams.

Which python code does the exact steps as given in question?

- A)

```
df = df.fillna(0)
print(df['Score'].sum())
```

- B)

```
df['Score'] = df['Score'].fillna(0)
print(df.sum())
```

- C)

```
print(df['Score'].sum())
```

- D)

```
print(df.dropna()['Score'].sum())
```

Correct Answer: A and B are both correct.

Explanation:

Both options A and B correctly achieve the desired result. Let's break down each option:

A) This code first fills all NaN values in the entire dataframe with 0 using df.fillna(0), then selects the 'Score' column and sums its values.

B) This code first selects the 'Score' column, then fills NaN values in that column with 0, and finally sums the values.

C) This option is incorrect because it will sum the 'Score' column without replacing None values with 0. The sum will come out to be correct though, because it skips None value by default.

D) This option is incorrect because it drops all rows with any None values before summing, replacing none with 0 isn't done here.

Both A & B's approaches will 1st replace None values with 0 in the 'Score' column and then sum the total score.

Question 8 (MCQ)

Let there be 2 pandas dataframes,

df1 = pd.DataFrame({'product': ['A', 'B', 'C', 'D'], 'qty': [1, 2, 3, 4]})

and

df2 = pd.DataFrame({'product': ['B', 'D', 'E', 'F'], 'price': [50, 60, 70, 80]})

Which dataframe will be returned if we perform Inner Join using appropriate pandas function? (Assume all necessary imports have been made)

- A) pd.DataFrame({'product': ['A', 'B', 'C', 'D', 'E', 'F'], 'qty': [1.0, 2.0, 3.0, 4.0, np.nan, np.nan], 'price': [np.nan, 50.0, np.nan, 60.0, 70.0, 80.0]})
- B) pd.DataFrame({'product': ['B', 'D', 'E', 'F'], 'qty': [2.0, 4.0, np.nan, np.nan], 'price': [50, 60, 70, 80]})
- C) pd.DataFrame({'product': ['B', 'D'], 'qty': [2, 4], 'price': [50, 60]})
- D) pd.DataFrame({'product': ['A', 'B', 'C', 'D'], 'qty': [1, 2, 3, 4], 'price': [np.nan, 50.0, np.nan, 60.0]})

Correct Answer: C

Explanation: Inner Join selects common value of the column present in both columns only. Pandas performs an inner join by default using merge.

Question 9 (MCQ)

Using same dataframes as in previous question.

Which dataframe will be returned if we perform df1 Left Join df2 using appropriate pandas function? (Assume all necessary imports have been made)

- A) pd.DataFrame({'product': ['A', 'B', 'C', 'D', 'E', 'F'], 'qty': [1.0, 2.0, 3.0, 4.0, np.nan, np.nan], 'price': [np.nan, 50.0, np.nan, 60.0, 70.0, 80.0]})
- B) pd.DataFrame({'product': ['B', 'D'], 'qty': [2, 4], 'price': [50, 60]})
- C) pd.DataFrame({'product': ['B', 'D', 'E', 'F'], 'qty': [2.0, 4.0, np.nan, np.nan], 'price': [50, 60, 70, 80]})
- D) pd.DataFrame({'product': ['A', 'B', 'C', 'D'], 'qty': [1, 2, 3, 4], 'price': [np.nan, 50.0, np.nan, 60.0]})

Correct Answer: D

Explanation: Left Join selects rows that are in both the dataframes and also, those which are only in left dataframe(which is df1).Use the code:

df1.merge(df2, on='product', how='left')

Question 10 (MCQ)

Using same dataframes as in previous question.

How many non-null (or, non-empty) values will be there in the resultant dataframe if we performed Full Outer Join using appropriate pandas function? (Assume all necessary imports have been made)

- A) 8
- B) 12
- C) 14
- D) 10

Correct Answer: C) 14

Explanation: Full Outer Join selects all rows present in both the dataframes. 1 row from df1 not present in df2 therefore, 2 empty values and similarly, 2 more empty values will be there since 1 row present in df2 is absent in df1 thus, total 4 empty values. Total cells present = Total columns _ total rows = 3 _ 6 = 18. Total Non-empty cells = 18 - 4 = 14. Print the resultant dataframe to visually observe this. Use the code:

df1.merge(df2, on='product', how='outer')

Question 11 (MSQ)

You're working with a pandas DataFrame df containing product information. The DataFrame has columns 'item_id', 'item', 'price', and 'sales'. Some entries contain NaN values.

Which of the following statements about handling NaN values in this DataFrame are true?

A) df.fillna(0) replaces all NaN values with 0
B) df['price'].fillna(df['price'].mean()) replaces NaN values in the 'price' column with the column's mean
C) df.ffill() fills NaN values using the previous valid value in each column
D) df.dropna() removes all rows containing at least one NaN value

Correct Answers: A, B, C, D

Explanation: All these statements are correct methods for handling NaN values in a pandas DataFrame:

- A) fills all NaN values with 0
- B) replaces NaN values in the 'price' column with the mean of that column
- C) uses forward fill to propagate the last valid value

- D) removes any row that contains at least one NaN value

Question 12 (MCQ)

You have a pandas DataFrame df with product information, including 'item_id', 'item', 'price', and 'sales'.

