Day 10 Assessment

Name: Krushnakumar Patle

Email: krishnapatle128@gmail.com

Batch: Data Engineering Batch-1

Numpy in Python

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy is a powerful numerical computing library in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of high-level mathematical functions to operate on these arrays. NumPy is a fundamental package for scientific computing with Python and is widely used in various fields such as machine learning, data analysis, and scientific research.

```
import numpy

arr = numpy.array([1, 2, 3, 4, 5])

print(arr)
```

```
[1 2 3 4 5]
```

```
[2] import numpy as np

    # Creating a NumPy array
    arr = np.array([1, 2, 3, 4, 5])

    # Performing element-wise operations
    arr_squared = arr ** 2

    # Printing the result
    print(arr_squared)
```

```
[ 1  4  9 16 25]
```

Dimensions in Arrays

A dimension in arrays is one level of array depth (nested arrays).

0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

```
[3] import numpy as np

    arr = np.array(42)

    print(arr)
```

```
42
```

1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

```
import numpy as np

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

print(arr)
```

```
[1 2 3 4 5]
```

Day 10 Assessment

2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

These are often used to represent matrix or 2nd order tensors.

```
import numpy as np

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

print(arr)
```

```
[[1 2 3]
 [4 5 6]]
```

3-D arrays

An array that has 2-D arrays (matrices) as its elements is called 3-D array.

These are often used to represent a 3rd order tensor.

```
[6] import numpy as np

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

print(arr)
```

```
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]
```

Day 10 Assessment

Pandas in Python

Pandas is a Python library used for working with data sets.

It has functions for analyzing, cleaning, exploring, and manipulating data.

```
import pandas as pd

# Creating a DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35],
        'City': ['New York', 'San Francisco', 'Los Angeles']}

df = pd.DataFrame(data)

# Displaying the DataFrame
print(df)
```

```
      Name  Age           City
0    Alice   25       New York
1      Bob   30  San Francisco
2  Charlie   35    Los Angeles
```

Count Values in Pandas Dataframe

Step 1: Importing libraries.

```
[8] # importing libraries
    import numpy as np
    import pandas as pd
```

Day 10 Assessment

Step 2: Creating Dataframe

```
# Creating dataframe with
# some missing values
import numpy as np
import pandas as pd
NaN = np.nan
dataframe = pd.DataFrame({'Name': ['Shobhit', 'Vaibhav',
                                   'Vimal', 'Sourabh',
                                   'Rahul', 'Shobhit'],
                          'Physics': [11, 12, 13, 14, NaN, 11],
                          'Chemistry': [10, 14, NaN, 18, 20, 10],
                          'Math': [13, 10, 15, NaN, NaN, 13]})



print(dataframe.count())
print (dataframe)
```

```
Name         6
Physics      5
Chemistry    5
Math         4
dtype: int64
      Name  Physics  Chemistry  Math
0  Shobhit     11.0       10.0  13.0
1  Vaibhav     12.0       14.0  10.0
2    Vimal     13.0        NaN  15.0
3  Sourabh     14.0       18.0   NaN
4    Rahul      NaN       20.0   NaN
5  Shobhit     11.0       10.0  13.0
```

Step 3: In this step, we just simply use the .count() function to count all the values of different columns.

```
[10] # using dataframe.count()
     # to count all values
     dataframe.count()
```

```
Name         6
Physics      5
Chemistry    5
Math         4
dtype: int64
```

We can see that there is a difference in count value as we have missing values. There are 5 values in the Name column,4 in Physics and Chemistry, and 3 in Math. In this case, it uses it's an argument with its default values.

Step 4: If we want to count all the values with respect to row then we have to pass axis=1 or 'columns'.

```
# we can pass either axis=1 or
# axos='columns' to count with respect to row
print(dataframe.count(axis = 1))

print(dataframe.count(axis = 'columns'))
```

```
0    4
1    4
2    3
3    3
4    2
5    4
dtype: int64
0    4
```

```
1    4
2    3
3    3
4    2
5    4
dtype: int64
```

Step 5: Now if we want to count null values in our dataframe.

```
# it will give the count
# of individual columns count of null values
print(dataframe.isnull().sum())

# it will give the total null
# values present in our dataframe
print("Total Null values count: ",
      dataframe.isnull().sum().sum())
```

```
Name         0
Physics      1
Chemistry    1
Math         2
dtype: int64
Total Null values count:  4
```

