In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
s = pd.Series([1,2,3],index=['a','b','c'],name='col1') #basic arguments are data ,column name and index names
```

In [3]:
```
s
```

Out[3]:
```
a    1
b    2
c    3
Name: col1, dtype: int64
```

In [4]:
```
df = pd.DataFrame([[1,2],[2,3],[3,4]],columns=['name','age'],index=['a','b','c']) #basic arguments are data ,column name and inde
```

In [5]:
```
df
```

Out[5]:

	name	age
a	1	2
b	2	3
c	3	4

In [6]:
```
# Loading a csv file ---- > arguments are inportant
#df = pd.read_csv('filename.csv',sep=',',names=['col1','col2'],index_col=0,encoding='utf-8',nrows=3)
```

In [7]:
```
#lets create a dataframe to practice more aggragate functions
```

In [8]:
```
data = np.random.random((10,5))
```

In [9]:
```
df = pd.DataFrame(data,columns=['col1','col2','col3','col4','col5'])
df
```

Out[9]:

	col1	col2	col3	col4	col5
0	0.982447	0.018281	0.860636	0.245460	0.622913
1	0.204444	0.322217	0.461457	0.647450	0.310468
2	0.017723	0.969699	0.644519	0.760223	0.250974
3	0.883462	0.231170	0.554110	0.700903	0.971225
4	0.532958	0.232694	0.600471	0.212731	0.017684
5	0.426037	0.196835	0.958470	0.879994	0.090494
6	0.346463	0.520383	0.683032	0.167199	0.063369
7	0.728574	0.789305	0.645111	0.939384	0.388174
8	0.116325	0.658116	0.702703	0.888054	0.735919
9	0.113493	0.208251	0.802944	0.228663	0.614883

In [10]:
```
np.array(df) #we can also convert dataframe to numpy array
```

Out[10]:
```
array([[0.98244687, 0.018281  , 0.86063613, 0.24546023, 0.62291317],
       [0.20444353, 0.32221743, 0.46145689, 0.6474499 , 0.31046774],
       [0.01772277, 0.96969919, 0.64451933, 0.76022325, 0.250974  ],
       [0.88346232, 0.23117024, 0.55411033, 0.70090279, 0.97122529],
       [0.53295756, 0.23269389, 0.6004709 , 0.21273128, 0.01768416],
       [0.42603743, 0.19683509, 0.95847037, 0.87999395, 0.09049379],
       [0.3464626 , 0.52038332, 0.68303157, 0.16719894, 0.06336879],
       [0.72857406, 0.78930481, 0.6451115 , 0.93938429, 0.38817368],
       [0.1163254 , 0.65811594, 0.70270331, 0.88805449, 0.73591852],
       [0.11349281, 0.20825113, 0.8029436 , 0.22866344, 0.61488266]])
```

In [11]:
```
type(df['col2']) # single column -- >this is series
df['col2']
```

Out[11]:
```
0    0.018281
1    0.322217
2    0.969699
3    0.231170
4    0.232694
5    0.196835
6    0.520383
7    0.789305
8    0.658116
9    0.208251
Name: col2, dtype: float64
```

In [12]:
```
# df['col1','col2'] -----> this is wrong approach
# for more than 1 column
```

```
type(df[['col1','col2']]) #this is dataframe
df[['col1','col2']]
```

Out[12]:

	col1	col2
0	0.982447	0.018281
1	0.204444	0.322217
2	0.017723	0.969699
3	0.883462	0.231170
4	0.532958	0.232694
5	0.426037	0.196835
6	0.346463	0.520383
7	0.728574	0.789305
8	0.116325	0.658116
9	0.113493	0.208251

In [13]:
```
df.head(4)
```

Out[13]:

	col1	col2	col3	col4	col5
0	0.982447	0.018281	0.860636	0.245460	0.622913
1	0.204444	0.322217	0.461457	0.647450	0.310468
2	0.017723	0.969699	0.644519	0.760223	0.250974
3	0.883462	0.231170	0.554110	0.700903	0.971225

In [14]:
```
