Practical Tutorial on Data Manipulation with Numpy and Pandas in Python

Numpy

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

suppose we have a list

In [2]:
```
l=list(range(1,10))
print(l)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Convert list into numpy

In [3]:
```
n1=np.array(l)
print(n1)
```

```
[1 2 3 4 5 6 7 8 9]
```

In [4]:
```
type(n1)
```

Out[4]:
```
numpy.ndarray
```

creating NumPy Using arange

In [5]:
```
n2=np.arange(2,30)
print(n2)
```

```
[ 2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
 26 27 28 29]
```

we can setup step too,

In [6]:
```
n2=np.arange(2,30,2)
print(n2)
```

```
[ 2  4  6  8 10 12 14 16 18 20 22 24 26 28]
```

Creating 3 row 5 column matrix using numpy array

In [7]:
```
n3=np.arange(1,16).reshape(3,5)       #inside array element should be from 1 to 15, exclusive of 16.
print(n3)
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]]
```

What happen if we mention 15 instead of 16?

In [8]:
```
n4=np.arange(1,15).reshape(3,5)
print(n4)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[8], line 1
----> 1 n4=np.arange(1,15).reshape(3,5)
      2 print(n4)

ValueError: cannot reshape array of size 14 into shape (3,5)
```

We will get error beacuse we did not give required number of elements

Other ways to create array

In [9]:
```
np.zeros(10, dtype='int')
```

Out[9]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [10]:
```
np.zeros(10, dtype='float')
```

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

In [11]:
```
np.ones((3,5), dtype='int')
```

Out[11]:
```
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]])
```

In [12]:
```
np.zeros((3,5),dtype='float')
```

Out[12]:

```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [13]:

```
#creating a matrix with a predefined value
np.full((3,5),1.23)
```

Out[13]:

```
array([[1.23, 1.23, 1.23, 1.23, 1.23],
       [1.23, 1.23, 1.23, 1.23, 1.23],
       [1.23, 1.23, 1.23, 1.23, 1.23]])
```

In [14]:

```
np.full((2,2),6)
```

Out[14]:

```
array([[6, 6],
       [6, 6]])
```

Using random

In [15]:

```
x1 = np.random.randint(10, size=6) #one dimension
x2 = np.random.randint(10, size=(3,4)) #two dimension
x3 = np.random.randint(10, size=(3,4,5)) #three dimension
```

In [16]:

```
print(x1)
```

```
[2 8 8 0 7 6]
```

In [17]:

```
print(x2)
```

```
[[2 5 3 4]
 [2 2 1 7]
 [8 5 0 2]]
```

In [18]:

```
print(x3)
```

```
[[[0 4 0 3 8]
  [2 3 2 5 8]
  [6 2 4 3 9]
  [4 7 4 5 1]]

 [[9 3 2 9 4]
  [4 3 7 2 9]
  [5 0 5 0 3]
  [4 9 9 4 0]]

 [[7 7 9 5 9]
  [6 9 2 5 4]
  [2 7 9 8 7]
  [8 5 9 7 9]]]
```

Array indexing and slicing

In [19]:

```
n3
```

Out[19]:

```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15]])
```

In [20]:

```
n3[0]      #zero position row
```

Out[20]:

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

In [21]:

```
n3[0,2]       #zero row and second column will give output of one element
```

Out[21]:

```
3
```

In [22]:

```
n3[0:2,1:3] #zero to one row plus 1 to 2nd column
```

Out[22]:

```
array([[2, 3],
       [7, 8]])
```

In [23]:

```
n3[:]
```

Out[23]:

```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15]])
```

In [24]:

```
n3[::-1]     #reverse the array
```

Out[24]:

```
array([[11, 12, 13, 14, 15],
       [ 6,  7,  8,  9, 10],
       [ 1,  2,  3,  4,  5]])
```

Array Concatenation

In [25]:

```
x = np.array([1, 2, 3])
y = np.array([3, 2, 1])
z = [21,21,21]
```

```
np.concatenate([x, y,z])
```

Out[25]:
```
array([ 1,  2,  3,  3,  2,  1, 21, 21, 21])
```

CONCATENATING 2D ARRAY

In [27]:
```
n2.reshape(2,7)
```

Out[27]:
```
array([[ 2,  4,  6,  8, 10, 12, 14],
       [16, 18, 20, 22, 24, 26, 28]])
```

In [34]:
```
n2.ndim
```

Out[34]:
```
1
```

In [30]:
```
n1=np.random.randint(15,size=(2,7))
print(n1)
```

```
[[ 3 12 12 13  7  5  0]
 [12  8  9 12  2  1 12]]
```

In [36]:
```
n1.ndim
```

Out[36]:
```
2
```

In [39]:
```
#we need another 2 D array
n2=np.random.randint(15,size=(2,7))
print(n2)
```

```
[[ 7  5 14 11  7 10  6]
 [ 8  2 13  2  2  5 14]]
```