Step 6:. Some examples to use .count() Now we want to count no of students whose physics marks are greater than 11.

```
[13] # count of student with greater
     # than 11 marks in physics
     print("Count of students with physics marks greater than 11 is->",
           dataframe[dataframe['Physics'] > 11]['Name'].count())

     # resultant of above dataframe
     dataframe[dataframe['Physics']>11]
```

```
Count of students with physics marks greater than 11 is-> 3
```

	Name	Physics	Chemistry	Math
1	Vaibhav	12.0	14.0	10.0
2	Vimal	13.0	NaN	15.0
3	Sourabh	14.0	18.0	NaN

Count of students whose physics marks are greater than 10,chemistry marks are greater than 11 and math marks are greater than 9.

```
# Count of students whose physics marks
# are greater than 10,chemistry marks are
# greater than 11 and math marks are greater than 9.
print("Count of students ->",
      dataframe[(dataframe['Physics'] > 10) &
                (dataframe['Chemistry'] > 11) &
                (dataframe['Math'] > 9)]['Name'].count())

# dataframe of above result
dataframe[(dataframe['Physics'] > 10 ) &
          (dataframe['Chemistry'] > 11 ) &
          (dataframe['Math'] > 9 )]
```

Day 10 Assessment

```
Count of students -> 1
```

	Name	Physics	Chemistry	Math
1	Vaibhav	12.0	14.0	10.0

Double-click (or enter) to edit

Below is the full implementation:

```
# importing Libraries
import pandas as pd
import numpy as np

# Creating dataframe using dictionary
NaN = np.nan
dataframe = pd.DataFrame({'Name': ['Shobhit', 'Vaibhav',
                                   'Vimal', 'Sourabh',
                                   'Rahul', 'Shobhit'],
                          'Physics': [11, 12, 13, 14, NaN, 11],
                          'Chemistry': [10, 14, NaN, 18, 20, 10],
                          'Math': [13, 10, 15, NaN, NaN, 13]})

print("Created Dataframe")
print(dataframe)

# finding Count of all columns
print("Count of all values wrt columns")
print(dataframe.count())

# Count according to rows
```

Day 10 Assessment

```
print("Count of all values wrt rows")
print(dataframe.count(axis=1))
print(dataframe.count(axis='columns'))

# count of null values
print("Null Values counts ")
print(dataframe.isnull().sum())
print("Total null values",
      dataframe.isnull().sum().sum())
# count of student with greater
# than 11 marks in physics
print("Count of students with physics marks greater than 11 is->",
      dataframe[dataframe['Physics'] > 11]['Name'].count())

# resultant of above dataframe
print(dataframe[dataframe['Physics'] > 11])
print("Count of students ->",
      dataframe[(dataframe['Physics'] > 10) &
                (dataframe['Chemistry'] > 11) &
                (dataframe['Math'] > 9)]['Name'].count())

print(dataframe[(dataframe['Physics'] > 10) &
                (dataframe['Chemistry'] > 11) &
                (dataframe['Math'] > 9)])
```

```
Created Dataframe
      Name  Physics  Chemistry  Math
0  Shobhit     11.0       10.0  13.0
1  Vaibhav     12.0       14.0  10.0
2    Vimal     13.0        NaN  15.0
3  Sourabh     14.0       18.0   NaN
4    Rahul      NaN       20.0   NaN
5  Shobhit     11.0       10.0  13.0
```

```
4    Rahul      NaN       20.0   NaN
5  Shobhit     11.0       10.0  13.0
Count of all values wrt columns
Name         6
Physics      5
Chemistry    5
Math         4
dtype: int64
Count of all values wrt rows
0    4
1    4
2    3
3    3
4    2
5    4
dtype: int64
0    4
1    4
2    3
3    3
4    2
5    4
dtype: int64
Null Values counts
Name         0
Physics      1
Chemistry    1
Math         2
dtype: int64
Total null values 4
Count of students with physics marks greater than 11 is-> 3
      Name  Physics  Chemistry  Math
1  Vaibhav     12.0       14.0  10.0
2    Vimal     13.0        NaN  15.0
3  Sourabh     14.0       18.0   NaN
Count of students -> 1
```

Types of Joins in Pandas

Pandas Inner Join

Inner join is the most common type of join you'll be working with. It returns a Dataframe with only those rows that have common characteristics. This is similar to the intersection of two sets.

