Built-inDataVisualization

June 15, 2024

[27]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

0.1 The Data

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

[28]:
```
df1 = pd.read_csv('df1',index_col=0)
df2 = pd.read_csv('df2')
```

[29]:
```
df1.head()
```

[29]:

	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

[30]:
```
df2.head()
```

[30]:

	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

0.2 Style Sheets

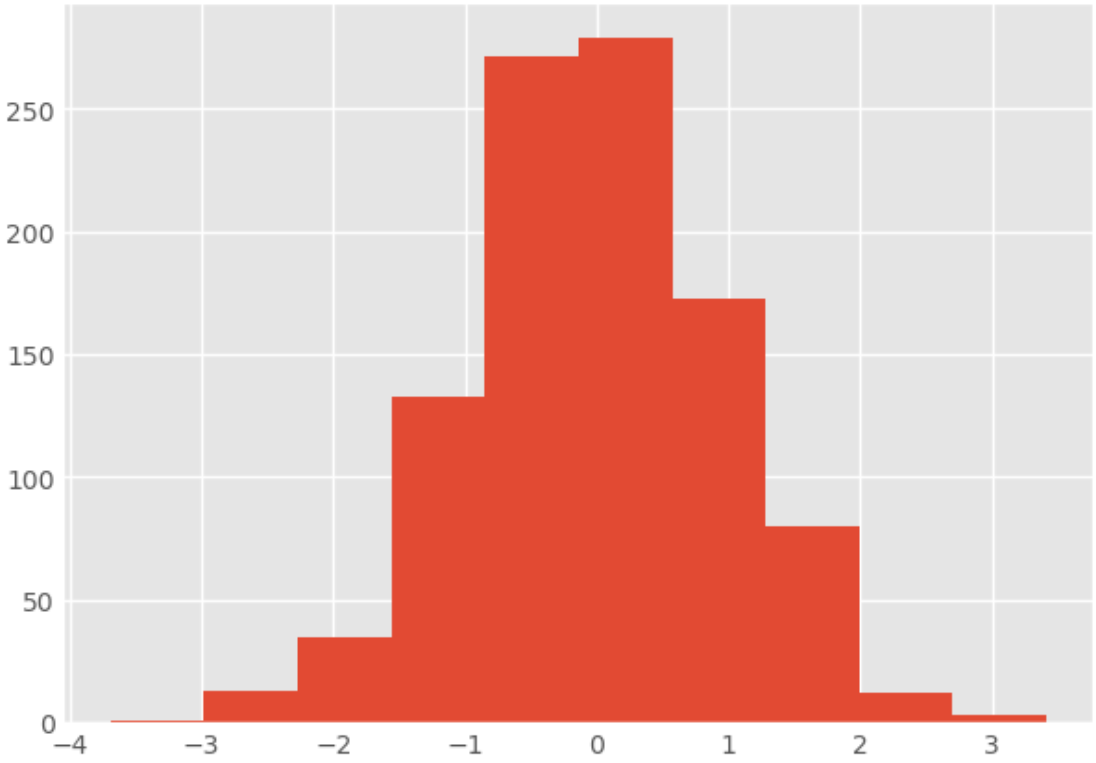
Matplotlib has style sheets you can use to make your plots look a little nicer. These style sheets include plot_bmh,plot_fivethirtyeight,plot_ggplot and more. They basically create a set of style rules that your plots follow. I recommend using them, they make all your plots have the same look and feel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

[31]:
```
df1['A'].hist()
```

[31]: <Axes: >



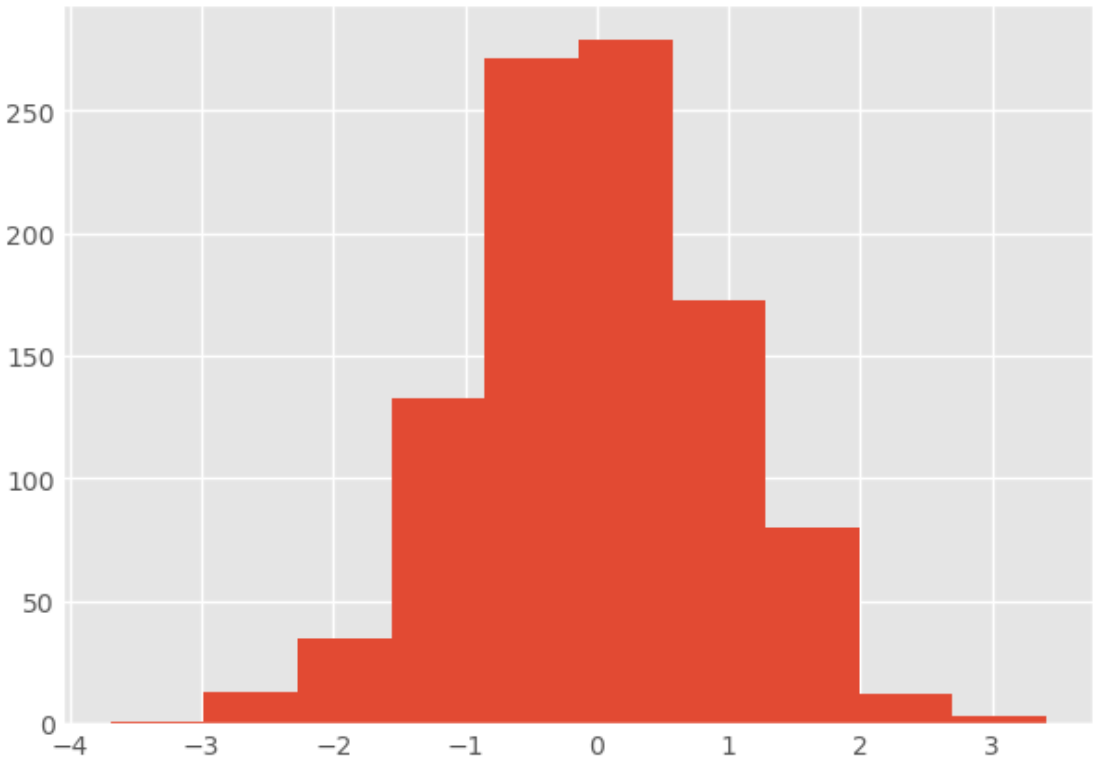
Call the style:

