In [1]:

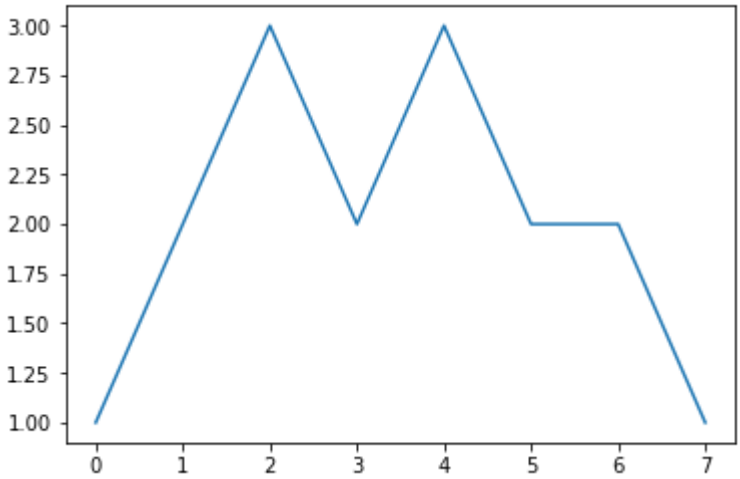
```
from matplotlib.pyplot import *
```

In [2]:

```
plot([1,2,3,2,3,2,2,1])
```

Out[2]:

```
[<matplotlib.lines.Line2D at 0x8622e10>]
```

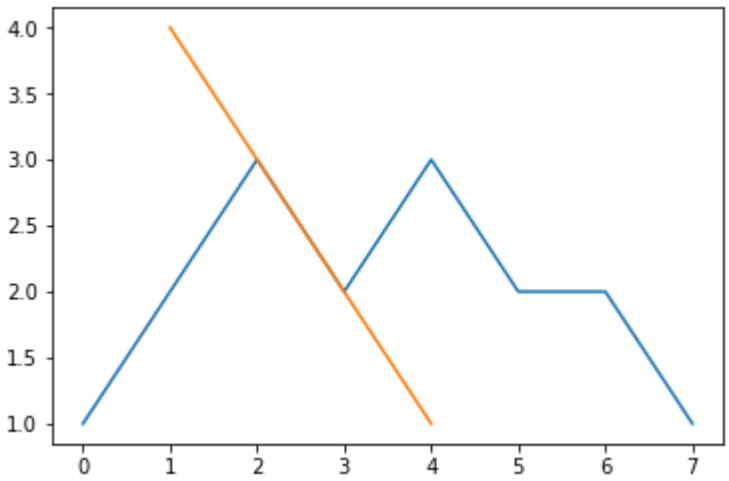


In [3]:

```
plot([1,2,3,2,3,2,2,1])
plot([4,3,2,1],[1,2,3,4])
```

Out[3]:

```
[<matplotlib.lines.Line2D at 0x88aef98>]
```



In [4]:

```
from matplotlib.pyplot import *
```

In [5]:

```
# some simple data
```

In [6]:

```
x = [1,2,3,4]
```

In [7]:

```
y = [5,4,3,2]
```

In [8]:

```
# create new figure
```

In [9]:

```
figure()
```

Out[9]:

```
<Figure size 432x288 with 0 Axes>
```

```
<Figure size 432x288 with 0 Axes>
```

In [10]:

```
# divide subplots into 2 x 3 grid
```

In [11]:

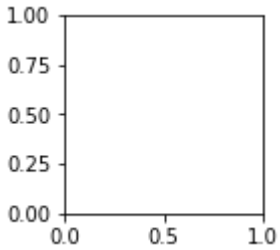
```
# and select #1
```

In [12]:

```
subplot(231)
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x8a7ae10>
```

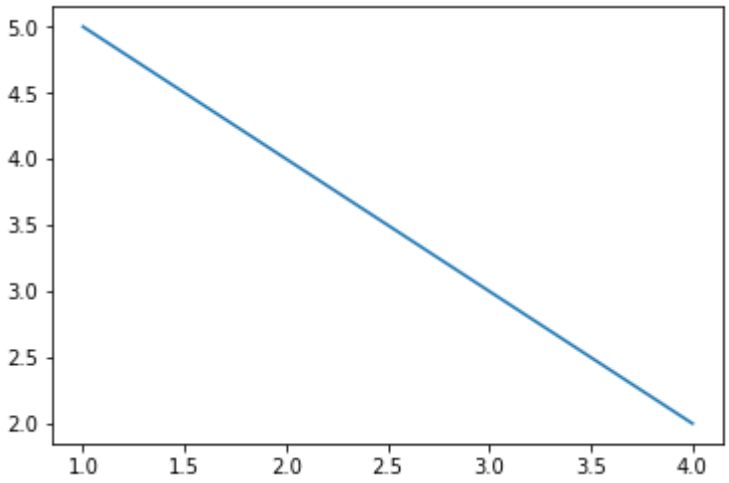


In [13]:

```
plot(x, y)
```

Out[13]:

```
[<matplotlib.lines.Line2D at 0x8c20080>]
```



In [14]:

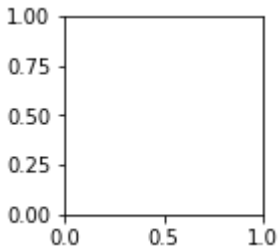
```
# select #2
```

In [15]:

```
subplot(232)
```

Out[15]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x8c5d438>
```

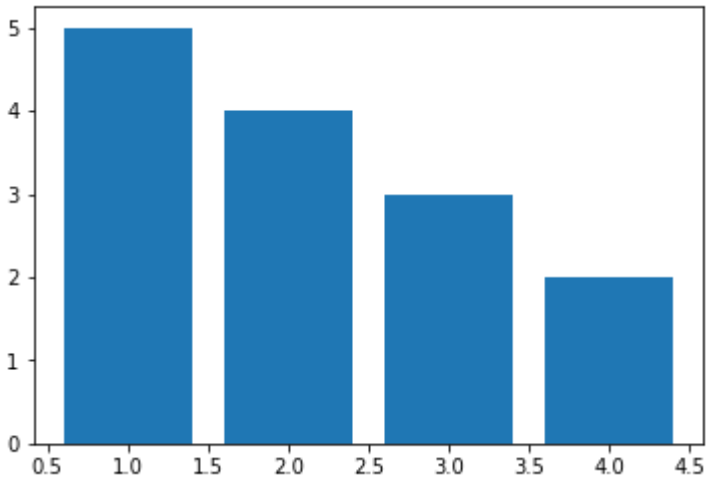


In [16]:

```
bar(x, y)
```

Out[16]:

<BarContainer object of 4 artists>



In [17]:

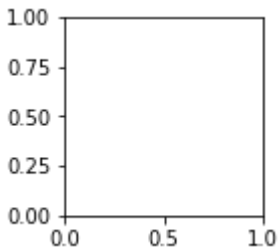
```
# horizontal bar-charts
```

In [18]:

```
subplot(233)
```

Out[18]:

<matplotlib.axes._subplots.AxesSubplot at 0x8ead630>

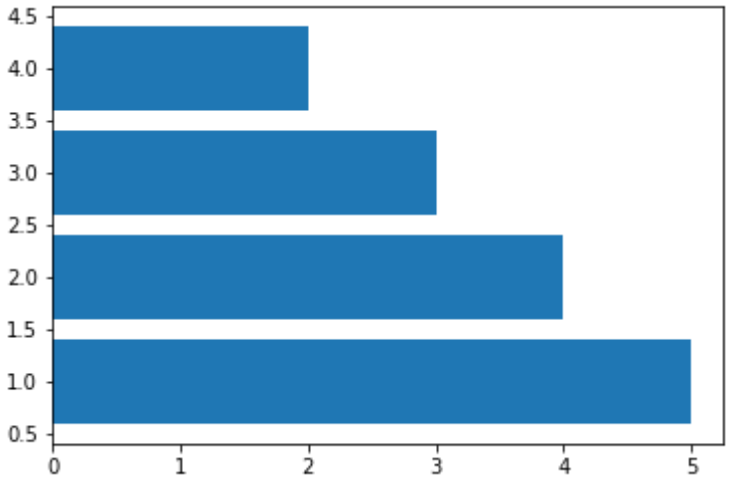


In [19]:

```
barh(x, y)
```

Out[19]:

<BarContainer object of 4 artists>



In [20]:

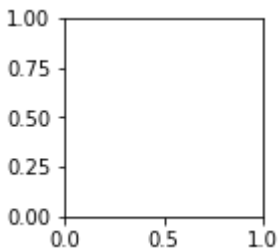
```
# create stacked bar charts
```

In [21]:

```
subplot(234)
```

Out[21]:

<matplotlib.axes._subplots.AxesSubplot at 0x8ead5f8>

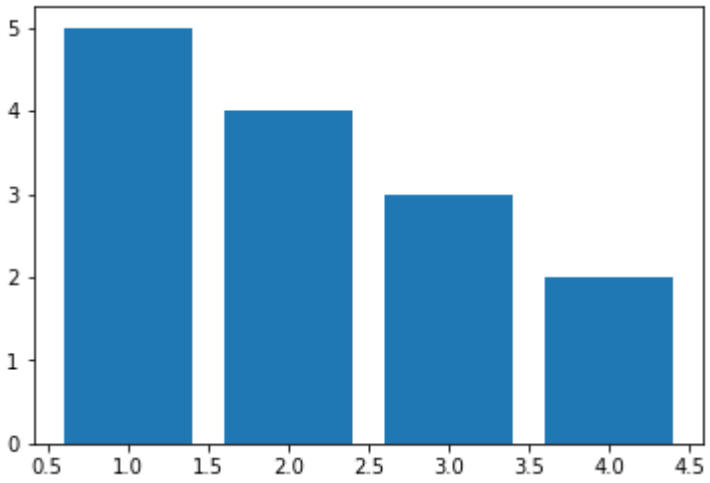


In [22]:

```
bar(x, y)
```

Out[22]:

<BarContainer object of 4 artists>



In [23]:

```
# we need nmore data for stacked bar charts
```

In [24]:

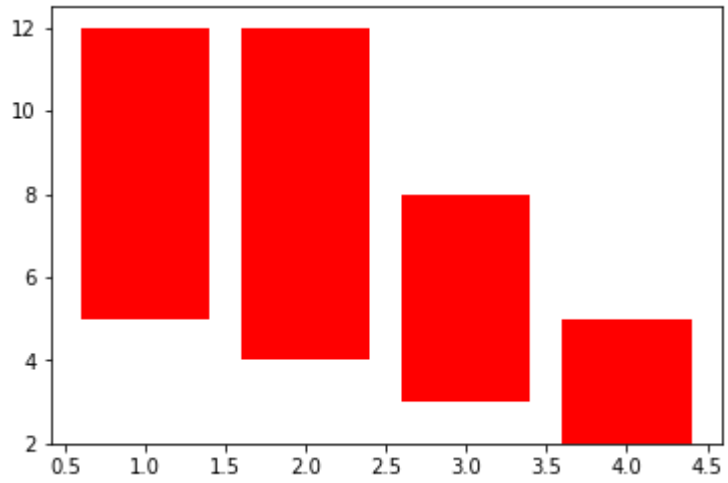
```
y1 = [7,8,5,3]
```

In [25]:

```
bar(x, y1, bottom=y, color = 'r')
```

Out[25]:

<BarContainer object of 4 artists>



In [26]:

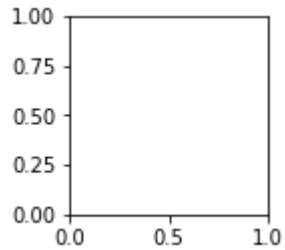
```
# box plot
```

In [27]:

