Markdown language

Data Analysis with Numpy and Pandas

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 [19]:
```
# 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]
```

Out[19]:
```
array([[ 1,  2,  3],
       [11, 22, 33]])
```

In [20]:
```
dummy_md_array[0:2, 0:3]
```

Out[20]:
```
array([[ 1,  2,  3],
       [11, 22, 33]])
```

In [21]:
```
# Access the last index of the nd-array
dummy_md_array[:, -1]
```

Out[21]:
```
array([  5,  55, 555])
```

In [22]:
```
# 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]
```

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

In [23]:
```
dummy_md_array
```

Out[23]:
```
array([[  1,   2,   3,   4,   5],
       [ 11,  22,  33,  44,  55],
       [111, 222, 333, 444, 555]])
```

In [24]:
```
# 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]
```

Out[24]:
```
array([[  1,   2,   3],
       [111, 222, 333]])
```

In [25]:
```
dummy_md_array[[0, 2], :-2]
```

Out[25]:
```
array([[  1,   2,   3],
       [111, 222, 333]])
```

Statistical operations on nd-arrays

Sum

In [26]:
```
# Sum of ALL elements in the md-array
dummy_md_array.sum()
```

Out[26]:
```
1845
```

In [27]:
```
# Row-wise sum of elements in md-array
dummy_md_array.sum(axis = 1)
```

Out[27]:
```
array([  15,  165, 1665])
```

In [28]:
```
# Column-wise sum of elements in md-array
dummy_md_array.sum(axis = 0)
```

Out[28]:
```
array([123, 246, 369, 492, 615])
```

Mean

In [29]:
```
# Mean of ALL elements in the md-array
dummy_md_array.mean()
```

Out[29]:
```
123.0
```

In [30]:
```
# Row-wise mean of elements in md-array
dummy_md_array.mean(axis = 1)
```

Out[30]:
```
array([  3.,  33., 333.])
```

In [31]:
```
# Column-wise mean of elements in md-array
dummy_md_array.mean(axis = 0)
```

Out[31]:
```
array([ 41.,  82., 123., 164., 205.])
```

Variance

In [32]:
```
# Variance of ALL elements in the md-array
dummy_md_array.var()
```

Out[32]:
```
30495.333333333332
```

In [33]:
```
# Row-wise variance of elements in md-array
dummy_md_array.var(axis = 1)
```

Out[33]:
```
array([2.0000e+00, 2.4200e+02, 2.4642e+04])
```

In [34]:
```
# Column-wise variance of elements in md-array
dummy_md_array.var(axis = 0)
```

Out[34]:
```
array([ 2466.66666667,  9866.66666667, 22200.        , 39466.66666667,
       61666.66666667])
```

Standard deviation

In [35]:
```
# Standard-deviation of ALL elements in the md-array
dummy_md_array.std()
```

Out[35]:
```
174.62913082682778
```

In [36]:
```
# Row-wise Standard-deviation of elements in md-array
dummy_md_array.std(axis = 1)
```

Out[36]:
```
array([  1.41421356,  15.55634919, 156.97770542])
```

In [37]:
```
# Column-wise Standard-deviation of elements in md-array
dummy_md_array.std(axis = 0)
```

Out[37]:
```
array([ 49.66554809,  99.33109617, 148.99664426, 198.66219234,
       248.32774043])
```

Add a constant number to a nd-array

In [38]:
```
dummy_md_array2 = dummy_md_array + 2.5 # Element wise operation
dummy_md_array2
```

Out[38]:
```
array([[  3.5,   4.5,   5.5,   6.5,   7.5],
       [ 13.5,  24.5,  35.5,  46.5,  57.5],
       [113.5, 224.5, 335.5, 446.5, 557.5]])
```

Summing 2 nd-arrays

In [39]:
```
print(dummy_md_array.shape)
print(dummy_md_array2.shape)
```

```
(3, 5)
(3, 5)
```

In [40]:
```
dummy_md_array + dummy_md_array2
```

Out[40]:
```
array([[   4.5,    6.5,    8.5,   10.5,   12.5],
       [  24.5,   46.5,   68.5,   90.5,  112.5],
       [ 224.5,  446.5,  668.5,  890.5, 1112.5]])
```

In [41]:
```
dummy_md_array[:, :-3] + dummy_md_array2
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[41], line 1
----> 1 dummy_md_array[:, :-3] + dummy_md_array2

ValueError: operands could not be broadcast together with shapes (3,2) (3,5)
```

In [42]:
```
dummy_md_array - dummy_md_array2
```

Out[42]:
```
array([[-2.5, -2.5, -2.5, -2.5, -2.5],
       [-2.5, -2.5, -2.5, -2.5, -2.5],
       [-2.5, -2.5, -2.5, -2.5, -2.5]])
```

In [43]:
```
cipher_np.dot(dummy_md_array[:, :-2], dummy_md_array2)
```

Out[43]:
```
array([[   371.,    727.,   1083.,   1439.,   1795.],
       [  4081.,   7997.,  11913.,  15829.,  19745.],
       [ 41181.,  80697., 120213., 159729., 199245.]])
```

In [44]:
```
cipher_np.multiply(dummy_md_array[:, :-2], dummy_md_array2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[44], line 1
----> 1 cipher_np.multiply(dummy_md_array[:, :-2], dummy_md_array2)

ValueError: operands could not be broadcast together with shapes (3,3) (3,5)
```

In [45]:
```
dummy_md_array * dummy_md_array2 # Element ise multiplication
```

Out[45]:
```
array([[3.500000e+00, 9.000000e+00, 1.650000e+01, 2.600000e+01,
        3.750000e+01],
       [1.485000e+02, 5.390000e+02, 1.171500e+03, 2.046000e+03,
        3.162500e+03],
       [1.259850e+04, 4.983900e+04, 1.117215e+05, 1.982460e+05,
        3.094125e+05]])
```

In [46]:
```
cipher_np.multiply(dummy_md_array, dummy_md_array2) # Element ise multiplication
```

Out[46]:
```
array([[3.500000e+00, 9.000000e+00, 1.650000e+01, 2.600000e+01,
        3.750000e+01],
       [1.485000e+02, 5.390000e+02, 1.171500e+03, 2.046000e+03,
        3.162500e+03],
       [1.259850e+04, 4.983900e+04, 1.117215e+05, 1.982460e+05,
        3.094125e+05]])
```

In [ ]:

Pandas

In [47]:
```
dummy_md_list = [[1, 2, 3, 4, 5], [11, 22, 33, 44, 55], [111, 222, 333, 444, 555]]
dummy_md_list
```

Out[47]:
```
[[1, 2, 3, 4, 5], [11, 22, 33, 44, 55], [111, 222, 333, 444, 555]]
```

In [48]:
```
dummy_md_array = cipher_np.array(dummy_md_list)
dummy_md_array
```

Out[48]:
```
array([[  1,   2,   3,   4,   5],
       [ 11,  22,  33,  44,  55],
       [111, 222, 333, 444, 555]])
```

Creating a dataframe

In [49]:
```
# cipher_pd.DataFrame(data = dummy_md_array)
dummy_df=cipher_pd.DataFrame(dummy_md_array)
dummy_df
```

Out[49]:

	0	1	2	3	4
0	1	2	3	4	5
1	11	22	33	44	55
2	111	222	333	444	555

In [50]:
```
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
```

Out[50]:

	Column 1	Column 2	Column 3	Column 4	Column 5
Row 1	1	2	3	4	5
Row 2	11	22	33	44	55
Row 3	111	222	333	444	555

Renaming a column name

In [51]:
```
dummy_df2 = dummy_df.rename(columns = {"Column 4": "Column Y"}) # Reassigning to a
dummy_df2
```

Out[51]:

	Column 1	Column 2	Column 3	Column Y	Column 5
Row 1	1	2	3	4	5
Row 2	11	22	33	44	55
Row 3	111	222	333	444	555

In [52]:
```
dummy_df.rename(columns = {"Column 4": "Column X"}) # Renaming inplace
```

Out[52]:

	Column 1	Column 2	Column 3	Column X	Column 5
Row 1	1	2	3	4	5
Row 2	11	22	33	44	55
Row 3	111	222	333	444	555

In [53]:
```
dummy_df  #dataframe remains same as original
```

Out[53]:

	Column 1	Column 2	Column 3	Column 4	Column 5
Row 1	1	2	3	4	5
Row 2	11	22	33	44	55
Row 3	111	222	333	444	555

In [54]:
```
dummy_df.rename(columns = {"Column 4": "Column X"},inplace=True)
dummy_df
```

Out[54]:

	Column 1	Column 2	Column 3	Column X	Column 5
Row 1	1	2	3	4	5
Row 2	11	22	33	44	55
Row 3	111	222	333	444	555

Renaming a row name

In [55]:
```
dummy_df.rename(index = {"Row 2": "Row X", "Row 3": "Row Y"}, inplace = True)
```

In [56]:
```
dummy_df
```

Out[56]:

	Column 1	Column 2	Column 3	Column X	Column 5
Row 1	1	2	3	4	5
Row X	11	22	33	44	55
Row Y	111	222	333	444	555

In [ ]:
```

```

In [ ]:

In [57]:
```
dummy_md_array.shape
```

Out[57]:
```
(3, 5)
```

In [58]:
```
dummy_df.shape
```

Out[58]:
```
(3, 5)
```

In [59]:
```
dummy_md_array[1]
```

Out[59]:
```
array([11, 22, 33, 44, 55])
```

In [60]:
```
dummy_md_array[1, 2]
```

Out[60]:
```
33
```

In [75]:
```
dummy_df.loc["Row X", "Column 3"]
```

Out[75]:
```
33
```

In [76]:
```
dummy_df.iloc[1, 2]
```

Out[76]:
```
33
```

Read CSV files

In [61]:
```
csv_df = cipher_pd.read_csv("./dummy_csv.csv",index_col=0)
csv_df
```

Out[61]:

	Col1	col2	col3	col4	col5	col6	col7
Row 1	1.0	2.0	3.0	4.0	5.0	6.0	7.0
Row 2	11.0	22.0	33.0	44.0	55.0	66.0	77.0
Row 3	111.0	222.0	333.0	444.0	555.0	666.0	777.0
Row 4	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [62]:
```
# Check the column names of the dataframe
csv_df.columns
```

Out[62]:
```
Index(['Col1', 'col2', 'col3', 'col4', 'col5', 'col6', 'col7'], dtype='object')
```

In [63]:
```
# Getv the row names of the dataframe
csv_df.index
```

Out[63]:
```
Index(['Row 1', 'Row 2', 'Row 3', 'Row 4'], dtype='object')
```

Reading XLSX data

In [65]:
```
mangoes_xlsx_df = cipher_pd.read_excel("./mangoes_basket.xlsx", engine = "openpyxl"
mangoes_xlsx_df
```

Out[65]:

	weight	length	diameter	age
Mango 1	1.0	10.0	10	1
Mango 2	1.4	12.0	21	2
Mango 3	1.1	1.0	11	5
Mango 4	332.0	3.0	13	7
Mango 5	1.3	NaN	9	1
Mango 6	1.6	NaN	10	3
Mango 7	1.7	12.0	11	5
Mango 8	1.9	34.0	22	4
Mango 9	1.8	23.0	33	0
Mango 10	1.2	21.0	13	10

In [66]:
```
# ALL Statistical properties of the dataframe
mangoes_xlsx_df.describe()
```

Out[66]:

	weight	length	diameter	age
count	10.000000	8.000000	10.000000	10.000000
mean	34.500000	14.500000	15.300000	3.800000
std	104.531282	10.967484	7.703535	3.084009
min	1.000000	1.000000	9.000000	0.000000
25%	1.225000	8.250000	10.250000	1.250000
50%	1.500000	12.000000	12.000000	3.500000
75%	1.775000	21.500000	19.000000	5.000000
max	332.000000	34.000000	33.000000	10.000000

In [67]:
```
# Series
mangoes_xlsx_df["length"]
```

Out[67]:
```
Mango 1     10.0
Mango 2     12.0
Mango 3      1.0
Mango 4      3.0
Mango 5      NaN
Mango 6      NaN
Mango 7     12.0
Mango 8     34.0
Mango 9     23.0
Mango 10    21.0
Name: length, dtype: float64
```

In [68]:
```
print(type(mangoes_xlsx_df["length"]))
```

```
<class 'pandas.core.series.Series'>
```

In [69]:
```
mangoes_xlsx_df["length"].mean()
```

Out[69]:
```
14.5
```

In [70]:
```
# Handle missing values
mangoes_xlsx_df.fillna(mangoes_xlsx_df["length"].mean(), inplace = True)
```

In [71]:
```
mangoes_xlsx_df["length"] + mangoes_xlsx_df["weight"]
```

Out[71]:
```
Mango 1      11.0
Mango 2      13.4
Mango 3       2.1
Mango 4     335.0
Mango 5      15.8
Mango 6      16.1
Mango 7      13.7
Mango 8      35.9
Mango 9      24.8
Mango 10     22.2
dtype: float64
```

In [72]:
```
mangoes_xlsx_df
```

Out[72]:

	weight	length	diameter	age
Mango 1	1.0	10.0	10	1
Mango 2	1.4	12.0	21	2
Mango 3	1.1	1.0	11	5
Mango 4	332.0	3.0	13	7
Mango 5	1.3	14.5	9	1
Mango 6	1.6	14.5	10	3
Mango 7	1.7	12.0	11	5
Mango 8	1.9	34.0	22	4
Mango 9	1.8	23.0	33	0
Mango 10	1.2	21.0	13	10

In [73]:
```
# mng_xls2=mangoes_xlsx_df.backfill()
```

In [74]:
```
#TRANSPOSE OPERATION
dummy_df2.T
```

Out[74]:

	Row 1	Row 2	Row 3
Column 1	1	11	111
Column 2	2	22	222
Column 3	3	33	333
Column Y	4	44	444
Column 5	5	55	555