df.tail(3)
```

Out[14]:

	col1	col2	col3	col4	col5
7	0.728574	0.789305	0.645111	0.939384	0.388174
8	0.116325	0.658116	0.702703	0.888054	0.735919
9	0.113493	0.208251	0.802944	0.228663	0.614883

In [15]:
```
# SELECTING ROWS AND COLUMNS
# for numpy arrays and lists we can select directly but for data frame we have to use loc() function
```

In [16]:
```
df.loc[3] #we cant use df[3]
# VERY IMPORTANT--> df.loc[rows]['columns'] --->this is how we can get data by slicing or setting
```

Out[16]:
```
col1    0.883462
col2    0.231170
col3    0.554110
col4    0.700903
col5    0.971225
Name: 3, dtype: float64
```

In [17]:
```
df.loc[3]['col3']
```

Out[17]:
```
0.5541103306595284
```

In [18]:
```
df.loc[:]['col1'] #----> return series
df.loc[:][['col1','col2']] #-----> return dataframe
```

Out[18]:

	col1	col2
0	0.982447	0.018281
1	0.204444	0.322217
2	0.017723	0.969699
3	0.883462	0.231170
4	0.532958	0.232694
5	0.426037	0.196835
6	0.346463	0.520383
7	0.728574	0.789305
8	0.116325	0.658116
9	0.113493	0.208251

In [19]:
```
df.loc[:]
```

Out[19]:

	col1	col2	col3	col4	col5
0	0.982447	0.018281	0.860636	0.245460	0.622913
1	0.204444	0.322217	0.461457	0.647450	0.310468
2	0.017723	0.969699	0.644519	0.760223	0.250974
3	0.883462	0.231170	0.554110	0.700903	0.971225
4	0.532958	0.232694	0.600471	0.212731	0.017684

5	0.426037	0.196835	0.958470	0.879994	0.090494
6	0.346463	0.520383	0.683032	0.167199	0.063369
7	0.728574	0.789305	0.645111	0.939384	0.388174
8	0.116325	0.658116	0.702703	0.888054	0.735919
9	0.113493	0.208251	0.802944	0.228663	0.614883

In [20]:
```
# DATA WRANGLING
```

In [21]:
```
# CONDITIONAL FILTERING
df[df['col2']>0.533] # this is very important
```

Out[21]:

	col1	col2	col3	col4	col5
2	0.017723	0.969699	0.644519	0.760223	0.250974
7	0.728574	0.789305	0.645111	0.939384	0.388174
8	0.116325	0.658116	0.702703	0.888054	0.735919

In [22]:
```
a = np.array(df)
a[a>0.5][0]
```

Out[22]:
```
0.9824468683572142
```

In [23]:
```
df.duplicated()
```

Out[23]:
```
0    False
1    False
2    False
3    False
4    False
5    False
6    False
7    False
8    False
9    False
dtype: bool
```

In [24]:
```
df.sort_values(['col1'])
```

Out[24]:

	col1	col2	col3	col4	col5
2	0.017723	0.969699	0.644519	0.760223	0.250974
9	0.113493	0.208251	0.802944	0.228663	0.614883
8	0.116325	0.658116	0.702703	0.888054	0.735919
1	0.204444	0.322217	0.461457	0.647450	0.310468
6	0.346463	0.520383	0.683032	0.167199	0.063369
5	0.426037	0.196835	0.958470	0.879994	0.090494
4	0.532958	0.232694	0.600471	0.212731	0.017684
7	0.728574	0.789305	0.645111	0.939384	0.388174
3	0.883462	0.231170	0.554110	0.700903	0.971225
0	0.982447	0.018281	0.860636	0.245460	0.622913

In [25]:
```
df.T
```

Out[25]:

	0	1	2	3	4	5	6	7	8	9
col1	0.982447	0.204444	0.017723	0.883462	0.532958	0.426037	0.346463	0.728574	0.116325	0.113493
col2	0.018281	0.322217	0.969699	0.231170	0.232694	0.196835	0.520383	0.789305	0.658116	0.208251
col3	0.860636	0.461457	0.644519	0.554110	0.600471	0.958470	0.683032	0.645111	0.702703	0.802944
col4	0.245460	0.647450	0.760223	0.700903	0.212731	0.879994	0.167199	0.939384	0.888054	0.228663
