# PANDAS THEORY QUESTIONS

## 1. What is the purpose of the pandas library in Python? What are the available data structures in pandas?

- The purpose of the pandas library in Python is to provide high-performance, easy-to-use data manipulation and analysis tools. It is built on top of the NumPy library and offers data structures and functions for efficiently handling and analyzing structured data.

- The two main data structures in pandas are:

    1. Series: A Series is a one-dimensional labeled array that can hold any data type. It is similar to a column in a spreadsheet or a single column of data in a SQL table. Each element in a Series has a corresponding label called an index.

    2. DataFrame: A DataFrame is a two-dimensional labeled data structure, which consists of columns and rows. It can be thought of as a tabular representation of data, similar to a spreadsheet or a SQL table. Each column in a DataFrame is a Series, and all the columns share a common index.

- These data structures in pandas provide powerful features for indexing, slicing, reshaping, aggregating, and analyzing data. They also support handling missing data, merging and joining datasets, and reading and writing data in various file formats.

## 2. What are the notable features provided by the pandas library?

- The pandas library provides several notable features for data manipulation and analysis, including:

    1. Data Structures: pandas provides powerful data structures such as Series and DataFrame that are designed to handle and manipulate tabular data efficiently. These data structures are built on top of NumPy arrays and provide a rich set of methods and functions for data manipulation.

    2. Data Cleaning and Preprocessing: pandas provides various functions for data cleaning and preprocessing, such as handling missing data, data imputation, data normalization, and data transformation. These functions help in preparing data for analysis.
    3. Data Manipulation and Aggregation: pandas provides various functions for data manipulation and aggregation, such as filtering, sorting, grouping, and pivoting. These functions help in exploring and analyzing the data efficiently.

    4. Data Visualization: pandas provides easy-to-use functions for data visualization, such as line charts, scatter plots, and histograms. These functions are built on top of the popular visualization library, matplotlib.

    5. Input and Output: pandas provides various functions for reading and writing data in different formats, such as CSV, Excel, SQL, and JSON. These functions help in working with data from various sources and exporting data to different formats.

    6. Time Series Analysis: pandas provides powerful functions for time series analysis, such as resampling, rolling, and shifting. These functions help in analyzing time series data efficiently.

- Overall, the pandas library provides a wide range of functionalities that make it a popular choice for data manipulation and analysis in Python.

## 3. What are the differences between a series and a dataframe in pandas?

- The main differences between a Series and a DataFrame in pandas are as follows:

| Attributes | Series | Dataframe |
|---|---|---|
| 1. Dimensionality | 1. A Series is a one-dimensional labeled array that can hold any data type. It is similar to a single column in a spreadsheet or a single column of data in a SQL table. | 1. A DataFrame is a two-dimensional labeled data structure that consists of columns and rows. It can be thought of as a table with multiple columns, similar to a spreadsheet or a SQL table. |
| 2. Structure | 2. A Series has a single column of data along with an associated index. The index provides labels for each element in the Series, which can be used for indexing and referencing. | 2. A DataFrame consists of multiple columns, where each column is a Series. All the columns share a common index, and each column can have its own label or name. |
| 3. Flexibility | 3. A Series has only one dimension | 3. A DataFrame provides more flexibility by allowing multiple columns and rows. This makes a DataFrame suitable for handling structured, tabular data, where different variables or attributes are stored in different columns. |
| 4. Data Operations | 4. With a Series, you can perform various operations such as indexing, slicing, and applying mathematical functions to the entire series. | 5. With a DataFrame, you can perform these operations on individual columns or rows, as well as apply operations across multiple columns or rows simultaneously. |
| 5. Visualization | A Series can be plotted as a line chart, histogram, or any other suitable | 6. A DataFrame, being a multi-dimensional structure, can be |

| | | |
|---|---|---|
| | visualization for one-dimensional data. | visualized using scatter plots, bar charts, heatmaps, or other visualizations that can represent relationships between multiple variables. |

- In summary, a Series is a one-dimensional labeled array, while a DataFrame is a two-dimensional labeled data structure with multiple columns. Series are suitable for working with single-variable data, while DataFrames are designed for handling multi-variable, structured data.

