# Pandas Advance

9 out of 11 correct

1. How can you re-index a pandas DataFrame in python?

- ○ by using index
- ◉ by using re-index
- ○ by using pandas
- ○ none of the above

**Explanation:** You can re-index a pandas DataFrame using the reindex() method or by assigning to the index property.

2. What is the output of the following code?

```
import pandas as pd

data = {'name': ['John', 'Jane', 'Bob'], 'age': [25, 30, 35]}
df = pd.DataFrame(data)

df_reindexed = df.reindex([2, 1, 0])

print(df_reindexed)
```

- ○ A DataFrame with rows in the original order
- ○ A DataFrame with rows in reverse order

- ○ A DataFrame with rows sorted by age
- ◉ A DataFrame with columns in reverse order

**Explanation:** The reindex() method is used to change the order of rows in the DataFrame. In this case, we're passing a list of row labels in reverse order, so the resulting DataFrame will have the rows in reverse order.

3. What is the output of the following code?

```
import pandas as pd

data = {'name': ['John', 'Jane', 'Bob'], 'age': [25, 30, 35]}
df = pd.DataFrame(data)

for index, row in df.iterrows():
    print(row['name'], row['age'])
```

- ◉ John 25, Jane 30, Bob 35
- ○ name John, age 25, name Jane, age 30, name Bob, age 35
- ○ ['John', 25], ['Jane', 30], ['Bob', 35]
- ○ None of the above

**Explanation:** The iterrows() method is used to iterate over rows of the DataFrame. In this case, we're printing out the values of the 'name' and 'age' columns for each row.

4. What is the best way to iterate over the rows of a Pandas DataFrame?

- ○ Using a for loop to iterate through the rows by index
- ○ Using the apply() method to apply a function to each row
- ◉ Using the iterrows() method

- Using the itertuples() method

**Explanation:** The iterrows() method returns an iterator that yields index and row data for each row. It's a convenient way to loop over the rows of a DataFrame, however, it is not very efficient and can be slow for large DataFrames

5. What is the difference between the iterrows() method and the itertuples() method for iterating over a Pandas DataFrame?

- The iterrows() method is faster but yields a Series, while the itertuples() method is slower but yields a named tuple.
- The iterrows() method yields a Series, while the itertuples() method yields a DataFrame.
- The iterrows() method yields a Series, while the itertuples() method yields a named tuple.
- The iterrows() method yields a DataFrame, while the itertuples() method yields a Series.

**Explanation:** The iterrows() method returns an iterator that yields index and row data for each row as a Series object, while the itertuples() method returns an iterator that yields a named tuple for each row, where the values are accessed by field names. The itertuples() method is faster than the iterrows() method, but is less flexible.

6. What is the output of the following code?

```
import pandas as pd

data = {'name': ['John', 'Jane', 'Bob'], 'age': [25, 30, 35]}
df = pd.DataFrame(data)

df['name_upper'] = df['name'].str.upper()

print(df)
```

- (●) A DataFrame with an extra column containing uppercase names
- ( ) A DataFrame with the 'name' column modified to contain uppercase names
- ( ) A DataFrame with an error
- ( ) None of the above

**Explanation:** We're using the str.upper() method to convert the 'name' column to uppercase, and then assigning the result to a new column called 'name_upper'.

7. How can you sort a pandas DataFrame by a specific column in ascending order?

- ( ) df.sort(column_name)
- (●) df.sort_values(column_name)
- ( ) df.sort_ascending(column_name)
- ( ) df.sort_up(column_name)

**Explanation:** The sort_values method in pandas is used to sort a DataFrame by one or multiple columns in ascending or descending order. By default, the method sorts the DataFrame in ascending order. To sort by a specific column, you can pass the column name as an argument to the sort_values method. The other answers are not valid methods in pandas.

8. Which of the following can be used to clean text data?

- ( ) Removing special characters
- ( ) Converting all text to lowercase
- ( ) Removing stop words
- (●) All of the above

**Explanation:** Removing special characters, converting all text to lowercase, and removing stop words are all common preprocessing steps used to clean text data.

9. What is the output of the following code?

```
import pandas as pd

data = {'name': ['Alice', 'Bob', 'Charlie'], 'age': [30, 25, 40]}
df = pd.DataFrame(data)

df_subset = df.loc[1:2, 'name']

print(df_subset)
```

- ○ A DataFrame with rows 1 and 2 and the 'name' column
- ○ A DataFrame with the 'name' column for rows 1 and 2
- ◉ A Series with the 'name' values for rows 1 and 2
- ○ An error

**Explanation:** We're using the loc[] method to select a subset of rows and columns from the DataFrame. In this case, we're selecting rows 1 and 2 and the 'name' column, and the resulting output is a Series with the 'name' values for those rows.

10. What is the output of the following code?

```
import pandas as pd

data = {'name': ['Alice', 'Bob', 'Charlie'], 'age': [30, 25, 40]}
df = pd.DataFrame(data)

max_age = df['age'].max()

print(max_age)
```

- ◉ The maximum age

- ○ The median age
- ○ The mean age
- ○ The mode age

**Explanation:** We're using the max() method to calculate the maximum of the 'age' column in the DataFrame.

11. What is the output of the following code?

```
import pandas as pd

data = {'date': ['2022-01-01', '2022-02-01', '2022-03-01'], 'sales': [100, 200, 300]}
df = pd.DataFrame(data)

df['date'] = pd.to_datetime(df['date'])
df['month'] = df['date'].dt.month

print(df)
```

- ◉ A DataFrame with an extra column containing the month
- ○ A DataFrame with the 'date' column modified to contain the month
- ○ A DataFrame with an error
- ○ None of the above

**Explanation:** We're using the to_datetime() method to convert the 'date' column to a datetime format, and then using the dt.month attribute to extract the month and assign it to a new column called 'month'.

Submit