col5	0.622913	0.310468	0.250974	0.971225	0.017684	0.090494	0.063369	0.388174	0.735919	0.614883

In [26]:
```
df['col1'].unique() #unique values
```

Out[26]:
```
array([0.98244687, 0.20444353, 0.01772277, 0.88346232, 0.53295756,
       0.42603743, 0.3464626 , 0.72857406, 0.1163254 , 0.11349281])
```

In [27]:
```
df.drop(['col3','col4'],axis=1)
```

Out[27]:

	col1	col2	col5
0	0.982447	0.018281	0.622913
1	0.204444	0.322217	0.310468
2	0.017723	0.969699	0.250974
3	0.883462	0.231170	0.971225
4	0.532958	0.232694	0.017684
5	0.426037	0.196835	0.090494

6	0.346463	0.520383	0.063369
7	0.728574	0.789305	0.388174
8	0.116325	0.658116	0.735919
9	0.113493	0.208251	0.614883

In [28]:
```
df+5 # add 5 to every element ,+,-,*,**,/
```

Out[28]:

	col1	col2	col3	col4	col5
0	5.982447	5.018281	5.860636	5.245460	5.622913
1	5.204444	5.322217	5.461457	5.647450	5.310468
2	5.017723	5.969699	5.644519	5.760223	5.250974
3	5.883462	5.231170	5.554110	5.700903	5.971225
4	5.532958	5.232694	5.600471	5.212731	5.017684
5	5.426037	5.196835	5.958470	5.879994	5.090494
6	5.346463	5.520383	5.683032	5.167199	5.063369
7	5.728574	5.789305	5.645111	5.939384	5.388174
8	5.116325	5.658116	5.702703	5.888054	5.735919
9	5.113493	5.208251	5.802944	5.228663	5.614883

In [29]:
```
df2 = df
pd.concat([df,df2])
```

Out[29]:

	col1	col2	col3	col4	col5
0	0.982447	0.018281	0.860636	0.245460	0.622913
1	0.204444	0.322217	0.461457	0.647450	0.310468
2	0.017723	0.969699	0.644519	0.760223	0.250974
3	0.883462	0.231170	0.554110	0.700903	0.971225
4	0.532958	0.232694	0.600471	0.212731	0.017684
5	0.426037	0.196835	0.958470	0.879994	0.090494
6	0.346463	0.520383	0.683032	0.167199	0.063369
7	0.728574	0.789305	0.645111	0.939384	0.388174
8	0.116325	0.658116	0.702703	0.888054	0.735919
9	0.113493	0.208251	0.802944	0.228663	0.614883
0	0.982447	0.018281	0.860636	0.245460	0.622913
1	0.204444	0.322217	0.461457	0.647450	0.310468
2	0.017723	0.969699	0.644519	0.760223	0.250974
3	0.883462	0.231170	0.554110	0.700903	0.971225
4	0.532958	0.232694	0.600471	0.212731	0.017684
5	0.426037	0.196835	0.958470	0.879994	0.090494
6	0.346463	0.520383	0.683032	0.167199	0.063369
7	0.728574	0.789305	0.645111	0.939384	0.388174
8	0.116325	0.658116	0.702703	0.888054	0.735919
9	0.113493	0.208251	0.802944	0.228663	0.614883

In [30]:
```
pd.concat([df,df2],axis=1)
```

Out[30]:

	col1	col2	col3	col4	col5	col1	col2	col3	col4	col5
0	0.982447	0.018281	0.860636	0.245460	0.622913	0.982447	0.018281	0.860636	0.245460	0.622913
1	0.204444	0.322217	0.461457	0.647450	0.310468	0.204444	0.322217	0.461457	0.647450	0.310468
2	0.017723	0.969699	0.644519	0.760223	0.250974	0.017723	0.969699	0.644519	0.760223	0.250974
3	0.883462	0.231170	0.554110	0.700903	0.971225	0.883462	0.231170	0.554110	0.700903	0.971225
4	0.532958	0.232694	0.600471	0.212731	0.017684	0.532958	0.232694	0.600471	0.212731	0.017684
5	0.426037	0.196835	0.958470	0.879994	0.090494	0.426037	0.196835	0.958470	0.879994	0.090494
6	0.346463	0.520383	0.683032	0.167199	0.063369	0.346463	0.520383	0.683032	0.167199	0.063369
7	0.728574	0.789305	0.645111	0.939384	0.388174	0.728574	0.789305	0.645111	0.939384	0.388174
8	0.116325	0.658116	0.702703	0.888054	0.735919	0.116325	0.658116	0.702703	0.888054	0.735919
9	0.113493	0.208251	0.802944	0.228663	0.614883	0.113493	0.208251	0.802944	0.228663	0.614883

In [31]:
```
copy = df.copy()
copy
```

Out[31]:

	col1	col2	col3	col4	col5
0	0.982447	0.018281	0.860636	0.245460	0.622913
1	0.204444	0.322217	0.461457	0.647450	0.310468

2	0.017723	0.969699	0.644519	0.760223	0.250974
3	0.883462	0.231170	0.554110	0.700903	0.971225
4	0.532958	0.232694	0.600471	0.212731	0.017684
5	0.426037	0.196835	0.958470	0.879994	0.090494
6	0.346463	0.520383	0.683032	0.167199	0.063369
7	0.728574	0.789305	0.645111	0.939384	0.388174
8	0.116325	0.658116	0.702703	0.888054	0.735919
9	0.113493	0.208251	0.802944	0.228663	0.614883

In [32]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [33]:
```
df = sns.load_dataset('iris')
df.head()
```

Out[33]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [34]:
```
# df = df.drop(['petal_width', 'sepal_length'], axis=1)
```

In [35]:
```
def get_outliers(data):
    threshhold = 3 # all the values after 3rd deviation are outliers
    std=np.std(data)
    mean = np.mean(data)
    outliers = []
    for i in data:
        z_score = (i-mean)/std
        if np.abs(z_score)>3:
            outliers.append(i)
    return outliers
```

In [36]:
```
get_outliers(df['sepal_width']) # we have only one outliers
```

Out[36]:
```
[4.4]
```

In [37]:
```
df = df.drop(df[df['sepal_width']==4.4].index,axis=0)
```

In [38]:
```
df.head(20)
```

Out[38]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa
10	5.4	3.7	1.5	0.2	setosa
11	4.8	3.4	1.6	0.2	setosa
12	4.8	3.0	1.4	0.1	setosa
13	4.3	3.0	1.1	0.1	setosa
14	5.8	4.0	1.2	0.2	setosa
16	5.4	3.9	1.3	0.4	setosa
17	5.1	3.5	1.4	0.3	setosa
18	5.7	3.8	1.7	0.3	setosa
19	5.1	3.8	1.5	0.3	setosa
20	5.4	3.4	1.7	0.2	setosa

In [39]:
```
df['sepal_width'].cumsum() #can be use in progress
```

Out[39]:
```
0     3.5
1     6.5
2     9.7
3    12.8
4    16.4
```

```
...
145    442.3
146    444.8
147    447.8
148    451.2
149    454.2
Name: sepal_width, Length: 149, dtype: float64
```

In [40]:
```
df['sepal_width'].sum()
```

Out[40]:
```
454.20000000000005
```

In [41]:
```
df['sepal_width'].sum().sum()
```

Out[41]:
```
454.20000000000005
```

In [42]:
```
df.mean()
```

```
C:\Users\manis\AppData\Local\Temp\ipykernel_16348\3698961737.py:1: FutureWarning: Dropping of nuisance columns in DataFrame red
uctions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns bef
ore calling the reduction.
  df.mean()
```

Out[42]:
```
sepal_length    5.844295
sepal_width     3.048322
petal_length    3.773154
petal_width     1.204698
dtype: float64
```

In [43]:
```
df.median() #in column petal_length mean is 3.773154 but median is 4.4 so we can say that here is a big outlier
```

```
C:\Users\manis\AppData\Local\Temp\ipykernel_16348\431873348.py:1: FutureWarning: Dropping of nuisance columns in DataFrame redu
ctions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns befo
re calling the reduction.
  df.median() #in column petal_length mean is 3.773154 but median is 4.4 so we can say that here is a big outlier
```

Out[43]:
```
sepal_length    5.8
sepal_width     3.0
petal_length    4.4
petal_width     1.3
dtype: float64
```

In [44]:
```
df.std()
```

```
C:\Users\manis\AppData\Local\Temp\ipykernel_16348\3390915376.py:1: FutureWarning: Dropping of nuisance columns in DataFrame red
uctions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns bef
ore calling the reduction.
  df.std()
```

Out[44]:
```
sepal_length    0.830775
sepal_width     0.423085
petal_length    1.761435
petal_width     0.761962
dtype: float64
```

In [45]:
```
df.sum()
```

Out[45]:
```
sepal_length                                                870.8
sepal_width                                                 454.2
petal_length                                                562.2
petal_width                                                 179.5
species         setosasetosasetosasetosasetosasetosasetosaseto...
dtype: object
```

In [46]:
```
df.cumsum()
```

Out[46]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	10.0	6.5	2.8	0.4	setosasetosa
2	14.7	9.7	4.1	0.6	setosasetosasetosa
3	19.3	12.8	5.6	0.8	setosasetosasetosasetosa
4	24.3	16.4	7.0	1.0	setosasetosasetosasetosasetosa
...	...	...	...	...	...
145	845.9	442.3	541.5	171.5	setosasetosasetosasetosasetosasetosasetosaseto...
146	852.2	444.8	546.5	173.4	setosasetosasetosasetosasetosasetosasetosaseto...
147	858.7	447.8	551.7	175.4	setosasetosasetosasetosasetosasetosasetosaseto...
148	864.9	451.2	557.1	177.7	setosasetosasetosasetosasetosasetosasetosaseto...
149	870.8	454.2	562.2	179.5	setosasetosasetosasetosasetosasetosasetosaseto...

149 rows × 5 columns

In [47]:
```
df.value_counts()
```

Out[47]:
```
sepal_length  sepal_width  petal_length  petal_width  species
5.8           2.7          5.1           1.9          virginica     2
6.2           2.2          4.5           1.5          versicolor    1
```

```
              2.9           4.3           1.3          versicolor    1
              3.4           5.4           2.3          virginica     1
6.3           2.3           4.4           1.3          versicolor    1
                                                                   ..
5.4           3.7           1.5           0.2          setosa        1
              3.9           1.3           0.4          setosa        1
                            1.7           0.4          setosa        1
5.5           2.3           4.0           1.3          versicolor    1
7.9           3.8           6.4           2.0          virginica     1
Length: 148, dtype: int64
```

In [48]:
```
df['sepal_width'].value_counts().index,df['sepal_width'].value_counts().values
```

Out[48]:
```
(Float64Index([3.0, 2.8, 3.2, 3.4, 3.1, 2.9, 2.7, 2.5, 3.5, 3.8, 3.3, 2.6, 2.3,
               3.6, 3.7, 2.4, 2.2, 3.9, 4.0, 4.1, 4.2, 2.0],
              dtype='float64'),
 array([26, 14, 13, 12, 11, 10,  9,  8,  6,  6,  6,  5,  4,  4,  3,  3,  3,
         2,  1,  1,  1,  1], dtype=int64))
```

In [49]:
```
df.describe() # statistics
```

Out[49]:

	sepal_length	sepal_width	petal_length	petal_width
count	149.000000	149.000000	149.000000	149.000000
mean	5.844295	3.048322	3.773154	1.204698
std	0.830775	0.423085	1.761435	0.761962
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.400000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.200000	6.900000	2.500000

In [50]:
```
df.stack()
```

Out[50]:
```
0    sepal_length          5.1
     sepal_width           3.5
     petal_length          1.4
     petal_width           0.2
     species            setosa
                        ...
149  sepal_length          5.9
     sepal_width           3.0
     petal_length          5.1
     petal_width           1.8
     species         virginica
Length: 745, dtype: object
```

In [51]:
```
df2 = df.stack()
df2
```

Out[51]:
```
0    sepal_length          5.1
     sepal_width           3.5
     petal_length          1.4
     petal_width           0.2
     species            setosa
                        ...
149  sepal_length          5.9
     sepal_width           3.0
     petal_length          5.1
     petal_width           1.8
     species         virginica
Length: 745, dtype: object
```

In [52]:
```
df2.unstack()
```

Out[52]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

149 rows × 5 columns

In [53]:
```
#AGGREGATION
```

In [54]:
```
group = df.groupby('species')
```

In [55]:
```
group.sum()
```

Out[55]:

species	sepal_length	sepal_width	petal_length	petal_width
setosa	244.6	167.0	71.6	11.9
versicolor	296.8	138.5	213.0	66.3
virginica	329.4	148.7	277.6	101.3

In [56]:
```
# we can also describe our data on the basis of classes of a perticular column
group.describe() # VERY VERY IMPORTANT FUNCTIONALITY
```

Out[56]:

	sepal_length								sepal_width		...	petal_length		petal_width						
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean	std	min	25%	50%	75%	max
species																				
setosa	49.0	4.991837	0.341465	4.3	4.800	5.0	5.2	5.8	49.0	3.408163	...	1.600	1.9	49.0	0.242857	0.104083	0.1	0.2	0.2	0.3	0.6
versicolor	50.0	5.936000	0.516171	4.9	5.600	5.9	6.3	7.0	50.0	2.770000	...	4.600	5.1	50.0	1.326000	0.197753	1.0	1.2	1.3	1.5	1.8
virginica	50.0	6.588000	0.635880	4.9	6.225	6.5	6.9	7.9	50.0	2.974000	...	5.875	6.9	50.0	2.026000	0.274650	1.4	1.8	2.0	2.3	2.5

3 rows × 32 columns

In [57]:
```
print(group.mean(),group.prod(),group.std())
```

```
            sepal_length  sepal_width  petal_length  petal_width
species
setosa          4.991837     3.408163      1.461224     0.242857
versicolor      5.936000     2.770000      4.260000     1.326000
virginica       6.588000     2.974000      5.552000     2.026000            sepal_length   sepal_width  petal_length   petal_w
idth
species
setosa      1.465079e+34  9.455094e+25  8.284399e+07  1.442779e-32
versicolor  3.926361e+38  9.581281e+21  2.151931e+31  7.642546e+05
virginica   6.884782e+40  3.488717e+23  1.317390e+37  1.347986e+15          sepal_length  sepal_width  petal_length  petal_w
idth
species
setosa          0.341465     0.355807      0.175376     0.104083
versicolor      0.516171     0.313798      0.469911     0.197753
virginica       0.635880     0.322497      0.551895     0.274650
```

In [58]:
```
group['sepal_width'].sum() #important to know
```

Out[58]:
```
species
setosa        167.0
versicolor    138.5
virginica     148.7
Name: sepal_width, dtype: float64
```

In [59]:
```
df['sepal_width'].apply(lambda a:a+12) #applying functonality to every element of a column
```

Out[59]:
```
0      15.5
1      15.0
2      15.2
3      15.1
4      15.6
       ...
145    15.0
146    14.5
147    15.0
148    15.4
149    15.0
Name: sepal_width, Length: 149, dtype: float64
```

In [60]:
```
# their are many data export funcions like to_csv,to_dict,to_string,to_axcel
```

In [61]:
```
df.fillna(0)
```

Out[61]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica

149	5.9	3.0	5.1	1.8	virginica

149 rows × 5 columns

In [62]:
```
df3 = pd.DataFrame([[1,2,3,5,'kalu'],[2,3,4,5,'ram']],index=['a','b'],columns=df.columns)
```

In [63]:
```
df3
```

Out[63]:

	sepal_length	sepal_width	petal_length	petal_width	species
a	1	2	3	5	kalu
b	2	3	4	5	ram

In [64]:
```
df.merge(df3,how='outer')
```

```
C:\Users\manis\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\reshape\merge.py:1207: UserWarning: You ar
e merging on int and float columns where the float values are not equal to their int representation.
  warnings.warn(
```

Out[64]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
146	6.5	3.0	5.2	2.0	virginica
147	6.2	3.4	5.4	2.3	virginica
148	5.9	3.0	5.1	1.8	virginica
149	1.0	2.0	3.0	5.0	kalu
150	2.0	3.0	4.0	5.0	ram

151 rows × 5 columns

In [ ]:
```

```