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

In [2]:
```
df1 = pd.read_csv('06 df1.csv')
df2 = pd.read_csv('06 df2.csv')
```

In [3]:
```
df1["Unnamed: 0"] = pd.to_datetime(df1["Unnamed: 0"])
df1.set_index('Unnamed: 0', inplace=True)
```

In [4]:
```
df1
```

Out[4]:

	A	B	C	D
Unnamed: 0				
2000-01-01	1.339091	-0.163643	-0.646443	1.041233
2000-01-02	-0.774984	0.137034	-0.882716	-2.253382
2000-01-03	-0.921037	-0.482943	-0.417100	0.478638
2000-01-04	-1.738808	-0.072973	0.056517	0.015085
2000-01-05	-0.905980	1.778576	0.381918	0.291436
...	...	...	...	...
2002-09-22	1.013897	-0.288680	-0.342295	-0.638537
2002-09-23	-0.642659	-0.104725	-0.631829	-0.909483
2002-09-24	0.370136	0.233219	0.535897	-1.552605
2002-09-25	0.183339	1.285783	-1.052593	-2.565844
2002-09-26	0.775133	-0.850374	0.486728	-1.053427

1000 rows × 4 columns

In [5]:
```
df1.index
```

Out[5]:
```
DatetimeIndex(['2000-01-01', '2000-01-02', '2000-01-03', '2000-01-04',
               '2000-01-05', '2000-01-06', '2000-01-07', '2000-01-08',
               '2000-01-09', '2000-01-10',
               ...
               '2002-09-17', '2002-09-18', '2002-09-19', '2002-09-20',
               '2002-09-21', '2002-09-22', '2002-09-23', '2002-09-24',
               '2002-09-25', '2002-09-26'],
              dtype='datetime64[ns]', name='Unnamed: 0', length=1000, freq=None)
```

In [6]:
```
df2
```

Out[6]:

	a	b	c	d
0	0.039762	0.218517	0.103423	0.957904
1	0.937288	0.041567	0.899125	0.977680
2	0.780504	0.008948	0.557808	0.797510
3	0.672717	0.247870	0.264071	0.444358
4	0.053829	0.520124	0.552264	0.190008
5	0.286043	0.593465	0.907307	0.637898
6	0.430436	0.166230	0.469383	0.497701
7	0.312296	0.502823	0.806609	0.850519
8	0.187765	0.997075	0.895955	0.530390
9	0.908162	0.232726	0.414138	0.432007

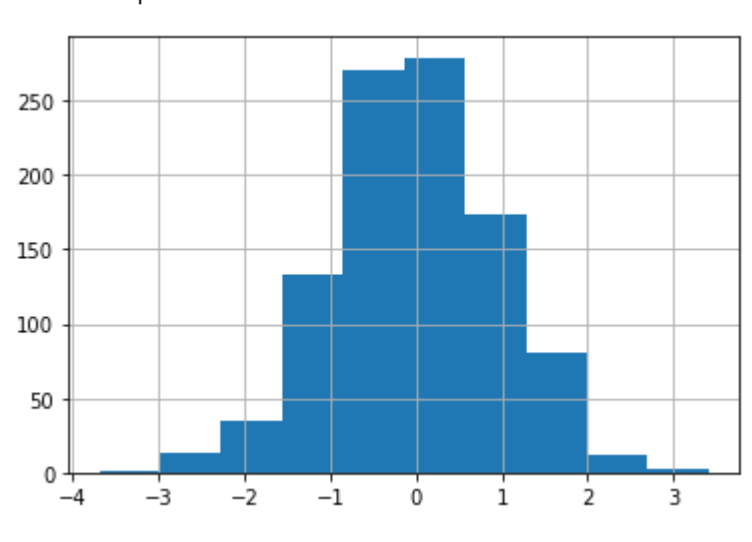
Style Sheets

Matplotlib has style sheets you can use to make your plots look a little nicer. These style sheets include plot_bmh,plot_fivethirtyeight,plot_ggplot and more. They basically create a set of style rules that your plots follow. I recommend using them, they make all your plots have the same look and feel more professional. You can even create your own if you want your company's plots to all have the same look (it is a bit tedious to create on though).