In [41]:
```
np.concatenate([n1,n2])
```

Out[41]:
```
array([[ 3, 12, 12, 13,  7,  5,  0],
       [12,  8,  9, 12,  2,  1, 12],
       [ 7,  5, 14, 11,  7, 10,  6],
       [ 8,  2, 13,  2,  2,  5, 14]])
```

In [43]:
```
#lets create 2 D numpy array to look shape

a=np.array([[1,2,3,4],[5,6,7,8]])
print(a)
```

```
[[1 2 3 4]
 [5 6 7 8]]
```

In [44]:
```
a.shape   #2 row and 4 column
```

Out[44]:
```
(2, 4)
```

In [45]:
```
a.shape= (4,2)  #change to 4 row and 2 column
```

In [46]:
```
print(a)
```

```
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
```

sum

In [48]:
```
a=np.array([[1,2],[3,4]])
b=np.array([[4,5],[6,7]])
```

In [49]:
```
print(a)
print(b)
```

```
[[1 2]
 [3 4]]
[[4 5]
 [6 7]]
```

In [51]:
```
np.sum([a,b])
```

Out[51]:
```
32
```

In [52]:
```
np.sum([a,b],axis=0) #add vertically
```

Out[52]:
```
array([[ 5,  7],
       [ 9, 11]])
```

In [53]:
```
np.sum([a,b],axis=1) #add horizantally
```

Out[53]:
```
array([[ 4,  6],
       [10, 12]])
```

Joining arrays - vstack, hstack,columnstack

In [54]:
```
a
```

Out[54]:
```
array([[1, 2],
       [3, 4]])
```

In [55]:
```
b
```

Out[55]:
```
array([[4, 5],
       [6, 7]])
```

In [56]:
```
np.vstack([a,b])        #joining vertically array upon array
```

Out[56]:
```
array([[1, 2],
       [3, 4],
       [4, 5],
       [6, 7]])
```

In [57]:
```
np.hstack([a,b])    #joining horizantally, array with array
```

Out[57]:
```
array([[1, 2, 4, 5],
       [3, 4, 6, 7]])
```

In [58]:
```
np.column_stack([a,b])
```

Out[58]:
```
array([[1, 2, 4, 5],
       [3, 4, 6, 7]])
```

PANDAS

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

In [3]:
```
#lets create series first
a=pd.Series([1,2,3,4,5])
```

In [4]:
```
print(a)
```

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

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

Out[5]:
```
pandas.core.series.Series
```

In [10]:
```
#In series we cannot have column, once we assign it will get convert to dataframe
```

DataFrame

In [17]:
```
#Creating dataframe from dictionary
dict={'Country': ['Russia','Colombia','Chile','Equador','Nigeria'],'Rank':[121,40,100,130,11]}
```

In [18]:
```
data=pd.DataFrame(dict)
data
```

Out[18]:

	Country	Rank
0	Russia	121
1	Colombia	40
2	Chile	100
3	Equador	130
4	Nigeria	11

In [19]:
```
data.describe()
```

Out[19]:

	Rank
count	5.000000
mean	80.400000
std	52.300096
min	11.000000
25%	40.000000
50%	100.000000
75%	121.000000
max	130.000000

In [20]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   Country  5 non-null      object
 1   Rank     5 non-null      int64 
dtypes: int64(1), object(1)
memory usage: 212.0+ bytes
```

In [22]:

```
#lets reset index, removing old one
data["position"]=["F","S","T","FU","FI"]
```

In [23]:

```
data
```

Out[23]:

	Country	Rank	position
0	Russia	121	F
1	Colombia	40	S
2	Chile	100	T
3	Equador	130	FU
4	Nigeria	11	FI

In [25]:

```
data.set_index("position")
```

Out[25]:

	Country	Rank
position		
F	Russia	121
S	Colombia	40
T	Chile	100
FU	Equador	130
FI	Nigeria	11

lets import new table so that we can use many other function and method of pandas

In [3]:

```
df=pd.read_csv(r"C:\Users\USER\Downloads\nba.csv")
df.head()
```

Out[3]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0

In [31]:

