Kumar Samyak week-03 report on Numpy, Pandas and operations related to it:-

Name: Kumar Samyak

Domain: Python

Date: 14/02/2024

→Introduction Numpy:-

NumPy is a Python library that provides powerful and versatile array computations, mathematical functions, and other tools for data analysis and visualization.

Here are some of its features, advantages, disadvantages, and applications:

Features:

NumPy supports large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

NumPy offers comprehensive numerical computing tools, such as random number generators, linear algebra routines, Fourier transforms, and more.

NumPy is interoperable with other Python libraries and supports a wide range of hardware and computing platforms.

NumPy is easy to use and has a high level syntax that makes it accessible and productive for programmers from any background or experience level.

Advantages:

NumPy is faster and more efficient than traditional Python lists, as it stores arrays at one continuous place in memory and is optimized to work with latest CPU architectures.

NumPy is the universal standard for working with numerical data in Python, and it is at the core of the scientific Python and PyData ecosystems3.

NumPy enables complex and sophisticated data analysis and visualization, as well as scientific and engineering applications.

Disadvantages:

NumPy arrays have a fixed size and type, which means they cannot be resized or changed dynamically, unlike Python lists.

NumPy arrays consume more memory than Python lists, as they store additional information such as shape, strides, and data type.

NumPy is not suitable for handling non-numerical data, such as text, images, or audio.

Applications:

NumPy is widely used in various domains of science and engineering, such as quantum computing, statistical computing, signal processing, image processing, graphs and networks, astronomy, cognitive psychology, bioinformatics, Bayesian inference, mathematical analysis, chemistry, geoscience, geographic processing, architecture and engineering, and more.

NumPy also works well with other array libraries, such as Dask, CuPy, JAX, and Xarray, which provide distributed, GPU-accelerated, or labeled array computations.

NumPy is the foundation for many popular machine learning and data science libraries, such as Pandas, Scikit-learn, TensorFlow, PyTorch, and Keras.

Numpy Operations:-

NumPy operations are functions or methods that can be applied to NumPy arrays to perform various mathematical or logical computations.

Some examples of NumPy operations are:

1.)Arithmetic operations: These are element-wise operations that perform basic arithmetic calculations on NumPy arrays, such as addition, subtraction, multiplication, division, exponentiation, etc.

For example:

```
import numpy as np

a = np.array([1, 2, 3])

b = np.array([4, 5, 6])

a + b

array([5, 7, 9])

a * b

array([ 4, 10, 18])

a ** b

array([   1,   32,  729])
```

2.)Aggregation operations: These are operations that reduce the dimensionality of NumPy arrays by computing summary statistics, such as sum, mean, median, min, max, standard deviation, etc.

For example:

```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]])

np.sum(a)

21

np.mean(a)

3.5
```

```
np.max(a, axis=0)
array([4, 5, 6])
```

3.)Logical operations: These are operations that perform boolean logic on NumPy arrays, such as and, or, not, xor, etc. They return arrays of True or False values based on the condition.

For example:

```
import numpy as np
a = np.array([True, False, True])
b = np.array([False, True, False])
np.logical_and(a, b)
array([False, False, False])
np.logical_or(a, b)
array([ True,  True,  True])
np.logical_not(a)
array([False,  True, False])
```

4.)Trigonometric operations: These are operations that perform trigonometric functions on NumPy arrays, such as sin, cos, tan, arcsin, arccos, arctan, etc. They return arrays of the corresponding trigonometric values in radians.

For example:

```
import numpy as np
a = np.array([0, np.pi/2, np.pi])
np.sin(a)
array([0., 1., 0.])
np.cos(a)
array([ 1.,  0., -1.])
```

np.tan(a)

array([0., 1.63312394e+16, -0.])

→Introduction to Pandas and it's operations:-

Pandas is a Python library that provides powerful and easy-to-use data analysis and manipulation tools, such as data frames, series, indexing, grouping, merging, reshaping, and plotting123. Here are some of its features, advantages, disadvantages, and applications:

Features:

Pandas supports various types of data, such as tabular, time series, matrix, text, and categorical.

Pandas offers a rich set of methods and functions to perform data cleaning, transformation, aggregation, and validation.

Pandas integrates well with other Python libraries and tools, such as NumPy, SciPy, Matplotlib, Seaborn, Scikit-learn, and more.

Pandas has a high-level and expressive syntax that makes it easy and intuitive to work with data.

Advantages:

Pandas is fast and efficient, as it uses optimized and vectorized operations on NumPy arrays under the hood.

Pandas is flexible and customizable, as it allows users to define their own data types, indexes, and functions.

Pandas is widely used and popular, as it has a large and active community of developers and users.

Pandas enables complex and sophisticated data analysis and visualization, as well as machine learning and data science applications.

Disadvantages:

Pandas can consume a lot of memory, as it stores data in memory and creates copies of data when modifying it.

Pandas can be slow and inefficient for large and complex data, as it may require looping, multiple operations, or custom solutions.

Pandas can be confusing and inconsistent, as it has many ways to do the same thing, and some features may not work as expected or change over time.

Applications:

Pandas is widely used in various domains of science, engineering, business, finance, economics, education, and more.

Pandas can handle various types of data sources, such as CSV, Excel, JSON, SQL, HTML, HDF5, and more.

Pandas can create various types of plots and charts, such as line, bar, pie, scatter, histogram, box, and more.

Pandas operations:-

Pandas operations are functions or methods that can be applied to Pandas data structures, such as Series and DataFrame, to perform various data analysis and manipulation tasks.

Some examples of Pandas operations are:

a)Sorting: This is an operation that arranges the data in a certain order, such as ascending or descending, based on one or more columns or indexes.

For example:

```
import pandas as pd
df = pd.DataFrame({'name': ['Alice', 'Bob', 'Charlie', 'David'],
                   'age': [25, 30, 35, 40],
                   'score': [85, 90, 95, 100]})
df.sort_values(by='age')
```

```
     name  age  score
0    Alice   25     85
1      Bob   30     90
2  Charlie   35     95
3    David   40    100
```

```
df.sort_index(ascending=False)
```

```
     name  age  score
3    David   40    100
2  Charlie   35     95
1      Bob   30     90
0    Alice   25     85
```

b)Filtering: This is an operation that selects a subset of data that satisfies a certain condition or criteria.

For example:

```
import pandas as pd
df = pd.DataFrame({'name': ['Alice', 'Bob', 'Charlie', 'David'],
                   'age': [25, 30, 35, 40],
                   'score': [85, 90, 95, 100]})
df[df['age'] > 30]
```

```
     name  age  score
2  Charlie   35     95
3    David   40    100
```

```
df[df['name'].str.startswith('D')]
```

```
   name  age  score
3  David   40    100
```

c)Grouping: This is an operation that splits the data into groups based on some criteria, and then applies a function to each group, such as aggregation, transformation, or filtering.

For example:

```
import pandas as pd

df = pd.DataFrame({'name': ['Alice', 'Bob', 'Charlie', 'David'],

                   'gender': ['F', 'M', 'M', 'M'],

                   'age': [25, 30, 35, 40],

                   'score': [85, 90, 95, 100]})

df.groupby('gender').mean()

        age  score

gender

F      25.0   85.0

M      35.0   95.0

df.groupby('gender').filter(lambda x: x['score'].max() > 90)

     name gender  age  score

1      Bob      M   30     90

2  Charlie      M   35     95

3    David      M   40    100
```