Here is how to use them.

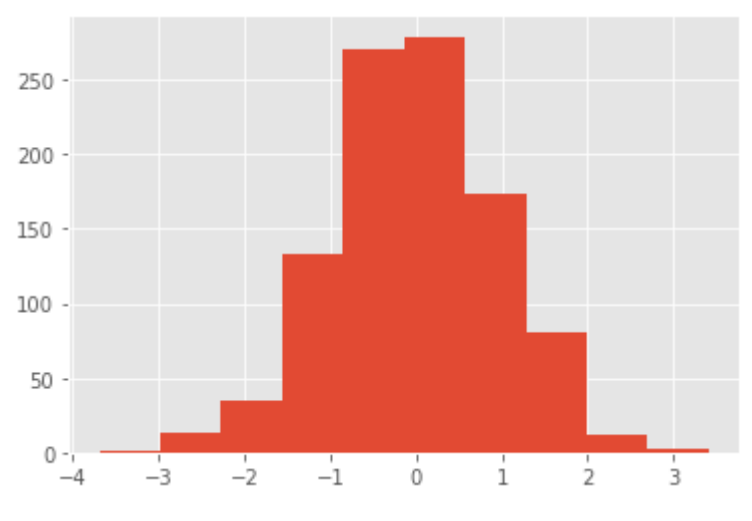
In [7]:
```
# Before plt.style.use() your plots look like this:
df1['A'].hist()
```

Out[7]: <AxesSubplot:>



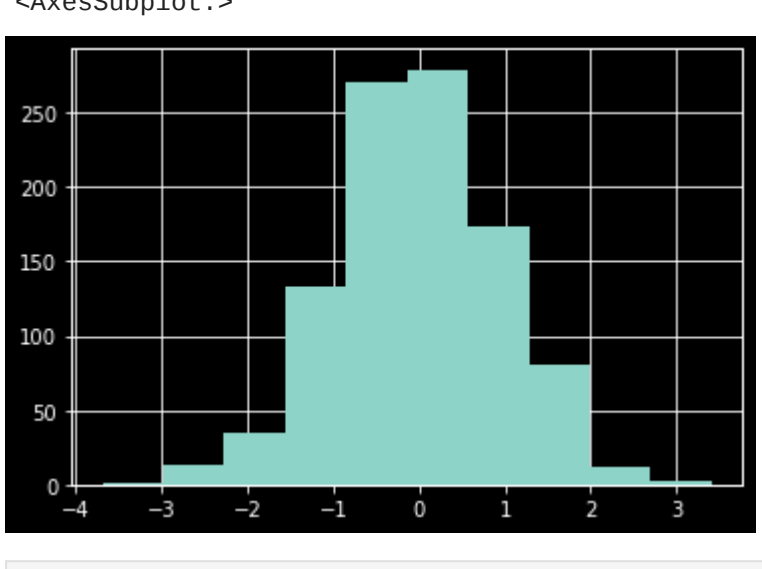
In [8]:
```
# After calling the stylesheet graph looks like this
plt.style.use('ggplot')
df1['A'].hist()
```

Out[8]: <AxesSubplot:>



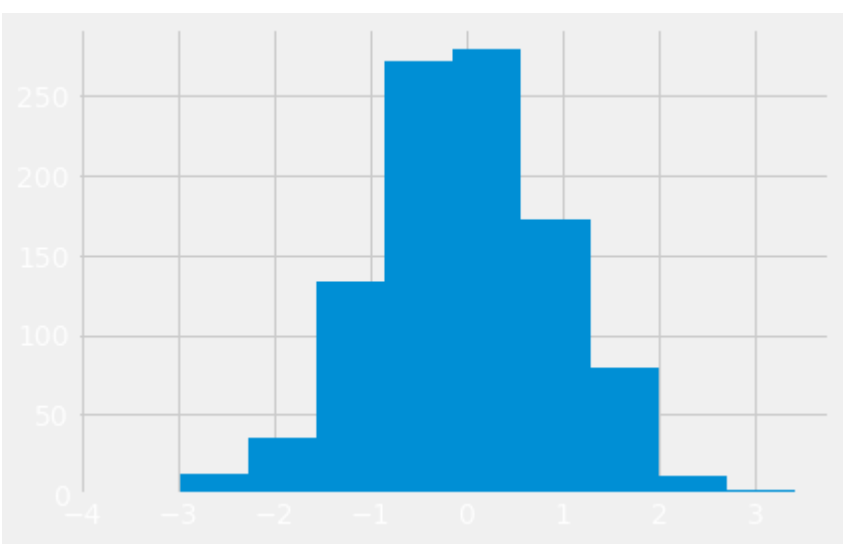
In [9]:
```
plt.style.use('dark_background')
df1['A'].hist()
```