[32]:
```
#import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

Now your plots look like this:

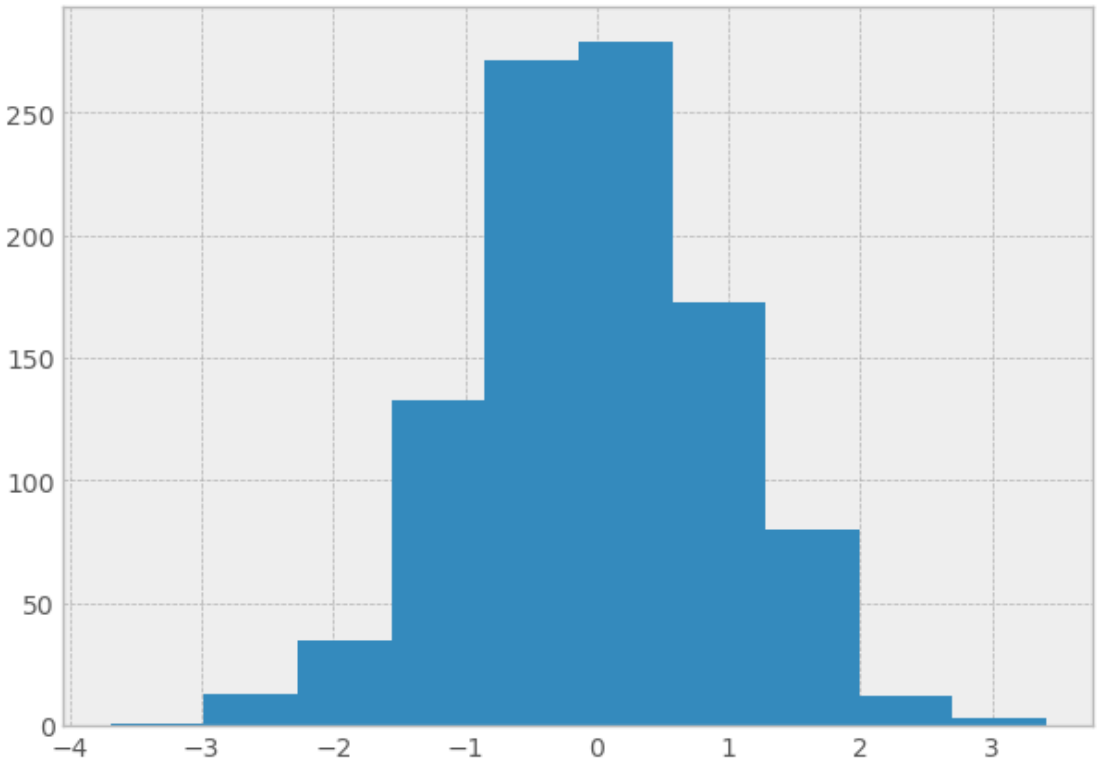
[33]:
```
df1['A'].hist()
```

[33]: <Axes: >



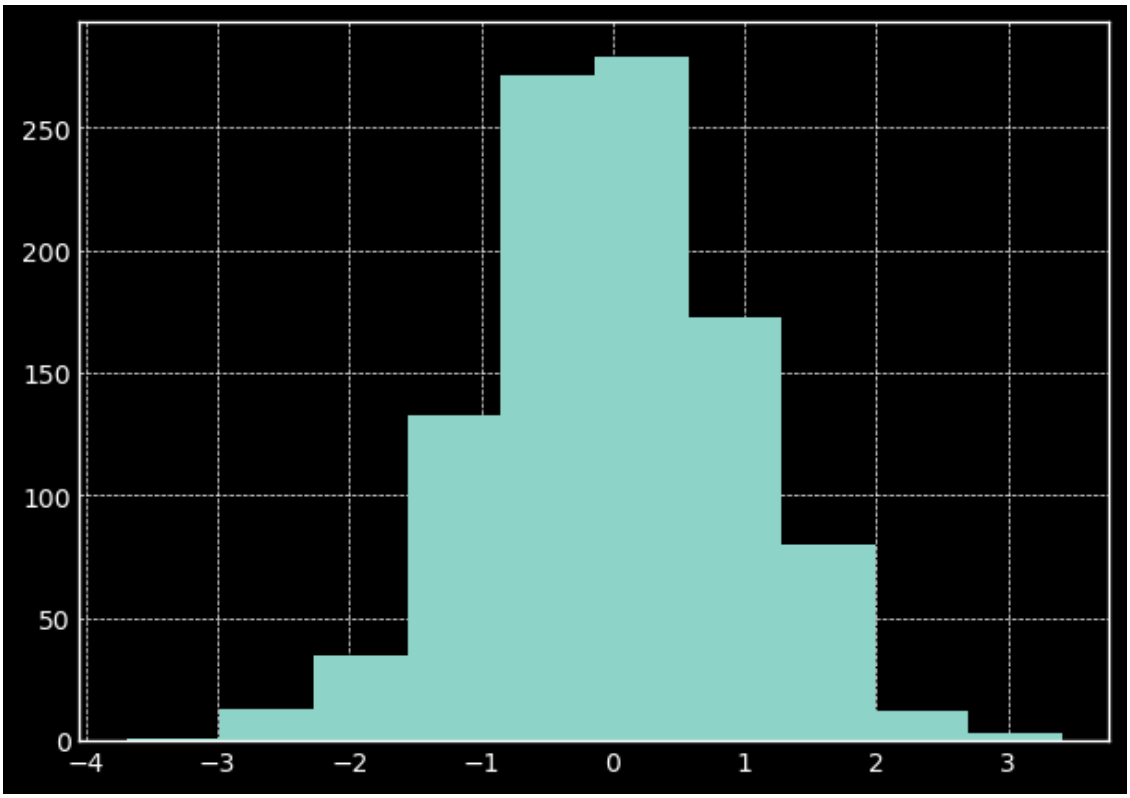
[34]:
```
plt.style.use('bmh')
df1['A'].hist()
```

[34]: <Axes: >



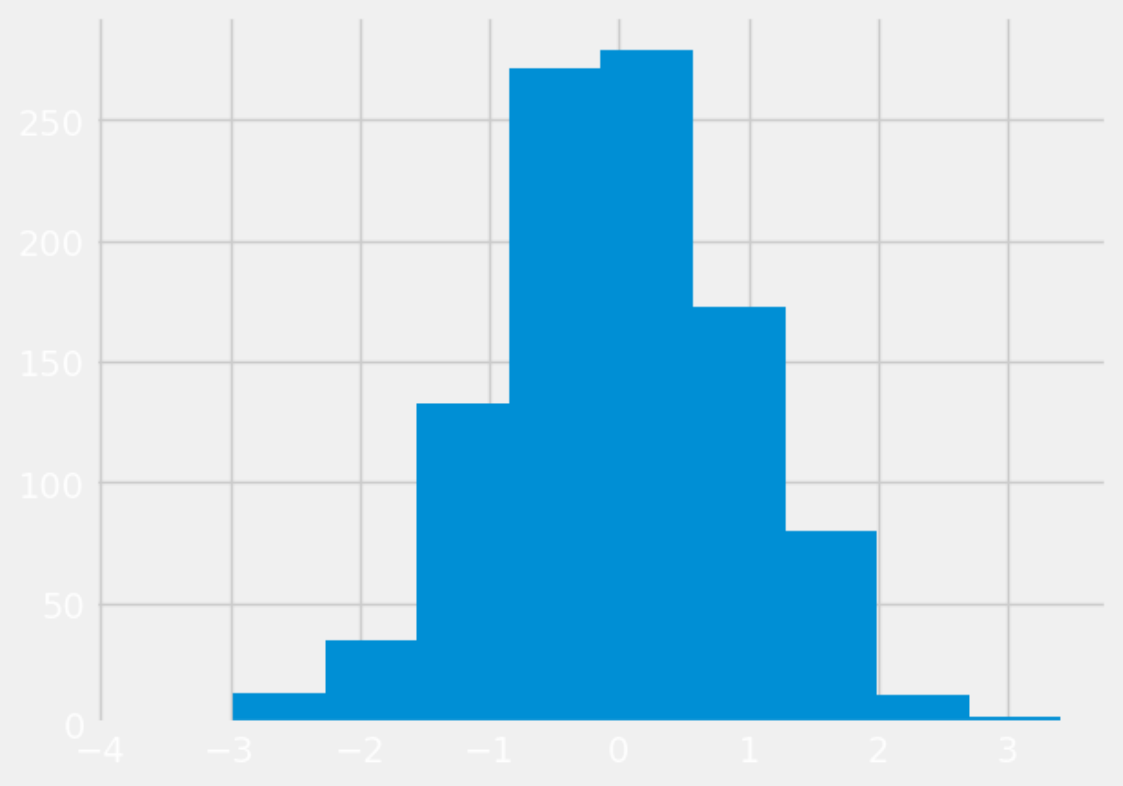
[35]:
```
plt.style.use('dark_background')
df1['A'].hist()
```

[35]: <Axes: >



[36]:
```
plt.style.use('fivethirtyeight')
df1['A'].hist()
```

[36]: <Axes: >



[37]:
```
plt.style.use('ggplot')
```

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

1 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

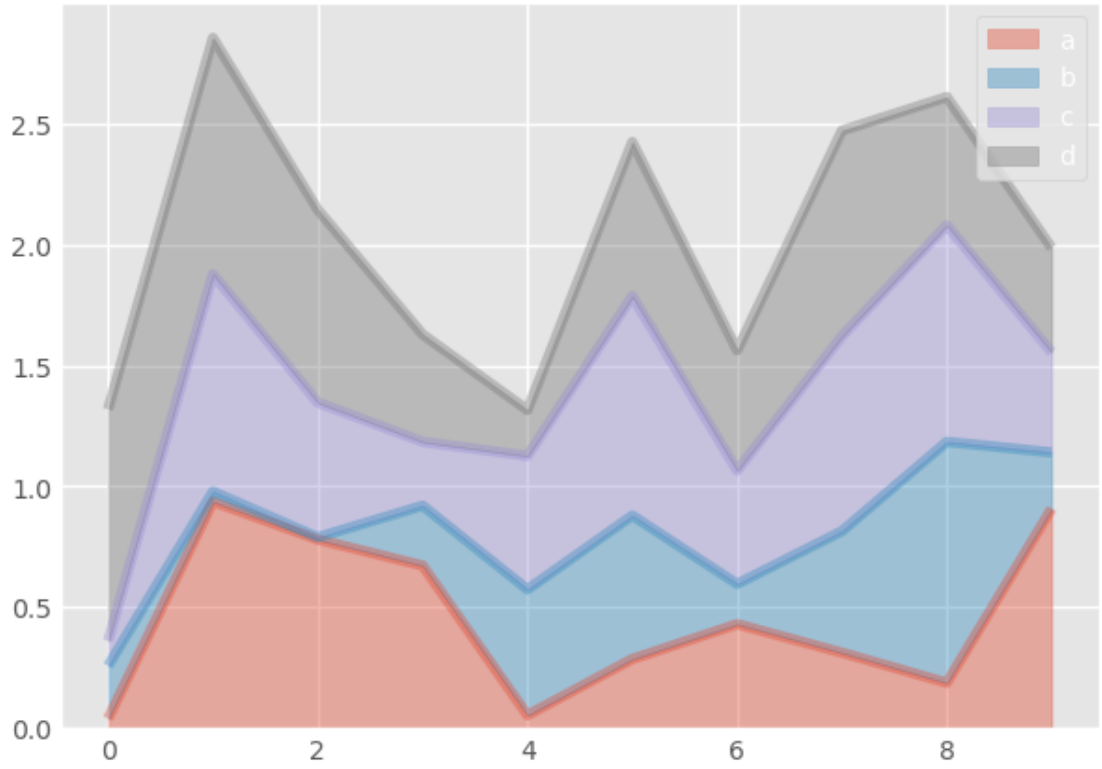
You can also just call df.plot(kind='hist') or replace that kind argument with any of the key terms shown in the list above (e.g. 'box','barh', etc..) _____

Let's start going through them!

1.1 Area

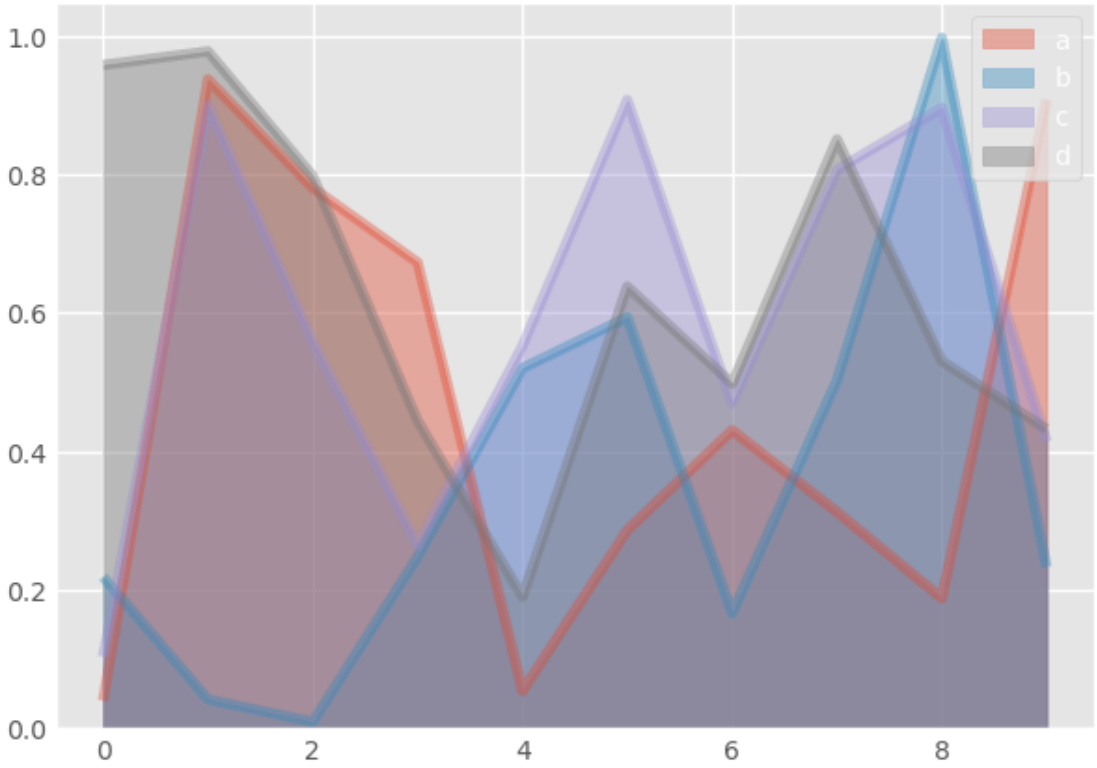
[38]:
```
df2.plot.area(alpha=0.4)
```

[38]: <Axes: >



[39]:
```
df2.plot.area(stacked=False,alpha=0.4)
```

[39]: <Axes: >



[40]:
```
df = pd.DataFrame({'sales': [3, 2, 3],'visits': [20, 42, 28],'day': [1, 2, 3],})
```

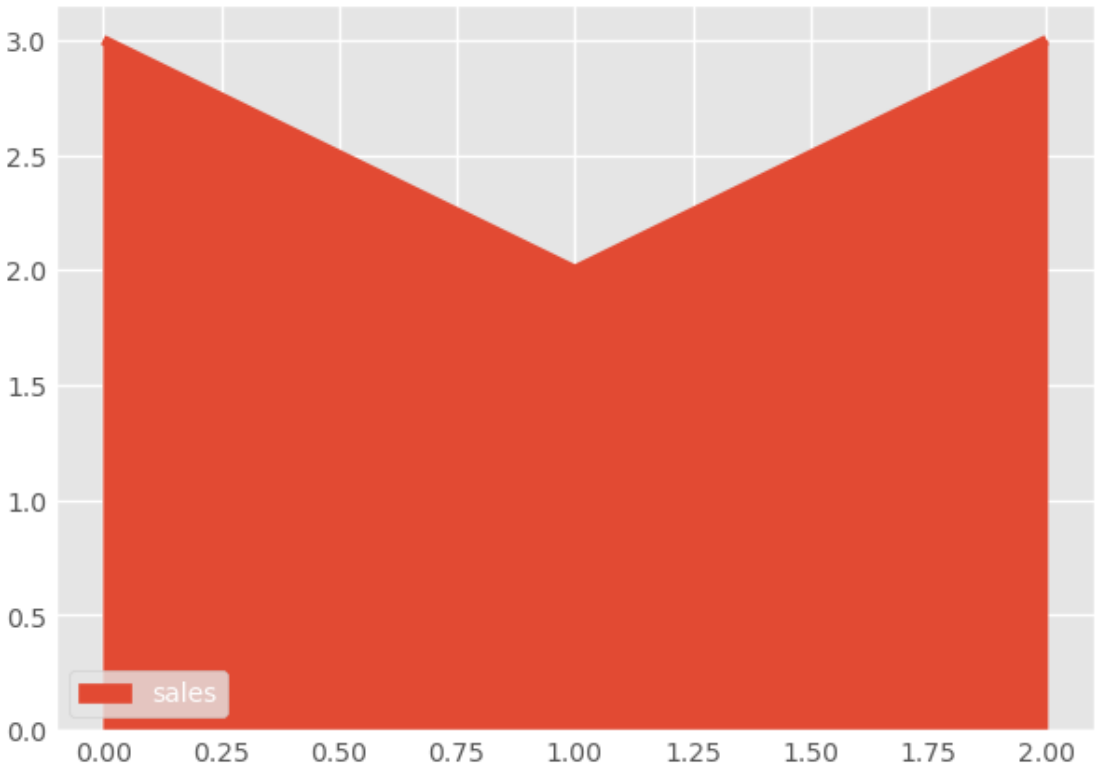
[41]:
```
df.head()
```

[41]:

	sales	visits	day
0	3	20	1
1	2	42	2
2	3	28	3

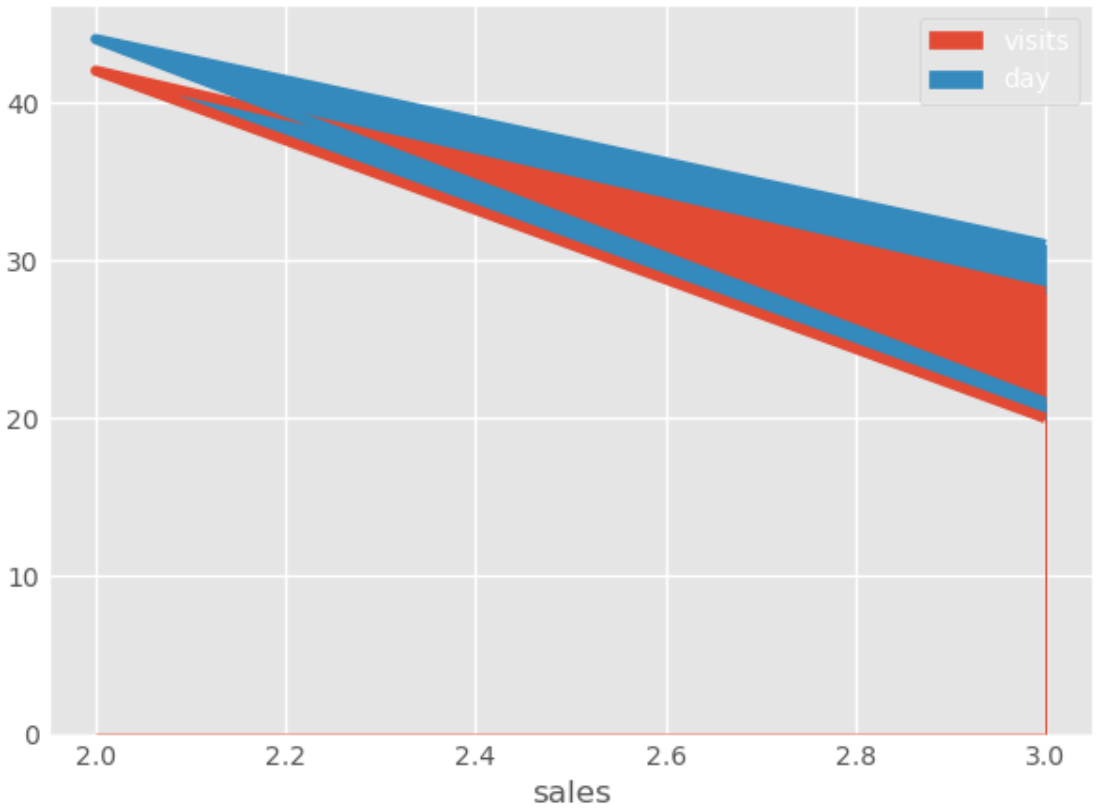
[43]:
```
df.plot.area(y='sales')
```

[43]: <Axes: >



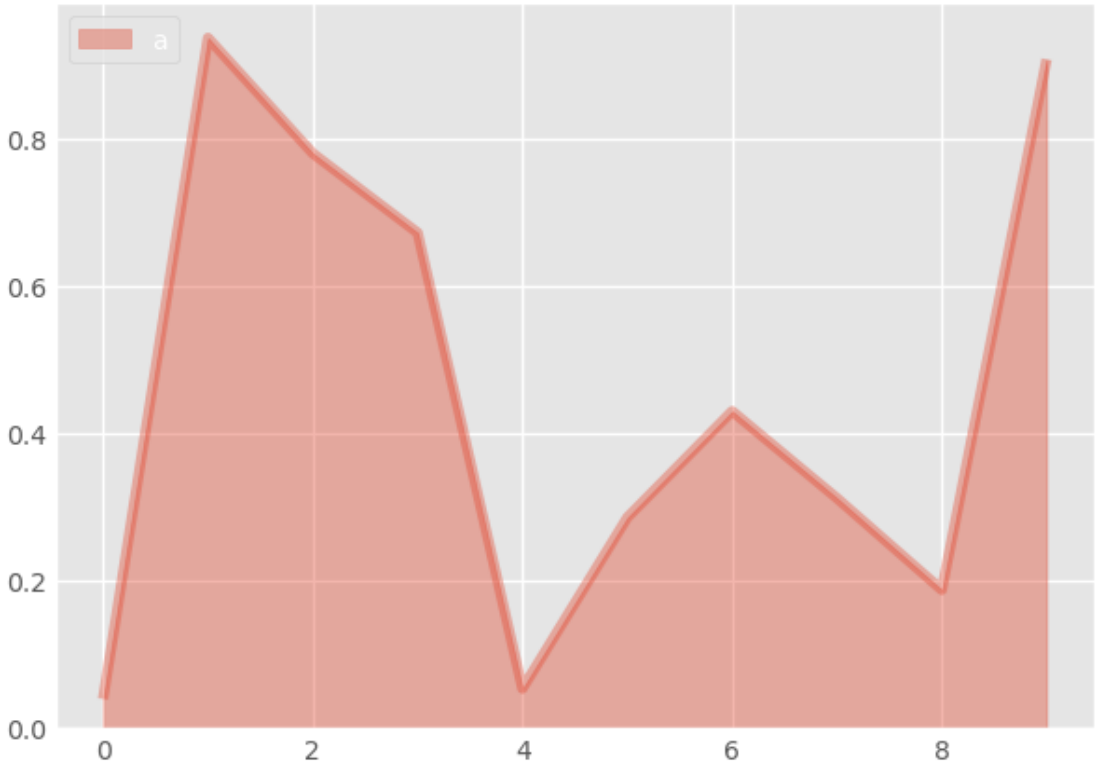
[44]:
```
df.plot.area(x='sales')
```

[44]: <Axes: xlabel='sales'>



[46]:
```
df2.plot.area(y='a',alpha=0.4)
```

[46]: <Axes: >



1.2 Barplots

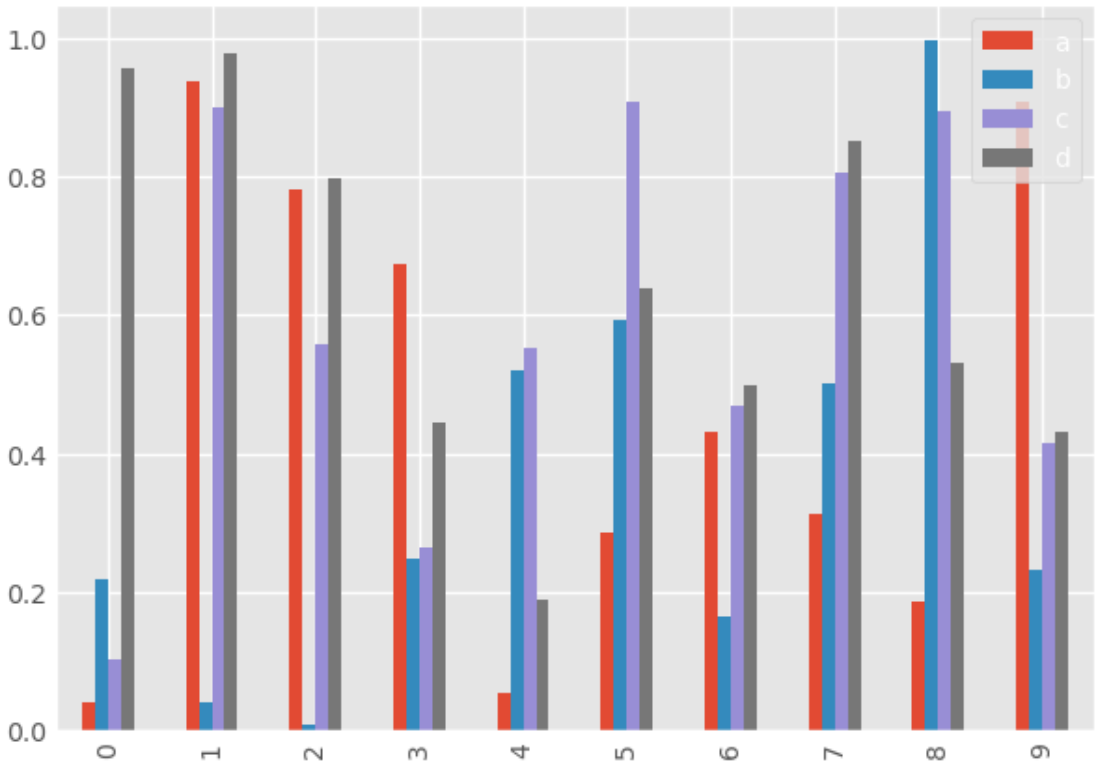
[47]:
```
df2.head()
```

[47]:

	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

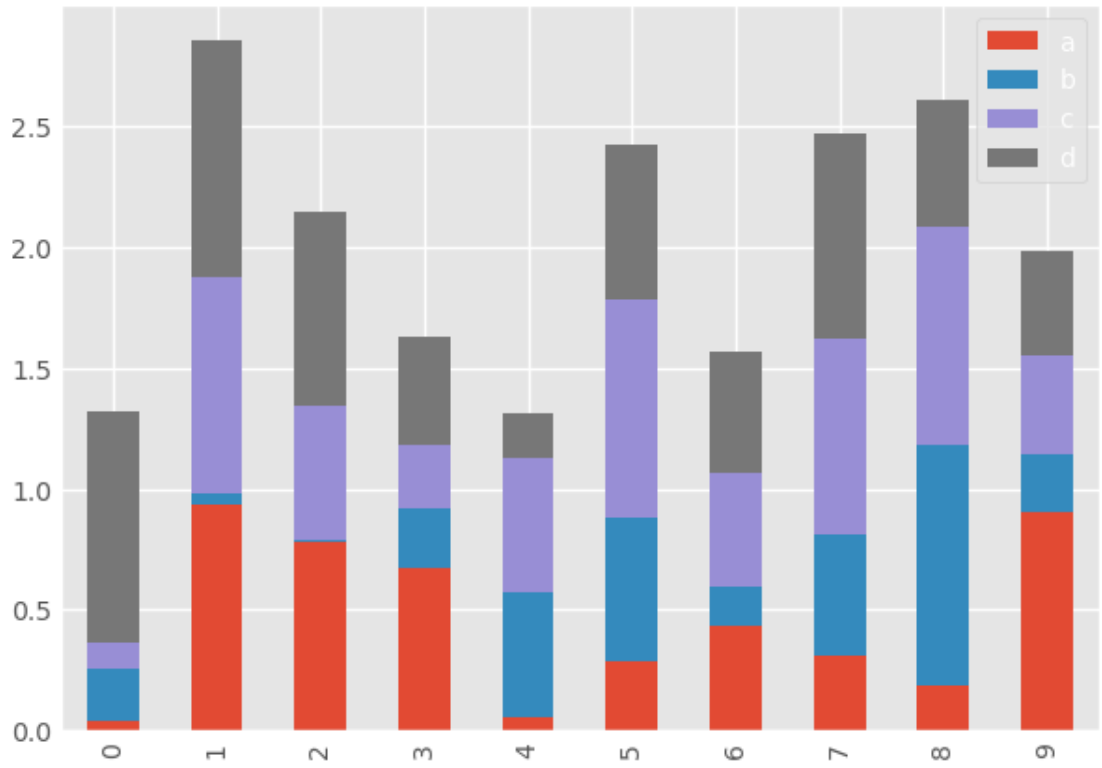
[48]:
```
df2.plot.bar()
```

[48]:
```
<Axes: >
```



[49]:
```
df2.plot.bar(stacked=True)
```

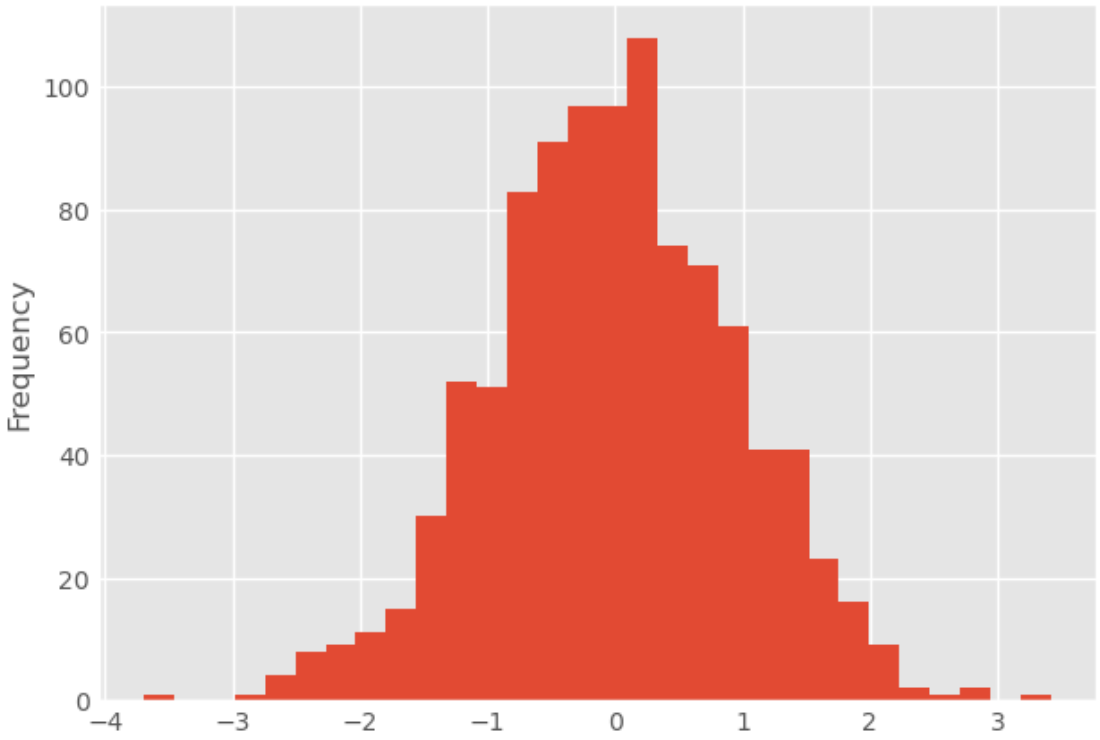
[49]: <Axes: >



1.3 Histograms

[51]:
```
df1['A'].plot.hist(bins=30)
```

[51]:
```
<Axes: ylabel='Frequency'>
```

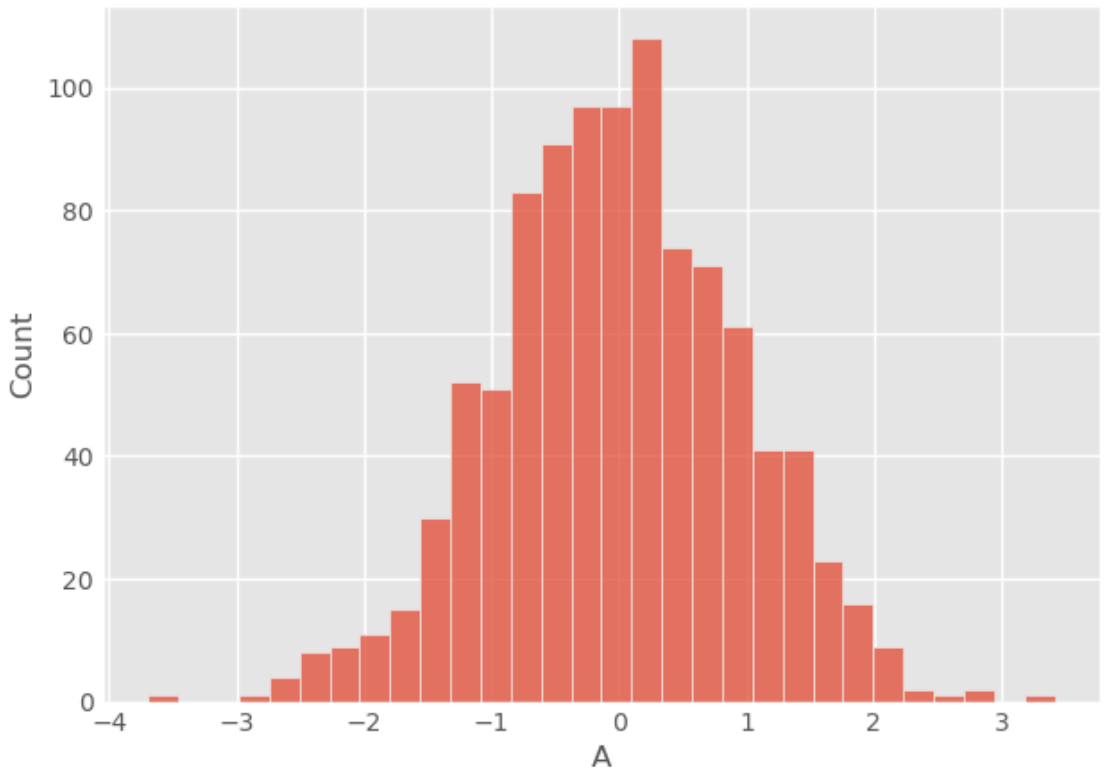


[52]:
```
import seaborn as sns
```

[56]:
```
sns.histplot(df1['A'],bins=30)
```

```
/home/fischer/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

[56]: <Axes: xlabel='A', ylabel='Count'>



[57]:
```
df1.head()
```

[57]:

	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

[61]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, 2000-01-01 to 2002-09-26
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       1000 non-null   float64
 1   B       1000 non-null   float64
 2   C       1000 non-null   float64
 3   D       1000 non-null   float64
dtypes: float64(4)
memory usage: 39.1+ KB
```

[59]:
```
# we want a sort of graph to show those info above
```

[62]:
```
df1.set_index('indice')
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[62], line 1
----> 1 df1.set_index('indice')

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/frame.py:5870, in␣
 ↪DataFrame.set_index(self, keys, drop, append, inplace, verify_integrity)
   5867                 missing.append(col)
   5869 if missing:
-> 5870     raise KeyError(f"None of {missing} are in the columns")
   5872 if inplace:
   5873     frame = self

KeyError: "None of ['indice'] are in the columns"
```

[71]:
```
df1 = pd.read_csv('df1')
df1
```

[71]:

	Date	A	B	C	D
0	2000-01-01	1.339091	-0.163643	-0.646443	1.041233
1	2000-01-02	-0.774984	0.137034	-0.882716	-2.253382
2	2000-01-03	-0.921037	-0.482943	-0.417100	0.478638
3	2000-01-04	-1.738808	-0.072973	0.056517	0.015085
4	2000-01-05	-0.905980	1.778576	0.381918	0.291436
..	…	…	…	…	…
995	2002-09-22	1.013897	-0.288680	-0.342295	-0.638537
996	2002-09-23	-0.642659	-0.104725	-0.631829	-0.909483
997	2002-09-24	0.370136	0.233219	0.535897	-1.552605
998	2002-09-25	0.183339	1.285783	-1.052593	-2.565844
999	2002-09-26	0.775133	-0.850374	0.486728	-1.053427

[1000 rows x 5 columns]

[72]:
```
df1.set_index('Date',inplace=True)
```

[73]:
```
df1.head()
```

[73]:

Date	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
```

1.4 Line Plots

[85]:
```
df1.plot.line(x='Date',y='A',figsize=(12,3),lw=1)
```

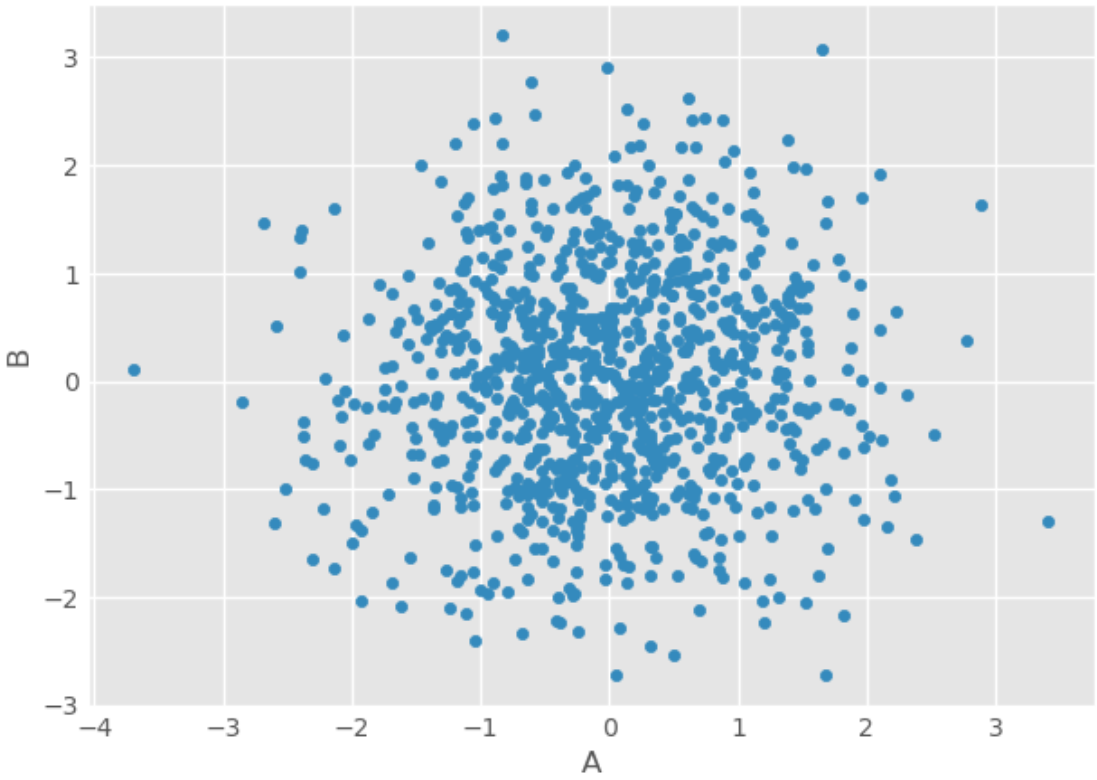
```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File ~/anaconda3/lib/python3.11/site-packages/pandas/core/indexes/base.py:3791,
 ↪in Index.get_loc(self, key)
   3790 try:
-> 3791     return self._engine.get_loc(casted_key)
   3792 except KeyError as err:

File index.pyx:152, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:181, in pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

KeyError: 'Date'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[85], line 1
----> 1 df1.plot.line(x='Date',y='A',figsize=(12,3),lw=1)

File ~/anaconda3/lib/python3.11/site-packages/pandas/plotting/_core.py:1101, in
 ↪PlotAccessor.line(self, x, y, **kwargs)
   1035 @Appender(
   1036     """
   1037     See Also
   (…)
   1093     self, x: Hashable | None = None, y: Hashable | None = None, **kwarg
   1094 ) -> PlotAccessor:
   1095     """
   1096     Plot Series or DataFrame as lines.
   1097
   1098     This function is useful to plot lines using DataFrame's values
   1099     as coordinates.
   1100     """
```

```
-> 1101     return self(kind="line", x=x, y=y, **kwargs)

File ~/anaconda3/lib/python3.11/site-packages/pandas/plotting/_core.py:996, in␣
 ↪PlotAccessor.__call__(self, *args, **kwargs)
    994 if is_integer(x) and not data.columns._holds_integer():
    995     x = data_cols[x]
--> 996 elif not isinstance(data[x], ABCSeries):
    997     raise ValueError("x must be a label or position")
    998 data = data.set_index(x)

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/frame.py:3893, in␣
 ↪DataFrame.__getitem__(self, key)
   3891 if self.columns.nlevels > 1:
   3892     return self._getitem_multilevel(key)
-> 3893 indexer = self.columns.get_loc(key)
   3894 if is_integer(indexer):
   3895     indexer = [indexer]

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/indexes/base.py:3798,␣
 ↪in Index.get_loc(self, key)
   3793     if isinstance(casted_key, slice) or (
   3794         isinstance(casted_key, abc.Iterable)
   3795         and any(isinstance(x, slice) for x in casted_key)
   3796     ):
   3797         raise InvalidIndexError(key)
-> 3798     raise KeyError(key) from err
   3799 except TypeError:
   3800     # If we have a listlike key, _check_indexing_error will raise
   3801     #  InvalidIndexError. Otherwise we fall through and re-raise
   3802     #  the TypeError.
   3803     self._check_indexing_error(key)

KeyError: 'Date'
```

1.5 Scatter Plots

[86]:
```
df1.plot.scatter(x='A',y='B')
```

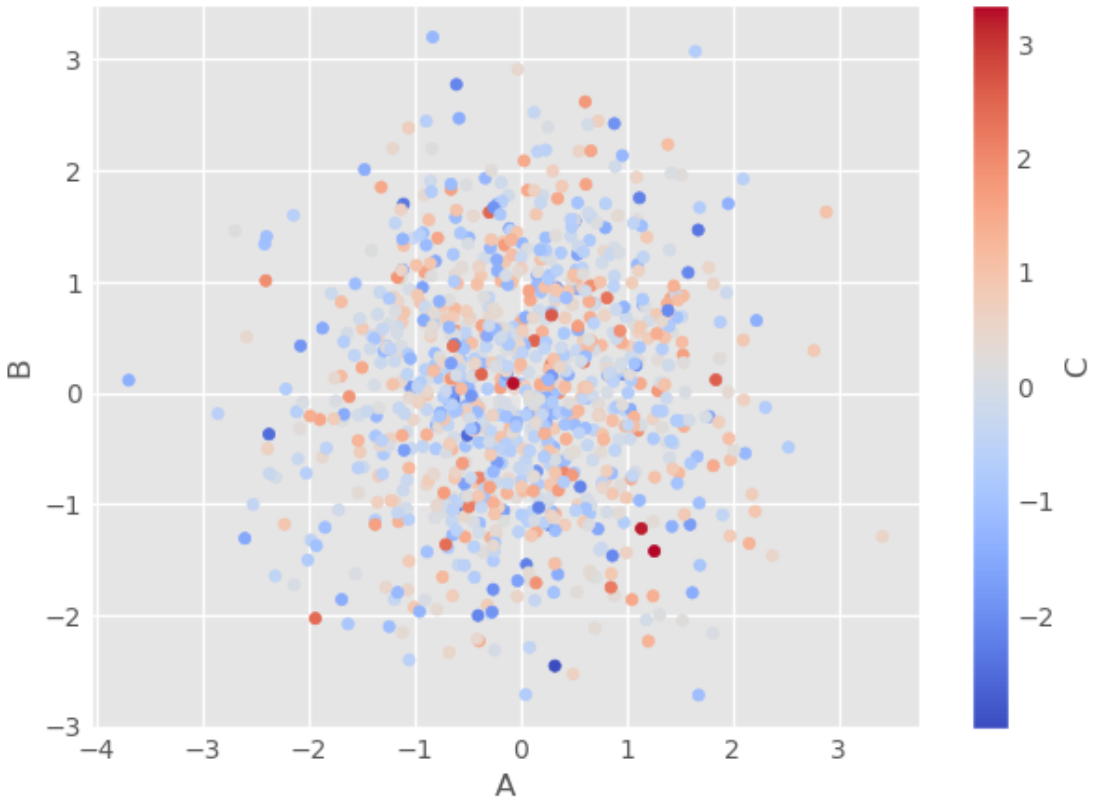
[86]:
```
<Axes: xlabel='A', ylabel='B'>
```



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

[93]:
```
df1.plot.scatter(x='A',y='B',c='C',cmap='coolwarm')
```

[93]:
```
<Axes: xlabel='A', ylabel='B'>
```

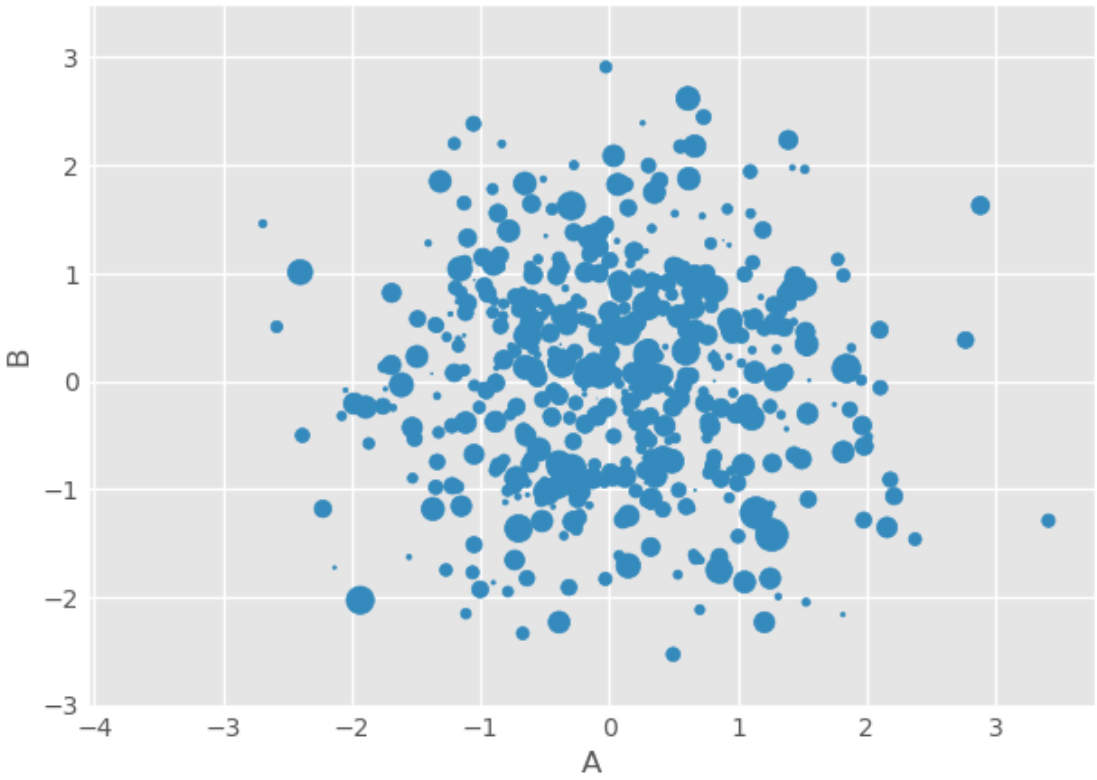


Or use s to indicate size based off another column. s parameter needs to be an array, not just the name of a column:

[94]:
```
df1.plot.scatter(x='A',y='B',s=df1['C']*50)
```

```
/home/fischer/anaconda3/lib/python3.11/site-
packages/matplotlib/collections.py:963: RuntimeWarning: invalid value
encountered in sqrt
  scale = np.sqrt(self._sizes) * dpi / 72.0 * self._factor
```

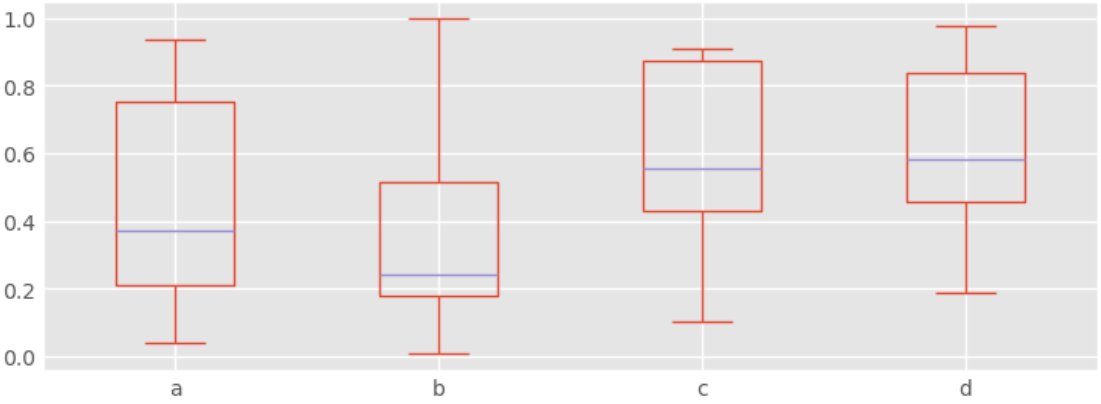
[94]:
```
<Axes: xlabel='A', ylabel='B'>
```



1.6 BoxPlots

[99]:
```
df2.plot.box(figsize=(8, 3)) # Can also pass a by= argument for groupby
```

[99]: <Axes: >

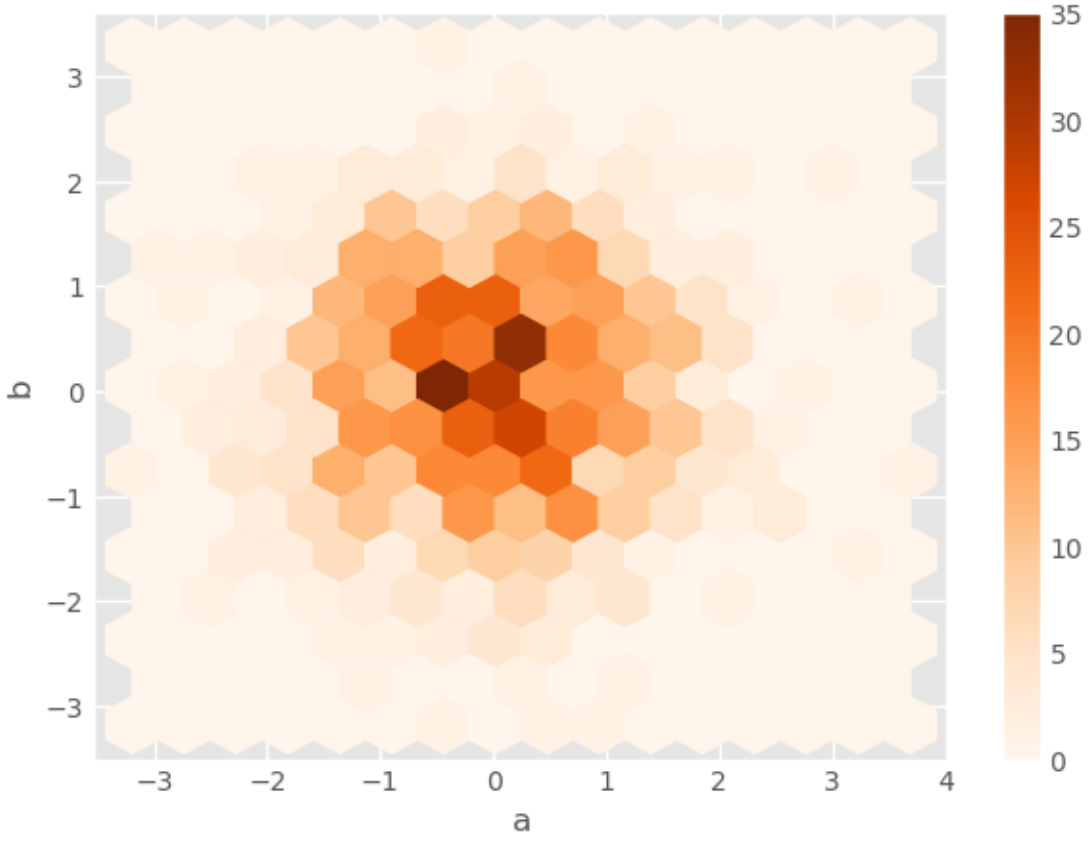


1.7 Hexagonal Bin Plot

Useful for Bivariate Data, alternative to scatterplot:

[101]:
```
df = pd.DataFrame(np.random.randn(1000, 2), columns=['a', 'b'])
df.plot.hexbin(x='a',y='b',gridsize=15,cmap='Oranges')
```

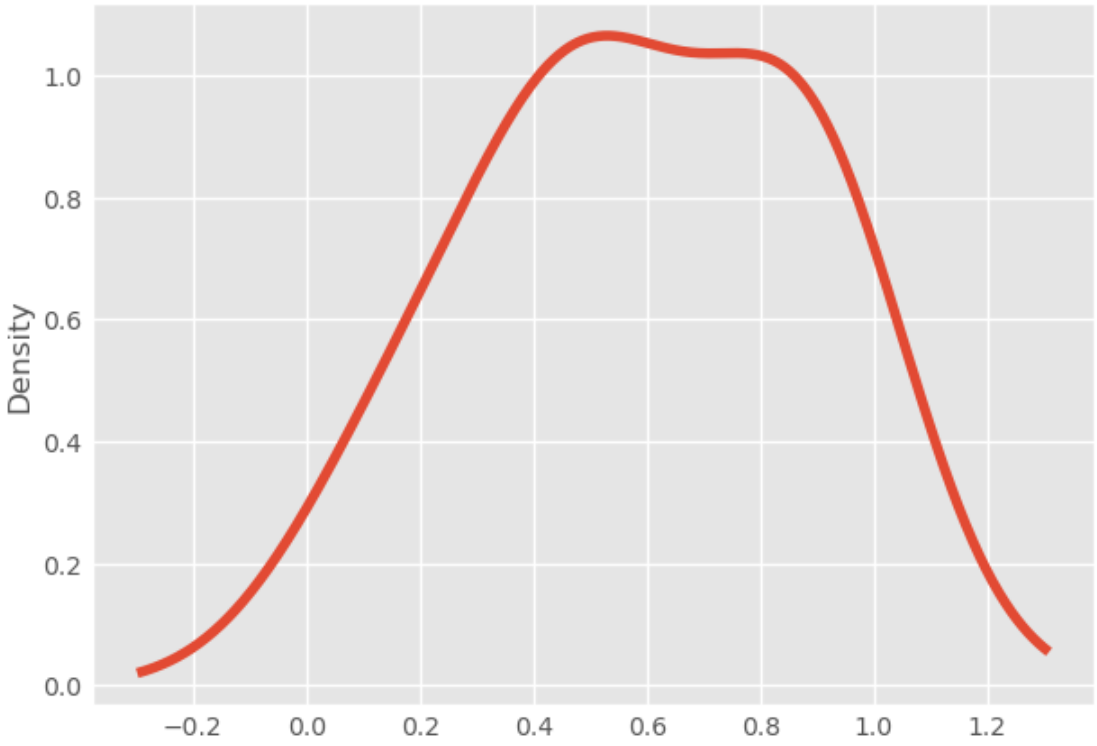
[101]:
```
<Axes: xlabel='a', ylabel='b'>
```



1.8 Kernel Density Estimation plot (KDE)

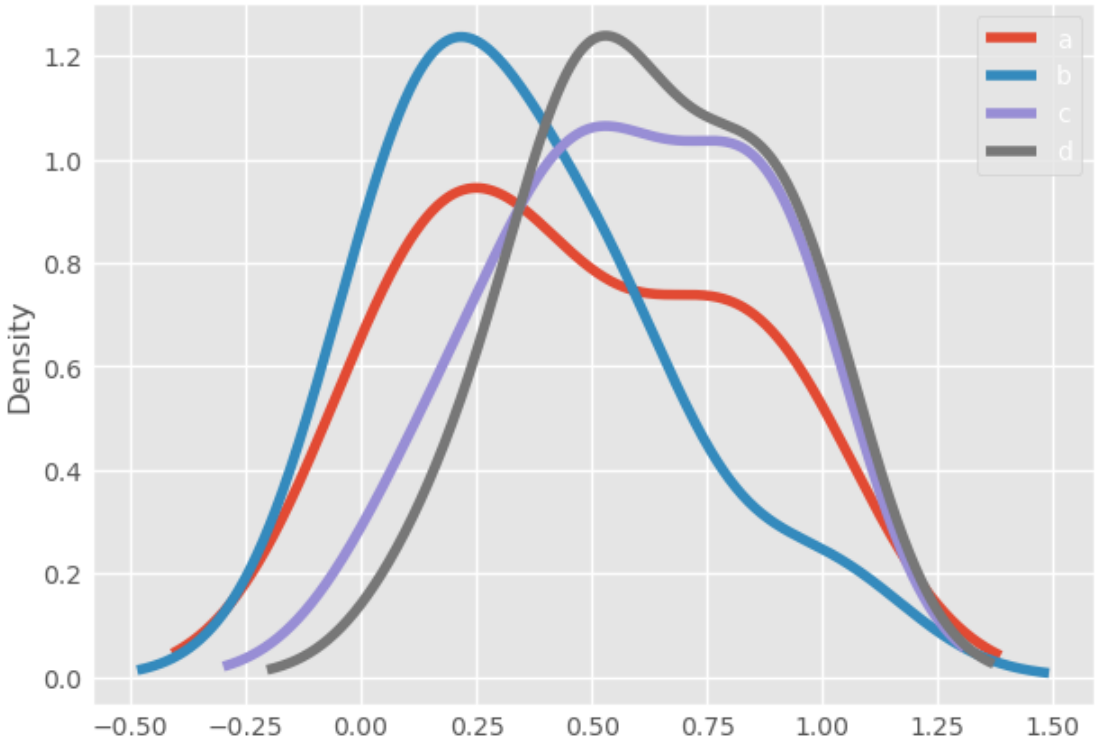
[107]:
```
df2['c'].plot.kde()
```

[107]:
```
<Axes: ylabel='Density'>
```



[106]:
```
df2.plot.density()
```

[106]:
```
<Axes: ylabel='Density'>
```


a
b
c
d
Density
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50

[ ]: