COURSE OUTCOME 1

PROGRAM - 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
```

```
'hello'
```

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

```
'hello world'
```

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']
```

```
'item2'
```

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 executed successfully and obtained the output.

PROGRAM - 2

AIM:

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

```
import numpy as pd
```

```
n=[1,2,3,4,5]
print (n)
```

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

```
pd.array(n)
```

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

```
m=[[1,2,3],[4,5,6],[7,8,9]]
pd.array(m)
```

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

```
pd.arange(0,10)
```

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

```
pd.arange(0,14,3)
```

```
array([ 0,  3,  6,  9, 12])
```

```
pd.zeros(3)
```

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

```
pd.zeros((5,5))
```

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

```
pd.ones(5)
```

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

```
pd.ones((5,5))
```

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

```
pd.eye(3)
```

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

```
pd.linspace(0,20,5)
```

```
array([ 0.,  5., 10., 15., 20.])
```

```
pd.linspace(1,6,3)
```

```
array([1. , 3.5, 6. ])
```

```
pd.linspace(0,100,10)
```

```
array([  0.        ,  11.11111111,  22.22222222,  33.33333333,
        44.44444444,  55.55555556,  66.66666667,  77.77777778,
        88.88888889, 100.        ])
```

```
pd.random.rand(5)
```

```
array([0.36673517, 0.52038518, 0.8821126 , 0.49032897, 0.78885939])
```

```
pd.random.rand(2)
```

```
     array([0.59655309, 0.78540801])

pd.random.rand(2,2)

     array([[0.00456417, 0.19040905],
            [0.88668811, 0.01155942]])


pd.random.randn(5)

     array([-0.7543355 , -0.02772456, -1.02653437,  0.82245293, -1.6574864 ])


pd.random.randint(1,10)

     5


pd.random.randint(1,100,10)

     array([58, 59, 35, 19, 85, 91, 28, 32, 30, 70])


arr=pd.arange(25)


pd.arange(25)

     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])


ranarr=pd.random.randint(0,25,5)


ranarr

     array([16, 11,  8, 21,  6])


arr.reshape(5,5)

     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]])



ranarr.max()
```

```
    21
```

```
ranarr.min()
```

```
    6
```

```
ranarr.argmax()
```

```
    3
```

```
ranarr.argmin()
```

```
    4
```

```
arr.shape
```

```
    (25,)
```

```
arr.dtype
```

```
    dtype('int64')
```

```
arr[5]
```

```
    5
```

```
arr[6]
```

```
    6
```

```
arr[1:6]
```

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

```
arr[1:6]=50
```

```
arr
```

```
    array([ 0, 50, 50, 50, 50, 50,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
           17, 18, 19, 20, 21, 22, 23, 24])
```

```
arr.reshape(5,5)
```

```
array([[ 0, 50, 50, 50, 50],
       [50,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

```
arr[1:5][3:5]=0
```

```
arr
```

```
array([ 0, 50, 50, 50,  0, 50,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

```
arr_copy=arr.copy()
```

```
arr
```

```
array([ 0, 50, 50, 50,  0, 50,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

```
slice_of_arr=arr[0:6]
```

```
arr
```

```
array([ 0, 50, 50, 50,  0, 50,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

```
A=pd.array([[5,10,15],[20,25,30],[35,40,45]])
A
```

```
array([[ 5, 10, 15],
       [20, 25, 30],
       [35, 40, 45]])
```

```
A[1:,:2]
```

```
array([[20, 25],
       [35, 40]])
```

```
A[:2,1:]

     array([[10, 15],
            [25, 30]])


A>5

     array([[False,  True,  True],
            [ True,  True,  True],
            [ True,  True,  True]])


A<5

     array([[False, False, False],
            [False, False, False],
            [False, False, False]])


A>10

     array([[False, False,  True],
            [ True,  True,  True],
            [ True,  True,  True]])


A<0

     array([[False, False, False],
            [False, False, False],
            [False, False, False]])


true=A>5


A[true]

     array([10, 15, 20, 25, 30, 35, 40, 45])


import numpy as pd


pd.arange(0,10)

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

```
a=pd.arange(0,10)
a
```

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

```
b=pd.arange(10,20)
```

```
c=pd.add(a,b)
```

```
c
```

```
array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28])
```

```
b
```

```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

```
c=pd.subtract(a,b)
```

```
c
```

```
array([-10, -10, -10, -10, -10, -10, -10, -10, -10, -10])
```

```
c=pd.subtract(b,a)
c
```

```
array([10, 10, 10, 10, 10, 10, 10, 10, 10, 10])
```

```
c=pd.multiply(a,b)
c
```

```
array([  0,  11,  24,  39,  56,  75,  96, 119, 144, 171])
```

```
c=pd.divide(a,b)
c
```

```
array([0.        , 0.09090909, 0.16666667, 0.23076923, 0.28571429,
       0.33333333, 0.375     , 0.41176471, 0.44444444, 0.47368421])
```

```
a+b
```

```
array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28])
```

```
a-b
```

```
array([-10, -10, -10, -10, -10, -10, -10, -10, -10, -10])
```

```
a*b
```

```
array([  0,  11,  24,  39,  56,  75,  96, 119, 144, 171])
```

```
a/b
a/b
```

```
array([0.        , 0.09090909, 0.16666667, 0.23076923, 0.28571429,
       0.33333333, 0.375     , 0.41176471, 0.44444444, 0.47368421])
```

```
a/a
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: RuntimeWarni
  """Entry point for launching an IPython kernel.
array([nan,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

```
1/a
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: RuntimeWarni
  """Entry point for launching an IPython kernel.
array([       inf, 1.        , 0.5       , 0.33333333, 0.25      ,
       0.2       , 0.16666667, 0.14285714, 0.125     , 0.11111111])
```

```
a/0
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: RuntimeWarni
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: RuntimeWarni
  """Entry point for launching an IPython kernel.
array([nan, inf, inf, inf, inf, inf, inf, inf, inf, inf])
```

SVD:

```
import numpy as np
A = np.arange(0,25)
```

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

```
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])
```

```
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 executedsuccessfully and obtained the output

PROGRAM - 3

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

MATPLOTLIB:

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

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

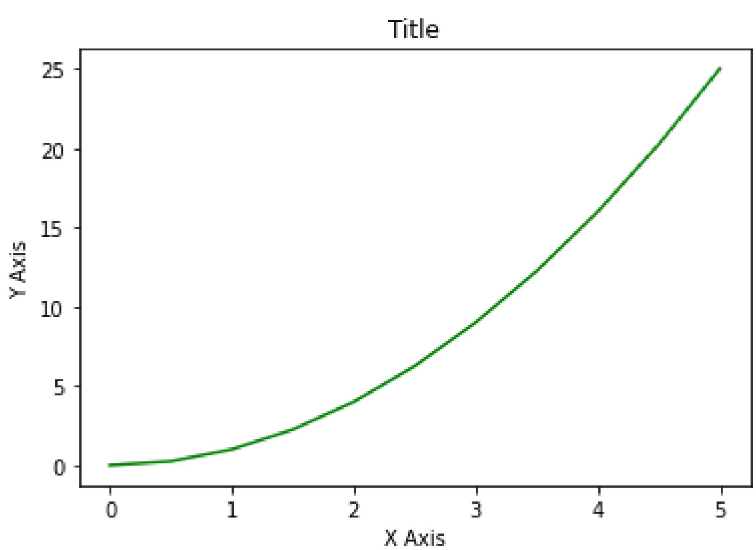
```
x
```

```
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.  ])
```

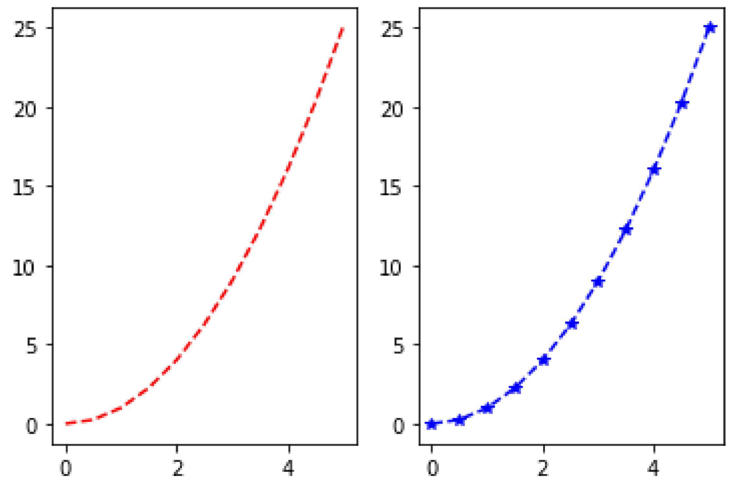
```
plt.plot(x,y,'g')#'r' is the color red
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Title')
plt.show()
```




```
CREATE MULTIPLOTS ON SAME CANVAS::
```

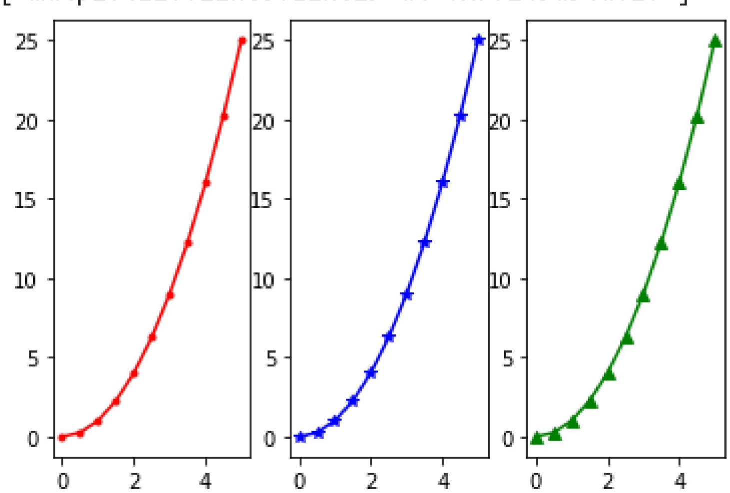
```
#plt.subplot(nrows,ncolumns,plot_number)
plt.subplot(1,2,1)
plt.plot(x,y,'r--')
plt.subplot(1,2,2)
plt.plot(x,y,'b*--')
```

```
[<matplotlib.lines.Line2D at 0x7f243e0f3890>]
```




```
plt.subplot(1,3,1)
plt.plot(x,y,'r.-')
plt.subplot(1,3,2)
plt.plot(x,y,'b*-')
plt.subplot(1,3,3)
plt.plot(x,y,'g^-')
```

```
[<matplotlib.lines.Line2D at 0x7f243d3ca610>]
```

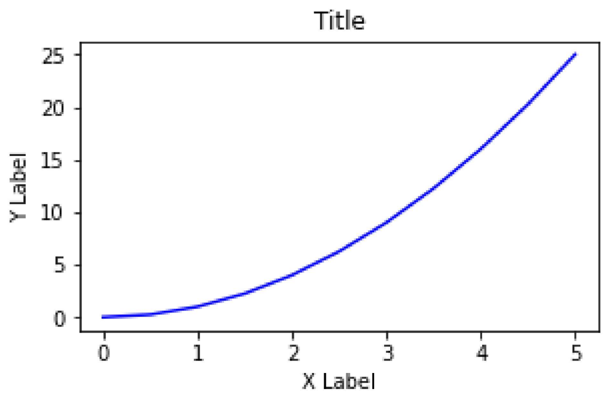


USING OBJECT ORIENTED:::

```
#create figure(empty canvas)
fig=plt.figure()
```

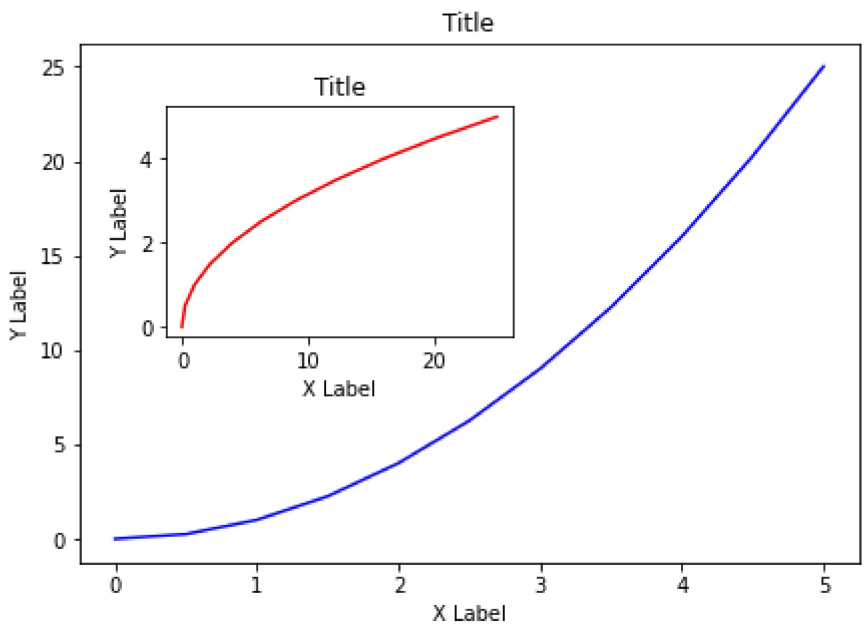
```
#add set of axes to figure
axes=fig.add_axes([0.3,0.7,0.6,0.5]) #left,bottom,width,height(range 0 to 1)
#plot on that set of axes
axes.plot(x,y,'b')
axes.set_xlabel('X Label')  # notice the use of set_ to begin methods
axes.set_ylabel('Y Label')
axes.set_title('Title')
```

```
Text(0.5, 1.0, 'Title')
```




```
#create blank canvas
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])
#larger figure axes1
axes1.plot(x,y,'b')
axes1.set_xlabel('X Label')
axes1.set_ylabel('Y Label')
axes1.set_title('Title')
#smaller figure axes 2
axes2.plot(y,x,'r')
axes2.set_xlabel('X Label')
axes2.set_ylabel('Y Label')
axes2.set_title('Title')
```

```
Text(0.5, 1.0, 'Title')
```

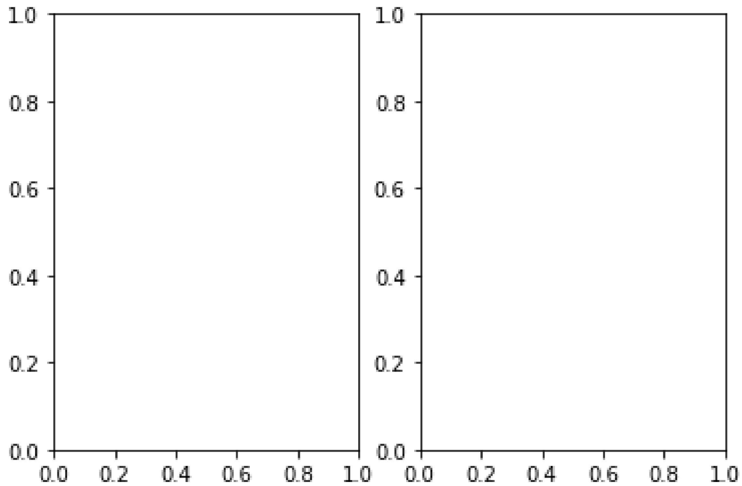


SUBPLOTS::

```
fig,axes=plt.subplots()
axes.plot(x,y,'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');
```

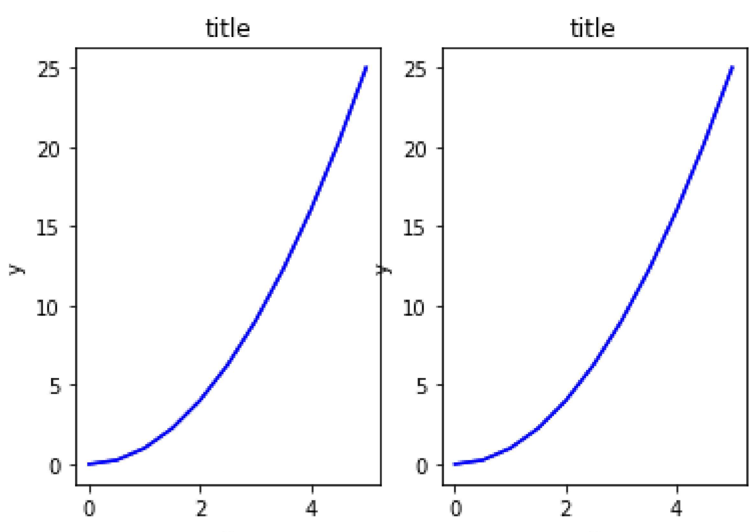
title

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



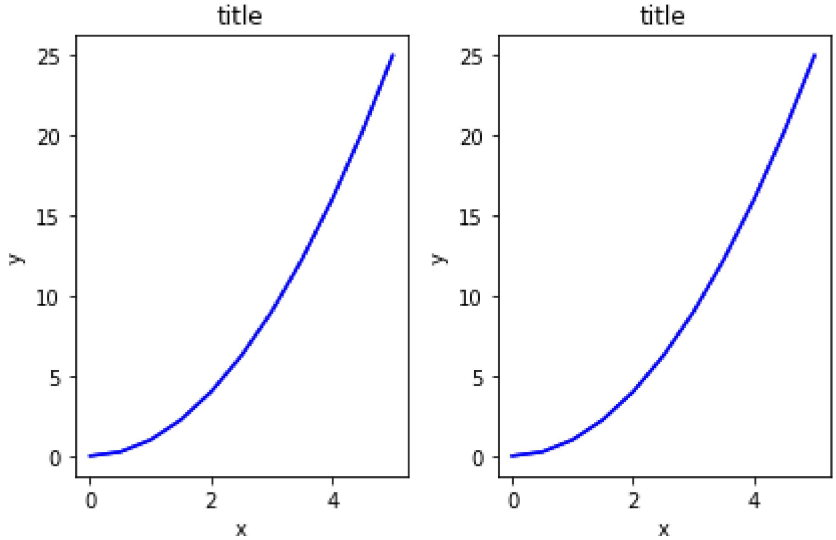

```
for ax in axes:
  ax.plot(x,y,'b')
  ax.set_xlabel('x')
  ax.set_ylabel('y')
  ax.set_title('title')
```

```
fig
```




```
fig.tight_layout()
```

```
fig
```

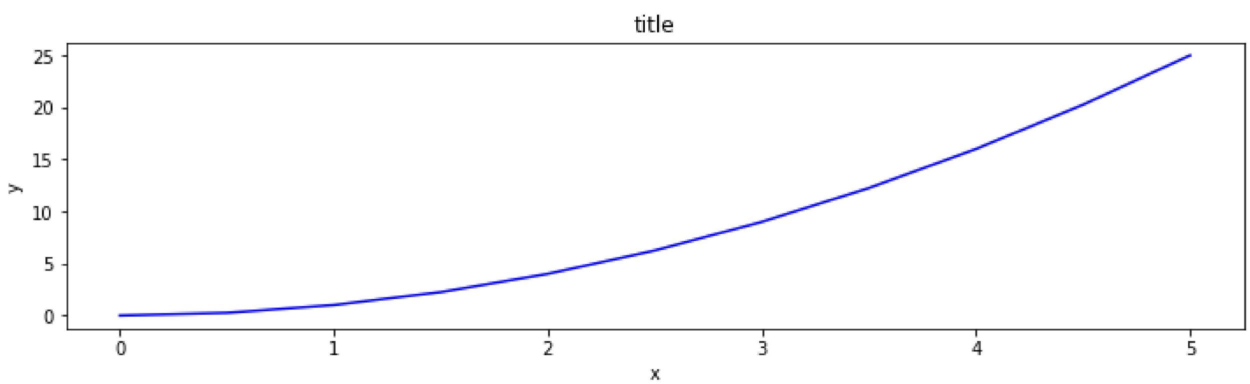



```
fig=plt.figure(figsize=(8,4),dpi=100)
```

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

```
fig,axes=plt.subplots(figsize=(12,3))
axes.plot(x,y,'b')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title')
```

```
Text(0.5, 1.0, 'title')
```



SAVING FIGURES

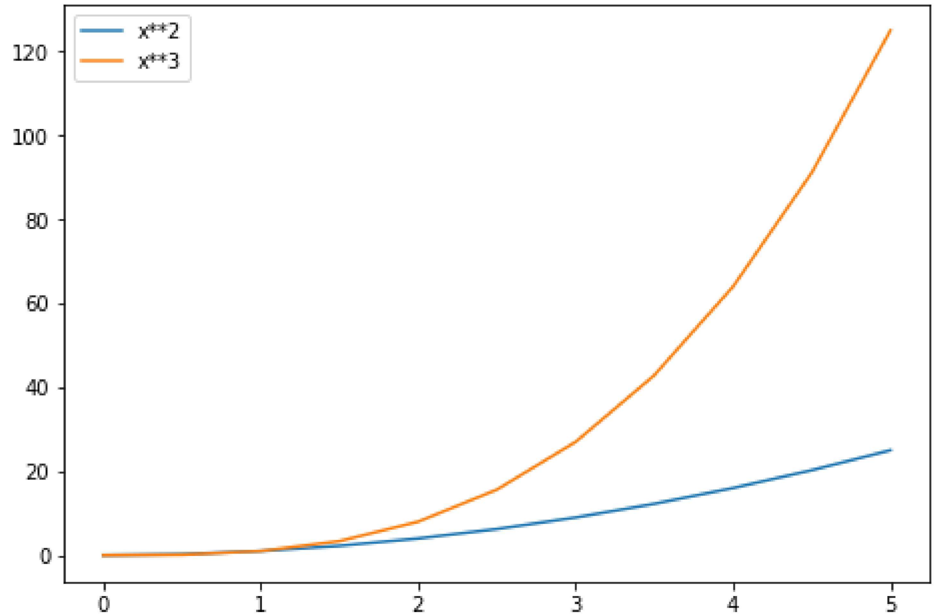
```
fig.savefig("filename.png")
fig.savefig("filename.png",dpi=200)
```

LEGENDS::

```
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()
```

```
<matplotlib.legend.Legend at 0x7f2430021a90>
```



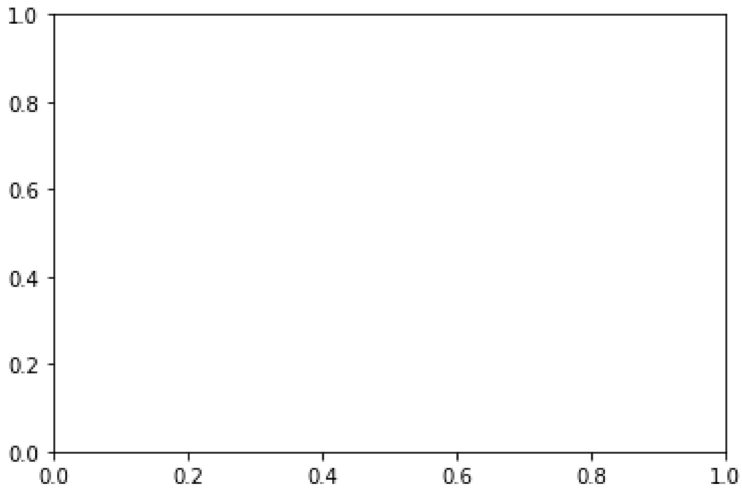

```
ax.legend(loc=1)  #upper right
ax.legend(loc=2)  #upper left
ax.legend(loc=3)  #lower left
ax.legend(loc=4)  #lower right
ax.legend(loc=0)  #let matplotlib decide the optimal location
```

```
#MATLAB style line color and style
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.plot(x,x**2,'b.-')  #blue line with dots
ax.plot(x,x**3,'g--')  #green dashed line
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-fb80fe789a59> in <module>()
      2 import matplotlib.pyplot as plt
      3 fig, ax = plt.subplots()
----> 4 ax.plot(x,x**2,'b.-')  #blue line with dots
      5 ax.plot(x,x**3,'g--')  #green dashed line

NameError: name 'x' is not defined
```

SEARCH STACK OVERFLOW



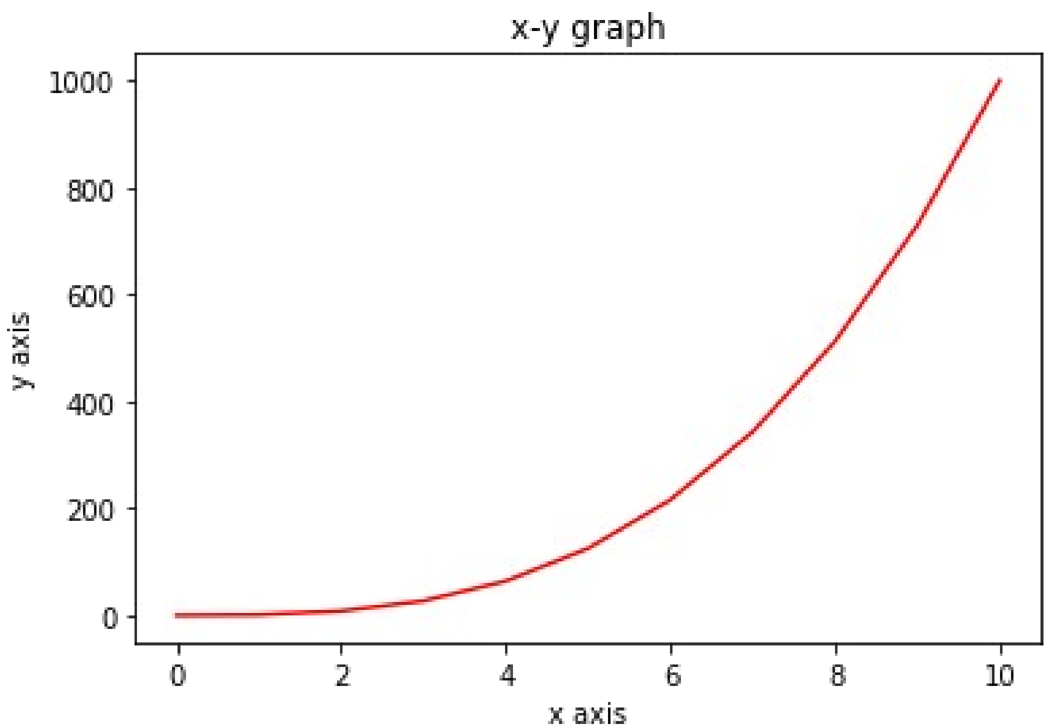
```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(0,10,11)
y=x ** 3
x
```

```
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

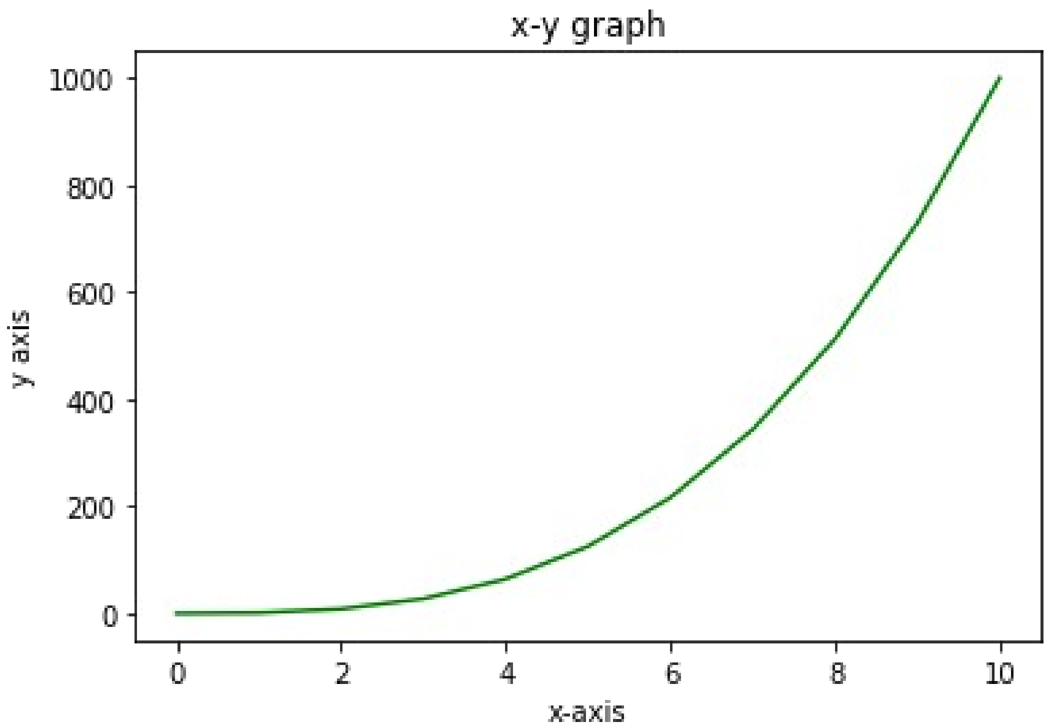
```
y
```

```
array([   0.,    1.,    8.,   27.,   64.,  125.,  216.,  343.,  512.,
        729., 1000.])
```

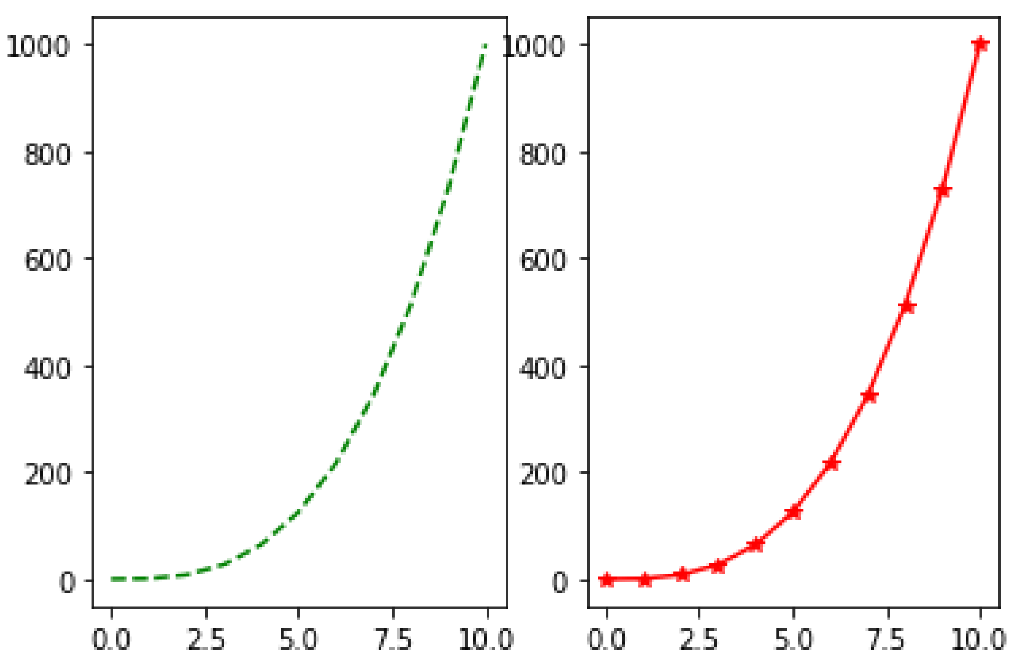
```
plt.plot(x,y,'red')
plt.xlabel("x axis")
plt.ylabel("y axis")
plt.title("x-y graph")
plt.show()
```



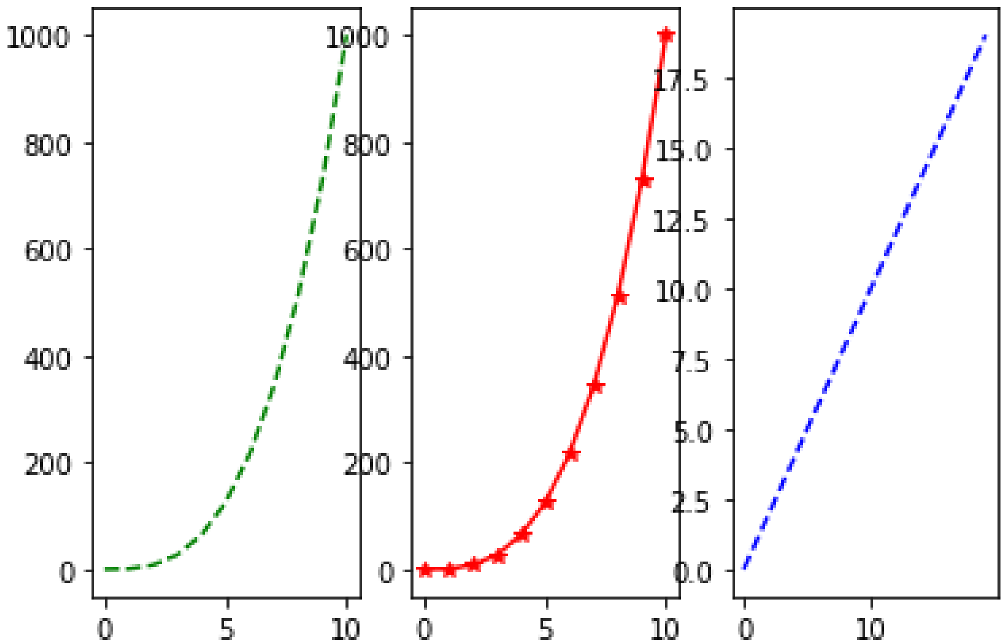

```
plt.plot(x,y,'g')
plt.xlabel("x-axis")
plt.ylabel("y axis")
plt.title("x-y graph")
plt.show()
```



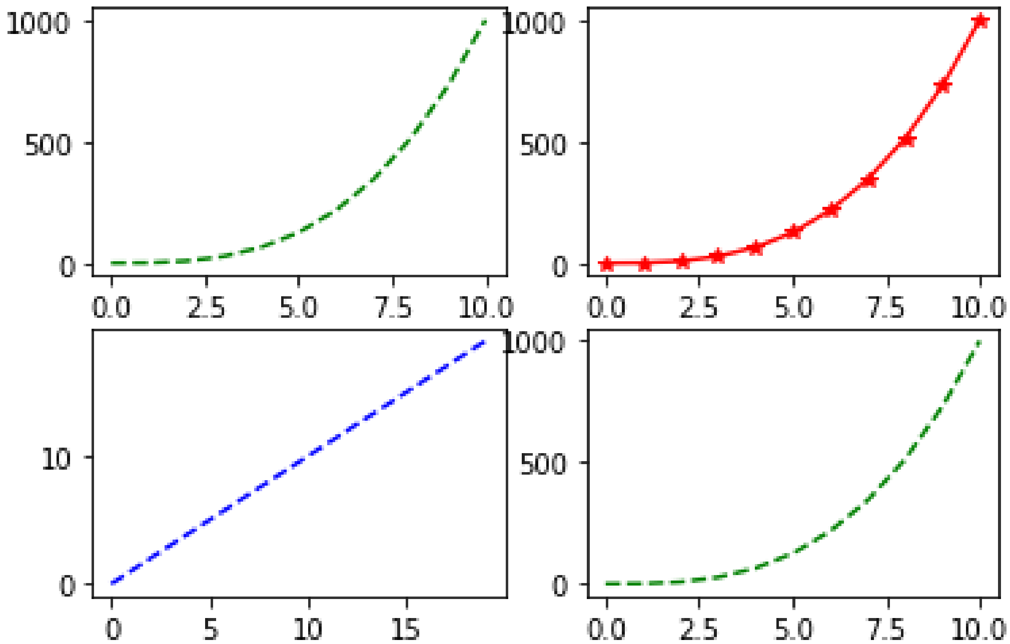

```
plt.subplot(1,2,1)
plt.plot(x,y,'g--')
plt.subplot(1,2,2)
plt.plot(x,y,'r*-')
plt.show()
```



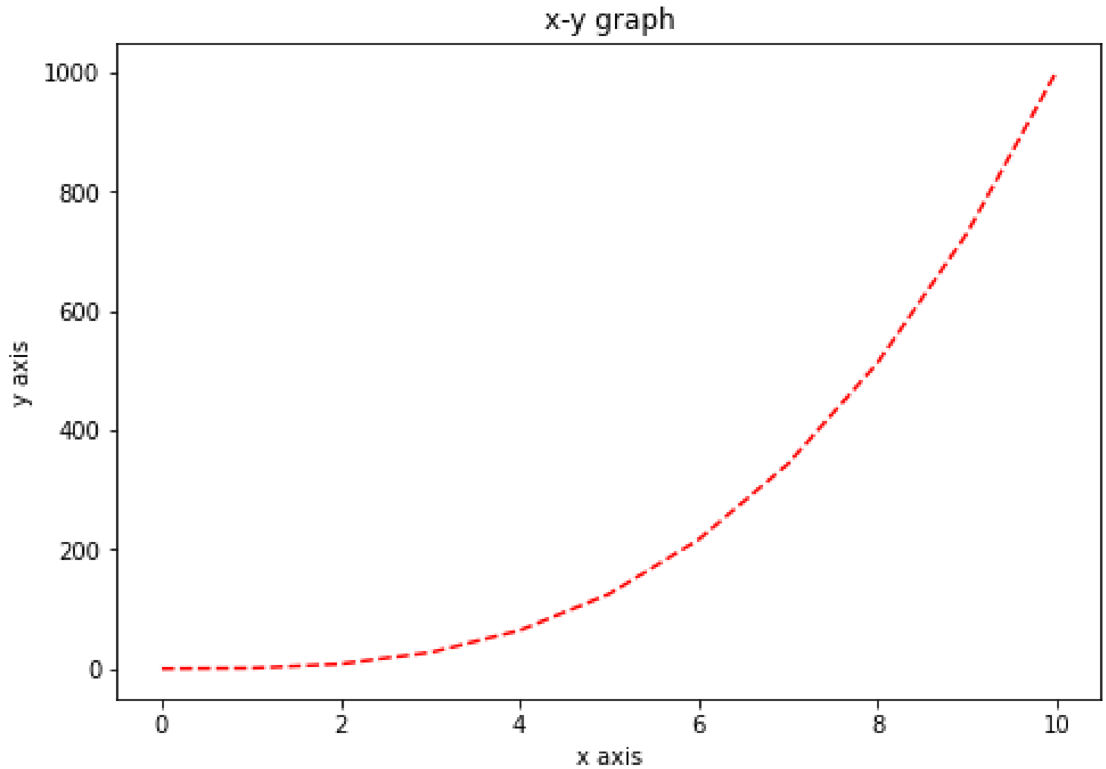
```
t=np.arange(0,20)
d=np.arange(0,20)
plt.subplot(1,3,1)
plt.plot(x,y,'g--')
plt.subplot(1,3,2)
plt.plot(x,y,'r*-')
plt.subplot(1,3,3)
plt.plot(t,d,'b--')
plt.show()
```



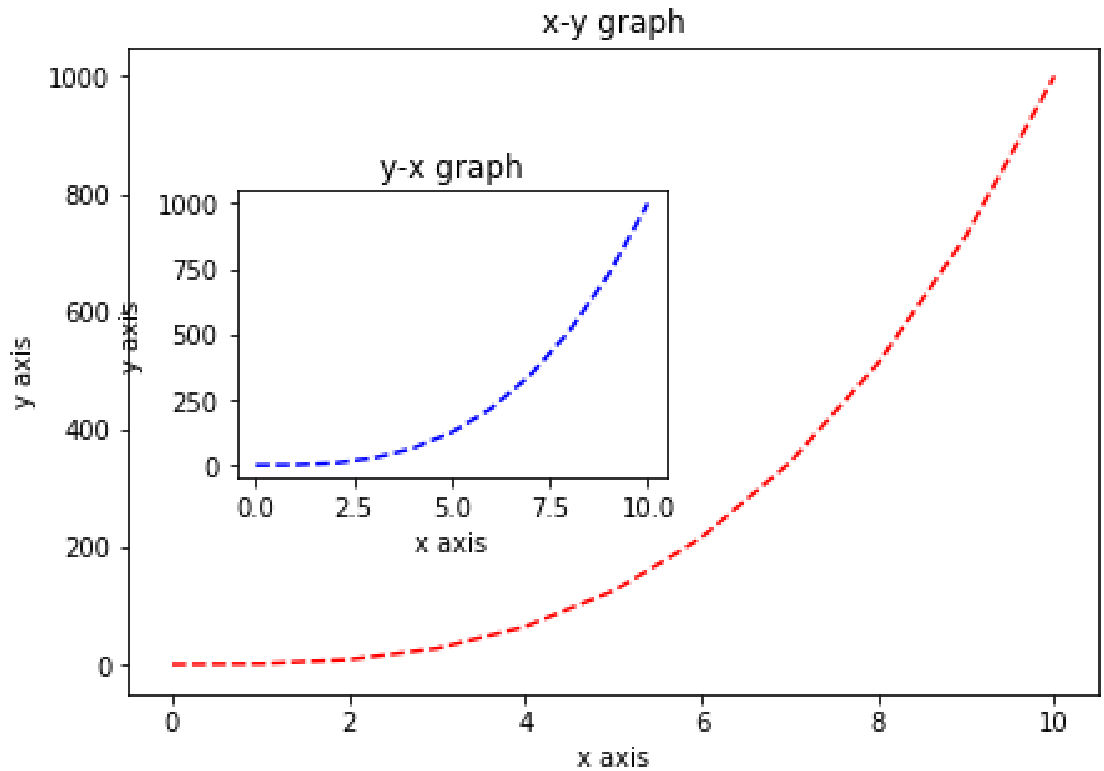

```
t=np.arange(0,20)
d=np.arange(0,20)
plt.subplot(2,2,1)
plt.plot(x,y,'g--')
plt.subplot(2,2,2)
plt.plot(x,y,'r*-')
plt.subplot(2,2,3)
plt.plot(t,d,'b--')
plt.subplot(2,2,4)
plt.plot(x,y,'g--')
plt.show()
```



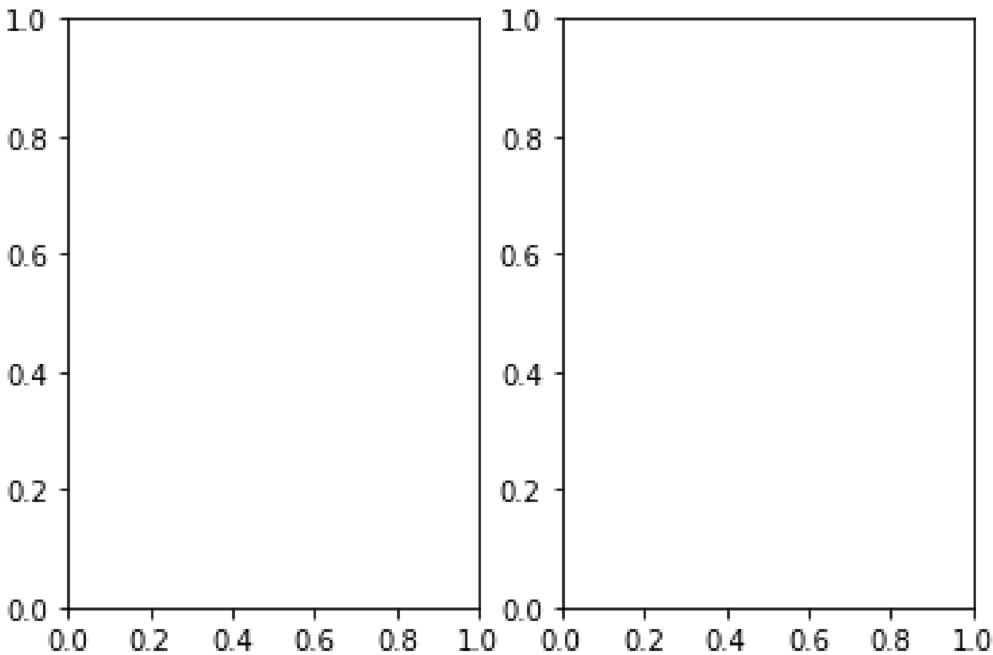

```
# using object oriented method
fig=plt.figure() #empty canvas
axes=fig.add_axes([2,2,1,1])  # add set of axes to figure
axes.plot(x,y,'r--')
axes.set_xlabel('x axis')
axes.set_ylabel('y axis')
axes.set_title("x-y graph")
plt.show()
```




```
fig=plt.figure() #empty canvas
axes1=fig.add_axes([0.3,0.3,0.9,0.9])
axes2=fig.add_axes([0.4,0.6,0.4,0.4])  # add set of axes to figure
#larger one
axes1.plot(x,y,'r--')
axes1.set_xlabel('x axis')
axes1.set_ylabel('y axis')
axes1.set_title("x-y graph")
#smaller one
axes2.plot(x,y,'b--')
axes2.set_xlabel('x axis')
axes2.set_ylabel('y axis')
axes2.set_title("y-x graph")
plt.show()
```

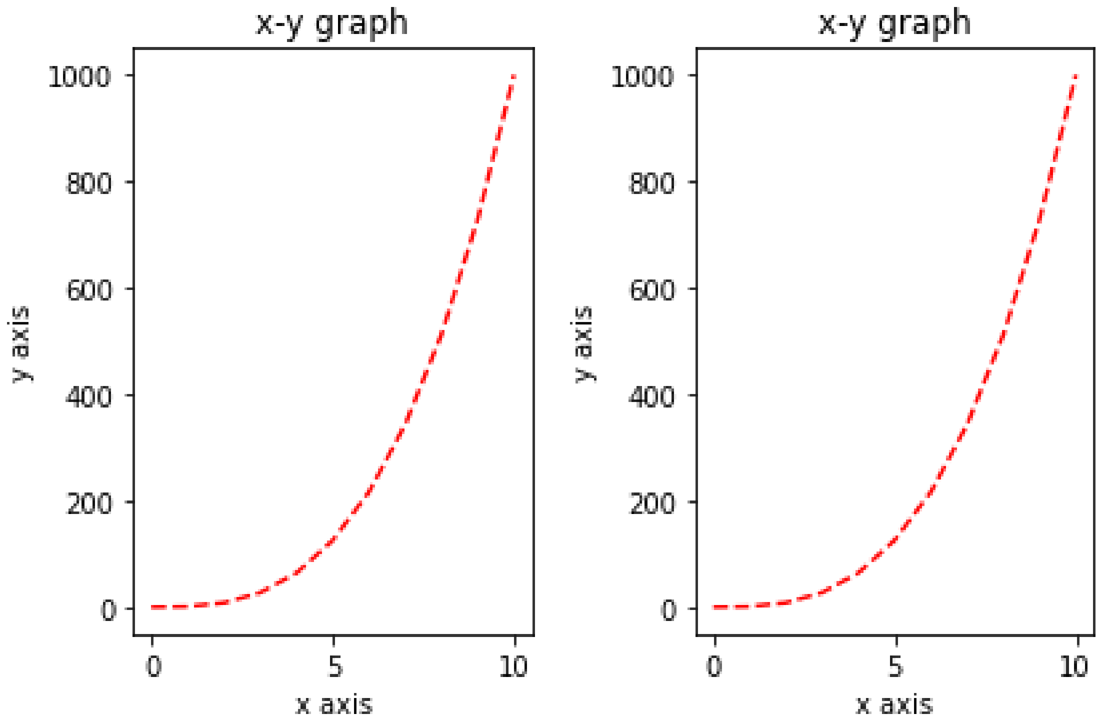



```
fig,axes=plt.subplots(nrows=1,ncols=2)
```

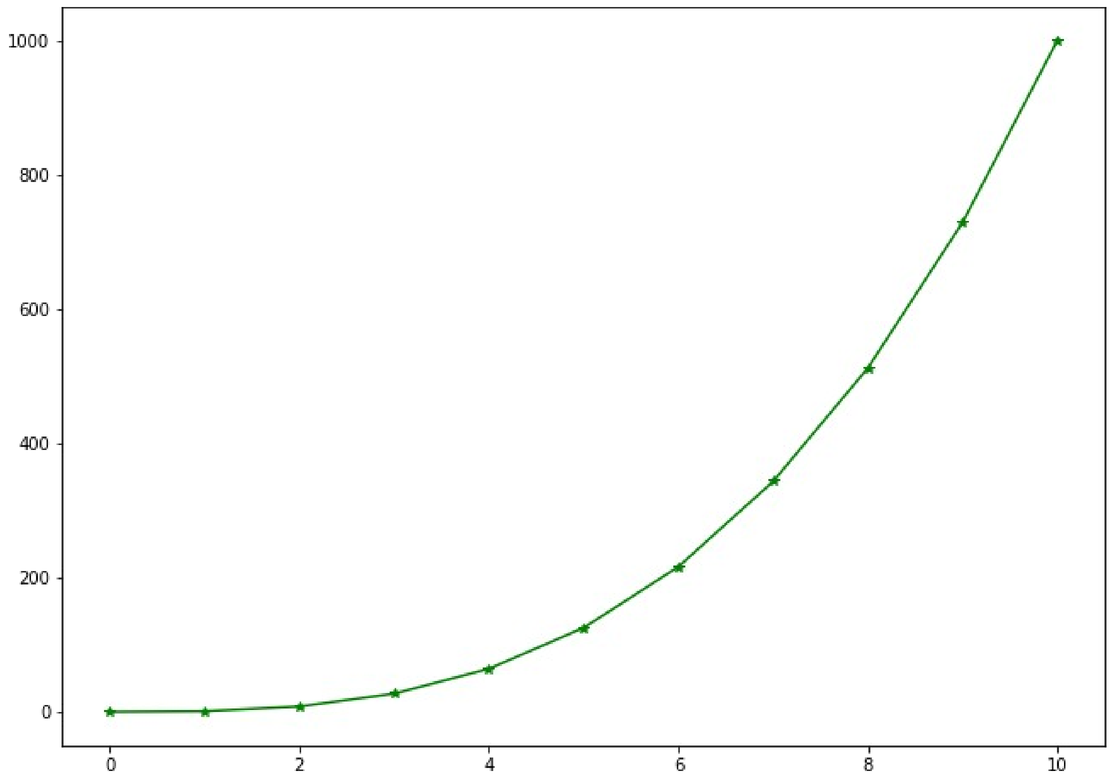



```
# use similar to plt.figure() except use tuple unpacking to grab fig
and axes
```

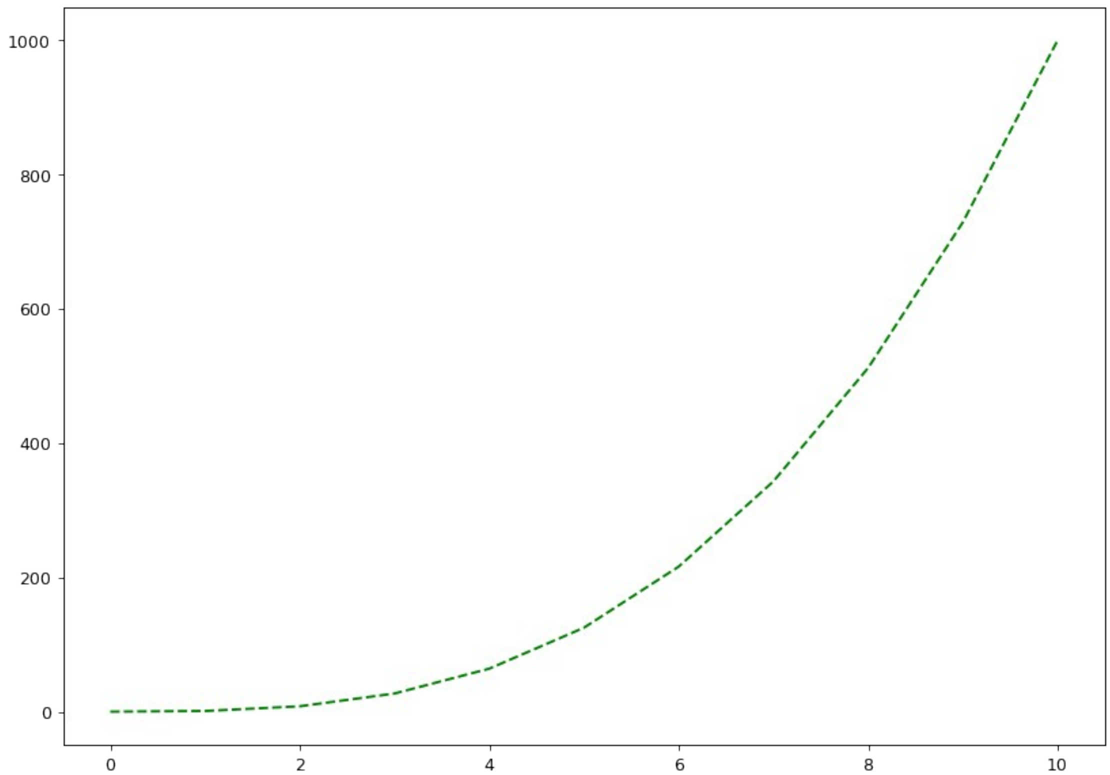
```
fig,axes=plt.subplots(nrows=1,ncols=2)
# iterate through this array
for ax in axes:
  ax.plot(x,y,'r--')  # use axes object to add stuff to plot
  ax.set_xlabel('x axis')
  ax.set_ylabel('y axis')
  ax.set_title("x-y graph")
fig.tight_layout()
```




```
fig,axes=plt.subplots(figsize=(11,8))#width and height
axes.plot(x,y,'g*-')
fig.show()
```

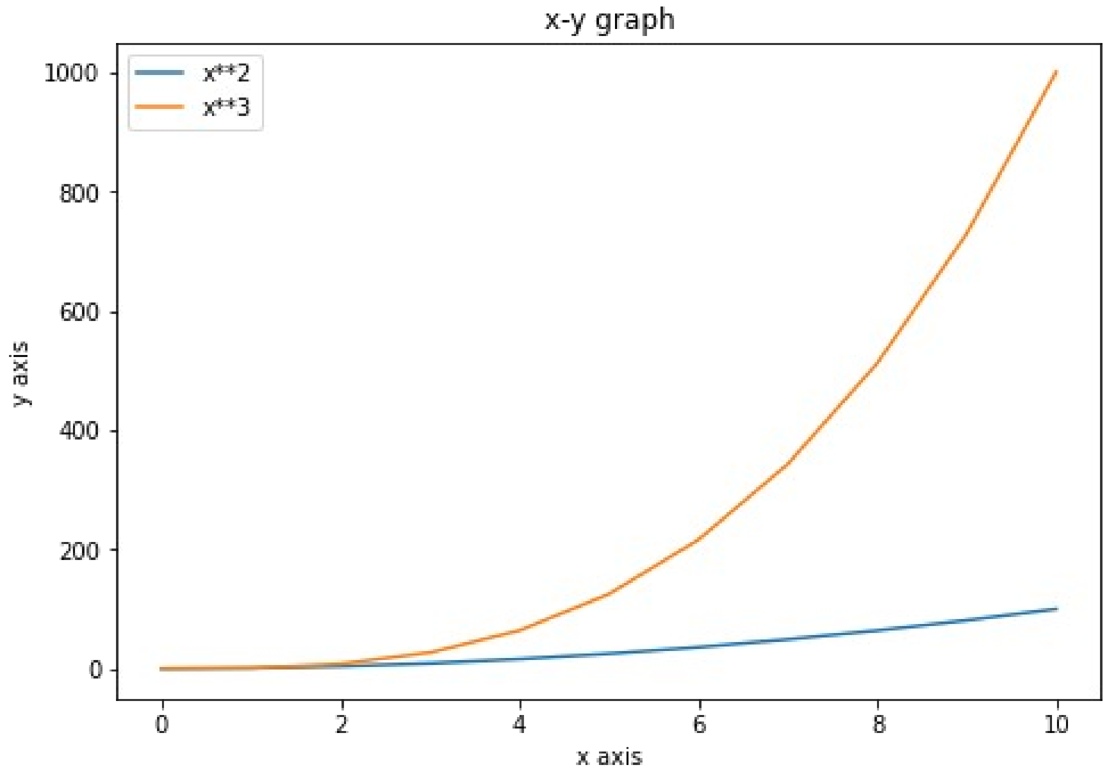



```
fig,axes=plt.subplots(figsize=(11,8),dpi=90)#width and height
axes.plot(x,y,'g--')
fig.show()
```

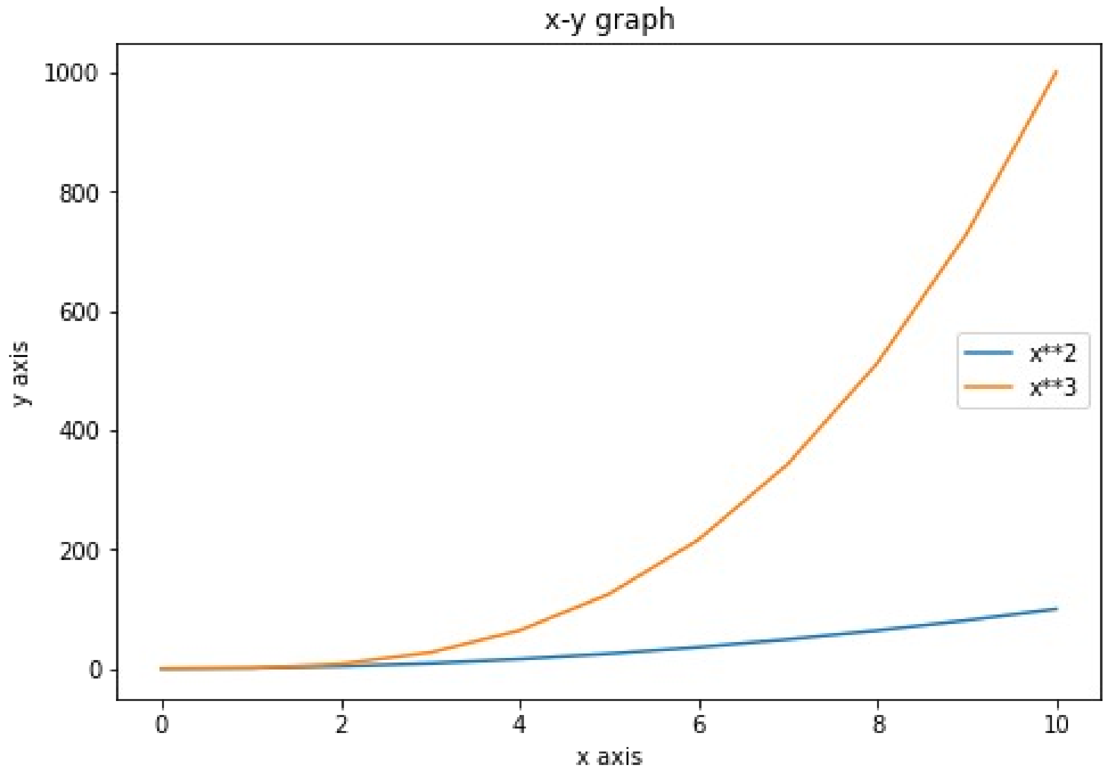



```
fig.savefig("filename1.png") # save figures

fig=plt.figure() #empty canvas
axes=fig.add_axes([2,2,1,1])  # add set of axes to figure
axes.plot(x,x**2,label="x**2")
axes.plot(x,x**3,label="x**3")
axes.set_xlabel('x axis')
axes.set_ylabel('y axis')
axes.set_title("x-y graph")
axes.legend() #legend function
plt.show()
```




```
fig=plt.figure() #empty canvas
axes=fig.add_axes([2,2,1,1])  # add set of axes to figure
axes.plot(x,x**2,label="x**2")
axes.plot(x,x**3,label="x**3")
axes.set_xlabel('x axis')
axes.set_ylabel('y axis')
axes.set_title("x-y graph")
axes.legend(loc=7) #legend function with loc
plt.show()
```

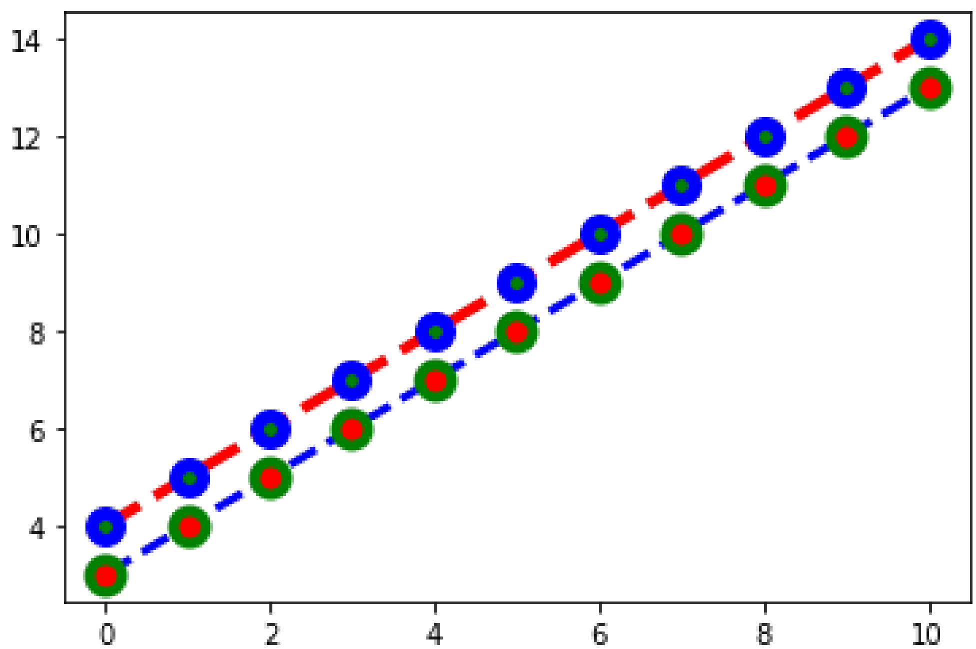



```
import matplotlib.pyplot as plt
import numpy as np

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.
```


14
12
10
8
6
4
0
2
4
6
8
10

SEABORN:

```
import seaborn as sns
```

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

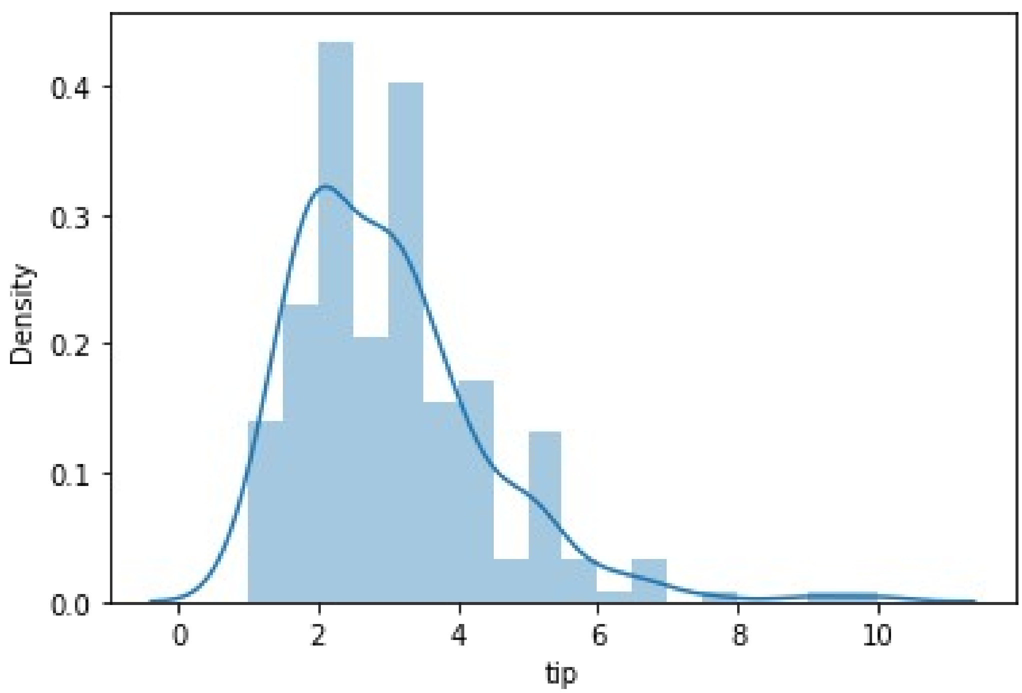
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

DISTPLOT: It shows the distribution of a univariate set of observations.

```
sns.distplot(tips['tip'])
```

```
/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 0x7f08a096b810>
```

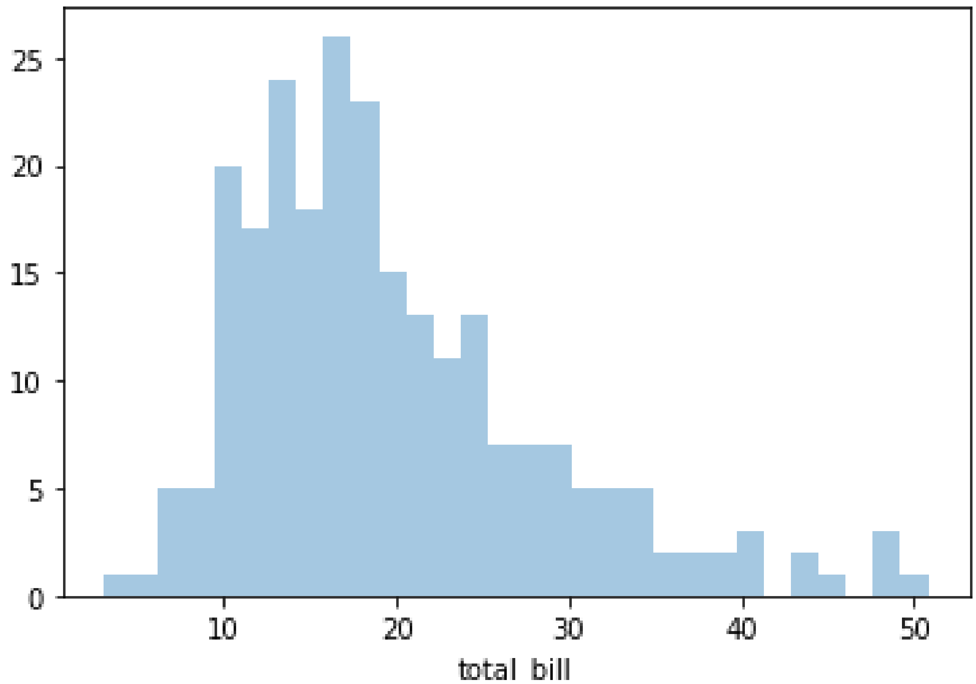


To remove the kde layer and just have the histogram use:

```
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)
```

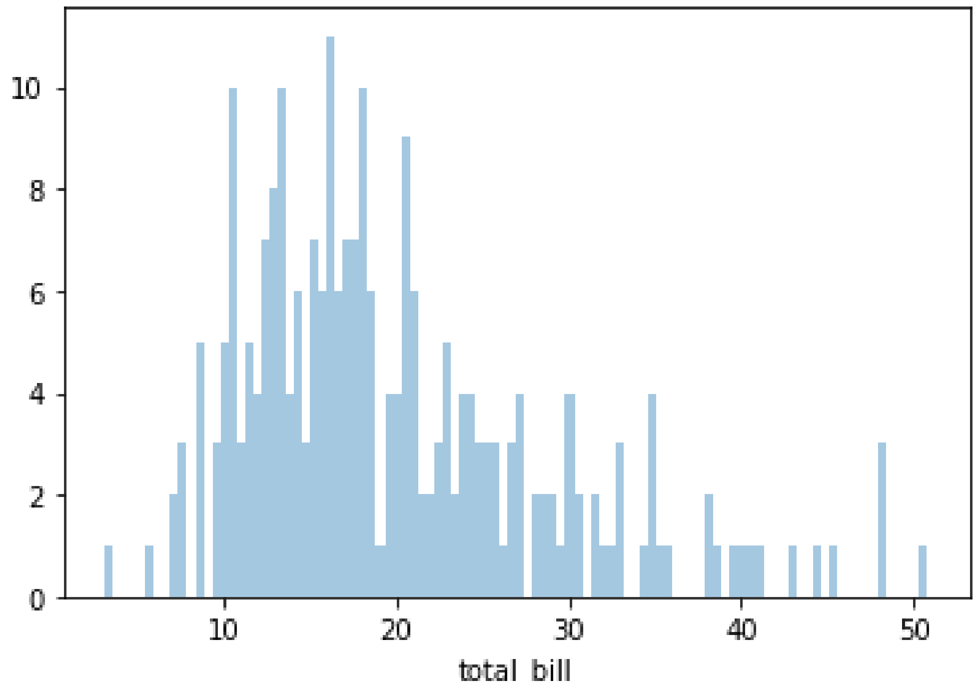
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f089d57b650>
```




```
sns.distplot(tips['total_bill'],kde=False,bins=100)
```

```
/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 0x7f089d04f410>
```

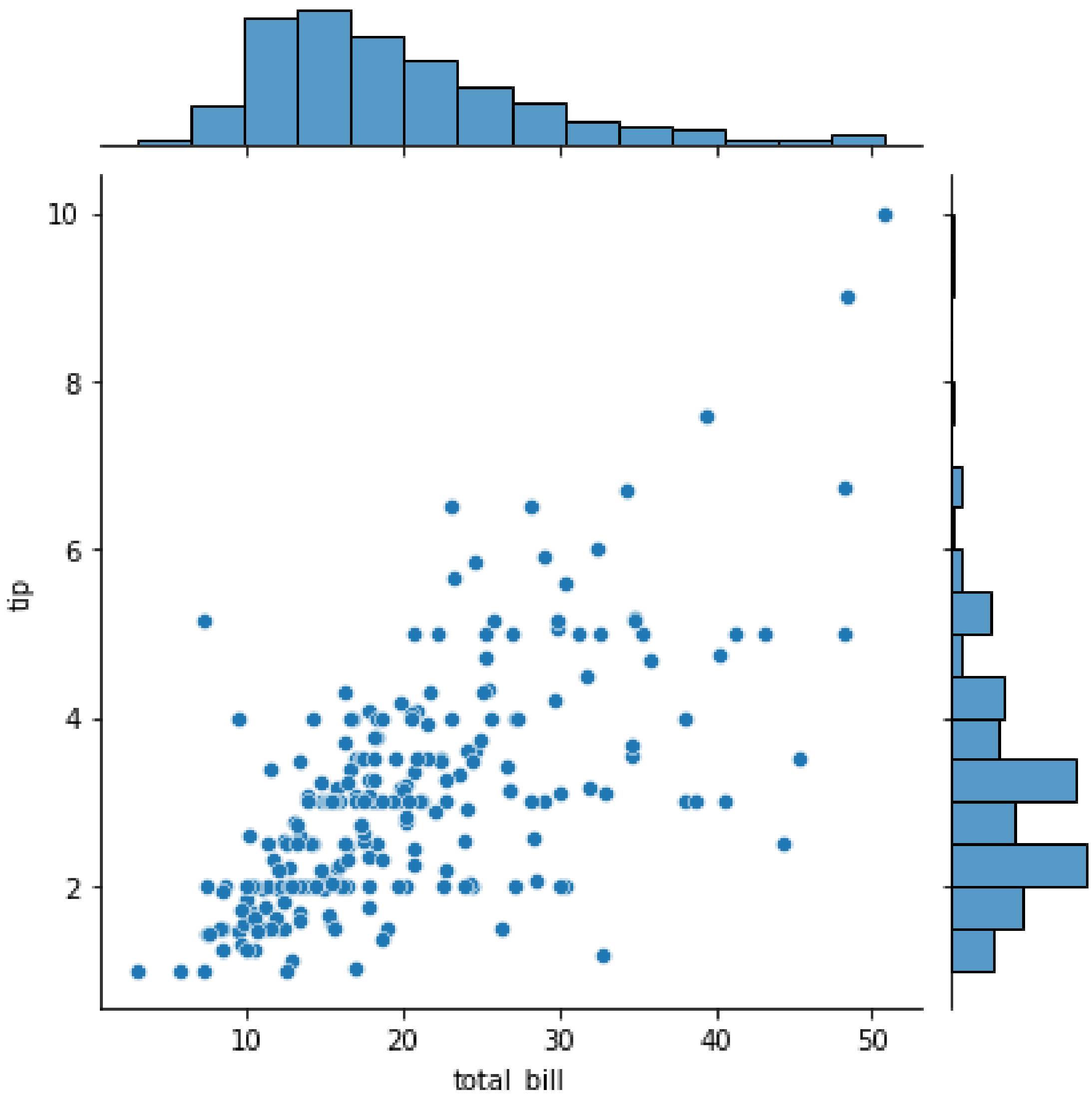


JOINTPLOT: It allows you to basically match up two displots for bivariate data.With your choice of what "kind" parameter to compare with:

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

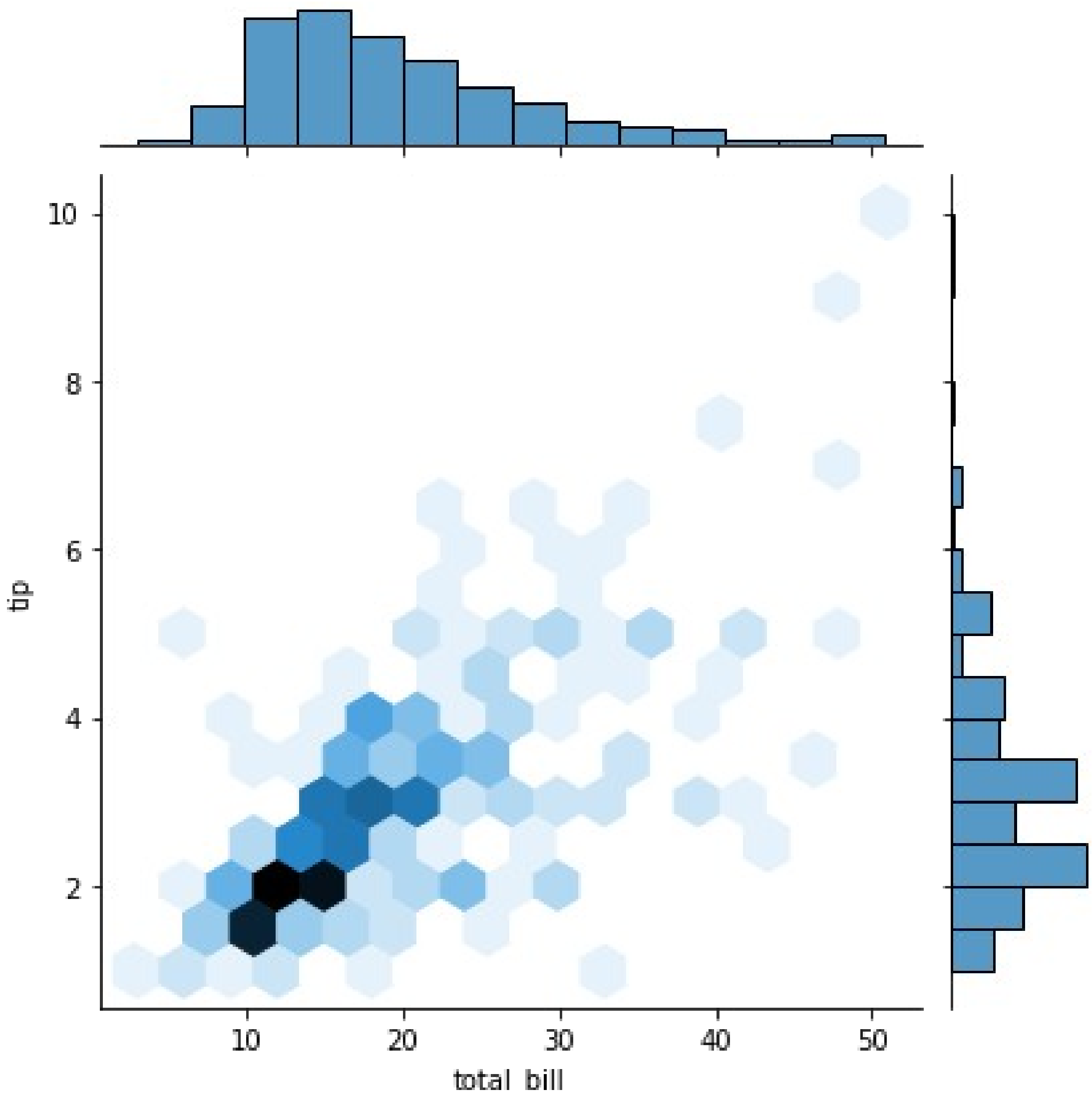
```
sns.jointplot(x='total_bill',y='tip',data=tips,kind='scatter')
```

```
<seaborn.axisgrid.JointGrid at 0x7f089d10fe10>
```



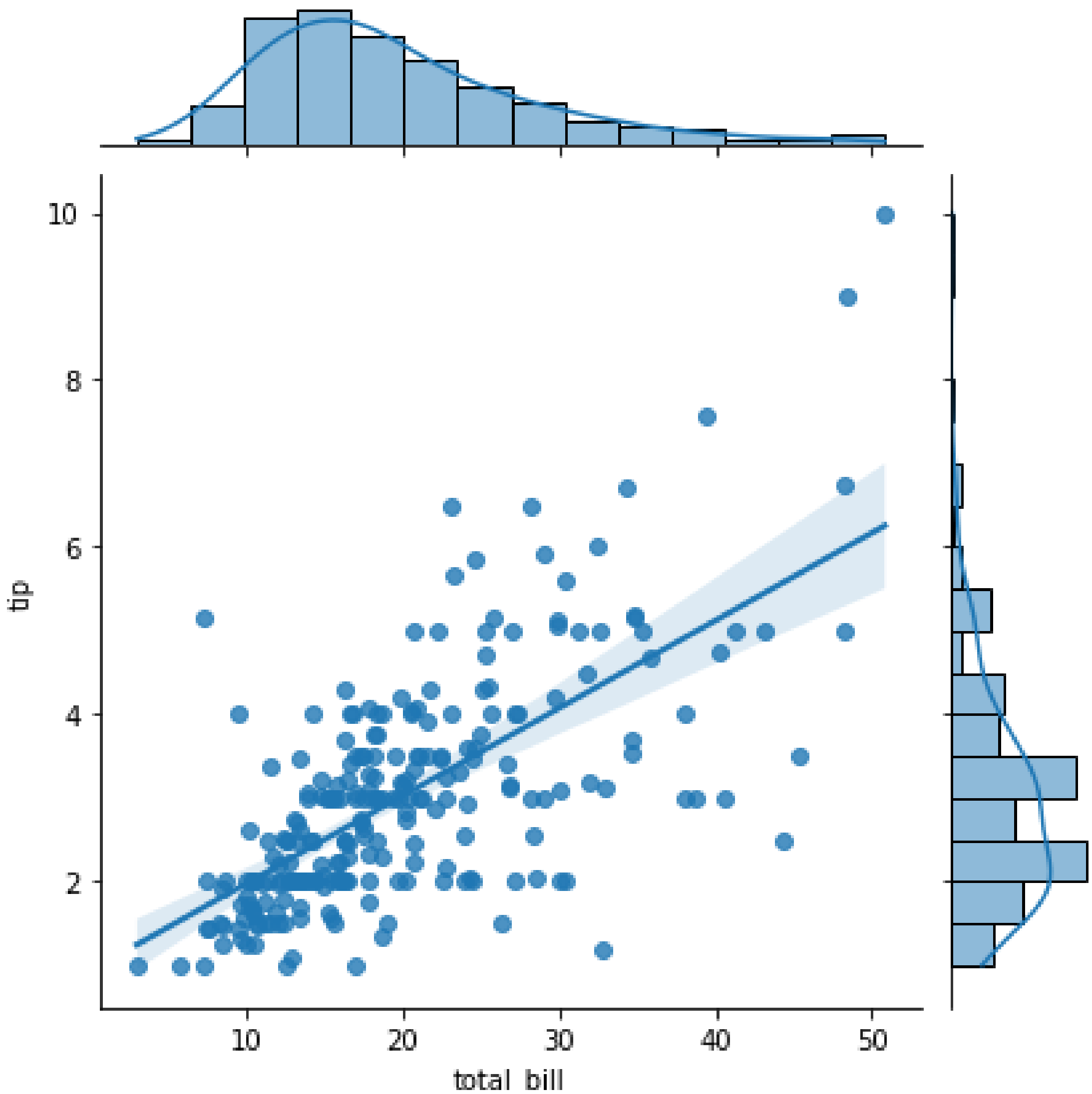

```
sns.jointplot(x='total_bill',y='tip',data=tips,kind='hex')
```

```
<seaborn.axisgrid.JointGrid at 0x7f0894541a50>
```



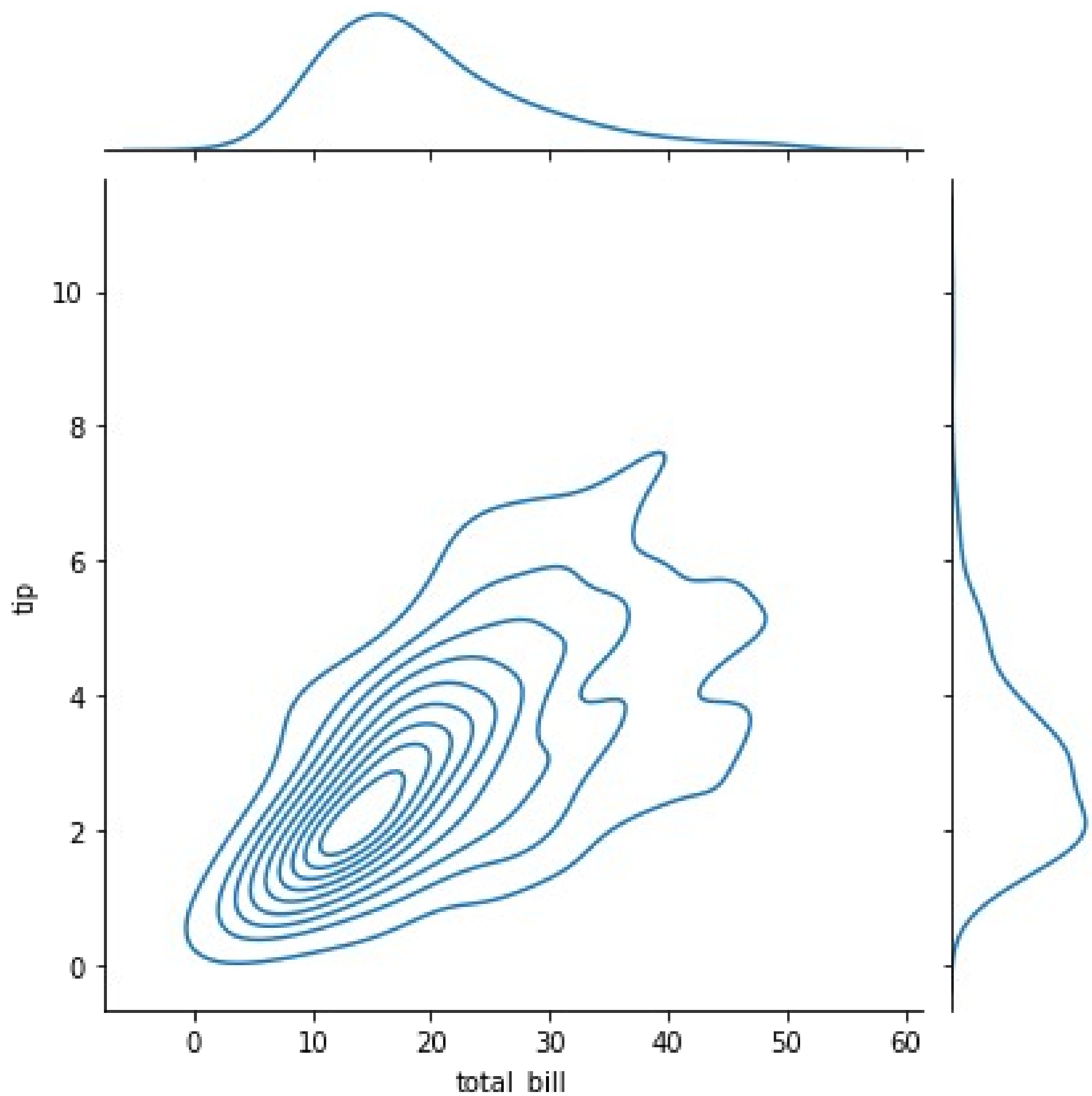

```
sns.jointplot(x='total_bill',y='tip',data=tips,kind='reg')
```

```
<seaborn.axisgrid.JointGrid at 0x7f08944ef550>
```



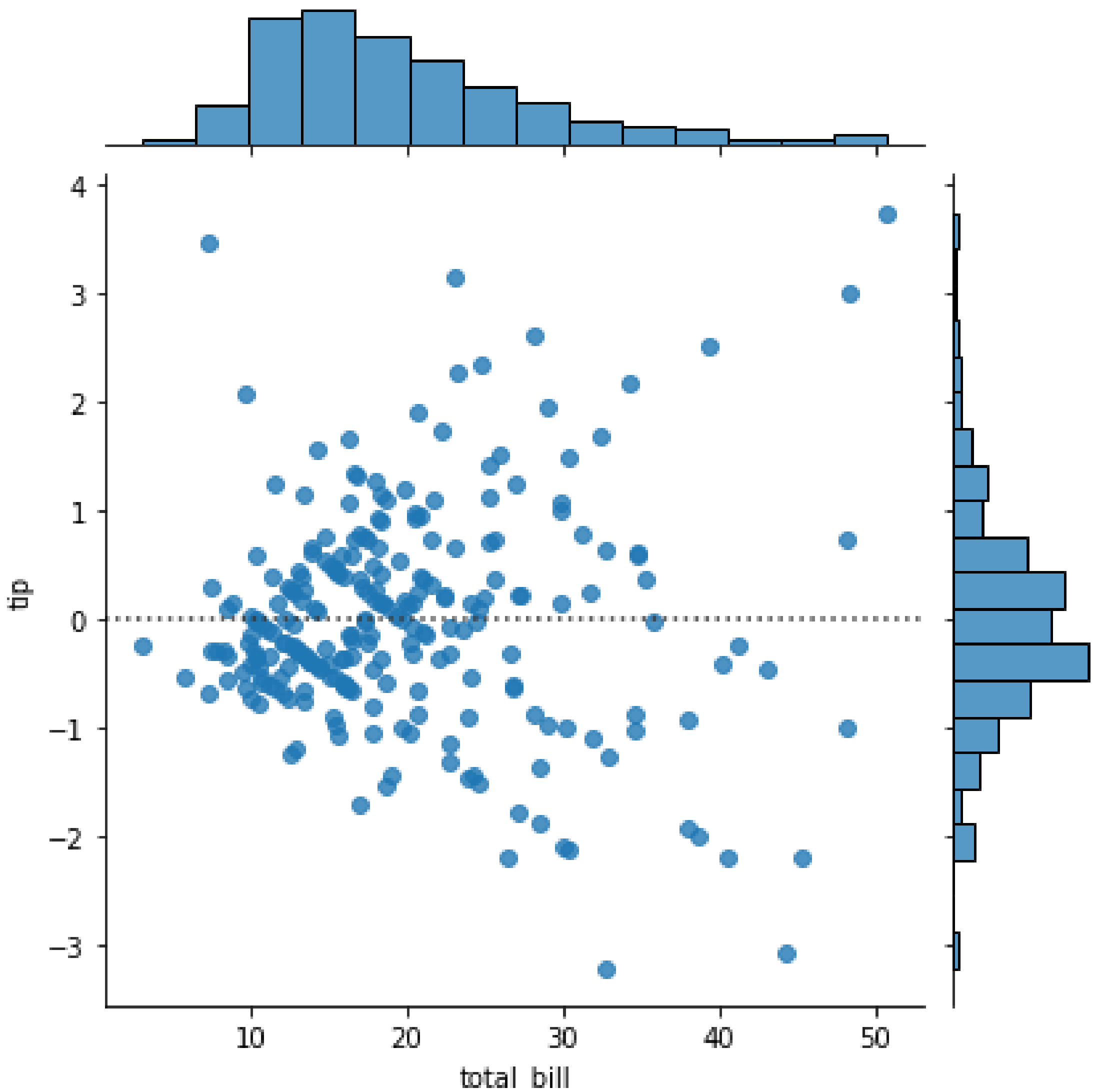

```
sns.jointplot(x='total_bill',y='tip',data=tips,kind='kde')
```

```
<seaborn.axisgrid.JointGrid at 0x7f089cdb6f50>
```




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

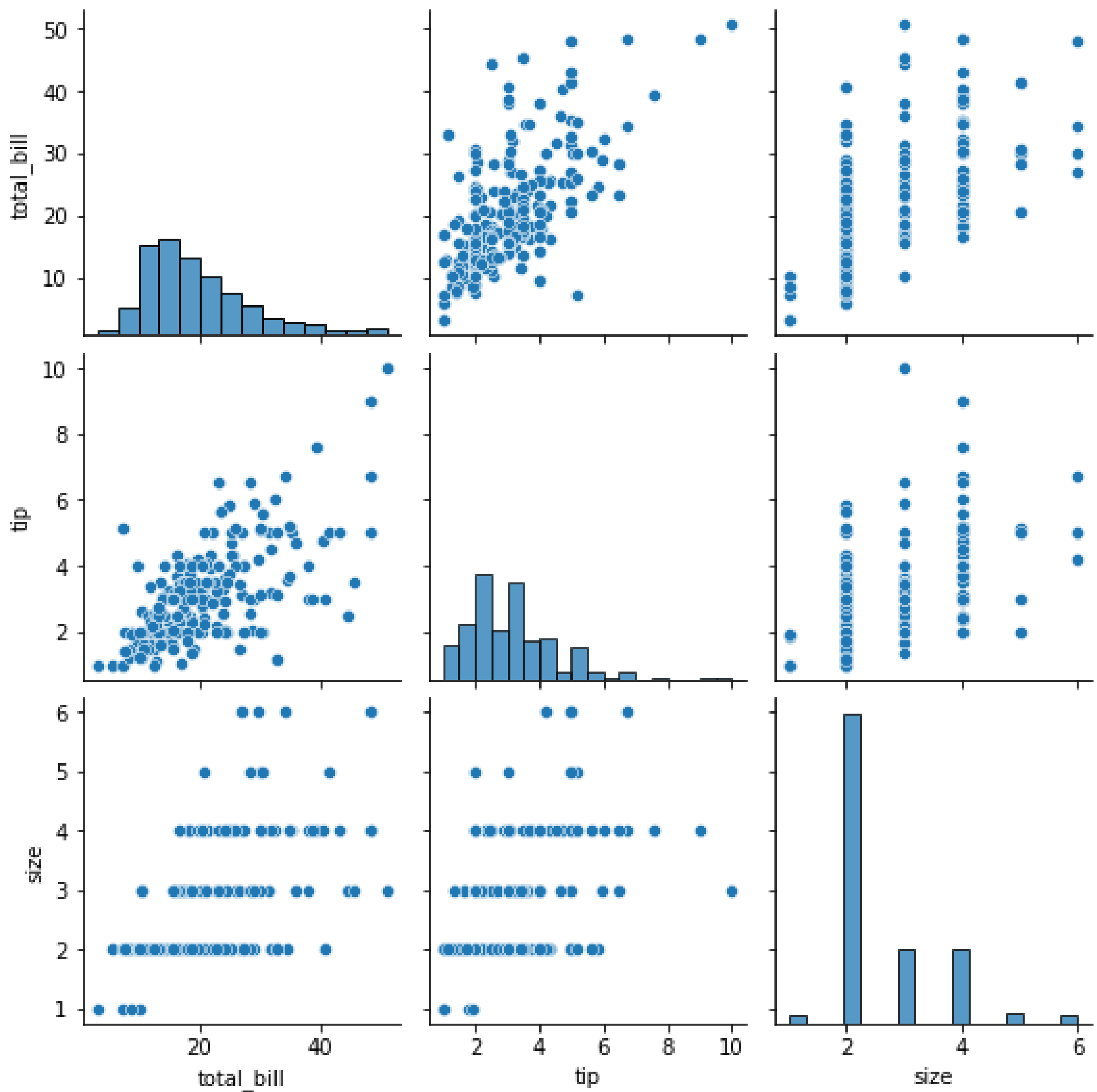
```
<seaborn.axisgrid.JointGrid at 0x7f08941966d0>
```



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

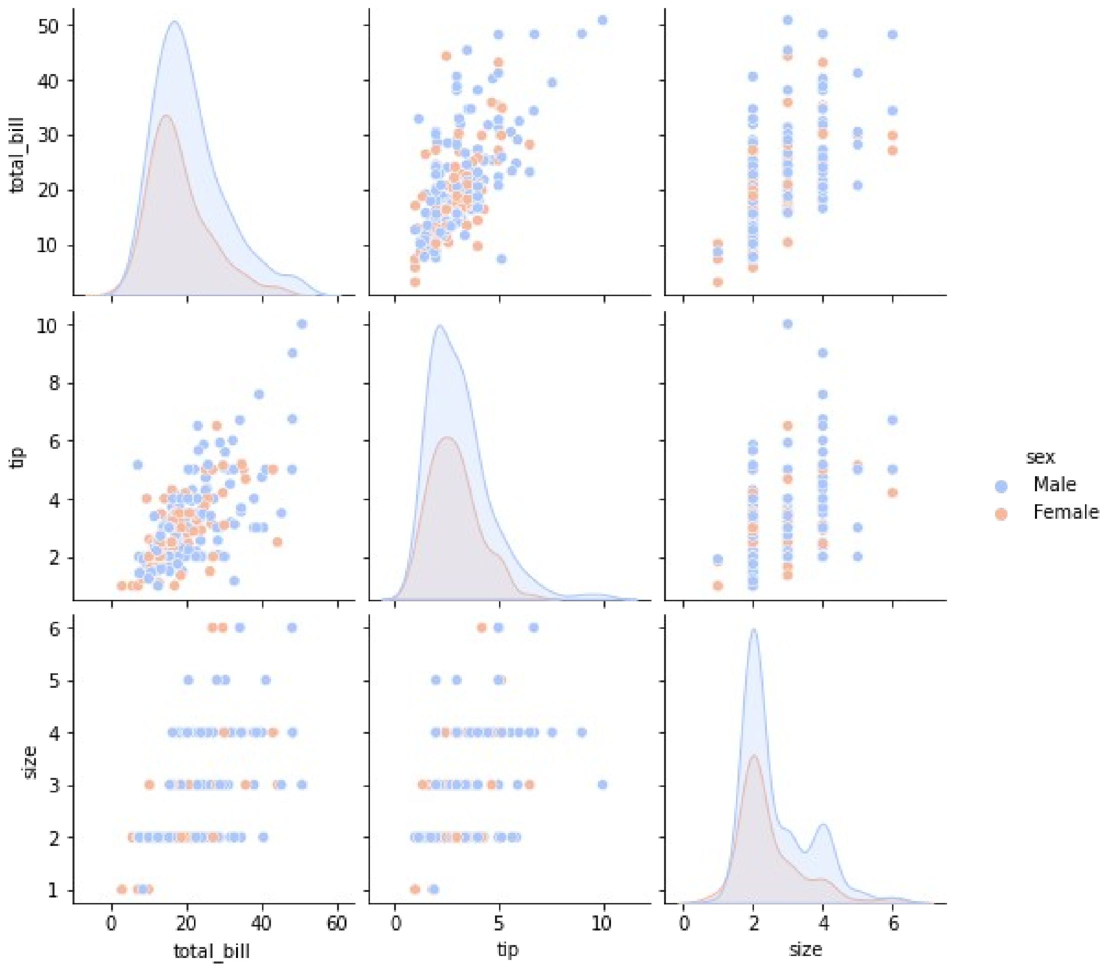
```
sns.pairplot(tips)
```

```
<seaborn.axisgrid.PairGrid at 0x7f0893deb610>
```




```
sns.pairplot(tips,hue='sex',palette='coolwarm')
```

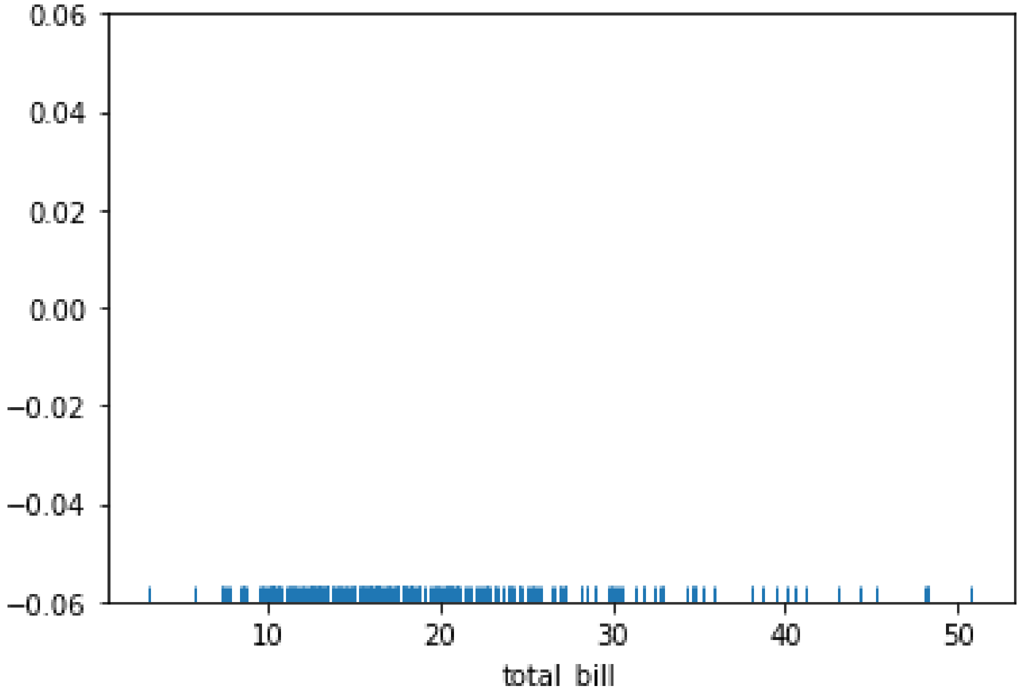
```
<seaborn.axisgrid.PairGrid at 0x7f08938ca450>
```



RUGPLOT: These are actually a very simple concept, they just draw a dash mark for every point on a univariate distribution. They are the building block of a KDE plot.

```
sns.rugplot(tips['total_bill'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0891bce7d0>
```



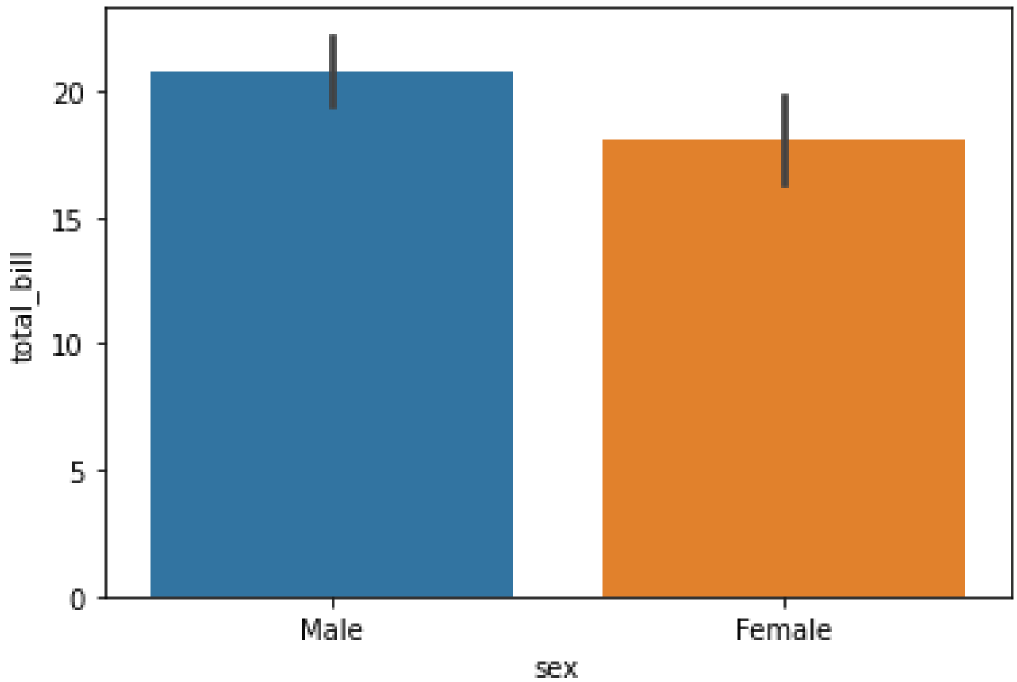
CATEGORICAL DATA PLOTS:

1. factorplot
2. boxplot
3. violinplot
4. stripplot
5. swarmplot
6. barplot
7. countplot

BARPLOT AND COUNTPLOT: These are similar plots allow you to get aggregate data off a categorical feature in your data. barplot is a general plot that allows you to aggregate the categorical data based off some function, by default the mean.

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f08934592d0>
```

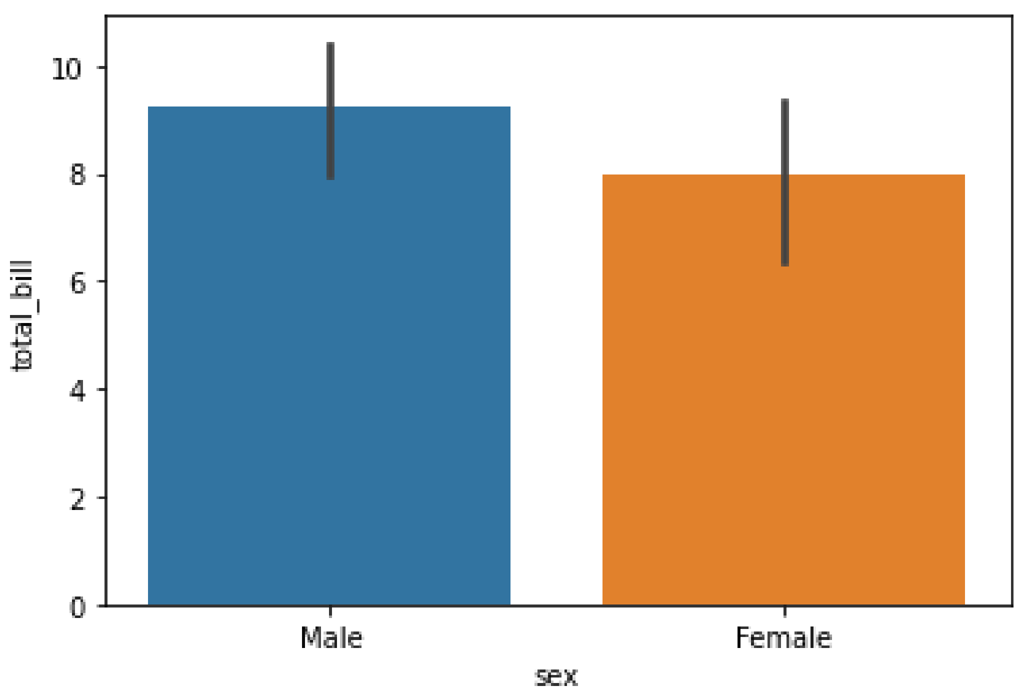



```
import numpy as np
```

You can change the estimator object to your own function, that converts a vector to a scalar.

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

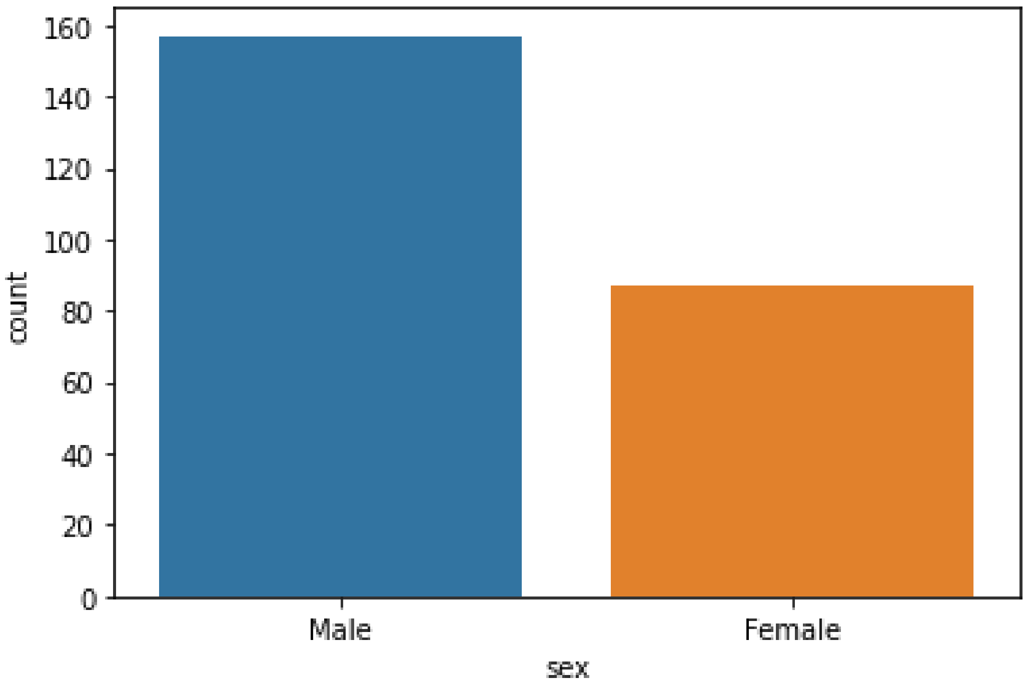
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0891abcb90>
```



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)
```

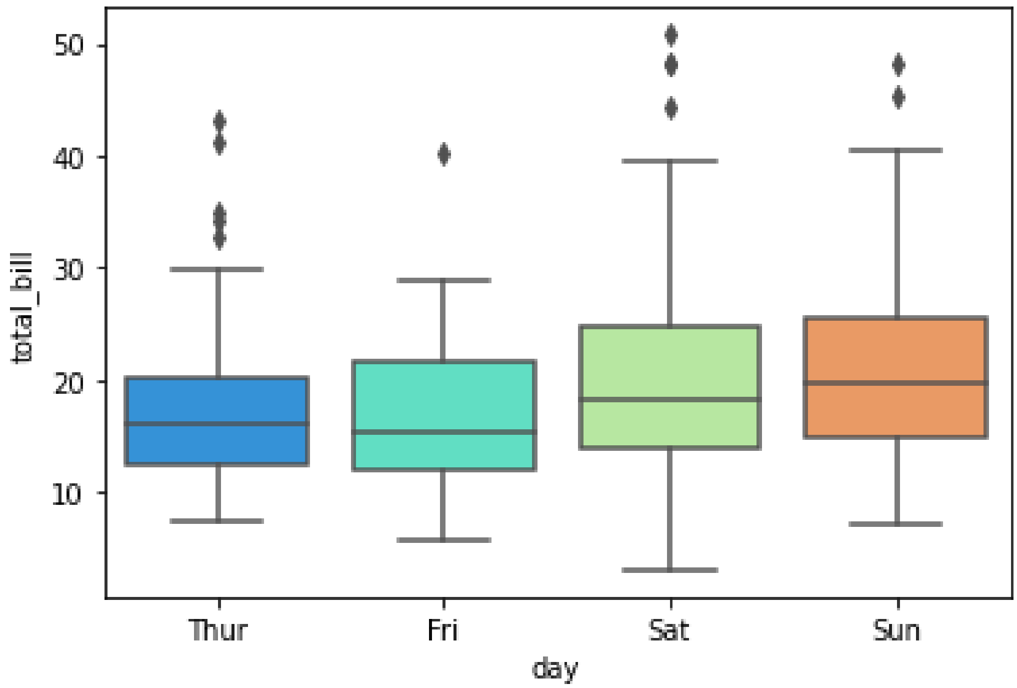
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f089193fad0>
```



BOXPLOT: These are used to shown the distribution of categorical data. A box plot(or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. The box shows the quartiles of a dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be "outliers" using a method that is a function of the inter_quartile range.

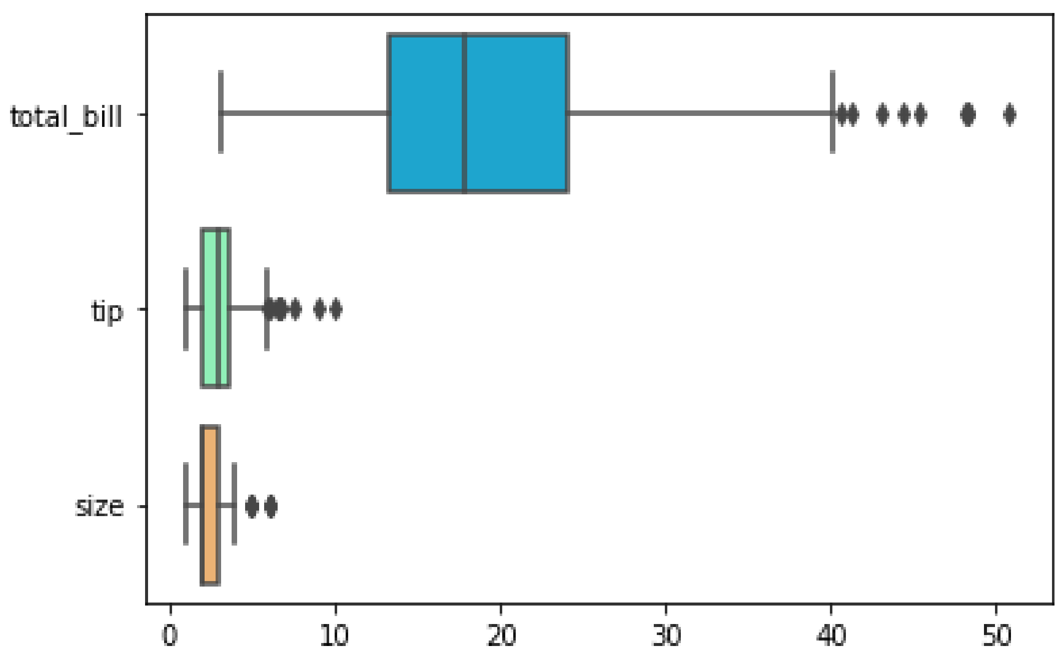
```
sns.boxplot(x="day",y="total_bill", data=tips,palette='rainbow')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0891b22b10>
```



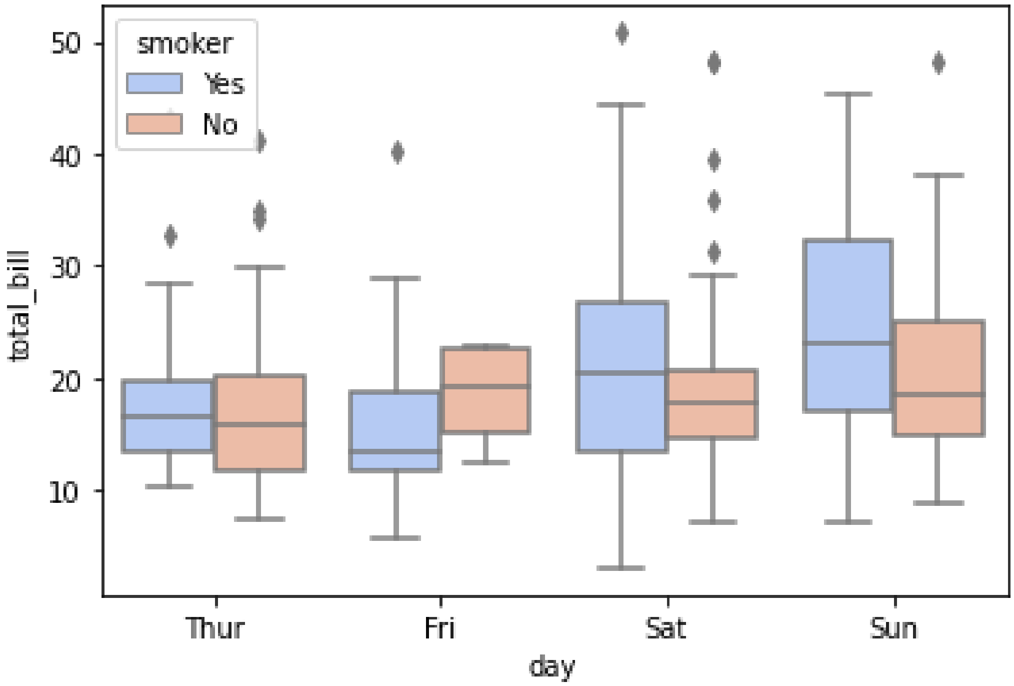

```
#can do entire dataframe with orient ='h'
sns.boxplot(data=tips,palette='rainbow',orient='h')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f08917c14d0>
```




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

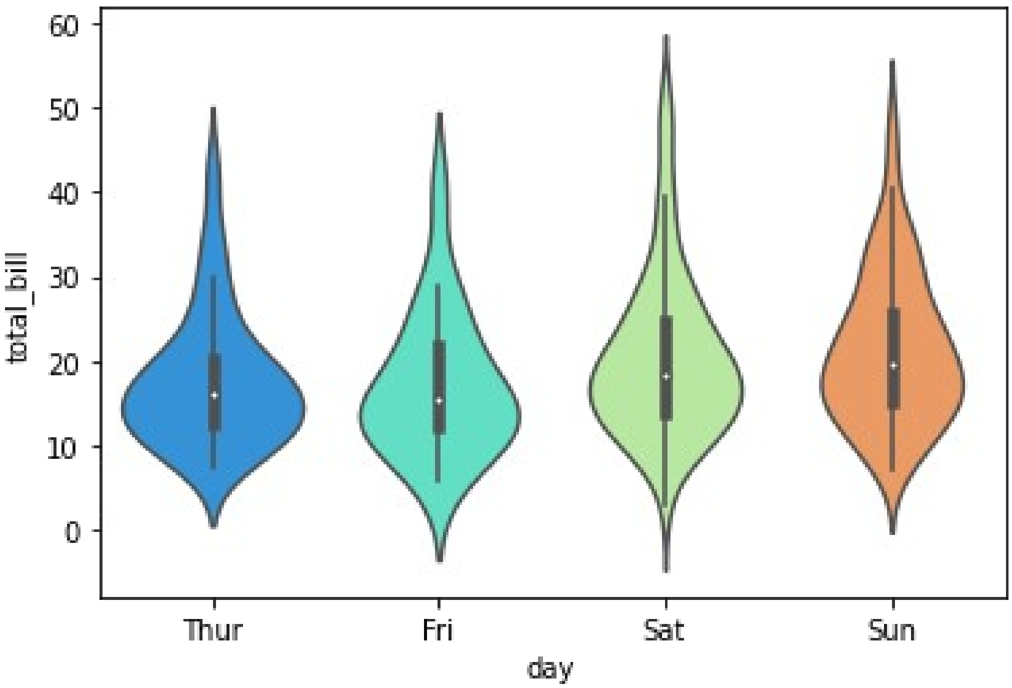
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f08916ceed0>
```



VIOLINPLOT: It plays a similar role as a box and whisker plot.It shows the distribution of quantitative data across several levels of one (or more) categorical variables such that those distributions can be compared. Unlike a box plot, in which all of the plot components corresponds to actual datapoints, the violin plot features a kernel density estimation(kde) of the underlying distribution.

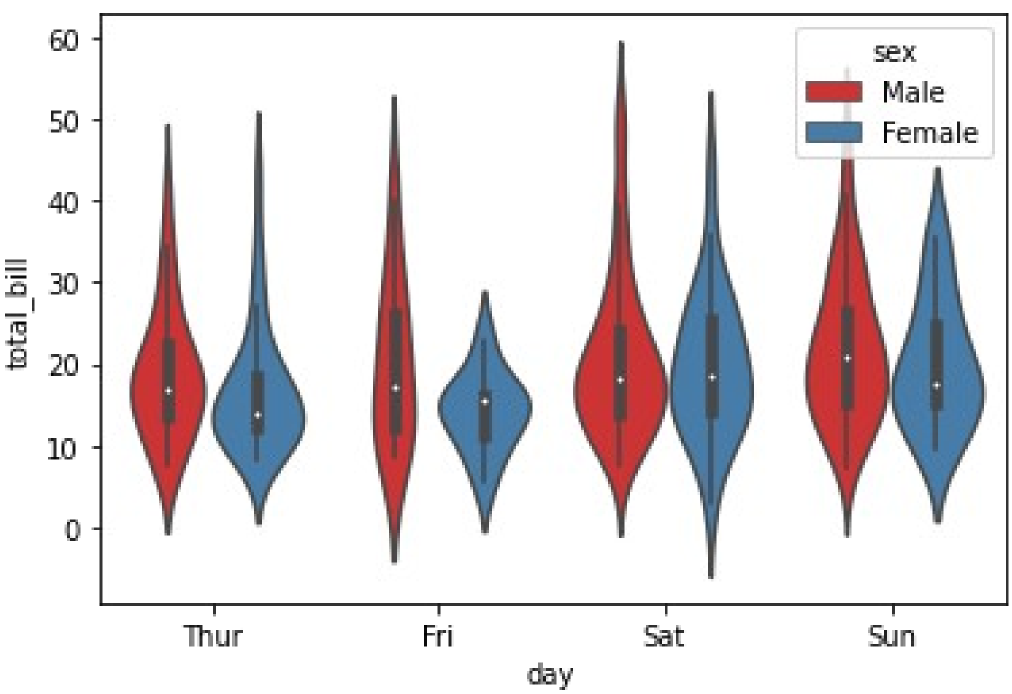
```
sns.violinplot(x="day",y="total_bill", data=tips,palette='rainbow')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f08918c1f10>
```



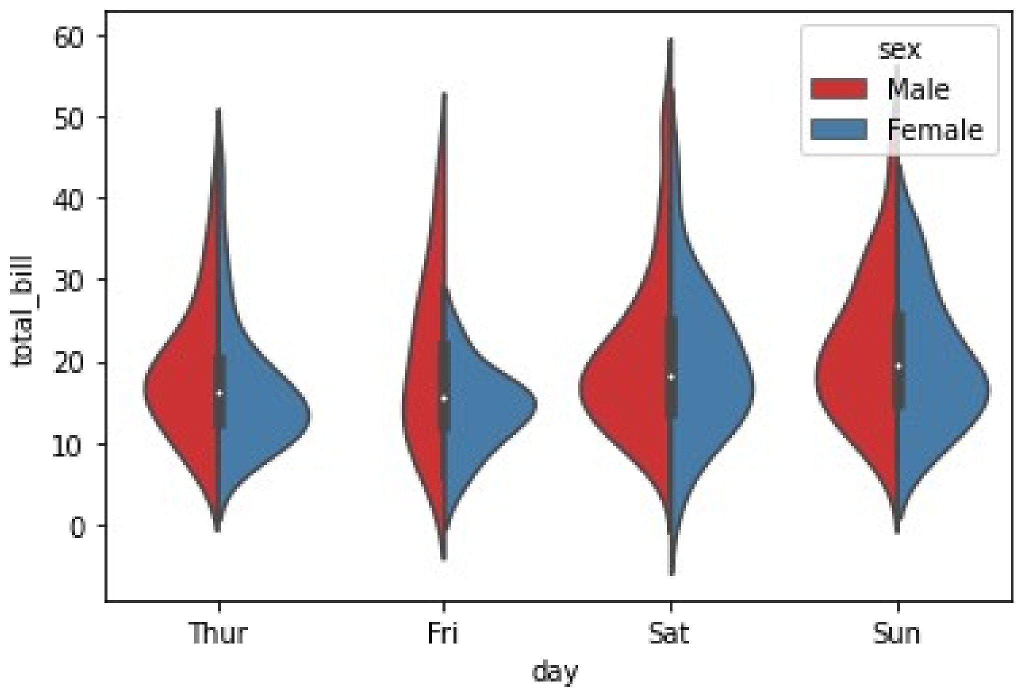

```
sns.violinplot(x="day",y="total_bill", data=tips,hue
='sex',palette='Set1')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0891592910>
```




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

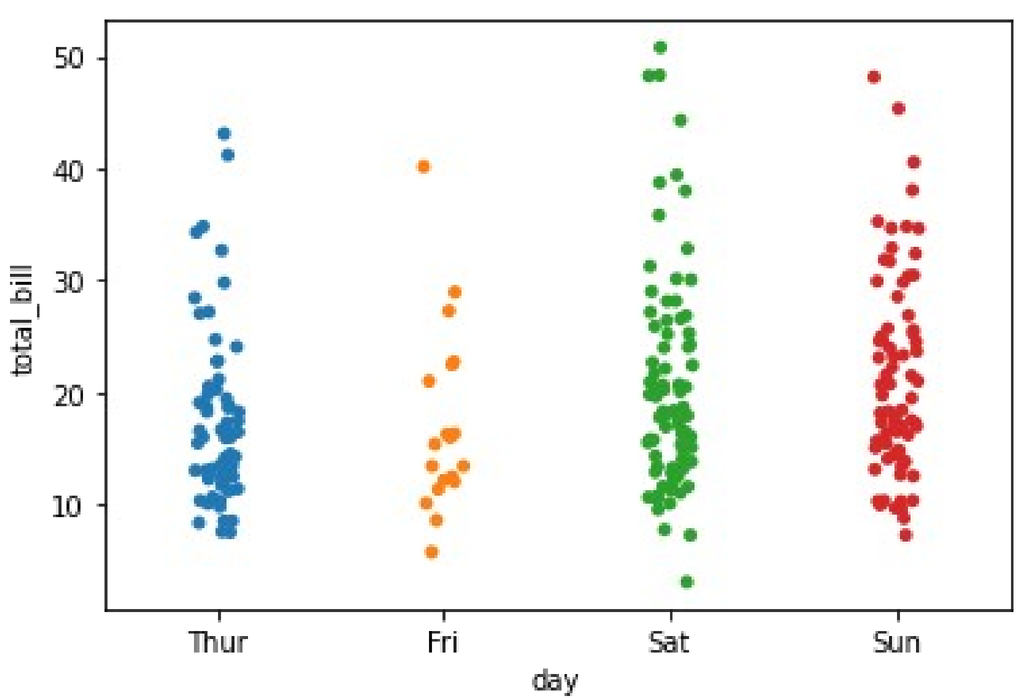
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f08914e5a50>
```



STRIPPLOT:

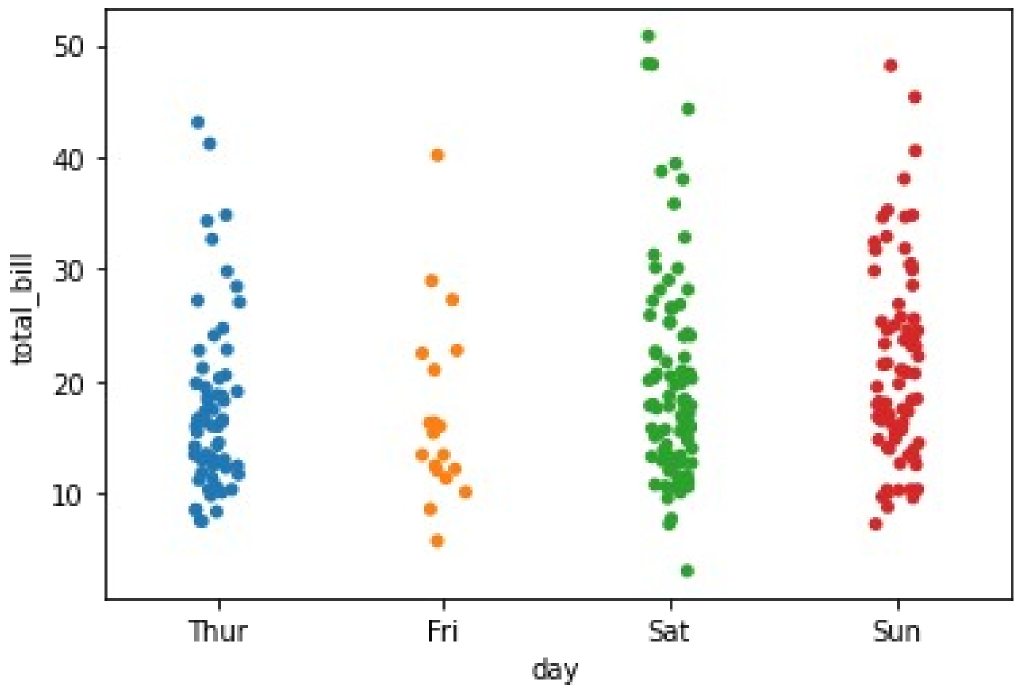
```
sns.stripplot(x="day",y="total_bill", data=tips)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0891438e50>
```



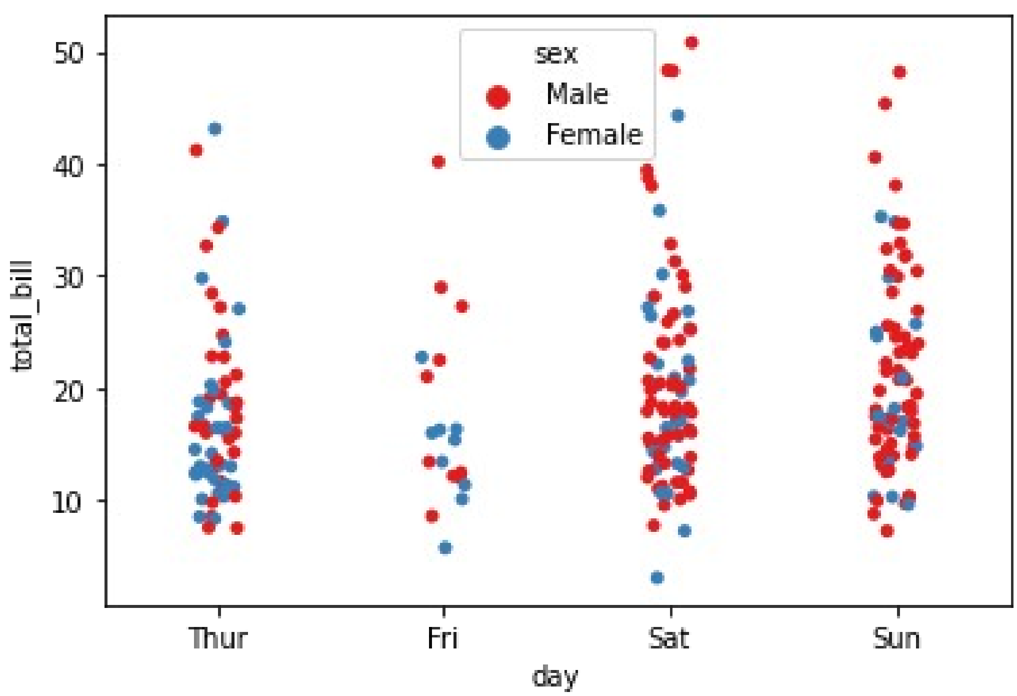
```
sns.stripplot(x="day",y="total_bill", data=tips,jitter=True)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f0891403ed0>




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

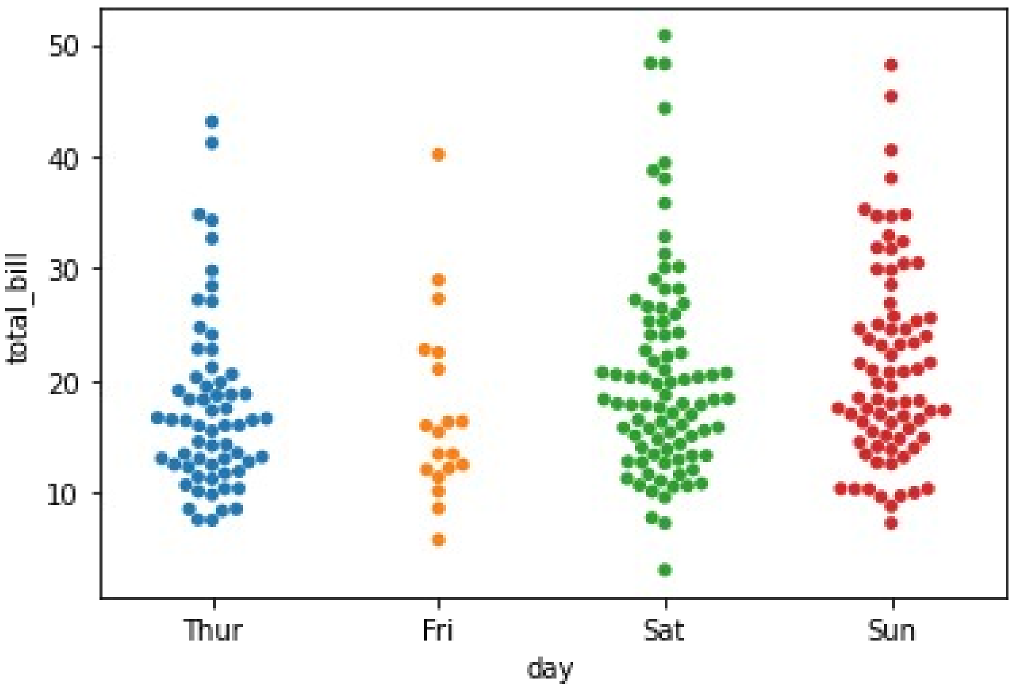
<matplotlib.axes._subplots.AxesSubplot at 0x7f0891367ed0>



SWARMPLOT:

```
sns.swarmplot(x="day",y="total_bill", data=tips)
```

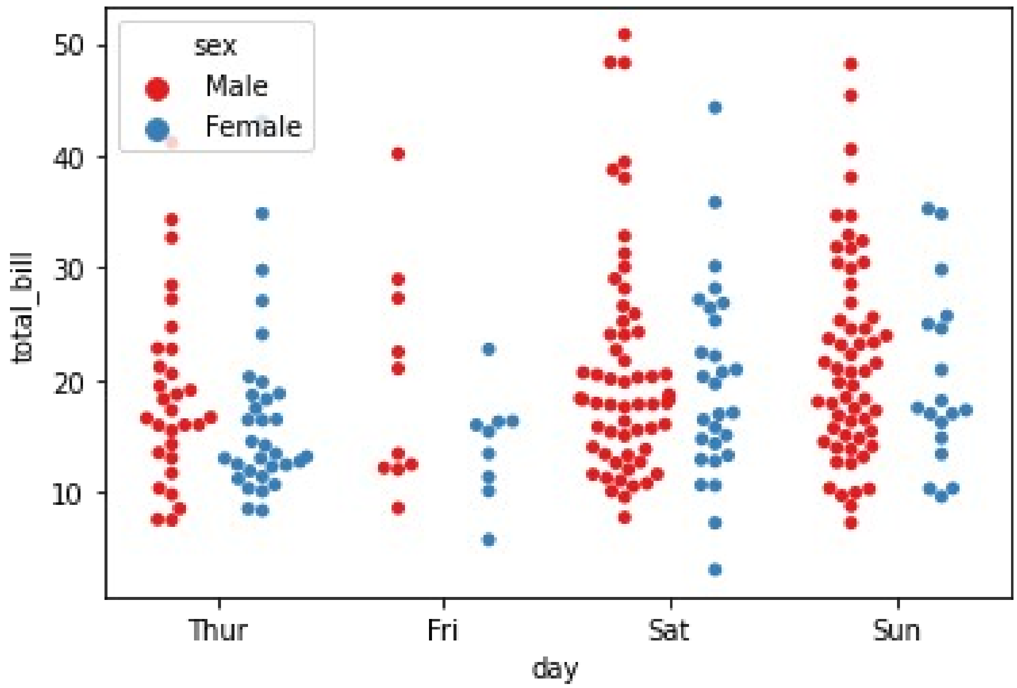
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f08912fa410>
```




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

```
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:3002:
UserWarning: The `split` parameter has been renamed to `dodge`.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296:
UserWarning: 5.1% of the points cannot be placed; you may want to
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f089125c050>
```

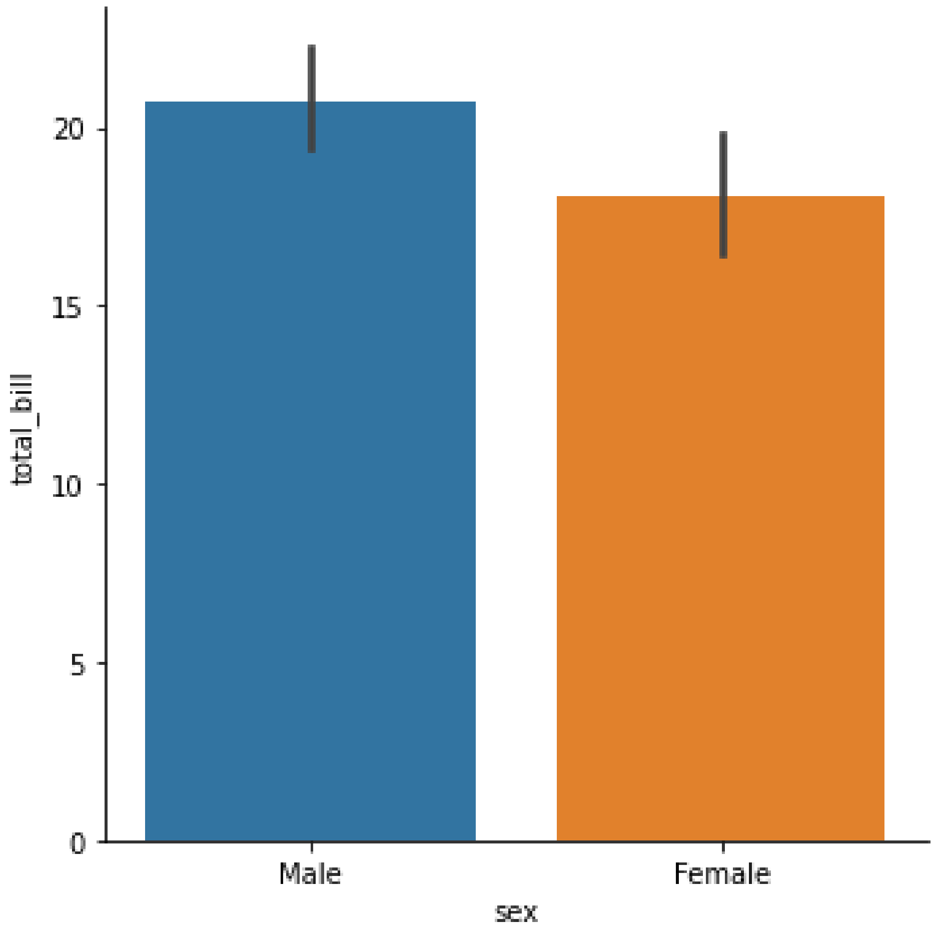


FACTORPLOT: It is the most general form of a categorical plot. It can take in a "kind" parameter to adjust the plot type.

```
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 0x7f089122a490>
```



RESULT:

The program executed successfully and obtained the output.

PROGRAM - 4

AIM: Programs to handle data using pandas.

DATASET:Salaries.csv

SERIES

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

To convert list,numpy, array, or dictionary to a series

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

```
pd.Series(my_list)
```

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

```
pd.Series(my_list,index=labels)
```

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

NumPy Arrays:

```
pd.Series(arr)
```

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

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

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

Dictionary:

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

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

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

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

Using an Index

```
series1 = pd.Series ([1,2,3,4,5] ,
index=['USA','SPAIN','GERMANY','JAPAN','UK'])
```

```
series1
```

```
USA        1
SPAIN      2
GERMANY    3
JAPAN      4
UK         5
dtype: int64
```

```
series2 = pd.Series ([1,2,3,4,5] ,
index=['USA','SPAIN','GERMANY','ITALY','UK'])
series2
```

```
USA        1
SPAIN      2
GERMANY    3
ITALY      4
UK         5
dtype: int64
```

```
series1 + series2
```

```
GERMANY     6.0
ITALY       NaN
JAPAN       NaN
SPAIN       4.0
UK         10.0
USA         2.0
dtype: float64
```

```
pd.Series('USA')
```

```
0    USA
dtype: object
```

DATAFRAMES

```
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
```

```
np.random.seed(95)
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 -0.664289 -0.582431  0.339579  0.599289
B  1.088540  0.097178 -0.142333  1.781748
C  0.588120  0.010733  3.455616 -1.154729
D  0.191124 -0.063445 -0.592004 -1.492861
E -0.393965  1.133565  0.185667 -1.558033
```

SELECTION AND INDEXING

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

```
A    0.599289
B    1.781748
C   -1.154729
D   -1.492861
E   -1.558033
Name: Z, dtype: float64
```

```
df[['Z','W']]
```

```
          Z         W
A  0.599289 -0.664289
B  1.781748  1.088540
C -1.154729  0.588120
D -1.492861  0.191124
E -1.558033 -0.393965
```

CREATING A NEW COLUMN BY ADDING W AND Y

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

df
```

```
          W         X         Y         Z       new
A -0.664289 -0.582431  0.339579  0.599289 -0.324709
B  1.088540  0.097178 -0.142333  1.781748  0.946207
C  0.588120  0.010733  3.455616 -1.154729  4.043737
D  0.191124 -0.063445 -0.592004 -1.492861 -0.400881
E -0.393965  1.133565  0.185667 -1.558033 -0.208298
```

REMOVING new COLUMN:

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

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

```
          W         X         Y         Z
A -0.664289 -0.582431  0.339579  0.599289
B  1.088540  0.097178 -0.142333  1.781748
C  0.588120  0.010733  3.455616 -1.154729
D  0.191124 -0.063445 -0.592004 -1.492861
E -0.393965  1.133565  0.185667 -1.558033
```

```
df
```

```
          W         X         Y         Z       new
A -0.664289 -0.582431  0.339579  0.599289 -0.324709
B  1.088540  0.097178 -0.142333  1.781748  0.946207
C  0.588120  0.010733  3.455616 -1.154729  4.043737
D  0.191124 -0.063445 -0.592004 -1.492861 -0.400881
E -0.393965  1.133565  0.185667 -1.558033 -0.208298
```

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

```
          W         X         Y         Z
A -0.664289 -0.582431  0.339579  0.599289
B  1.088540  0.097178 -0.142333  1.781748
C  0.588120  0.010733  3.455616 -1.154729
D  0.191124 -0.063445 -0.592004 -1.492861
E -0.393965  1.133565  0.185667 -1.558033
```

DROP ROWS:

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

```
W   -0.664289
X   -0.582431
Y    0.339579
Z    0.599289
Name: A, dtype: float64
```

```
df.loc[0]
```

```
df. iloc[0]
```

```
W   -0.664289
X   -0.582431
Y    0.339579
Z    0.599289
Name: A, dtype: float64
```

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

```
0.3395794850282506
```

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

```
          W         Y
A -0.664289  0.339579
B  1.088540 -0.142333
```

CONDITIONAL SELECTION:

```
df
```

```
          W         X         Y         Z
A -0.664289 -0.582431  0.339579  0.599289
B  1.088540  0.097178 -0.142333  1.781748
C  0.588120  0.010733  3.455616 -1.154729
D  0.191124 -0.063445 -0.592004 -1.492861
E -0.393965  1.133565  0.185667 -1.558033
```

```
df>0
```

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

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

```
          W         X         Y         Z
A       NaN       NaN  0.339579  0.599289
B  1.088540  0.097178       NaN  1.781748
C  0.588120  0.010733  3.455616       NaN
D  0.191124       NaN       NaN       NaN
E       NaN  1.133565  0.185667       NaN
```

```
df<0
```

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

```
df[df<0]
```

```
          W         X         Y         Z
A -0.664289 -0.582431       NaN       NaN
B       NaN       NaN -0.142333       NaN
C       NaN       NaN       NaN -1.154729
D       NaN -0.063445 -0.592004 -1.492861
E -0.393965       NaN       NaN -1.558033
```

```
df[df['W']<0]
```

```
          W         X         Y         Z
A -0.664289 -0.582431  0.339579  0.599289
E -0.393965  1.133565  0.185667 -1.558033
```

```
df['W']<0
```

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

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

```
A    0.339579
E    0.185667
Name: Y, dtype: float64
```

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

```
          W         X         Y         Z
A -0.664289 -0.582431  0.339579  0.599289
E -0.393965  1.133565  0.185667 -1.558033
```

```
df[(df['W']<0) | (df['Y']>0)]
```

```
          W         X         Y         Z
A -0.664289 -0.582431  0.339579  0.599289
C  0.588120  0.010733  3.455616 -1.154729
E -0.393965  1.133565  0.185667 -1.558033
```

MISSING DATA

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

```
df = pd.DataFrame({'A': [1,2,np.NaN],'B':[5,np.NaN,np.NaN],'C':
[1,2,3]})
df
```

```
     A    B  C
0  1.0  5.0  1
1  2.0  NaN  2
2  NaN  NaN  3
```

```
df.dropna()
```

```
     A    B  C
0  1.0  5.0  1
```

```
df.dropna(axis = 1)
```

```
   C
0  1
1  2
2  3
```

```
df.dropna(thresh=2)
```

```
     A    B  C
0  1.0  5.0  1
1  2.0  NaN  2
```

```
df.fillna(value='FILL VALUE')
```

```
            A           B  C
0           1           5  1
1           2  FILL VALUE  2
2  FILL VALUE  FILL VALUE  3
```

```
df['A'].fillna(value = df['A'].mean())
```

```
----------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-1-dc2f07cd5363> in <module>()
----> 1 df['A'].fillna(value = df['A'].mean())

NameError: name 'df' is not defined
```

OPERATIONS IN PANDAS

```
df = pd.DataFrame({'col1':[1,2,3,4],'col2':[444,555,666,444],'col3':
['ab','cd','ef','gh']})
df
```

```
   col1  col2 col3
0     1   444   ab
1     2   555   cd
2     3   666   ef
3     4   444   gh
```

INFO ON UNIQUE VALUES

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

```
array([444, 555, 666])
```

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

```
3
```

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

```
444    2
555    1
666    1
Name: col2, dtype: int64
```

SELECTING DATA:

```
#Select from dataframe using criteria from multiple columns
newdf = df[(df['col2']>3)& (df["col2"]==555)]
newdf
```

```
   col1  col2 col3
1     2   555   cd
```

APPLYING FUNCTIONS:

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

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

```
10
```

```
del df['col2']
df
```

```
  col3
0   ab
1   cd
2   ef
3   gh
```

GET COLUMN AND INDEX NAMES:

```
df.columns
```

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

```
df.index
```

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

SORTING AND ORDERING A DATAFRAME

```
df
```

```
  col3
0   ab
1   cd
2   ef
3   gh
```

```
df.sort_values(by='col3') #inplace =False by Default
```

```
  col3
0   ab
1   cd
2   ef
3   gh
```

```
df.sort_values(by='col3', inplace =True)
df
```

```
  col3
0   ab
1   cd
2   ef
3   gh
```

```
df = pd.DataFrame({'col1':[1,2,3,4],'col2':[444,555,666,444]})
df
```

```
   col1  col2
0     1   444
1     2   555
2     3   666
3     4   444
```

```
df.sort_values(by='col1')
```

```
   col1  col2
0     1   444
1     2   555
2     3   666
3     4   444
```

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

```
   col1  col2
0     1   444
3     4   444
1     2   555
2     3   666
```

FIND NULL CVALUES OR CHECK FOR NULL VALUES

```
df.isnull()
```

```
    col1   col2
0  False  False
3  False  False
1  False  False
2  False  False
```

```
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.