Out[9]: <AxesSubplot:>



In [10]:
```
plt.style.use('fivethirtyeight')
df1['A'].hist()
```

Out[10]: <AxesSubplot:>



Plot Types

There are several plot types built-in to pandas, most of them statistical plots by nature:

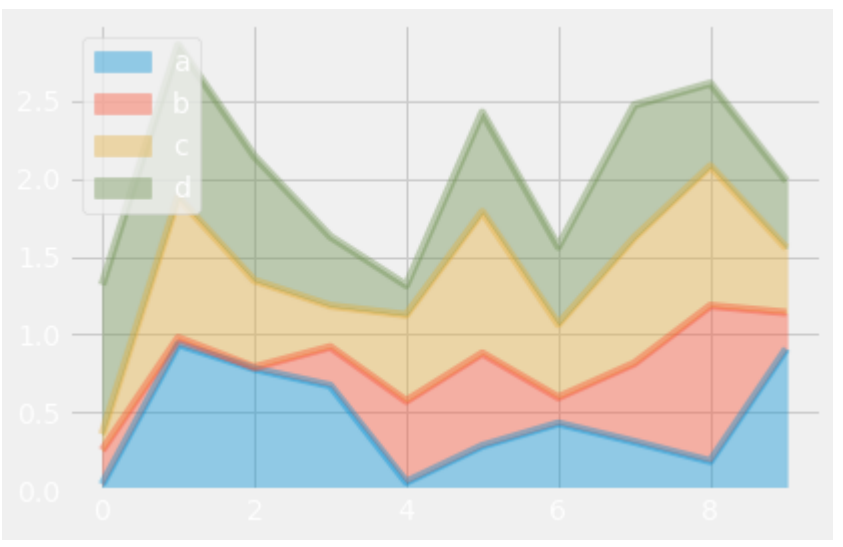
- df.plot.area
- df.plot.barh
- df.plot.density
- df.plot.hist
- df.plot.line
- df.plot.scatter
- df.plot.bar
- df.plot.box
- df.plot.hexbin
- df.plot.kde
- df.plot.pie

You can also just call **df.plot(kind='hist')** or replace that **kind argument** with any of the key terms shown in the list above (e.g. 'box','barh', etc..)

Area

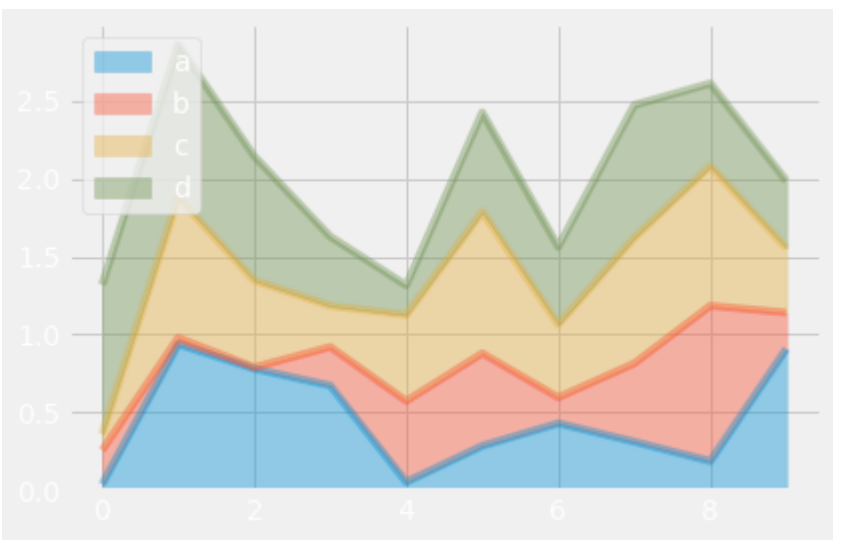
In [11]:
```
df2.plot.area(alpha=0.4)
```

