Plotting and Visualisation

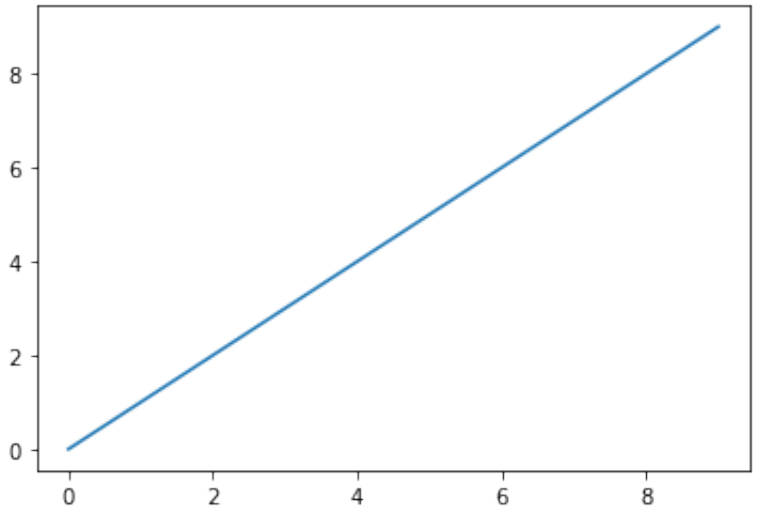
matplotlib is the main module used for plotting 2D graphics in python. The graphics are exportable into many different file formats

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
data = np.arange(10)
data
```

Out[1]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

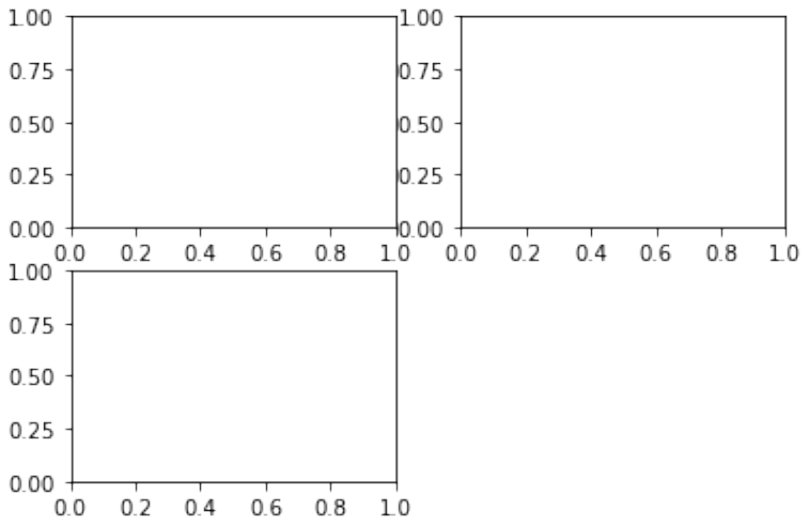
In [2]:
```
plt.plot(data)
```

Out[2]:
```
[<matplotlib.lines.Line2D at 0x17e4a68b7f0>]
```



Figures and subplots

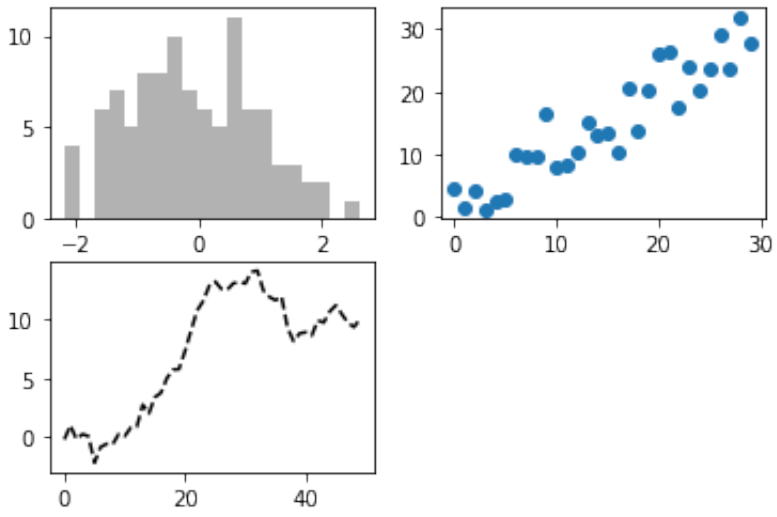
In [5]:
```
fig = plt.figure() #creates a new figure
ax1 = fig.add_subplot(2, 2, 1) #means figure should be 2x2 so up to fo
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
```



one annoyance when using jupyter notebooks is that plots reset when you use a new cell so all your plotting
code must be used in a single cell

In [9]:
```
fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
# when you issue a plotting command. matplotlib draws on the last figu
plt.plot(np.random.randn(50).cumsum(), 'k--') # 'k--' is a style optic
_=ax1.hist(np.random.randn(100), bins=20, color='k', alpha=0.3)
ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
```

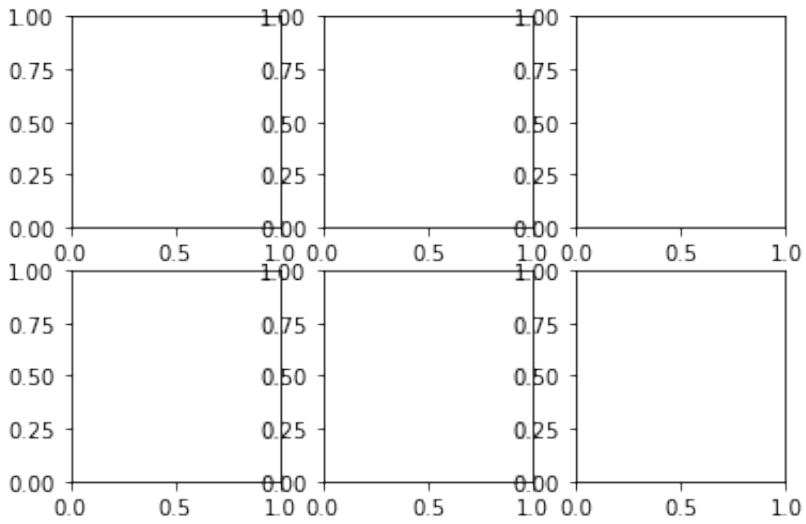
Out[9]: <matplotlib.collections.PathCollection at 0x17e4da73a30>



Creating a figure with a grid of subplots is a very common task. So matplot lib includes plt.subplots, that creates a new figure and returns a NumPy array containing the created subplot objects

In [13]:
```
fig, axis = plt.subplots(2, 3)
axis
```

Out[13]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000017E4
A5EFA90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000017E4
D402790>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000017E4
AD08E50>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000017E4
B1958B0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000017E4
D359640>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000017E4
D42AFA0>]],
      dtype=object)
```



The axis array can be indexed like a 2D array. for example, axes[0, 1]. You can also indicate subplots should use the same x, y axis. using sharex and sharey, respectively.