```
#lets sort_values by Age - Ascending to descending and we need only two output column Name and age
df[["Name", "Age"]].sort_values(by="Age", ascending=True)
```

Out[31]:

	Name	Age
226	Rashad Vaughn	19.0
122	Devin Booker	19.0
40	Kristaps Porzingis	20.0
401	Tyus Jones	20.0
427	Cliff Alexander	20.0
...	...	...
102	Pablo Prigioni	39.0
298	Tim Duncan	40.0
400	Kevin Garnett	40.0
304	Andre Miller	40.0
457	NaN	NaN

458 rows × 2 columns

In [32]:
```
#But we all know its not permanently saved, until we keep inplace=True
```

In [33]:
```
#We can sort two column at a same time too
df.sort_values(by=['Name','Age'],ascending=[True,False],inplace=False)
```

Out[33]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
152	Aaron Brooks	Chicago Bulls	0.0	PG	31.0	6-0	161.0	Oregon	2250000.0
356	Aaron Gordon	Orlando Magic	0.0	PF	20.0	6-9	220.0	Arizona	4171680.0
328	Aaron Harrison	Charlotte Hornets	9.0	SG	21.0	6-6	210.0	Kentucky	525093.0
404	Adreian Payne	Minnesota Timberwolves	33.0	PF	25.0	6-10	237.0	Michigan State	1938840.0
312	Al Horford	Atlanta Hawks	15.0	C	30.0	6-10	245.0	Florida	12000000.0
...	...	...	...	...	...	...	...	...	...
270	Xavier Munford	Memphis Grizzlies	14.0	PG	24.0	6-3	180.0	Rhode Island	NaN
402	Zach LaVine	Minnesota Timberwolves	8.0	PG	21.0	6-5	189.0	UCLA	2148360.0
271	Zach Randolph	Memphis Grizzlies	50.0	PF	34.0	6-9	260.0	Michigan State	9638555.0
237	Zaza Pachulia	Dallas Mavericks	27.0	C	32.0	6-11	275.0	NaN	5200000.0
457	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

458 rows × 9 columns

In [34]:
```
df.duplicated().any()
```

Out[34]:
```
False
```

In [35]:
```
#False means there is not any duplicated values
```

In [36]:
```
df.isnull().any()
```

Out[36]:
```
Name        True
Team        True
Number      True
Position    True
Age         True
Height      True
Weight      True
College     True
Salary      True
dtype: bool
```

In [37]:
```
#means there is null values
```

In [38]:
```
df.isnull().sum()
```

Out[38]:
```
Name         1
Team         1
Number       1
Position     1
Age          1
Height       1
Weight       1
College     85
Salary      12
dtype: int64
```

In [42]:
```
df=df.dropna()  #drop null values and not available values
```

In [43]:

```
df.isnull()
```

Out[43]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
449	False	False	False	False	False	False	False	False	False
451	False	False	False	False	False	False	False	False	False
452	False	False	False	False	False	False	False	False	False
453	False	False	False	False	False	False	False	False	False
456	False	False	False	False	False	False	False	False	False

364 rows × 9 columns

In [44]:

```
df.isnull().any()
```

Out[44]:

```
Name        False
Team        False
Number      False
Position    False
Age         False
Height      False
Weight      False
College     False
Salary      False
dtype: bool
```

About index and column

In [46]:

```
df.head(6)
```

Out[46]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
8	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0

In [51]:

```
#lets change College to University
df.rename(columns={"College":"University"},inplace=True)
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_12876\2713448321.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.rename(columns={"College":"University"},inplace=True)
```

In [52]:

```
df
```

Out[52]:

	Name	Team	Number	Position	Age	Height	Weight	University	Salary	University
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0	Texas
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0	Marquette
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0	Georgia State
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0	LSU
7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0	Gonzaga
...	...	...	...	...	...	...	...	...	...	...
449	Rodney Hood	Utah Jazz	5.0	SG	23.0	6-8	206.0	Duke	1348440.0	Duke
451	Chris Johnson	Utah Jazz	23.0	SF	26.0	6-6	206.0	Dayton	981348.0	Dayton
452	Trey Lyles	Utah Jazz	41.0	PF	20.0	6-10	234.0	Kentucky	2239800.0	Kentucky
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0	Butler
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0	Kansas

364 rows × 10 columns

In [53]:

```
#lets drop university
df.drop("University",axis=1)
```

Out[53]:

	Name	Team	Number	Position	Age	Height	Weight	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	6796117.0
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	1148640.0
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	1170960.0
7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	2165160.0
...	...	...	...	...	...	...	...	...
449	Rodney Hood	Utah Jazz	5.0	SG	23.0	6-8	206.0	1348440.0
451	Chris Johnson	Utah Jazz	23.0	SF	26.0	6-6	206.0	981348.0
452	Trey Lyles	Utah Jazz	41.0	PF	20.0	6-10	234.0	2239800.0
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	2433333.0
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	947276.0

364 rows × 8 columns

Matplotlib

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Line

In [11]:

```
x=np.arange(1,10)
print(x)
```

```
[1 2 3 4 5 6 7 8 9]
```

In [12]:

```
y=2*x
print(y)
```

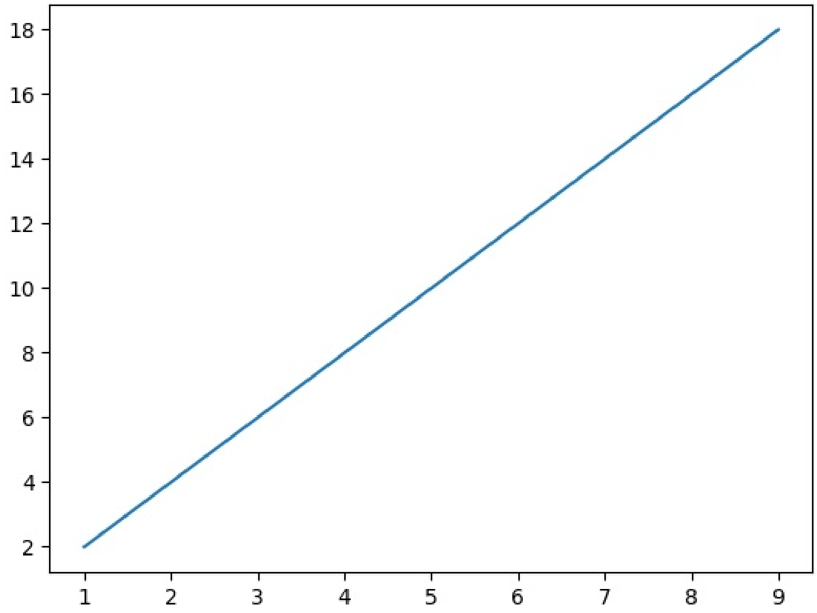
```
[ 2  4  6  8 10 12 14 16 18]
```

In [13]:

```
plt.plot(x,y)
```

Out[13]:

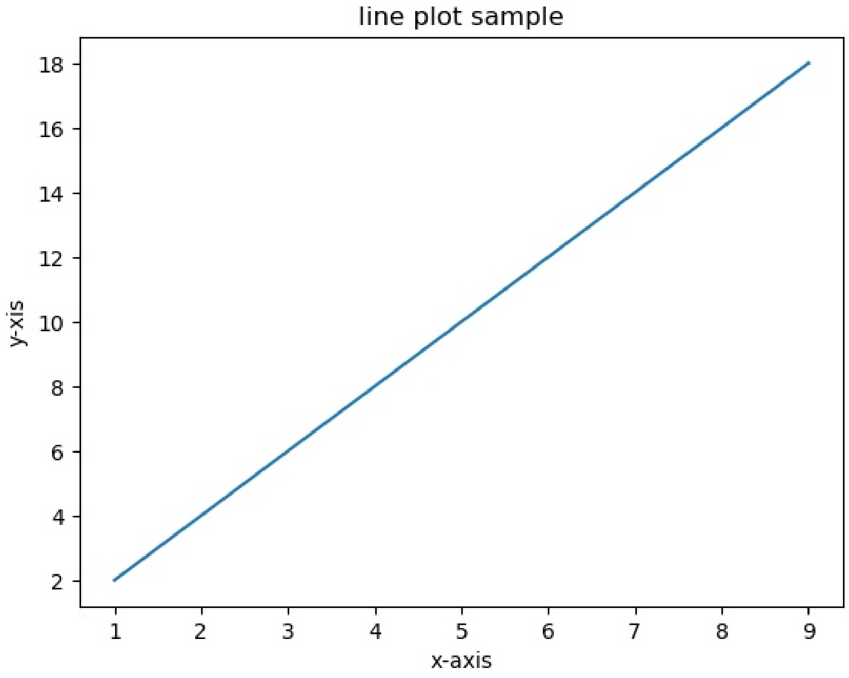
```
[<matplotlib.lines.Line2D at 0x20bcfc3b610>]
```



In [14]:
```
#It look very easy and it is easy, but just note we are just seeing sample and learning matplotlib
```

In [15]:
```
#lets put name and label
plt.plot(x,y)
plt.title("line plot sample")
plt.xlabel("x-axis")
plt.ylabel("y-xis")
```

Out[15]:
```
Text(0, 0.5, 'y-xis')
```



In [18]:
```
#So, in line plot we can draw 2 line like lets made one more array to see the example

y2=3*x
print(y2)
```

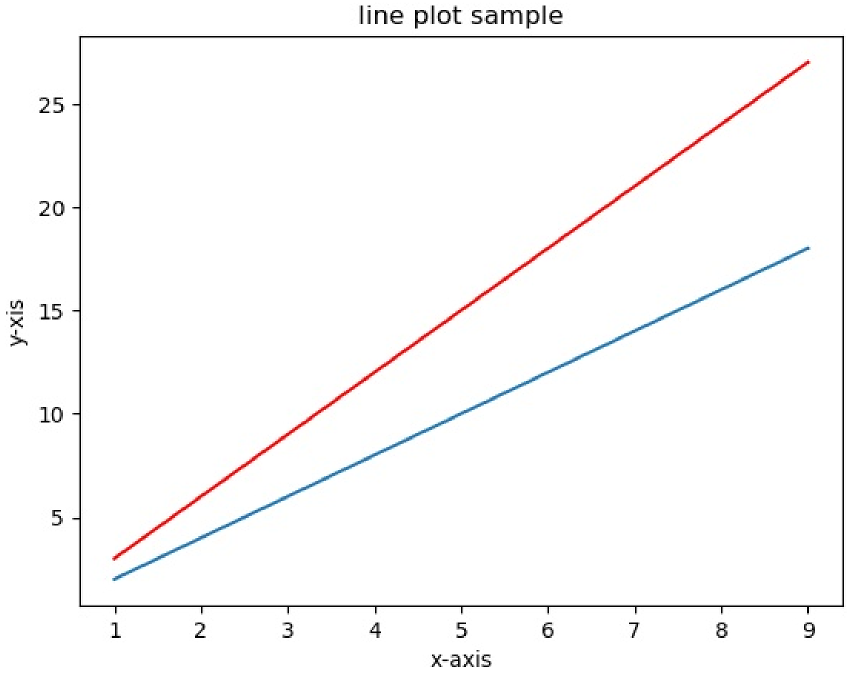
```
[ 3  6  9 12 15 18 21 24 27]
```

In [19]:

```
plt.plot(x,y)
plt.plot(x,y2,color="red")
plt.title("line plot sample")
plt.xlabel("x-axis")
plt.ylabel("y-xis")
```

Out[19]:

```
Text(0, 0.5, 'y-xis')
```



In real data sets, we can compare two stock prices or two product sales with price in x with this method.

Bar-plot

In [20]:

```
data = {
    "Bananas": 300,
    "Oranges": 250,
    "Apples": 200,
    "Strawberries": 175,
    "Pineapples": 150
}
```

In [21]:

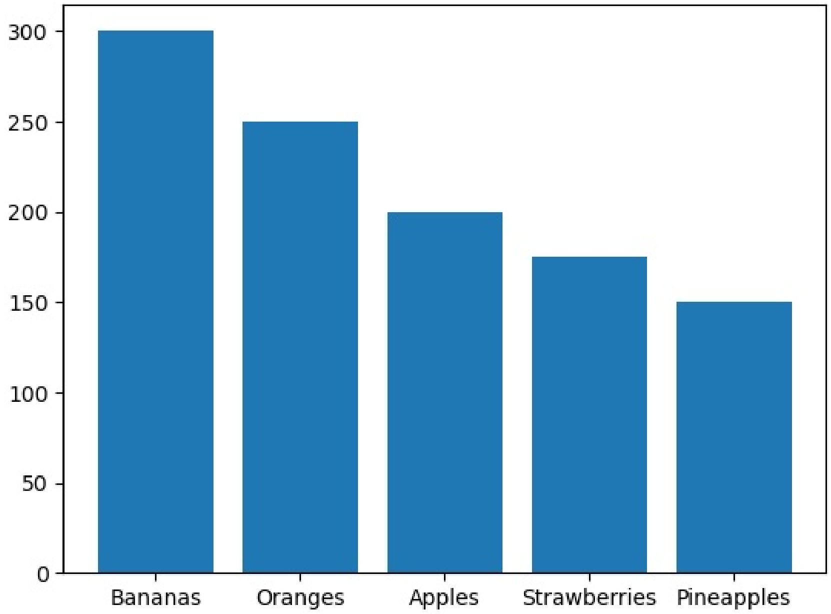
```
fruits=list(data.keys())      #because bar plot only take list
price=list(data.values())
```

In [22]:

```
plt.bar(fruits,price)      #first is x and it should be categorical value and y should be numerical
```

Out[22]:

```
<BarContainer object of 5 artists>
```



In [46]:

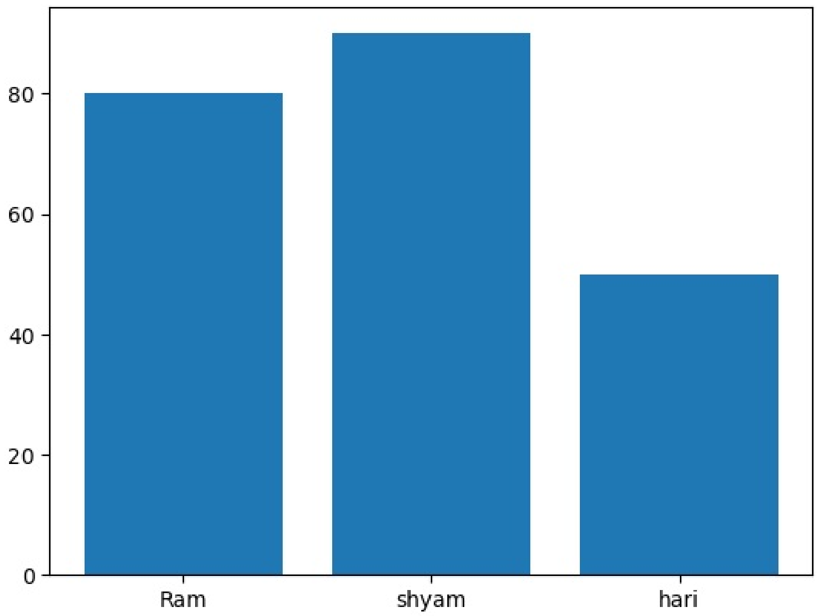
```
#so we can directly use list to create bar?

name=["Ram","shyam","hari"]
marks=[80,90,50]
```

In [24]:

```
plt.bar(name,marks)
```

Out[24]:

```
<BarContainer object of 3 artists>
```

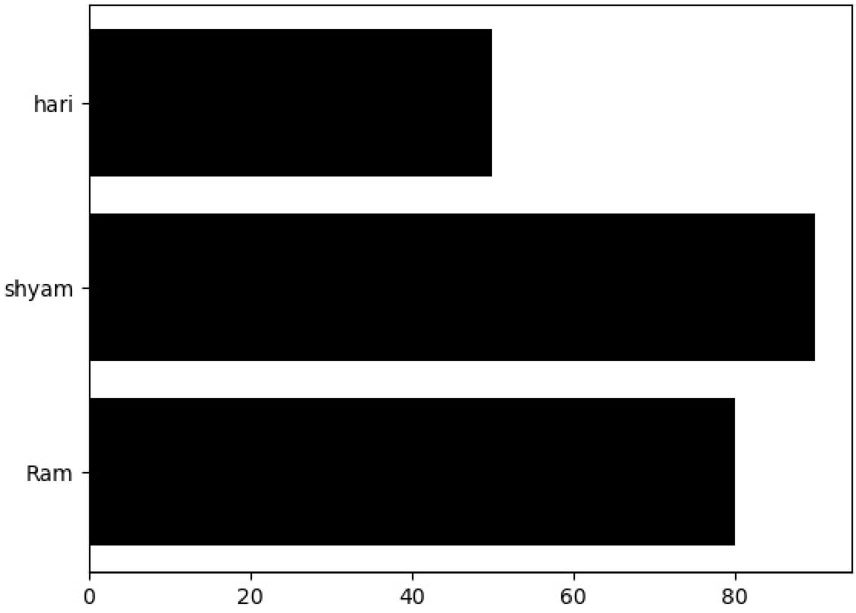


In [26]:

```
#Now suppose we want this graph horizantally
plt.barh(name,marks, color="black")
```

Out[26]:

```
<BarContainer object of 3 artists>
```



Scatter plot

In [27]:

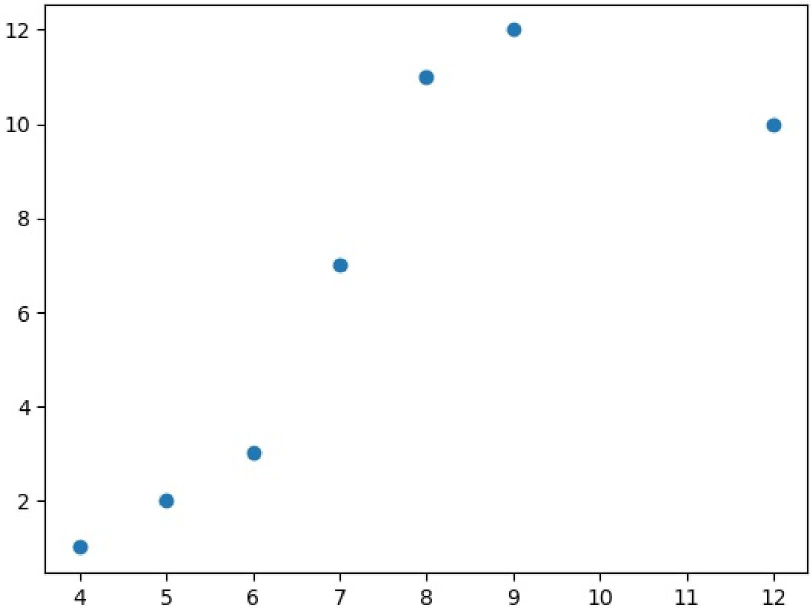
```
x=[4,5,6,7,8,9,12]
y=[1,2,3,7,11,12,10]
```

In [28]:

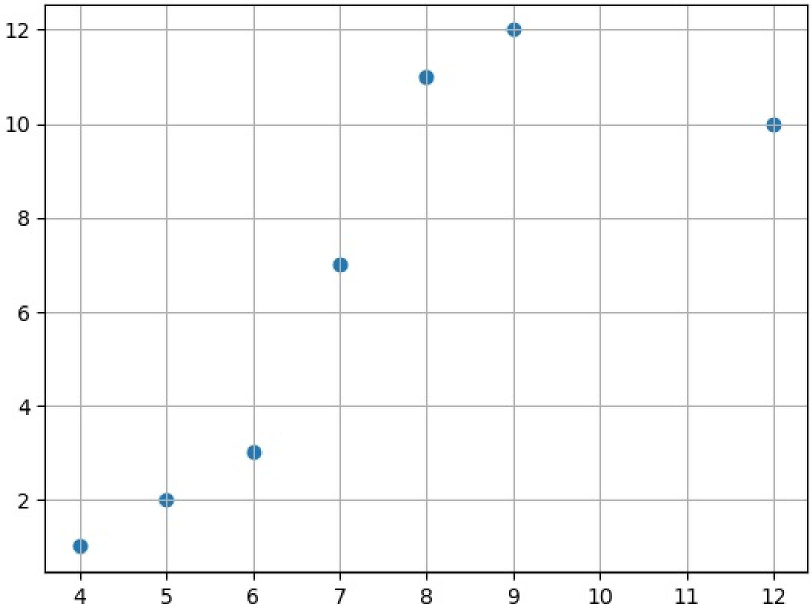
```
plt.scatter(x,y)
```

Out[28]:

```
<matplotlib.collections.PathCollection at 0x20bd4f5e610>
```



```
In [29]: #if needed grid
         plt.scatter(x,y)
         plt.grid(True)
```



Histogram