## 4. What are the various methods for creating a series in pandas?

- There are several methods available in pandas to create a Series. Here are some common ways to create a Series in pandas:

1. From a List or Array.
2. From a Dictionary.
3. Using a Scalar Value.
4. From a Range.
5. Using Custom Index.
6. From a CSV or Text File.

- These methods provide different ways to create a Series in pandas, depending on the available data and the desired index or labels for the Series.

## 5. How can a dataframe be created in different ways?

- A DataFrame in pandas can be created in different ways. Here are some common methods to create a DataFrame:

1. From a Dictionary of Lists or Arrays.
2. From a List of Dictionaries.
3. From a NumPy Array.
4. From an External File (CSV, Excel, etc.).
5. From a Series.

- These methods allow you to create a DataFrame in pandas using different data structures such as dictionaries, arrays, files, or existing Series. The choice of method depends on the available data and the desired structure of the DataFrame.

## 6. Can you provide some examples of statistical functions available in the pandas library for Python?

- Certainly! The pandas library in Python provides a wide range of statistical functions for data analysis. Here are some examples of statistical functions available in pandas.

    1. mean() : Computes the mean of the values in a Series or DataFrame.
    2. median(): Computes the median of the values in a Series or DataFrame.
    3. mode(): Computes the mode(s) of the values in a Series or DataFrame.
    4. std(): Computes the standard deviation of the values in a Series or DataFrame.
    5. var(): Computes the variance of the values in a Series or DataFrame.
    6. sum(): Computes the sum of the values in a Series or DataFrame.
    7. min(): Computes the minimum value in a Series or DataFrame.
    8. max(): Computes the maximum value in a Series or DataFrame.
    9. count(): Counts the number of non-null values in a Series or DataFrame.
    10. describe(): Generates descriptive statistics of a Series or DataFrame, including count, mean, standard deviation, minimum, quartiles, and maximum.
    11. corr(): Computes the correlation between columns in a DataFrame.
    12. cov(): Computes the covariance between columns in a DataFrame.

- These are just a few examples of the statistical functions available in pandas. The library provides many more functions for performing various statistical calculations and analysis on Series and DataFrames.

## 7. How can you sort a dataframe in pandas?

- In pandas, you can sort a DataFrame based on one or more columns using the sort_values() function. Here's how you can sort a DataFrame in pandas.

    1. Sorting by a Single Column: By default, the sort_values() function sorts the DataFrame in ascending order based on the specified column.
    2. Sorting in Descending Order: To sort the DataFrame in descending order, you can set the ascending parameter to False.

    3. Sorting by Multiple Columns: If you want to sort the DataFrame based on multiple columns, you can pass a list of column names to the sort_values() function. The DataFrame will be sorted first by the first column specified, and then by subsequent columns in the list.

    4. Inplace Sorting: By setting the inplace parameter to True, the DataFrame will be sorted in place, meaning the original DataFrame will be modified.

    5. Sorting by Index: If you want to sort the DataFrame based on the index, you can use the sort_index() function.

- These are some of the common ways to sort a DataFrame in pandas. The sort_values() function provides flexibility for sorting based on single or multiple columns, in ascending or descending order.

## 8. What is the process of setting an index for a pandas dataframe?

- In pandas, you can set an index for a DataFrame using the set_index() function. The process of setting an index involves specifying a column or a combination of columns that will become the new index of the DataFrame.

- Here's how you can set an index for a pandas DataFrame:

```
# Set a single column as the index
df.set_index('Column_Name', inplace=True)

# Set multiple columns as the index
df.set_index(['Column1', 'Column2'], inplace=True)
```

- In the first example, a single column is specified as the index by passing the column name to the set_index() function. By setting the inplace parameter to True, the original DataFrame will be modified, and the specified column will become the new index.

- In the second example, multiple columns are specified as the index by passing a list of column names to the set_index() function. The DataFrame will be indexed hierarchically based on the order of the columns in the list.

- Alternatively, you can also create a new DataFrame with the desired index using the DataFrame constructor:

```
# Create a new DataFrame with a specific index
df_new = pd.DataFrame(data, index=index)
```

- Here, data represents the data to be included in the DataFrame, and index is the index you want to set for the DataFrame. This approach allows you to create a DataFrame with a custom index directly.