```
subplot(235)
```

Out[27]:

<matplotlib.axes._subplots.AxesSubplot at 0x9e7a860>

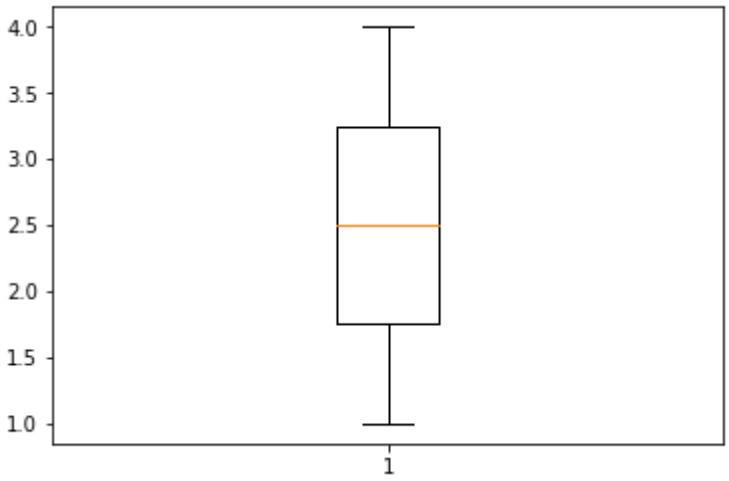


In [28]:

```
boxplot(x)
```

Out[28]:

```
{'boxes': [<matplotlib.lines.Line2D at 0x9f73278>],
 'caps': [<matplotlib.lines.Line2D at 0x9f73ac8>,
  <matplotlib.lines.Line2D at 0x9f73e48>],
 'fliers': [<matplotlib.lines.Line2D at 0x9f83588>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x9f83208>],
 'whiskers': [<matplotlib.lines.Line2D at 0x9f73320>,
  <matplotlib.lines.Line2D at 0x9f73780>]}
```



In [29]:

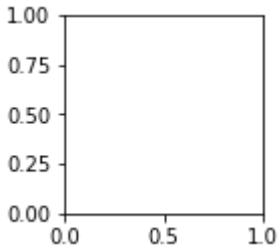
```
# scatter plot
```

In [30]:

```
subplot(236)
```

Out[30]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x9f8f7f0>
```

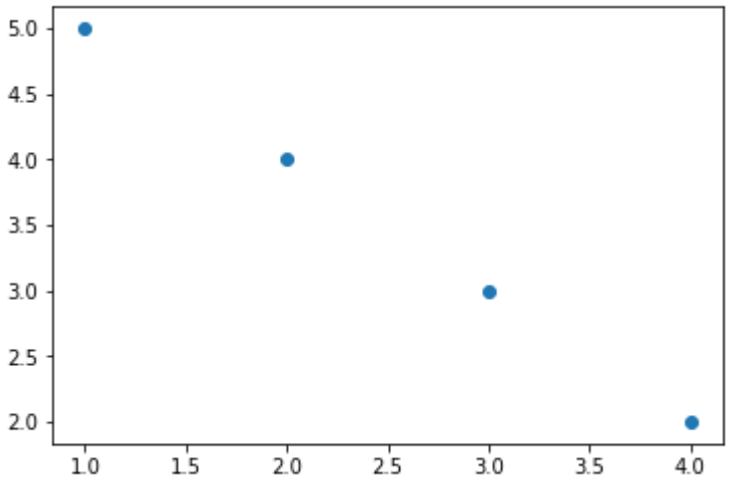


In [31]:

```
scatter(x, y)
```

Out[31]:

<matplotlib.collections.PathCollection at 0xa293668>



In [32]:

```
from pylab import *
```

In [33]:

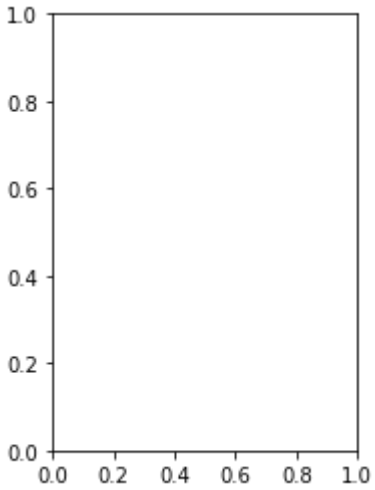
```
dataset = [113, 115, 119, 121, 124,
           124, 125, 126, 126, 126,
           127, 127, 128, 129, 130,
           130, 131, 132, 133, 136]
```

In [34]:

```
subplot(121)
```

Out[34]:

<matplotlib.axes._subplots.AxesSubplot at 0xa391908>

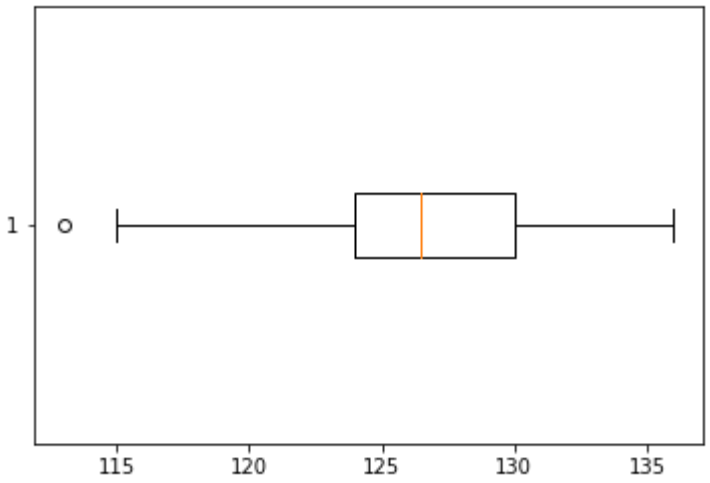


In [35]:

```
boxplot(dataset, vert=False)
```

Out[35]:

```
{'boxes': [<matplotlib.lines.Line2D at 0xa5dc160>],
 'caps': [<matplotlib.lines.Line2D at 0xa5dc9b0>,
  <matplotlib.lines.Line2D at 0xa5dcd30>],
 'fliers': [<matplotlib.lines.Line2D at 0xa5ec470>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0xa5ec0f0>],
 'whiskers': [<matplotlib.lines.Line2D at 0xa5dc208>,
  <matplotlib.lines.Line2D at 0xa5dc630>]}
```

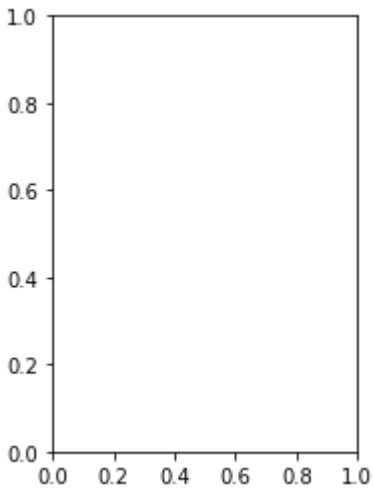


In [36]:

```
subplot(122)
```

Out[36]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xa605908>
```

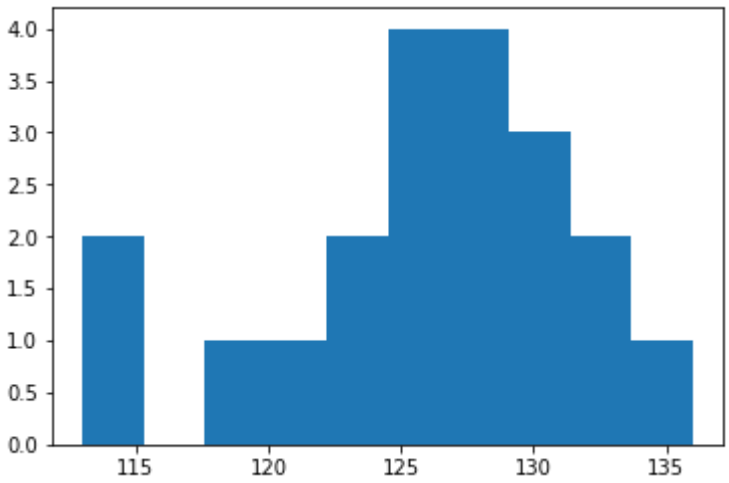


In [37]:

```
hist(dataset)
```

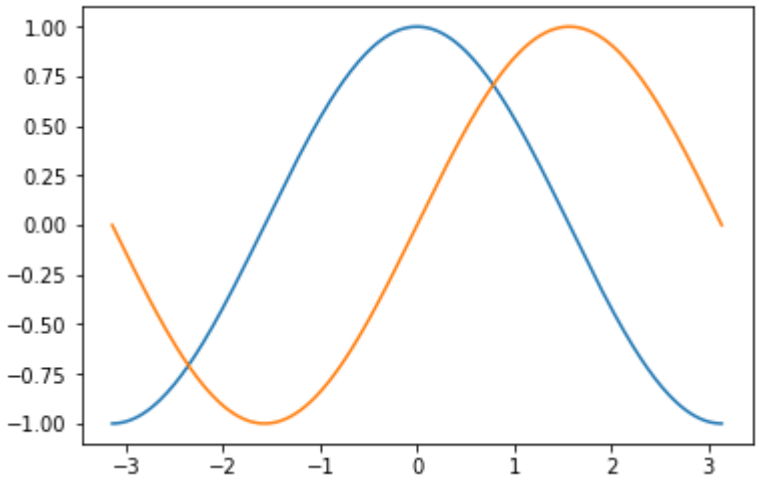
Out[37]:

```
(array([2., 0., 1., 1., 2., 4., 4., 3., 2., 1.]),
 array([113. , 115.3, 117.6, 119.9, 122.2, 124.5, 126.8, 129.1, 131.4,
        133.7, 136. ]),
 <a list of 10 Patch objects>)
```



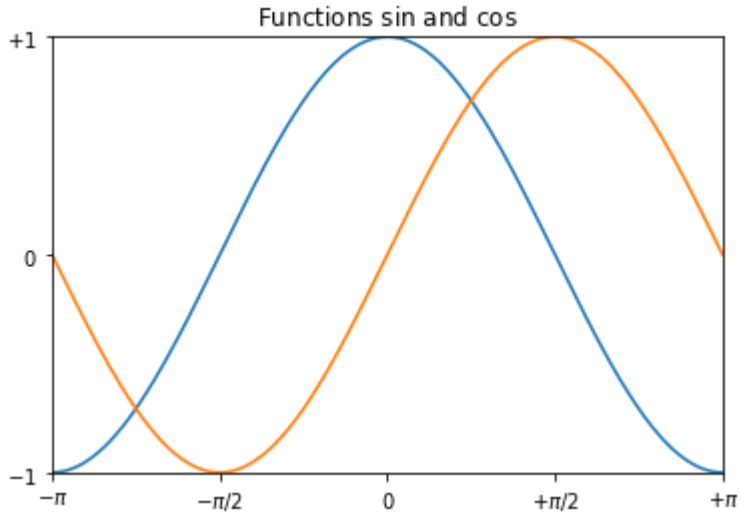
In [47]:

```
import matplotlib.pyplot as pl
import numpy as np

x = np.linspace(-np.pi, np.pi, 256, endpoint=True)

y = np.cos(x)
y1 = np.sin(x)

pl.plot(x,y)
pl.plot(x,y1)

pl.show()
```



In [48]:

```
from pylab import *
import numpy as np

#generate uniformly distributed
# 256 points from -pi to pi, inclusive
x = np.linspace(-np.pi, np.pi, 256, endpoint=True)


# these are vectorised versions
# of math.cos, and math.sin in built-in Python maths
# compute cos for every x
y = np.cos(x)

# compute sin for every x
y1 = np.sin(x)

# plot cos
plot(x, y)

# plot xin
plot(x, y1)

# define plot title
title("Functions $\sin$ and $\cos$")

# set x limit
xlim(-3.0, 3.0)
# set y limit
ylim(-1.0, 1.0)

# format ticks at specific values
xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
       [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
yticks([-1, 0, +1],[r'$-1$', r'$0$', r'$+1$'])

show()
```



In [49]:

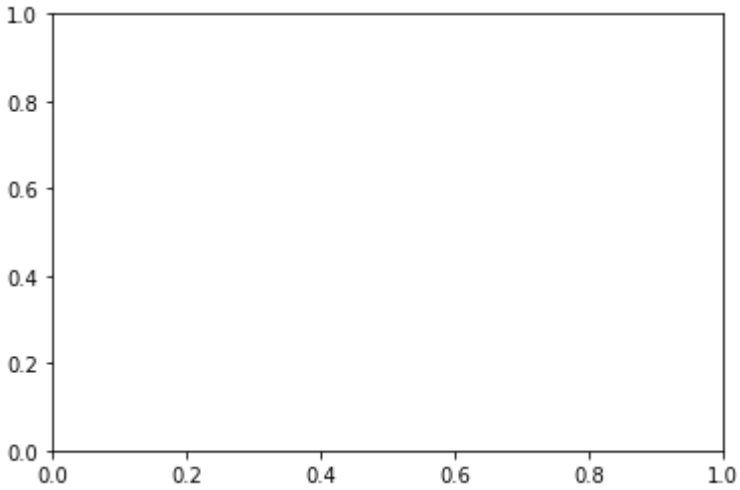
```
from matplotlib.pylab import *
```

In [50]:

```
axis()
```

Out[50]:

(0.0, 1.0, 0.0, 1.0)



In [52]:

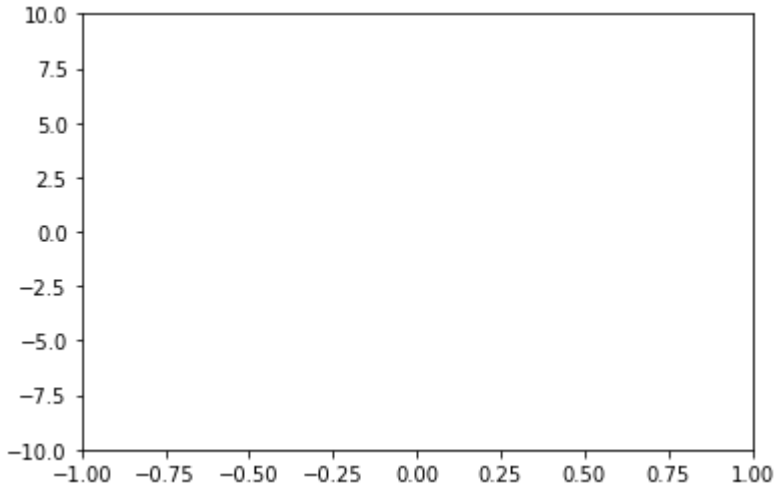
```
l = [-1, 1, -10, 10]
```

In [53]:

```
axis(l)
```

Out[53]:

[-1, 1, -10, 10]

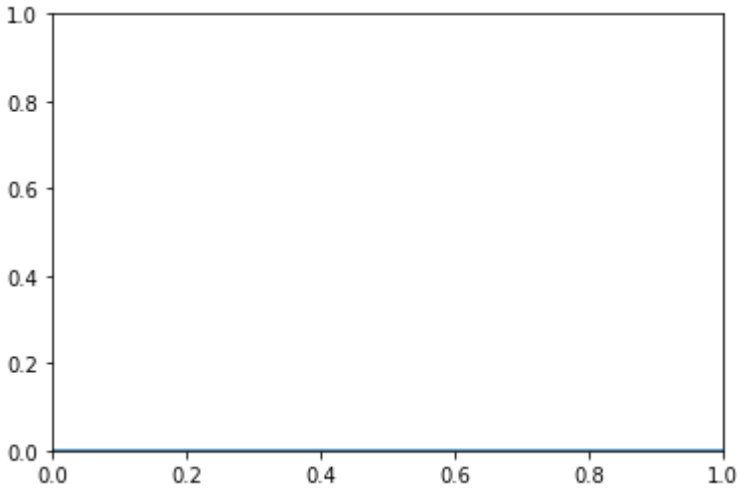


In [54]:

```
axhline()
```

Out[54]:

<matplotlib.lines.Line2D at 0xb7dbb70>

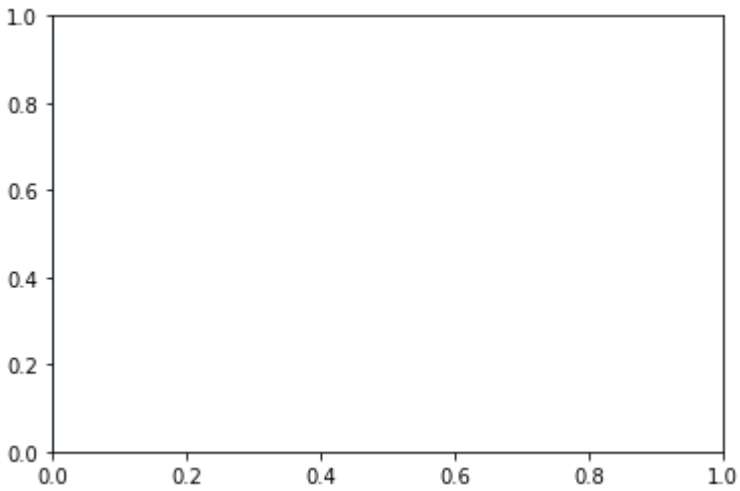


In [55]:

```
axvline()
```

Out[55]:

<matplotlib.lines.Line2D at 0xb6ef3c8>

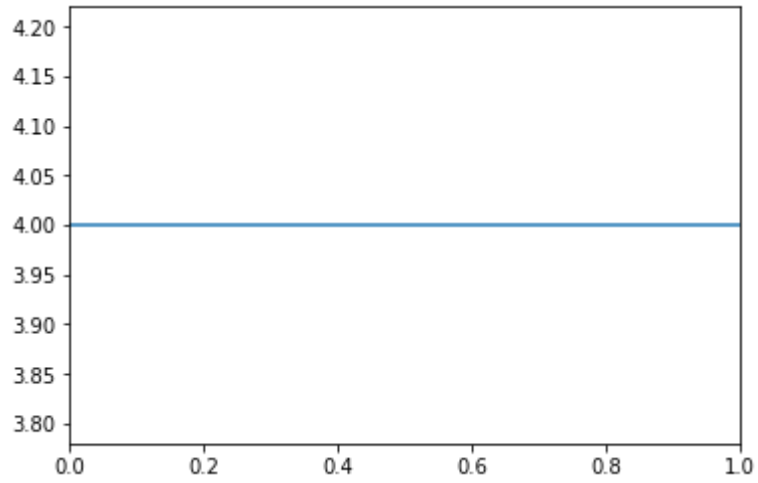


In [56]:

```
axhline(4)
```

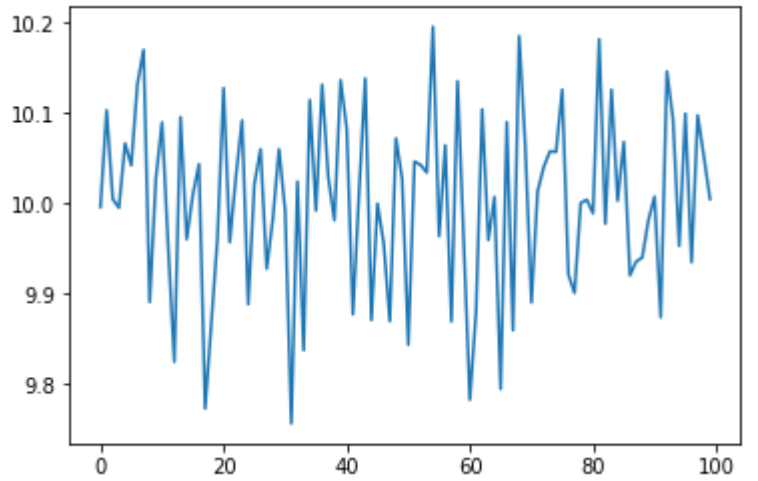
Out[56]:

<matplotlib.lines.Line2D at 0xae08a90>



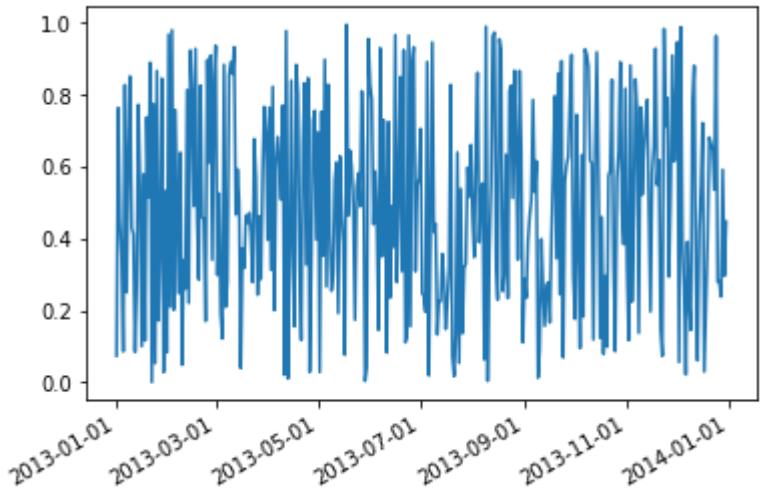
In [57]:

```
from pylab import *

# get current axis
ax = gca()

# set view to tight, and maximum number of tick intervals to 10
ax.locator_params(tight=True, nbin = 10)

# generate 100 normal distribution values
ax.plot(np.random.normal(10, .1, 100))

show()
```



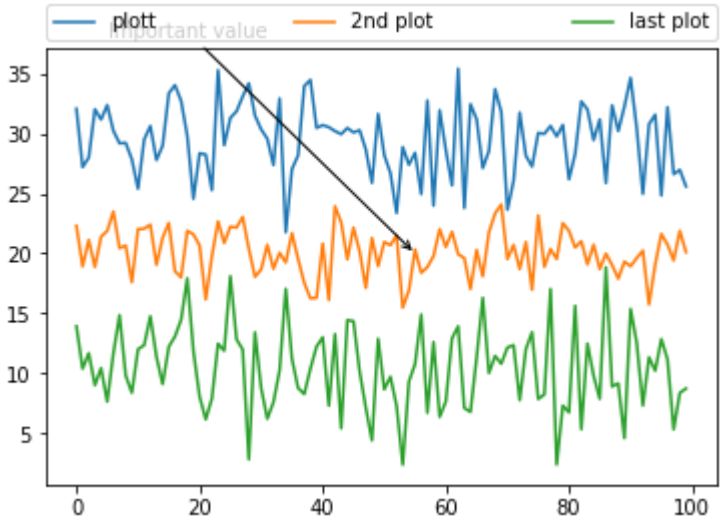
In [58]:

```
from pylab import *
import matplotlib as mpl
import datetime

fig = figure()

# get current axis
ax = gca()

# set some daterange
start = datetime.datetime(2013, 01, 01)
stop = datetime.datetime(2013, 12, 31)
delta = datetime.timedelta(days = 1)

# convert dates for matplotlib
dates = mpl.dates.drange(start, stop, delta)

# generate some random values
values = np.random.rand(len(dates))

ax = gca()

# create plot with dates
ax.plot_date(dates, values, linestyle='-', marker='')

# specify formater
date_format = mpl.dates.DateFormatter('%Y-%m-%d')

# apply formater
ax.xaxis.set_major_formatter(date_format)

# autoformat date labels
# rotates lables by 30 degrees by default
# use rotate param to specify different ratation degree
# use bottom parram to gice more room to date labels
fig.autofmt_xdate()

show()
```



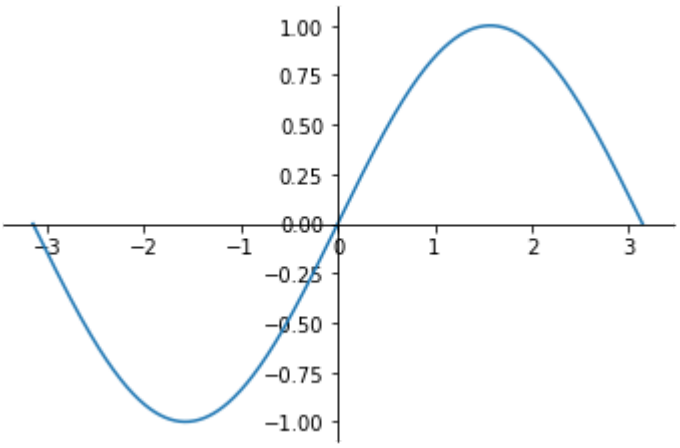
In [61]:

```
from matplotlib.pyplot import *

# generate different normal distributions
x1 = np.random.normal(30, 3, 100)
x2 = np.random.normal(20, 2, 100)
x3 = np.random.normal(10, 3, 100)

# plot them
plot(x1, label='plott')
plot(x2, label='2nd plot')
plot(x3, label='last plot')

# generate a legend box
legend(bbox_to_anchor=(0., 1.02, 1., .102),loc=3,
       ncol=3, mode="expand", borderaxespad=0.)

# annotate an important value
annotate("Important value", (55,20), xycoords='data',
         xytext=(5, 38),
         arrowprops=dict(arrowstyle='->'))

show()
```



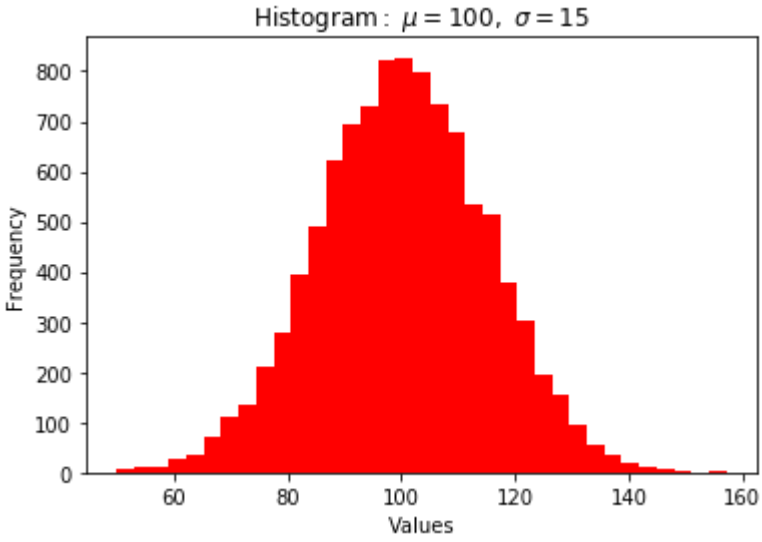
In [62]:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-np.pi, np.pi, 500, endpoint=True)
y = np.sin(x)

plt.plot(x, y)

ax = plt.gca()

# hide two spines
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')

# move bottom and left spine to 0,0
ax.spines['bottom'].set_position(('data',0))
ax.spines['left'].set_position(('data',0))

# move ticks positions
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')

plt.show()
```



In [63]:

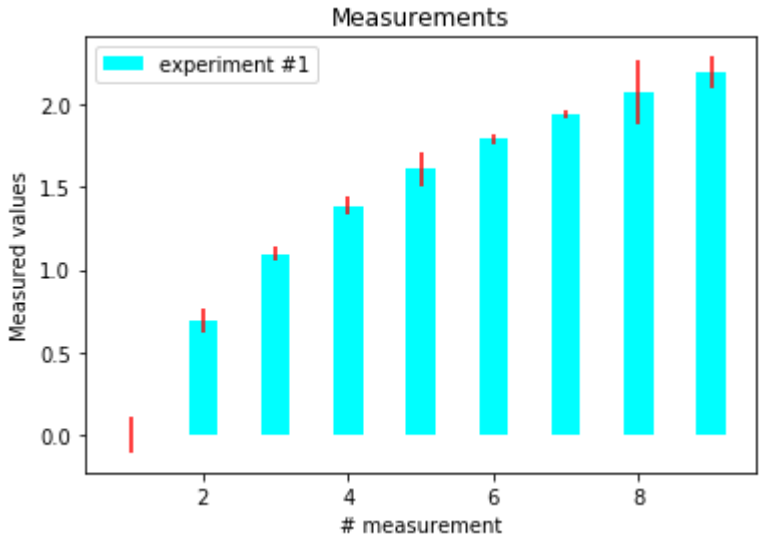
```
import numpy as np
import matplotlib.pyplot as plt

mu = 100
sigma = 15
x = np.random.normal(mu, sigma, 10000)

ax = plt.gca()

# the histogram of the data
ax.hist(x, bins=35, color='r')

ax.set_xlabel('Values')
ax.set_ylabel('Frequency')

ax.set_title(r'$\mathrm{Histogram:}\ \mu=%d,\ \sigma=%d$' % (mu, sigma))

plt.show()
```



In [65]:

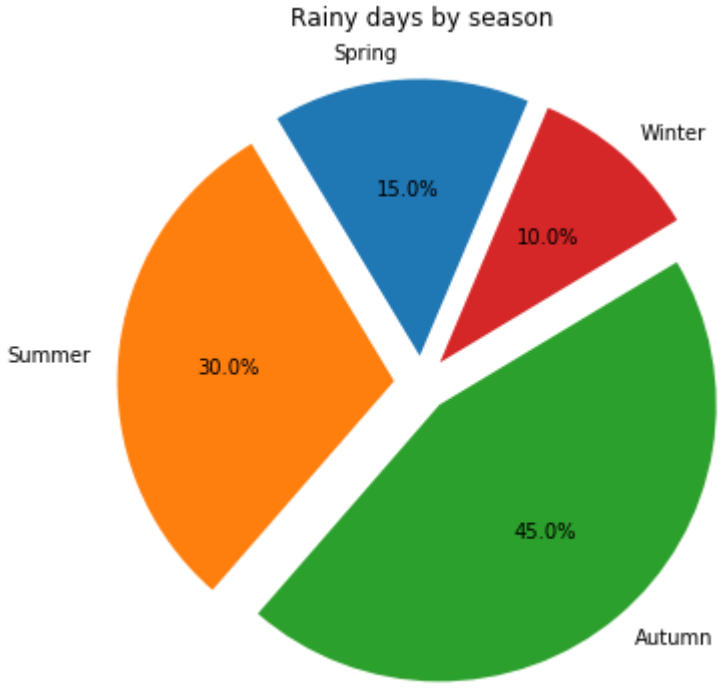
```
import numpy as np
import matplotlib.pyplot as plt

# generate number of measurements
x = np.arange(0, 10, 1)

# values computed from "measured"
y = np.log(x)

# add some error samples from standard normal distribution
xe = 0.1 * np.abs(np.random.randn(len(y)))

# draw and show errorbar
plt.bar(x, y, yerr=xe, width=0.4, align='center', ecolor='r',
        color='cyan', label='experiment #1');

# give some explainations
plt.xlabel('# measurement')
plt.ylabel('Measured values')
plt.title('Measurements')
plt.legend(loc='upper left')

plt.show()
```

```
D:\Anaconda\lib\site-packages\ipykernel_launcher.py:8: RuntimeWarning: divide by zer
o encountered in log
```



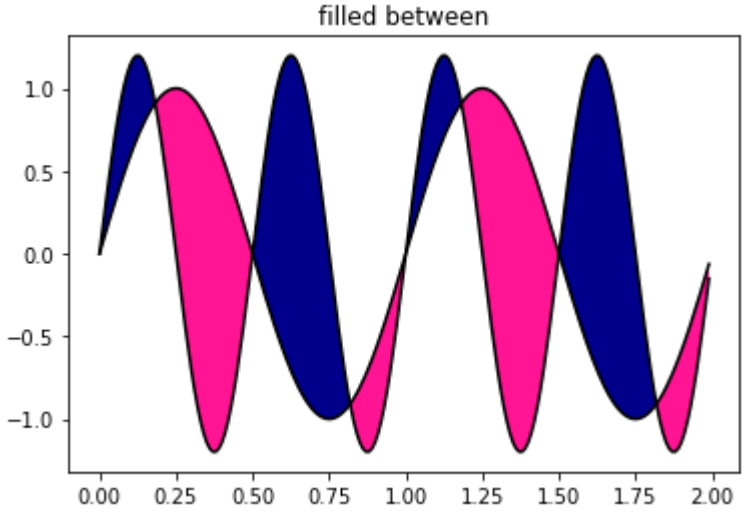
In [67]:

```
from pylab import *

# make a square figure and axes
figure(1, figsize=(6,6))
ax = axes([0.1, 0.1, 0.8, 0.8])

# the slices will be ordered
# and plotted counter_clockwise.
labels = 'Spring', 'Summer', 'Autumn', 'Winter'

# fractions are either x/sum(x) or x if sum(x) <= 1
x = [15, 30, 45, 10]

# explode must be len(x) sequence or None
explode=(0.1, 0.1, 0.1, 0.1)

pie(x, explode=explode, labels=labels,
   autopct='%1.1f%%', startangle=67)

title('Rainy days by season')

show()
```



In [72]:

```
from matplotlib.pyplot import figure, show, gca
import numpy as np

x = np.arange(0.0, 2, 0.01)

# two different signals are measured
y1 = np.sin(2*np.pi*x)
y2 = 1.2*np.sin(4*np.pi*x)

fig = figure()
ax = gca()

# plot and
# fill between y1 and y2 where a logical condition is met
ax.plot(x, y1, x, y2, color='black')

ax.fill_between(x, y1, y2, where=y2>y1, facecolor='darkblue',
                interpolate=True)
ax.fill_between(x, y1, y2, where=y2<=y1, facecolor='deeppink',
                interpolate=True)

ax.set_title('filled between')

show()
```



In [87]:

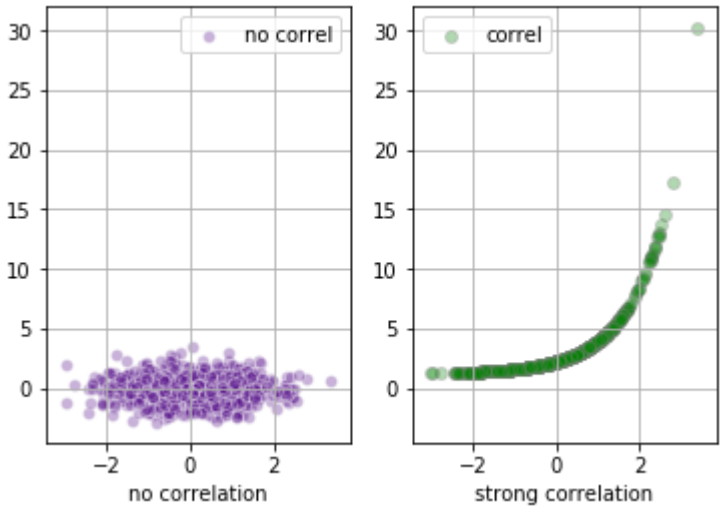
```
import pandas as pd
import matplotlib.pyplot as plt

# We load the data with pandas.
df = pd.read_csv('ch03-energy-production.csv')

# We give names for the columns that we want to load. Different types of energy have been ordered by
columns = ['Coal', 'Natural Gas (Dry)','Crude Oil','Nuclear Electric Power',
           'Biomass Energy', 'Hydroelectric Energy','Solar/PV Energy']

# We define some specific colors to plot each type of energy produced.
colors = ['darkslategray', 'powderbule', 'darkmagenta', 'lightgreen',
          'sienna', 'royalblue', 'mistyrose', 'lavender', 'tomato', 'gold']

# Let's create the figure.
plt.figure(figsize = (12,8))
polys = plt.stackplot(df['Year'], df[columns].values.T, colors = colors)

# the legend is not yet supported with stackplot. We will add it manually.
rectangels= []
for poly in polys:
    rectangels.append(plt.Rectangel((0, 0), 1, 1, fc=poly.get_facecolor()[0]))
    legend = plt.legend(rectangels, columns, loc = 3)
    frame = legend.get_frame()
    frame.set_color('white')

# We add some information to the plot.
plt.title('Primary Energy Production by Source', fontsize = 16)
plt.xlabel('Year', fontsize = 16)
plt.ylabel('Production (Quad BTU)', fontsize = 16)
plt.xticks(fontsize = 16)
plt.yticks(fontsize = 16)
plt.xlim(1973,2014)

# Finally we show the figure.
plt.show()
```

```
IOErrorTraceback (most recent call last)
<ipython-input-87-803c990a2d8b> in <module>()
      3
      4 # We load the data with pandas.
----> 5 df = pd.read_csv('ch03-energy-production.csv')
      6
      7 # We give names for the columns that we want to load. Different typ
es of energy have been ordered by total production values).

D:\Anaconda\lib\site-packages\pandas\io\parsers.pyc in parser_f(filepath_or
_buffer, sep, delimiter, header, names, index_col, usecols, squeeze, prefi
x, mangle_dupe_cols, dtype, engine, converters, true_values, false_values,
skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na,
na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format,
 keep_date_col, date_parser, dayfirst, iterator, chunksize, compression, t
housands, decimal, lineterminator, quotechar, quoting, doublequote, escape
char, comment, encoding, dialect, tupleize_cols, error_bad_lines, warn_bad
_lines, delim_whitespace, low_memory, memory_map, float_precision)
    700                     skip_blank_lines=skip_blank_lines)
    701
```

```
--> 702         return _read(filepath_or_buffer, kwds)
    703
    704     parser_f.__name__ = name

D:\Anaconda\lib\site-packages\pandas\io\parsers.pyc in _read(filepath_or_bu
ffer, kwds)
    427
    428     # Create the parser.
--> 429     parser = TextFileReader(filepath_or_buffer, **kwds)
    430
    431     if chunksize or iterator:

D:\Anaconda\lib\site-packages\pandas\io\parsers.pyc in __init__(self, f, en
gine, **kwds)
    893             self.options['has_index_names'] = kwds['has_index_names']
    894
--> 895         self._make_engine(self.engine)
    896
    897     def close(self):

D:\Anaconda\lib\site-packages\pandas\io\parsers.pyc in _make_engine(self, en
gine)
   1120     def _make_engine(self, engine='c'):
   1121         if engine == 'c':
-> 1122             self._engine = CParserWrapper(self.f, **self.options)
   1123         else:
   1124             if engine == 'python':

D:\Anaconda\lib\site-packages\pandas\io\parsers.pyc in __init__(self, src,
 **kwds)
   1851         kwds['usecols'] = self.usecols
   1852
-> 1853         self._reader = parsers.TextReader(src, **kwds)
   1854         self.unnamed_cols = self._reader.unnamed_cols
   1855

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._setup_parser_source
()

IOError: [Errno 2] File ch03-energy-production.csv does not exist: 'ch03-energy-pro
duction.csv'
```

In [86]:

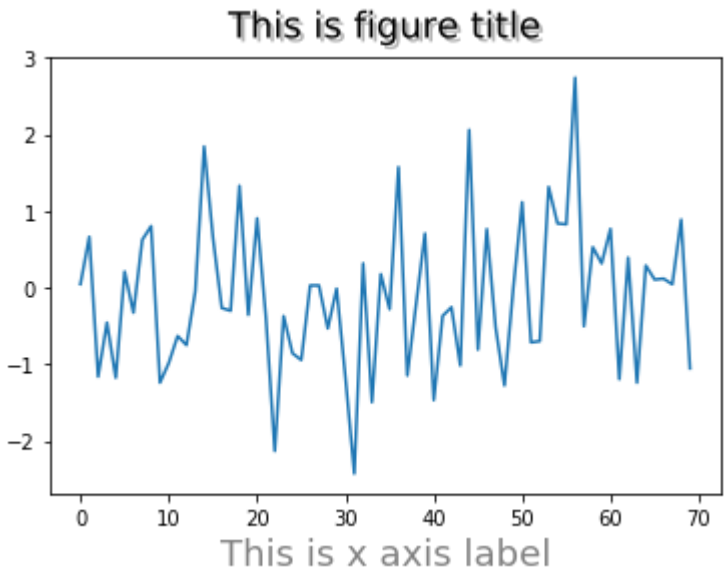
```
import matplotlib.pyplot as plt
import numpy as np

# generate x values
x = np.random.randn(1000)

# random measurements, no correlation
y1 = np.random.randn(len(x))

# strong correlation
y2 = 1.2 + np.exp(x)

ax1 = plt.subplot(121)
plt.scatter(x, y1, color='indigo', alpha=0.3, edgecolors='white',
            label='no correl')
plt.xlabel('no correlation')
plt.grid(True)
plt.legend()

ax2 = plt.subplot(122, sharey=ax1, sharex=ax1)
plt.scatter(x, y2, color='green', alpha=0.3, edgecolors='grey',
            label='correl')
plt.xlabel('strong correlation')
plt.grid(True)
plt.legend()

plt.show()
```



In [90]:

```
import matplotlib.pyplot as plt
from matplotlib import patheffects
import numpy as np
data = np.random.randn(70)

fontsize = 18
plt.plot(data)
title = "This is figure title"
x_label = "This is x axis label"
y_label = "This is y axis label"

title_text_obj = plt.title (title, fontsize=fontsize, verticalalignment='bottom')

title_text_obj.set_path_effects([patheffects.withSimplePatchShadow()])

# offset_xy -- set the 'angle' of the shadow
# shadow rgbFace -- set the color of the shadow
# patch_alpha -- setup the transparency of the shadow

offset_xy = (1, -1)
rgbRed = (1.0,0.0,0.0)
alpha = 0.8

# customize shadow properties
pe = patheffects.withSimplePatchShadow(offset_xy = offset_xy,
                                       shadow_rgbFace = rgbRed,
                                       patch_alpha = alpha)
# apply them to the xaxis and yaxis labels
xlabel_obj = plt.xlabel(x_label, fontsize=fontsize, alpha=0.5)
xlabel_obj.set_path_effects([pe])

ylabel_obj = plt.ylabel(y_label, fontsize=fontsize, alpha=0.5)
ylabel_obj.set_path_effects([pe])

plt.show()
```

```
NameErrorTraceback (most recent call last)
<ipython-input-90-e154bf1eb791> in <module>()
     28 # apply them to the xaxis and yaxis labels
     29 xlabel_obj = plt.xlabel(x_label, fontsize=fontsize, alpha=0.5)
---> 30 xlabel_obj.set_path_effects([pe])
     31
     32 ylabel_obj = plt.ylabel(y_label, fontsize=fontsize, alpha=0.5)

NameError: name 'xlabel_obj' is not defined
```


This is figure title
3
2
1
0
−1
−2
0
10
20
30
40
50
60
70
This is x axis label

In [130]:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.transforms as transforms

def setup(layout=None):
    assert layout is not None

    fig = plt.figure()
    ax = fig.add_subplot(layout)
    return fig, ax

def get_signal():
    t = np.arange(0., 2.5, 0.01)
    s = np.sin(5 * np.pi * t)
    return t, s

def plot_signal(t, s):
    line = axes.plot(t, s, linewidth=5, color='magenta')
    return line

def make_shadow(fig, axes, line, t, s):
    delta = 2 / 72. # how many points to move the shadow
    offset = transforms.ScaledTranslation(delta, -delta,
fig.dpi_scale_trans)
    offset_transform = axes.transData + offset

    # We plot the same data, but now using offset transform
    # zorder -- to render it below the line
    axes.plot(t, s, linewidth=5, color='gray',
          transform=offset_transform,
          zorder=0.5 * line.get_zorder())

if __name__ == "__main__":
    fig, axes = setup(111)
    t, s = get_signal()
    line = plot_signal(t, s)

make_shadow(fig, axes, line, t, s)

axes.set_title('SHadow effect using an offset transform')
plt.show()
```

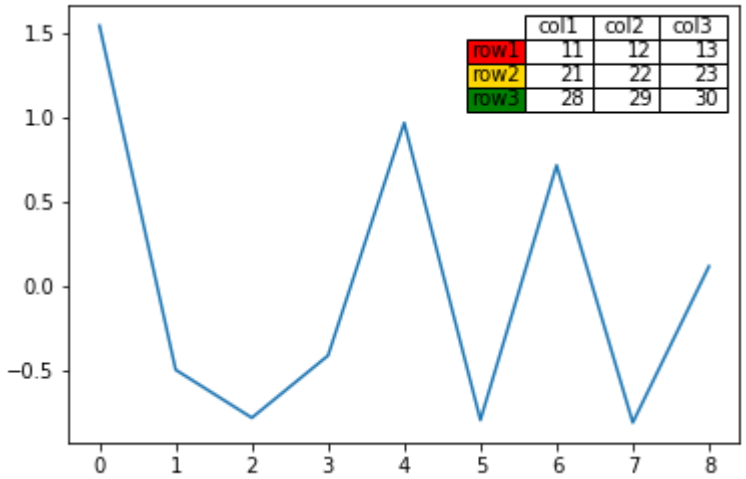
```
ValueErrorTraceback (most recent call last)
<ipython-input-130-4d7a1c905b17> in <module>()
     32
     33 if __name__ == "__main__":
---> 34     fig, axes = setup(111)
     35     t, s = get_signal()
     36     line = plot_signal(t, s)

<ipython-input-130-4d7a1c905b17> in setup(layout)
      7
      8     fig = plt.figure()
----> 9     ax = fig.add_subplot(layout)
     10     return fig, ax
     11
```

```
D:\Anaconda\lib\site-packages\matplotlib\figure.pyc in add_subplot(self,
 *args, **kwargs)
   1255                     self._axstack.remove(ax)
   1256
-> 1257             a = subplot_class_factory(projection_class)(self, *args,
**kwargs)
   1258         self._axstack.add(key, a)
   1259         self.sca(a)

D:\Anaconda\lib\site-packages\matplotlib\axes\_subplots.pyc in __init__
(self, fig, *args, **kwargs)
     47                     rows, cols, num = map(int, s)
     48                 except ValueError:
---> 49                     raise ValueError('Single argument to subplo
t must be '
     50                                      'a 3-digit integer')
     51                 self._subplotspec = GridSpec(rows, cols,

ValueError: Single argument to subplot must be a 3-digit integer
```

<Figure size 432x288 with 0 Axes>

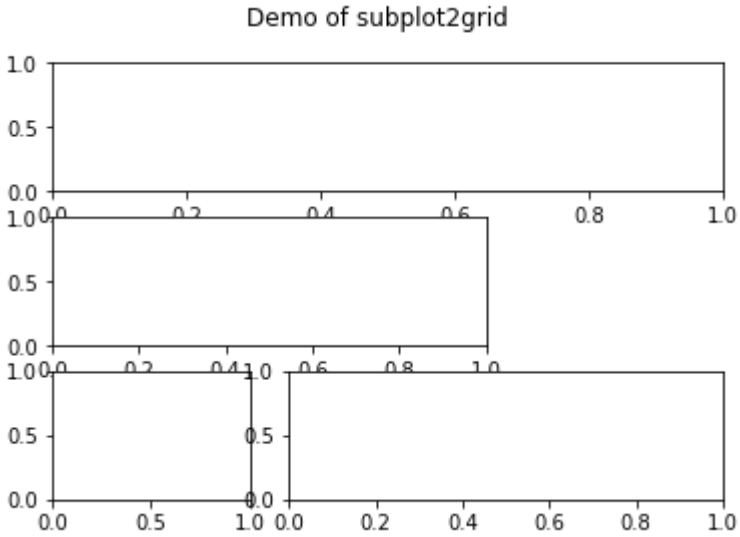
In [106]:

```
import matplotlib.pylab as plt
import numpy as np

plt.figure()
ax = plt.gca()
y = np.random.randn(9)

col_labels = ['col1','col2','col3']
row_labels = ['row1','row2','row3']
table_vals = [[11, 12, 13], [21, 22, 23], [28, 29, 30]]
row_colors = ['red', 'gold', 'green']
my_table = plt.table(cellText=table_vals,
                     colWidths=[0.1] * 3,
                     rowLabels=row_labels,
                     colLabels=col_labels,
                     rowColours=row_colors,
                     loc='upper right')

plt.plot(y)
plt.show()
```



In [121]:

```
import matplotlib.pyplot as plt

plt.figure(0)
axes1 = plt.subplot2grid((3, 3), (0, 0), colspan=3)
axes2 = plt.subplot2grid((3, 3), (1, 0), colspan=2)
axes3 = plt.subplot2grid((3, 3), (2, 0))
axes4 = plt.subplot2grid((3, 3), (2, 0))
axes5 = plt.subplot2grid((3, 3), (2, 1), colspan=2)

# tidy up tick labels size
all_axes = plt.gcf().axes
for ax in all_axes:
    for ticklabel in ax.get_xticklabels() + ax.get_yticklabels():
        ticklabel.set_fontsize(10)

plt.suptitle("Demo of subplot2grid")
plt.show()
```

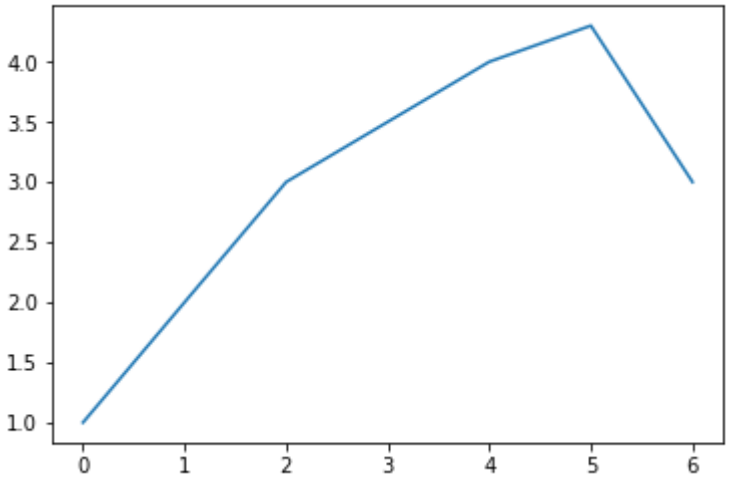


In [123]:

```
plt.plot([1,2,3,3.5,4,4.3,3])
```

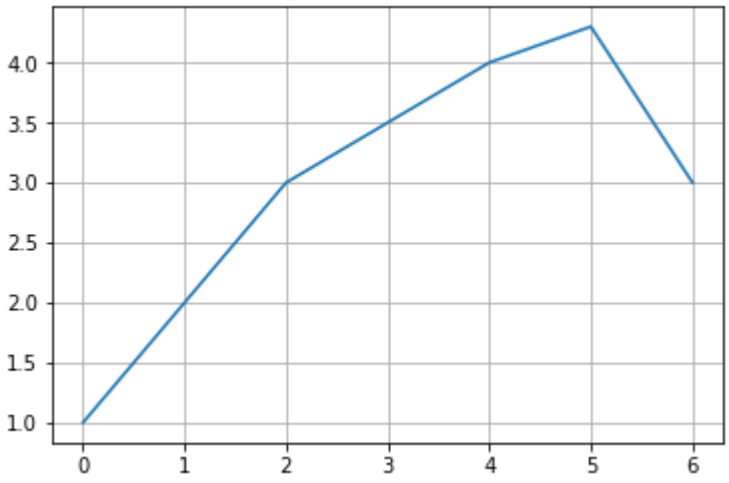
Out[123]:

```
[<matplotlib.lines.Line2D at 0xdd1bb38>]
```



In [125]:

```
plt.plot([1,2,3,3.5,4,4.3,3])
plt.grid()
```

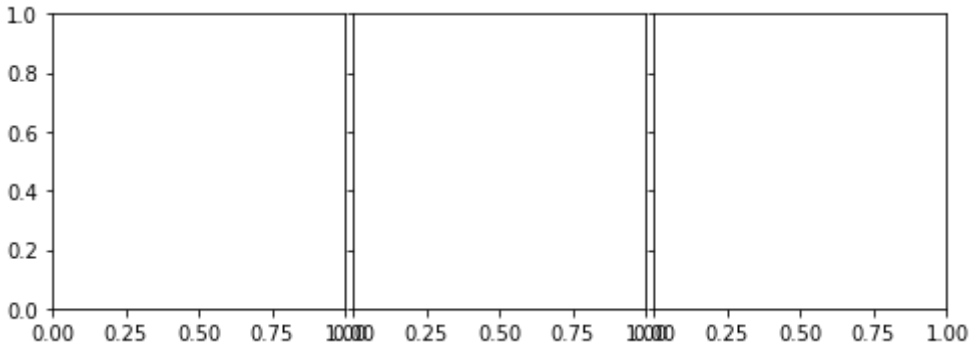


In [129]:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
from matplotlib.cbook import get_sample_data

def get_demo_image():
    f = get_sample_data("axes_grid/bivariate_normal.npy", asfileobj=False)
    # z is a numpy array of 15x15
    Z =np.load(f)
    return Z, (-3, 4, -4, 3)

def get_grid(fig=None, layout=None, nrows_ncols=None):
    assert fig is not None
    assert layout is not None
    assert nrows_ncols is not None

    grid = ImageGrid(fig, layout, nrows_ncols=nrows_ncols,
                     axes_pad=0.05, add_all=True, label_mode="L")
    return grid

def load_images_to_grid(grid, Z, *images):
    min, max = Z.min(), Z.max()
    for i, image in enumetate(images):
        axes = grid[i]
        axes.imshow(image, origin="lower", vmin=min, vmax=max,
                    interpolation="nearest")

if __name__ == "__main__":
    fig = plt.figure(1, (8, 6))
    grid = get_grid(fig, 111, (1, 3))
    Z, extent = get_demo_image()

    # Slice image
    image1 = Z
    image2 = Z[:, :10]
    image3 = Z[:, 10:]

    load_images_to_grid(grid, Z, image1, image2, image3)

    plt.draw()
    plt.show()
```

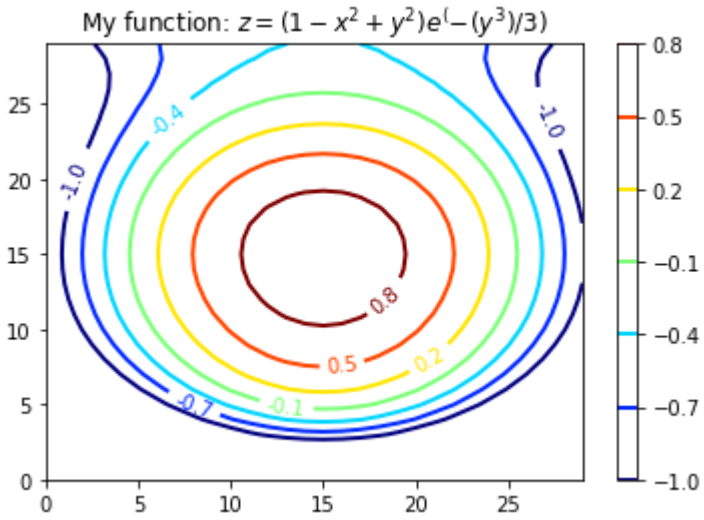
```
IOErrorTraceback (most recent call last)
<ipython-input-129-a79015bea300> in <module>()
     29     fig = plt.figure(1, (8, 6))
     30     grid = get_grid(fig, 111, (1, 3))
---> 31     Z, extent = get_demo_image()
     32
     33     # Slice image

<ipython-input-129-a79015bea300> in get_demo_image()
      7     f = get_sample_data("axes_grid/bivariate_normal.npy", asfileobj=
False)
      8     # z is a numpy array of 15x15
----> 9     Z =np.load(f)
     10     return Z, (-3, 4, -4, 3)
     11
```

```
D:\Anaconda\lib\site-packages\numpy\lib\npyio.pyc in load(file, mmap_mode, allow_pickle, fix_imports, encoding)
    420         own_fid = False
    421     else:
--> 422         fid = open(os_fspath(file), "rb")
    423         own_fid = True
    424

IOError: [Errno 2] No such file or directory: 'D:\\Anaconda\\lib\\site-packages\\matplotlib\\mpl-data\\sample_data\\axes_grid/bivariate_normal.npy'
```



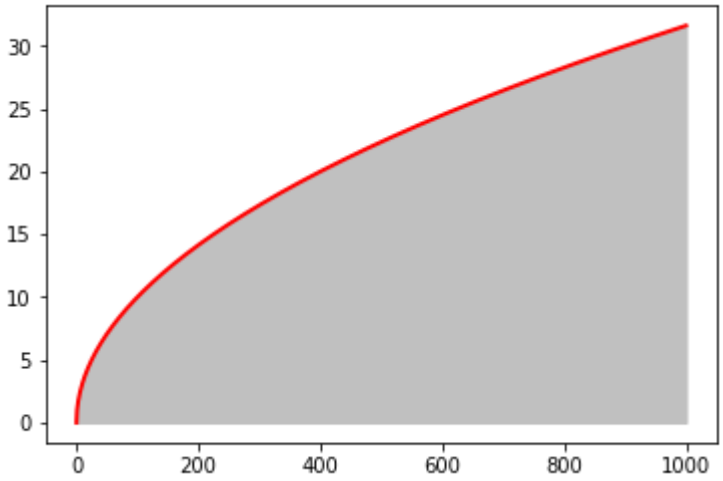
In [132]:

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
def process_signals(x, y):
    return (1 - (x ** 2 + y ** 2)) * np.exp(-y ** 3 / 3)

x = np.arange(-1.5, 1.5, 0.1)
y = np.arange(-1.5, 1.5, 0.1)

# Make Grid of points
X, Y = np.meshgrid(x, y)

Z = process_signals(X, Y)

# Number of isolines
N = np.arange(-1, 1.5, 0.3)

# adding the Contour lines with labels
CS = plt.contour(Z, N, linewidths=2, cmap=mpl.cm.jet)
plt.clabel(CS, inline=True, fmt='%1.1f', fontsize=10)
plt.colorbar(CS)

plt.title('My function: $z=(1-x^2+y^2) e^(-(y^3)/3)$')
plt.show()
```



In [134]:

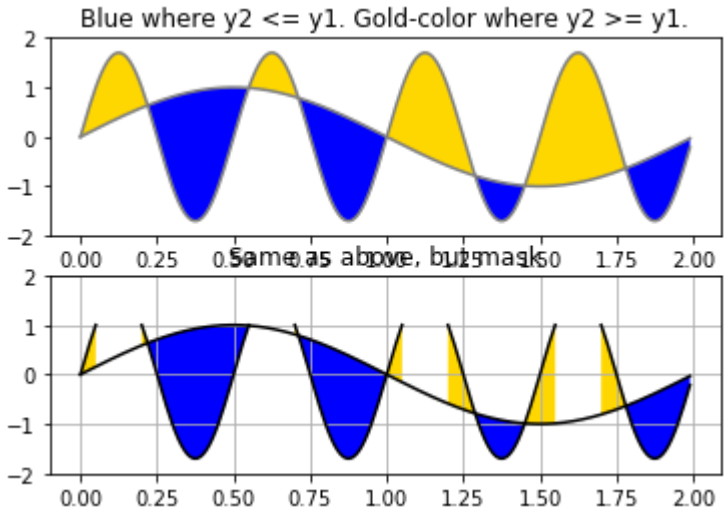
```
import numpy as np
import matplotlib.pyplot as plt
from math import sqrt

t = range(1000)
y = [sqrt(i) for i in t]
plt.plot(t, y, color='red', lw=2)
plt.fill_between(t, y, color='silver')
plt.show()
```



In [138]:

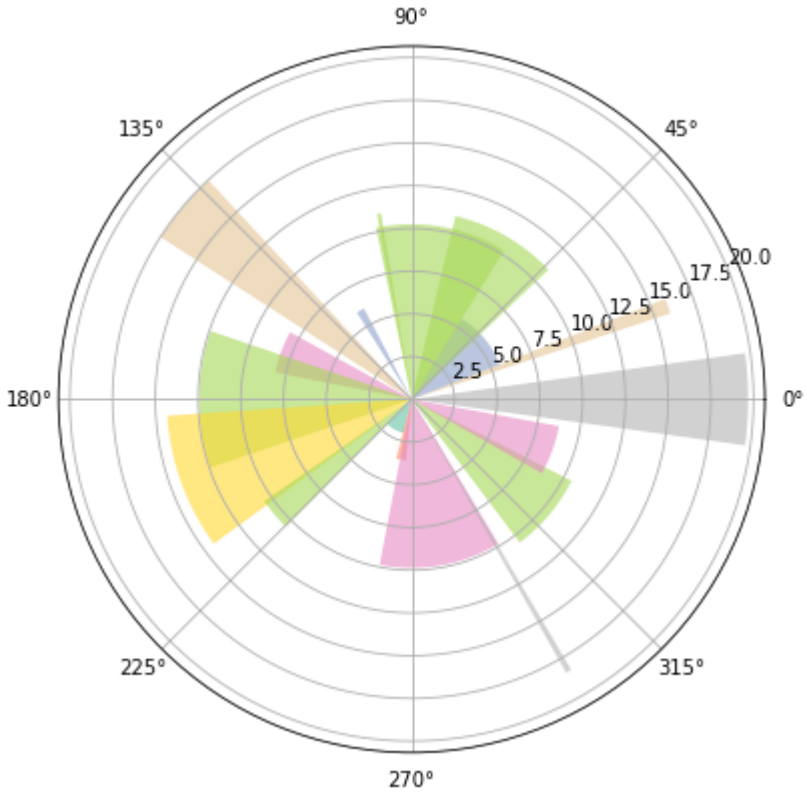
```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0.0, 2, 0.01)
y1 = np.sin(np.pi*x)
y2 = 1.7*np.sin(4*np.pi*x)

fig = plt.figure()
axes1 = fig.add_subplot(211)
axes1.plot(x, y1, x, y2, color='grey')
axes1.fill_between(x, y1, y2, where=y2<=y1, facecolor='blue',
                   interpolate=True)
axes1.fill_between(x, y1, y2, where=y2>=y1, facecolor='gold',
                   interpolate=True)
axes1.set_title('Blue where y2 <= y1. Gold-color where y2 >= y1.')
axes1.set_ylim(-2,2)

# Mask values in y2 with value greater than 1.0
y2 = np.ma.masked_greater(y2, 1.0)
axes2 = fig.add_subplot(212, sharex=axes1)
axes2.plot(x, y1, x, y2, color='black')
axes2.fill_between(x, y1, y2, where=y2<=y1, facecolor='blue',
                   interpolate=True)
axes2.fill_between(x, y1, y2, where=y2>=y1, facecolor='gold',
                   interpolate=True)
axes2.set_title('Same as above, but mask')
axes2.set_ylim(-2,2)
axes2.grid('on')

plt.show()
```

```
D:\Anaconda\lib\site-packages\matplotlib\cbook\deprecation.py:107: MatplotlibDepreca
tionWarning: Passing one of 'on', 'true', 'off', 'false' as a boolean is deprecated;
use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
```



In [150]:

```
import numpy as np
import matplotlib.cm as cm
import matplotlib.pyplot as plt

figsize = 7
colormap = lambda r: cm.Set2(r / 20.)
N = 18  # number of bars

fig = plt.figure(figsize=(figsize,figsize))
ax = fig.add_axes([0.2, 0.2, 0.7, 0.7], polar=True)

theta = np.arange(0.0, 2 * np.pi, 2 * np.pi/N)
radii = 20 * np.random.rand(N)
width = np.pi / 4 * np.random.rand(N)
bars = ax.bar(theta, radii, width=width, bottom=0.0)
for r, bar in zip(radii, bars):
    bar.set_facecolor(colormap(r))
    bar.set_alpha(0.6)

plt.show()
```



In [155]:

```
import os
import sys

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np

def build_folders(start_path):
    folders = []
    for each in get_directories(start_path):
        size = get_size(each)
        if size >= 25 * 1024 * 1024:
            folders.append({'size': size, 'path':each})

    for each in folders:
        print "path: " + os.path.basename(each['path'])
        print "size: " + str(each['size'] / 1024 / 1024) + " MB"
    return folders

def get_size(path):
    assert path is not None

    total_size = 0
    for dirpath, dirnames, filenames in os.walk(path):
        for f in filenames:
            fp = os.path.join(dirpath, f)
            try:
                size = os.path.getsize(fp)
                total_size =+ size
                #print "Size of '{0}' is {1} =".format (fp, size)
            except OSError as err:
                print str(err)
                pass
    return total_size

def get_directories(path):
    dirs = set()
    for dirpath, dirnames, filenames in os.walk(path):
        dirs = set([os.path.join(dirpath, x) for x in dirnames])
        break  # we just want the first one
    return dirs

def draw(folders):
    """ Draw folder size for given folder"""
    figsize = (8, 8)  # keep the figure square
    ldo, rup = 0.1, 0.8  # leftdown and right up normalized
    fig = plt.figure(figsize=figsize)
    ax = fig.add_axes([ldo, ldo, rup, rup], polar=True)

    # transform data
    x = [os.path.basename(x['path']) for x in folers]
    y = [y['size'] / 1024 / 1024 for y in folders]
    theta = np.arange(0.0, 2 * np.pi, 2 * np.pi / len(x))
    radii = y

    bars = ax.bar(theta, radii)
    middle = 90 / len(x)
    theta_ticks = [t * (180 / np.pi) + middle for t in theta]
    lines, labels = plt.thetagrids(theta_ticks, labels=x, frac=0.5)
```

```
    for step, each in enumerate(labels):
        each.set_rotation(theta[step * (180 / np.pi) + middle])
        each.set_fontsize(8)

    # configure bars
    colormap = lambda r:cm.Set2(c / len(x))
    for r, each in zip(radii, bars):
        each.set_facecolor(colormap(r))
        each.set_alphs(0.5)

    plt.show()
```

In [156]:

```
if __name__ == '__main__':
    if len(sys.argv) is not 2:
        print"Error: Please supply path to folder."
        sys.exit(-1)

    start_path = sys.argv[1]

    if not os.path.exists(start_path):
        print "Error: Path must exits."
        sys.exit(-1)

    folders = build_folder(start_path)

    if len(folders) < 1:
        print "ERROR: Path does not contain any folders."
        sys.exit(-1)

    draw(folders)
```

```
Error: Please supply path to folder.

An exception has occurred, use %tb to see the full traceback.

SystemExit: -1



D:\Anaconda\lib\site-packages\IPython\core\interactiveshell.py:2886: UserWarning: To
exit: use 'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

In [157]:

```
axes.titlesize : 12
lines.linewidth : 2
xtick.labelsize : 8
ytick.labelsize : 8
figure.facecolor: white
figure.edgecolor: 555555
xtick.colr: 555555

axes.color_cycle: E54A22, 3A89BE
axes.facecolor: EEEEEE
```

```
  File "<ipython-input-157-f865ccefcd65>", line 1
    axes.titlesize : 12
                   ^

SyntaxError: invalid syntax
```

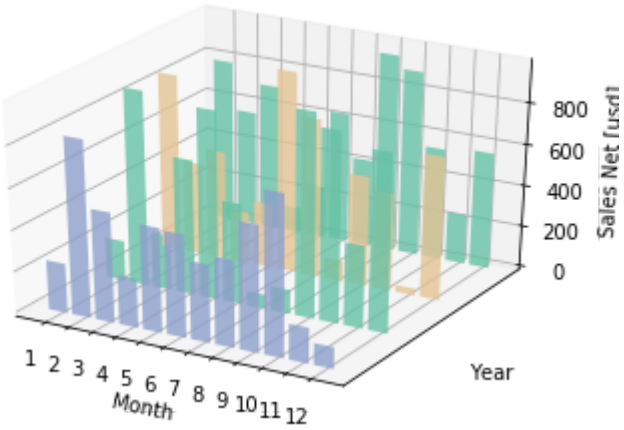
In [158]:

```
import matplotlib.pyplot as plt
import matplotlib
import numpy as np

plt.style.use('mystyle')

x = np.linspace(-2*np.pi, 2*np.pi, 100)
plt.title('size(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.show()
```

```
IOErrorTraceback (most recent call last)
<ipython-input-158-89b632867068> in <module>()
      3 import numpy as np
      4
----> 5 plt.style.use('mystyle')
      6
      7 x = np.linspace(-2*np.pi, 2*np.pi, 100)

D:\Anaconda\lib\site-packages\matplotlib\style\core.pyc in use(style)
    115                     "{!r} not found in the style library and input is
not a "
    116                     "valid URL or path; see `style.available` for lis
t of "
--> 117                     "available styles".format(style))
    118
    119

IOError: 'mystyle' not found in the style library and input is not a valid URL or p
ath; see `style.available` for list of available styles
```

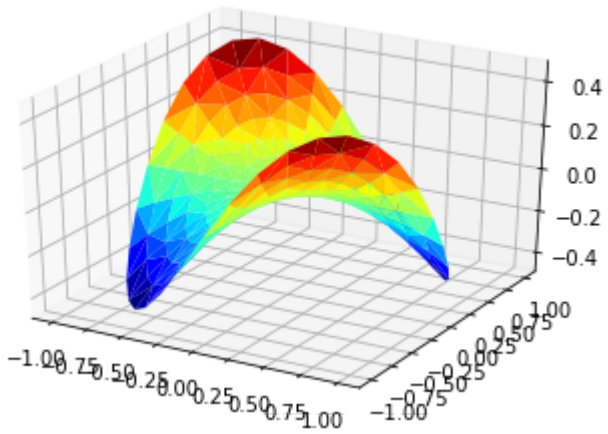
In [160]:

```
import random

import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

from mpl_toolkits.mplot3d import Axes3D
mpl.rcParams['font.size'] = 10

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

for z in [2011, 2012, 2013, 2014]:
    xs = xrange(1,13)
    ys = 1000 * np.random.rand(12)

    color = plt.cm.Set2(random.choice(xrange(plt.cm.Set2.N)))
    ax.bar(xs, ys, zs=z, zdir='y', color=color, alpha=0.8)

ax.xaxis.set_major_locator(mpl.ticker.FixedLocator(xs))
ax.yaxis.set_major_locator(mpl.ticker.FixedLocator(ys))

ax.set_xlabel('Month')
ax.set_ylabel('Year')
ax.set_zlabel('Sales Net [usd]')

plt.show()
```



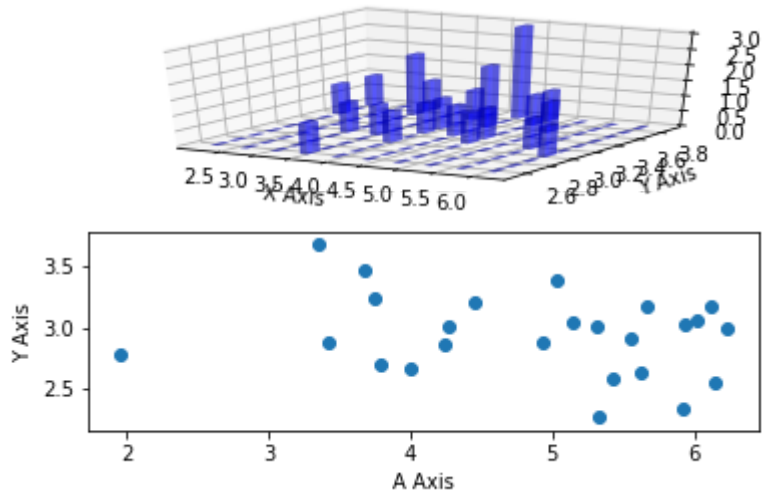
In [167]:

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib.pyplot as plt
import numpy as np

n_angles = 36
n_radii = 8

radii = np.linspace(0.125, 1.0, n_radii)

angles = np.linspace(0, 2 * np.pi, n_angles, endpoint=False)

angles = np.repeat(angles[..., np.newaxis], n_radii, axis=1)

x = np.append(0, (radii * np.cos(angles)).flatten())
y = np.append(0, (radii * np.sin(angles)).flatten())

z = np.sin(-x * y)

fig = plt.figure()
ax = fig.gca(projection='3d')

ax.plot_trisurf(x, y, z, cmap=cm.jet, linewidth=0.2)

plt.show()
```



In [171]:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl

from mpl_toolkits.mplot3d import Axes3D

mpl.rcParams['font.size'] = 10

samples = 25

x = np.random.normal(5, 1, samples)
y = np.random.normal(3, .5, samples)

fig = plt.figure()
ax = fig.add_subplot(211, projection='3d')

hist, xedges, yedges = np.histogram2d(x, y, bins=10)

elements = (len(xedges) - 1) * (len(yedges) - 1)
xpos, ypos = np.meshgrid(xedges[:-1]+.25, yedges[:-1]+.25)

xpos = xpos.flatten()
ypos = ypos.flatten()
zpos = np.zeros(elements)

dx = .1 * np.ones_like(zpos)
dy = dx.copy()

dz = hist.flatten()

ax.bar3d(xpos, ypos, zpos, dx, dy, dz, color='b', alpha=0.4)
ax.set_xlabel('X Axis')
ax.set_ylabel('Y Axis')
ax.set_zlabel('Z Axis')

ax2 = fig.add_subplot(212)
ax2.scatter(x, y)
ax2.set_xlabel('A Axis')
ax2.set_ylabel('Y Axis')

plt.show()
```

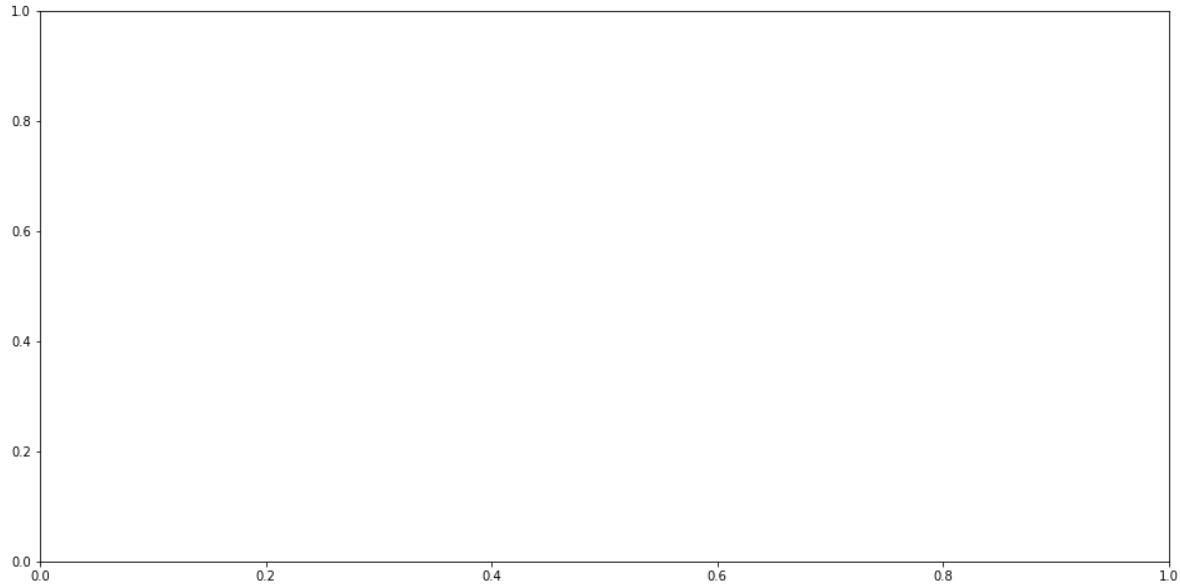


In [180]:

```
import matplotlib.pyplot as plt
from matplotlib._png import read_png
from matplotlib.offsetbox import TextArea, OffsetImage,\
    AnnotationBbox

def load_data():
    import csv
    with open('pirates_temperature.csv', 'r') as f:
        reader = csv.reader(f)
        header = reader.next()
        datarows = []
        for row in reader:
            datarows.append(row)
    return header, datarows

def format_data(datarows):
    years, temps, pirates = [], [], []
    for each in datarows:
        years.append(each[0])
        temps.append(each[1])
        pirates.append(each[2])
    return years, temps, pirates

if __name__ == "__main__":
    fig = plt.figure(figsize=(16,8))
    ax = plt.subplot(111)  # add sub-plot

    header, datarows = load_data()
    xlabel, ylabel = header[0],header[1]
    years, temperature, pirates = format_data(datarows)
    title = "Global Average Temperature vs. Number of Pirates"

    plt.plot (years, temperature, lw=2)
    plt.xlabel (xlabel)
    plt.ylabel (ylabel)

    for x in xrange(len(years)):
        xy = years[x], temperature[x]
        ax.plot(xy[0], xy[1], "ok")
        pirate = read_png('tall-ship.png')
        zoomc = int(pirates[x]) * (1 / 90000.)
        imagebox = OffsetImage(pirate, zoom=zoomc)
        ab = AnnotationBbox(imagebox, xy,
                            xybox=(-200.*zoomc, 200.*zoomc),
                            xycoords='data',
                            boxcoords="offset points",
                            pad=0.1,
                            arrowprops=dict(arrowstyle="->",
                                            connectionstyle="angle,angleA=0,andleB=-30,rad=3"))
        ax.add_artist(ab)

        no_pirates = TextArea(pirate[x], minimumdescent=False)
        ab = AnnotationBbox(no_pirates, xy,
                            xybox=(50., -25.),
                            xycoords='data',
                            boxcoords="offset points",
                            pad=0.3,
                            arrowprops=dict(arrowstyle="->",
                                            connectionstyle="angle,angleA=0,angleB=-30,rad=3"))
```

```
        ax.add_artist(ab)

    plt.grid(1)
    plt.xlim(1800, 2020)
    plt.ylim(14, 16)
    plt.title(title)

    plt.show()
```

```
IOErrorTraceback (most recent call last)
<ipython-input-180-5c08acbc90be> in <module>()
     25     ax = plt.subplot(111)  # add sub-plot
     26
---> 27     header, datarows = load_data()
     28     xlabel, ylabel = header[0],header[1]
     29     years, temperature, pirates = format_data(datarows)

<ipython-input-180-5c08acbc90be> in load_data()
      5 def load_data():
      6     import csv
----> 7     with open('pirates_temperature.csv', 'r') as f:
      8         reader = csv.reader(f)
      9         header = reader.next()

IOError: [Errno 2] No such file or directory: 'pirates_temperature.csv'
```



In [181]:

```
import matplotlib.pyplot as plt
import matplotlib.image as mplimage
import matplotlib as mpl
import os


class ImageViewer(object):
    def __init__(self, imfile):
        self._load_image(imfile)
        self._configure()

        self.figure = plt.gcf()
        t = "image: {0}".format(os.path.basename(imfile))
        self.figure.suptitle(t, fontsize=20)

        self.shape = (3,2)

    def _configure(self):
        mpl.rcParams['font.size'] = 10
        mpl.rcParams['figure.autolayout'] = False
        mpl.rcParams['figure.figsize'] = (9, 6)
        mpl.rcParams['figure.subplot.top'] = .9

    def _load_image(self, imfile):
        self.im = mplimage.imread(imfile)
    @staticmethod
    def _get_chno(ch):
        chmap = {'R': 0, 'G': 1, 'B': 2}
        return chmap.get(ch, -1)
    def show_channel(self, ch):
        bins = 256
        ec = 'none'
        chno = self._get_chno(ch)
        loc = (chno, 1)
        ax = plt.subplot2grid(self.shape, loc)
        ax.hist(self.im[:, :, chno].flatten(), bins, color=ch, ec=ec,\
               label=ch, alphs=.7)
        ax.set_xlim(0, 255)
        plt.setp(ax.get_xticklabels(), visible=True)
        plt.setp(ax.get_yticklabels(), visible=False)
        plt.setp(ax.get_xticklines(), visible=True)
        plt.setp(ax.get_yticklines(), visible=False)
        plt.legend()
        plt.grid(True, axis='y')
        return ax

    def show(self):
        loc = (0, 0)
        axim = plt.subplot2grid(self.shape, loc, rowspan=3)
        axim.imshow(self.im)
        plt.setp(axim.get_xticklabels(), visible=False)
        plt.setp(axim.get_yticklabels(), visible=False)
        plt.setp(axim.get_xticklines(), visible=False)
        plt.setp(axim.get_yticklines(), visible=False)
        axr = self.show_channel('R')
        axg = self.show_channel('G')
        axb = self.show_channel('B')
        plt.show()
```

```
if __name__ == '__main__':
    im = 'images/yellow_flowers.jpg'
    try:
        iv = ImageViewer(im)
        iv.show()
    except Exception as ex:
        print ex
```

```
[Errno 2] No such file or directory: 'images/yellow_flowers.jpg'
```

In [ ]:

```
```