Markdown language

Data Analysis with Numpy and Pandas

In this class we will be studying about the following contents:

1. Intro to Numpy
2. Creating an array
3. Indexing and Slicing
4. Statistical Operations using Numpy
5. Introduction to Pandas
6. Introduction to Series and Dataframe
7. Working with .csv
8. Working with .xlsx
9. Re-indexing
10. Handling missing Values

In [1]:
```
import numpy as cipher_np #Importing numpy
import pandas as cipher_pd
```

In [2]:
```
dummy_list = [1, 2, 3, 4, 5]  #List creation
dummy_list
```

Out[2]:
```
[1, 2, 3, 4, 5]
```

In [3]:
```
type(dummy_list)
```

Out[3]:
```
list
```

In [4]:
```
dummy_array = cipher_np.array(dummy_list) #Passing list as numpy array
dummy_array
```

Out[4]:
```
array([1, 2, 3, 4, 5])
```

In [5]:
```
type(dummy_array)
```

Out[5]:
```
numpy.ndarray
```

Indexing

In [6]:
```
print(dummy_list[0])
print(dummy_array[0])
```

```
1
1
```

Slicing

In [7]:
```
len(dummy_array)
```

Out[7]:
```
5
```

In [8]:
```
# create a random nd-array
cipher_np.random.randint(-10, 100, (4, 5))
```

Out[8]:
```
array([[18, 42, 74, 41, 16],
       [51, 24, -2, 38, 50],
       [85, 34, -5, 98, 47],
       [85, 92, 14, 43, 47]])
```

In [9]:
```
# Create an nd-array of 1's
cipher_np.ones((3, 4))
```

Out[9]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [10]:
```
# Create an nd-array of 0's
cipher_np.zeros((3, 4))
```

Out[10]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [11]:
```
dummy_md_list = [[1, 2, 3, 4, 5], [11, 22, 33, 44, 55], [111, 222, 333, 444, 555]]
dummy_md_list
```

Out[11]:
```
[[1, 2, 3, 4, 5], [11, 22, 33, 44, 55], [111, 222, 333, 444, 555]]
```

In [12]:
```
dummy_md_array = cipher_np.array(dummy_md_list)
dummy_md_array
```

Out[12]:
```
array([[  1,   2,   3,   4,   5],
       [ 11,  22,  33,  44,  55],
       [111, 222, 333, 444, 555]])
```

In [13]:
```
# dummy_md_array = dummy_md_array[:, :-3] ## XXXXXXX
```

In [14]:
```
# Checking the order of the multi-dimensional array
dummy_md_array.shape
```

Out[14]:
```
(3, 5)
```

In [15]:
```
# Get ALL rows and ALL FIRST 3 COLUMNS of the multi-dimensional array
# Expected output:
# [[1, 2, 3],
# [11, 22, 33],
# [111, 222, 333]]

# How to do slicing in nd-array?
```

```
    # "," - Before the comma, we mention row indices (as list or as single integers
    # ":" - Collon is used to access all the rows or all the columns in the nd-arra
    # ":4" - This slices the nd-array till 4th index

dummy_md_array[:, :]
```

Out[15]:

```
array([[  1,   2,   3,   4,   5],
       [ 11,  22,  33,  44,  55],
       [111, 222, 333, 444, 555]])
```

In [16]:

```
dummy_md_array[:, [0, 1,2]]
```

Out[16]:

```
array([[  1,   2,   3],
       [ 11,  22,  33],
       [111, 222, 333]])
```

In [17]:

```
dummy_md_array[:, 0:3]
```

Out[17]:

```
array([[  1,   2,   3],
       [ 11,  22,  33],
       [111, 222, 333]])
```

In [18]:

```
# Get the first 2 rows and first 3 columns of the multi-dimensional array
# Expected output:
# [[  1,   2,   3],
# [ 11,  22,  33]]

dummy_md_array[[:2], [:3]]
```

```
  Cell In[18], line 6
    dummy_md_array[[:2], [:3]]
                    ^
SyntaxError: invalid syntax
```

In [ ]:

```
dummy_md_array[0:2, 0:3]
```

In [ ]:

```
# Access the last index of the nd-array
dummy_md_array[:, -1]
```

In [ ]:

```
# Get the nd-array with all rows and till before the last column index
# Expected output:
# [[1, 2, 3, 4],
# [11, 22, 33, 44],
# [111, 222, 333, 444]]

dummy_md_array[:, :-1]
```

In [ ]:

```
dummy_md_array
```

In [ ]:

```
# Get the nd-array with (1st and 3rd rows) and (till before the last 2 columns) ind
# Expected output:
# [[1, 2, 3],
# [111, 222, 333]]

dummy_md_array[[0, 2],0:3]
```

In [ ]:

```
dummy_md_array[[0, 2], :-2]
```

Statistical operations on nd-arrays

Sum

In [ ]:

```
# Sum of ALL elements in the md-array
dummy_md_array.sum()
```

In [ ]:

```
# Row-wise sum of elements in md-array
dummy_md_array.sum(axis = 1)
```

In [ ]:

```
# Column-wise sum of elements in md-array
dummy_md_array.sum(axis = 0)
```