```
In [30]: #lets load pokemon data
         dataset=pd.read_csv(r"C:\Users\USER\Downloads\pokemon_data.csv")
         dataset.head()
```

Out[30]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	1	False

In [31]:

```
#lets see which data is numerical and which is categorical
dataset.info()
```

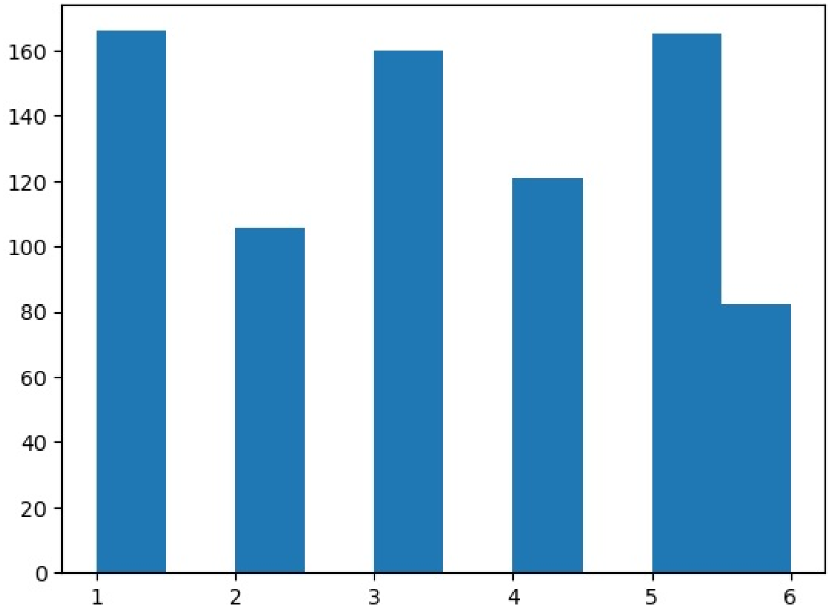
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   #           800 non-null    int64 
 1   Name        800 non-null    object
 2   Type 1      800 non-null    object
 3   Type 2      414 non-null    object
 4   HP          800 non-null    int64 
 5   Attack      800 non-null    int64 
 6   Defense     800 non-null    int64 
 7   Sp. Atk     800 non-null    int64 
 8   Sp. Def     800 non-null    int64 
 9   Speed       800 non-null    int64 
 10  Generation  800 non-null    int64 
 11  Legendary   800 non-null    bool  
dtypes: bool(1), int64(8), object(3)
memory usage: 69.7+ KB
```

In [35]:

```
#so lets see histogram on Generation
plt.hist(dataset["Generation"])
```

Out[35]:

```
(array([166.,   0., 106.,   0., 160.,   0., 121.,   0., 165.,  82.]),
 array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. , 5.5, 6. ]),
 <BarContainer object of 10 artists>)
```



What is difference between bar plot and histogram?

- Bar plot is use to understand the distribution of categorical data whereas histogram is for continous data.

Box plot

In [38]:

```
df
```

Out[38]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
...	...	...	...	...	...	...	...	...	...
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0
454	Raul Neto	Utah Jazz	25.0	PG	24.0	6-1	179.0	NaN	900000.0
455	Tibor Pleiss	Utah Jazz	21.0	C	26.0	7-3	256.0	NaN	2900000.0
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0
457	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

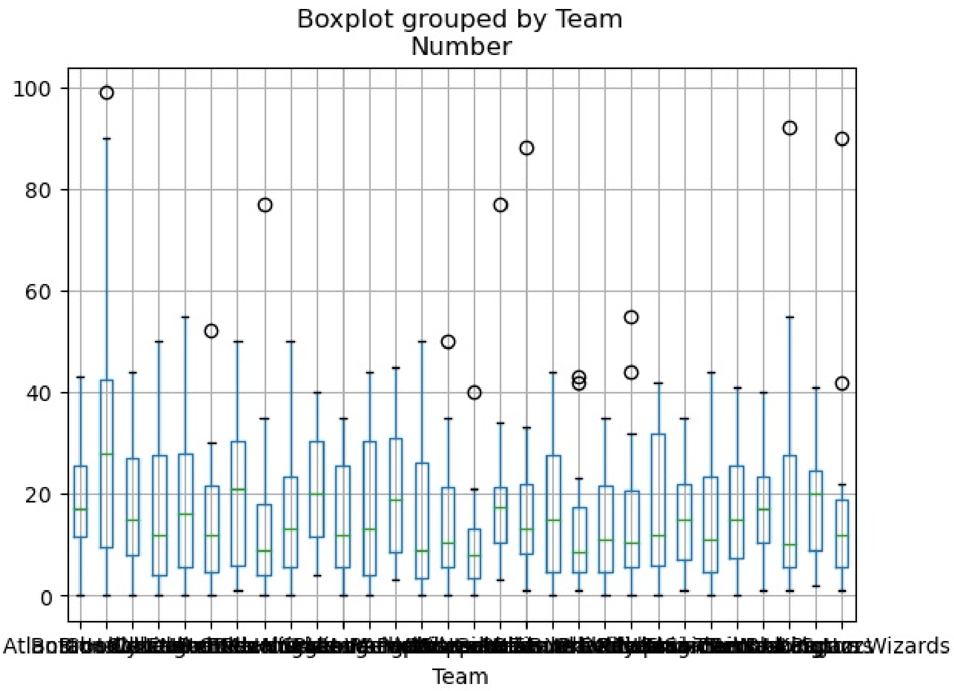
458 rows × 9 columns

In [39]:

```
df.boxplot(column="Number",by="Team")
```

Out[39]:

```
<Axes: title={'center': 'Number'}, xlabel='Team'>
```



In [40]:

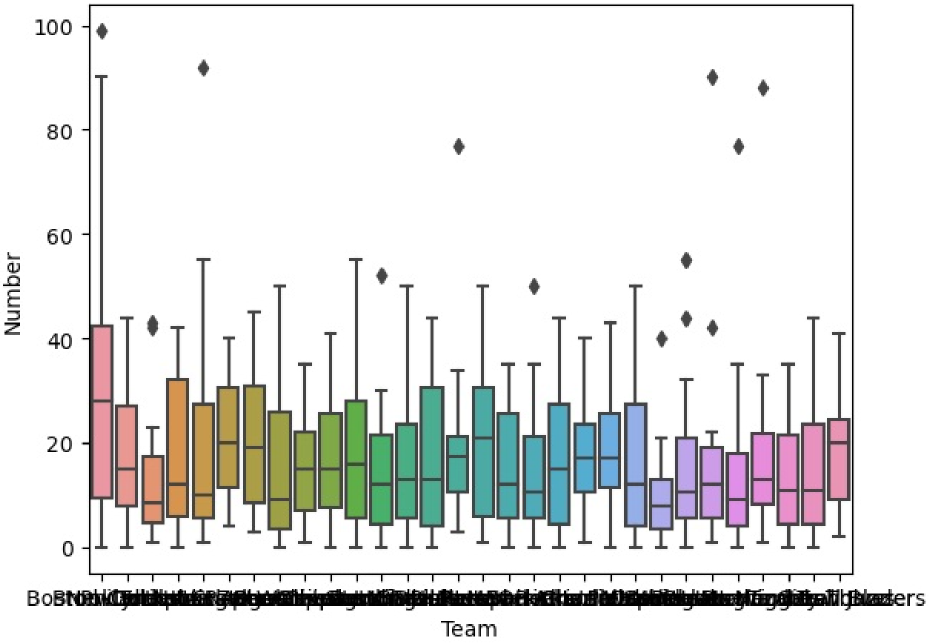
```
#instead of matplotlib box plot, lets try seaborn boxplotlib
import seaborn as sns
```

In [45]:

```
sns.boxplot(y=df["Number"],x=df["Team"])
```

Out[45]:

```
<Axes: xlabel='Team', ylabel='Number'>
```

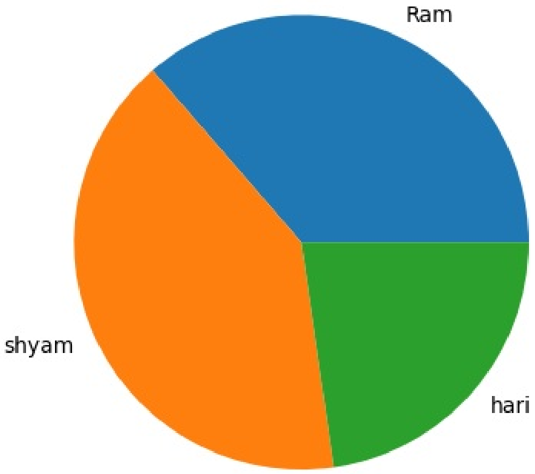


In [47]:
```
#so seaborn box plot is better
```

Pie chart

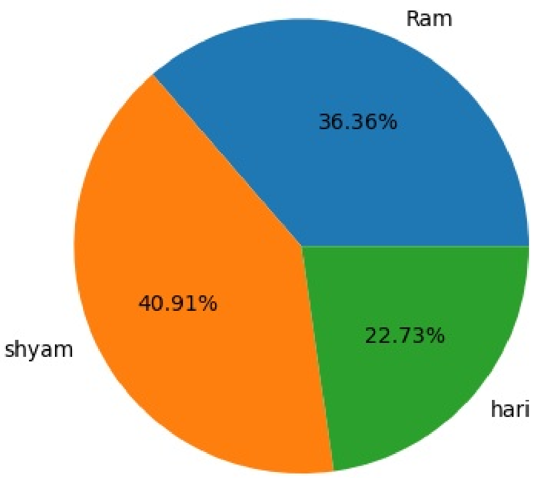
In [48]:
```
name=["Ram","shyam","hari"]
marks=[80,90,50]
```

In [51]:
```
plt.pie(marks,labels=name)   #first attribute is numerical
plt.show()
```



In [53]:

```
#if want percentage
plt.pie(marks,labels=name,autopct='%0.2f%%')   #first attribute is numerical
plt.show()
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js