What will be the output of the following code?

```
df_copy = df.copy()

df_copy['price']  = df_copy['price'].fillna(df_copy['price'].median())

print(df_copy['price'].isnull().sum())
```

A) The total number of rows in the DataFrame
B) The number of NaN values in the original 'price' column
C) 0
D) The median price value

Correct Answer: C) 0

Explanation: This code creates a copy of the original DataFrame, then fills NaN values in the 'price' column with the median price. The isnull().sum() method then counts the number of remaining NaN values in the 'price' column. Since all NaN values have been filled with the median, the result will be 0, indicating no remaining NaN values in the 'price' column. fillna() does not modify df in place unless inplace=True is specified.

Question 13 (MSQ) – DataFrame Merging

You have two DataFrames:

```
import pandas as pd


# DataFrame 1: Item Specifications

df1 = pd.DataFrame({

   'item_id': [1, 2, 3, 4, 5],

   'item_name': ['Laptop', 'Tablet', 'Smartphone', 'Headphones', 'Keyboard'],

   'brand': ['Dell', 'Samsung', 'Apple', 'Sony', 'Logitech'],

   'price': [1200, 300, 800, 100, 75]

})


# DataFrame 2: Sales Data

df2 = pd.DataFrame({
```

```
    'item_id': [1, 2, 3, 4, 6],  # Note: item_id 6 is in df2 but not in df1
    'sales_date': ['2024-01-15', '2024-01-20', '2024-01-25', '2024-01-30', '2024-02-05'],
    'quantity_sold': [5, 10, 8, 12, 3]
})
```

You perform the following merge operation:

```
merged_df = pd.merge(df1, df2, on='item_id', how='outer')
```

Which of the following statements about merged_df are correct?

- **A)** All rows from both df1 and df2 will be included, with NaN values where there was no match.
- **B)** Only rows where item_id exists in both df1 and df2 will be included.
- **C)** The resulting DataFrame will have 6 rows.
- **D)** The price column will have NaN for item_id = 6.

Correct Answers: A, C, D

Explanation:

- **Option A** is correct because an **outer merge** includes all rows from both DataFrames, filling unmatched values with NaN.
- **Option B** is incorrect because an **outer merge** includes all rows, not just matching ones. That would be an **inner merge**.
- **Option C** is correct because df1 has 5 rows and df2 has 5 rows, but one (item_id=6) exists only in df2, leading to a total of **6 rows**.
- **Option D** is correct because price is present in df1 but item_id=6 is missing from df1, leading to NaN in price. This happens because price is defined in df1, and since df1 has no row for item_id=6, the merged DataFrame assigns NaN to this column.

Question 14 (MCQ) – Handling Missing Values

You have the following DataFrame with missing values:

```
import numpy as np

data = {
    'item_id': [101, 102, 103, 104, 105],
    'item': ['Apple', 'Banana', 'Orange', np.nan, 'Grapes'],
```

```
    'price': [50, np.nan, 30, 40, np.nan],
    'sales': [200, 150, np.nan, 180, 220]
}

df = pd.DataFrame(data)
```

You apply the following operation:

```
df['item'] = df['item'].fillna(df['item'].mode()[0])
```

What will be the effect of this operation?

- **A)** All missing values in the 'item' column will be replaced with 'Apple'.
- **B)** The missing values in 'item' will be filled with the most frequently occurring item name.
- **C)** The missing values in 'item' will be replaced with NaN as mode() does not work on string columns.
- **D)** The operation will throw an error if there are multiple modes.

Correct Answer: B

Explanation:

- **Option A** is incorrect because the mode is **not necessarily 'Apple'**; it depends on the most frequent value in the column.
- **Option B** is correct because fillna(df['item'].mode()[0]) fills missing values with the most frequently occurring value. If multiple values share the highest frequency, mode()[0] selects the first one, ensuring consistent behavior.
- **Option C** is incorrect because mode() works on string columns and returns the most common value.
- **Option D** is incorrect because mode()[0] selects the first mode when there are multiple, but it does not raise an error.

Question 15 (MCQ) – Forward Fill and Backward Fill

You have the following DataFrame:

```
import pandas as pd
import numpy as np

data = {
```

```
    'Student': ['A', 'B', 'C', 'D'],
    'Test1': [90, np.nan, 85, 88],
    'Test2': [np.nan, 78, 82, 80],
    'Test3': [88, np.nan, 89, np.nan]
}

df = pd.DataFrame(data)
```

You apply the following operation:

```
df.fillna(method='ffill', axis=0)
```

What will be the effect of this operation?

- **A)** Missing values will be filled using the mean of the respective columns.
- **B)** Missing values will be filled using the next valid value in the same column.
- **C)** Missing values will be replaced with the most recent non-null value in the same column.
- **D)** Missing values will be dropped from the DataFrame.

Correct Answer: C

Explanation:

Axis=0 ensures that missing values are filled downwards within each column (column-wise forward fill)

- **Option A** is incorrect because ffill does not use the mean; it propagates the last valid observation forward.
- **Option B** is incorrect because bfill (backward fill), not ffill, uses the next valid value.
- **Option C** is correct because ffill (forward fill) : missing values will be replaced with the most recent non-null value in the same column.
- **Option D** is incorrect because ffill does not drop missing values; it fills them instead.