00-Pandas Built-in Data Visualization

October 19, 2022

---

1 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's baked into pandas for easier usage!

For more information on the following topics visit https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html For a nice graphic showing the anatomy of a figure with terminology, visit https://matplotlib.org/faq/usage_faq.html

Let's dive in!

1.1 Imports

```
[1]: import numpy as np
     import pandas as pd
     %matplotlib inline
```

NOTE: %matplotlib inline is specific to jupyter notebooks. It allows plots to appear inside the notebook.Note that we are not importing matplotlib here, all of our plots are coming out of pandas.

1.2 The Data

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

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

2 Plot Types

There are several plot types built into pandas; most of them are statistical by nature:

You can also call specific plots by passing their name as an argument, as with `df.plot(kind='area')`.

Let's start going through them! First we'll look at the data:

[3]:
```
df1.head()
```

[3]:
```
                   A         B         C         D
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
```

[4]:
```
df2.head()
```

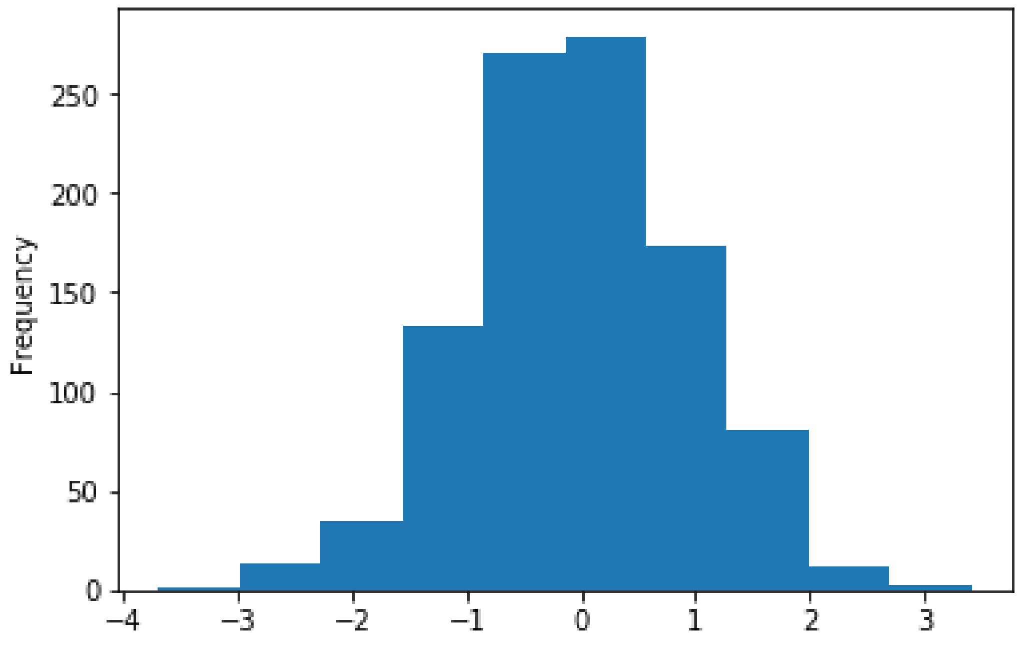
[4]:
```
          a         b         c         d  e
0  0.039762  0.218517  0.103423  0.957904  x
1  0.937288  0.041567  0.899125  0.977680  y
2  0.780504  0.008948  0.557808  0.797510  x
3  0.672717  0.247870  0.264071  0.444358  z
4  0.053829  0.520124  0.552264  0.190008  y
```

2.1 Histograms

This is one of the most commonly used plots. Histograms describe the distribution of continuous data by dividing the data into "bins" of equal width, and plotting the number of values that fall into each bin. [reference]

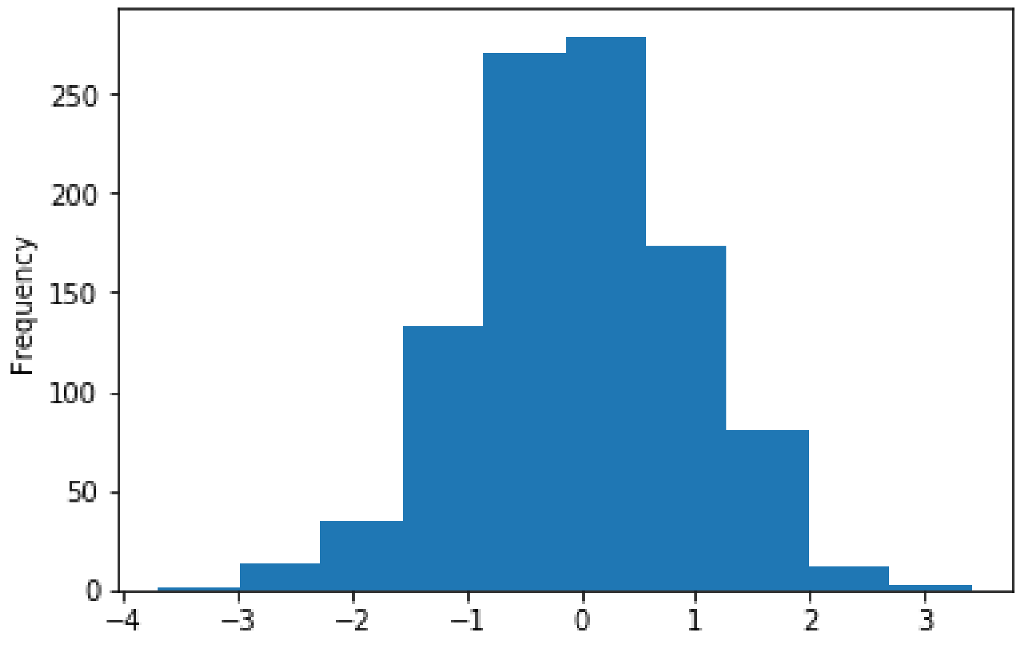
[5]:
```
df1['A'].plot.hist()
```

[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x199f7379c50>
```



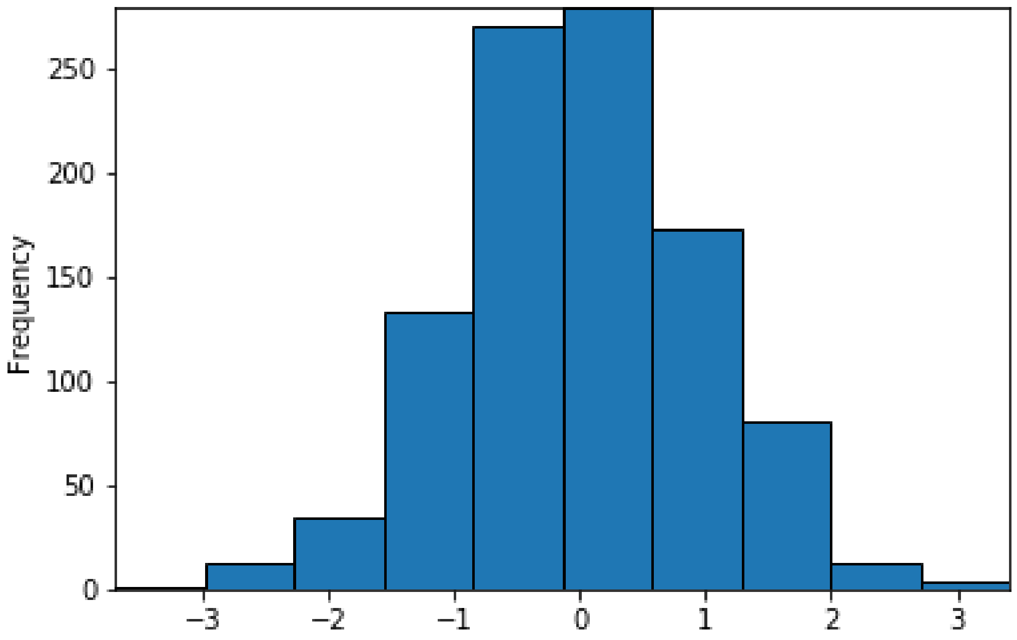
NOTE: To avoid seeing Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x2640e47af60>in jupyter you can add a semicolon to the end of the plot line.

```
[6]: df1['A'].plot.hist();
```



We can add settings to do things like bring the x- and y-axis values to the edge of the graph, and insert lines between vertical bins:

```
[7]: df1['A'].plot.hist(edgecolor='k').autoscale(enable=True, axis='both',␣
     ↪tight=True)
```

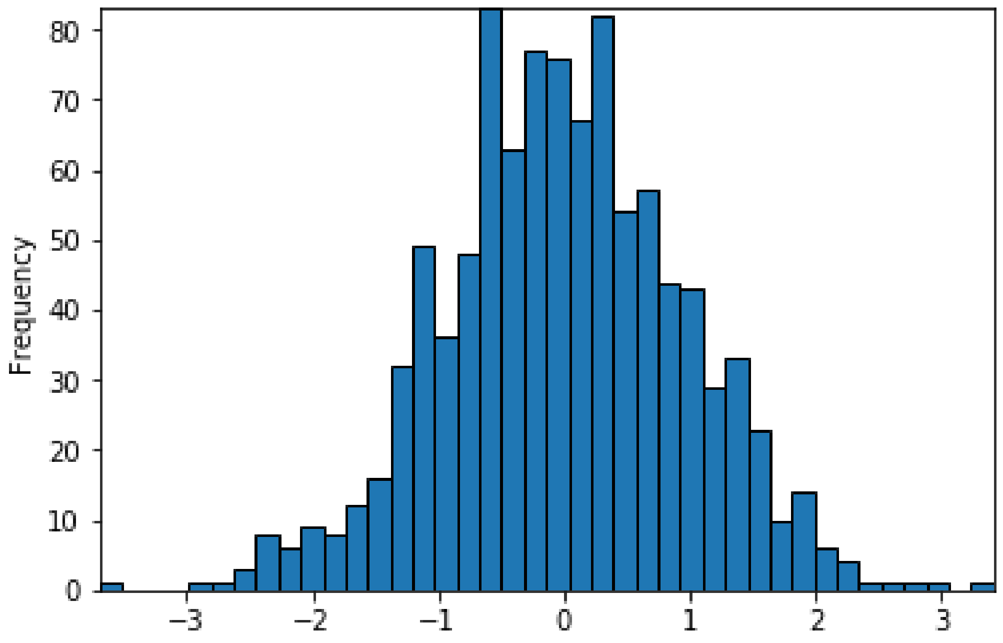


You can use any matplotlib color spec for **edgecolor**, such as `'b'`, `'g'`, `'r'`, `'c'`, `'m'`, `'y'`, `'k'`, `'w'`, or the string representation of a float value for shades of grey, such as `'0.5'`

For **autoscale** the axis can be set to `'x'`, `'y'` or `'both'`

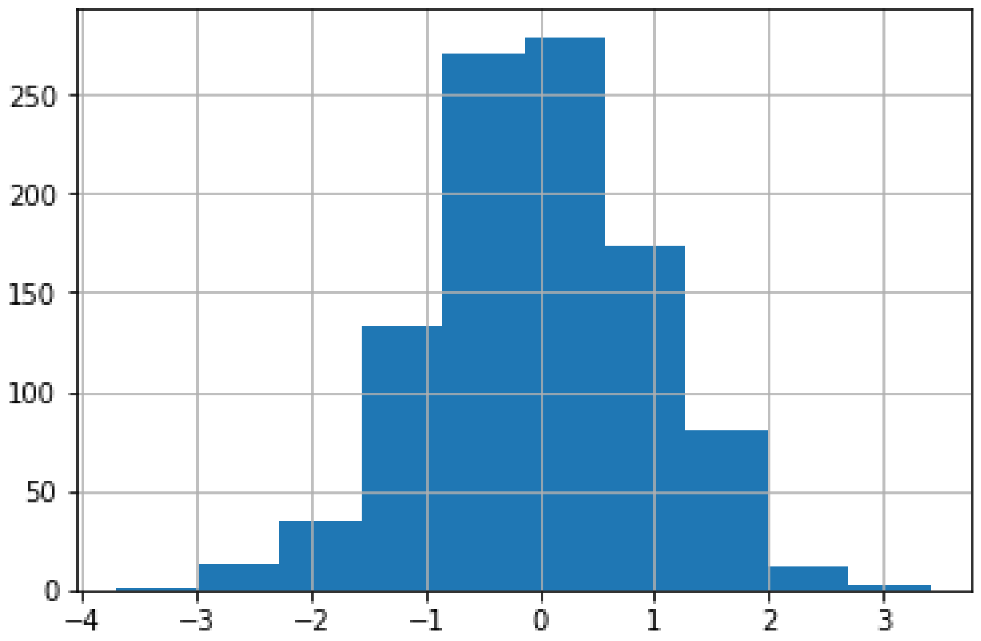
We can also change the number of bins (the range of values over which frequencies are calculated) from the default value of 10:

```
[8]: df1['A'].plot.hist(bins=40, edgecolor='k').autoscale(enable=True, axis='both',␣
     ↪tight=True)
```

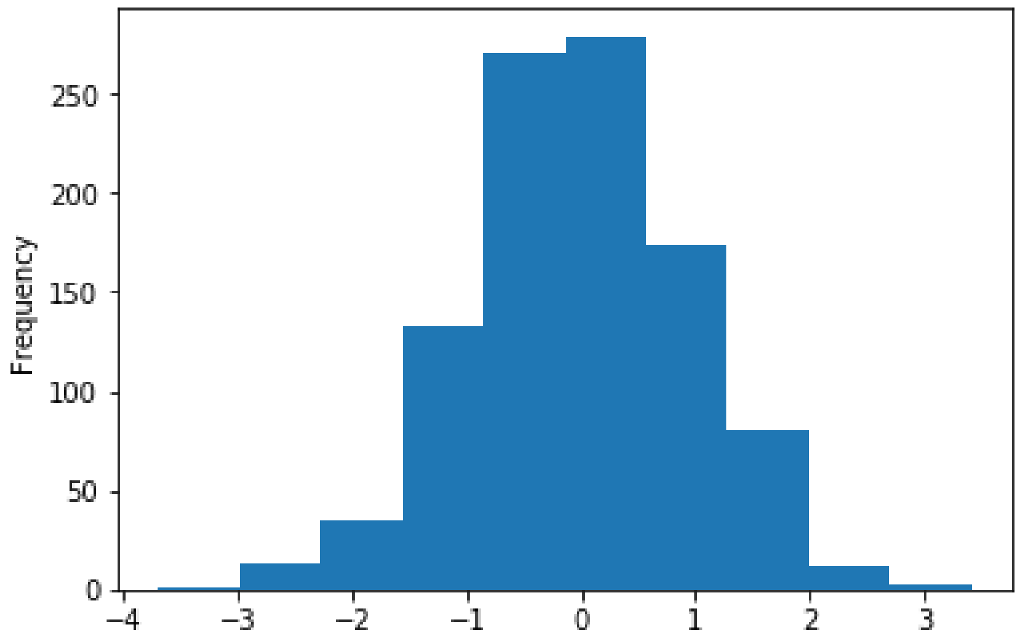


NOTE: Histograms are also accessible with df.hist(), but some changes are made to the default formatting (dropped y-axis label, addition of gridlines)

[9]:
```
df1['A'].hist();
```



[10]:
```
df1['A'].hist(grid=False).set_ylabel("Frequency");
```

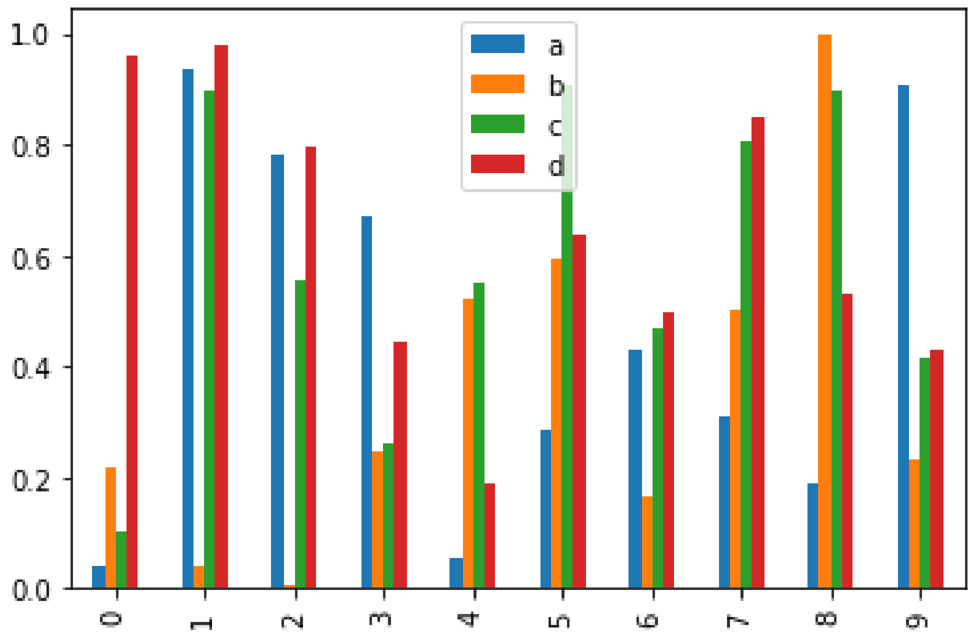


For more on using df.hist() visit https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.hist.html

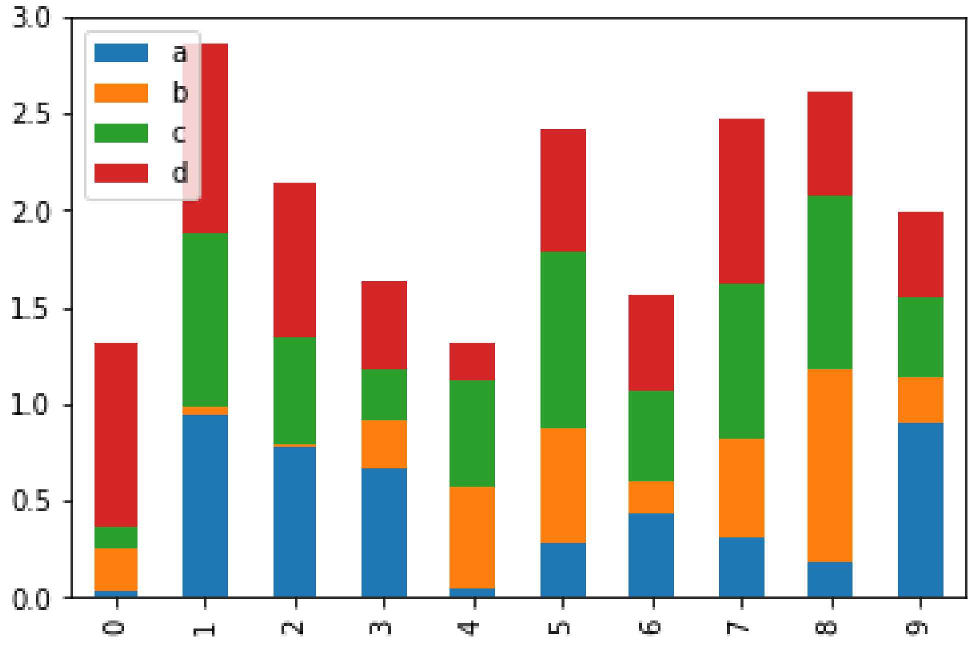
2.2 Barplots

Barplots are similar to histograms, except that they deal with discrete data, and often reflect multiple variables. [reference]

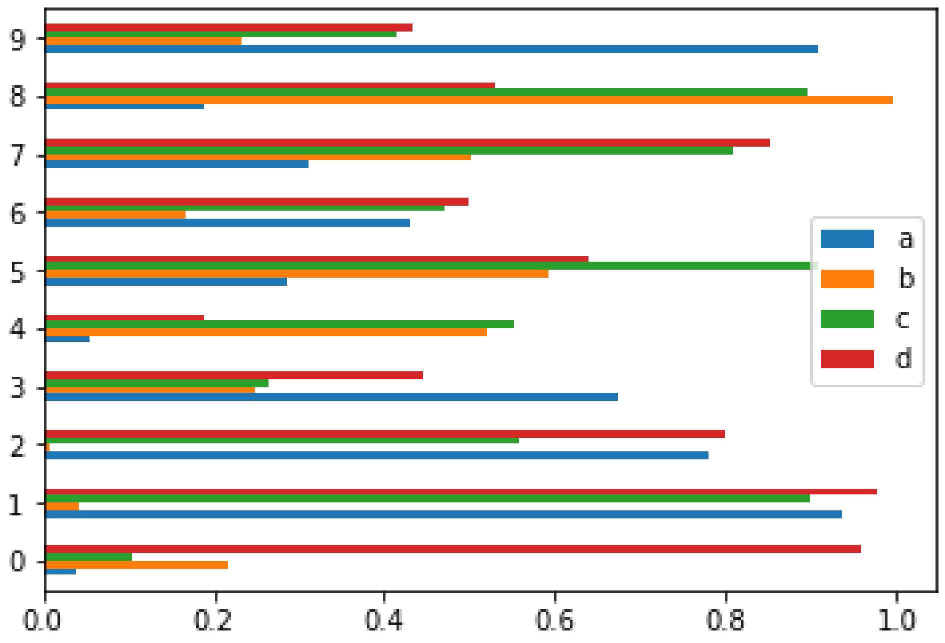
[11]:
```
df2.plot.bar();
```



```
[12]: df2.plot.bar(stacked=True);
```



```
[13]: # USE .barh() TO DISPLAY A HORIZONTAL BAR PLOT
      df2.plot.barh();
```

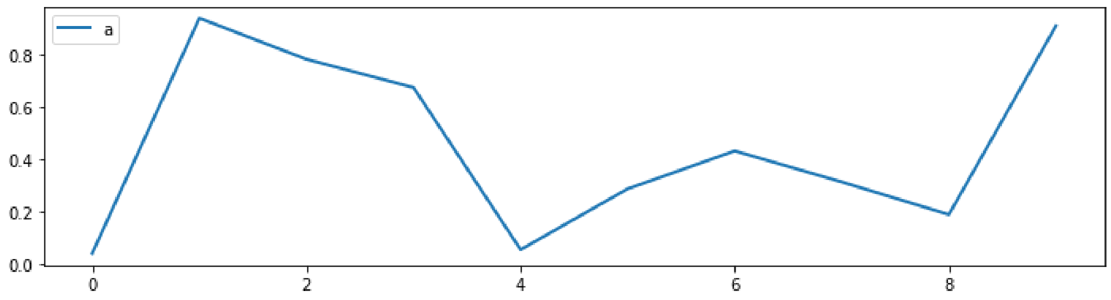


2.3 Line Plots

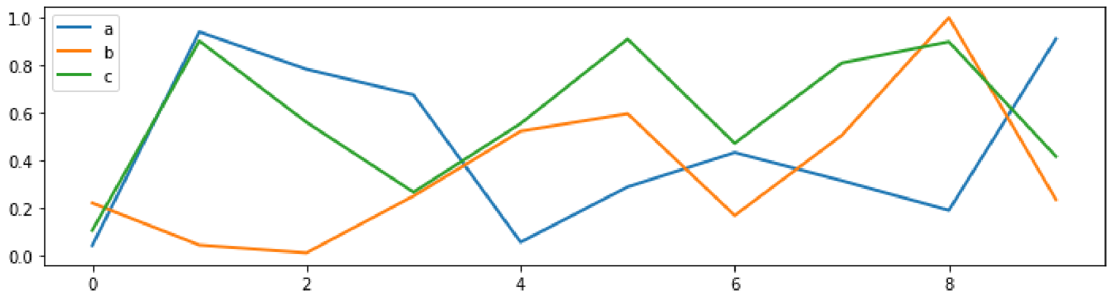
Line plots are used to compare two or more variables. By default the x-axis values are taken from the index. [reference]

Line plots happen to be the default pandas plot. They are accessible through df.plot() as well as df.plot.line()

[14]:
```
df2.plot.line(y='a',figsize=(12,3),lw=2);
```



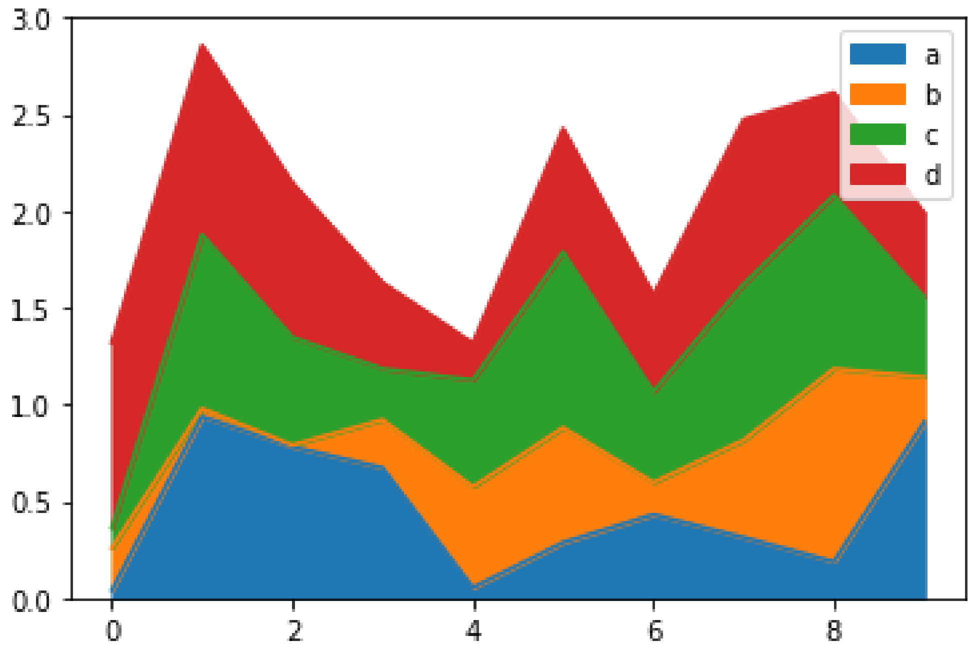
[15]:
```
df2.plot.line(y=['a','b','c'],figsize=(12,3),lw=2);
```



2.4 Area Plots

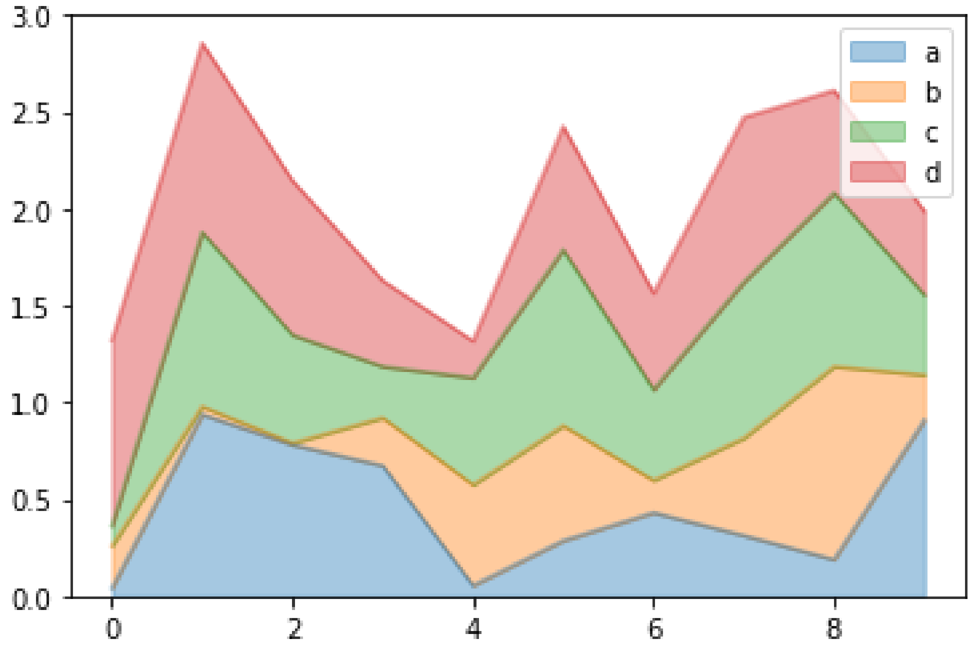
Area plots represent cumulatively stacked line plots where the space between lines is emphasized with colors. [reference]

```
[16]: df2.plot.area();
```



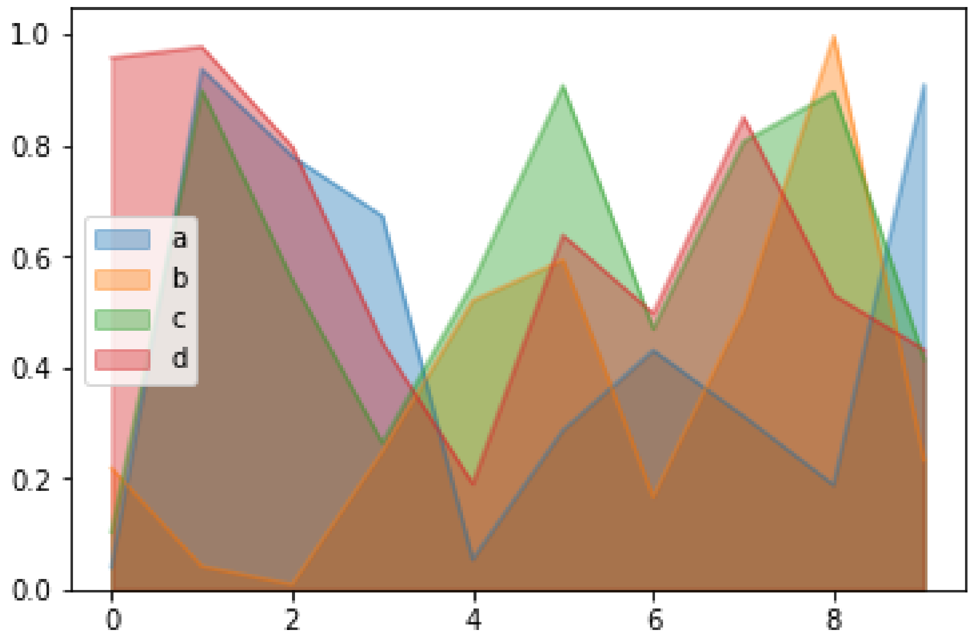
It often helps to mute the colors by passing an alpha transparency value between 0 and 1.

```
[17]: df2.plot.area(alpha=0.4);
```



To produce a blended area plot, pass a stacked=False argument:

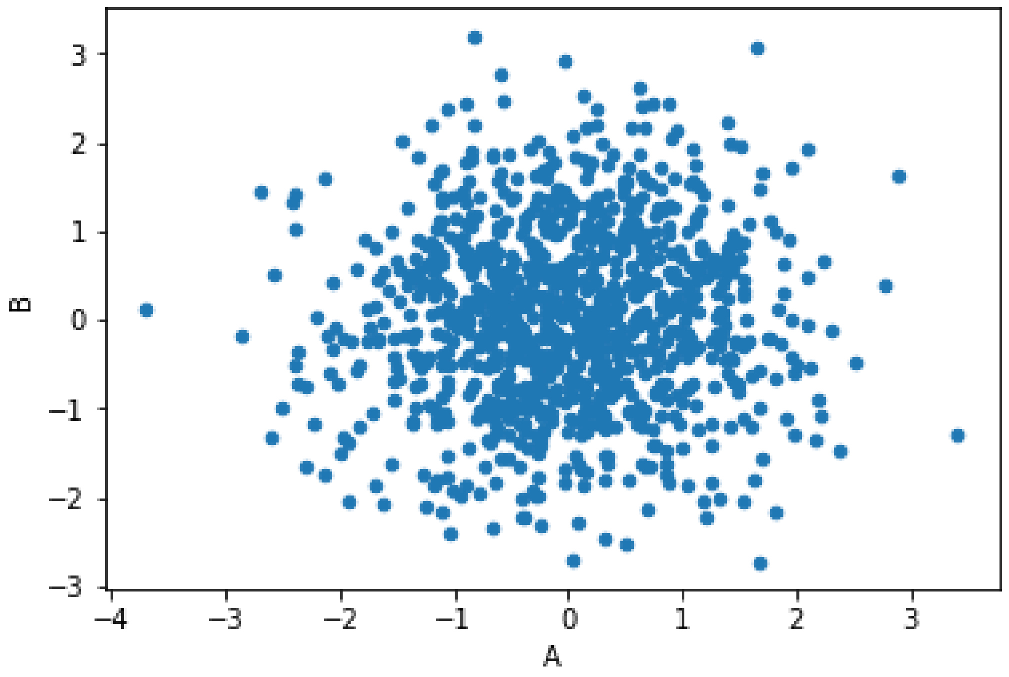
```
[18]: df2.plot.area(stacked=False, alpha=0.4);
```



2.5 Scatter Plots

Scatter plots are a useful tool to quickly compare two variables, and to look for possible trends. [reference]

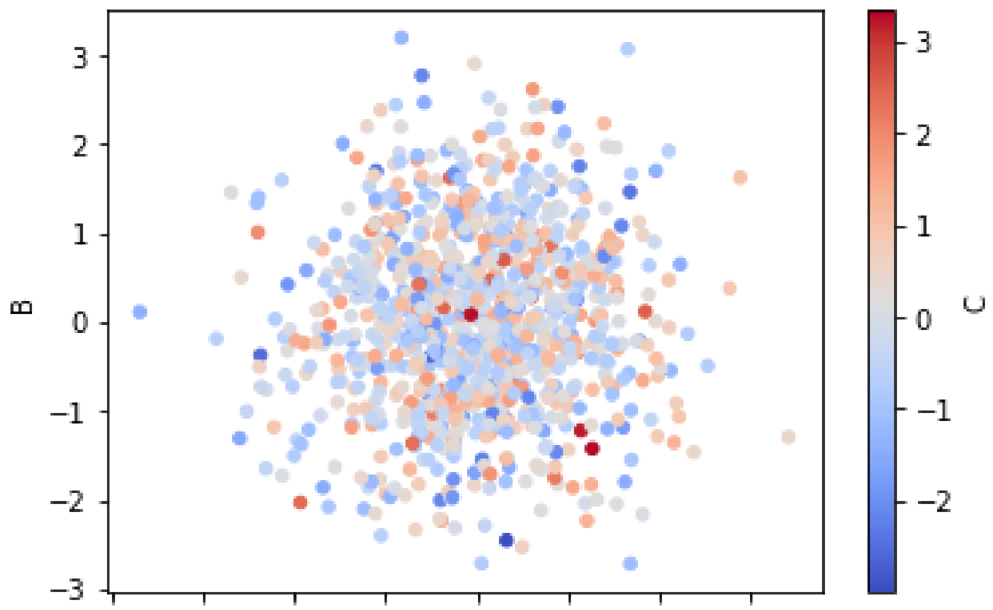
[19]:
```
df1.plot.scatter(x='A',y='B');
```



2.5.1 Scatter plots with colormaps

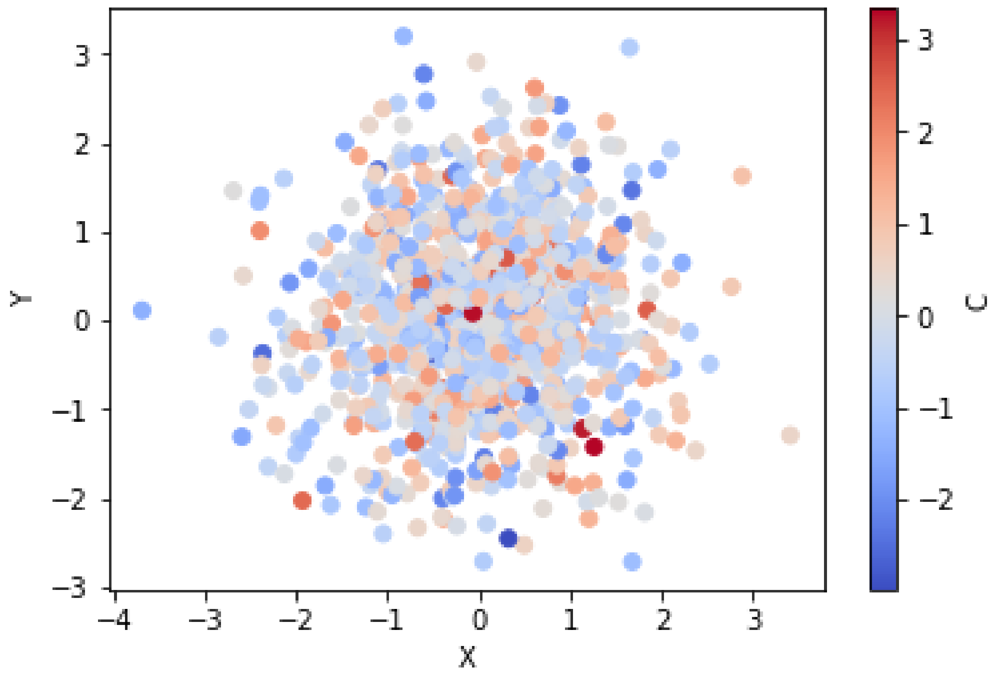
You can use c to color each marker based off another column value. Use `cmap` to indicate which colormap to use. For all the available colormaps, check out: http://matplotlib.org/users/colormaps.html

[20]:
```
df1.plot.scatter(x='A',y='B',c='C',cmap='coolwarm');
```



NOTE: As of pandas 0.23.4 there is a known issue where colormaps chop off the x-axis tics.This is due to be fixed in an upcoming release. For now, the following matplotlib code will work:

```
[21]: import matplotlib.pyplot as plt
      plt.scatter(df1['A'],df1['B'],c=df1['C'],cmap='coolwarm')
      plt.colorbar().set_label('C')
      plt.xlabel('X')
      plt.ylabel('Y')
      plt.show()
```

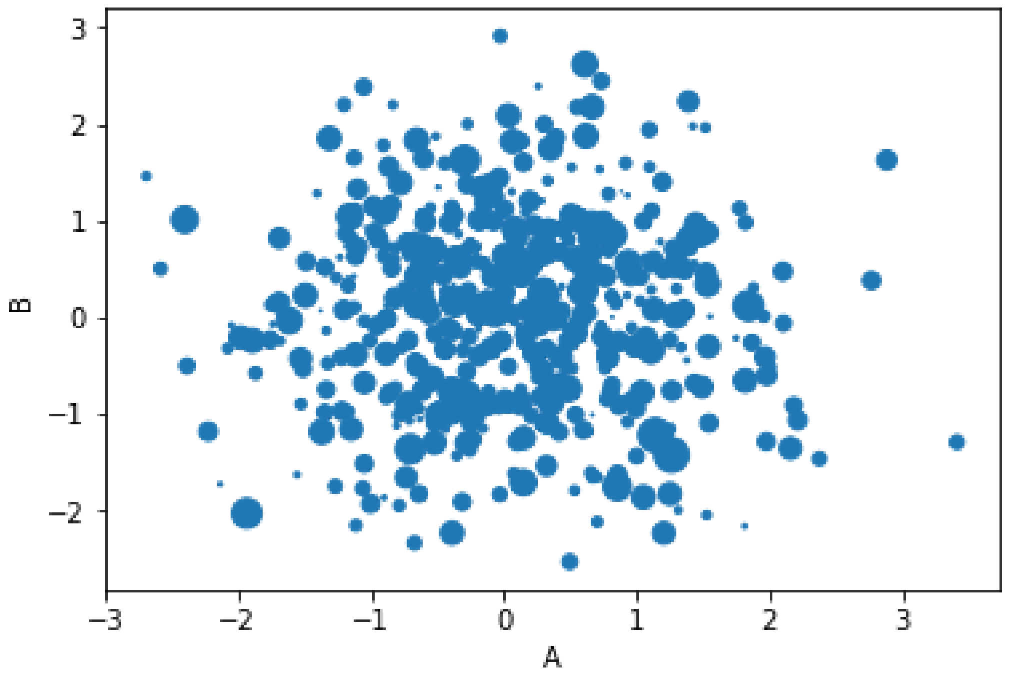


2.5.2 Scatter plots with sized markers

Alternatively you can use s to indicate marker size based off another column. The s parameter needs to be an array, not just the name of a column:

```
[22]: df1.plot.scatter(x='A',y='B',s=df1['C']*50);
```

```
C:\Anaconda3\envs\tsa_course\lib\site-packages\matplotlib\collections.py:874:
RuntimeWarning: invalid value encountered in sqrt
  scale = np.sqrt(self._sizes) * dpi / 72.0 * self._factor
```



The warning above appeared because some `df1['C']` values are negative. We can fix this finding the minimum value, writing a function that adds to each value, and applying our function to the data with .apply(func).

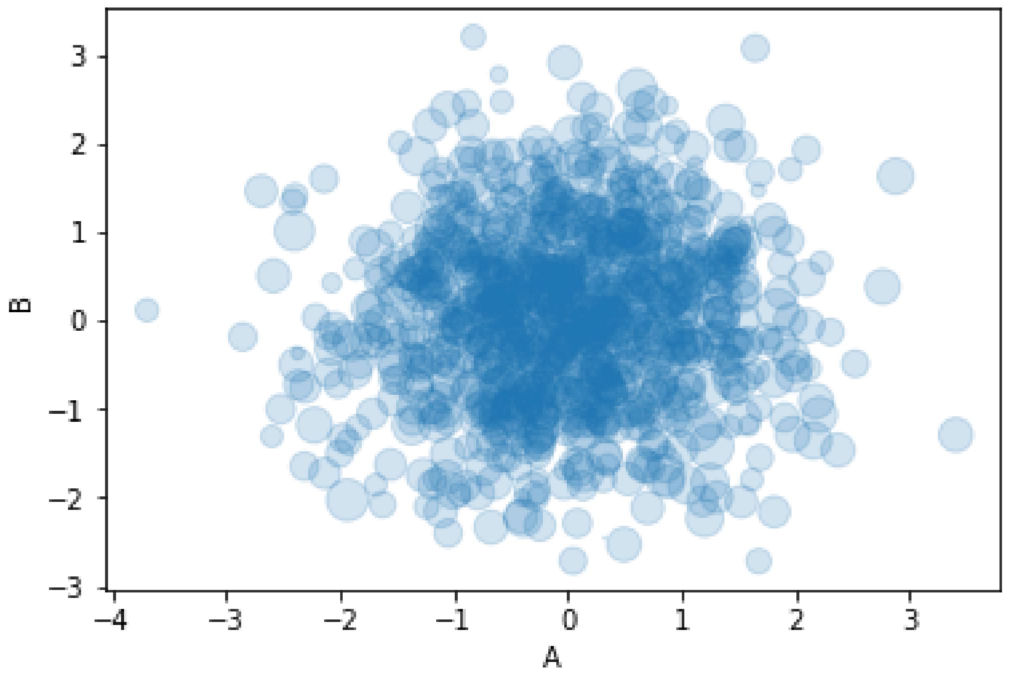
Also, these data points have a lot of overlap. We can address this issue by passing in an alpha blending value between 0 and 1 to make markers more transparent.

```
[23]: df1['C'].min()
```

```
[23]: -2.987971138961773
```

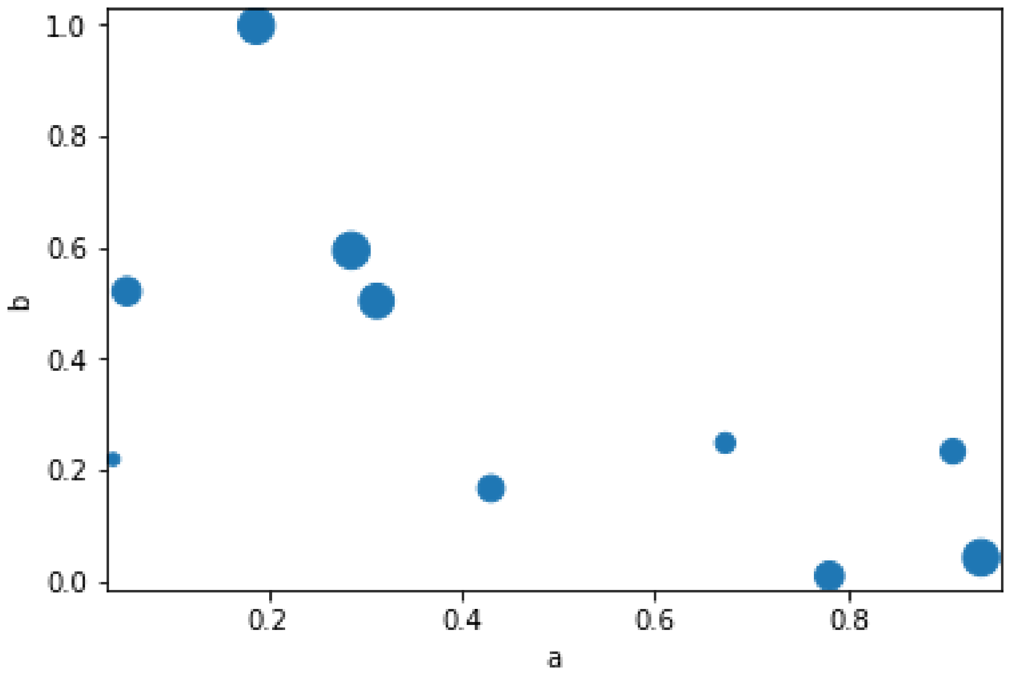
```
[24]: def add_three(val):
          return val+3

      df1.plot.scatter(x='A',y='B',s=df1['C'].apply(add_three)*45, alpha=0.2);
```



Let's see what this looks like with our smaller dataset. Here we'll also apply .autoscale() to tighten the axes.

[25]:
```
df2.plot.scatter(x='a',y='b',s=df2['c']*200).autoscale(enable=True,␣
↪axis='both', tight=True);
```

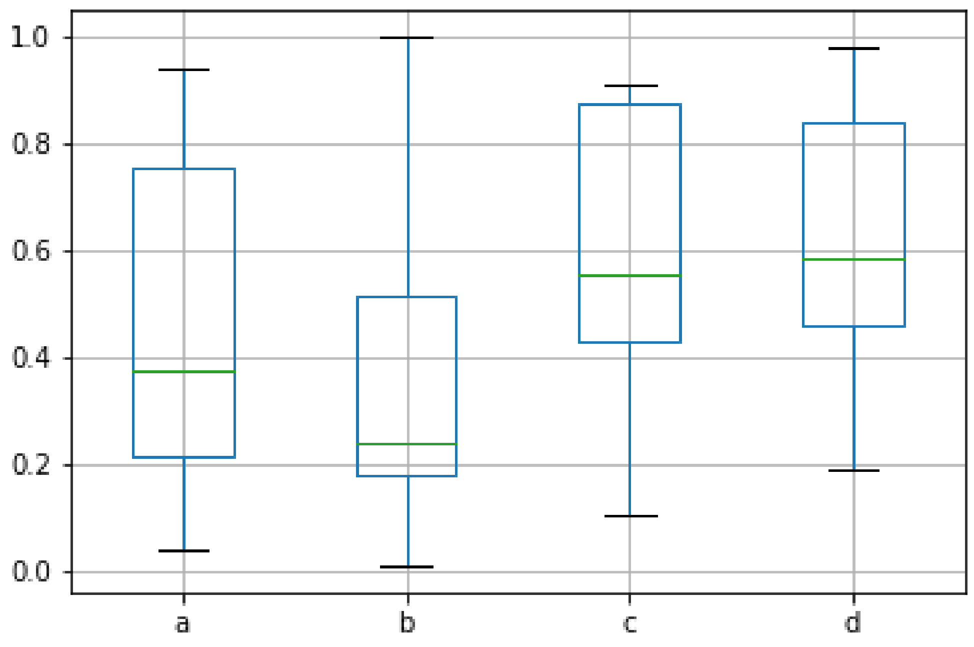


2.6 BoxPlots

Box plots, aka ”box and whisker diagrams”, describe the distribution of data by dividing data into quartiles about the mean. Look here for a description of boxplots. [reference]

NOTE: At this time we recommend using df.boxplot() instead of df.plot.box() as certain operations such as groupby currently do not work properly in df.plot.box(). For more information visit https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.boxplot.html

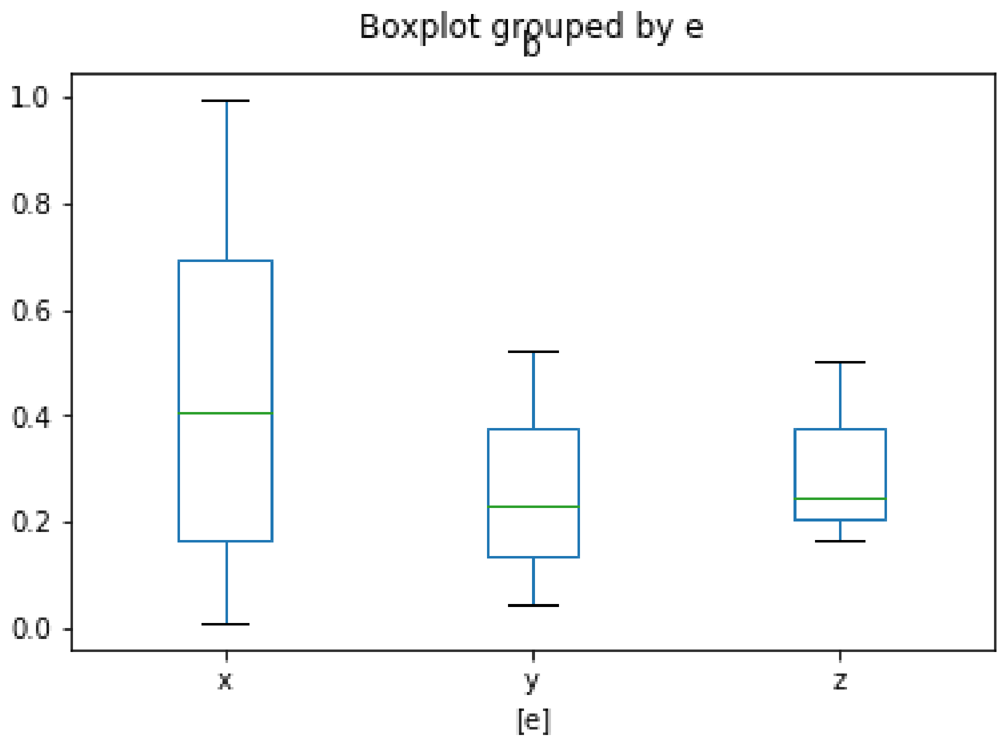
[26]:
```
df2.boxplot();
```



2.6.1 Boxplots with Groupby

To draw boxplots based on groups, first pass in a list of columns you want plotted (including the groupby column), then pass by='columname' into .boxplot(). Here we'll group records by the 'e' column, and draw boxplots for the 'b' column.

```
[27]: df2[['b','e']].boxplot(by='e', grid=False);
```

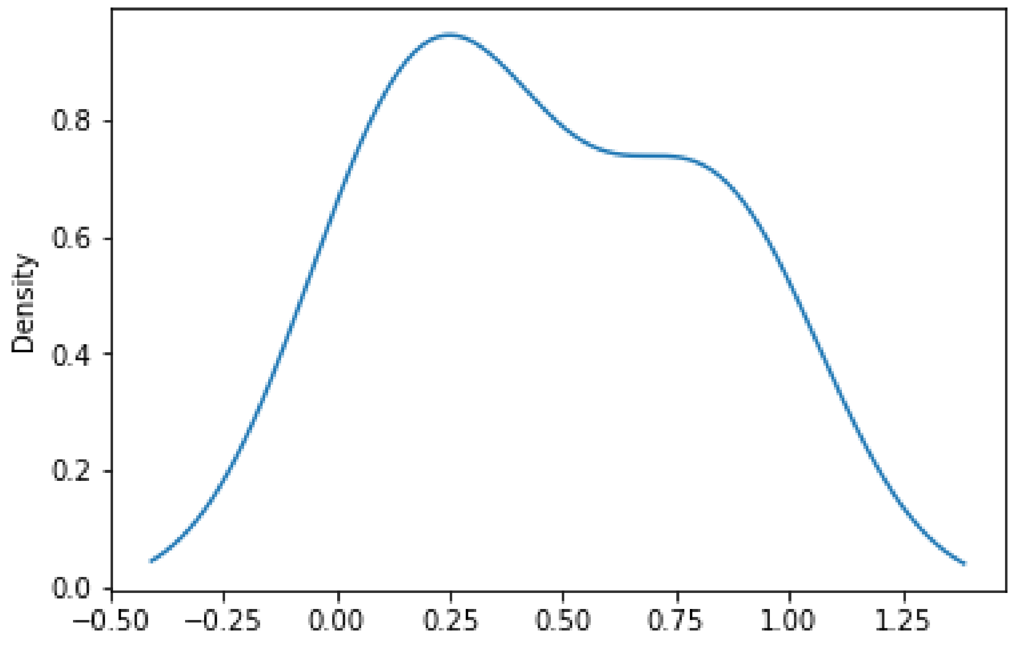


In the next section on Customizing Plots we'll show how to change the title and axis labels.

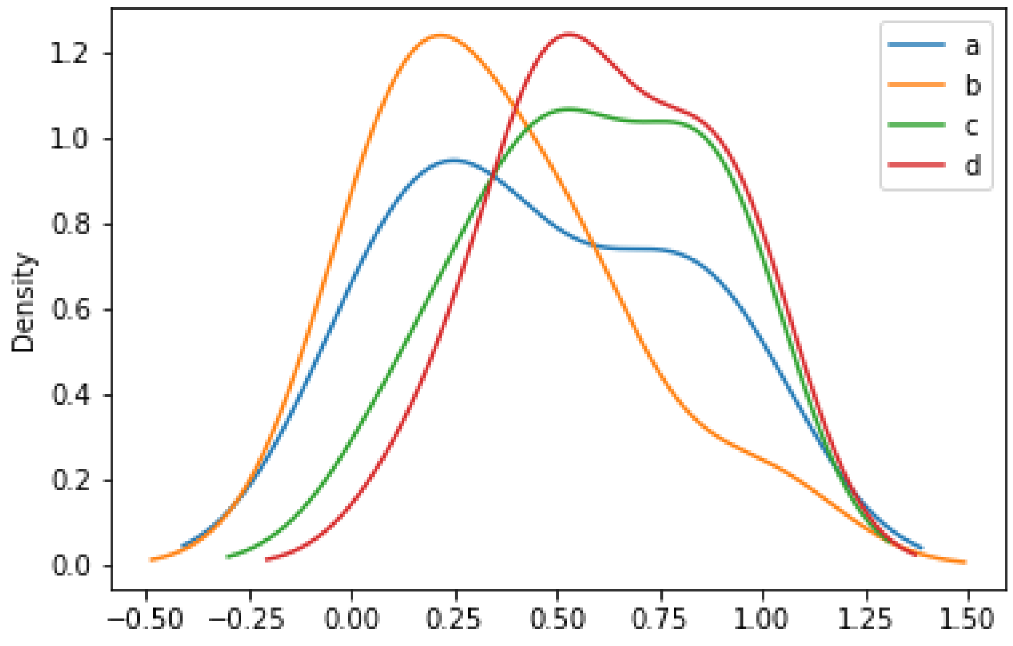
2.7 Kernel Density Estimation (KDE) Plot

These plots are accessible either through df.plot.kde() or df.plot.density() [reference]

[28]:
```
df2['a'].plot.kde();
```



[29]:
```
df2.plot.density();
```

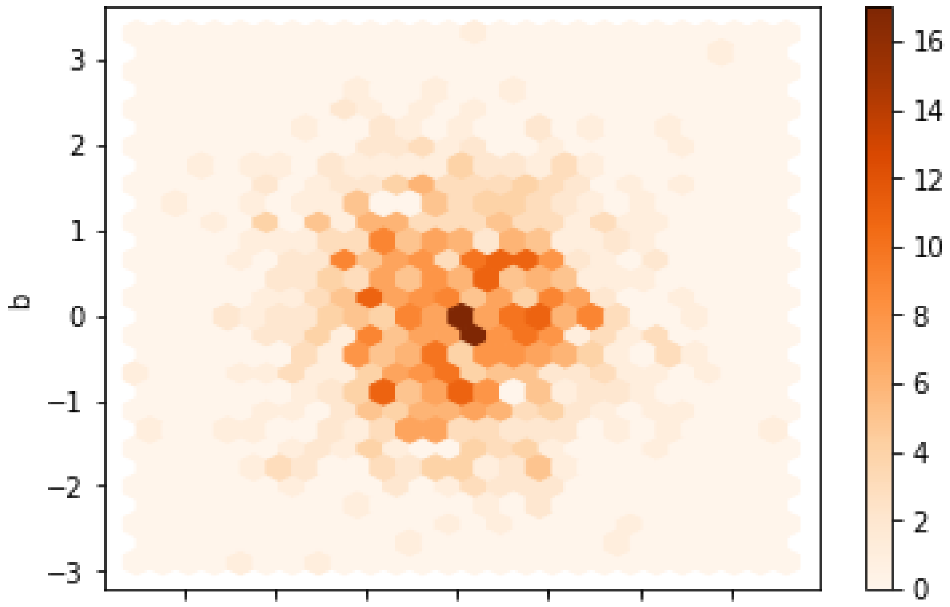


2.8 Hexagonal Bin Plot

Useful for Bivariate Data, alternative to scatterplot. [reference]

```
[30]: # FIRST CREATE A DATAFRAME OF RANDOM VALUES
df = pd.DataFrame(np.random.randn(1000, 2), columns=['a', 'b'])

# MAKE A HEXAGONAL BIN PLOT
df.plot.hexbin(x='a',y='b',gridsize=25,cmap='Oranges');
```



That's it! If you're familiar with matplotlib, hopefully you can see why this method of plotting will be a lot easier to use. Plus, a lot of the plot calls accept the same additional arguments as their parent matplotlib plt. call.

3 Great Job!