Mean

In [ ]:

```
# Mean of ALL elements in the md-array
dummy_md_array.mean()
```

In [ ]:

```
# Row-wise mean of elements in md-array
dummy_md_array.mean(axis = 1)
```

In [ ]:

```
# Column-wise mean of elements in md-array
dummy_md_array.mean(axis = 0)
```

Variance

In [ ]:

```
# Variance of ALL elements in the md-array
dummy_md_array.var()
```

In [ ]:

```
# Row-wise variance of elements in md-array
dummy_md_array.var(axis = 1)
```

In [ ]:

```
# Column-wise variance of elements in md-array
dummy_md_array.var(axis = 0)
```

Standard deviation

In [ ]:

```
# Standard-deviation of ALL elements in the md-array
dummy_md_array.std()
```

In [ ]:

```
# Row-wise Standard-deviation of elements in md-array
dummy_md_array.std(axis = 1)
```

In [ ]:

```
# Column-wise Standard-deviation of elements in md-array
dummy_md_array.std(axis = 0)
```

Add a constant number to a nd-array

```
In [ ]: dummy_md_array2 = dummy_md_array + 2.5 # Element wise operation
        dummy_md_array2
```

Summing 2 nd-arrays

```
In [ ]: print(dummy_md_array.shape)
        print(dummy_md_array2.shape)
```

```
In [ ]: dummy_md_array + dummy_md_array2
```

```
In [ ]: dummy_md_array[:, :-3] + dummy_md_array2
```

```
In [ ]: dummy_md_array - dummy_md_array2
```

```
In [ ]: cipher_np.dot(dummy_md_array[:, :-2], dummy_md_array2)
```

```
In [ ]: cipher_np.multiply(dummy_md_array[:, :-2], dummy_md_array2)
```

```
In [ ]: dummy_md_array * dummy_md_array2 # Element ise multiplication
```

```
In [ ]: cipher_np.multiply(dummy_md_array, dummy_md_array2) # Element ise multiplication
```

```
In [ ]:
```

Pandas

```
In [ ]: dummy_md_list = [[1, 2, 3, 4, 5], [11, 22, 33, 44, 55], [111, 222, 333, 444, 555]]
        dummy_md_list
```

```
In [ ]: dummy_md_array = cipher_np.array(dummy_md_list)
        dummy_md_array
```

Creating a dataframe

```
In [ ]: # cipher_pd.DataFrame(data = dummy_md_array)
        dummy_df=cipher_pd.DataFrame(dummy_md_array)
        dummy_df
```

```
In [ ]: dummy_df = cipher_pd.DataFrame(data = dummy_md_array,
                                       columns = ["Column 1", "Column 2", "Column 3", "Colu
                                       index = ["Row 1", "Row 2", "Row 3"])
        dummy_df
```

Renaming a column name

In [ ]:

```
dummy_df2 = dummy_df.rename(columns = {"Column 4": "Column Y"}) # Reassigning to a
dummy_df2
```

In [ ]:

```
dummy_df.rename(columns = {"Column 4": "Column X"}) # Renaming inplace
```

In [ ]:

```
dummy_df  #dataframe remains same as original
```

In [ ]:

```
dummy_df.rename(columns = {"Column 4": "Column X"},inplace=True)
dummy_df
```

Renaming a row name

In [ ]:

```
dummy_df.rename(index = {"Row 2": "Row X", "Row 3": "Row Y"}, inplace = True)
```

In [ ]:

```
dummy_df
```

In [ ]:

```

```

In [ ]:

```
dummy_md_array.shape
```

In [ ]:

```
dummy_df.shape
```

In [ ]:

```
dummy_md_array[1]
```

In [ ]:

```
dummy_md_array[1, 2]
```

In [ ]:

```
dummy_df.loc["Row X", "Column 3"]
```

In [ ]:

```
dummy_df.iloc[1, 2]
```

Read CSV files

In [ ]:

```
csv_df = cipher_pd.read_csv("./dummy_csv.csv",index_col=0)
csv_df
```

In [ ]:

```
# Check the column names of the dataframe
csv_df.columns
```

In [ ]:

```
# Getv the row names of the dataframe
csv_df.index
```

Reading XLSX data

In [ ]:

```
mangoes_xlsx_df = cipher_pd.read_excel("./mangoes_basket.xlsx", engine = "openpyxl"
mangoes_xlsx_df
```

In [ ]:

```
# ALL Statistical properties of the dataframe
mangoes_xlsx_df.describe()
```

In [ ]:

```
# Series
mangoes_xlsx_df["length"]
```

In [ ]:

```
print(type(mangoes_xlsx_df["length"]))
```

In [ ]:

```
mangoes_xlsx_df["length"].mean()
```

In [ ]:

```
# Handle missing values
mangoes_xlsx_df.fillna(mangoes_xlsx_df["length"].mean(), inplace = True)
```

In [ ]:

```
mangoes_xlsx_df["length"] + mangoes_xlsx_df["weight"]
```

In [ ]:

```
mangoes_xlsx_df
```

In [ ]:

```
# mng_xls2=mangoes_xlsx_df.backfill()
```

In [ ]:

```
#TRANSPOSE OPERATION
dummy_df2.T
```