Out[11]: <AxesSubplot:>



In [12]:
```
df2.plot(kind = "area", alpha=0.4)
```

Out[12]: <AxesSubplot:>



Barplots

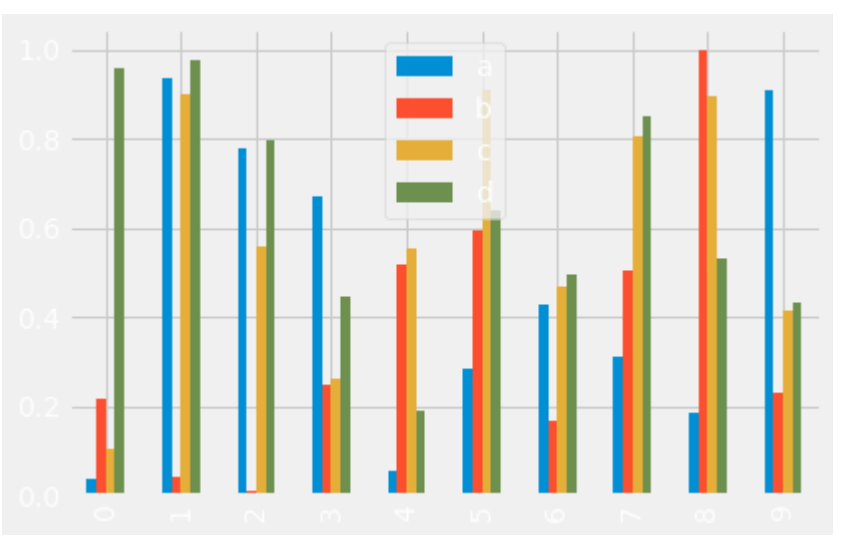
In [13]:
```
df2
```

Out[13]:

	a	b	c	d
0	0.039762	0.218517	0.103423	0.957904
1	0.937288	0.041567	0.899125	0.977680
2	0.780504	0.008948	0.557808	0.797510
3	0.672717	0.247870	0.264071	0.444358
4	0.053829	0.520124	0.552264	0.190008
5	0.286043	0.593465	0.907307	0.637898
6	0.430436	0.166230	0.469383	0.497701
7	0.312296	0.502823	0.806609	0.850519
8	0.187765	0.997075	0.895955	0.530390
9	0.908162	0.232726	0.414138	0.432007

