What is pandas in python?

Pandas stands for pannel data

Pandas is a popular open-source data manipulation and analysis library for Python. It provides easy-to-use data structures and functions needed to manipulate and analyze structured data seamlessly. The two primary data structures in Pandas are Series and DataFrame.

Data Structure:

**Series** - It is a one-dimensional array-like structure with homogeneous data. its values are mutable i.e. it can be changed but the size of the series is immutable.

Pd.Series([1,2,3])

Pd.Series(data = [1,2,3], index=[‘a’,’b’,’c’])

Pd.Series(np.array([1,2,3]))

Pd.Series(data = np.array([1,2,3]), index=[‘a’,’b’,’c’])

**DataFrame** - It is a two-dimensional array-like structure with heterogeneous data. and the data is aligned in a tabular manner. Both size and values of DataFrame are mutable.

data = [['mark', 20], ['zack', 16], ['ron', 24]]

# Create the pandas DataFrame

df = pd.DataFrame(data, columns=['Name', 'Age'])

data = {'Name': ['Max', 'Lara', 'Koke', 'muller'],

'Age': [10, 31, 91, 48]}

# Create DataFrame

df = pd.DataFrame(data)

**Panel** - The Pandas have a third type of data structure known as Panel, which is a 3D data structure capable of storing heterogeneous data but it isn’t that widely used

**11. What is NumPy?**

[**NumPy**](https://www.interviewbit.com/numpy-cheat-sheet/) is one of the most widely used, versatile, simple, open-source, python-based, general-purpose packages that is used for processing arrays. NumPy is an abbreviation for **NUM**erical **PY**thon. Due to its highly optimized tools, it provides high-performance and powerful N-dimensional array processing capabilities that are explicitly designed to handle complex arrays. It is most commonly used in performing scientific computations and various broadcasting functions because of its popularity, powerful performance, and flexibility to perform various operations.

**1. Is iterating over a Pandas Dataframe a good practice? If not what are the important conditions to keep in mind before iterating?**

Ideally, iterating over pandas DataFrames is **definitely not the best practice** and one should only consider doing so when it is absolutely necessary and no other function is applicable. The iteration process through DataFrames is very inefficient. Pandas provide a lot of functions using which an operation can be executed without iterating through the dataframe. There are certain conditions that need to be checked before

Before attempting to iterate through pandas objects, we must first ensure that none of the below-stated conditions aligns with our use case:

* **Applying a function to rows:** A common use case of iteration is when it comes to applying a function to every row, which is designed to work only one row at a time and cannot be applied on the full DataFrame or Series. In such cases, it’s always recommended to use apply() method instead of iterating through the pandas object.

**How would you iterate over rows in a DataFrame in Pandas?**

Although it is not a good practice to iterate over rows in Pandas if there is no other alternative we do so using either iterrows() or itertuples() built-in methods.

* **pandas.DataFrame.iterrows():** This method is used to iterate over DataFrame rows as (index, Series) pairs. There is only one drawback for this method it does not preserve the dtypes across rows due to the fact that it converts each row into a Series. If you need to preserve the dtypes of the pandas object, then one should use itertuples() method instead.
* **import** pandas **as** pd
* # Define a dictionary containing students data
* data = {'Name': ['Sneha', 'Shreya',
* 'Sabhya', 'Riya'],
* 'Age': [22, 18, 10, 19],
* 'Stream': ['Computer', 'Commerce',
* 'Arts', 'Mechanical'],
* 'Percentage': [89, 93, 97, 73]}
* # Convert the dictionary into DataFrame
* df = pd.DataFrame(data, columns=['Name', 'Age',
* 'Stream', 'Percentage'])
* print("Given Dataframe :\n", df)
* print("\nIterating over rows using iterrows() method :\n")
* # iterate through each row and select
* # 'Name' and 'Age' columns respectively.
* **for** index, row **in** df.iterrows():
* print(row["Name"], row["Age"])

### How are iloc() and loc() different?

### How will you sort a DataFrame?

### What is the difference between join() and merge() in Pandas?

Diff in fillna and interpolate

**fillna:**

* The **fillna** method is used to fill missing values with a specified constant value or using a particular method.
* You can replace NaN (missing) values with a constant value or use methods like 'ffill' (forward fill) or 'bfill' (backward fill) to fill missing values based on the surrounding values.
* It's a simple way to replace missing values without considering the relationships between different data points.

import pandas as pd

# Create a DataFrame with missing values

data = {'A': [1, 2, np.nan, 4, 5],

'B': [10, np.nan, np.nan, 40, 50]}

df = pd.DataFrame(data)

# Fill missing values with a constant

df.fillna(0, inplace=True)

# Forward fill missing values

# df.ffill(inplace=True)

**interpolate:**

* The **interpolate** method is used to fill missing values by performing interpolation, which means estimating missing values based on the values of neighboring data points.
* It considers the relationship between data points and estimates missing values using various interpolation methods (linear, polynomial, etc.).
* It is more sophisticated than **fillna** as it takes into account the structure and trends in the data.

import pandas as pd

# Create a DataFrame with missing values

data = {'A': [1, 2, np.nan, 4, 5],

'B': [10, np.nan, np.nan, 40, 50]}

df = pd.DataFrame(data)

# Interpolate missing values using linear interpolation

df.interpolate(method='linear', inplace=True)

interpolate means :  The process of estimating the value of a function at a point from its values at nearby points.

Joins in pandas:

Left

Right

Outer

Inner

Cross

Apply function

The **apply** function in Pandas is a powerful and flexible method used for applying a function along the axis of a DataFrame (either rows or columns) or on specific elements of a Series. The primary purpose of the **apply** function is to perform custom operations on the data in a Pandas DataFrame or Series

Diff in concat and append:

* **concat:**
  + More general-purpose, can concatenate along both rows and columns.
  + Allows concatenation of multiple DataFrames in a single call.
  + Offers more flexibility with additional parameters.
* **append:**
  + Specifically designed for appending along rows (**axis=0**).
  + Convenient for adding one DataFrame to another row-wise.
  + A shortcut for a specific use case, providing a concise syntax.

Concat can add 2 or more df

Append can only append one df to other

What is melt:

The **melt** function in Pandas is used to transform or reshape a DataFrame from wide format to long format. The primary purpose of **melt** is to unpivot a DataFrame by converting columns into rows, making it easier to work with and analyze the data.