- Setting an index is useful for data alignment, filtering, grouping, and various other operations

## 9. How do you add a new row to an existing pandas dataframe?

- To add a new row to an existing pandas DataFrame, you can use the loc indexer or the append() function.
- Here's how you can do it:

    1. Using loc indexer: The loc indexer is used to access a specific location in the DataFrame. By appending the new_row dictionary to the DataFrame using the append() function, a new row is added. The ignore_index=True parameter ensures that the index of the new row is automatically adjusted.

    2. Using append() function: Instead of using a dictionary, you can create a new row as a Series object with the same column names as the DataFrame. The append() function is then used to append the new_row Series to the DataFrame.

- Both approaches will add a new row to the DataFrame, and the ignore_index=True parameter ensures that the index is adjusted appropriately. It's important to reassign the modified DataFrame to the original variable (df in the examples) to capture the changes.

## 10. What is the procedure for adding a new column to a pandas dataframe?

- Adding a new column to a pandas DataFrame involves assigning a new Series or array to the DataFrame with the desired column name.

- Here's the procedure for adding a new column to a pandas DataFrame:

```
import pandas as pd

# Create a DataFrame
df = pd.DataFrame({'Name': ['sujit', 'rohit', 'sham'],
                   'Age': [26, 21, 26]})

# Add a new column
df['City'] = ['pune', 'mumbai', 'delhi']
```

- In the example above, a new column named 'City' is added to the DataFrame df. To add the column, you assign a new Series or array to the DataFrame using the desired column name as the key. The length of the new Series or array must match the length of the existing DataFrame.

- Alternatively, you can create a new column based on existing columns using operations or functions applied to the DataFrame:

```
# Create a new column based on existing columns
df['Birth_Year'] = pd.datetime.now().year - df['Age']
```

- In this example, a new column named 'Birth_Year' is created based on the 'Age' column. The pd.datetime.now().year gives the current year, and by subtracting the 'Age' from it, the birth year is calculated and assigned to the new column.

- The procedure for adding a new column to a pandas DataFrame is straightforward. Simply assign a new Series or array to the DataFrame using the desired column name as the key, or create a new column based on existing columns using operations or functions.

## 11. How can you convert a numpy array into a dataframe using pandas?

- You can convert a NumPy array into a DataFrame using the pandas library in Python.

- Here is how you can do it:

```
import pandas as pd
import numpy as np

# Create a NumPy array
my_array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Convert NumPy array to DataFrame
df = pd.DataFrame(my_array)
```

- In the code above, we first create a NumPy array called my_array. Then, the NumPy array is passed as an argument to the pd.DataFrame() function, which converts it into a DataFrame df.

- By default, the DataFrame will have numerical indexes and column labels starting from 0. However, you can customize the index and column labels by passing additional arguments to the pd.DataFrame() function.

- For example:

```
df = pd.DataFrame(my_array, index=['row1', 'row2', 'row3'], columns=['col1', 'col2', 'col3'])
```

- In this case, we provide the index argument to specify custom row labels ('row1', 'row2', 'row3') and the columns argument to specify custom column labels ('col1', 'col2', 'col3').

- By converting a NumPy array into a DataFrame, you can take advantage of the additional functionality provided by pandas for data manipulation, analysis, and visualization.

## 12. What is the method to delete a row from a pandas dataframe?

- To delete a row from a pandas DataFrame, you can use the drop() method. The drop() method allows you to remove rows based on their index or condition.

- Here are two common methods for deleting a row from a DataFrame:

    1. Deleting a Row by Index:
       # Delete row by index
       df.drop(index, inplace=True)

       In this method, you specify the index label of the row you want to delete and set the inplace parameter to True to modify the original DataFrame.

    2. Deleting Rows Based on a Condition:
       # Delete rows based on a condition
       df = df[condition]

       In this method, you define a condition that evaluates to True for the rows you want to keep and False for the rows you want to delete. The resulting DataFrame will only contain the rows that satisfy the condition.

- Example:

```
# Delete rows where the 'Age' column is less than 30
df = df[df['Age'] >= 30]
```

  This code will keep only the rows where the 'Age' column has a value greater than or equal to 30 and remove the rows that do not meet this condition.

- It's important to note that both methods return a modified DataFrame. If you want to keep the original DataFrame unchanged, you can assign the modified DataFrame to a new variable.

- By using the appropriate method based on your requirements, you can easily delete a row or multiple rows from a pandas DataFrame.