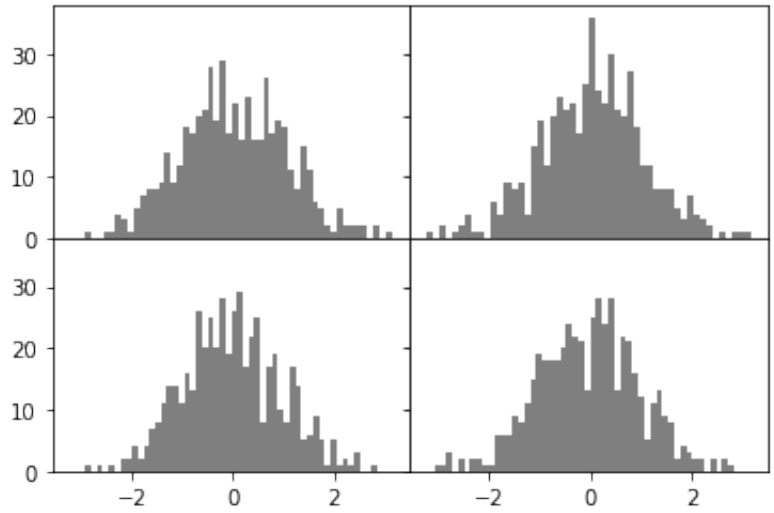
pyplot options

nrows -- Number of rows of subplots
ncols -- Number of cols of subplots
sharex -- All subplots should use the same x_axis ticks
sharey -- All subpots should use the same y-axis ticks
subplot_kw -- Dict of keywords passed to add_subplot call used to ceate each subplot
**fig_kw -- Additional keywords to subplots are used when creatin the figure, such as plt.subplots(2, 2, figsize=(8, 6))

Adjusting the Spacing Around Subplots

By defualt matlplotlib leaves a certain amount of padding around the outside of the subplots and spacing between plots. this can be resized using subplots_adjust method on Figure objects.

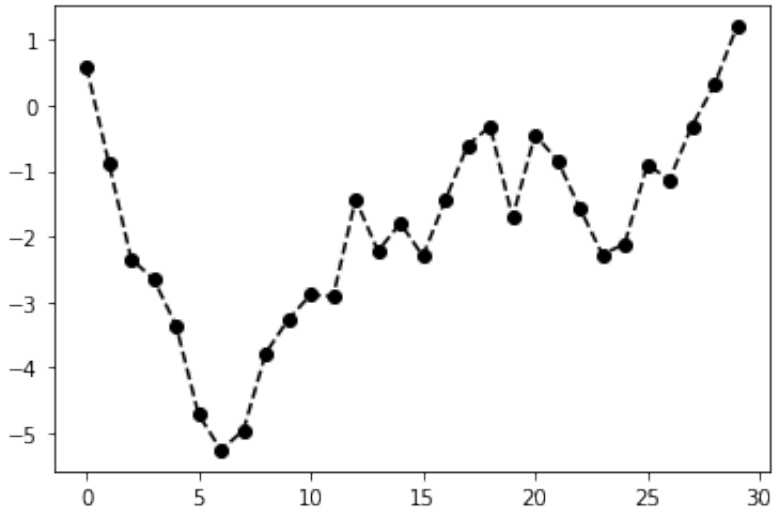
In [21]:
```
fig, axes = plt.subplots(2, 2, sharex=True, sharey=True)
for i in range(2):
    for j in range(2):
        axes[i,j].hist(np.random.randn(500), bins=50, color='k', alpha
plt.subplots_adjust(wspace=0, hspace=0) #hspace and wspace ontrols the
```



Colors, Markers, and Line Styles

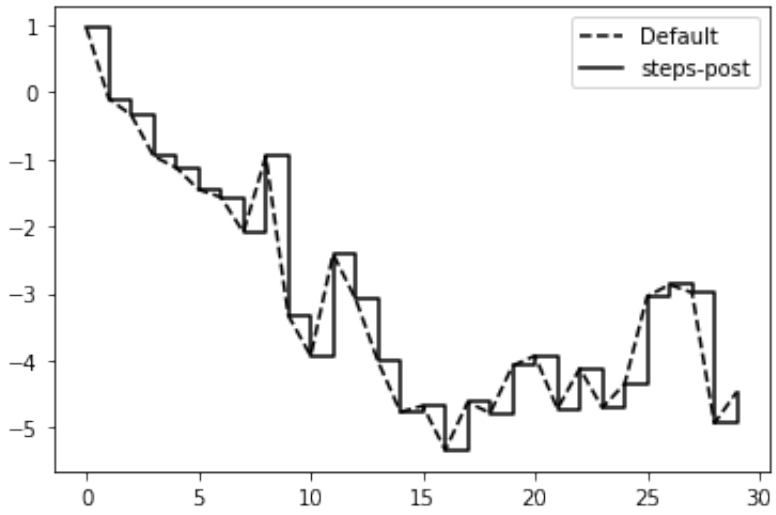
In [22]:
```
from numpy.random import randn
plt.plot(randn(30).cumsum(), 'ko--')
```

Out[22]:
```
[<matplotlib.lines.Line2D at 0x17e4f981a60>]
```



In [28]:
```
data = np.random.randn(30).cumsum()
plt.plot(data, 'k--', label='Default')
plt.plot(data, 'k-', drawstyle='steps-post', label='steps-post')
plt.legend(loc='best')
```

Out[28]:
```
<matplotlib.legend.Legend at 0x17e4d276160>
```



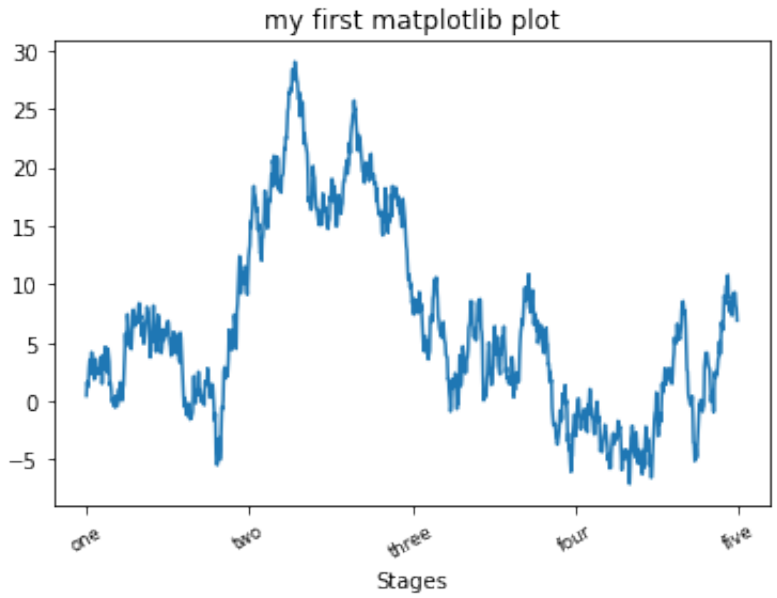
Ticks, Labels and Legends

The pyplot interface consists of methods such as xlim, sticks and xticklabels. Thes control the plot range, tick locations, and tick labels respectively. They can be used in two ways.

- Called with no arguments returns the current parameter values
- called with parameters sets the parameter value

In [39]:
```
fig = plt.figure()
ax = fig.add_subplot(1,1, 1)
ax.plot(np.random.randn(1000).cumsum())
ticks = ax.set_xticks([0,250,500,750,1000])
labels = ax.set_xticklabels(['one', 'two', 'three', 'four', 'five'], r
# The rotation option sets the x tick labels at a 30-degree rotation
# set_xlabel gives a name to the x-axis and set_title the subplot titl
ax.set_title('My first matplotlib plot')
ax.set_xlabel('stages')
# Modifying the y axis is a similar process
props = {
    'title': 'my first matplotlib plot',
    'xlabel': 'Stages'
}
ax.set(**props)
```

