COURSE OUTCOME-1

PROGRAM NO-1

Aim: Review of python programming – Programs review the fundamentals of python

Datatypes

```
#numbers
```

```
3+3 #addition
```

```
6
```

```
4-3 #subtraction
```

```
1
```

```
10*5  #multiplication
```

```
50
```

```
10/5   #divison
```

```
2.0
```

```
5**2  #power
```

```
25
```

```
8%2    #modulo function
```

```
0
```

Strings

```
'hello' #single quotes
```

```
{"type":"string"}
```

```
"hello world"  #double quotes
```

```
{"type":"string"}
```

print

```
#variable assignmnet
x=22
y=20
z=x+y
print (z)
```

```
42
```

```
a= 'tanu'
b='manu'
print('my name is :{}, and my friend is :{}'.format(a,b))
```

```
my name is :tanu, and my friend is :manu
```

List

```
my_list=[1,2,3,4]
my_list.append(6)
my_list
```

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

```
my_list[3]
```

```
4
```

```
my_list[0:2]
```

```
[1, 2]
```

```
my_list[2:]
```

```
[3, 4, 6]
```

```
my_list[:2]
```

```
[1, 2]
```

```
my_list[1]= 34
my_list
```

```
[1, '34', 3, 4, 6]
```

Dictionary

```
d = {'key1':'item1','key2':'item2'}
d
```

```
{'key1': 'item1', 'key2': 'item2'}
```

```
d['key2']
```

```
{"type":"string"}
```

Comparison Operators

```
2>5
```

```
False
```

```
5>2
```

```
True
```

```
3 == 5
```

```
False
```

Tuples

```
t=(1,2,3)
t
```

```
(1, 2, 3)
```

```
t[1]
```

```
2
```

Sets

```
s={1,2,3,2,4,5,6,1,2,7}
s
```

```
{1, 2, 3, 4, 5, 6, 7}
```

Logic Operators

```
(1>2) or (2<3)
```

```
True
```

```
(3>4) and (4>5)
```

```
False
```

if else statements

```
if 2> 3:
  print("correct")
else:
   print('wrong')
```

```
wrong
```

```
if 1 == 2:
    print('first')
elif 2 == 2:
    print('second')
else:
    print('Last')
```

```
second
```

Loops

```
a=[1,2,3,4,5,6] #for loop
for i in a:
  print(i)
```

```
1
2
3
4
5
6
```

```
i=1           #while loop
while i<7:
  print('i is:{}'.format(i))
  i=i+1
```

```
i is:1
i is:2
i is:3
i is:4
i is:5
i is:6
```

Range

```
list(range(10))
```

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

```
for i in range(10):
  print(i)
```

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

Lambda

```
def a(var):
  return var**2
```

```
a(5)
```

```
25
```

functions

```
def my_func(param1='default'):

    print(param1)
```

```
my_func
```

```
<function __main__.my_func>
```

```
my_func()
```

```
default
```

```
def cube(x):
  print(x**3)
```

```
a=cube(8)
```

```
512
```

Result: The program is executed successfully and obtained the output.

PROGRAM NO-2

Aim: Matrix operations (using vectorization) and transformation using python and SVD using Python.

```
import numpy as np
```

```
a = np.zeros(10)
```

```
a
```

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

```
np.ones(10)
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
np.ones(10)
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
a = np.ones(10)
```

```
a*5
```

```
array([5., 5., 5., 5., 5., 5., 5., 5., 5., 5.])
```

```
a =np.arange(10,51)
```

```
a
```

```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
26,
       27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
43,
       44, 45, 46, 47, 48, 49, 50])
```

```
b = np.arange(10,51,2)
```

```
b
```

```
array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40,
42,
       44, 46, 48, 50])
```

```
a = np.arange(0,9)
```

```
a.reshape(3,3)
```

```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

```
a = np.array(a)
```

```
a
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

```
np.eye(3)
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
np.random.rand(1)
```

```
array([0.58625916])
```

Use numpy to generate an array of 25 random numbers sampled from a standard normal distribution

```
a = np.random.randn(25)
```

```
a
```

```
array([-0.04896616, -0.8041291 ,  1.13550601, -0.53619693,
2.69545043,
       -0.48403258, -0.69521474, -1.50046868, -1.46007021,
1.12750033,
        0.12596504,  0.40128316,  2.26360832,  0.10771633,
0.15204888,
       -0.03720986,  0.4573639 , -1.33667719,  0.02730499, -
1.07894103,
       -0.01669391, -0.5611958 ,  0.43010856, -1.42591487,
1.77139352])
```

```
a = np.random.randn(5,5)
```

```
a
```

```
array([[ 0.29482346, -0.80666764,  1.79146308,  1.39839   , -
0.86338596],
       [ 1.36779522, -0.29962507,  0.40596963, -0.29847113, -
2.63526737],
       [-1.6010666 , -0.63444688, -1.22025455, -0.69600626, -
1.63989681],
       [ 0.54234526,  1.70864562, -1.46383217,  0.47120988,  0.4767304
],
       [-0.32111072,  1.07868078, -3.085043  , -0.82972413,  1.0549771
]])
```

create the following matrix the number should be ranging between 0 and 1

```
np.linspace(0,100,1)
```

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

```
np.linspace(0,1,100)
```

```
array([0.        , 0.01010101, 0.02020202, 0.03030303, 0.04040404,
       0.05050505, 0.06060606, 0.07070707, 0.08080808, 0.09090909,
       0.1010101 , 0.11111111, 0.12121212, 0.13131313, 0.14141414,
       0.15151515, 0.16161616, 0.17171717, 0.18181818, 0.19191919,
       0.2020202 , 0.21212121, 0.22222222, 0.23232323, 0.24242424,
       0.25252525, 0.26262626, 0.27272727, 0.28282828, 0.29292929,
       0.3030303 , 0.31313131, 0.32323232, 0.33333333, 0.34343434,
       0.35353535, 0.36363636, 0.37373737, 0.38383838, 0.39393939,
       0.4040404 , 0.41414141, 0.42424242, 0.43434343, 0.44444444,
       0.45454545, 0.46464646, 0.47474747, 0.48484848, 0.49494949,
       0.50505051, 0.51515152, 0.52525253, 0.53535354, 0.54545455,
       0.55555556, 0.56565657, 0.57575758, 0.58585859, 0.5959596 ,
       0.60606061, 0.61616162, 0.62626263, 0.63636364, 0.64646465,
       0.65656566, 0.66666667, 0.67676768, 0.68686869, 0.6969697 ,
       0.70707071, 0.71717172, 0.72727273, 0.73737374, 0.74747475,
       0.75757576, 0.76767677, 0.77777778, 0.78787879, 0.7979798 ,
       0.80808081, 0.81818182, 0.82828283, 0.83838384, 0.84848485,
       0.85858586, 0.86868687, 0.87878788, 0.88888889, 0.8989899 ,
       0.90909091, 0.91919192, 0.92929293, 0.93939394, 0.94949495,
       0.95959596, 0.96969697, 0.97979798, 0.98989899, 1.        ])
```

```
a = np.arange(1,100,1)
```

```
a
```

```
array([ 1,  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, 30, 31, 32, 33,
34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
68,
       69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
85,
       86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

```
a/100
```

```
array([0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1 ,
0.11,
       0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2 , 0.21,
0.22,
       0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3 , 0.31, 0.32,
0.33,
       0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4 , 0.41, 0.42, 0.43,
0.44,
       0.45, 0.46, 0.47, 0.48, 0.49, 0.5 , 0.51, 0.52, 0.53, 0.54,
```

```
0.55,
       0.56, 0.57, 0.58, 0.59, 0.6 , 0.61, 0.62, 0.63, 0.64, 0.65,
0.66,
       0.67, 0.68, 0.69, 0.7 , 0.71, 0.72, 0.73, 0.74, 0.75, 0.76,
0.77,
       0.78, 0.79, 0.8 , 0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87,
0.88,
       0.89, 0.9 , 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98,
0.99])
```

create an array of 20 lineraly spaced points between 0 and 1

```
np.linspace(0,1,20)
```

```
array([0.        , 0.05263158, 0.10526316, 0.15789474, 0.21052632,
       0.26315789, 0.31578947, 0.36842105, 0.42105263, 0.47368421,
       0.52631579, 0.57894737, 0.63157895, 0.68421053, 0.73684211,
       0.78947368, 0.84210526, 0.89473684, 0.94736842, 1.        ])
```

```
a = np.arange(1,26).reshape(5,5)
```

```
a
```

```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25]])
```

```
a[2:,1:]
```

```
array([[12, 13, 14, 15],
       [17, 18, 19, 20],
       [22, 23, 24, 25]])
```

```
a[3:4,4:]
```

```
array([[20]])
```

```
a[3,4]
```

```
20
```

```
a[0:3,1:2]
```

```
array([[ 2],
       [ 7],
       [12]])
```

```
a[4:,0:]
```

```
array([[21, 22, 23, 24, 25]])
```

```
a[3:,0:]
```

```
array([[16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25]])
```

get the sum of all values in the matrix

create the following matrix the number should be ranging between 0 and 1

```
np.linspace(0,100,1)
```

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

```
a.sum()
```

```
325
```

```
a.std()
```

```
7.211102550927978
```

```
import numpy as np
```

```
A = np.arange(0,25)
```

```
A
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

SVD

```
from scipy.linalg import svd
```

```
a = np.arange(1,19).reshape(6,3)
```

```
U, s, VT = svd(a)
```

```
U
```

```
array([[-0.07736219,  0.71960032, -0.09075777, -0.25083666, -
0.45979172,
        -0.4400296 ],
       [-0.19033085,  0.50893247,  0.58409372,  0.4022013 ,
0.06897105,
         0.44392965],
       [-0.3032995 ,  0.29826463, -0.34118019, -0.54123699,
0.50890998,
         0.38822268],
       [-0.41626816,  0.08759679, -0.61888418,  0.65636038,
0.03282454,
        -0.06437079],
       [-0.52923682, -0.12307105,  0.37872289, -0.04363171,
0.43069535,
        -0.61149707],
```

```
       [-0.64220548, -0.33373889,  0.08800553, -0.22285632, -
0.58160921,
         0.28374512]])

s

array([4.58945322e+01, 1.64070530e+00, 1.74146424e-15])

VT

array([[-0.52903535, -0.57607152, -0.62310769],
       [-0.74394551, -0.03840487,  0.66713577],
       [ 0.40824829, -0.81649658,  0.40824829]])

U, s, VT = svd(a,full_matrices=True)

U

array([[-0.07736219,  0.71960032, -0.09075777, -0.25083666, -
0.45979172,
        -0.4400296 ],
       [-0.19033085,  0.50893247,  0.58409372,  0.4022013 ,
0.06897105,
         0.44392965],
       [-0.3032995 ,  0.29826463, -0.34118019, -0.54123699,
0.50890998,
         0.38822268],
       [-0.41626816,  0.08759679, -0.61888418,  0.65636038,
0.03282454,
        -0.06437079],
       [-0.52923682, -0.12307105,  0.37872289, -0.04363171,
0.43069535,
        -0.61149707],
       [-0.64220548, -0.33373889,  0.08800553, -0.22285632, -
0.58160921,
         0.28374512]])

U, s, VT = svd(a,full_matrices=False)

U

array([[-0.07736219,  0.71960032, -0.09075777],
       [-0.19033085,  0.50893247,  0.58409372],
       [-0.3032995 ,  0.29826463, -0.34118019],
       [-0.41626816,  0.08759679, -0.61888418],
       [-0.52923682, -0.12307105,  0.37872289],
       [-0.64220548, -0.33373889,  0.08800553]])

from numpy import diag

from numpy import dot

a= (U @ np.diag(s) @ VT)
```

```
a
```

```
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.],
       [ 7.,  8.,  9.],
       [10., 11., 12.],
       [13., 14., 15.],
       [16., 17., 18.]])
```

Result: The program is executed successfully and obtained the output.

PROGRAM NO-3

Aim: Programs using matplotlib / plotly / bokeh / seaborn for data visualisation.

```
import matplotlib.pyplot as plt
```

```
import numpy as np
x=np.linspace(0,5,11)
y = x**2
```

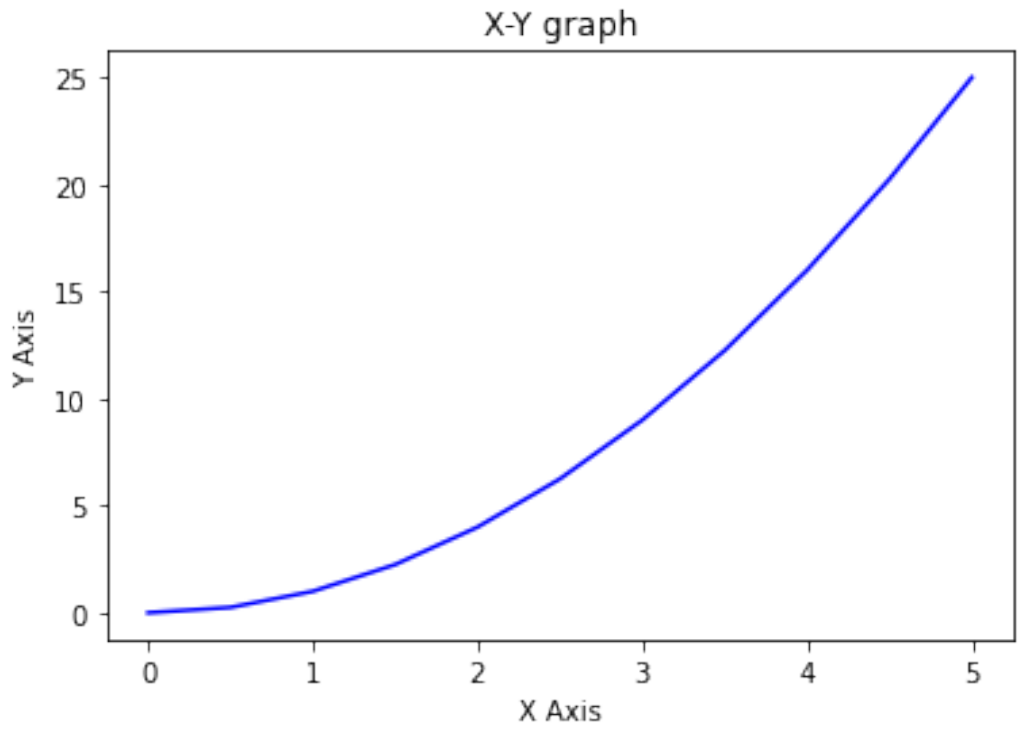
```
x
```

```
array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])
```

```
y
```

```
array([ 0.  ,  0.25,  1.  ,  2.25,  4.  ,  6.25,  9.  , 12.25, 16.  ,
       20.25, 25.  ])
```

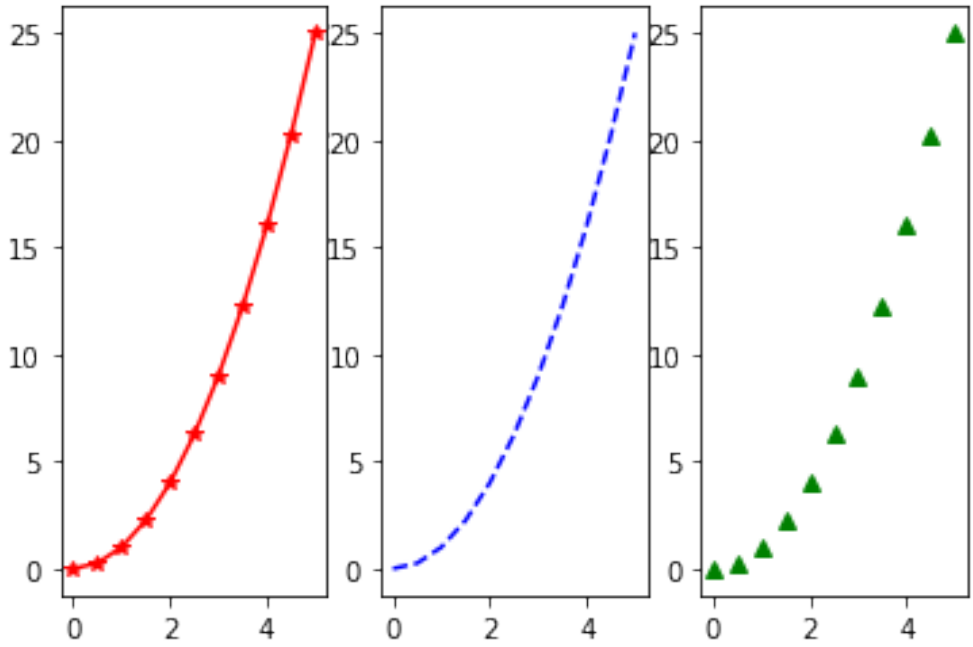
```
plt.plot(x,y, 'b') # ''r' is the color blue
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('X-Y graph')
plt.show()
```




```
#plt.subplot(nrows,ncols,plot_number)
plt.subplot(1,3,2)
plt.plot(x,y,'b--')
plt.subplot(1,3,1)
```

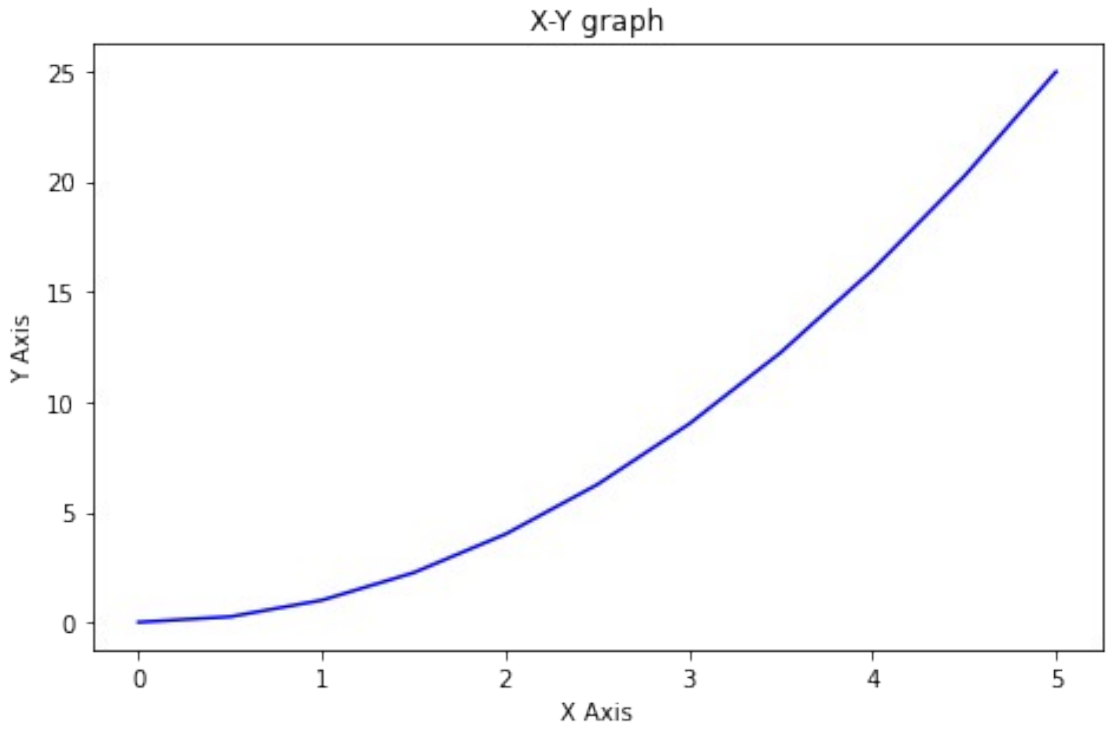
```
plt.plot(x,y,'r*-')
plt.subplot(1,3,3)
plt.plot(x,y,'g^')
```

```
[<matplotlib.lines.Line2D at 0x7f680f517790>]
```




```
#using object oriented
#create figure(empty canvas)
fig = plt.figure()
#add set of axes to figure
axes = fig.add_axes([0.5,0.9,0.99,0.89]) #left,bottom,width,heigth
axes.plot(x,y,'b')
axes.set_xlabel('X Axis')
axes.set_ylabel('Y Axis')
axes.set_title('X-Y graph')
```

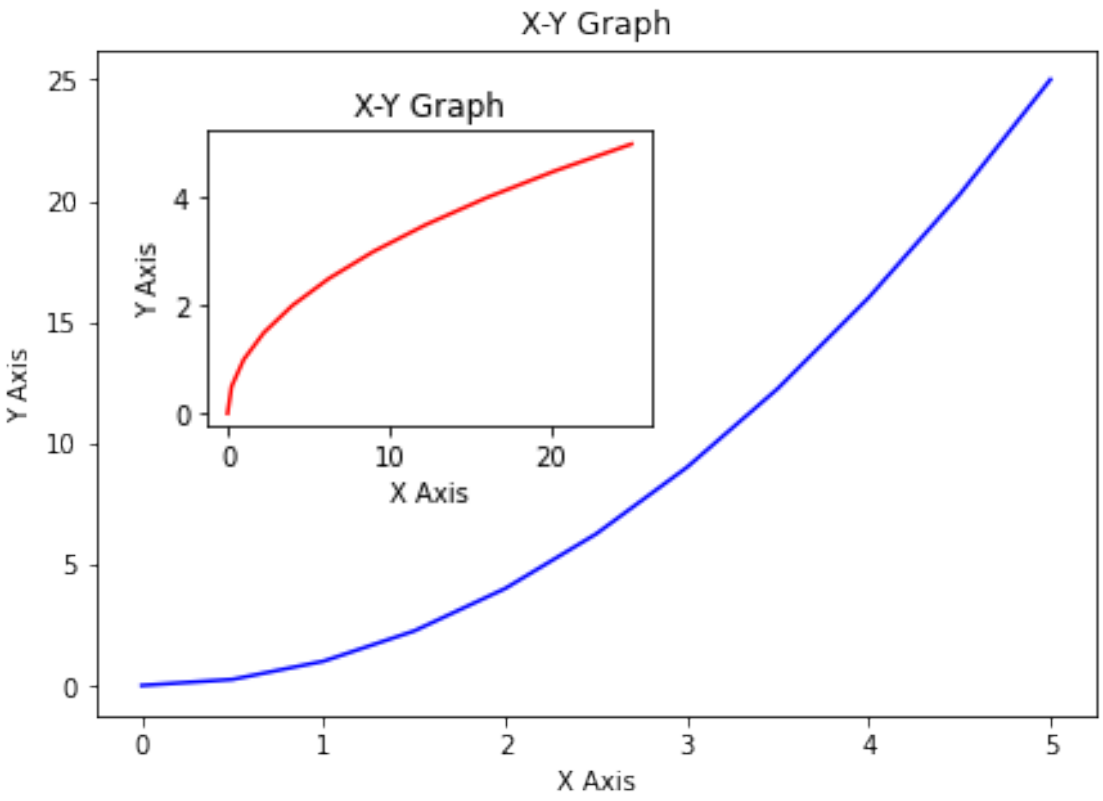
```
Text(0.5, 1.0, 'X-Y graph')
```



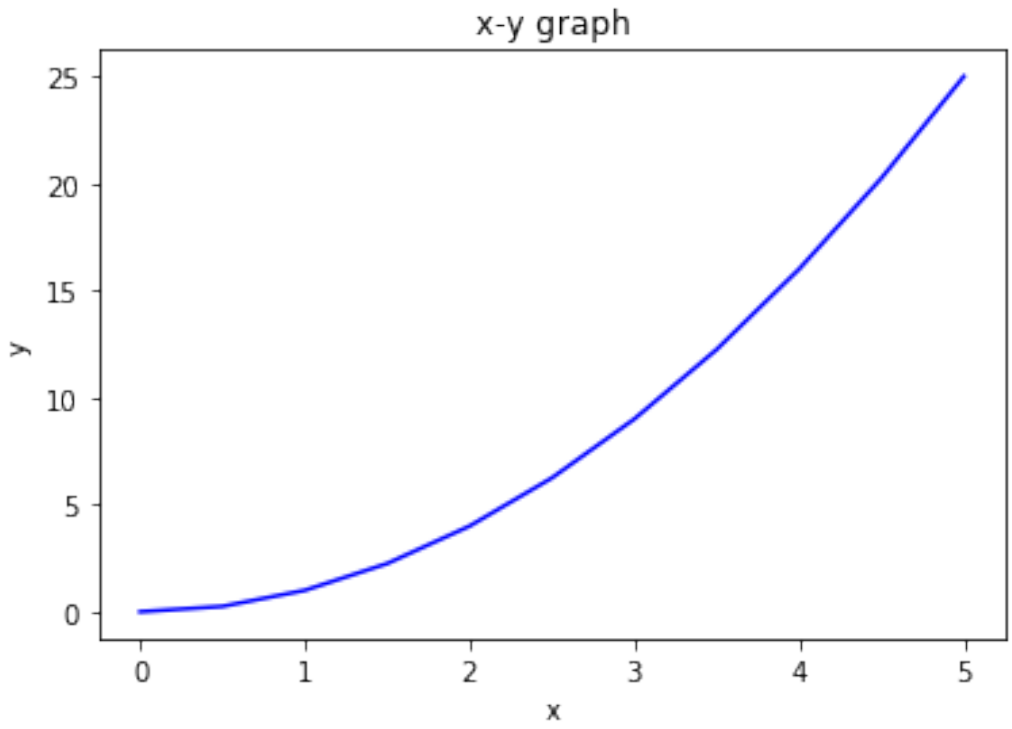

```
fig = plt.figure()
axes1 = fig.add_axes([0.7,0.6,0.9,0.9])
axes2 = fig.add_axes([0.8,0.99,0.4,0.4])
#Main Figure Axes1
axes1.plot(x,y,'b')
axes1.set_xlabel('X Axis')
axes1.set_ylabel('Y Axis')
axes1.set_title('X-Y Graph')

#Insert Figure Axes2
axes2.plot(y,x,'r')
axes2.set_xlabel('X Axis')
axes2.set_ylabel('Y Axis')
axes2.set_title('X-Y Graph')

Text(0.5, 1.0, 'X-Y Graph')
```

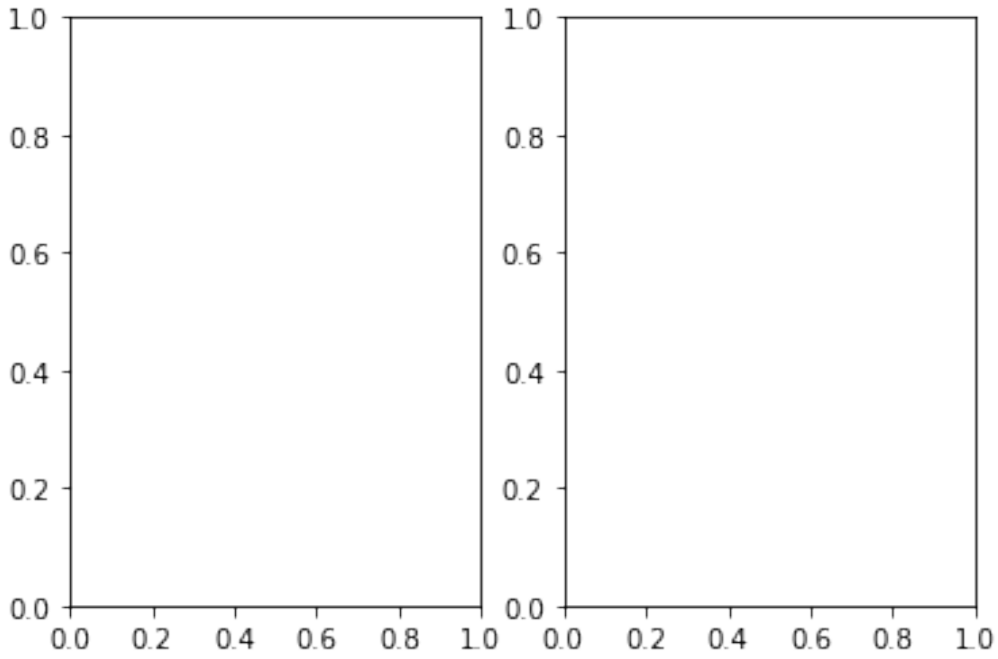



```
fig, axes = plt.subplots()
axes.plot(x,y,'b')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('x-y graph');
```




```
fig,axes = plt.subplots(nrows=1,ncols=2)
```