0s

```
# importing pandas
import pandas as pd

# Creating dataframe a
a = pd.DataFrame()

# Creating Dictionary
d = {'id': [1, 2, 10, 12],
     'val1': ['a', 'b', 'c', 'd']}

a = pd.DataFrame(d)

# Creating dataframe b
b = pd.DataFrame()

# Creating dictionary
d = {'id': [1, 2, 9, 8],
     'val1': ['p', 'q', 'r', 's']}
b = pd.DataFrame(d)

# inner join
df = pd.merge(a, b, on='id', how='inner')

# display dataframe
```

Day 10 Assessment

	id	val1_x	val1_y
0	1	a	p
1	2	b	q

Pandas Left Join

With a left outer join, all the records from the first Dataframe will be displayed, irrespective of whether the keys in the first Dataframe can be found in the second Dataframe. Whereas, for the second Dataframe, only the records with the keys in the second Dataframe that can be found in the first Dataframe will be displayed.

```
# importing pandas
import pandas as pd

# Creating dataframe a
a = pd.DataFrame()

# Creating Dictionary
d = {'id': [1, 2, 10, 12],
     'val1': ['a', 'b', 'c', 'd']}

a = pd.DataFrame(d)

# Creating dataframe b
b = pd.DataFrame()

# Creating dictionary
d = {'id': [1, 2, 9, 8],
```

	id	val1_x	val1_y
0	1	a	p
1	2	b	q
2	10	c	NaN
3	12	d	NaN

Pandas Right Outer Join

For a right join, all the records from the second Dataframe will be displayed. However, only the records with the keys in the first Dataframe that can be found in the second Dataframe will be displayed.

```
# importing pandas
import pandas as pd

# Creating dataframe a
a = pd.DataFrame()

# Creating Dictionary
d = {'id': [1, 2, 10, 12],
     'val1': ['a', 'b', 'c', 'd']}

a = pd.DataFrame(d)

# Creating dataframe b
b = pd.DataFrame()

# Creating dictionary
```

Day 10 Assessment

```
# Creating dataframe b
b = pd.DataFrame()

# Creating dictionary
d = {'id': [1, 2, 9, 8],
     'val1': ['p', 'q', 'r', 's']}
b = pd.DataFrame(d)

# right outer join
df = pd.merge(a, b, on='id', how='right')

# display dataframe
df
```

	id	val1_x	val1_y
0	1	a	p
1	2	b	q
2	9	NaN	r
3	8	NaN	s

Pandas Full Outer Join

A full outer join returns all the rows from the left Dataframe, and all the rows from the right Dataframe, and matches up rows where possible, with NaNs elsewhere. But if the Dataframe is complete, then we get the same output.

```
# importing pandas
import pandas as pd

# Creating dataframe a
a = pd.DataFrame()

# Creating Dictionary
d = {'id': [1, 2, 10, 12],
     'val1': ['a', 'b', 'c', 'd']}

a = pd.DataFrame(d)

# Creating dataframe b
b = pd.DataFrame()

# Creating dictionary
d = {'id': [1, 2, 9, 8],
     'val1': ['p', 'q', 'r', 's']}
b = pd.DataFrame(d)

# full outer join
df = pd.merge(a, b, on='id', how='outer')

# display dataframe
df
```

Day 10 Assessment

	id	val1_x	val1_y
0	1	a	p
1	2	b	q
2	10	c	NaN
3	12	d	NaN
4	9	NaN	r
5	8	NaN	s

Pandas Index Join

To merge the Dataframe on indices pass the left_index and right_index arguments as True i.e. both the Dataframes are merged on an index using default Inner Join.

```
[22] # importing pandas
     import pandas as pd

     # Creating dataframe a
     a = pd.DataFrame()

     # Creating Dictionary
     d = {'id': [1, 2, 10, 12],
          'val1': ['a', 'b', 'c', 'd']}

     a = pd.DataFrame(d)

     # Creating dataframe b
     b = pd.DataFrame()
```

```
# Creating dataframe b
b = pd.DataFrame()

# Creating dictionary
d = {'id': [1, 2, 9, 8],
     'val1': ['p', 'q', 'r', 's']}
b = pd.DataFrame(d)

# index join
df = pd.merge(a, b, left_index=True, right_index=True)

df
```

	id_x	val1_x	id_y	val1_y
0	1	a	1	p
1	2	b	2	q
2	10	c	9	r
3	12	d	8	s