Lecture 6-1

Introducing data visualization in Python: Matplotlib

Week 6 Monday

Miles Chen, PhD

Adapted from *Python for Data Science* by Jake VanderPlas

References:

https://matplotlib.org/stable/api/pyplot_summary.html

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html#module-matplotlib.pyplot

Always: `import matplotlib.pyplot as plt`

Lecture 6-1

Introducing data visualization in Python: Matplotlib

Week 6 Monday

Miles Chen, PhD

Adapted from *Python for Data Science* by Jake VanderPlas

References:

https://matplotlib.org/stable/api/pyplot_summary.html

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html#module-matplotlib.pyplot

Always: `import matplotlib.pyplot as plt`

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

Creating a simple plot.

First, we create arrays of the values to plot.

We create an array of 500 values from 0 to pi.

In [2]:
```
x = np.linspace(0, np.pi * 2, 500)
```

We then calculate the y values we wish to plot. In this case, we'll keep it simple and calculate sin of x (in radians).

In [3]:
```
y = np.sin(x)
```

Line Plot

To create a plot, you can call `plt.plot(x, y)`. To have it appear, you call `plt.show()`. `plt.show()` is much like calling `print()`. In Jupyter, if you do not call `plt.show()`, the plot will still often appear, but it is generally considered good practice to call `plt.show()`.

Prior to calling `plt.show()`, you can call optionally other functions which will modify the plot, such as adding a title or changing the axis limits.

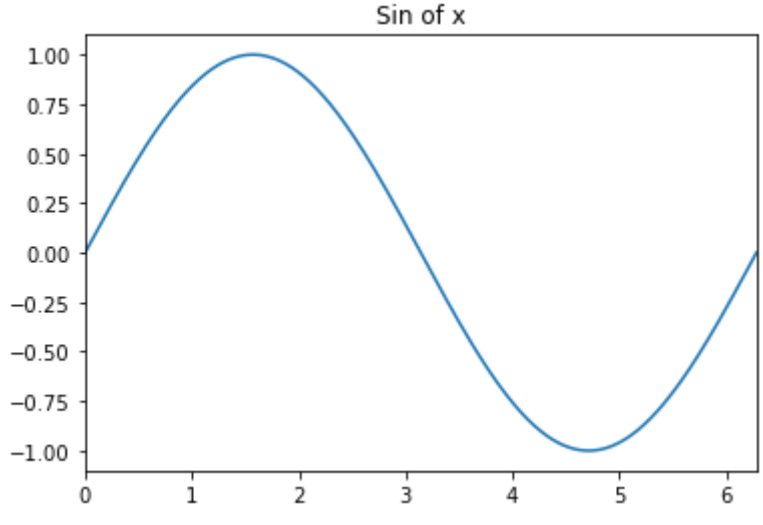
Line Plot

To create a plot, you can call `plt.plot(x, y)`. To have it appear, you call `plt.show()`. `plt.show()` is much like calling `print()`. In Jupyter, if you do not call `plt.show()`, the plot will still often appear, but it is generally considered good practice to call `plt.show()`.

Prior to calling `plt.show()`, you can call optionally other functions which will modify the plot, such as adding a title or changing the axis limits.

In [4]:

```
plt.plot(x, y)
plt.title("Sin of x") # optional
plt.xlim(0,2 * np.pi) # optional
plt.show()
```

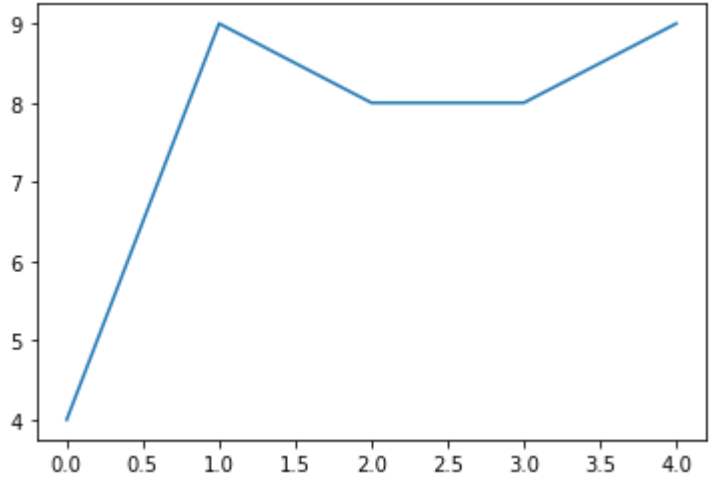


The default behavior of `plt.plot()` is to plot the points and connect them with lines.

The default behavior of `plt.plot()` is to plot the points and connect them with lines.

In [5]:
```
x = np.arange(0, 5, 1)
print(x)
y = np.random.randint(0, 10, 5)
print(y)
plt.plot(x, y)
plt.show()
```

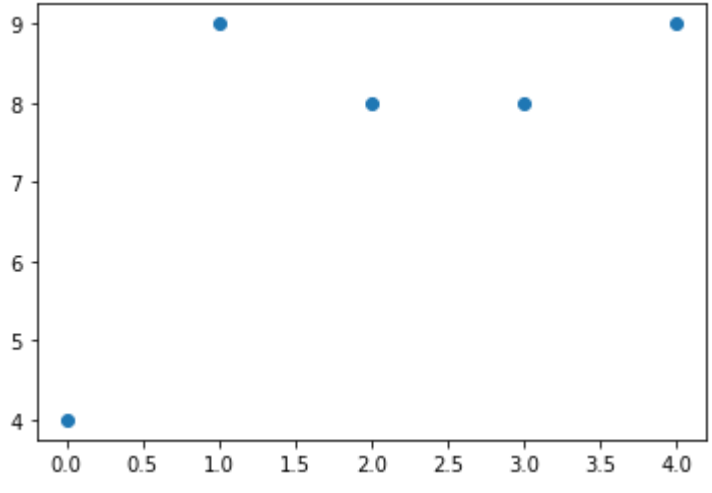
```
[0 1 2 3 4]
[4 9 8 8 9]
```



Scatterplot

If you do not want the points to be connected with a line, you can ask for a scatterplot.

In [6]:
```
plt.scatter(x,y)
plt.show()
```



Multiple functions or sequences on the same graph

Repeated calls to `plt.plot()` will add lines to the same plot.

Multiple functions or sequences on the same graph

Repeated calls to `plt.plot()` will add lines to the same plot.

In [7]:

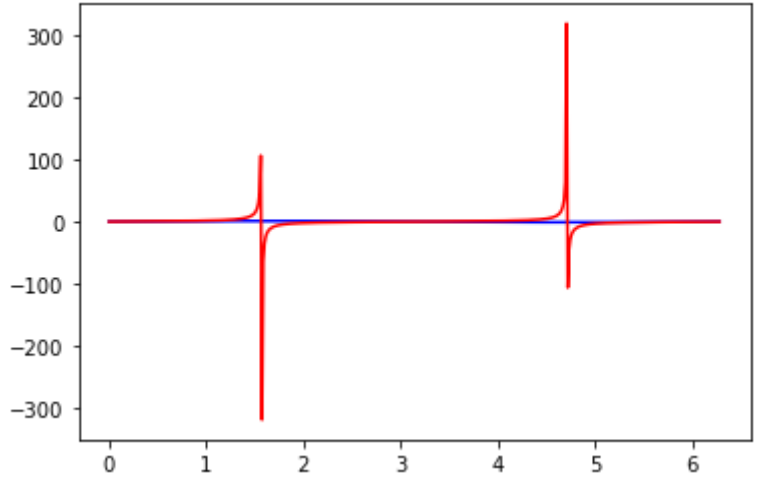
```
x = np.linspace(0, np.pi * 2, 500)
y = np.sin(x)
z = np.tan(x)
w = np.cos(x)
```

Multiple functions or sequences on the same graph

Repeated calls to `plt.plot()` will add lines to the same plot.

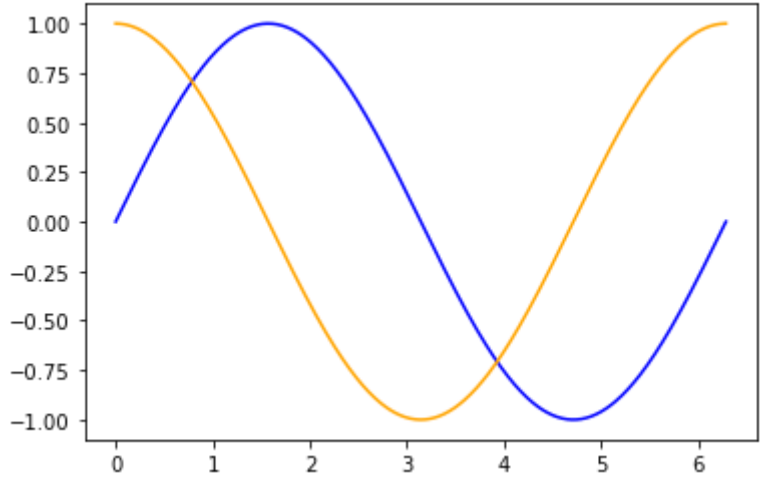
In [7]:
```
x = np.linspace(0, np.pi * 2, 500)
y = np.sin(x)
z = np.tan(x)
w = np.cos(x)
```

In [8]:
```
plt.plot(x, y, 'blue')
plt.plot(x, z, 'red')
plt.show() # the scale of the two functions make this graph undesirable
```



In [9]:

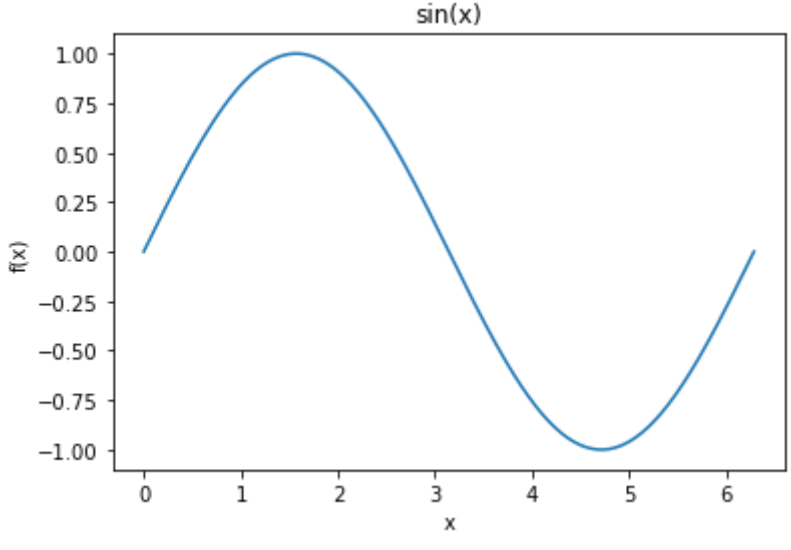
```
plt.plot(x, y, 'blue')
plt.plot(x, w, 'orange')
plt.show()
```



changing labels and axis limits

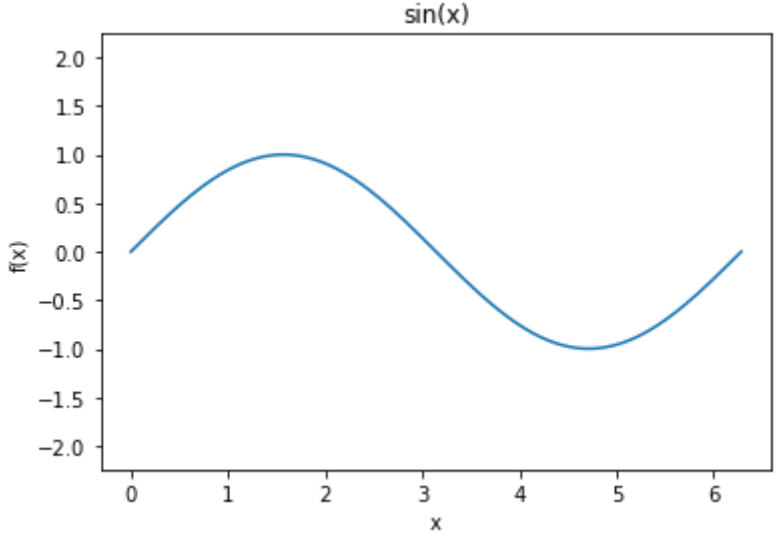
changing labels and axis limits

In [10]:
```
plt.plot(x, y)
plt.title('sin(x)')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.show()
```



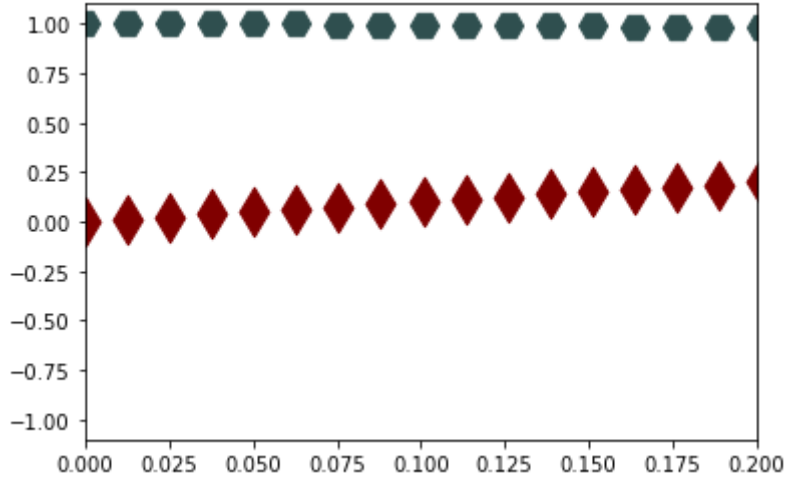
In [11]:

```
plt.plot(x, y)
plt.title('sin(x)')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.xlim(0, 2 * np.pi)
plt.axis('equal')  # aspect ratio # investigate later
plt.show()
```



In [12]:

```
plt.scatter(x, y, s = 300, marker = 'd', color = 'maroon')
plt.scatter(x, w, s = 200, marker = 'H', color = 'darkslategrey')
plt.xlim([0, .2])
plt.show()
```

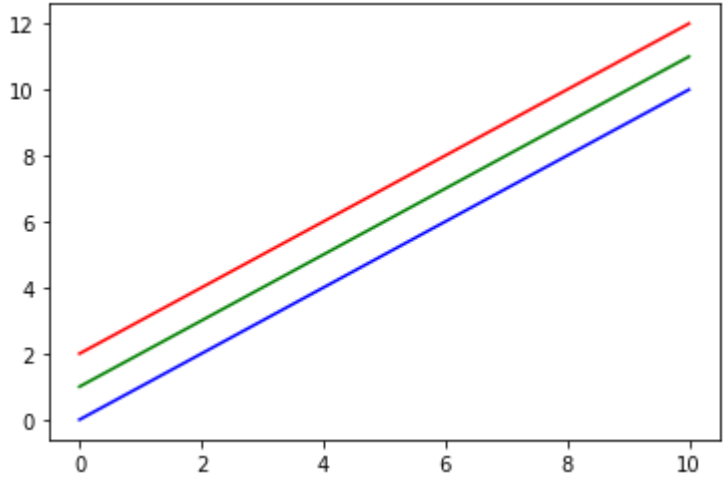


Plot options: colors

Plot options: colors

In [13]:

```
# colors
x = np.linspace(0, 10, 1000)
plt.plot(x, x, color = 'b')
plt.plot(x, x+1, color = 'g')
plt.plot(x, x+2, color = 'r')
plt.show()
```



List of colors and single character shortcuts

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

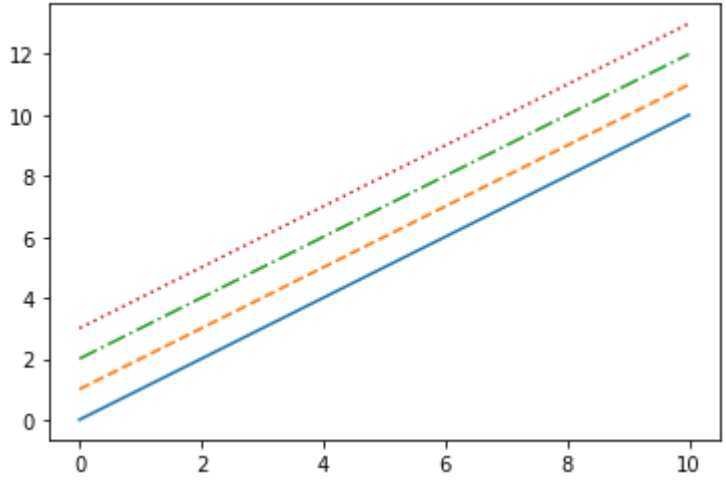
List of Named Colors: https://matplotlib.org/examples/color/named_colors.html

Plot options: line type

Plot options: line type

In [14]:

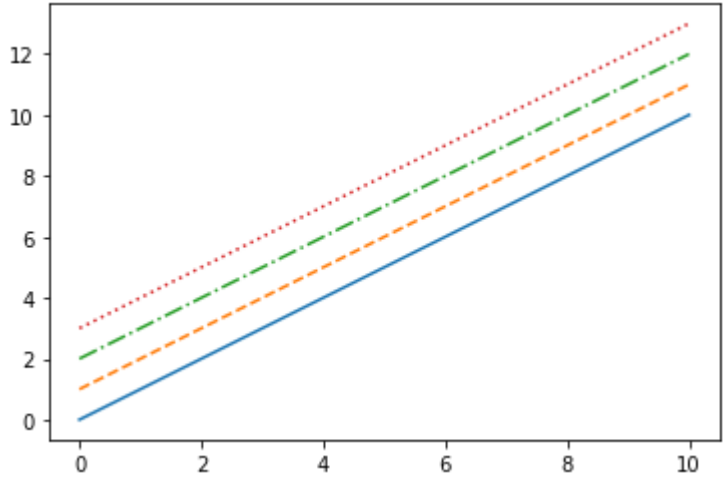
```
# line style
x = np.linspace(0, 10, 1000)
plt.plot(x,  x , linestyle = '-')  # solid
plt.plot(x, x+1, linestyle = '--') # dashed
plt.plot(x, x+2, linestyle = '-.') # dash dot
plt.plot(x, x+3, linestyle = ':')  # dotted
plt.show()

# default behavior uses different colors for multiple lines
```



Plot options: line type

In [14]:

```
# line style
x = np.linspace(0, 10, 1000)
plt.plot(x,  x , linestyle = '-')  # solid
plt.plot(x, x+1, linestyle = '--') # dashed
plt.plot(x, x+2, linestyle = '-.') # dash dot
plt.plot(x, x+3, linestyle = ':')  # dotted
plt.show()

# default behavior uses different colors for multiple lines
```



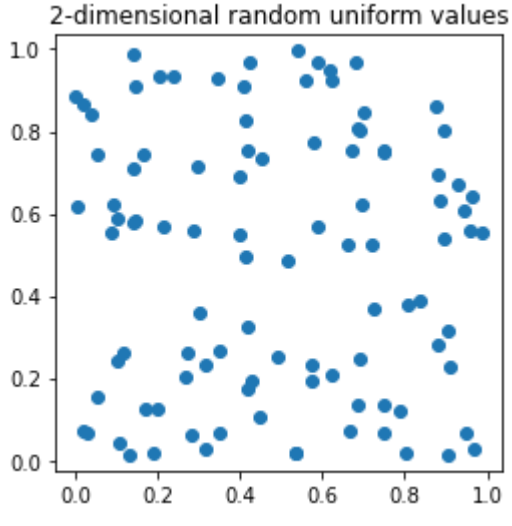
https://matplotlib.org/examples/lines_bars_and_markers/line_styles_reference.html

Plot options: figure size, axis, title

Plot options: figure size, axis, title

In [15]:

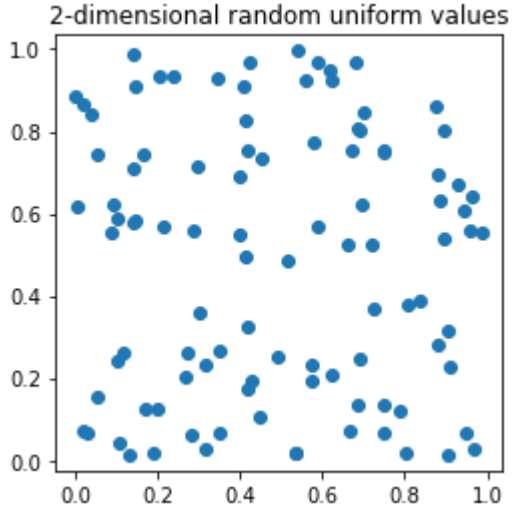
```
np.random.seed(1)
x = np.random.random(100)
y = np.random.random(100)
plt.figure(figsize = (4,4))  # define the properties of the figure first
plt.scatter(x,y)             # then add content
plt.axis('equal')            # alter properties
plt.title('2-dimensional random uniform values')
plt.show()
```



Plot options: figure size, axis, title

In [15]:

```
np.random.seed(1)
x = np.random.random(100)
y = np.random.random(100)
plt.figure(figsize = (4,4))  # define the properties of the figure first
plt.scatter(x,y)             # then add content
plt.axis('equal')            # alter properties
plt.title('2-dimensional random uniform values')
plt.show()
```



documentation for figure() https://matplotlib.org/api/_as_gen/matplotlib.pyplot.figure.html

adding point size and color options to a scatter plot

adding point size and color options to a scatter plot

In [16]:
```
z = 100 * np.random.randint(1,11,100)  # use for size
w = np.random.randint(0,5,100) # use for color
print(z)
print(w)
```

```
[1000  600  500 1000  600  300  600  700  700  900  800  800  800  300
  700  100  600  300  200  900  600 1000  500 1000  200  300  100  500
  800  100  700  300  500  400  700  800  700  400  100  700  500  800
  700  300 1000  600 1000 1000 1000  900  700  500  300 1000  500  100
  100  400  500 1000  400 1000  200  300  600  500  100  900  300  400
 1000 1000  500  500  900  300  200  700  400  900 1000  800  100  600
  300  300  900  600  100  600 1000  900  700  700  100  500  800  400
  100  200]
[2 0 1 4 2 3 4 4 2 1 2 0 3 3 2 0 0 0 0 2 4 0 4 1 2 1 2 4 1 3 1 1 2 4 1 0 2
 1 2 0 0 3 4 1 0 4 0 3 2 4 3 2 4 2 4 0 0 4 2 2 4 2 3 0 0 4 3 4 3 3 4 0 3 1
 4 4 3 2 2 2 2 2 0 2 1 2 3 0 0 1 1 3 3 3 1 3 3 3 1 3]
```

adding point size and color options to a scatter plot

In [16]:
```
z = 100 * np.random.randint(1,11,100)  # use for size
w = np.random.randint(0,5,100) # use for color
print(z)
print(w)
```

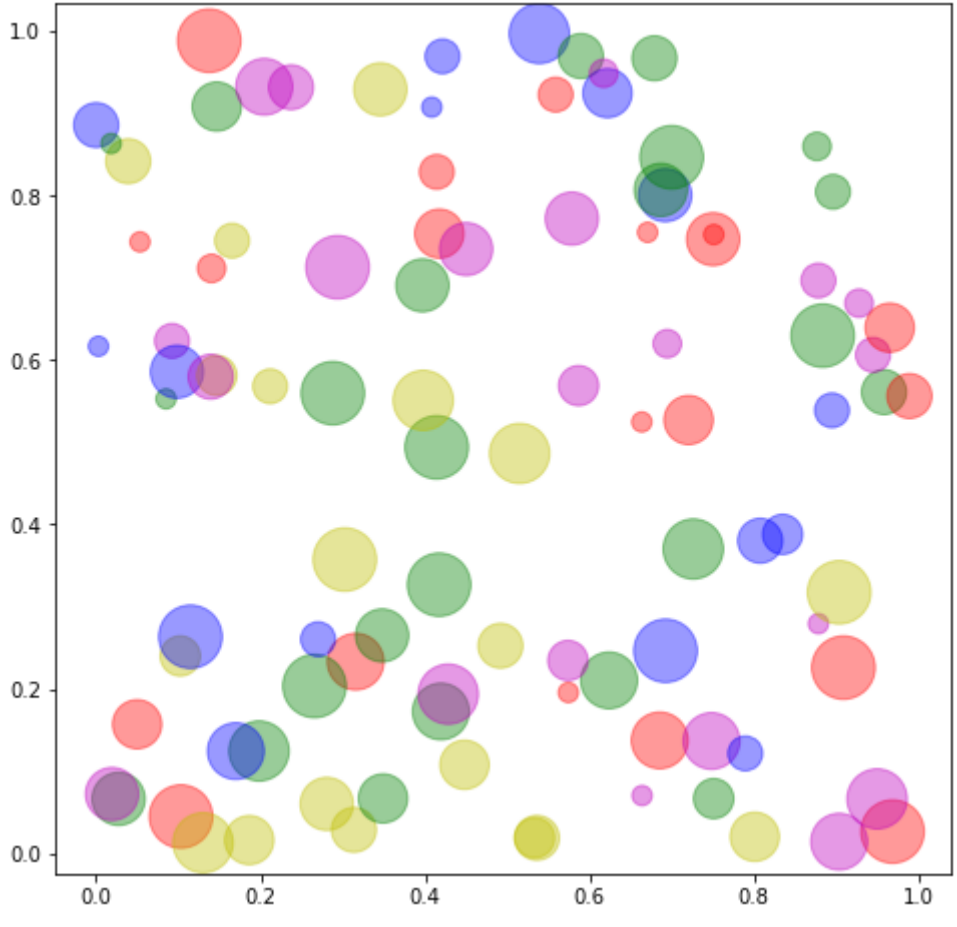
```
[1000  600  500 1000  600  300  600  700  700  900  800  800  800  300
  700  100  600  300  200  900  600 1000  500 1000  200  300  100  500
  800  100  700  300  500  400  700  800  700  400  100  700  500  800
  700  300 1000  600 1000 1000 1000  900  700  500  300 1000  500  100
  100  400  500 1000  400 1000  200  300  600  500  100  900  300  400
 1000 1000  500  500  900  300  200  700  400  900 1000  800  100  600
  300  300  900  600  100  600 1000  900  700  700  100  500  800  400
  100  200]
[2 0 1 4 2 3 4 4 2 1 2 0 3 3 2 0 0 0 0 2 4 0 4 1 2 1 2 4 1 3 1 1 2 4 1 0 2
 1 2 0 0 3 4 1 0 4 0 3 2 4 3 2 4 2 4 0 0 4 2 2 4 2 3 0 0 4 3 4 3 3 4 0 3 1
 4 4 3 2 2 2 2 2 0 2 1 2 3 0 0 1 1 3 3 3 1 3 3 3 1 3]
```

In [17]:
```
colors = np.array(['r','b','g','m','y'])
c = colors[w]
print(c)
```

```
['g' 'r' 'b' 'y' 'g' 'm' 'y' 'y' 'g' 'b' 'g' 'r' 'm' 'm' 'g' 'r' 'r' 'r'
 'r' 'g' 'y' 'r' 'y' 'b' 'g' 'b' 'g' 'y' 'b' 'm' 'b' 'b' 'g' 'y' 'b' 'r'
 'g' 'b' 'g' 'r' 'r' 'm' 'y' 'b' 'r' 'y' 'r' 'm' 'g' 'y' 'm' 'g' 'y' 'g'
 'y' 'r' 'r' 'y' 'g' 'g' 'y' 'g' 'm' 'r' 'r' 'y' 'm' 'y' 'm' 'm' 'y' 'r'
 'm' 'b' 'y' 'y' 'm' 'g' 'g' 'g' 'g' 'g' 'r' 'g' 'b' 'g' 'm' 'r' 'r' 'b'
 'b' 'm' 'm' 'm' 'b' 'm' 'm' 'm' 'b' 'm']
```

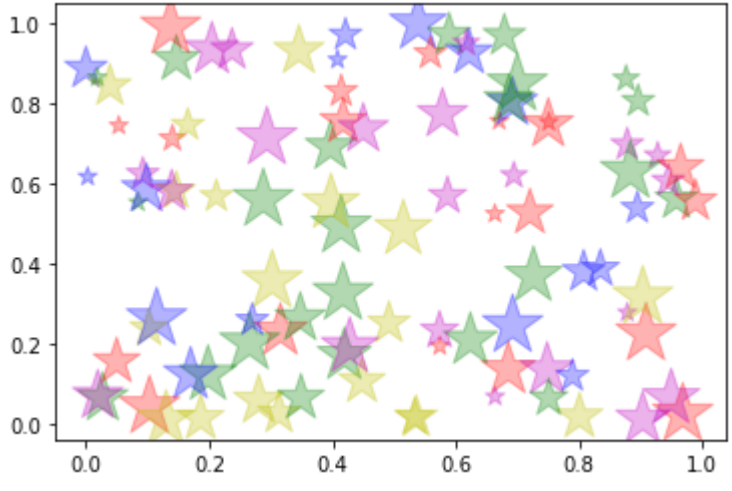
In [18]:

```
# we can map the property size (s) to a variable z
# we can map the color (c) to a variable c
plt.figure(figsize = (8,8))
plt.scatter(x, y, s = z, c = c, alpha = 0.4 )  # alpha modifies transparency
plt.axis('equal')
plt.show()
```



In [19]:

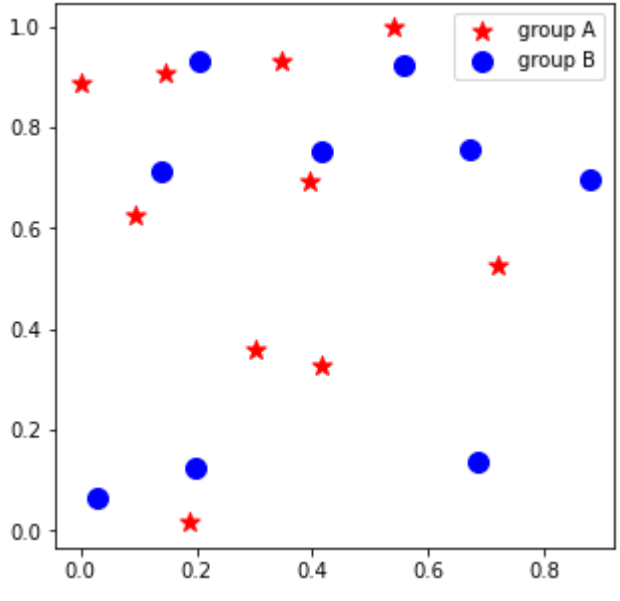
```
# you can change the point markers
plt.scatter(x, y, s = z, c = c, marker = "*", alpha = 0.3)
# But you can't map the markers to a variable
plt.show()
```



character	description
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker

In [20]:

```
# You can call 'scatter' multiple times to plot different groups, each with its own label
# then call plt.legend() to add a legend with the appropriate labels
plt.figure(figsize = (5,5))
plt.scatter(x[0:10] , y[0:10] , s = 100, c = 'red' , marker = "*", label = 'group A')
plt.scatter(x[11:20], y[11:20], s = 100, c = 'blue', marker = "o", label = 'group B')
plt.legend()
plt.show()
```

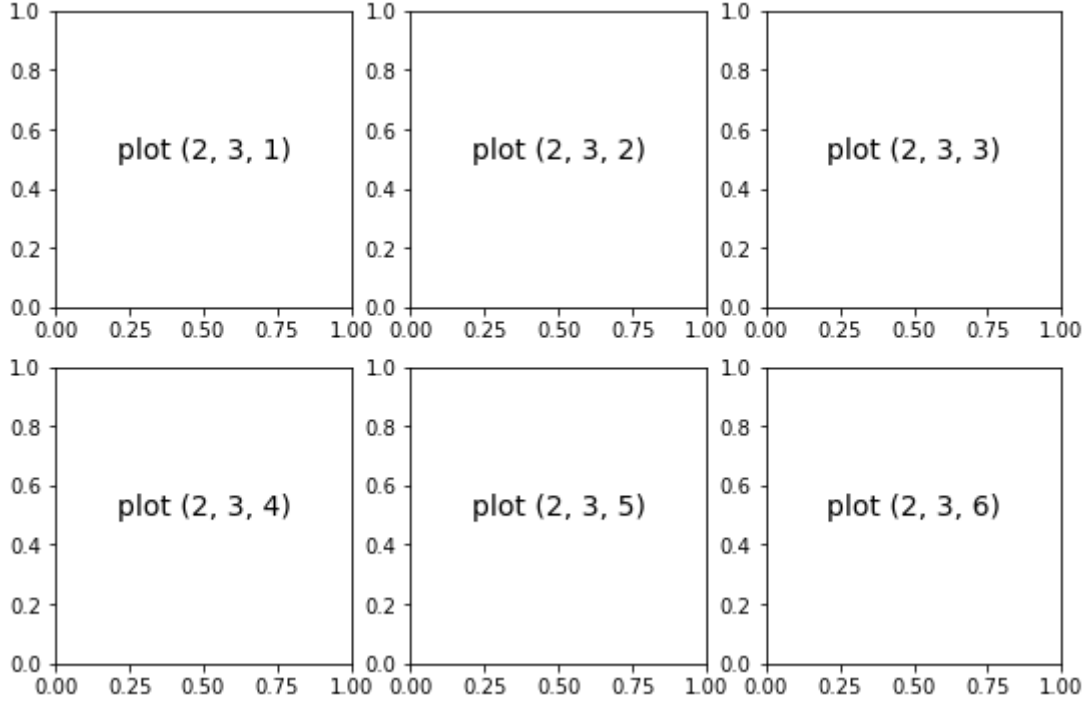


Complex layouts with subplots

Complex layouts with subplots

In [21]:

```
# Subplots
plt.figure(figsize = (9,6))
for i in range(1, 7):
    plt.subplot(2, 3, i)   # two rows of subplots, three columns of plots, work in plot i
    plt.text(0.5, 0.5, 'plot ' + str((2, 3, i)), fontsize = 14, ha='center')
    # plt.text adds text, first two will be the xy position, the character string
```

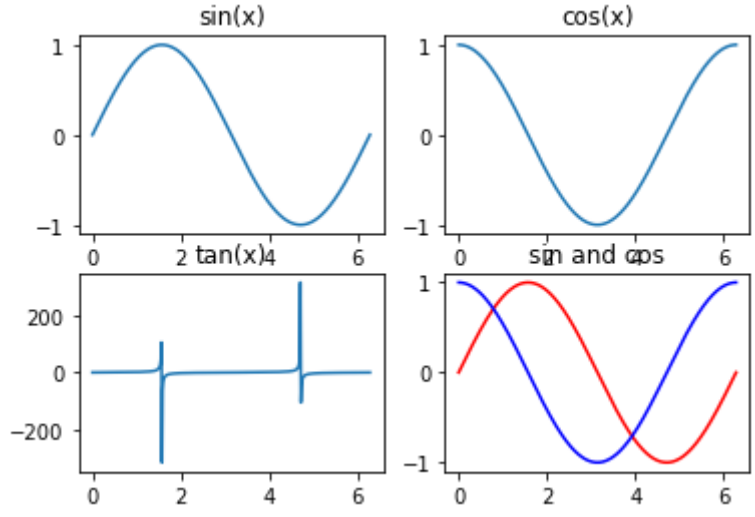


Subplots example

Subplots example

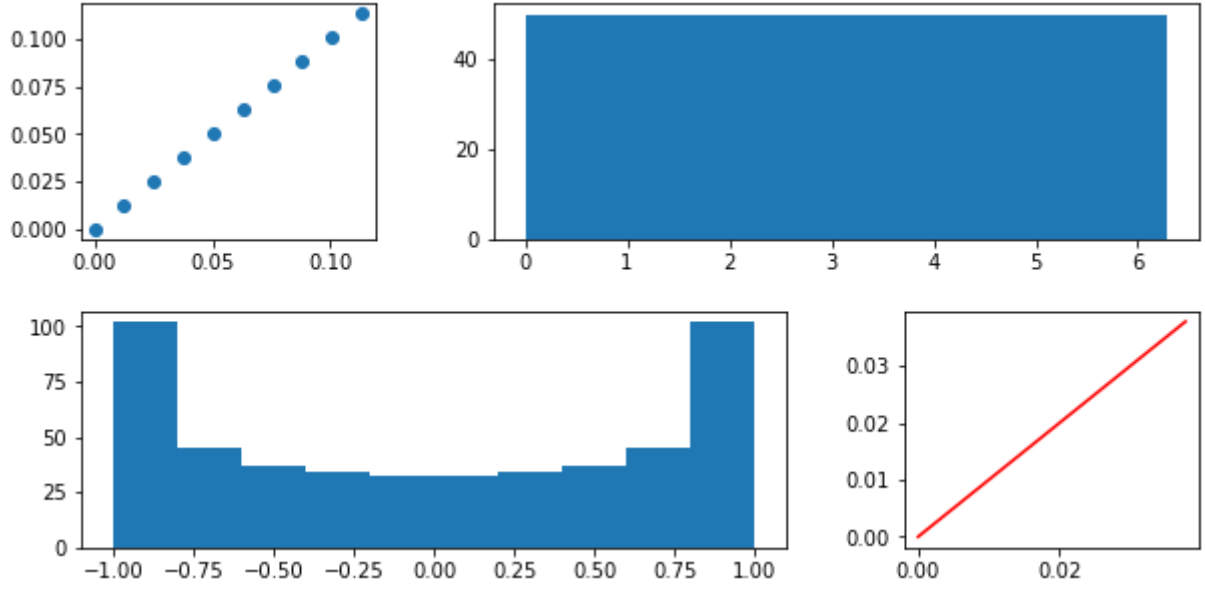
In [22]:

```
x = np.linspace(0, np.pi * 2, 500)
y = np.sin(x)
z = np.tan(x)
w = np.cos(x)
plt.subplot(2,2,1)  # number of rows, number of columns, which plot you want to draw
plt.plot(x, y)
plt.title("sin(x)")

plt.subplot(2,2,2) # subplot 2 will be upper right
plt.plot(x, w)
plt.title("cos(x)")

plt.subplot(2,2,3) # subplot 3 will be lower left
plt.plot(x, z)
plt.title("tan(x)")


plt.subplot(2,2,4)
plt.plot(x,y, 'red')
plt.plot(x,w, 'blue')
plt.title("sin and cos")

plt.show()
```


sin(x)
cos(x)
tan(x)
sin and cos
1
0
−1
200
−200
0
2
4
6

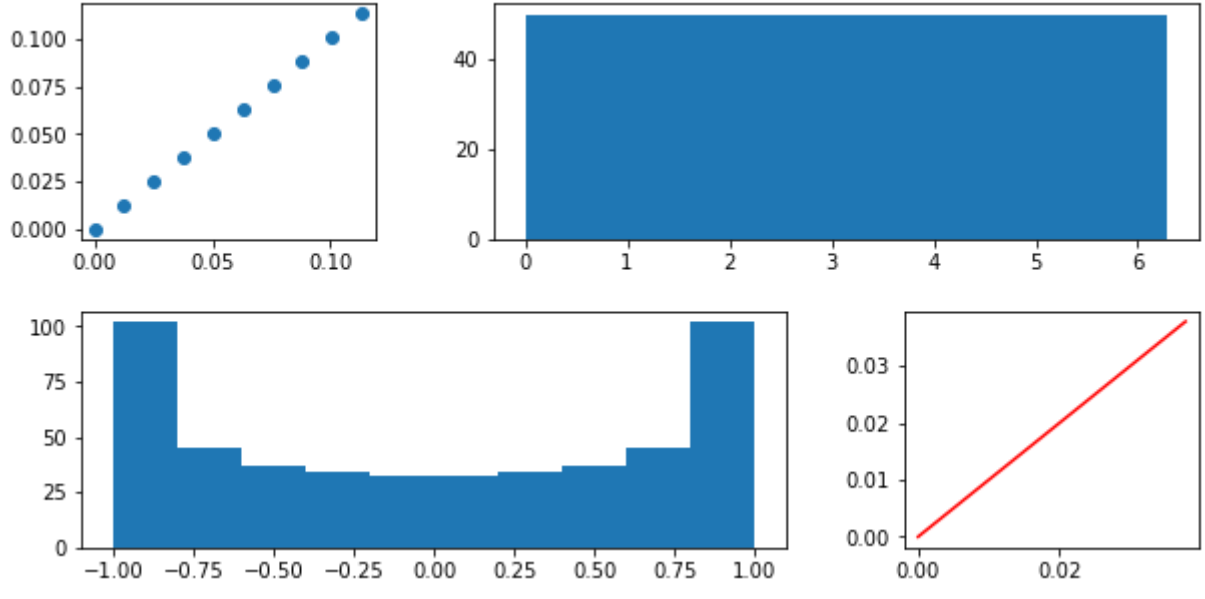
In [23]:

```
plt.figure(figsize = (10,5))
# More Complicated Grids
grid = plt.GridSpec(2, 3, wspace=0.4, hspace=0.3)
plt.subplot(grid[0, 0]) # gridspec uses 0 based indexing
plt.scatter(x[0:10], y[0:10])
plt.subplot(grid[0, 1:3])  # top row, columns 1:3
plt.hist(x)
plt.subplot(grid[1, :2])
plt.hist(y)
plt.subplot(grid[1, 2])
plt.plot(x[0:4], y[0:4], c = 'r')
plt.show()
```



In [23]:

```
plt.figure(figsize = (10,5))
# More Complicated Grids
grid = plt.GridSpec(2, 3, wspace=0.4, hspace=0.3)
plt.subplot(grid[0, 0]) # gridspec uses 0 based indexing
plt.scatter(x[0:10], y[0:10])
plt.subplot(grid[0, 1:3])  # top row, columns 1:3
plt.hist(x)
plt.subplot(grid[1, :2])
plt.hist(y)
plt.subplot(grid[1, 2])
plt.plot(x[0:4], y[0:4], c = 'r')
plt.show()
```



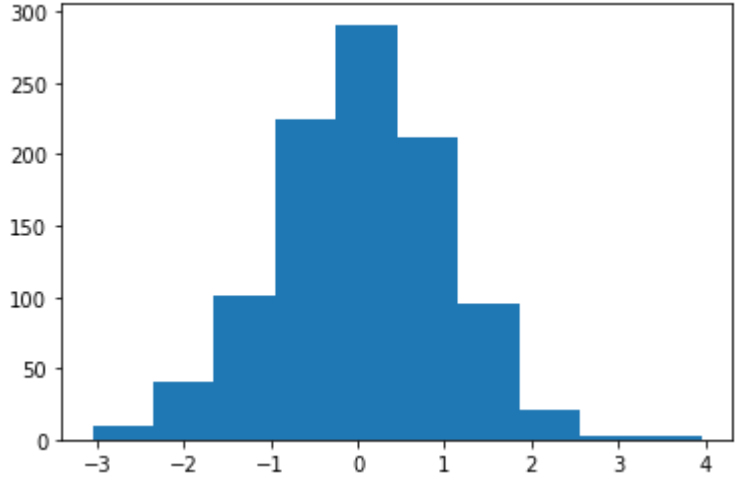
subplot documentation: https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html?highlight=subplot#matplotlib.pyplot.subplot

Histograms

Histograms

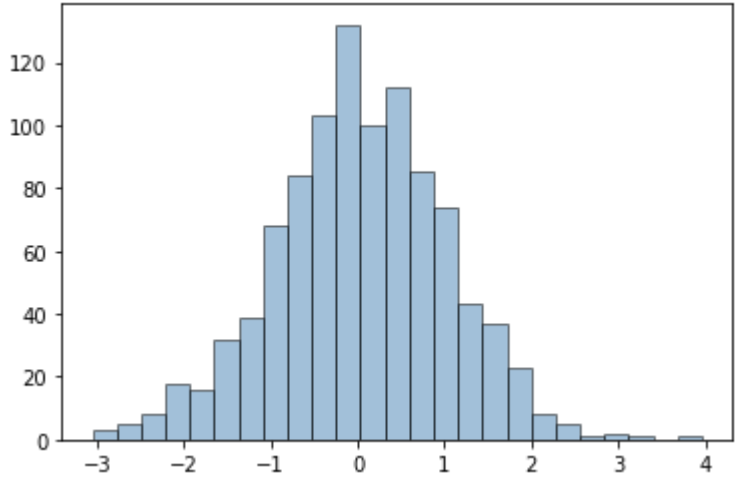
In [24]:

```
# more histograms

data = np.random.randn(1000)

plt.hist(data)
plt.show()
```



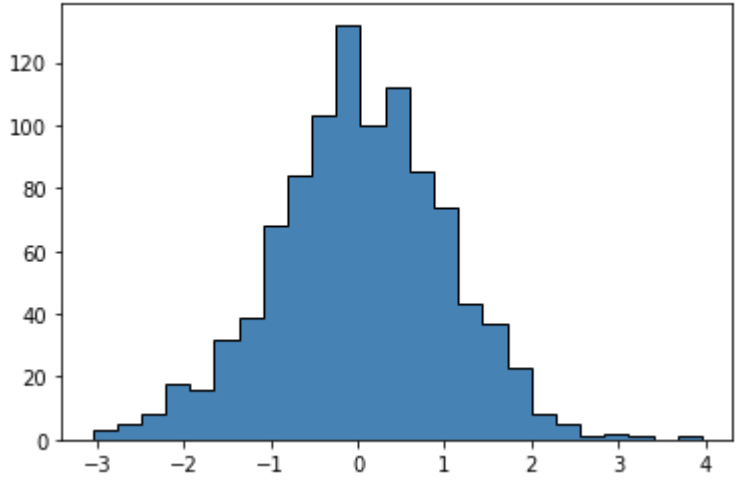
In [25]:

```
# alter bins and color options
plt.hist(data, bins=25, alpha=0.5,
         color='steelblue', edgecolor='black')
plt.show()
```



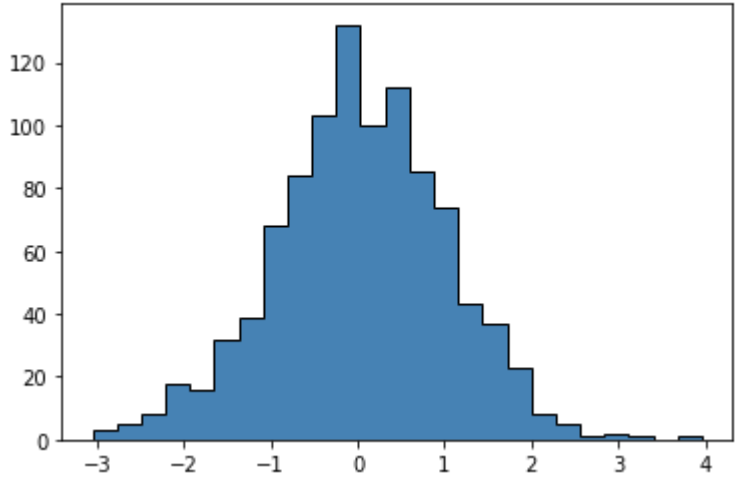
In [26]:

```
# histtype = stepfilled
plt.hist(data, bins=25, alpha=1,
         histtype = 'stepfilled', color='steelblue', edgecolor='black')
plt.show()
```



In [26]:

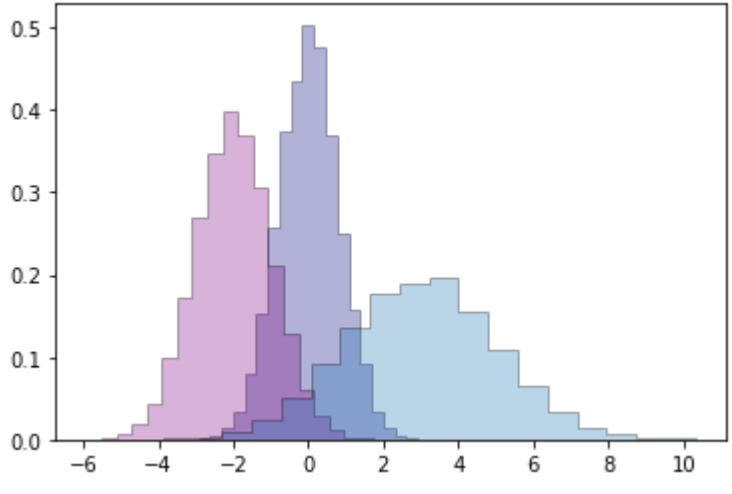
```
# histtype = stepfilled
plt.hist(data, bins=25, alpha=1,
         histtype = 'stepfilled', color='steelblue', edgecolor='black')
plt.show()
```



hist documentation: https://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html?highlight=hist#matplotlib.pyplot.hist

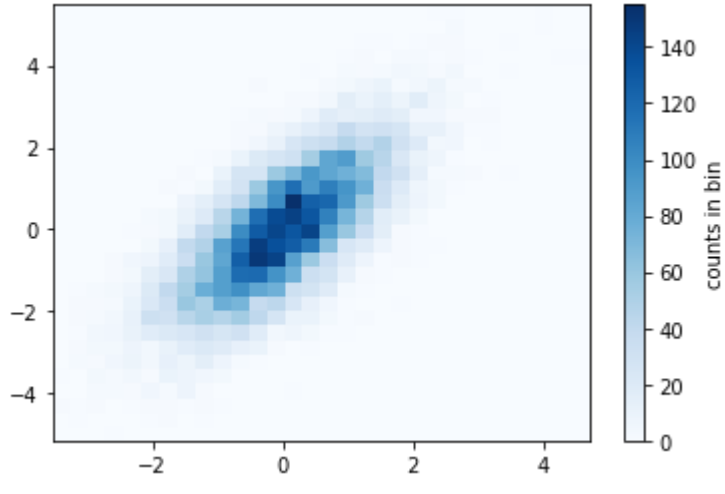
In [27]:

```
x1 = np.random.normal(0, 0.8, 10000)
x2 = np.random.normal(-2, 1, 10000)
x3 = np.random.normal(3, 2, 10000)

# set up a dictionary with arguments
kwargs = dict(histtype='stepfilled', alpha=0.3, density=True, bins=20, edgecolor = 'black')

# use the same arguments for all of the histograms without the need to copy and paste
plt.hist(x1, **kwargs, color = 'navy')
plt.hist(x2, **kwargs, color = "purple")
plt.hist(x3, **kwargs)
plt.show()
```



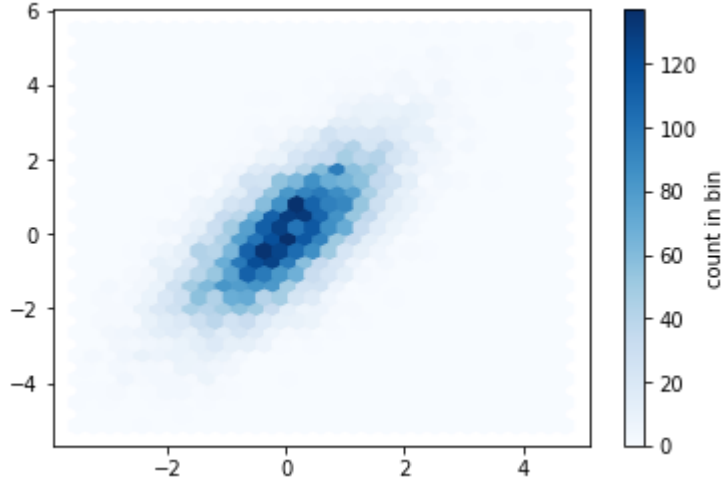
In [28]:

```
# 2d histograms
mean = [0, 0]
cov = [[1, 1],
       [1, 2]]
x, y = np.random.multivariate_normal(mean, cov, 10000).T

plt.hist2d(x, y, bins=30, cmap='Blues')
cb = plt.colorbar()
cb.set_label('counts in bin')
plt.show()
```



In [29]:

```
plt.hexbin(x, y, gridsize=30, cmap='Blues')
cb = plt.colorbar(label='count in bin')
```



two dimensional functions

two dimensional functions

In [30]:
```
x = np.array([0,1,2,3])
y = np.array([-1,0,1, 2])
print(x)
print(y)
X, Y = np.meshgrid(x,y) # sees x has length 3, y has length 4, so the result is a 4 x 3 grid
```

```
[0 1 2 3]
[-1  0  1  2]
```

two dimensional functions

In [30]:
```
x = np.array([0,1,2,3])
y = np.array([-1,0,1, 2])
print(x)
print(y)
X, Y = np.meshgrid(x,y) # sees x has length 3, y has length 4, so the result is a 4 x 3 grid
```

```
[0 1 2 3]
[-1  0  1  2]
```

In [31]:
```
def f(x, y):
    return x * y
X, Y = np.meshgrid(x,y)
Z = f(X, Y)
print(Z)
```

```
[[ 0 -1 -2 -3]
 [ 0  0  0  0]
 [ 0  1  2  3]
 [ 0  2  4  6]]
```

In [32]:

```
x = np.linspace(-3, 3, 101)
y = np.linspace(-3, 3, 101)
X, Y = np.meshgrid(x, y)

def g (x, y):
    return np.exp(-(x**2)) * np.exp(-(y**2)) # product of 2 normal distributions

Z = g(X, Y)
```

In [32]:

```
x = np.linspace(-3, 3, 101)
y = np.linspace(-3, 3, 101)
X, Y = np.meshgrid(x, y)

def g (x, y):
    return np.exp(-(x**2)) * np.exp(-(y**2)) # product of 2 normal distributions

Z = g(X, Y)
```

In [33]:

```
Z.shape
```

Out[33]:

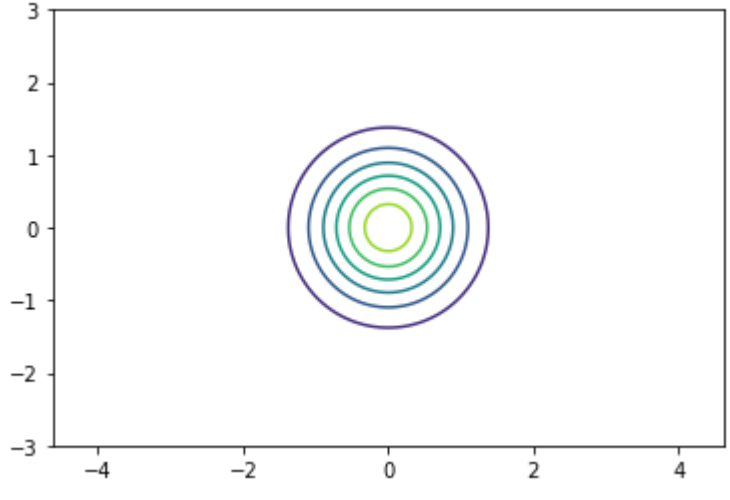
```
(101, 101)
```

In [32]:
```
x = np.linspace(-3, 3, 101)
y = np.linspace(-3, 3, 101)
X, Y = np.meshgrid(x, y)

def g (x, y):
    return np.exp(-(x**2)) * np.exp(-(y**2)) # product of 2 normal distributions

Z = g(X, Y)
```

In [33]:
```
Z.shape
```

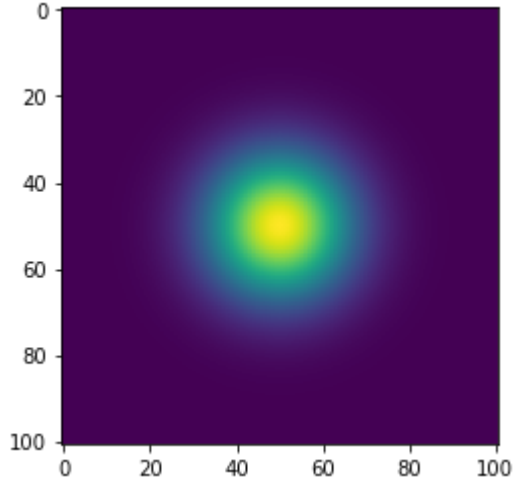
Out[33]:
```
(101, 101)
```

In [34]:
```
plt.contour(X, Y, Z)
plt.axis('equal')
plt.show()
```



In [35]:

```
plt.imshow(Z)
plt.show()
```



Plotting Directly from a Pandas Series

Plotting Directly from a Pandas Series

In [36]:
```
ts = pd.Series(np.random.randn(1000),
   index=pd.date_range('1/1/2000', periods=1000))
```

Plotting Directly from a Pandas Series

In [36]:
```
ts = pd.Series(np.random.randn(1000),
   index=pd.date_range('1/1/2000', periods=1000))
```

In [37]:
```
ts
```

Out[37]:
```
2000-01-01    1.161351
2000-01-02   -0.399784
2000-01-03    0.688814
2000-01-04   -0.253723
2000-01-05   -0.136547
                ...
2002-09-22   -0.068322
2002-09-23    0.453897
2002-09-24   -2.010975
2002-09-25    0.871980
2002-09-26   -1.478780
Freq: D, Length: 1000, dtype: float64
```

Plotting Directly from a Pandas Series

In [36]:
```
ts = pd.Series(np.random.randn(1000),
   index=pd.date_range('1/1/2000', periods=1000))
```

In [37]:
```
ts
```

Out[37]:
```
2000-01-01    1.161351
2000-01-02   -0.399784
2000-01-03    0.688814
2000-01-04   -0.253723
2000-01-05   -0.136547
                ...
2002-09-22   -0.068322
2002-09-23    0.453897
2002-09-24   -2.010975
2002-09-25    0.871980
2002-09-26   -1.478780
Freq: D, Length: 1000, dtype: float64
```

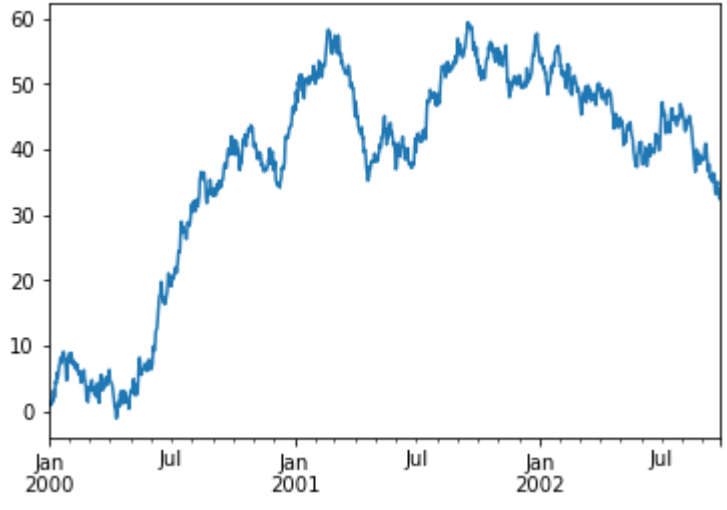
In [38]:
```
ts = ts.cumsum()
```

In [39]:
```
ts
```

Out[39]:
```
2000-01-01     1.161351
2000-01-02     0.761566
2000-01-03     1.450381
2000-01-04     1.196657
2000-01-05     1.060111
                 ...
2002-09-22    34.602416
2002-09-23    35.056313
2002-09-24    33.045338
2002-09-25    33.917318
2002-09-26    32.438537
Freq: D, Length: 1000, dtype: float64
```

In [40]:
```
ts.plot()
```

Out[40]: <AxesSubplot:>



Plotting Directly from a Pandas Data Frame

DataFrames in Pandas support basic visualizations directly.

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html

Plotting Directly from a Pandas Data Frame

DataFrames in Pandas support basic visualizations directly.

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html

Example 1:

A data frame with several columns of time-series data

Plotting Directly from a Pandas Data Frame

DataFrames in Pandas support basic visualizations directly.

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html

Example 1:

A data frame with several columns of time-series data

In [41]:
```
df = pd.DataFrame(np.random.randn(1000, 4), index = ts.index, columns = list('ABCD'))
```

Plotting Directly from a Pandas Data Frame

DataFrames in Pandas support basic visualizations directly.

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html

Example 1:

A data frame with several columns of time-series data

In [41]:
```
df = pd.DataFrame(np.random.randn(1000, 4), index = ts.index, columns = list('ABCD'))
```

In [42]:
```
df = df.cumsum()
df
```

Out[42]:

	A	B	C	D
2000-01-01	1.228413	-0.455540	-0.532579	-1.028464
2000-01-02	2.621425	0.000538	-0.742925	1.427698
2000-01-03	2.170190	-1.408664	-1.085901	2.218128
2000-01-04	1.959056	-1.738246	-0.956508	2.225535
2000-01-05	1.871284	-0.332684	-0.550996	0.869890
...	...	...	...	...
2002-09-22	8.019642	-1.526915	51.653375	-3.242759
2002-09-23	8.525085	-2.704152	50.186367	-2.390122
2002-09-24	8.122309	-0.464831	50.595956	-2.749689
2002-09-25	10.585510	0.196054	51.958605	-1.728218
2002-09-26	11.858522	0.356034	49.999471	-3.080693

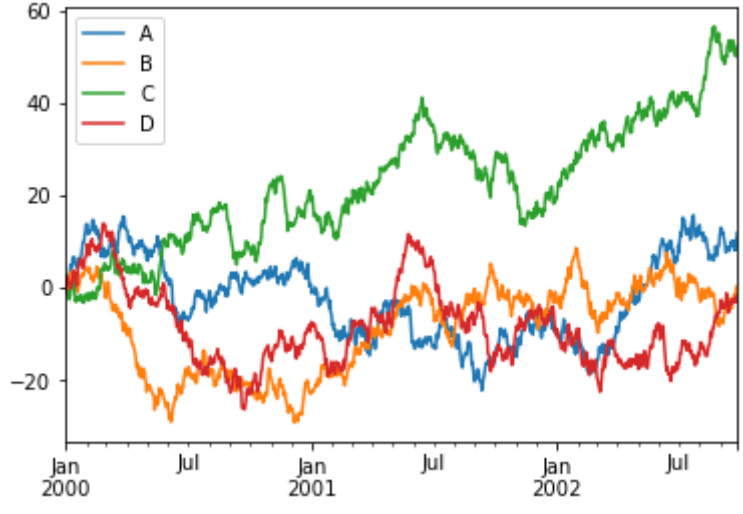
1000 rows × 4 columns

Calling plot on the data frame will produce a plot with a line for each column with a legend

Calling plot on the data frame will produce a plot with a line for each column with a legend

In [43]:
```
plt.figure()
df.plot()
```

Out[43]:
```
<AxesSubplot:>

<Figure size 432x288 with 0 Axes>
```



You can also call plot on a single series and specify kind = "bar" to create a bar plot

You can also call plot on a single series and specify kind = "bar" to create a bar plot

In [44]:
```
df.iloc[5]
```

Out[44]:
```
A    2.575327
B    1.572246
C   -2.606275
D    0.626440
Name: 2000-01-06 00:00:00, dtype: float64
```

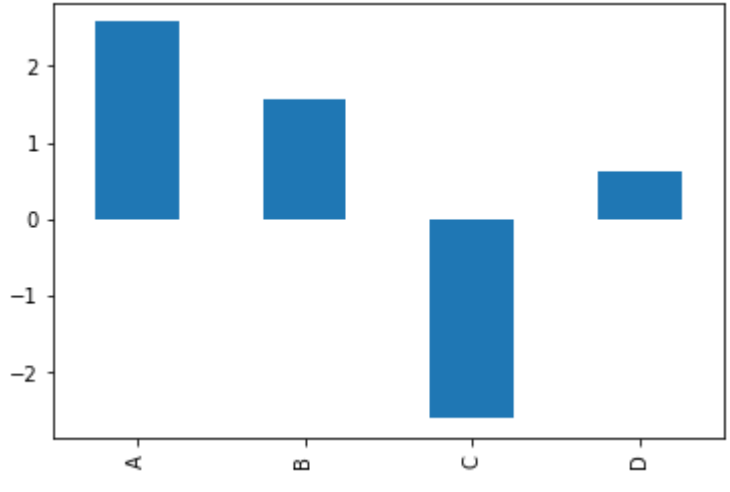
You can also call plot on a single series and specify kind = "bar" to create a bar plot

In [44]:
```
df.iloc[5]
```

Out[44]:
```
A    2.575327
B    1.572246
C   -2.606275
D    0.626440
Name: 2000-01-06 00:00:00, dtype: float64
```

In [45]:
```
df.iloc[5].plot(kind = "bar")
```

Out[45]:
```
<AxesSubplot:>
```



Here's a data frame with some random numbers.

Here's a data frame with some random numbers.

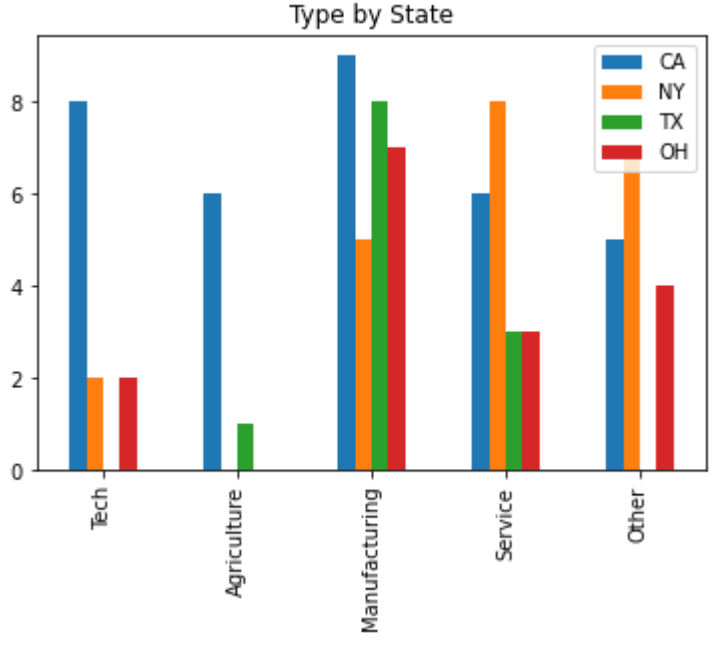
In [46]:
```
# These are random numbers and don't mean anything
df2 = pd.DataFrame(np.random.randint(0,10,size = (5, 4)), columns=['CA', 'NY', 'TX', 'OH'],
                   index = ["Tech","Agriculture","Manufacturing","Service","Other"])
df2
```

Out[46]:

	CA	NY	TX	OH
Tech	8	2	0	2
Agriculture	6	0	1	0
Manufacturing	9	5	8	7
Service	6	8	3	3
Other	5	7	0	4

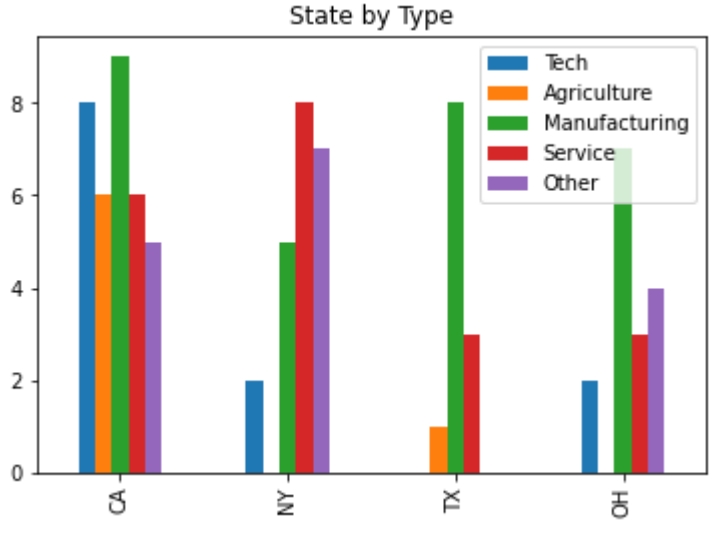
In [47]:
```
df2.plot.bar()
plt.title("Type by State")
```

Out[47]:
```
Text(0.5, 1.0, 'Type by State')
```



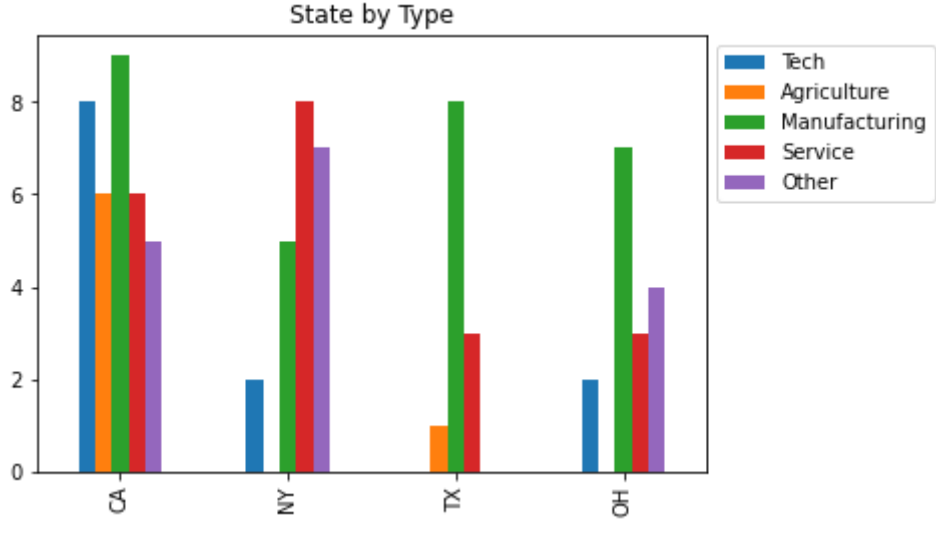
In [48]:
```
df2.T.plot.bar()
plt.title("State by Type")
```

Out[48]:
```
Text(0.5, 1.0, 'State by Type')
```



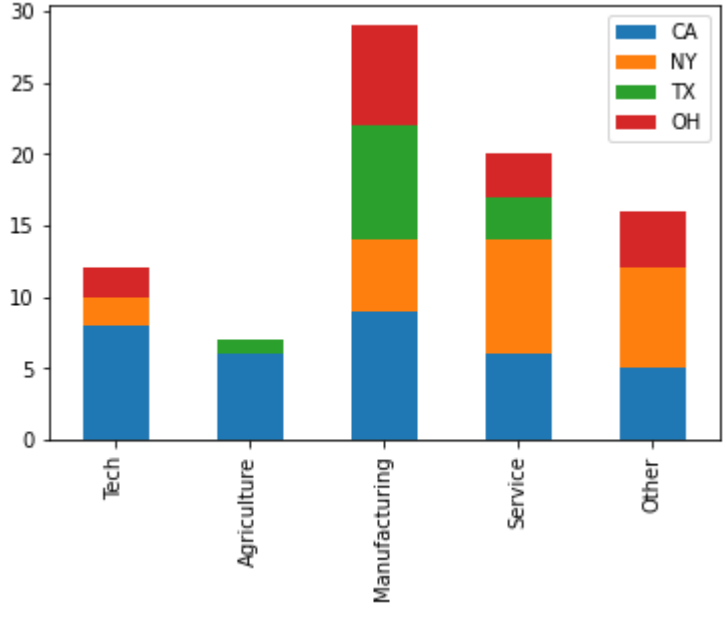
In [56]:

```
df2.T.plot.bar()
plt.title("State by Type")
plt.legend(bbox_to_anchor=(1.0, 1.0), loc='upper left') # positions the legend outside the box
plt.show()
```



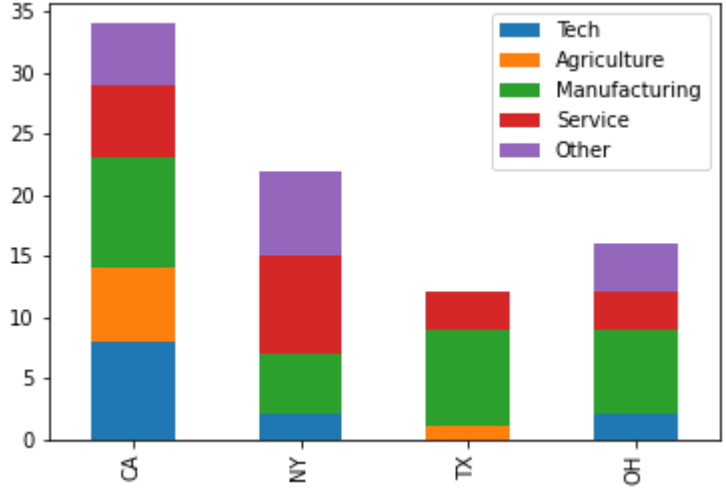
In [50]:
```
df2.plot.bar(stacked=True)
```

Out[50]: <AxesSubplot:>



In [51]:
```
df2.T.plot.bar(stacked=True)
```

Out[51]: <AxesSubplot:>



A data frame consisting of random die rolls.

Column 'one' shows the sum after one roll. (min is 1, max is 6, dist is uniform)

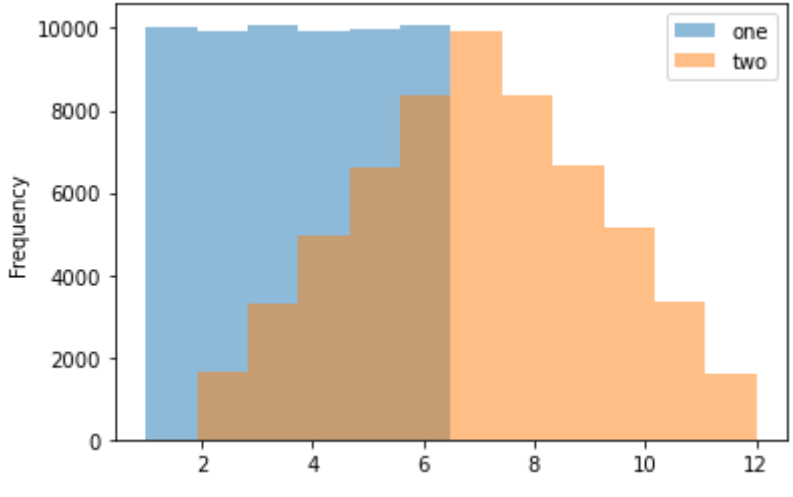
Column 'two' shows the sum after two rolls. (min is 2, max is 12, dist is triangular)

A data frame consisting of random die rolls.

Column 'one' shows the sum after one roll. (min is 1, max is 6, dist is uniform)

Column 'two' shows the sum after two rolls. (min is 2, max is 12, dist is triangular)

In [52]:
```
df = pd.DataFrame(np.random.randint(1, 7, 60000), columns = ['one'])
df['two'] = df['one'] + np.random.randint(1, 7, 60000)
ax = df.plot.hist(bins=12, alpha=0.5)
```



Column 'three' shows the sum after three rolls. (min is 3, max is 18)

And so on... as we add more dice rolls, we see the central limit theorem start to take effect.

Column 'three' shows the sum after three rolls. (min is 3, max is 18)

And so on... as we add more dice rolls, we see the central limit theorem start to take effect.

In [53]:

```
df['three'] = df['two'] + np.random.randint(1, 7, 60000)
df['four'] = df['three'] + np.random.randint(1, 7, 60000)
df['five'] = df['four'] + np.random.randint(1, 7, 60000)
```

Column 'three' shows the sum after three rolls. (min is 3, max is 18)

And so on... as we add more dice rolls, we see the central limit theorem start to take effect.

In [53]:
```
df['three'] = df['two'] + np.random.randint(1, 7, 60000)
df['four'] = df['three'] + np.random.randint(1, 7, 60000)
df['five'] = df['four'] + np.random.randint(1, 7, 60000)
```

In [54]:
```
df.plot.hist(bins = 30, alpha = 0.5)
```

Out[54]:
```
<AxesSubplot:ylabel='Frequency'>
```