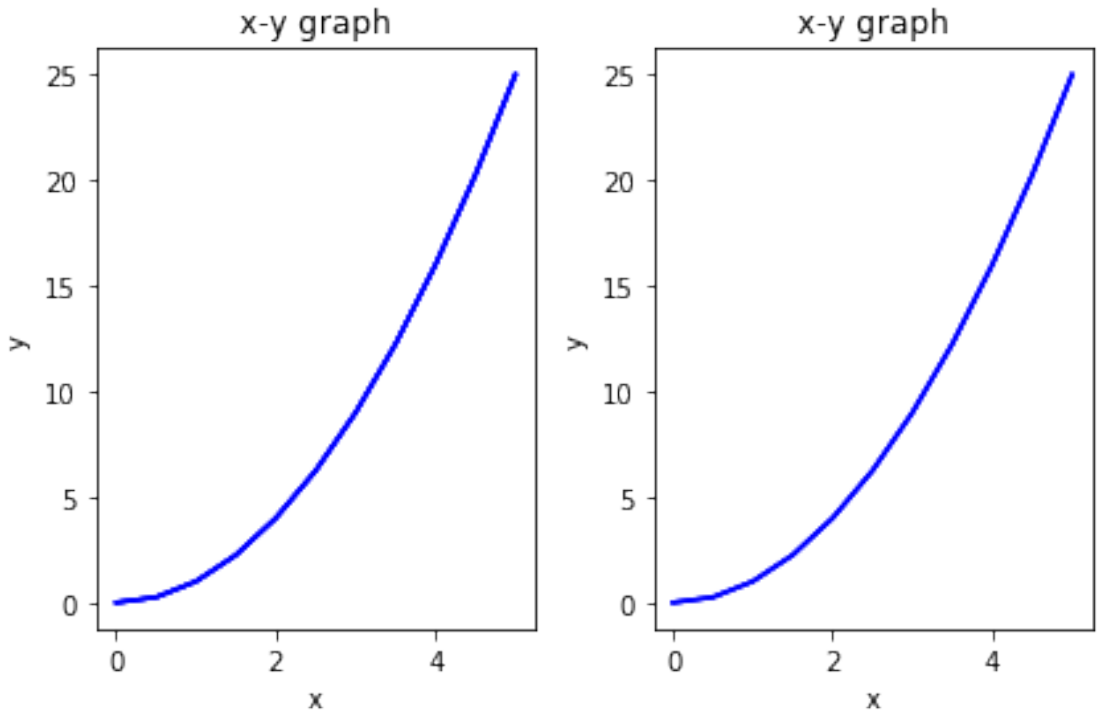
```
Text(0.5, 1.0, 'X-Y Graph')
```




```
for ax in axes:
  ax.plot(x,y,'b')
  ax.set_xlabel('x')
```

```
  ax.set_ylabel('y')
  ax.set_title('x-y graph')
  fig.tight_layout()

fig
```

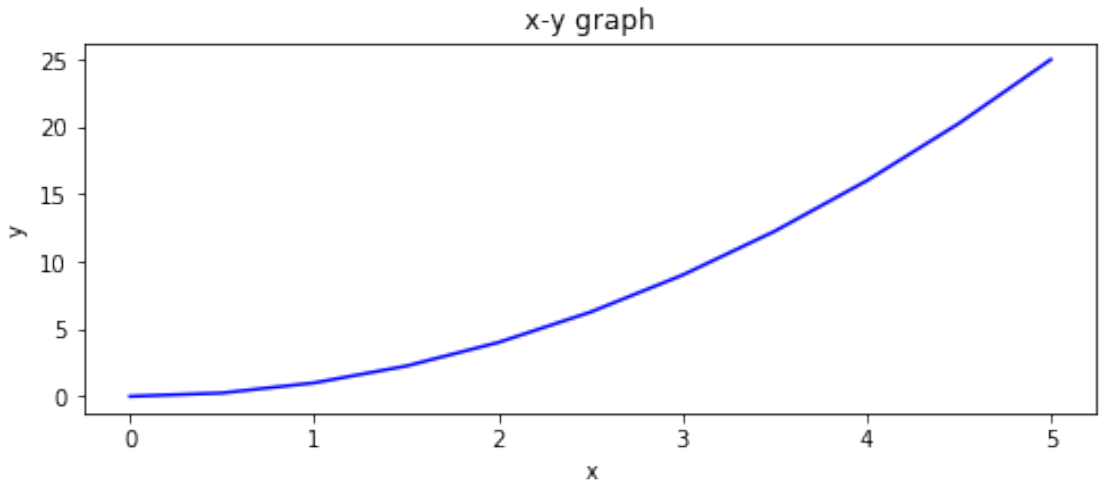



```
fig = plt.figure(figsize = (8,4),dpi = 100)
```

```
<Figure size 800x400 with 0 Axes>
```

```
fig,axes = plt.subplots(figsize=(8,3))
axes.plot(x,y,'b')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('x-y graph')
```

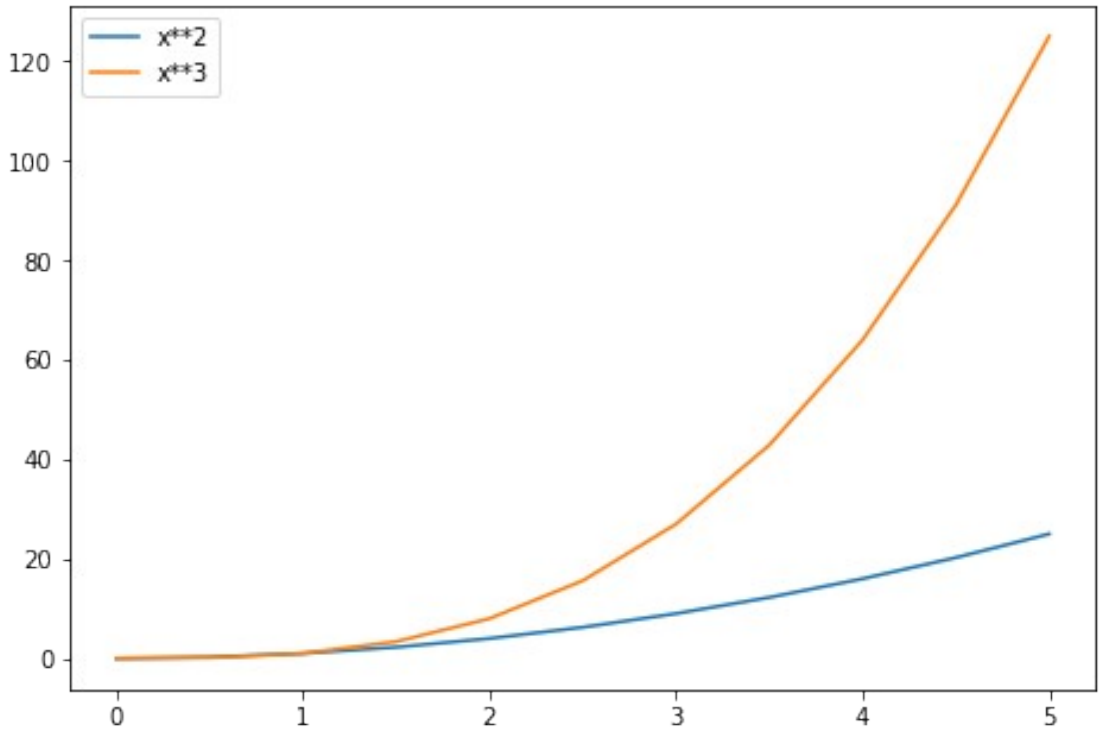
```
Text(0.5, 1.0, 'x-y graph')
```




```
fig.savefig('filename.png',dpi =200)
```

```
#legend
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.plot(x,x**2,label='x**2')
ax.plot(x,x**3,label = 'x**3')
ax.legend(loc = 2)
```

```
<matplotlib.legend.Legend at 0x7f6801eb5b10>
```



```
ax.legend(loc = 1) #upper right
ax.legend(loc =2)#upper left
ax.legend(loc = 3) #lower left
ax.legend(loc = 4) #lower right
ax.legend(loc = 0 )#let matplotlib decide the optimal location
```

```
<matplotlib.legend.Legend at 0x7f6801f50e90>
```

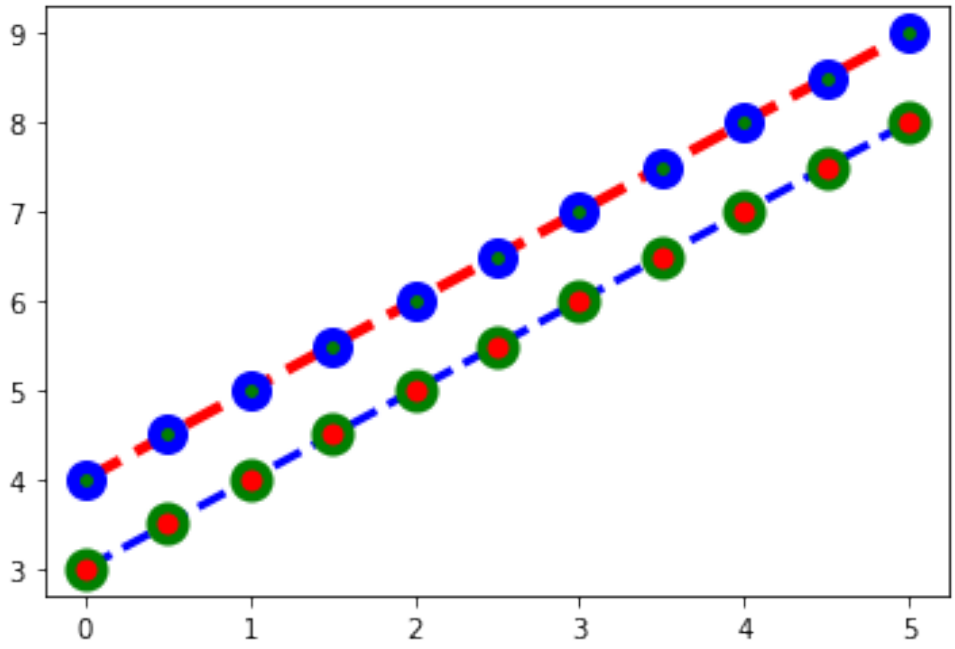
Line and Marker styles

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

```
plt.plot(x,x+3,color='b',lw=3,ls='-',marker='o',markersize=12,linestyl
e='dashed',markeredgecolor='green',markeredgewidth=4,markerfacecolor='
red')
plt.plot(x,x+4,color='r',lw=4,ls='-',marker='o',markersize=10,linestyl
e='dashed',markeredgecolor='blue',markeredgewidth=5,markerfacecolor='g
reen')
```

```
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1:
MatplotlibDeprecationWarning: Saw kwargs ['ls', 'linestyle'] which are
all aliases for 'linestyle'.  Kept value from 'linestyle'.  Passing
multiple aliases for the same property will raise a TypeError in 3.3.
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2:
MatplotlibDeprecationWarning: Saw kwargs ['ls', 'linestyle'] which are
all aliases for 'linestyle'.  Kept value from 'linestyle'.  Passing
multiple aliases for the same property will raise a TypeError in 3.3.
```


9
8
7
6
5
4
3
0
1
2
3
4
5

Distribution Plots

Type of distribution plots are:

1.displot

2.jointplot

3.pairplot

4.rugplot

5.kdeplot

```
import seaborn as sns
%matplotlib inline
```

```
Data
Seaborn comes with built-in data sets
```

```
tips = sns.load_dataset("tips")
```

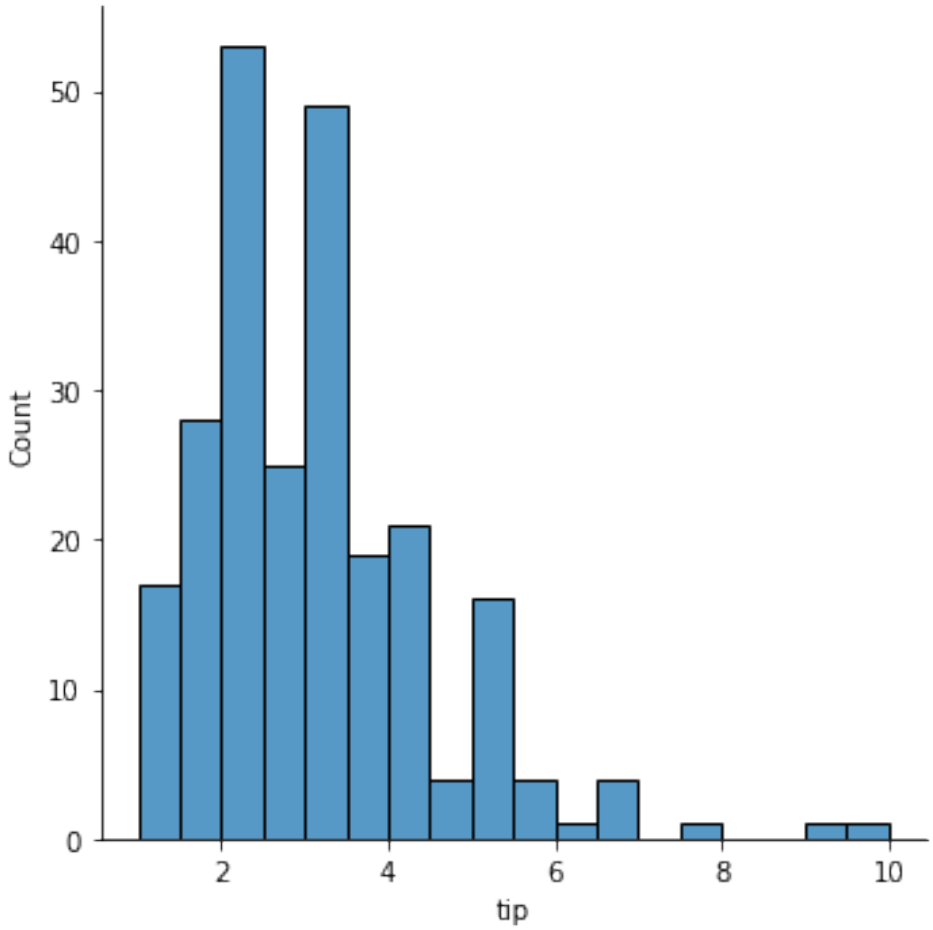
```
tips.head()
```

```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female     No  Sun  Dinner     4
```

```
displot
The displot shows the distribution of a univariate set of
observations.
```

```
sns.displot(tips["tip"])
```

```
<seaborn.axisgrid.FacetGrid at 0x7fdc95e60350>
```

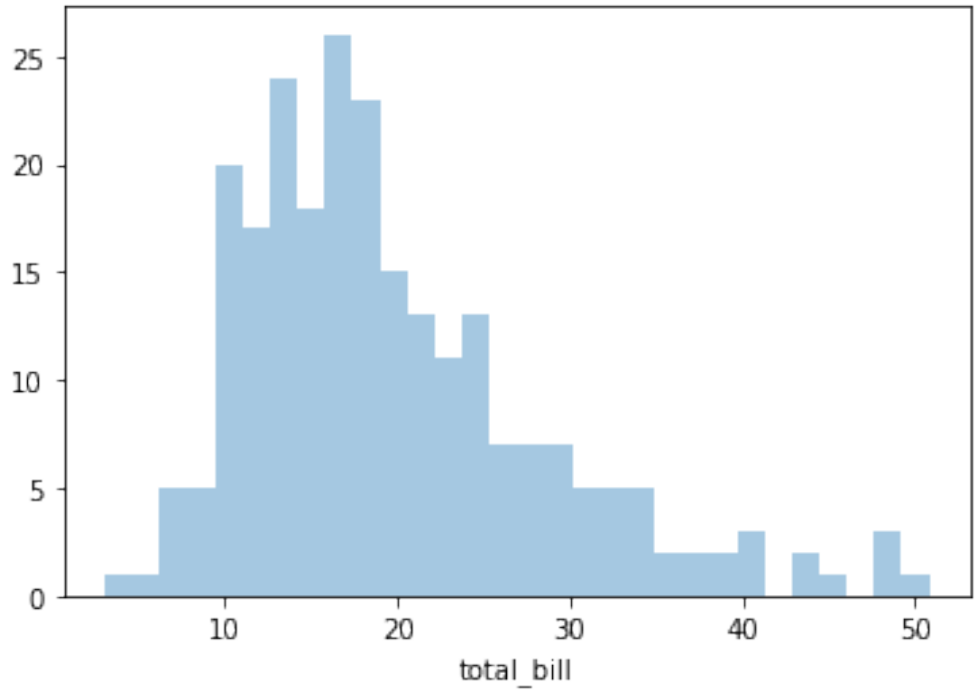


to remove the kde layer **and** just have the histogram use

```
sns.distplot(tips['total_bill'],kde=False,bins=30)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdc8b44dd10>
```



