Pandas Built-in Data Visualization

In this lecture we will learn about pandas built-in capabilities for data visualization! It's built-off of matplotlib, but it baked into pandas for easier usage!

Let's take a look!

Imports

In [18]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

The Data

There are some fake data csv files you can read in as dataframes:

In [19]:
```
df1 = pd.read_csv('df1',index_col=0)
df2 = pd.read_csv('df2')
```

In [20]:
```
df2
```

Out[20]:

	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

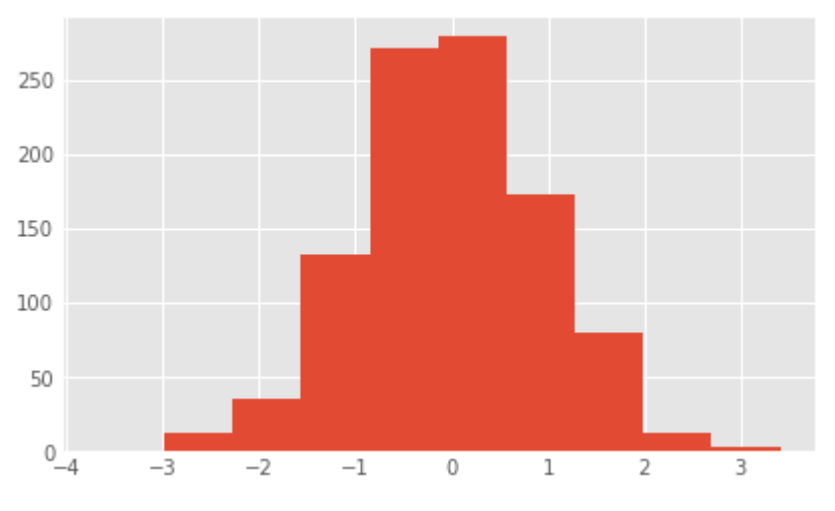
Matplotlib has style sheets you can use to make your plots look a little nicer. These style sheets include plot_bmh,plot_fivethirtyeight,plot_ggplot and more. They basically create a set of style rules that your plots follow. I recommend using them, they make all your plots have the same look and feel 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.

Before plt.style.use() your plots look like this:

In [21]:
```
df1['A'].hist()
```

Out[21]: <AxesSubplot:>



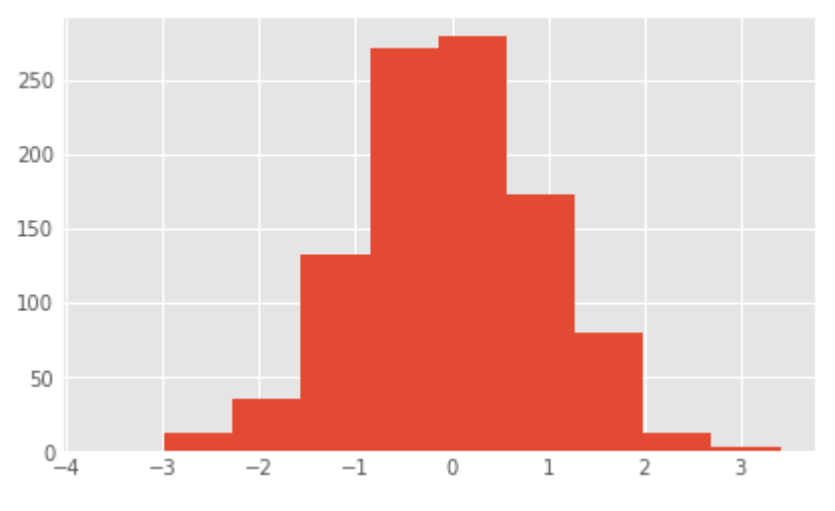
Call the style:

In [22]:
```
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

Now your plots look like this:

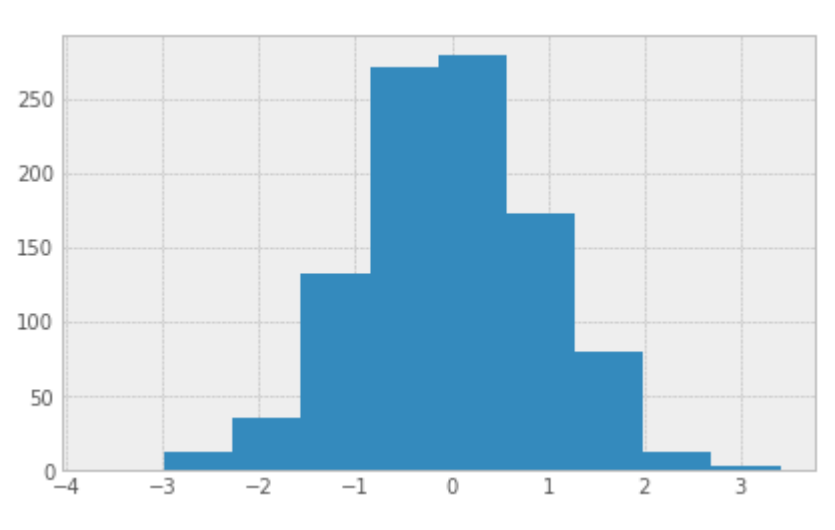
In [23]:
```
df1['A'].hist()
```

Out[23]: <AxesSubplot:>



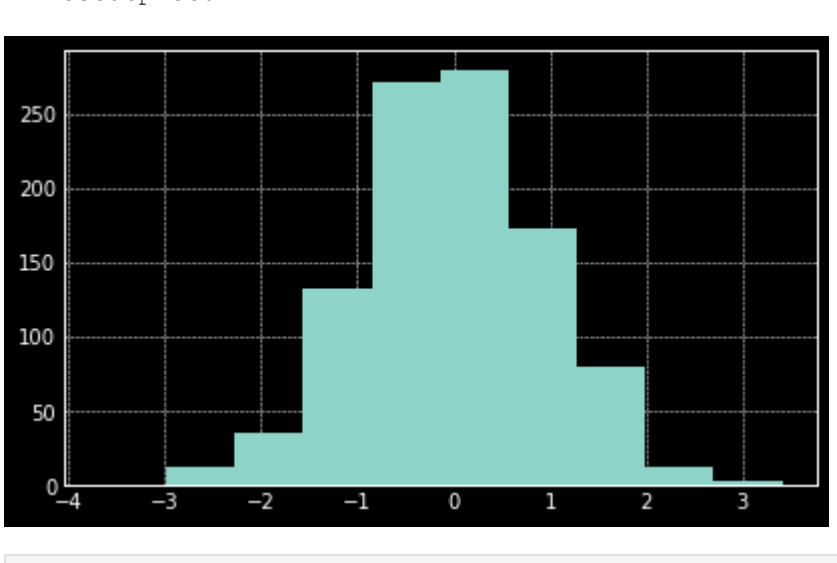
In [24]:
```
plt.style.use('bmh')
df1['A'].hist()
```

Out[24]: <AxesSubplot:>



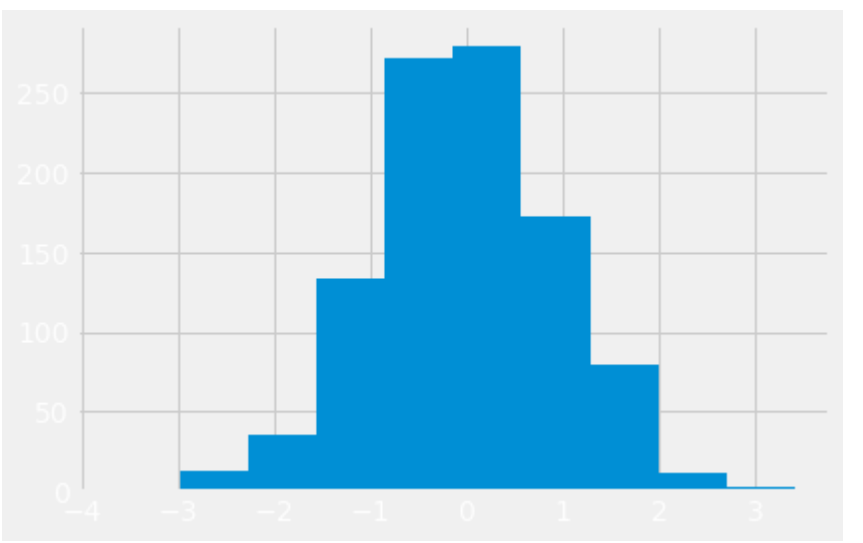
In [25]:
```
plt.style.use('dark_background')
df1['A'].hist()
```

Out[25]: <AxesSubplot:>



In [26]:
```
plt.style.use('fivethirtyeight')
df1['A'].hist()
```

Out[26]: <AxesSubplot:>



In [27]:
```
plt.style.use('ggplot')
```

Let's stick with the ggplot style and actually show you how to utilize pandas built-in plotting capabilities!

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

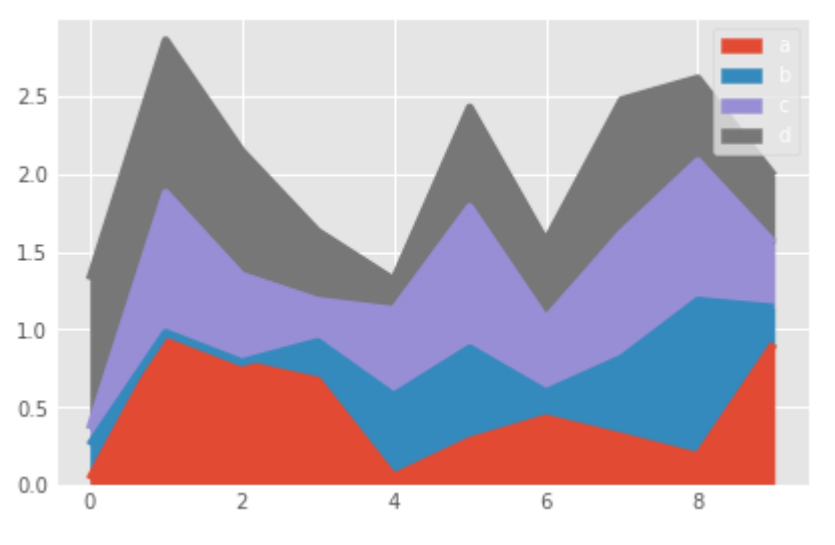
You can also just call df.plot(kind='hist') or replace that kind argument with any of the key terms shown in the list above (e.g. 'box','barh', etc..)

Let's start going through them!

Area

In [28]:
```
df2.plot.area(alpha=1)
```

Out[28]: <AxesSubplot:>



Barplots

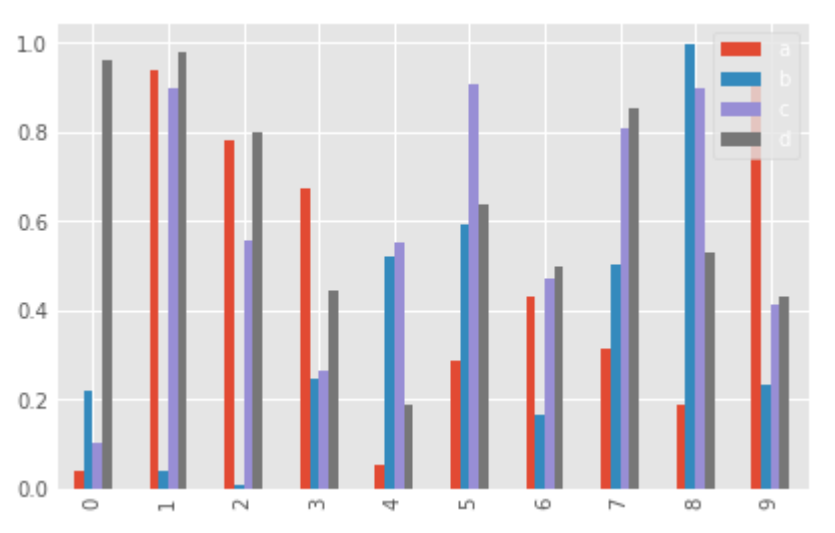
In [29]:
```
df2.head()
```

Out[29]:

	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

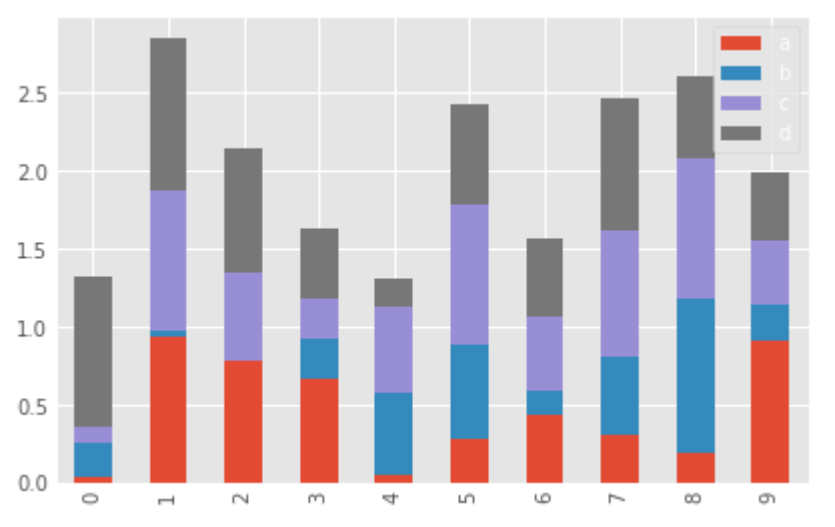
In [30]:
```
df2.plot.bar()
```

Out[30]: <AxesSubplot:>



In [31]:
```
df2.plot.bar(stacked=True)
```

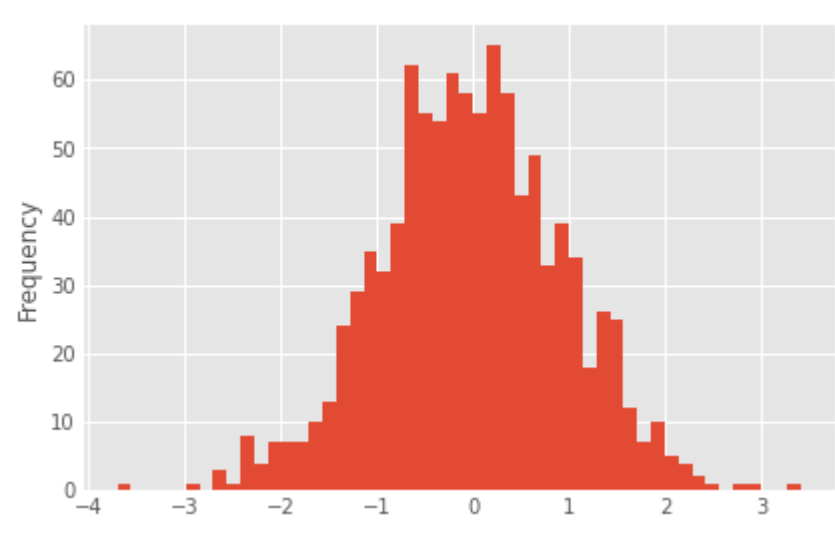
Out[31]: <AxesSubplot:>



Histograms

In [32]:
```
df1['A'].plot.hist(bins=50)
```

Out[32]: <AxesSubplot:ylabel='Frequency'>



Line Plots

In [33]:
```
df2.plot.line(x=df2.index,y='B',figsize=(12,3),lw=1)
```

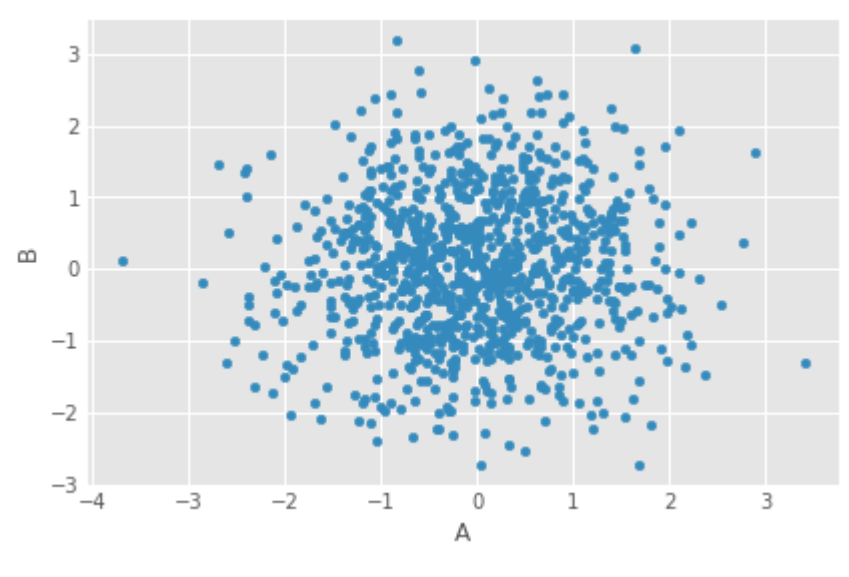
```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [33], in <cell line: 1>()
----> 1 df2.plot.line(x=df2.index,y='B',figsize=(12,3),lw=1)

File ~\anaconda3\lib\site-packages\pandas\plotting\_core.py:1041, in PlotAccessor.line(self, x, y, **kwargs)
    976 @Appender(
    977     """
    978     See Also
   (...)
   1033 @Appender(_bar_or_line_doc)
   1034 def line(self, x=None, y=None, **kwargs):
   1035     """
   1036     Plot Series or DataFrame as lines.
   1037 
   1038     This function is useful to plot lines using DataFrame's values
   1039     as coordinates.
   1040     """
-> 1041     return self(kind="line", x=x, y=y, **kwargs)

File ~\anaconda3\lib\site-packages\pandas\plotting\_core.py:937, in PlotAccessor.__call__(self, *args, **kwargs)
    935 if is_integer(x) and not data.columns.holds_integer():
    936     x = data_cols[x]
--> 937 elif not isinstance(data[x], ABCSeries):
    938     raise ValueError("x must be a label or position")
    939 data = data.set_index(x)

File ~\anaconda3\lib\site-packages\pandas\core\frame.py:3511, in DataFrame.__getitem__(self, key)
   3509     if is_iterator(key):
   3510         key = list(key)
-> 3511     indexer = self.columns._get_indexer_strict(key, "columns")[1]
   3513 # take() does not accept boolean indexers
   3514 if getattr(indexer, "dtype", None) == bool:

File ~\anaconda3\lib\site-packages\pandas\core\indexes\base.py:5782, in Index._get_indexer_strict(self, key, axis_name)
   5779 else:
   5780     keyarr, indexer, new_indexer = self._reindex_non_unique(keyarr)
-> 5782 self._raise_if_missing(keyarr, indexer, axis_name)
   5784 keyarr = self.take(indexer)
   5785 if isinstance(key, Index):
   5786     # GH 42790 - Preserve name from an Index

File ~\anaconda3\lib\site-packages\pandas\core\indexes\base.py:5842, in Index._raise_if_missing(self, key, indexer, axis_name)
   5840 if use_interval_msg:
   5841     key = list(key)
-> 5842 raise KeyError(f"None of [{key}] are in the [{axis_name}]")
   5844 not_found = list(ensure_index(key)[missing_mask.nonzero()[0]].unique())
   5845 raise KeyError(f"{not_found} not in index")

KeyError: 'None of [RangeIndex(start=0, stop=10, step=1)] are in the [columns]'
```

In [ ]:
```
df1.plot.line()
```

Scatter Plots

In [34]:
```
df1.plot.scatter(x='A',y='B')
```

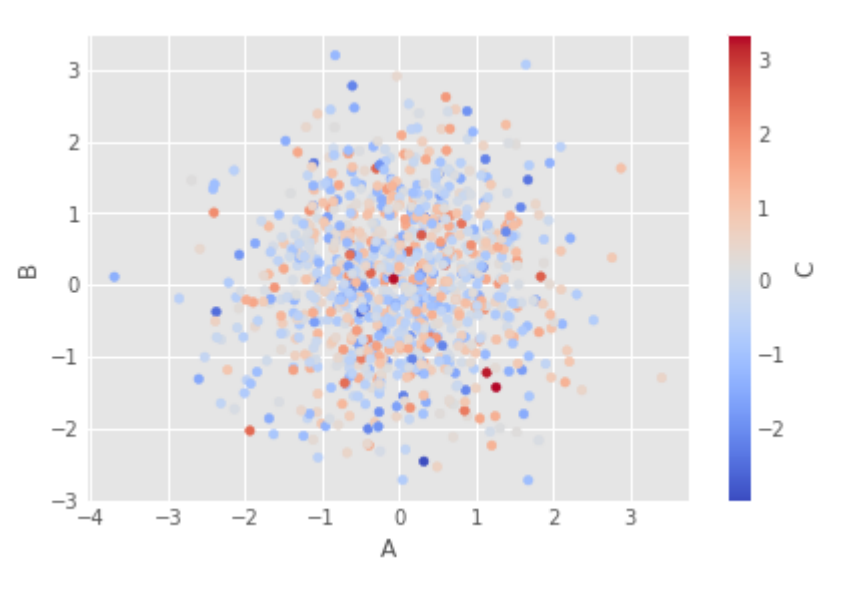
Out[34]: <AxesSubplot:xlabel='A', ylabel='B'>



You can use c to color based off another column value Use cmap to indicate colormap to use. For all the colormaps, check out: http://matplotlib.org/users/colormaps.html

In [35]:
```
df1.plot.scatter(x='A',y='B',c='C',cmap='coolwarm')
```

Out[35]: <AxesSubplot:xlabel='A', ylabel='B'>

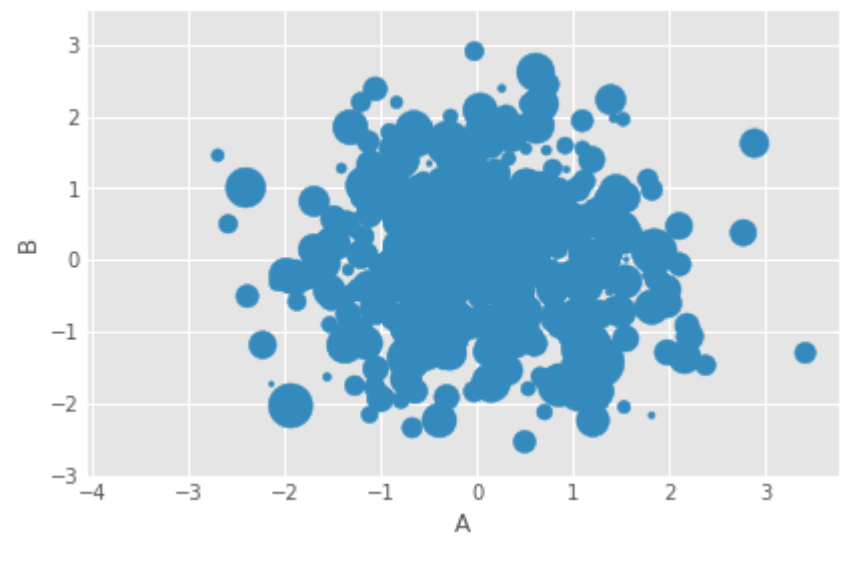


Or use s to indicate size based off another column. s parameter needs to be an array, not just the name of a column:

In [38]:
```
df1.plot.scatter(x='A',y='B',s=df1['C']*200)
```

```
C:\Users\Shubham kumawat\anaconda3\lib\site-packages\matplotlib\collections.py:982: RuntimeWarning: invalid value encountered in sqrt
  scale = np.sqrt(self._sizes) * dpi / 72.0 * self._factor
```

Out[38]: <AxesSubplot:xlabel='A', ylabel='B'>



BoxPlots

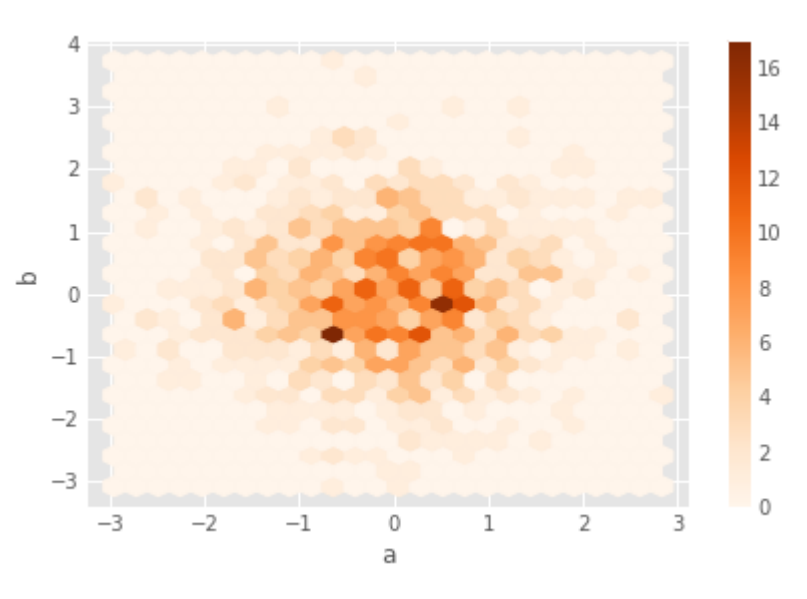
In [ ]:
```
df2.plot.box() # Can also pass a by= argument for groupby
```

Hexagonal Bin Plot

Useful for Bivariate Data, alternative to scatterplot:

In [39]:
```
df = pd.DataFrame(np.random.randn(1000, 2), columns=['a', 'b'])
df.plot.hexbin(x='a',y='b',gridsize=25,cmap='Oranges')
```

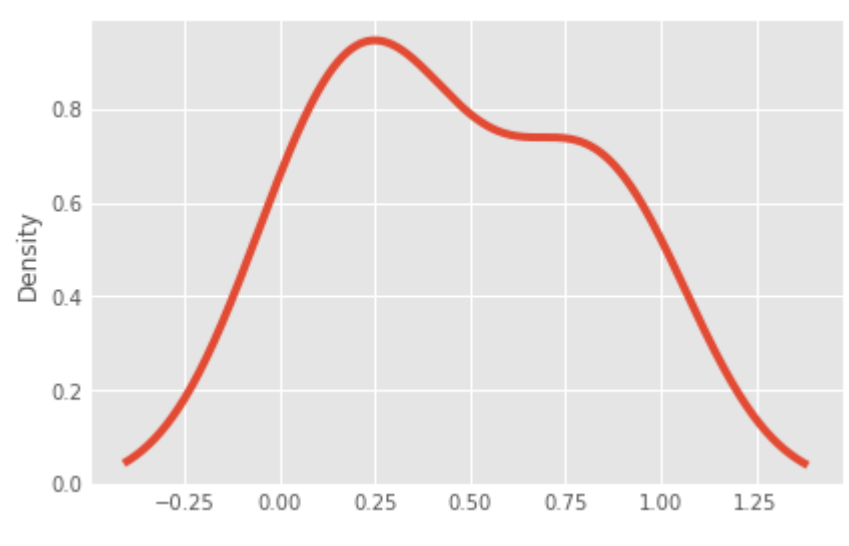
Out[39]: <AxesSubplot:xlabel='a', ylabel='b'>



Kernel Density Estimation plot (KDE)

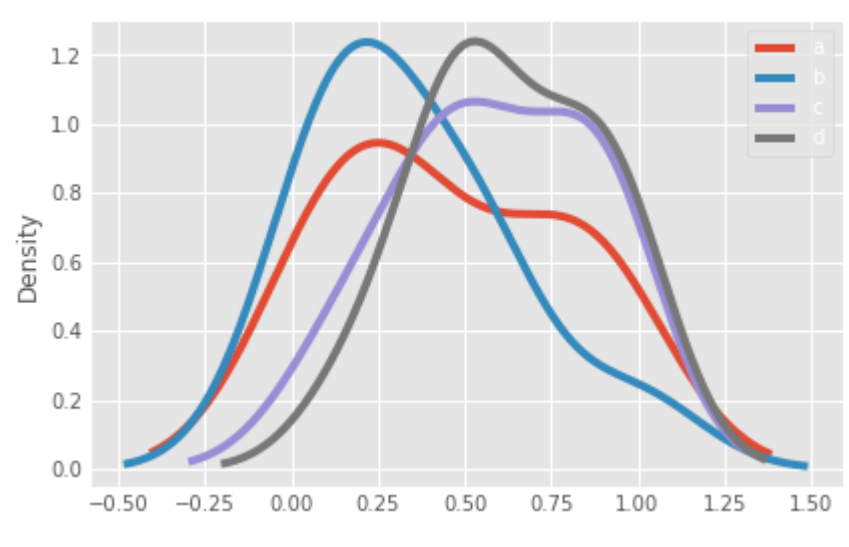
In [36]:
```
df2['a'].plot.kde()
```

Out[36]: <AxesSubplot:ylabel='Density'>



In [37]:
```
df2.plot.density()
```

Out[37]: <AxesSubplot:ylabel='Density'>



That's it! Hopefully you can see why this method of plotting will be a lot easier to use than full-on matplotlib, it balances ease of use with control over the figure. A lot of the plot calls also accept additional arguments of their parent matplotlib plt. call.

Great Job!