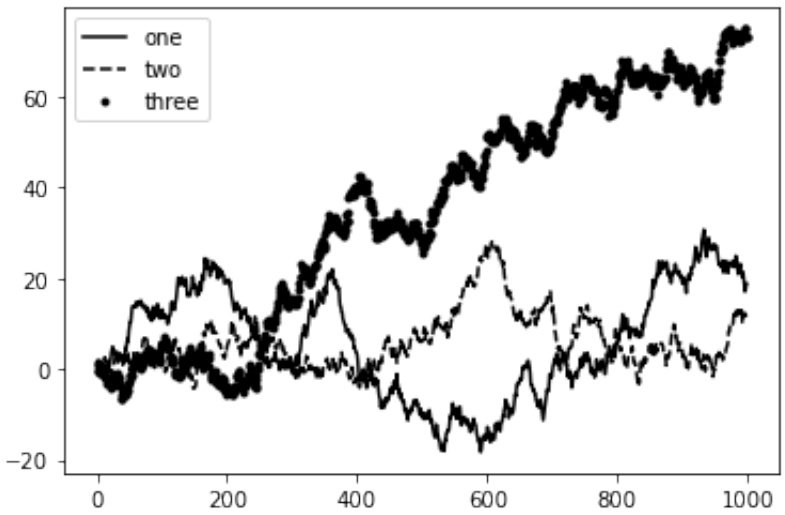
Out[39]:
```
[Text(0.5, 0, 'Stages'), Text(0.5, 1.0, 'my first matplotlib plot')]
```



Adding lengends

In [44]:
```
from numpy.random import randn
fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)
ax.plot(randn(1000).cumsum(), 'k', label='one')
ax.plot(randn(1000).cumsum(), 'k--', label='two')
ax.plot(randn(1000).cumsum(), 'k.', label='three')
ax.legend(loc='best')
```

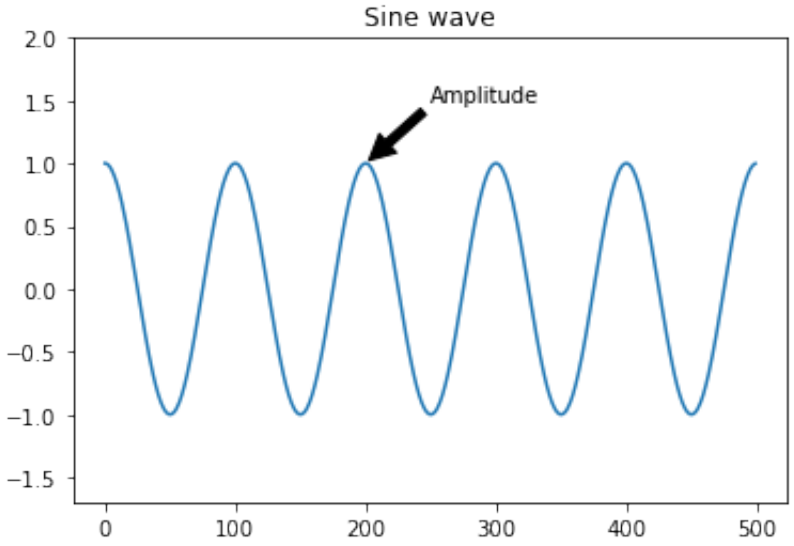
Out[44]:
```
<matplotlib.legend.Legend at 0x17e4f9c4b80>
```



Annotationas and Drawing on a Subplot

in addition to plot types you may whish to draw plot annotations e.g. arrows, text and annotate. below is a plot of the 2007 crash

In [40]:
```
from datetime import datetime
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure() #create the figure
ax = fig.add_subplot(1,1,1) #add one plot to the figure
t = np.arange(0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
ax.plot(s)
ax.annotate('Amplitude', xy=(200, 1), xytext=(250, 1.5), arrowprops=di
ax.set_ylim([-1.7, 2.00])
ax.set_title('Sine wave')
plt.savefig('examplefigures/figpath.svg')
```



Saving Plots to Files

In [53]:
```
fig = plt.figure()
ax = fig.add_subplot()
ax.plot(np.arange(10))
ax.set_title('Test Graph')
# To get the same plot as a PNG with minimal whitespace around the plo
plt.savefig('examplefigures/test_graph.png', dpi=400, bbox_inces='tigh
```

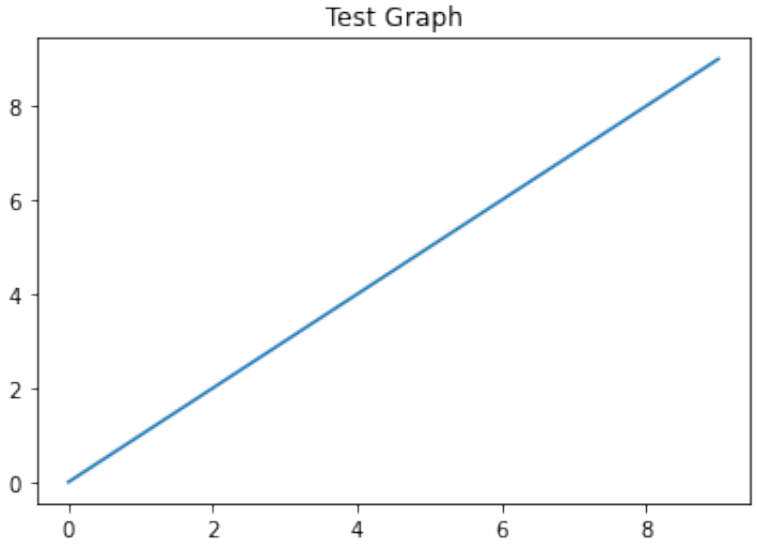


Figure.savefig options

fname -- String containing a filepath or a python file-like object.
dpi -- The figure resolution in dots per inch, defaults to 100
facecolor, edgecolor -- The color of the figure background outside the subplots'w' (white) by defaults
format -- The explicit file format to use('png', 'pdf', 'svg', 'ps', 'eps)

matplotlib Configuration

matplotlib comes configured with color schemes and defaults that are geared primarily towards preparing figures for publications. Fortunately you can edit all the default behaviour via an extensive set of global parameters. One way to modify it is to use rc.

In [55]:
```
plt.rc('figure', figsize=(10,10))
```

the first argument to rc is the component you which to customize then a sequence of keyword arguments indicating the new parameters. An easy way to write them is as a dict

In [58]:
```
font_options  = {'family': 'monospace',
                 'weight': 'bold',
                 'size': '12'}
plt.rc('font', **font_options)
```

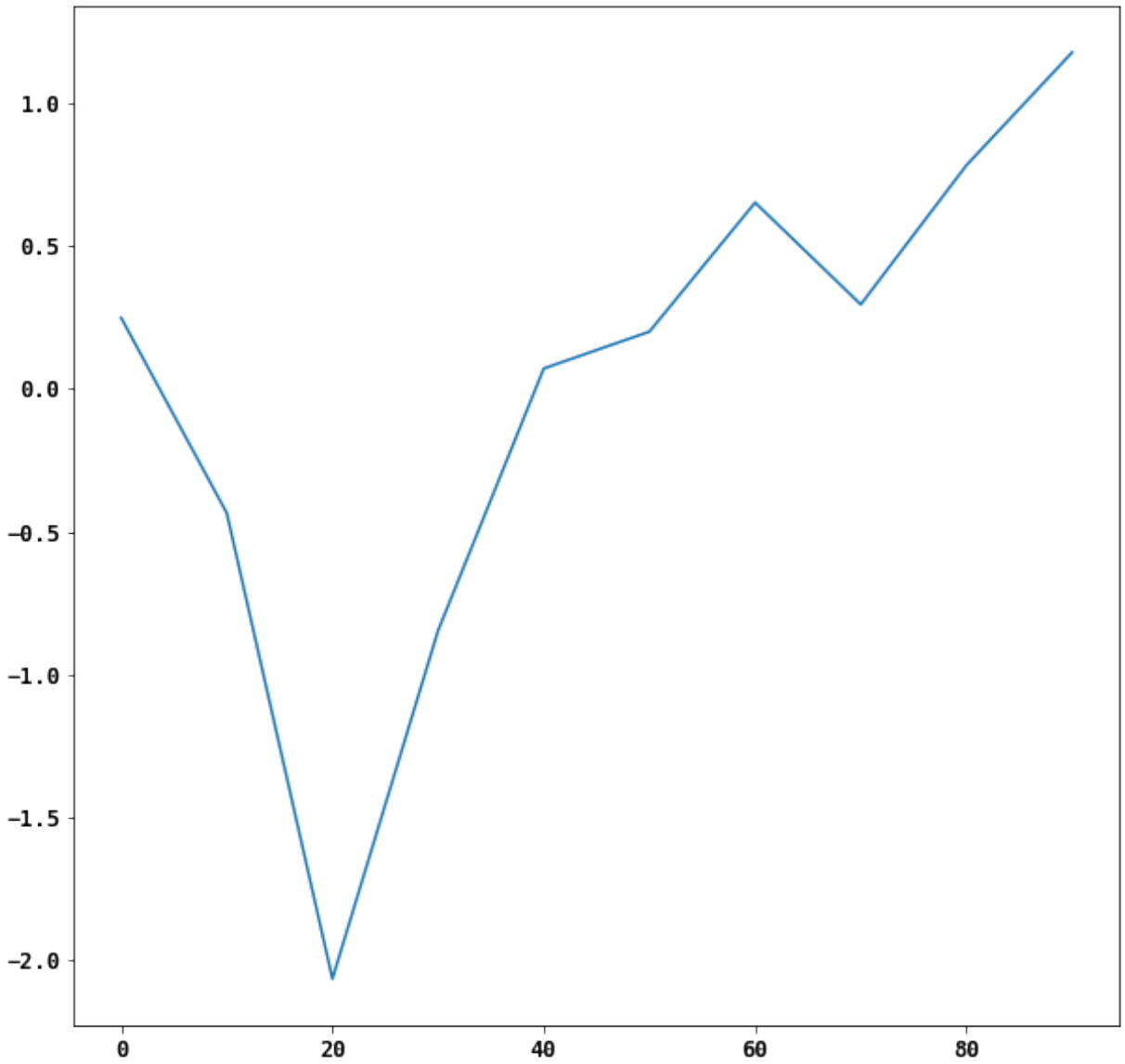
Plotting with pandas and seaborn

importing seaborn alters matplotlibs default schemes. Generally making them less Boring! as matplotlib is fairly low level. Seaborn simplifie creating many common visualization types

Line Plots

In [60]:
```
s = pd.Series(np.random.randn(10).cumsum(), index=np.arange(0, 100, 10
s.plot()
```

Out[60]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2bb524fdd90>
```

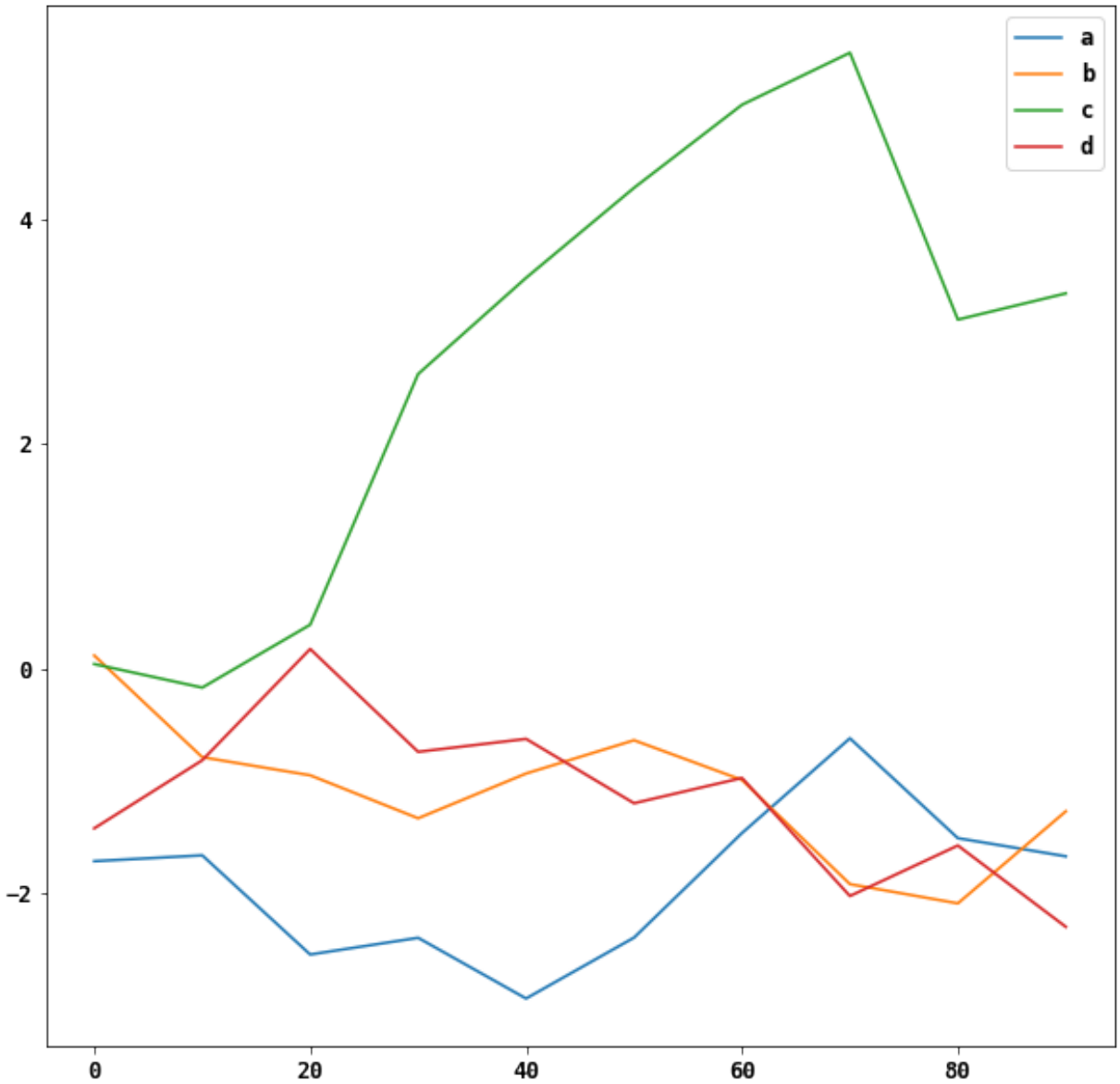


The series objects index is passed to matplot lib for plotting on the x-axis.

DataFrame's plot method plots each of its columns as a differnet line on the same subplot, creating a legend automatically

In [66]:
```
df = pd.DataFrame(np.random.randn(10, 4).cumsum(0), columns=['a', 'b',
df.plot()
```

Out[66]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2bb52e34910>
```



Series.plot method arguments

label -- label for plot legend ax -- matplotlib subplot object to plot on; if nothing s passed, uses active matplotlib subplot
style -- style string, like 'k0--' to be passed to matplot lib
alpha -- The plot fill opacitiy
kind -- can be 'area', 'bar', 'barh', 'density', 'hist', 'kde', 'line', 'pie'
logy -- use logarithmic scaling on the y-ais
use-index -- use the object index for tick labels 0-- rot -- rotation of tick labels in degrees
xticks -- Values to use for x-axis ticks
yticks -- Vlaues to use for y-axs ticks
xlim -- x-axis limits ylim -- y-axis limits grid -- Display axis grid

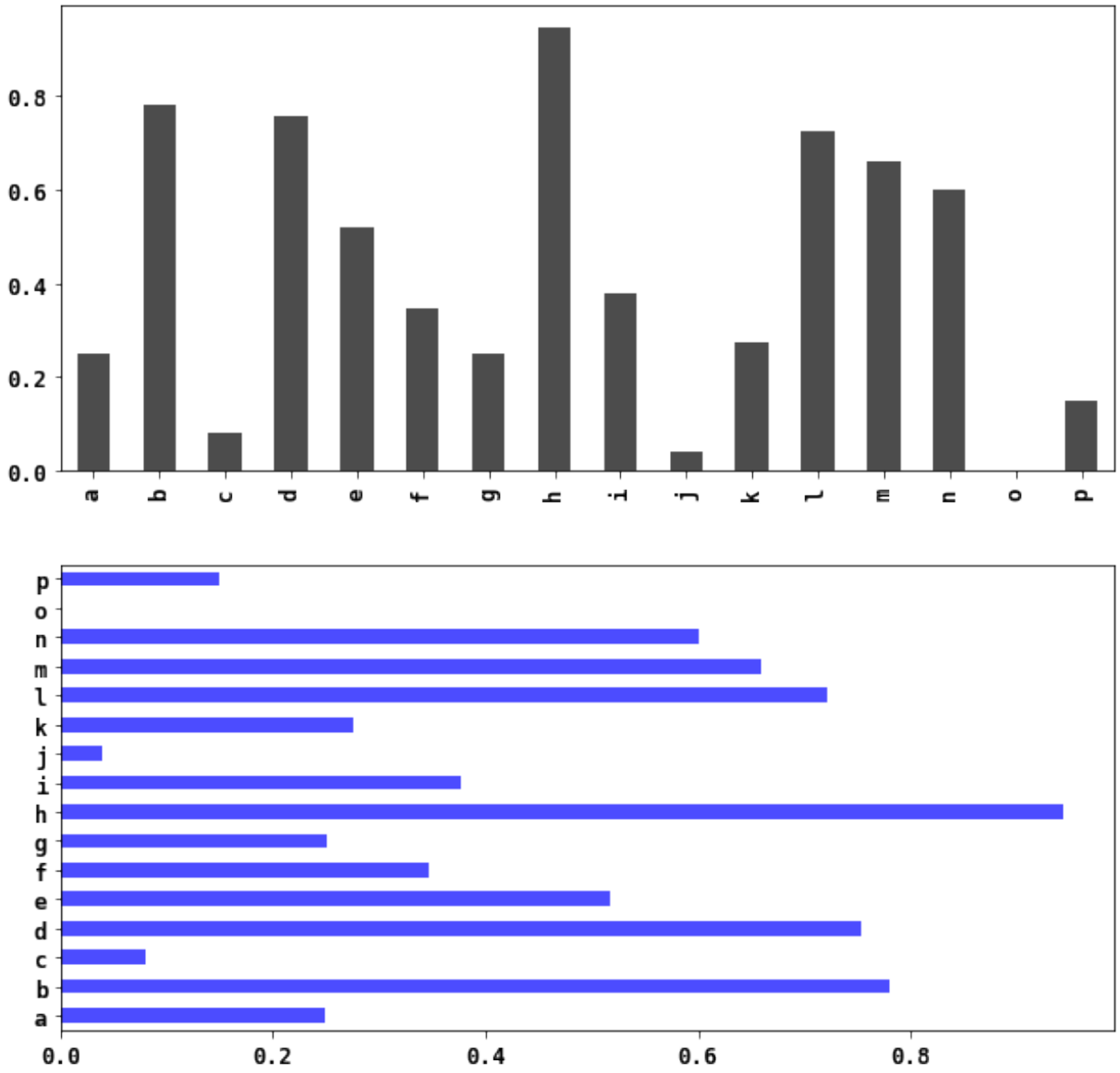
DataFrame specific plot arguments.

subplots -- Plot each Dataframe column in a seperate subplot
sharex -- if subplots=True, share the same x-axis, linking ticks and limits
sharey -- if subplots=True, share the smae y-axis
figsize -- size of figure to create as tuple
title -- Plot title as string
legend -- Add a subplot legend(true by default)
sort_columns -- Plot columns in alphabetical order; by default uses existing column order

Bar Plots

In [71]:
```
#plot.bar() and plot.barh() make vertical and horizontal bar plots
fig, axes = plt.subplots(2, 1)
data = pd.Series(np.random.rand(16), index=list('abcdefghijklmnop'))
data.plot.bar(ax=axes[0], color='k', alpha=0.7)
data.plot.barh(ax=axes[1], color='b', alpha=0.7)
```

Out[71]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2bb52558070>
```



With DataFrames, bar plots group the valuesin each row together in a group of bars side by side.

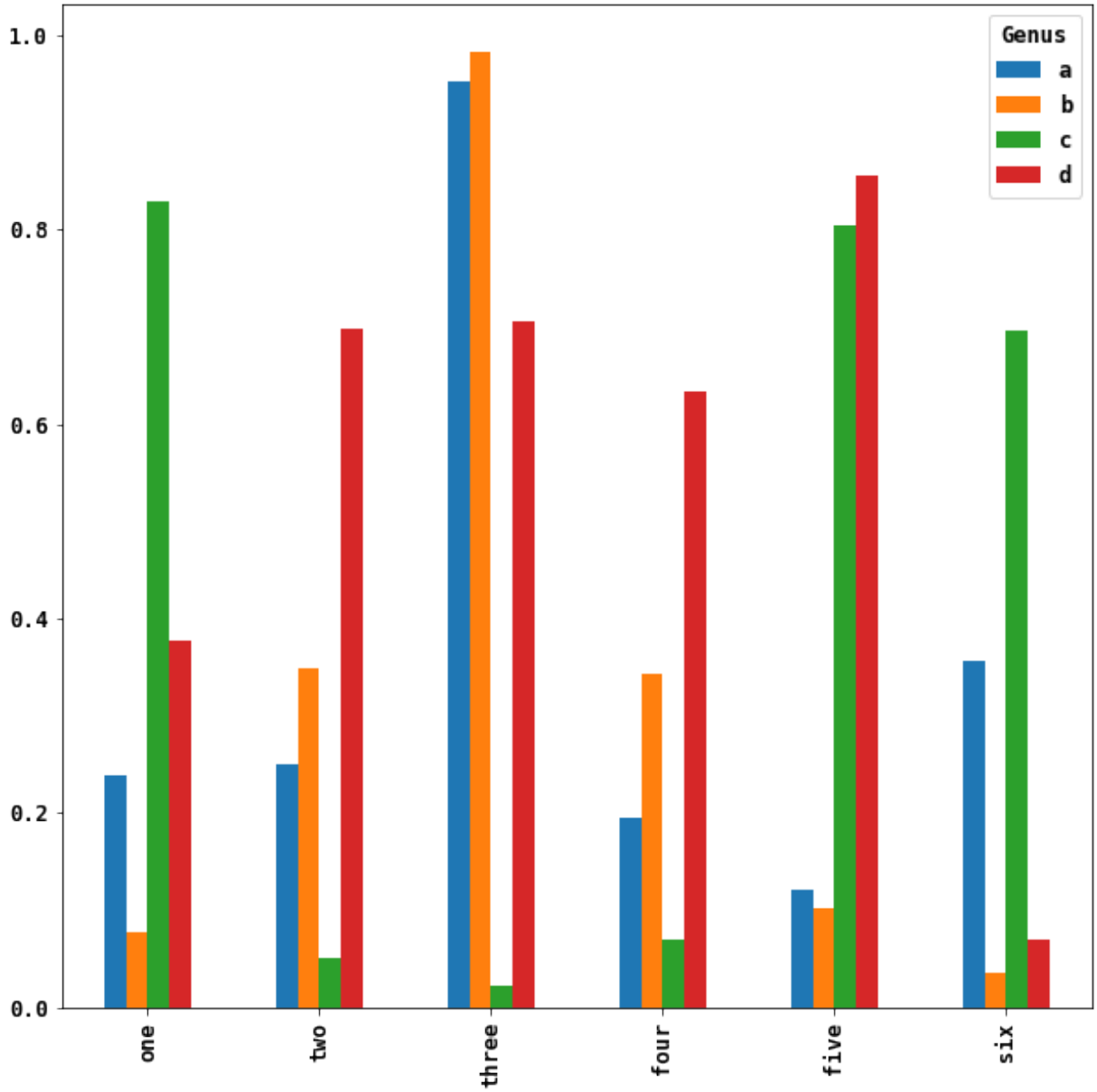
In [86]:
```
df = pd.DataFrame(np.random.rand(6, 4), index=['one', 'two', 'three',
df
```

Out[86]:

Genus	a	b	c	d
one	0.237560	0.076171	0.828325	0.376640
two	0.249386	0.349180	0.049642	0.698064
three	0.952891	0.982944	0.022680	0.705912
four	0.194681	0.343588	0.069240	0.632524
five	0.121594	0.101015	0.804863	0.855175
six	0.356628	0.036145	0.696381	0.069714

In [87]:
```
df.plot.bar()
```

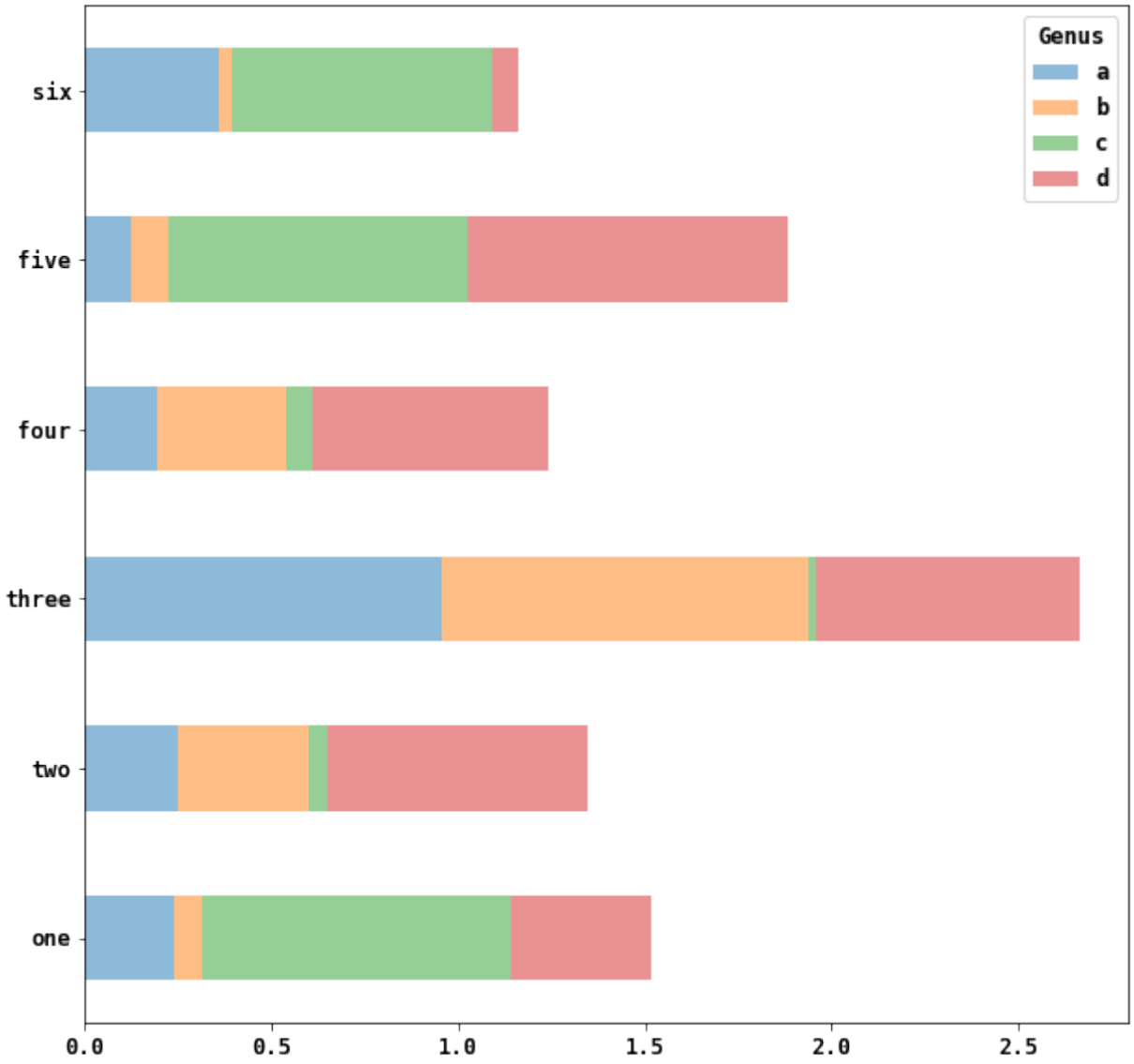
Out[87]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2bb4c905280>
```



Notice that the index name 'Genus' is used for the legend. We create stacked bar plots from a DataFrame by passing stacked=True

In [88]:
```
df.plot.barh(stacked=True, alpha=0.5)
```

Out[88]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2bb4d82dbe0>
```



In [142]:
```
home = pd.read_csv('exampledata/homes.csv', header=0)
home.head()
```

Out[142]:

	Sell	List	Living	Rooms	Beds	Baths	Age	Acres	Taxes
0	142	160	28	10	5	3	60	0.28	3167
1	175	180	18	8	4	1	12	0.43	4033
2	129	132	13	6	3	1	41	0.33	1471
3	138	140	17	7	3	1	22	0.46	3204
4	232	240	25	8	4	3	5	2.05	3613

In [168]:
```
beds = home['Sell']
rooms = home['Rooms'].unique()
rooms.sort()
rooms_price = {}
for i in rooms:
    rooms_price[str(i)] = home.loc[i].mean()

rooms_price
```

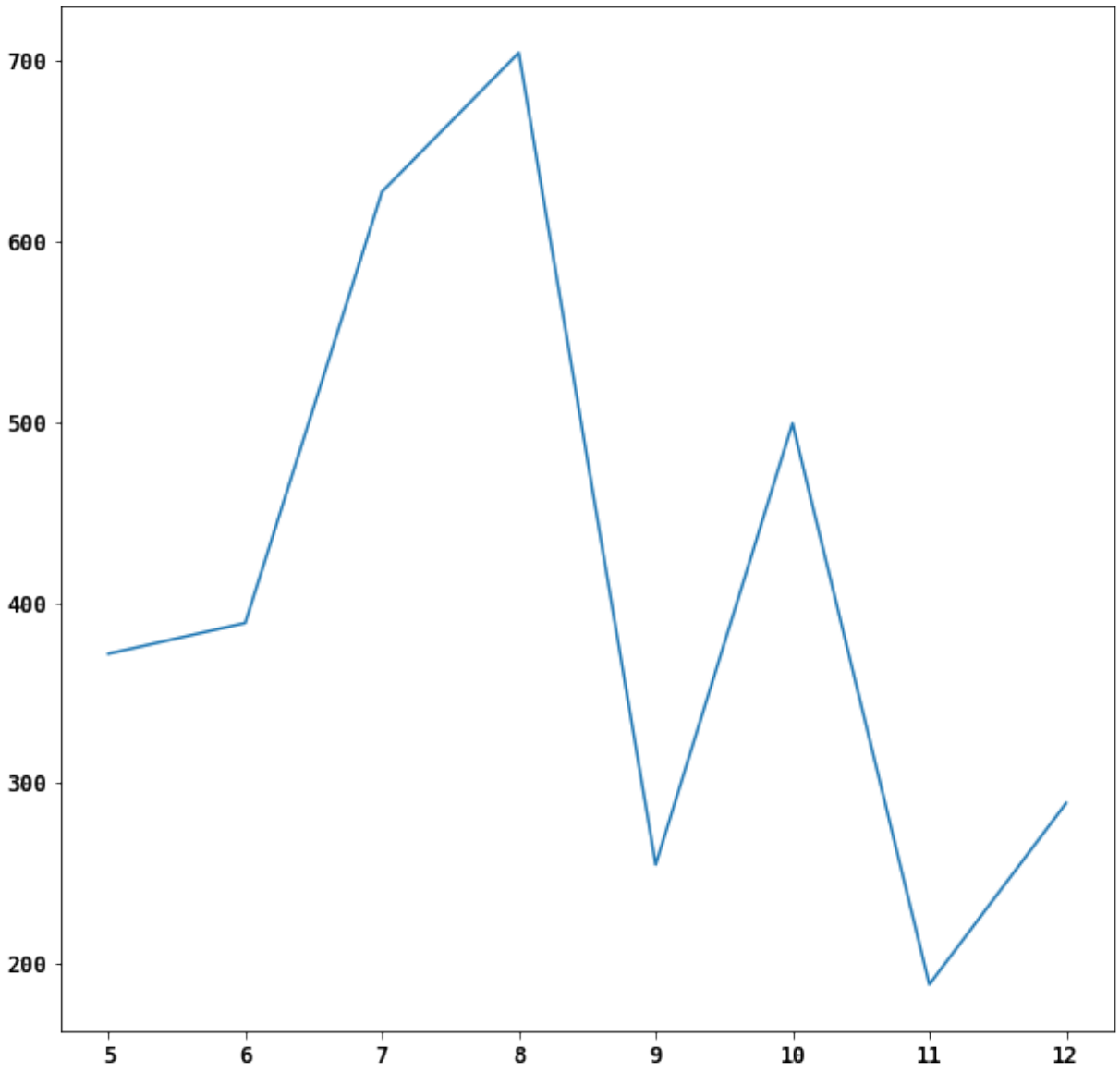
Out[168]:
```
{'5': 371.61888888888893,
 '6': 388.6666666666667,
 '7': 627.1355555555556,
 '8': 703.9477777777778,
 '9': 255.03333333333336,
 '10': 499.0422222222222,
 '11': 188.85000000000002,
 '12': 289.0677777777778}
```

In [171]:
```
s = pd.Series(rooms_price)
s
```

Out[171]:
```
5     371.618889
6     388.666667
7     627.135556
8     703.947778
9     255.033333
10    499.042222
11    188.850000
12    289.067778
dtype: float64
```

In [172]:
```
plt.plot(s)
```

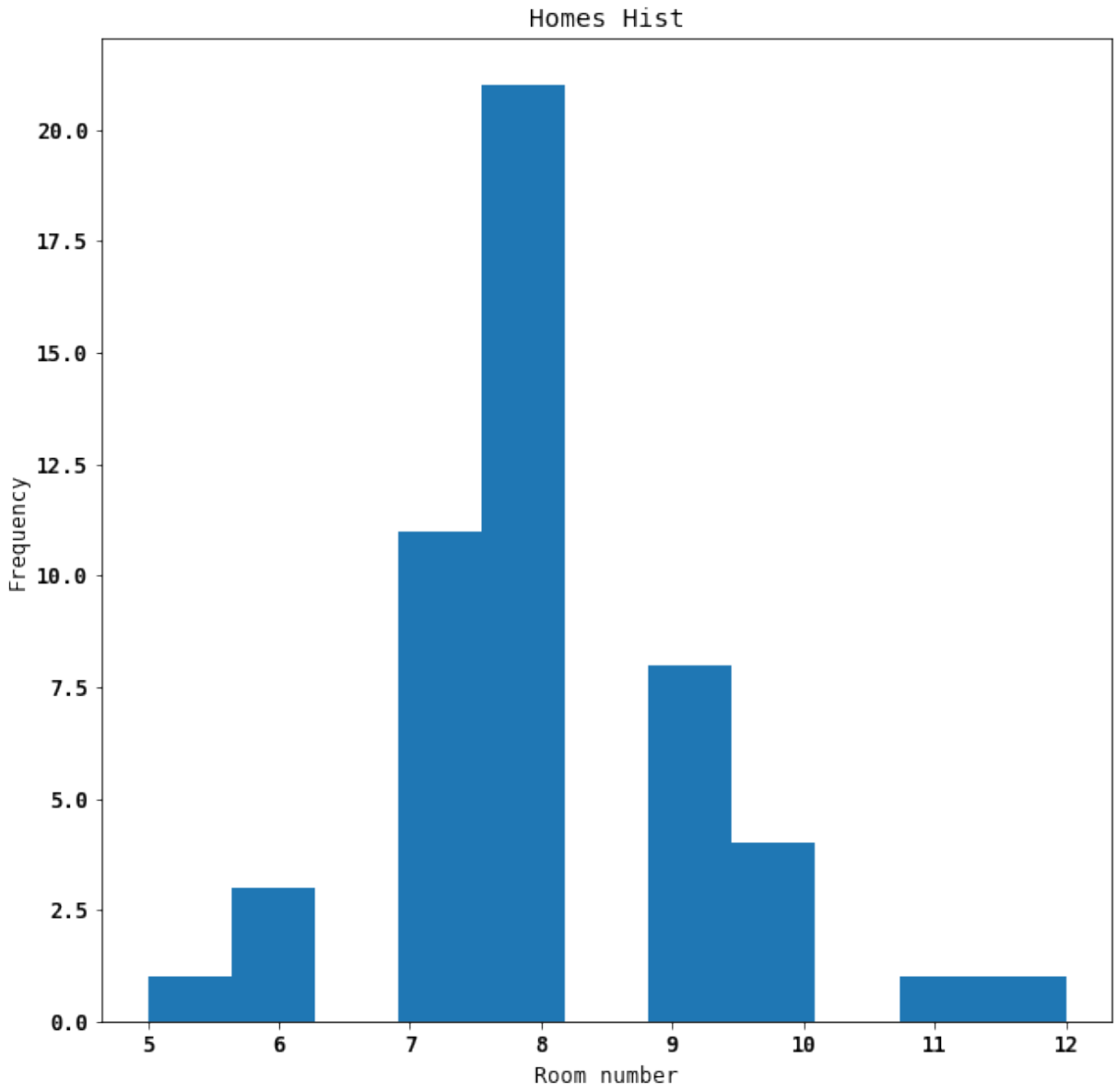
Out[172]:
```
[<matplotlib.lines.Line2D at 0x2bb51060730>]
```



Histograms and Density Plots

In [177]:
```
fig = plt.figure()
ax = fig.add_subplot()
ax = home['Rooms'].plot.hist(bins=11)
ax.set_xlabel('Room number')
ax.set_title('Homes Hist')
```

Out[177]:
```
Text(0.5, 1.0, 'Homes Hist')
```

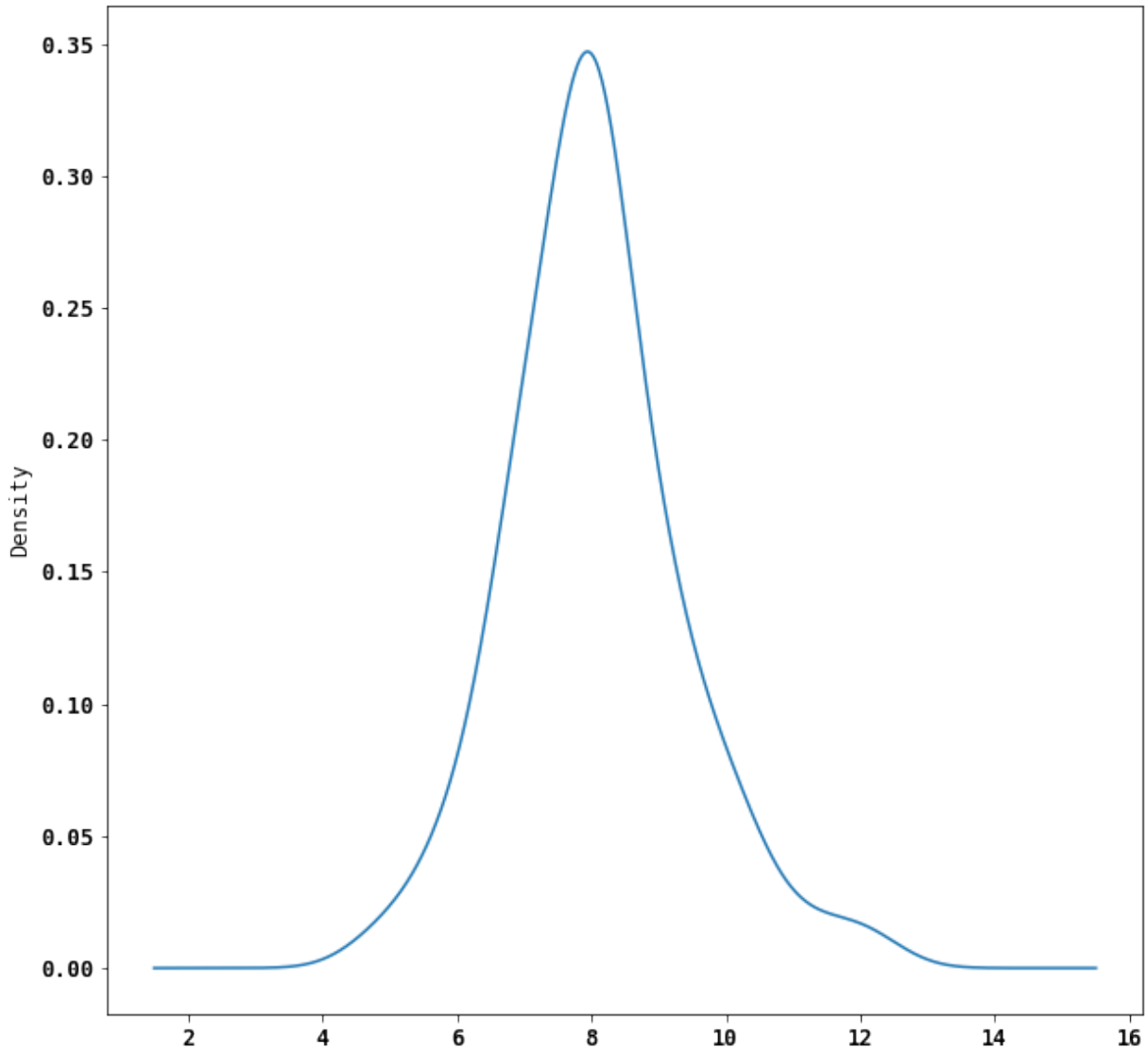


Density plot

it is formed by computing an estimate of a continuous probability distribution that might have generated the obsereved data. The usual procedure is to approximate this distibution as a mixture of "kernels" that is, simpler distributions like the normal distribution. Thus, density plots are also known as kernel density estimate plost. Using plot.kde makes a density plot using the conventional mixture-of-normals estimate.

In [180]:
```
home['Rooms'].plot.density()
```

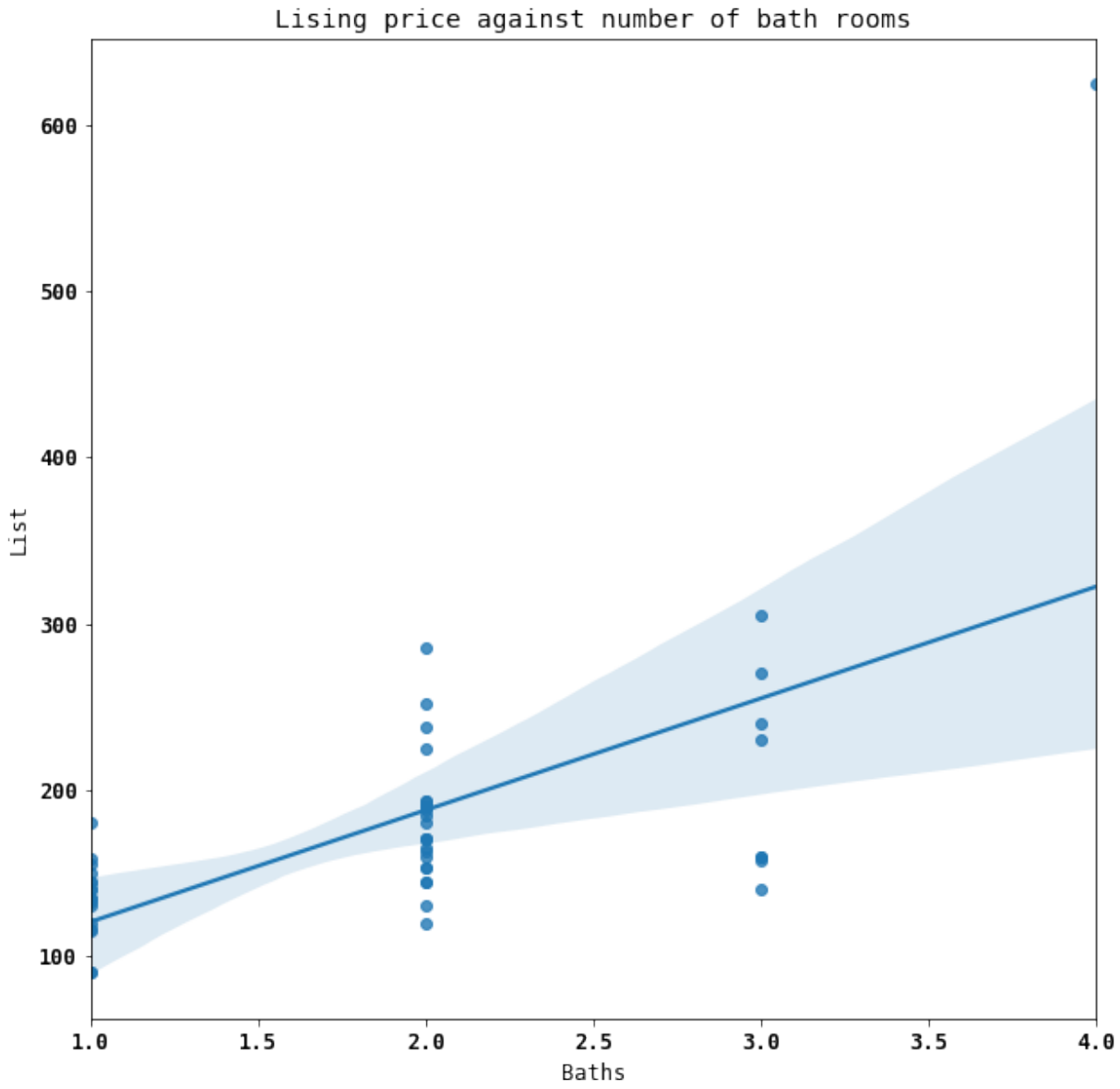
Out[180]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2bb544c0eb0>
```



Scatter Plots

In [198]:
```
data = home[['List', 'Baths']]
import seaborn as sns
sns.regplot('Baths', 'List', data=data)
plt.title('Lising price against number of bath rooms')
```

Out[198]: Text(0.5, 1.0, 'Lising price against number of bath rooms')



In [ ]:
```

```