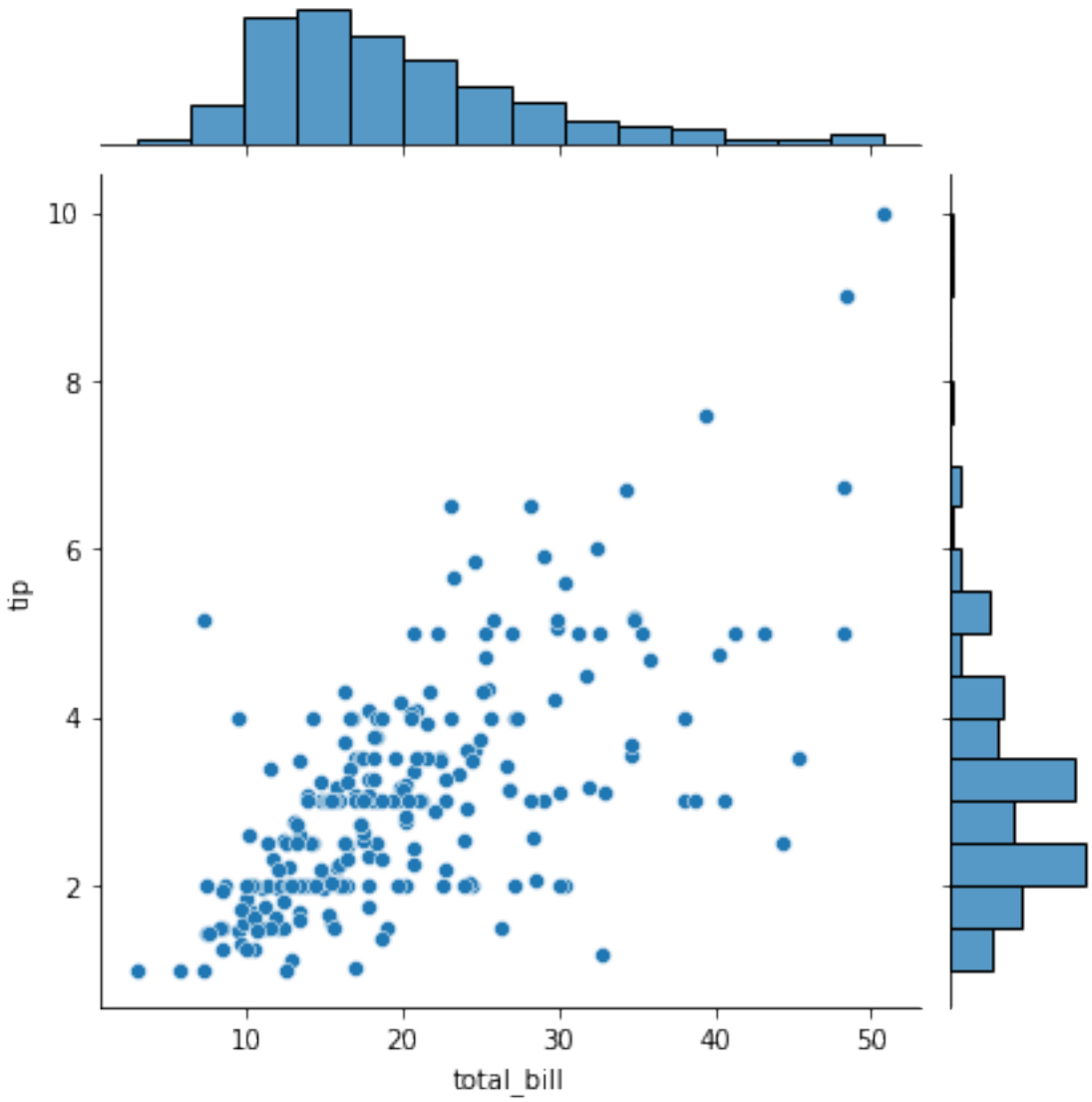
Jointplot

jointplot allows to basically match up two distplots for bivariate data. with your choice of what kind parameter to compare with:

1. "scatter"
2. 'reg'
3. 'resid'
4. 'kde' 5.'hex'

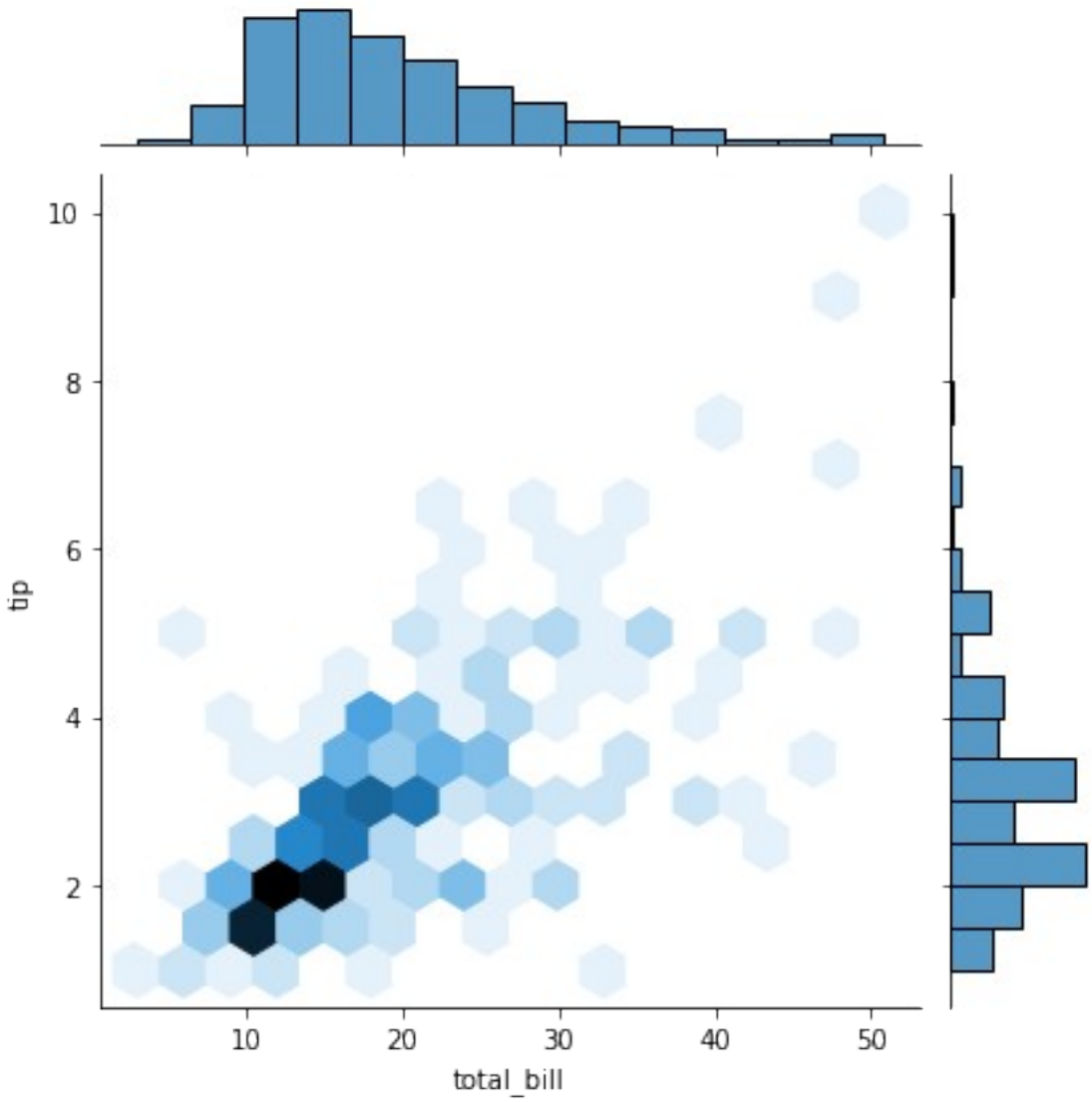
```
sns.jointplot(x ='total_bill',y='tip',data=tips,kind='scatter')
```

```
<seaborn.axisgrid.JointGrid at 0x7fdc8b3bf810>
```



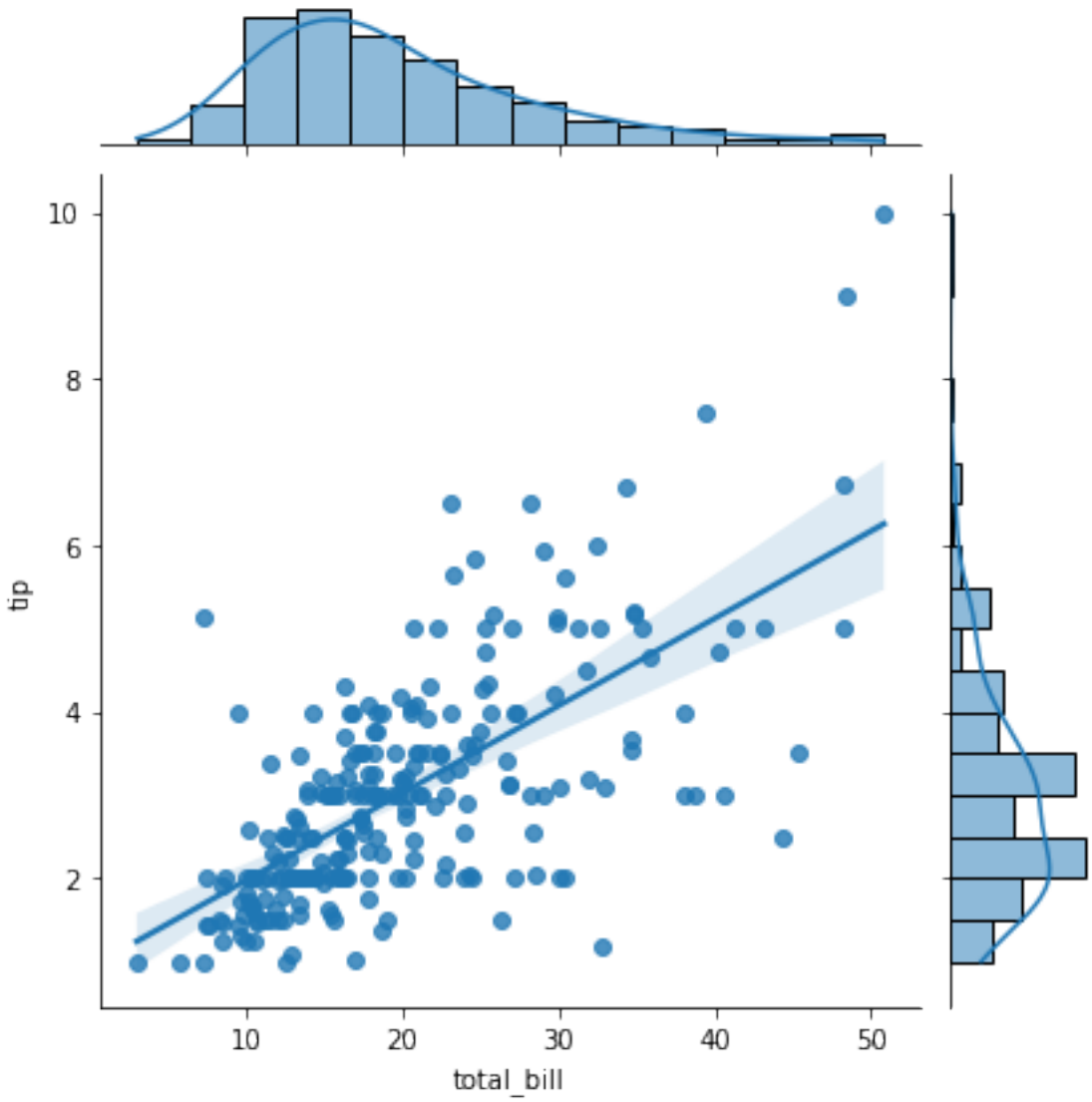

```
sns.jointplot(x='total_bill',y='tip',data=tips,kind='hex')
```

```
<seaborn.axisgrid.JointGrid at 0x7fdc8b200b50>
```



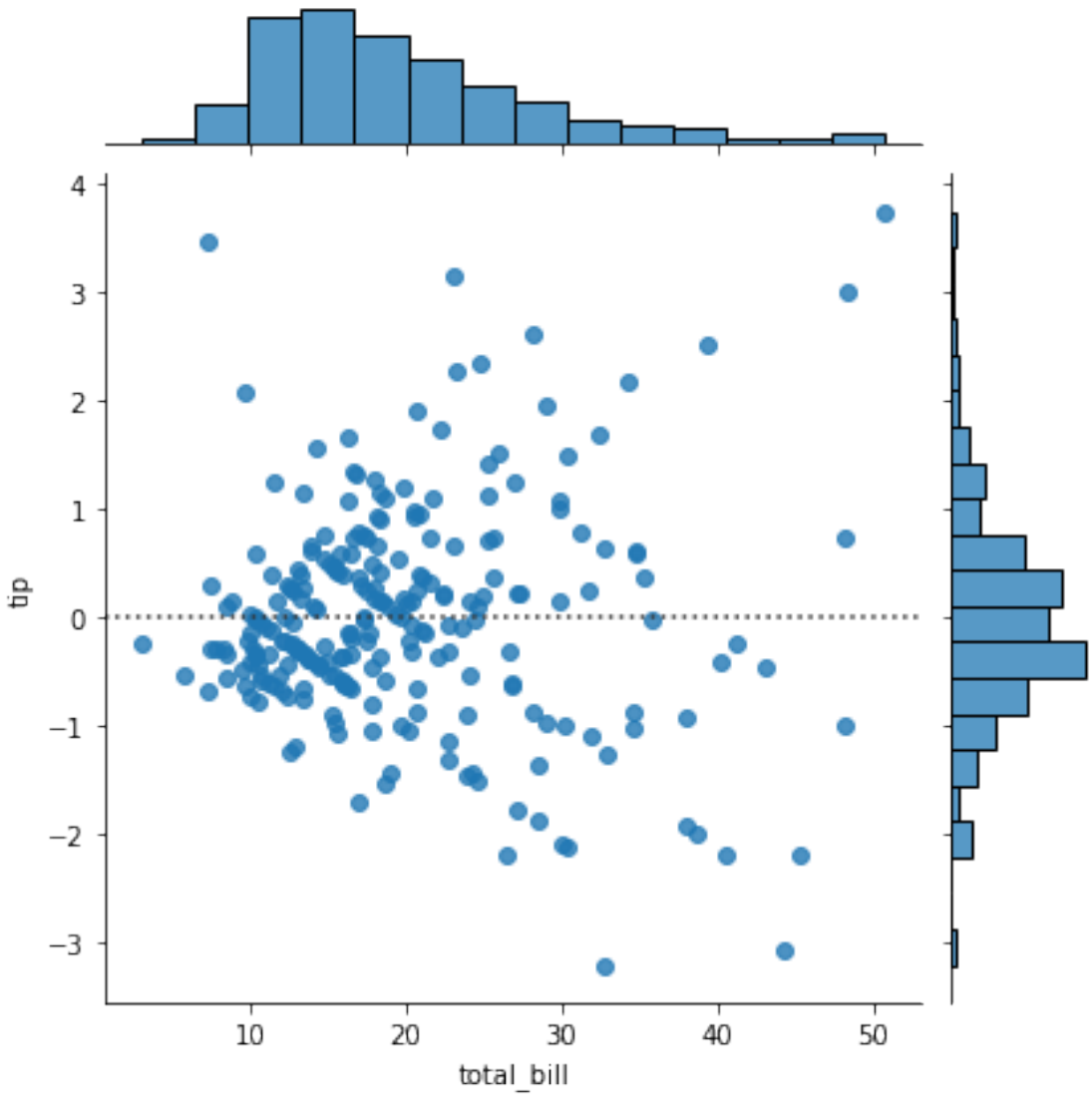

```
sns.jointplot(x='total_bill',y='tip',data=tips,kind='reg')
```

```
<seaborn.axisgrid.JointGrid at 0x7fdc8b3736d0>
```



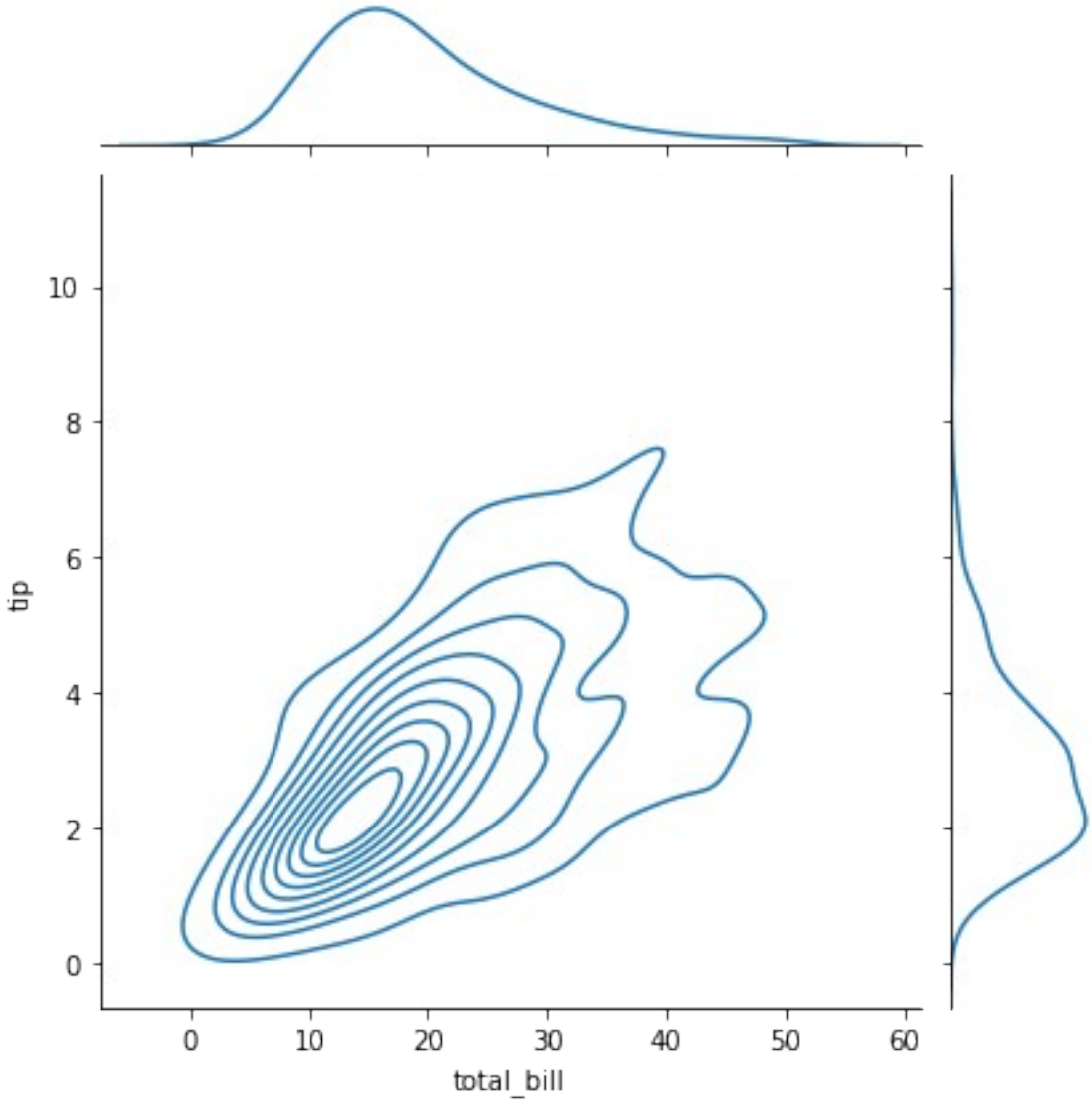

```
sns.jointplot(x='total_bill',y='tip',data=tips,kind='resid')
```

```
<seaborn.axisgrid.JointGrid at 0x7fdc88f72050>
```




```
sns.jointplot(x='total_bill',y='tip',data=tips,kind='kde')
```

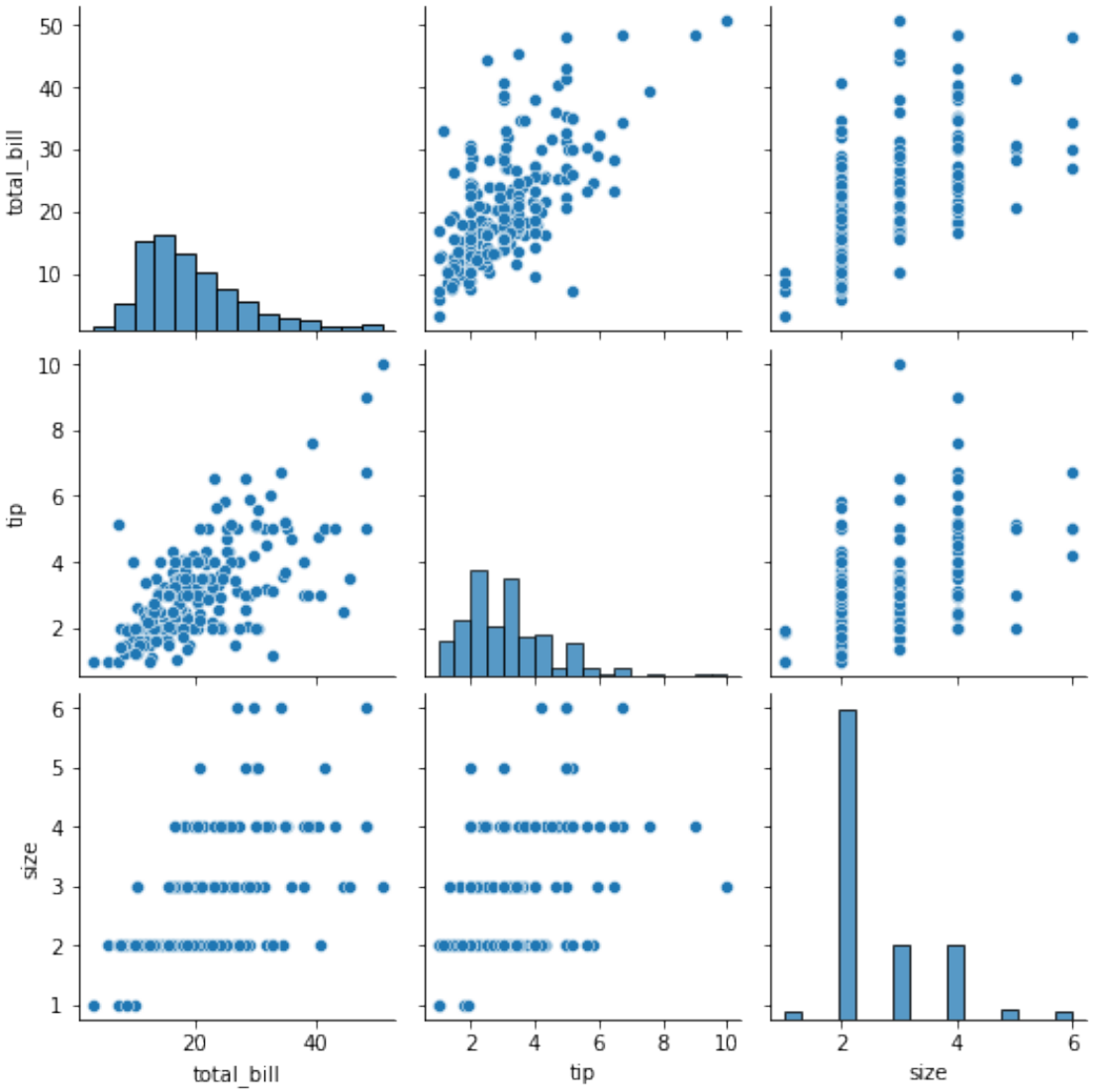
```
<seaborn.axisgrid.JointGrid at 0x7fdc88e0e3d0>
```



pairplot
pairplot will plot pairwise relationships across an entire
dataframe(for the numerical columns)
and supports a color hue argument(for categorical columns).

```
sns.pairplot(tips)
```

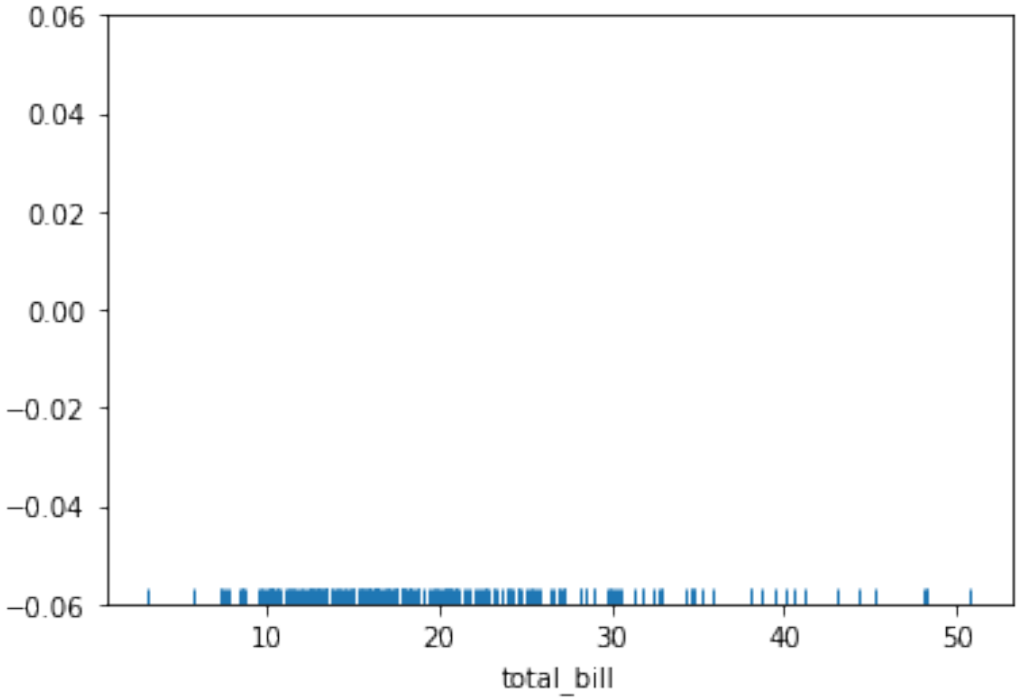
```
<seaborn.axisgrid.PairGrid at 0x7fdc88e0e110>
```




```
rugplot
rugplots are actually a very simple concept, they just draw a dash
mark for every point on a univariate distribution
they are the building blocks of a KDE plot.

sns.rugplot(tips['total_bill'])

<matplotlib.axes._subplots.AxesSubplot at 0x7fdc8861d310>
```


0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
10
20
30
40
50
total_bill

CATEGORICAL PLOTS

1. factorplot
2. boxplott 3.violinplot
3. stripplot
4. swamplot
5. barplot 7.countplot

```
import seaborn as sns
%matplotlib inline
```

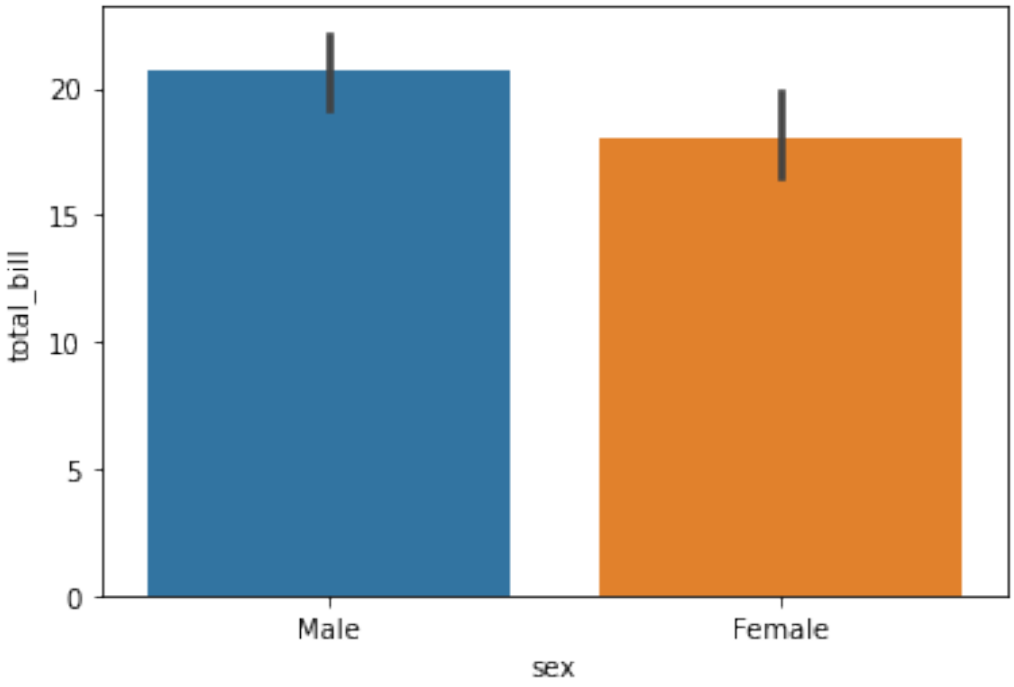
```
tips = sns.load_dataset('tips')
tips.head()
```

```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female     No  Sun  Dinner     4
```

```
barplot and countplot
These are similar plots allow to get aggregate data off a categorical
feature in your data.
barplot is a general plot that allows to aggregate the categorical
data based on some function,by default the mean"
```

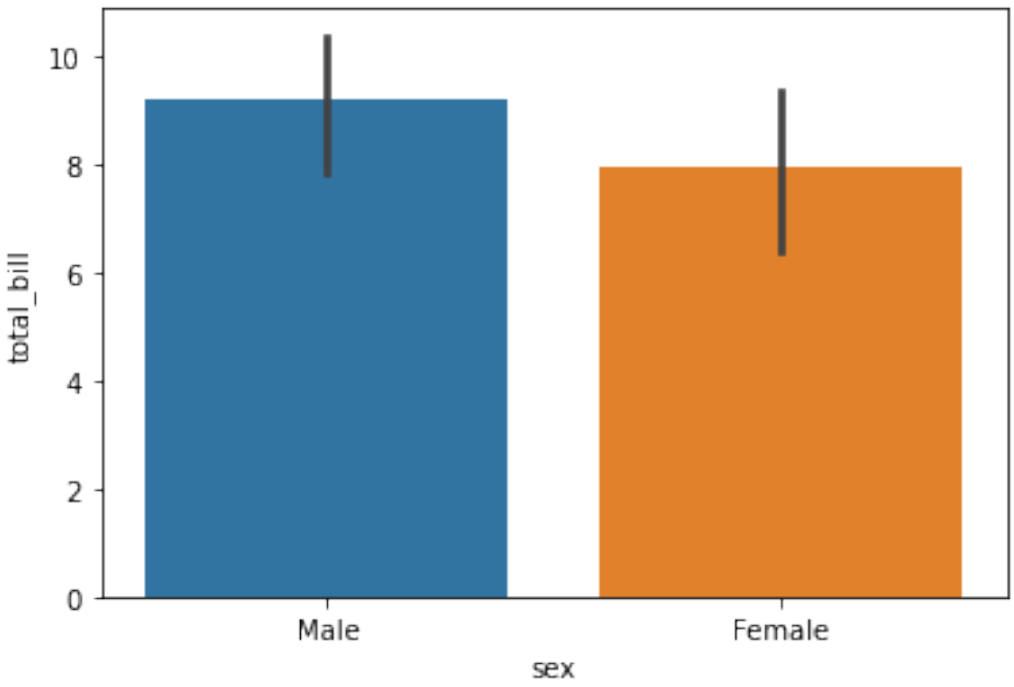
```
sns.barplot(x= 'sex',y='total_bill',data=tips)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f39503b3110>
```




```
import numpy as np
sns.barplot(x= 'sex',y='total_bill',data=tips,estimator=np.std)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f39551f4d10>
```

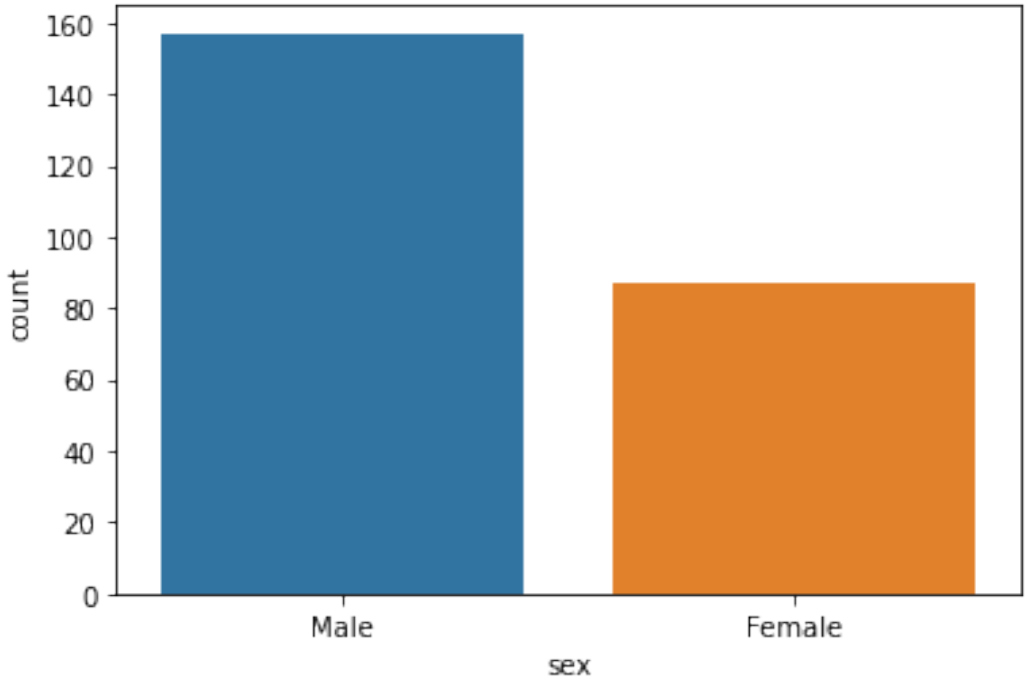


countplot
this is essentially the same as barplot except the estimator is explicitly counting the number of occurences. Which is why we only

pass the x
value:

```
sns.countplot(x='sex',data=tips)
```

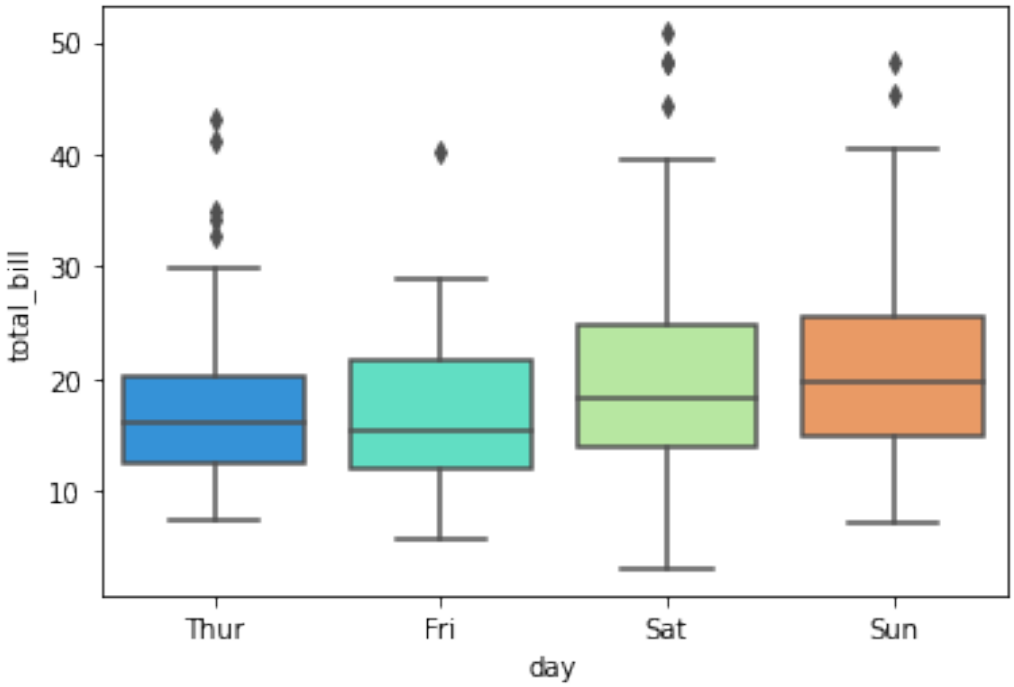
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f394fe26e90>
```



boxplot and voilinplot
boxplot and voilinplot are used to show th e distribution of
categorical datas. A boxplot shows the distribution of quantitative
data in a way
that facilitates comparisons between variables or across levels of a
categorical varibles. The box shows the quartiles of the dataset while
the
whiskers extend to show the rest of the distribution, except fro
points that are determined to be 'outliers' using a method that is a
function of
the inter_quartile range.

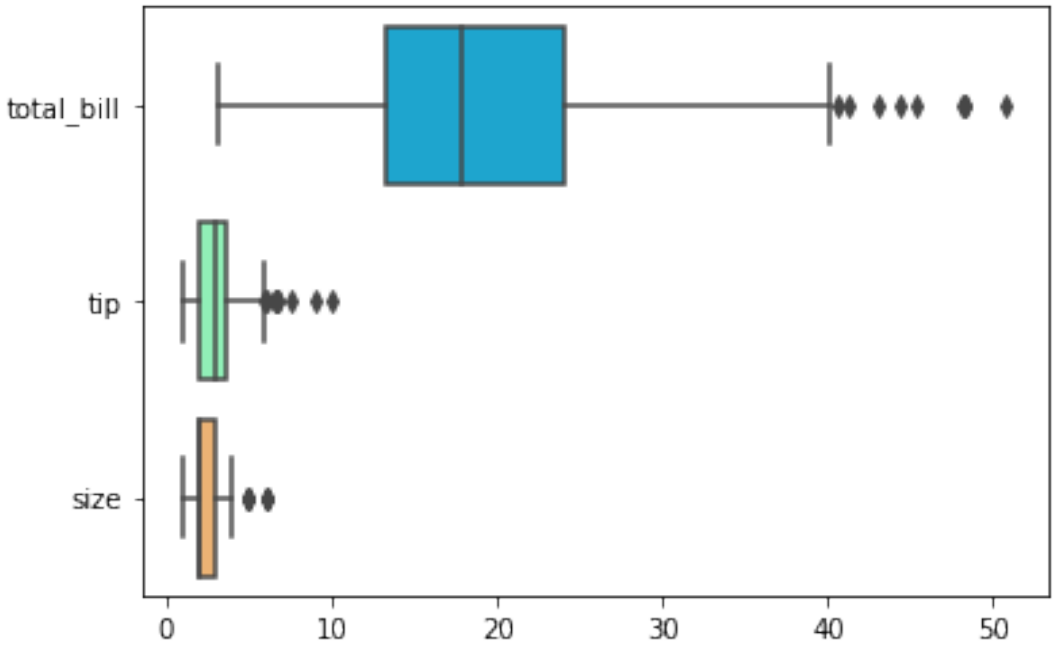
```
sns.boxplot(x='day',y='total_bill',data=tips,palette='rainbow')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f394fda03d0>
```



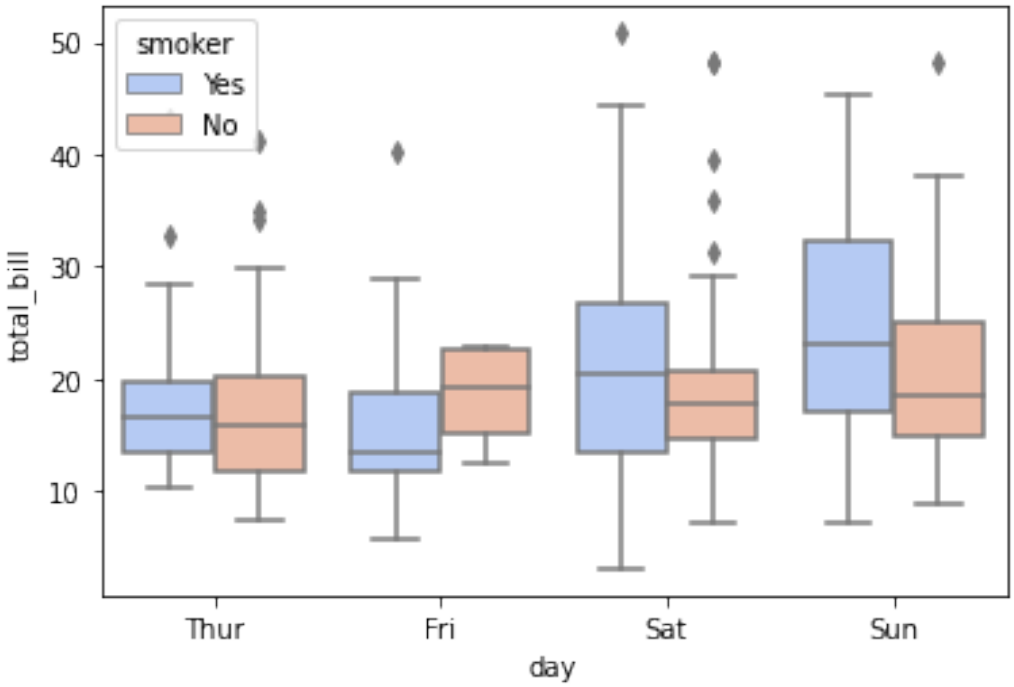

```
sns.boxplot(data=tips,palette='rainbow',orient='h')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f394fce9a90>
```




```
sns.boxplot(x='day',y='total_bill',hue ='smoker',
data=tips,palette='coolwarm')
```

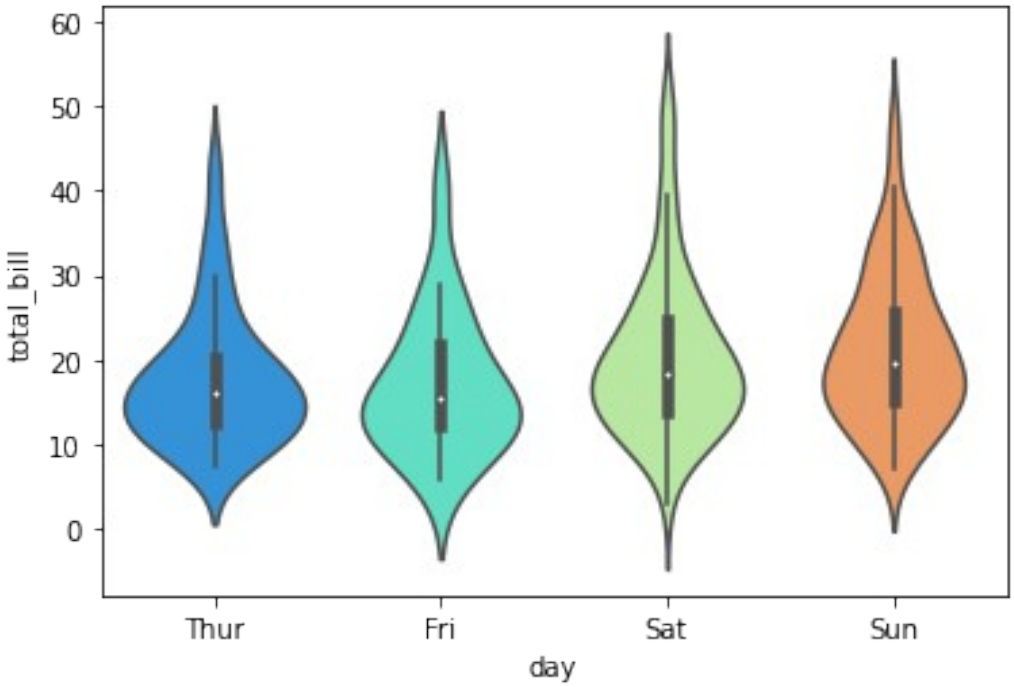
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f394fc05610>
```



voilinplot

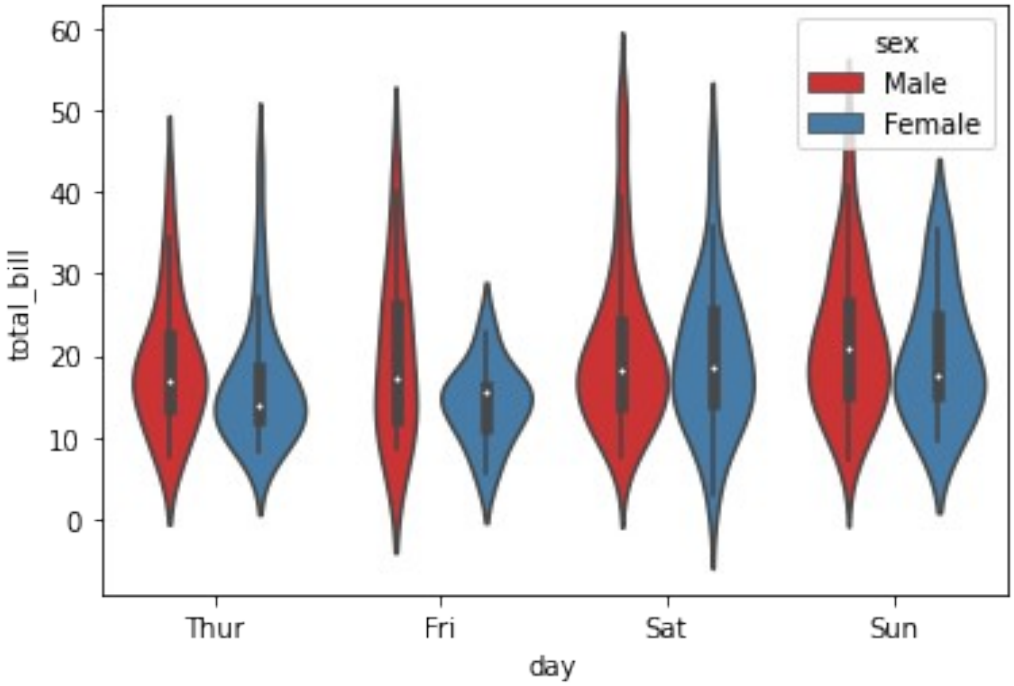
```
sns.violinplot(x='day',y='total_bill',data=tips,palette='rainbow')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f394fab3f50>
```



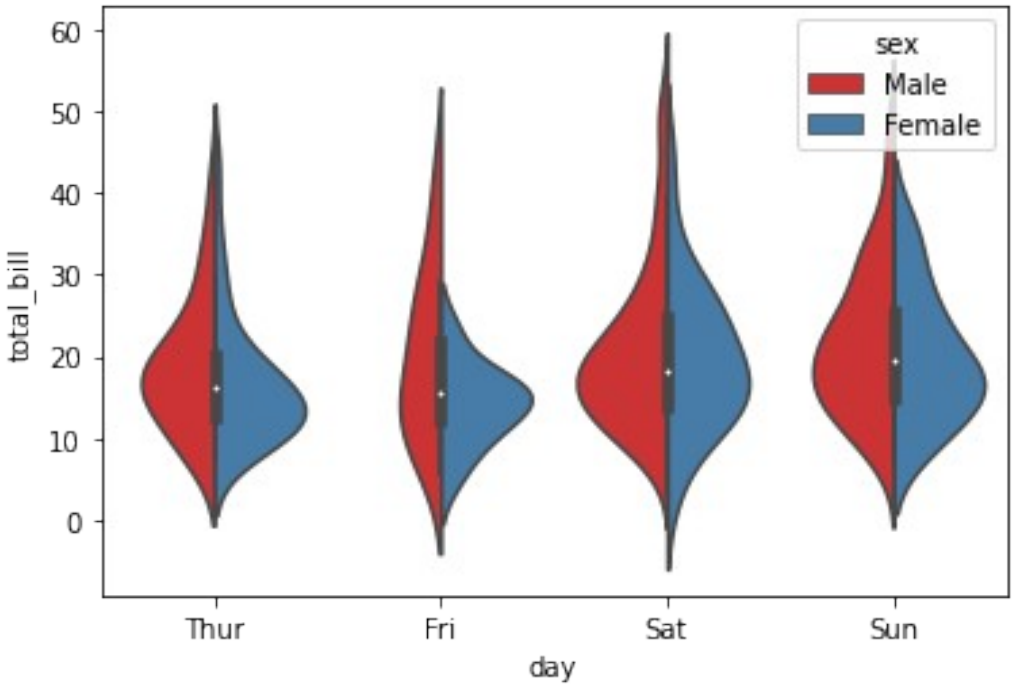

```
sns.violinplot(x='day',y='total_bill',data=tips,hue =
'sex',palette='Set1')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f394c7b0fd0>
```




```
sns.violinplot(x='day',y='total_bill',data=tips,hue = 'sex',split=
True,palette='Set1')
```

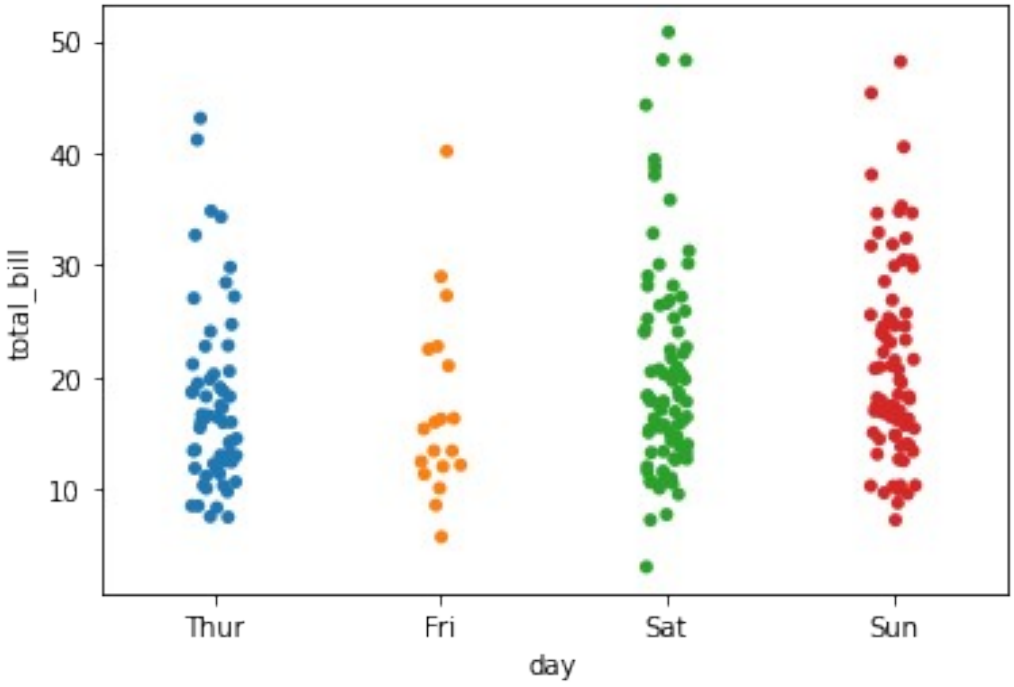
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f394c693350>
```




```
stripplot
```

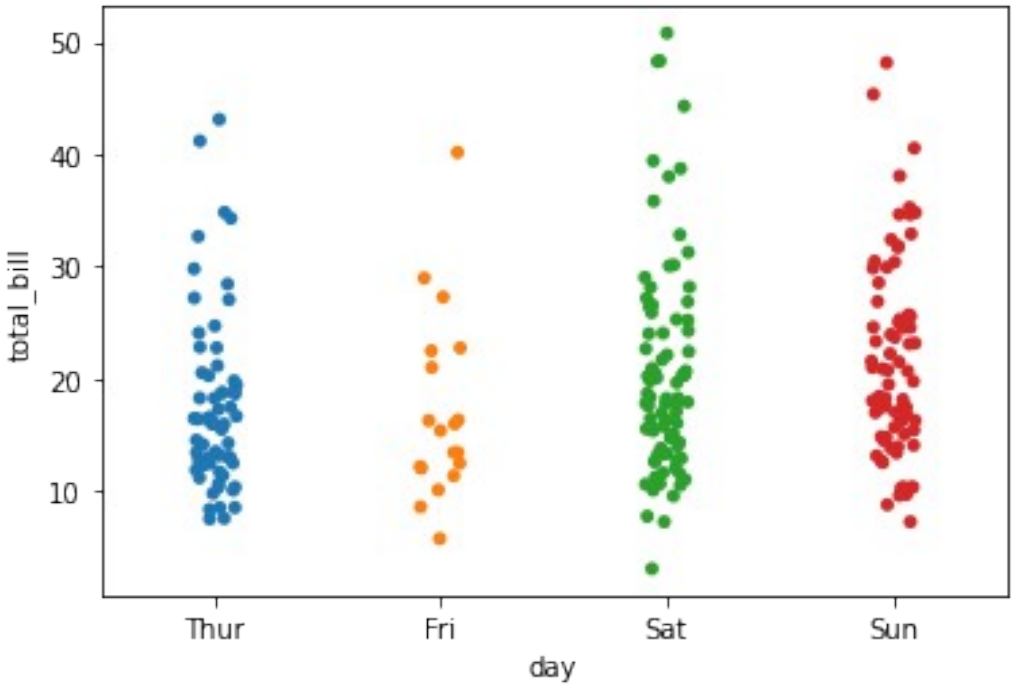
```
sns.stripplot(x='day', y='total_bill',data=tips)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f395035fe50>
```



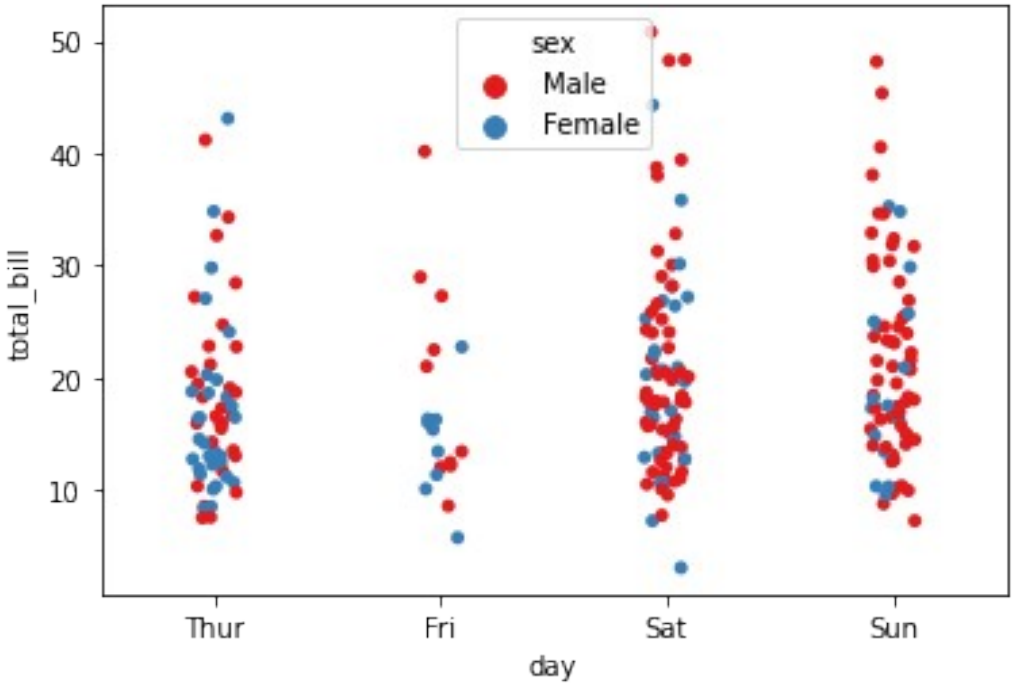

```
sns.stripplot(x='day', y='total_bill',data=tips,jitter=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f394c66d410>
```



```
sns.stripplot(x='day',
y='total_bill',data=tips,jitter=True,hue='sex',palette='Set1')
```

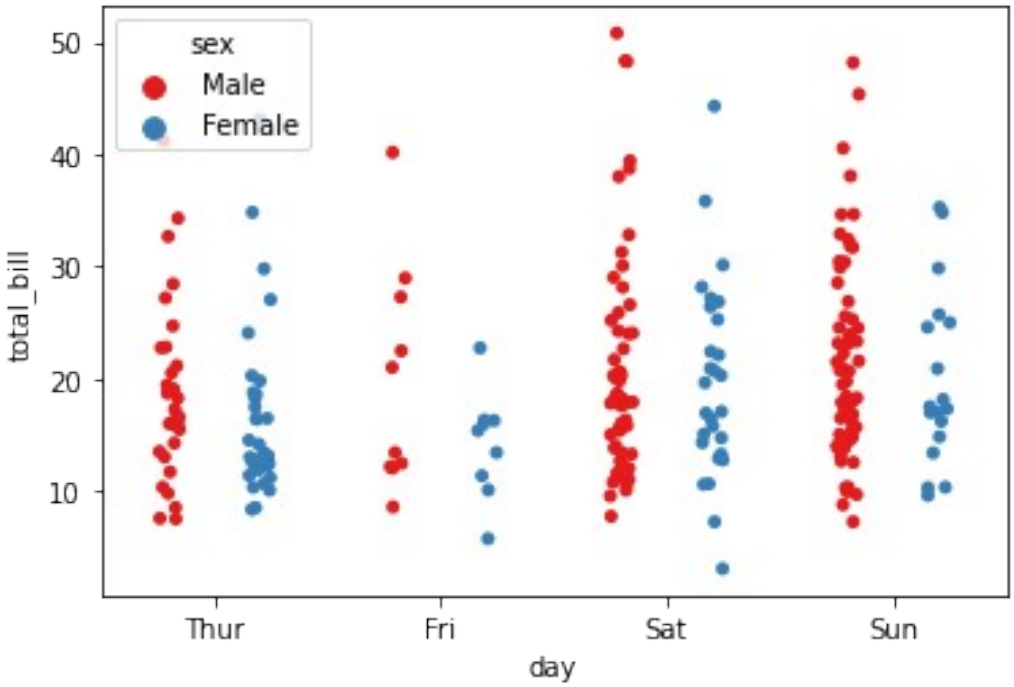
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f394c6832d0>
```




```
sns.stripplot(x='day',
y='total_bill',data=tips,jitter=True,hue='sex',palette='Set1',split =
True)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:2805:
UserWarning: The `split` parameter has been renamed to `dodge`.
  warnings.warn(msg, UserWarning)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f394c4ddf90>
```

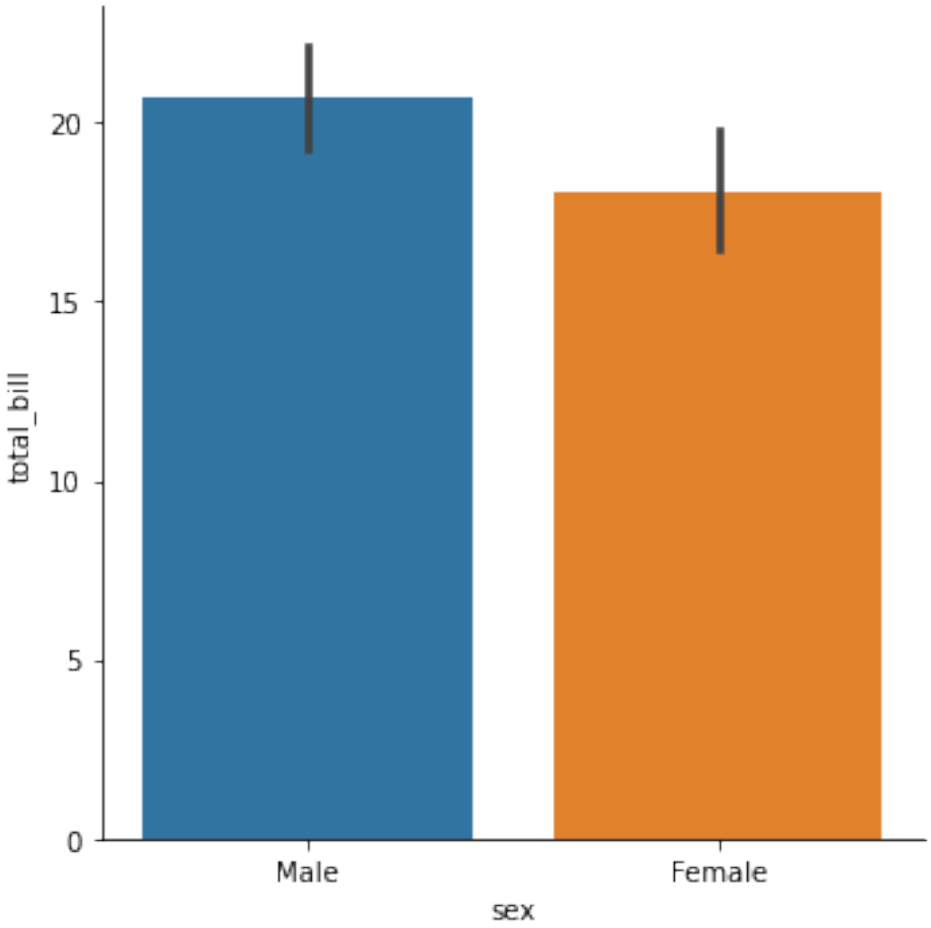



```
factorplot
factorplot is the most general form of a categorical plot. it can take
in a kind parameter to adjust the plot type:

sns.factorplot(x='sex',y='total_bill',data=tips,kind='bar')

/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:3717:
UserWarning: The `factorplot` function has been renamed to `catplot`.
The original name will be removed in a future release. Please update
your code. Note that the default `kind` in `factorplot` (``'point'``)
has changed ``'strip'`` in `catplot`.
  warnings.warn(msg)

<seaborn.axisgrid.FacetGrid at 0x7f394c3f1450>
```


20
15
10
5
0
total_bill
Male
Female
sex

PROGRAM NO-4

Aim: Programs to handle data using pandas.

Series #datatype used in pandas

```
import pandas as pd
import numpy as np
```

To convert numpy array, list into series

```
labels = ['a','b','c']
list1 = [10,20,30]
a = np.array([10,20,30])
d= {'a':10,'b':20,'c':30}

pd.Series(data = list1)
```

```
0    10
1    20
2    30
dtype: int64
```

```
pd.Series(data = list1,index = labels)
```

```
a    10
b    20
c    30
dtype: int64
```

```
pd.Series(list1,labels)
```

```
a    10
b    20
c    30
dtype: int64
```

```
pd.Series(a)
```

```
0    10
1    20
2    30
dtype: int64
```

```
pd.Series(d)
```

```
a    10
b    20
c    30
dtype: int64
```

```
pd.Series(a,labels)
```

```
a    10
b    20
c    30
dtype: int64
```

```
ser1 = pd.Series([10,20,30,40], index =
['apple','mango','grapes','cherry'])
```

```
ser2 = pd.Series([10,30,40,50], index =
['apple','grapes','cherry','kiwi'])
```

```
ser1 +ser2
```

```
apple     20.0
cherry    80.0
grapes    60.0
kiwi       NaN
mango      NaN
dtype: float64
```

```
ser1["mango"]
```

```
20
```

Dataframes #collect of number of series

```
from numpy.random import randn
np.random.seed(101)
```

```
df = pd.DataFrame(randn(5,4),index='A B C D E'.split(),columns='W X Y
Z'.split())
```

```
df
```

```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

Selection and indexing

```
df['Z']
```

```
A    0.503826
B    0.605965
C   -0.589001
D    0.955057
E    0.683509
Name: Z, dtype: float64
```

```
df[['Z','X']]
```

```
          Z         X
A  0.503826  0.628133
B  0.605965 -0.319318
C -0.589001  0.740122
D  0.955057 -0.758872
E  0.683509  1.978757
```

Creating a new column

```
df['new'] = df['Z'] + df['W']
```

```
df
```

```
          W         X         Y         Z       new
A  2.706850  0.628133  0.907969  0.503826  3.210676
B  0.651118 -0.319318 -0.848077  0.605965  1.257083
C -2.018168  0.740122  0.528813 -0.589001 -2.607169
D  0.188695 -0.758872 -0.933237  0.955057  1.143752
E  0.190794  1.978757  2.605967  0.683509  0.874303
```

```
df.drop(columns ='new',axis=1)
```

```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

```
df.drop(columns ='new')
```

```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

```
df
```

```
          W         X         Y         Z       new
A  2.706850  0.628133  0.907969  0.503826  3.210676
B  0.651118 -0.319318 -0.848077  0.605965  1.257083
C -2.018168  0.740122  0.528813 -0.589001 -2.607169
D  0.188695 -0.758872 -0.933237  0.955057  1.143752
E  0.190794  1.978757  2.605967  0.683509  0.874303
```

```
df.drop(columns ='new',axis = 1,inplace=True)
```

```
df
```

```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
```

```
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

```
df
```

```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

selecting row

```
df.loc['A']
```

```
W    2.706850
X    0.628133
Y    0.907969
Z    0.503826
Name: A, dtype: float64
```

```
df.iloc[2]
```

```
W   -2.018168
X    0.740122
Y    0.528813
Z   -0.589001
Name: C, dtype: float64
```

Selecting a subset of rows and columns

- List item
- List item
- List item
- List item

```
df.loc['A','Y']
```

```
0.9079694464765431
```

```
df.loc[['A','B'],['X','Y']]
```

```
          X         Y
A  0.628133  0.907969
B -0.319318 -0.848077
```

Conditional selection

```
df
```

```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

```
df>0
```

```
       W      X      Y      Z
A   True   True   True   True
B   True  False  False   True
C  False   True   True  False
D   True  False  False   True
E   True   True   True   True
```

```
df[df>0]
```

```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118       NaN       NaN  0.605965
C       NaN  0.740122  0.528813       NaN
D  0.188695       NaN       NaN  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

```
df[df['W']>0]
```

```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

```
df['W']>0
```

```
A     True
B     True
C    False
D     True
E     True
Name: W, dtype: bool
```

```
df[df['W']>0] ['Y']
```

```
A    0.907969
B   -0.848077
D   -0.933237
E    2.605967
Name: Y, dtype: float64
```

```
df[df['W']>0][['X','Y']]
```

```
          X         Y
A  0.628133  0.907969
B -0.319318 -0.848077
D -0.758872 -0.933237
E  1.978757  2.605967
```

```
df[(df['W']>0) & (df['Y']>1)]
```

```
          W         X         Y         Z
E  0.190794  1.978757  2.605967  0.683509
```

OPERATIONS IN PANDAS

```
df = pd.DataFrame({'col1':[1,2,3,4],'col2':[44,55,66,44],'col3':
['abc','def','ghi','xyz']})
df.head()
```

```
   col1  col2 col3
0     1    44  abc
1     2    55  def
2     3    66  ghi
3     4    44  xyz
```

```
df['col2'].unique()
```

```
array([44, 55, 66])
```

```
df['col3'].nunique()
```

```
4
```

```
df['col3'].value_counts()
```

```
abc    1
def    1
ghi    1
xyz    1
Name: col3, dtype: int64
```

Selecting data

```
#select from dataframe using criteria from multiple columns
newdf= df[(df['col1']>2) & (df['col2']==44)]
```

```
newdf
```

```
   col1  col2 col3
3     4    44  xyz
```

Applying Function

```
def times2(x):
  return x*2
```

```
df['col1'].apply(times2)
```

```
0    2
1    4
2    6
3    8
Name: col1, dtype: int64
```

```
df['col3'].apply(len)
```

```
0    3
1    3
2    3
3    3
Name: col3, dtype: int64
```

```
df['col1'].sum()
```

```
10
```

```
del df['col1']
```

```
df
```

```
   col2 col3
0    44  abc
1    55  def
2    66  ghi
3    44  xyz
```

```
df.columns
```

```
Index(['col2', 'col3'], dtype='object')
```

```
df.index
```

```
RangeIndex(start=0, stop=4, step=1)
```

Sorting and Ordering a dataframe

```
df.sort_values(by='col2') #inplace = false by default
```

```
   col2 col3
0    44  abc
3    44  xyz
1    55  def
2    66  ghi
```

```
df.sort_values(by='col2', inplace = True)
```

```
df
```

```
   col2 col3
0    44  abc
3    44  xyz
1    55  def
2    66  ghi
```

Find null values or check for null values

```
df.isnull()
```

```
    col2   col3
0  False  False
3  False  False
1  False  False
2  False  False
```

Result: The program is executed successfully and obtained the output.

Pandas using dataset

Dataset: Salaries.csv

```
import pandas as pd
import numpy as np

df=pd.read_csv('/content/Salaries.csv')

df
```

```
             Id       EmployeeName  ...         Agency  Status
0             1     NATHANIEL FORD  ...  San Francisco     NaN
1             2       GARY JIMENEZ  ...  San Francisco     NaN
2             3     ALBERT PARDINI  ...  San Francisco     NaN
3             4  CHRISTOPHER CHONG  ...  San Francisco     NaN
4             5    PATRICK GARDNER  ...  San Francisco     NaN
...         ...                ...  ...            ...     ...
148649   148650      Roy I Tillery  ...  San Francisco     NaN
148650   148651       Not provided  ...  San Francisco     NaN
148651   148652       Not provided  ...  San Francisco     NaN
148652   148653       Not provided  ...  San Francisco     NaN
148653   148654          Joe Lopez  ...  San Francisco     NaN

[148654 rows x 13 columns]
```

```
df.head()
```

```
   Id       EmployeeName  ...         Agency  Status
0   1     NATHANIEL FORD  ...  San Francisco     NaN
1   2       GARY JIMENEZ  ...  San Francisco     NaN
2   3     ALBERT PARDINI  ...  San Francisco     NaN
3   4  CHRISTOPHER CHONG  ...  San Francisco     NaN
4   5    PATRICK GARDNER  ...  San Francisco     NaN

[5 rows x 13 columns]
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148654 entries, 0 to 148653
Data columns (total 13 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Id                148654 non-null  int64
 1   EmployeeName      148654 non-null  object
 2   JobTitle          148654 non-null  object
 3   BasePay           148045 non-null  float64
 4   OvertimePay       148650 non-null  float64
 5   OtherPay          148650 non-null  float64
 6   Benefits          112491 non-null  float64
```

```
 7   TotalPay          148654 non-null  float64
 8   TotalPayBenefits  148654 non-null  float64
 9   Year              148654 non-null  int64
 10  Notes             0 non-null       float64
 11  Agency            148654 non-null  object
 12  Status            0 non-null       float64
dtypes: float64(8), int64(2), object(3)
memory usage: 14.7+ MB
```

what is the average BasePay?

```
df['BasePay'].mean()
```

```
66325.44884050643
```

what is the highest amount of OvertimePay in the dataset?

```
df['OvertimePay'].max()
```

```
245131.88
```

what is the job title of JOSEPH DRISCOLL? Note: Use all caps,otherwise you may get answer that doesn't match up

```
df[df['EmployeeName'] == 'JOSEPH DRISCOLL']['JobTitle']
```

```
24    CAPTAIN, FIRE SUPPRESSION
Name: JobTitle, dtype: object
```

HOW MUCH DOES JOSEPH DRISCOLL MAKE(INCLUDING BENEFITS)?

```
df['TotalPayBenefits'] [df['EmployeeName']== 'JOSEPH DRISCOLL']
```

```
24    270324.91
Name: TotalPayBenefits, dtype: float64
```

WHAT IS THE NAME OF THE HIGHEST PAID PERSON(INCLUDING BENEFITS)?

```
df[df['TotalPay'].max()==df['TotalPay']]
```

```
   Id    EmployeeName  ...         Agency  Status
0   1  NATHANIEL FORD  ...  San Francisco     NaN

[1 rows x 13 columns]
```

WHAT IS THE NAME OF THE LOWEST PAID PERSON(INCLUDING BENEFITS)?

```
df[df['TotalPay'].min()==df['TotalPay']]
```

```
            Id EmployeeName  ...         Agency  Status
148653  148654    Joe Lopez  ...  San Francisco     NaN

[1 rows x 13 columns]
```

WHAT WAS THE AVERAGE(MEAN) BasePay OF ALL EMPLOYEES PER YEAR? (2011-2014)

```
df.groupby('Year').mean()['BasePay']
```

```
Year
2011    63595.956517
2012    65436.406857
2013    69630.030216
2014    66564.421924
Name: BasePay, dtype: float64
```

HOW MANY UNIQUE JOB TITLES ARE THERE?

```
df['JobTitle'].nunique()
```

```
2159
```

WHAT ARE THE TOP 5 MOST COMMON JOBS?

```
df['JobTitle'].value_counts().head()
```

```
Transit Operator                7036
Special Nurse                   4389
Registered Nurse                3736
Public Svc Aide-Public Works    2518
Police Officer 3                2421
Name: JobTitle, dtype: int64
```

HOW MANY JOB TITLES WERE REPRESENTED BY ONLY ONE PERSON IN 2013?(EG: JOB TITLE WITH ONLY ONE OCCURENCE IN 2013)

```
(df[df['Year']==2013]['JobTitle'].value_counts()==1).sum()
```

```
202
```

HOW MANY PEOPLE HAVE THE WORD CHIEF IN THEIR JOB TITLE?

```
df['JobTitle'].apply(lambda str:('chief' in str.lower())).sum()
```

```
627
```

```
def find_chief(job_title):
    if 'chief' in job_title.lower().split():
        return True
    else:
        return False

df = pd.read_csv('Salaries.csv')

sum(df['JobTitle'].apply(lambda x: find_chief(x)))
```

```
477
```

Result: The program executed successfully and obtained the output.