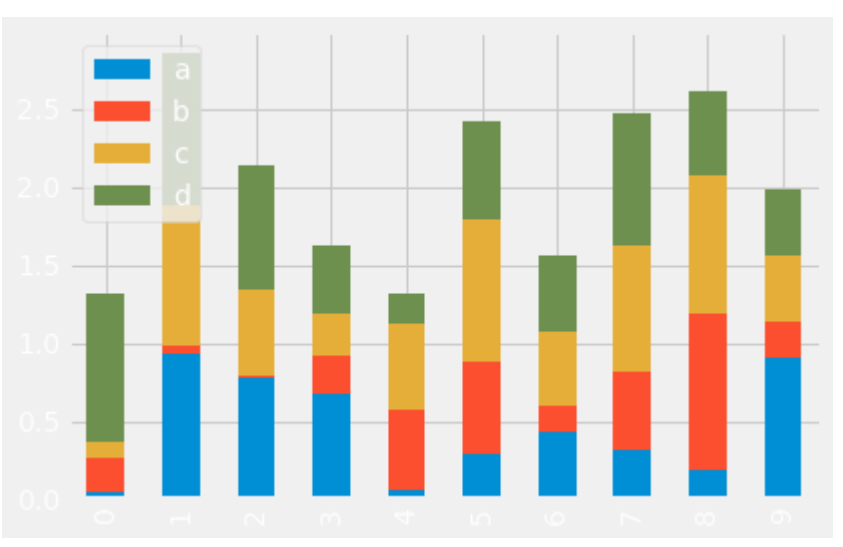
In [14]:
```
df2.plot.bar()
```

Out[14]: <AxesSubplot:>



In [15]:
```
df2.plot.bar(stacked=True)
```

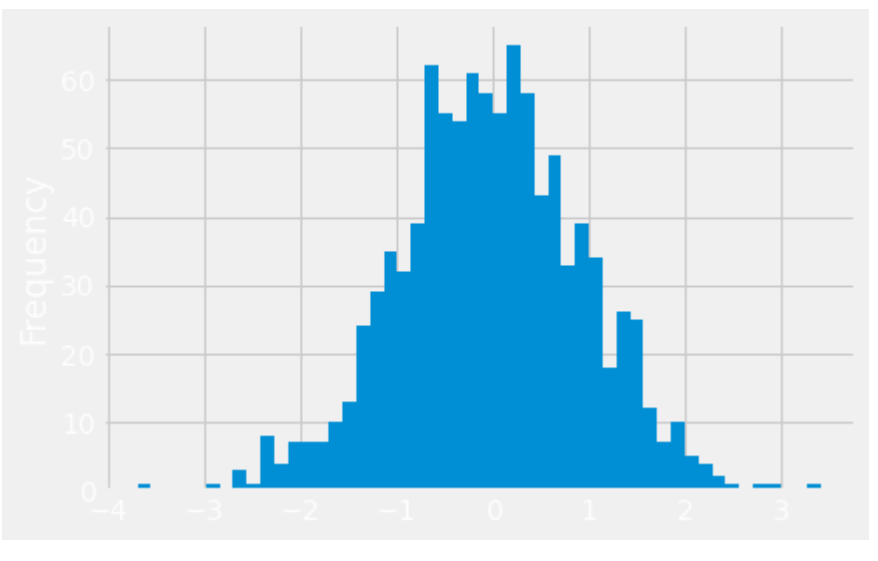
Out[15]: <AxesSubplot:>



Histograms

In [16]:
```
df1['A'].plot.hist(bins=50)
```

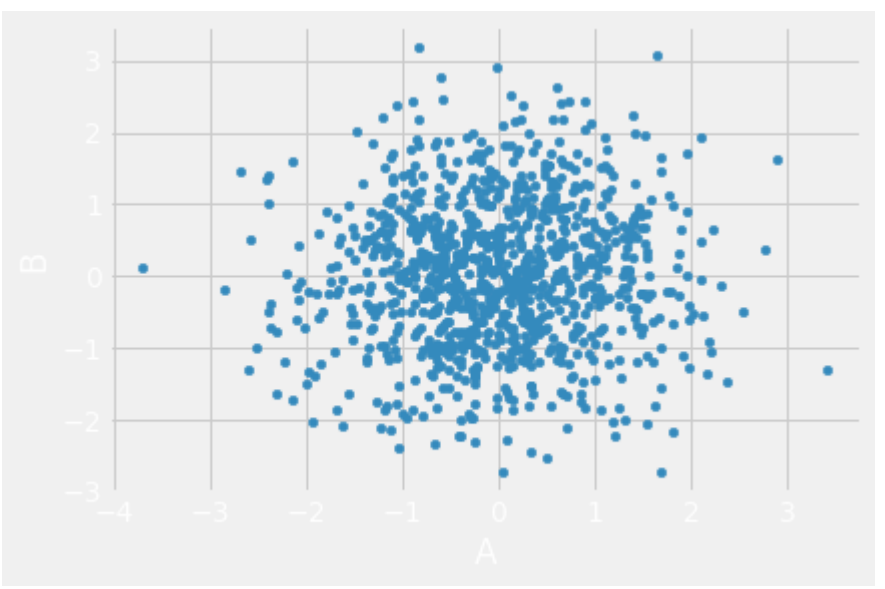
Out[16]: <AxesSubplot:ylabel='Frequency'>



Scatter Plots

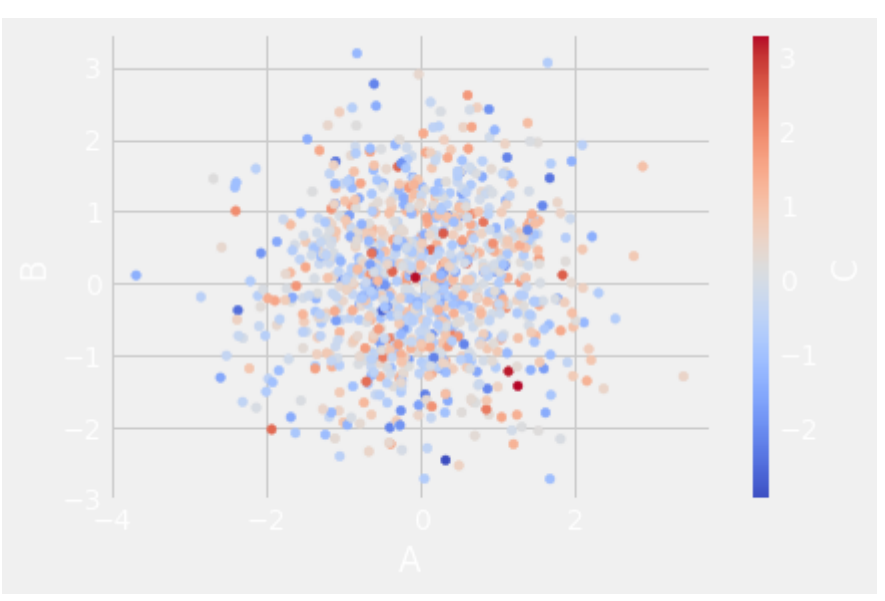
In [17]:
```
df1.plot.scatter(x='A',y='B')
```

Out[17]: <AxesSubplot:xlabel='A', ylabel='B'>



In [18]:
```
df1.plot.scatter(x='A',y='B',c='C',cmap='coolwarm')
```

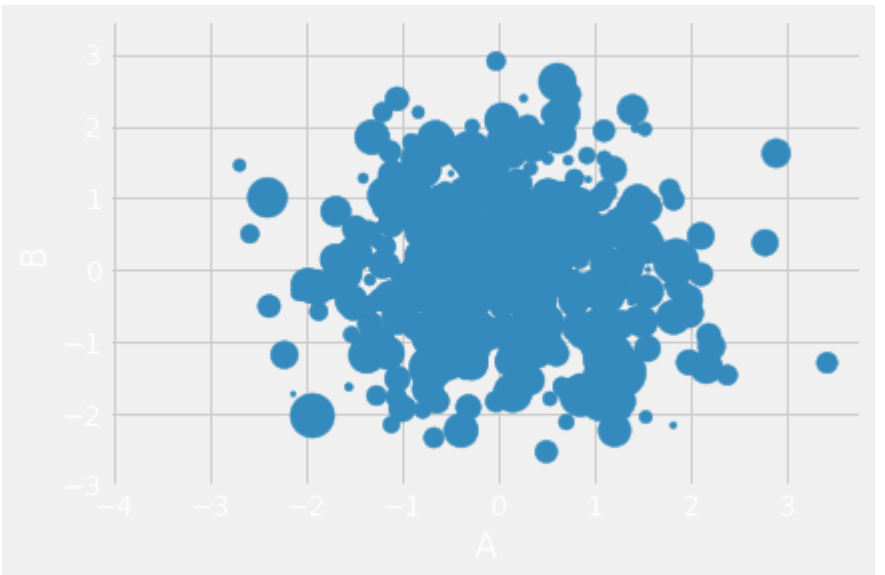
Out[18]: <AxesSubplot:xlabel='A', ylabel='B'>



In [19]:
```
# Or use s to indicate size based off another column. s parameter needs to be an array, not just the name of a column:
df1.plot.scatter(x='A',y='B',s=df1['C']*200)
```

```
C:\Users\Yasin\anaconda3\lib\site-packages\matplotlib\collections.py:922: RuntimeWarning: invalid value encountered in sqrt
  scale = np.sqrt(self._sizes) * dpi / 72.0 * self._factor
```

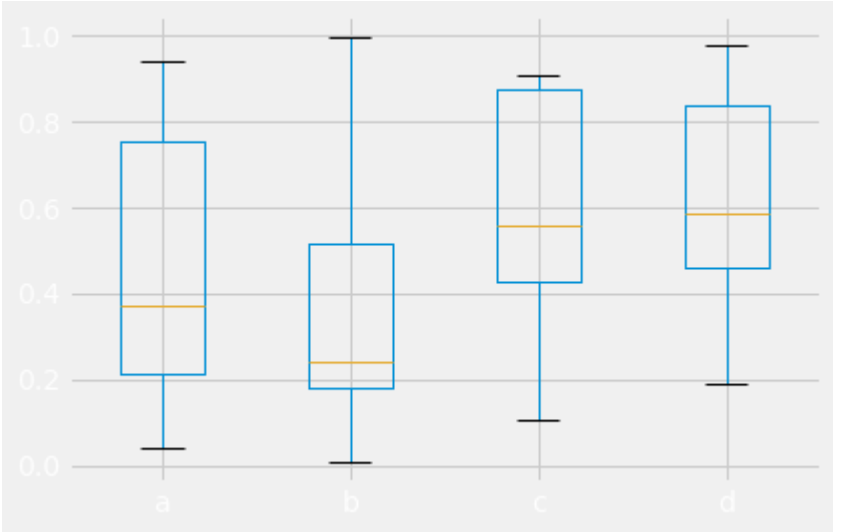
Out[19]: <AxesSubplot:xlabel='A', ylabel='B'>



BoxPlots

In [20]:
```
df2.plot.box() # Can also pass a by= argument for groupby
```

Out[20]: <AxesSubplot:>

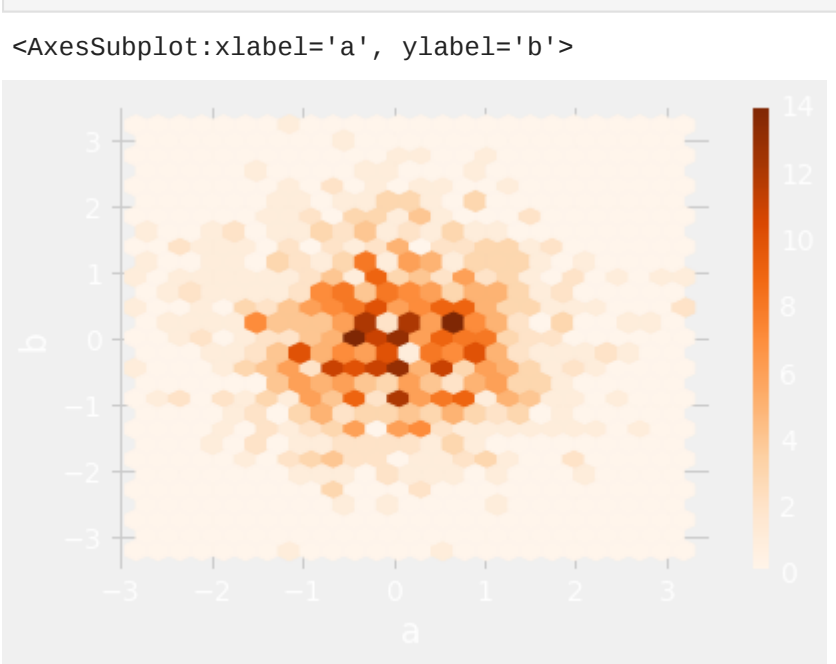


Hexagonal Bin Plot

Useful for Bivariate Data, alternative to scatterplot:

In [21]:
```
df = pd.DataFrame(np.random.randn(1000, 2), columns=['a', 'b'])
df.plot.hexbin(x='a',y='b',gridsize=25,cmap='Oranges')
```

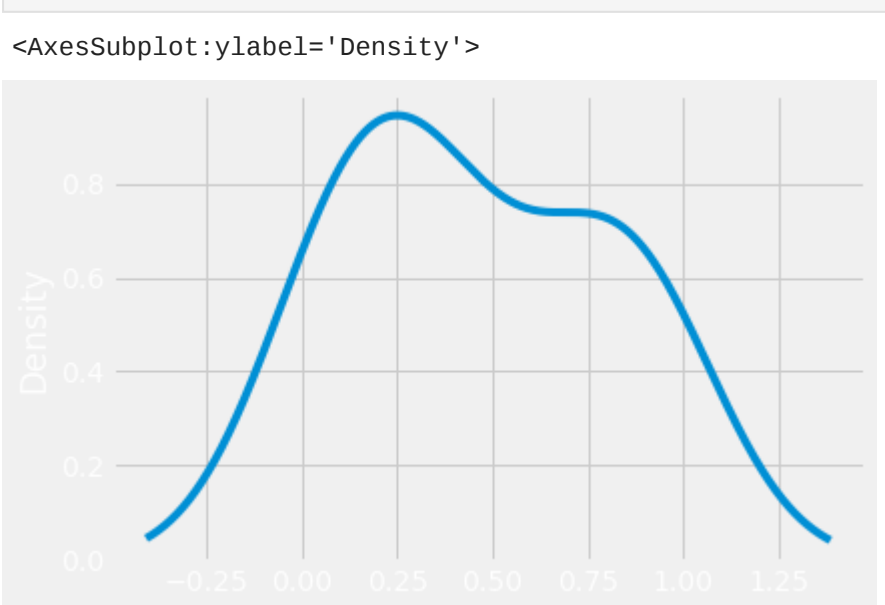
Out[21]: <AxesSubplot:xlabel='a', ylabel='b'>



Kernel Density Estimation plot (KDE)

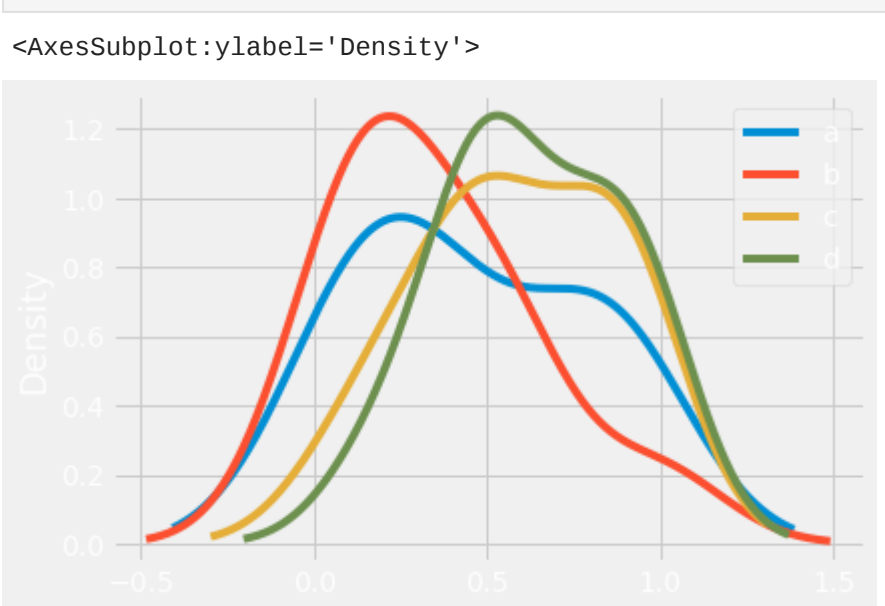
In [22]:
```
df2['a'].plot.kde()
```

Out[22]: <AxesSubplot:ylabel='Density'>



In [23]:
```
df2.plot.density() # the mean of all these distribution is KDE
```

Out[23]: <AxesSubplot:ylabel='Density'>



In [ ]: