visualization

November 28, 2018

1 Disclaimer: Vote!

- What: Akademischer Senat, Akademische Versammlung (elects President), StuPa, ...
- Where: Beneath Mensa
- When

	4. Dec (Tuesday)	5. Dec (Wednesday)	6. Dec (Thursday)
Von	9:30	11:30	9:30
Bis	14:00	16:00	12:30

2 Python for Data Science

Visualization

...make both calculations and graphs. Both sorts of output should be studied; each will contribute to understanding.

F. J. Anscombe, 1973

(and echoed in nearly all talks about data visualization...)

3 Anscombe's Quartet

Wikipedia on Anscombe

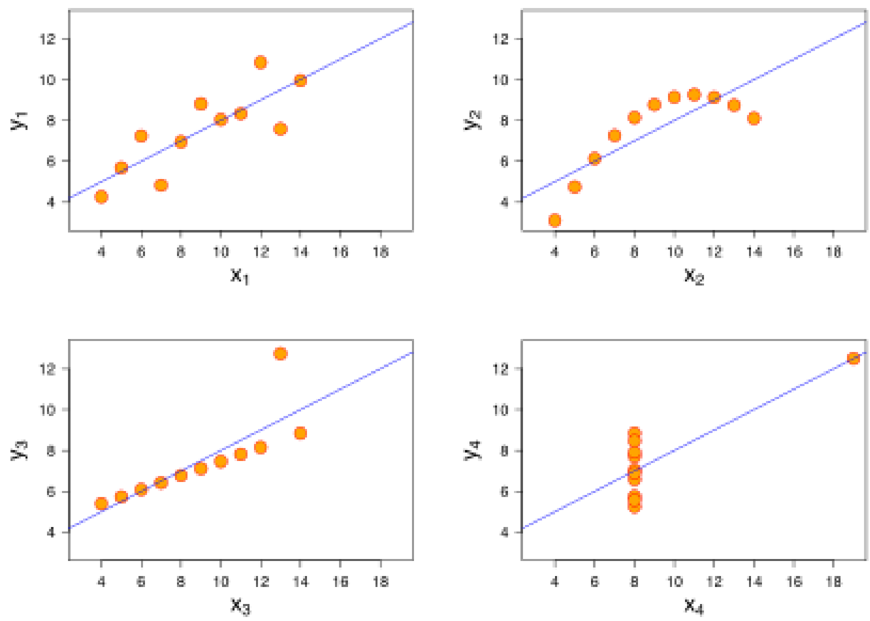
4 Anscombe's Quartet - Reloaded

DataSaurus Rex

Justin Matejka and George Fitzmaurice, *Same Stats, Different Graphs: Generating Datasets with Varied Appearance and Identical Statistics through Simulated Annealing*

5 Overview

- Plotting Basics with `matplotlib`
- Plotting complex data from `pandas`
- Even more abstraction with `seaborn`



Anscombe's Quartet

- Map visualizations with `folium`

6 Other plotting libraries

- ggpy
- HoloViews
- Altair
- bokeh
- plotly

7 Matplotlib

- Multi-platform data visualization library built on NumPy
- Works well on many operating systems / graphics backends
- Many output formats
- First version released in 2003 by John D. Hunter
- Substantial support by Space Telescope Science Institute (researchers behind Hubble telescope)

7.1 Importing Matplotlib

```
In [486]: import matplotlib.pyplot as plt
          plt.style.use('ggplot')
          import matplotlib
          matplotlib.rcParams['lines.linewidth'] = 2
```

```
matplotlib.rcParams['savefig.dpi'] = 300
matplotlib.rcParams['figure.figsize'] = [12,6]
```

7.2 Three ways of plotting

- In Jupyter Notebook (Convenient, code and figures in one place)
- From Ipython Shell
- From Python script (Reproducible, versionable code)

7.2.1 Plotting from a script

Write a file called *myplot.py* containing the following:

```
# ------- file: myplot.py ------
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))

plt.savefig('myplot.pdf')

plt.show()
```

Running this script from the command-line prompt will result in a window opening with your figure displayed:

```
$ python myplot.py
```

(On OSX you might need to try `pythonw` instead)

7.3 Plotting from an IPython shell

IPython is built to work well with Matplotlib if you specify Matplotlib mode. To enable this mode, use the `%matplotlib` magic command:

```
In [1]: %matplotlib
Using matplotlib backend: TkAgg

In [2]: import matplotlib.pyplot as plt
```

Any `plt` plot command will cause a figure window to open, and further commands can be run to update the plot.

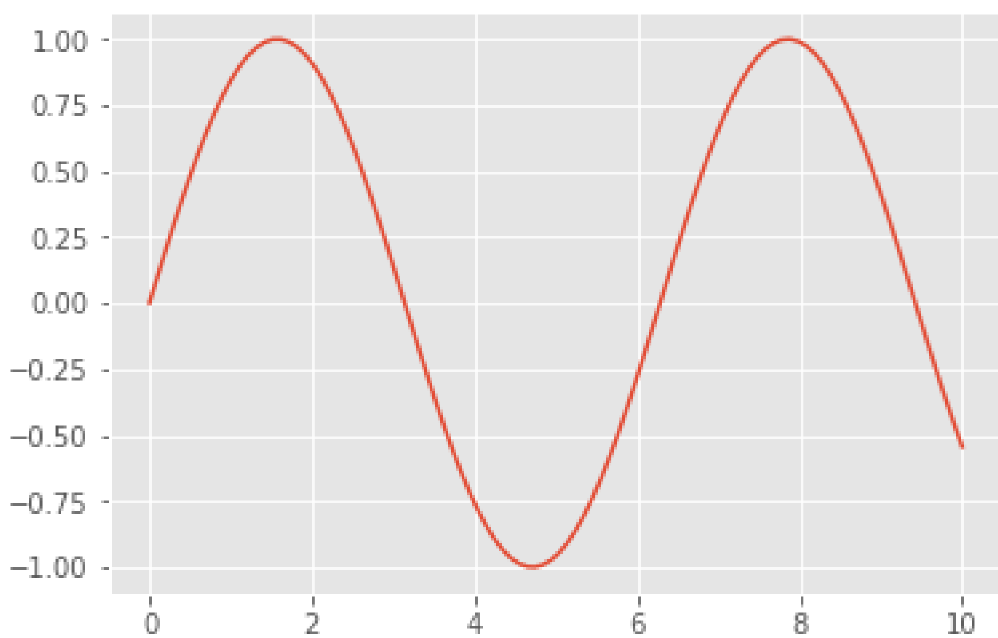
To force an update of a plot, use `plt.draw()`. Using `plt.show()` in Matplotlib mode is not required.

7.3.1 Plotting from a Jupyter Notebook

- Like in IPython, use the `%matplotlib` magic command
- Two options of embedding graphics directly in the notebook:
  - `%matplotlib notebook`:*interactive* plots
  - `%matplotlib inline`: *static* images of your plot embedded in the notebook

For this book, we will generally opt for `%matplotlib inline`

```
In [17]: %matplotlib inline
         import matplotlib.pyplot as plt
         plt.style.use('ggplot')
         import numpy as np
         x = np.linspace(0, 10, 100)

         fig = plt.figure()
         plt.plot(x, np.sin(x), '-');
```



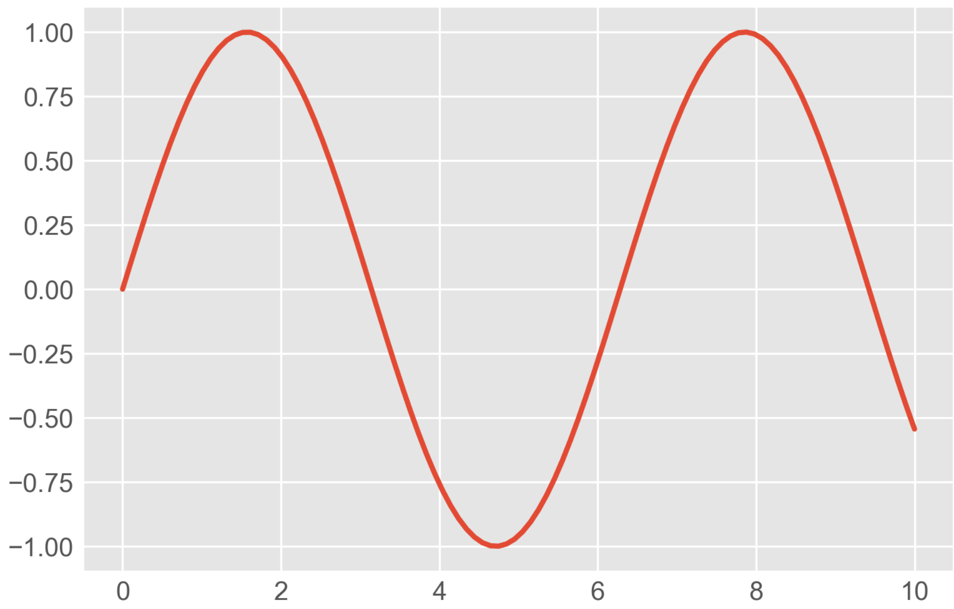
7.3.2 Saving Figures to File

- Variety of output formats
- No need to call `plt.show()` before

```
In [489]: fig.savefig('my_figure.png')
```

In [490]:
```
from IPython.display import Image
Image('my_figure.png')
```

Out[490]:



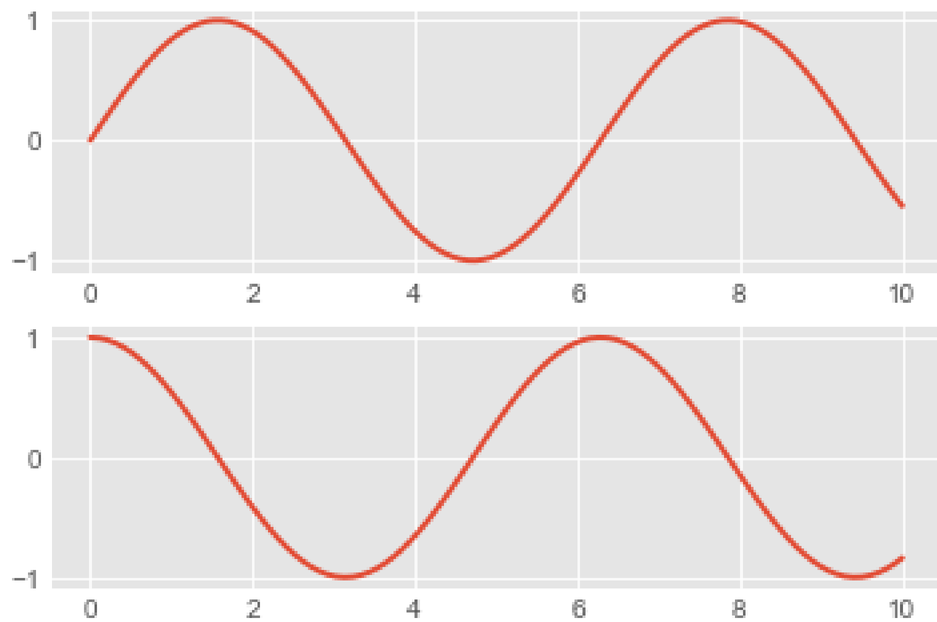
In [424]:
```
fig.canvas.get_supported_filetypes()
```

Out[424]:
```
{'ps': 'Postscript',
 'eps': 'Encapsulated Postscript',
 'pdf': 'Portable Document Format',
 'pgf': 'PGF code for LaTeX',
 'png': 'Portable Network Graphics',
 'raw': 'Raw RGBA bitmap',
 'rgba': 'Raw RGBA bitmap',
 'svg': 'Scalable Vector Graphics',
 'svgz': 'Scalable Vector Graphics',
 'jpg': 'Joint Photographic Experts Group',
 'jpeg': 'Joint Photographic Experts Group',
 'tif': 'Tagged Image File Format',
 'tiff': 'Tagged Image File Format'}
```

7.3.3 MATLAB Style API

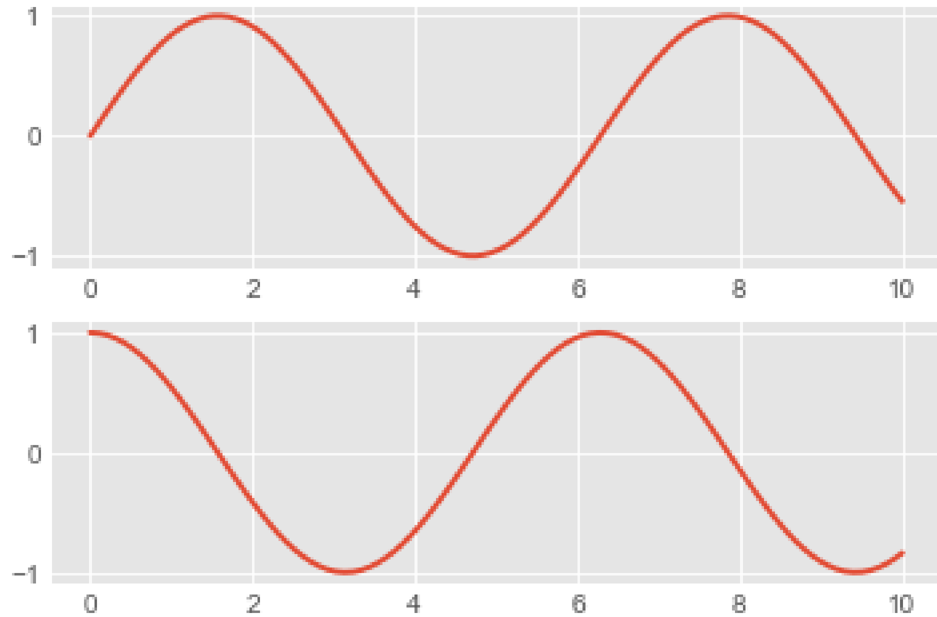
In [491]:
```
plt.figure()  # create a plot figure
```

```
# create the first of two panels and set current axis
plt.subplot(2, 1, 1) # (rows, columns, panel number)
plt.plot(x, np.sin(x))

# create the second panel and set current axis
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x));
```



7.3.4 Object Oriented API

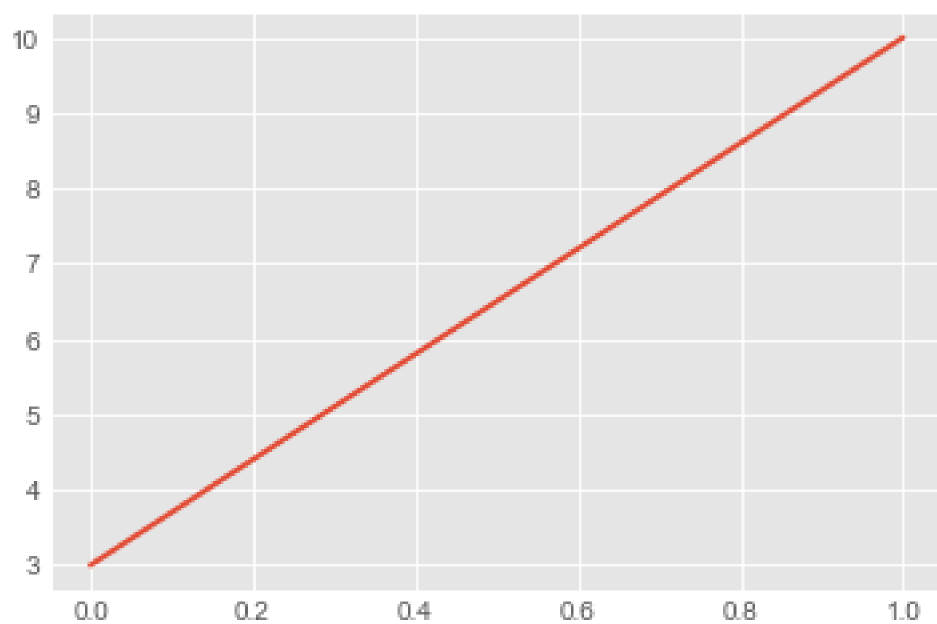
In [492]:
```
# First create a grid of plots
# ax will be an array of two Axes objects
fig, ax = plt.subplots(2)

# Call plot() method on the appropriate object
ax[0].plot(x, np.sin(x))
ax[1].plot(x, np.cos(x));
```



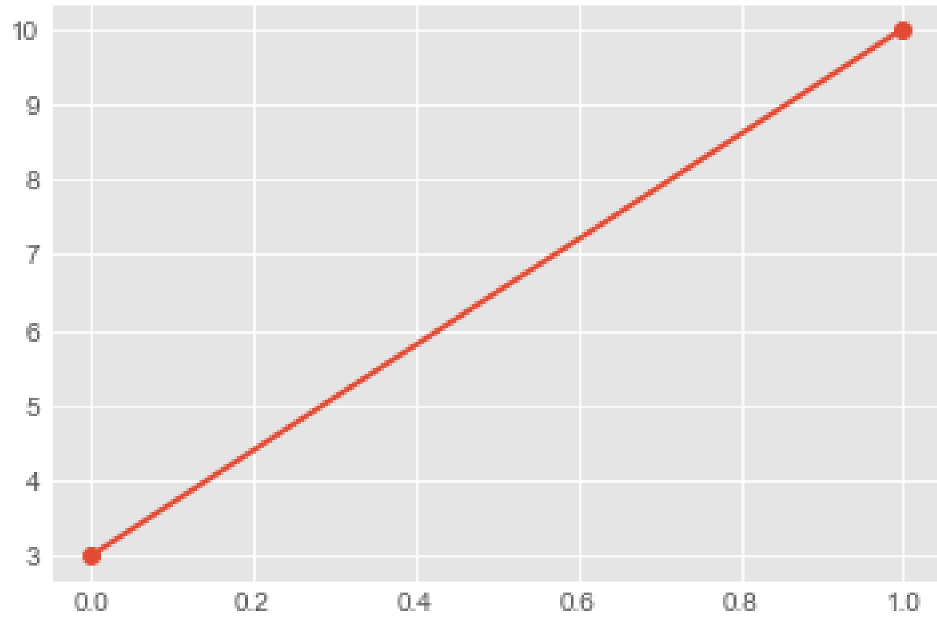
7.4 Simple Line Plots

Simple here means functions of the type $y = f(x)$

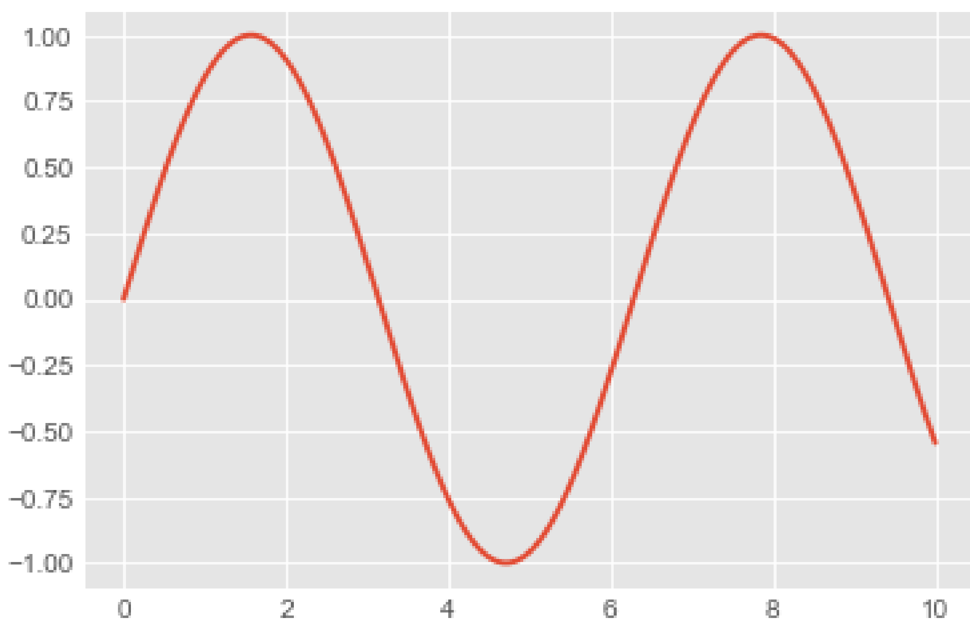
```
In [493]: plt.plot([0,1],[3,10]);
```



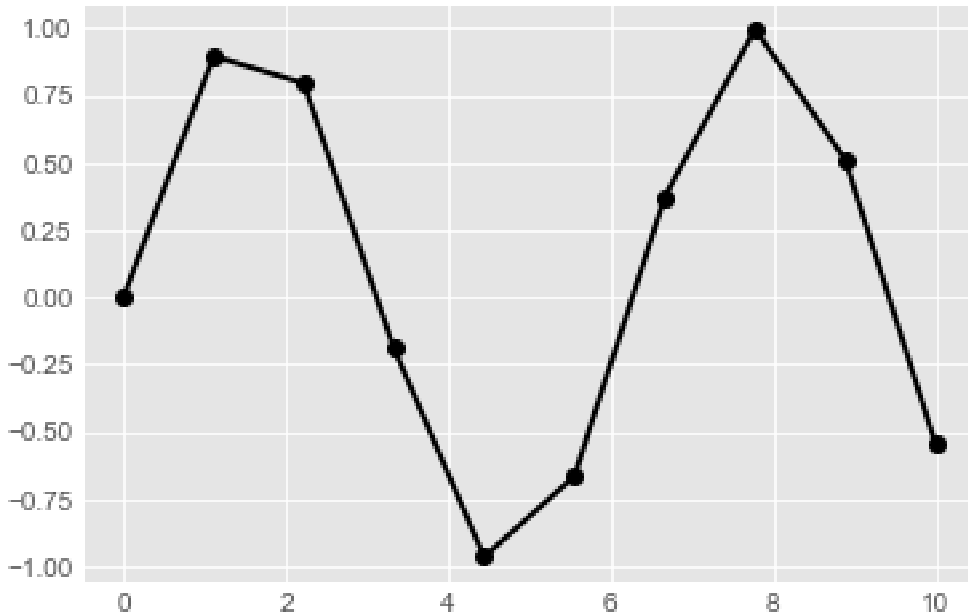
In [494]:
```
plt.plot([0,1],[3,10],'-o');
```



In [495]:
```
x = np.linspace(0, 10, 1000)
plt.plot(x, np.sin(x));
```

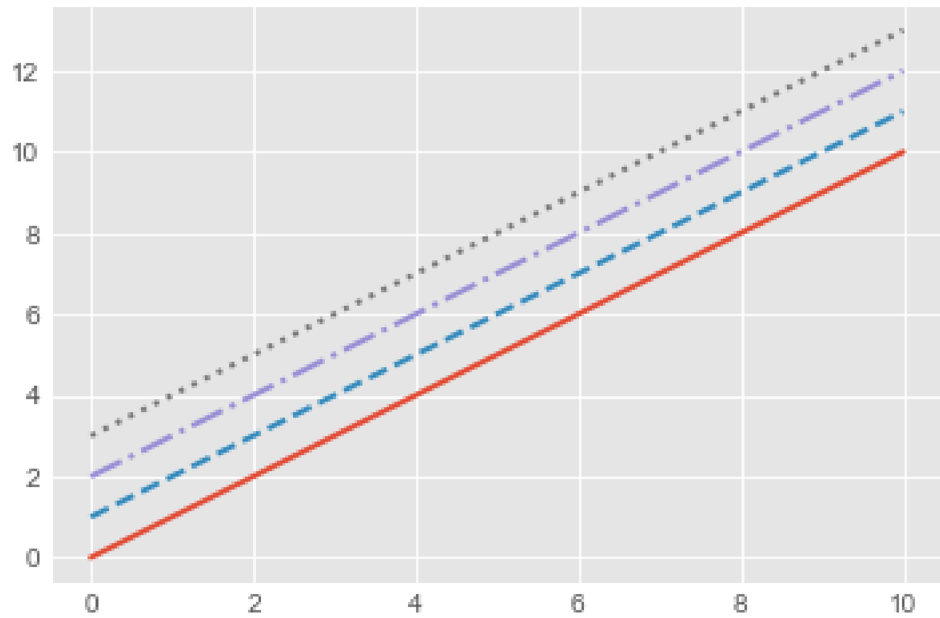


```
In [496]: x = np.linspace(0, 10, 10)
          plt.plot(x, np.sin(x), 'k-o');
```

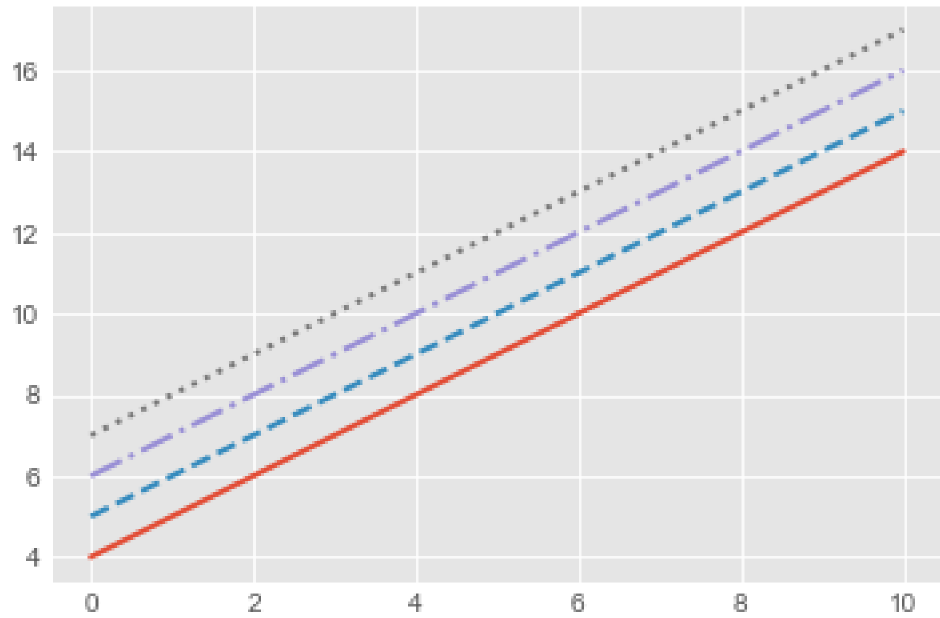


7.4.1 Line Styles

```
In [497]: x = np.linspace(0, 10, 1000)
          plt.plot(x, x + 0, linestyle='solid')
          plt.plot(x, x + 1, linestyle='dashed')
          plt.plot(x, x + 2, linestyle='dashdot')
          plt.plot(x, x + 3, linestyle='dotted');
```

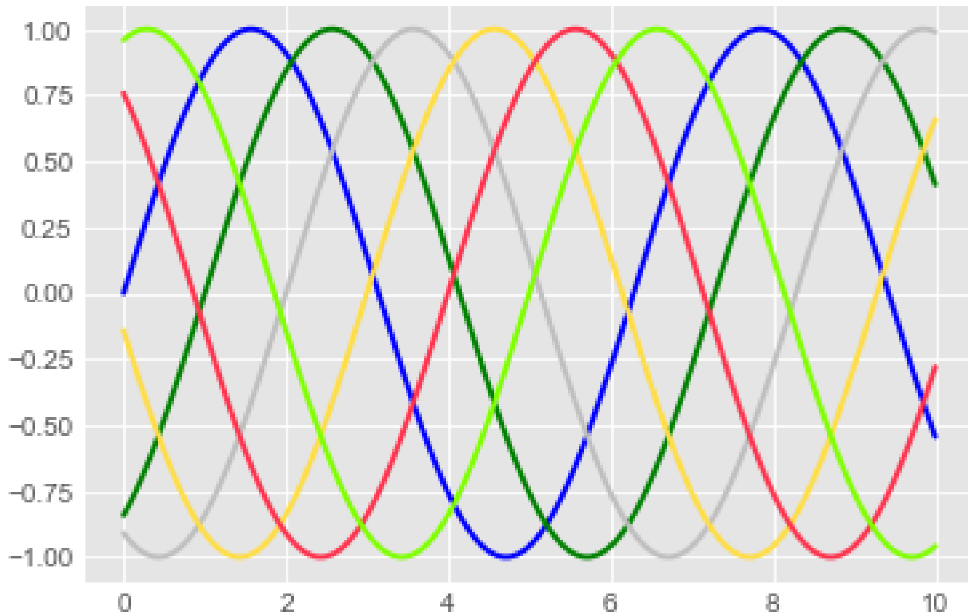


In [498]:
```
# For short, you can use the following codes:
plt.plot(x, x + 4, linestyle='-')  # solid
plt.plot(x, x + 5, linestyle='--') # dashed
plt.plot(x, x + 6, linestyle='-.') # dashdot
plt.plot(x, x + 7, linestyle=':');  # dotted
```

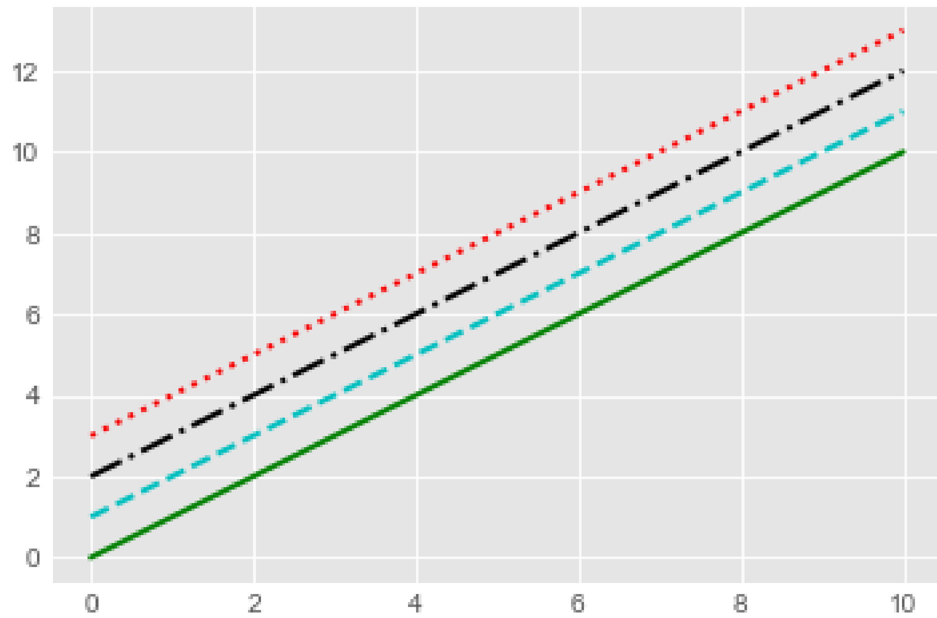


7.4.2 Line Colors

```
In [499]: x = np.linspace(0, 10, 1000)
          plt.plot(x, np.sin(x - 0), color='blue')        # specify color by name
          plt.plot(x, np.sin(x - 1), color='g')           # short color code (rgbcmyk)
          plt.plot(x, np.sin(x - 2), color='0.75')        # Grayscale between 0 and 1
          plt.plot(x, np.sin(x - 3), color='#FFDD44')     # Hex code (RRGGBB from 00 to FF)
          plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3)) # RGB tuple, values 0 to 1
          plt.plot(x, np.sin(x - 5), color='chartreuse'); # all HTML color names supported
```

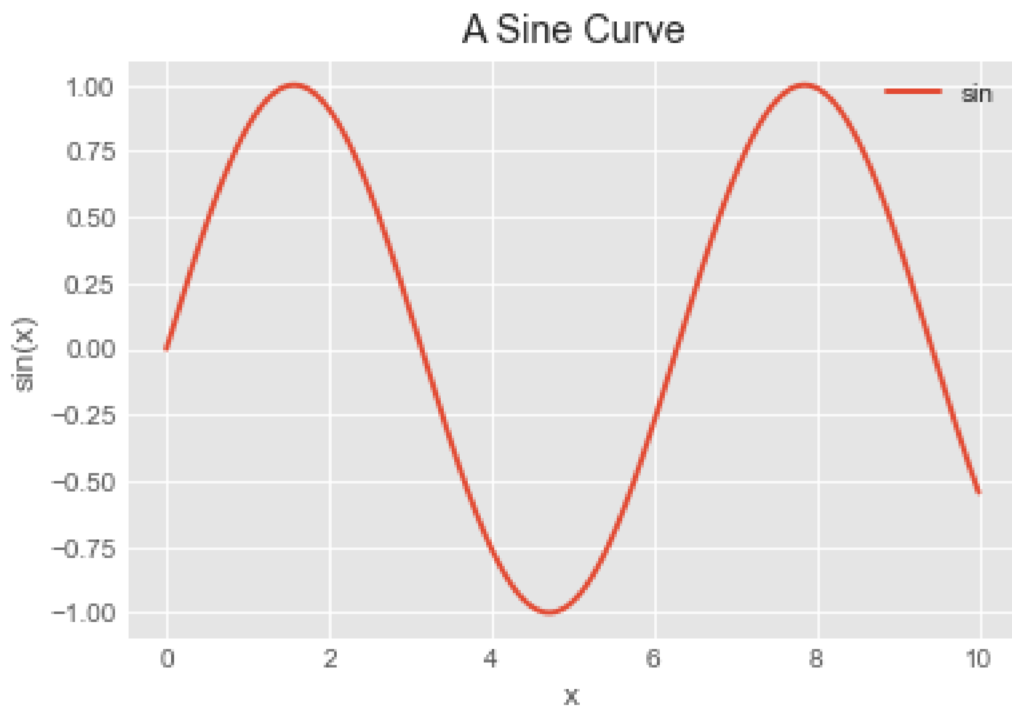



```
In [500]: # If you really don't want to write a lot

          x = np.linspace(0, 10, 1000)

          plt.plot(x, x + 0, '-g')  # solid green
          plt.plot(x, x + 1, '--c') # dashed cyan
          plt.plot(x, x + 2, '-.k') # dashdot black
          plt.plot(x, x + 3, ':r');  # dotted red
```



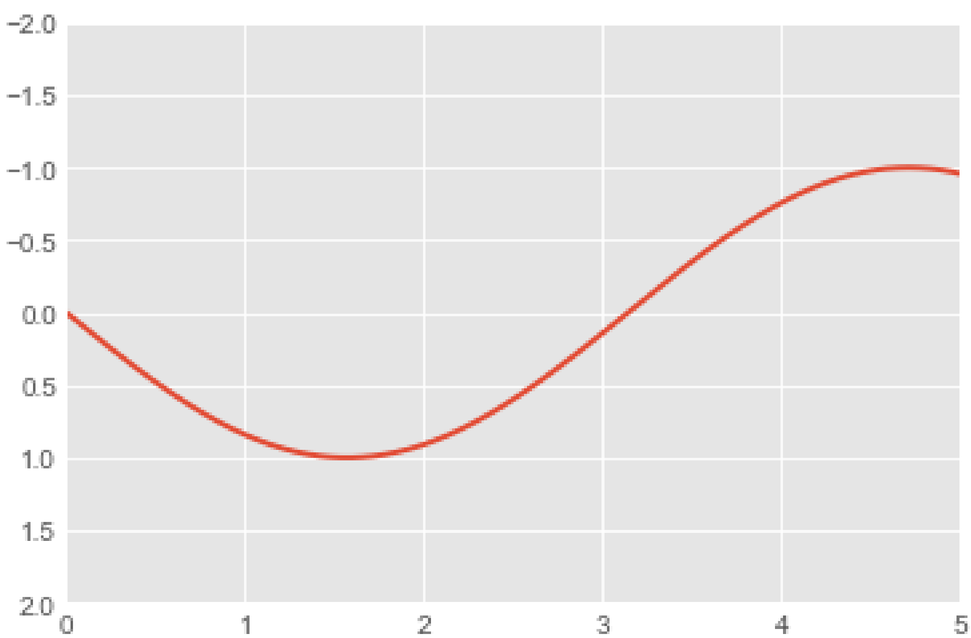
7.4.3 Annotations

```
In [501]: plt.plot(x, np.sin(x))
          plt.title("A Sine Curve")
          plt.xlabel("x")
          plt.ylabel("sin(x)");
          plt.legend(["sin"])

Out[501]: <matplotlib.legend.Legend at 0x1a2ba2b208>
```



7.4.4 Axis Limit

```
In [502]: plt.plot(x, np.sin(x))
          plt.xlim(0, 5)
          plt.ylim(2, -2);
```

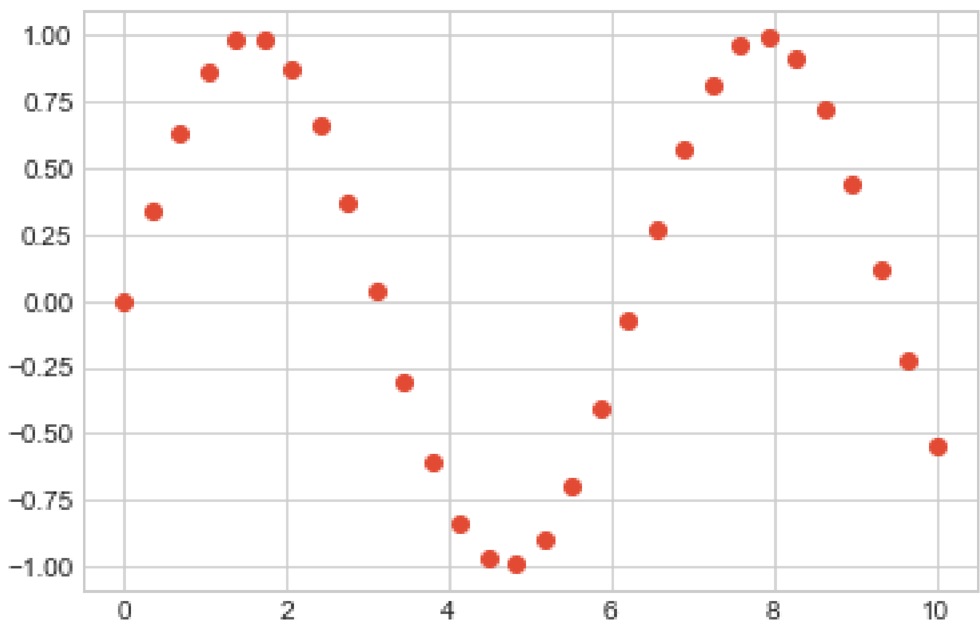


7.4.5 For more on styles

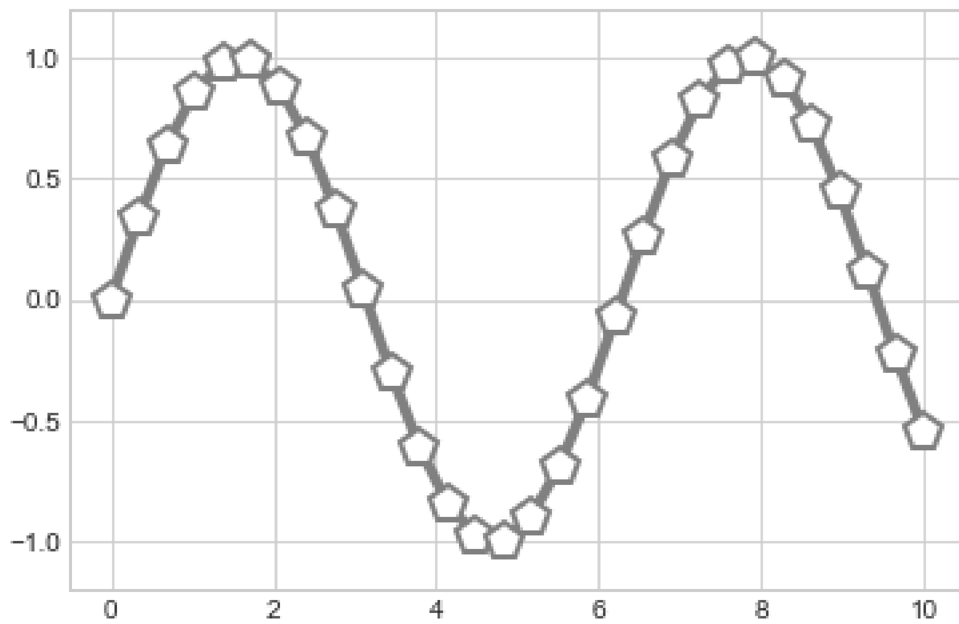
SciPy Lecture Notes on Matplotlib

7.5 Scatter Plots

```
In [503]: plt.style.use('seaborn-whitegrid')
          x = np.linspace(0, 10, 30)
          y = np.sin(x)
          plt.plot(x, y, 'o');
```

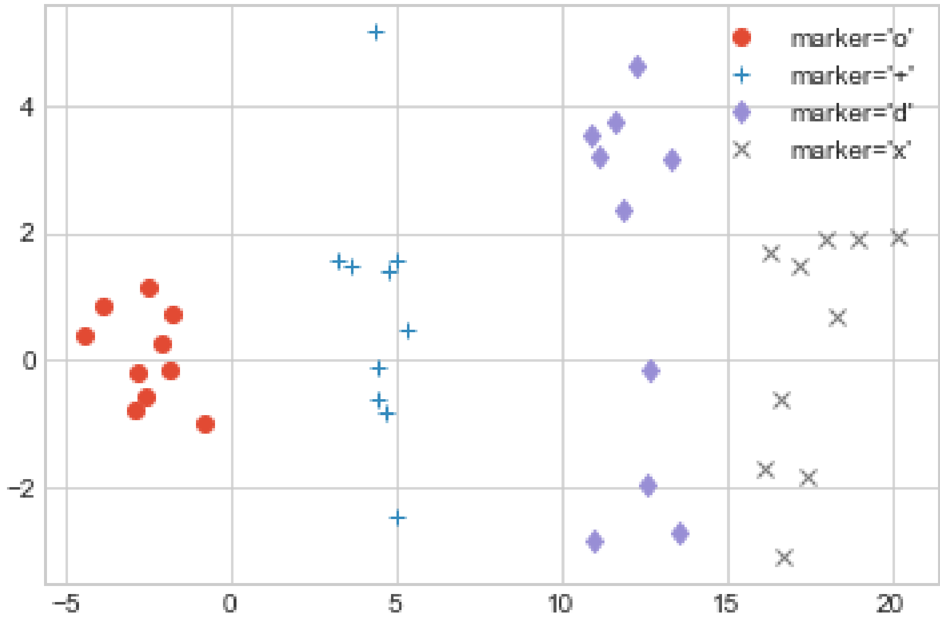



```
In [504]: plt.plot(x, y, '-p', color='gray',
                   markersize=15, linewidth=4,
                   markerfacecolor='white',
                   markeredgecolor='gray',
                   markeredgewidth=2)
          plt.ylim(-1.2, 1.2);
```



7.6 More Markers

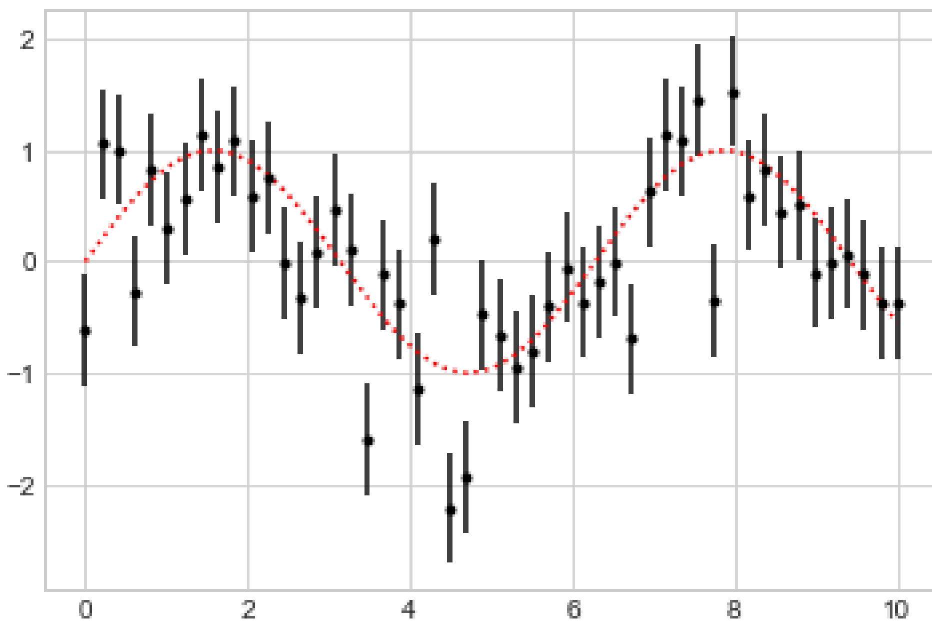
```
In [505]: means = [-2,5,12,18]
          stds = [1,5,6,3]
          markers = ['o', '+', 'd', 'x']

          plt.figure()
          for mu,stdev,mark in zip(means,stds,markers):
              x = np.random.multivariate_normal([mu,0],np.diag([1,stdev]),10)
              plt.plot(x[:,0],x[:,1], mark, label="marker='{}'".format(mark))

          plt.legend(numpoints=1);
```



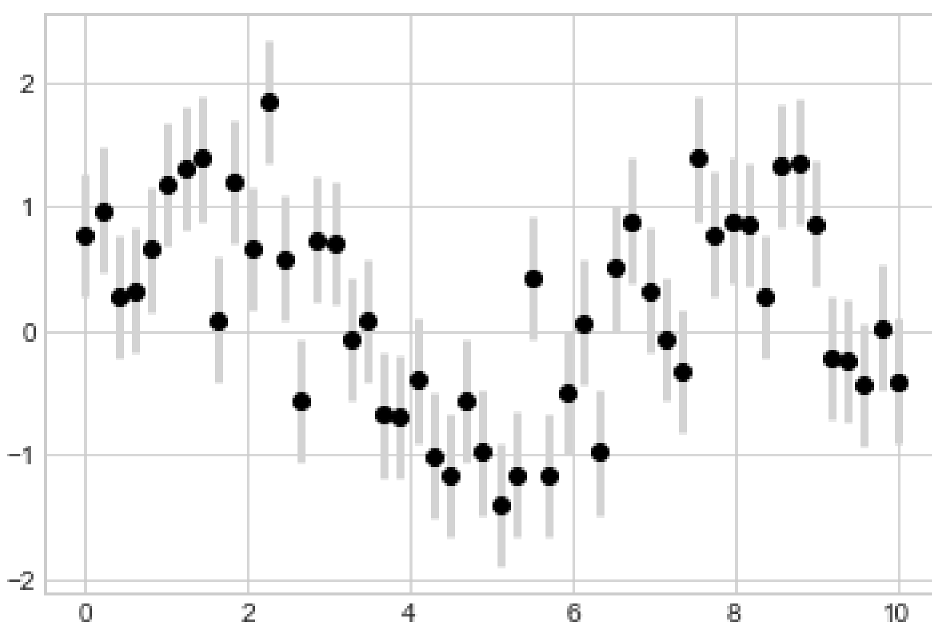

```
In [506]: import time
          import pandas as pd
          # Get Gapminder Life Expectancy data (csv file is hosted on the web)
          url = 'https://python-graph-gallery.com/wp-content/uploads/gapminderData.csv'
          data = pd.read_csv(url)
          data.to_csv("data/gapminder.csv",index=False)
```

7.7 Visualizing Errors - Errorbars

```
In [20]: plt.style.use('seaborn-whitegrid')

         N = 50
         x = np.linspace(0, 10, N)
         error = .5
         y = np.sin(x)
         y_measured = y + error * np.random.randn(N)

         plt.plot(x, y, 'r:', label='true')
         plt.errorbar(x, y_measured, yerr=error, fmt='.k', label='measured');
```



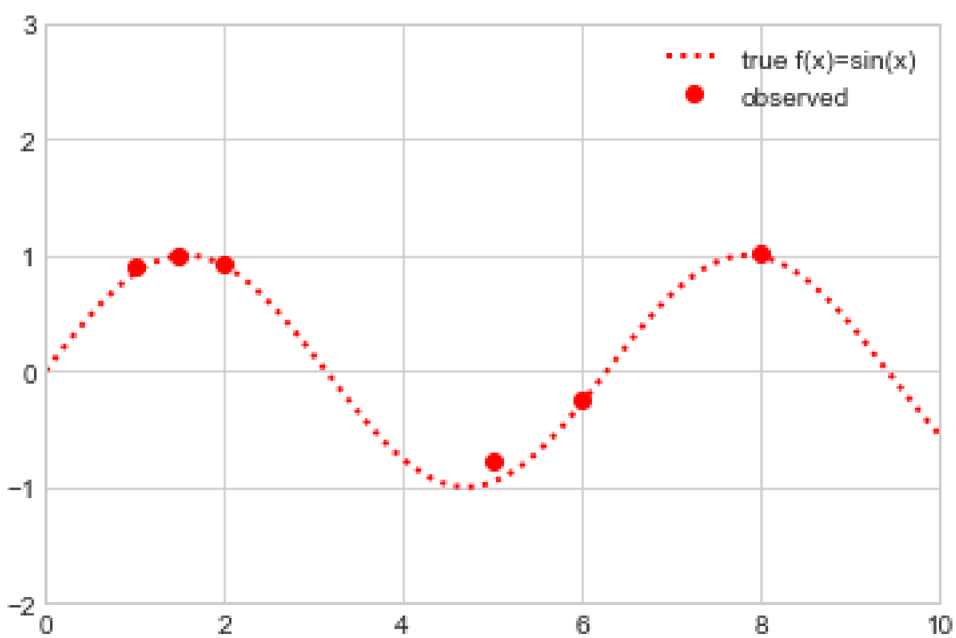

```
In [508]: plt.errorbar(x, y_measured, yerr=error, fmt='o', color='black',
                       ecolor='lightgray', elinewidth=3, capsize=0);
```



7.8 Visualizing Errors - Errortubes

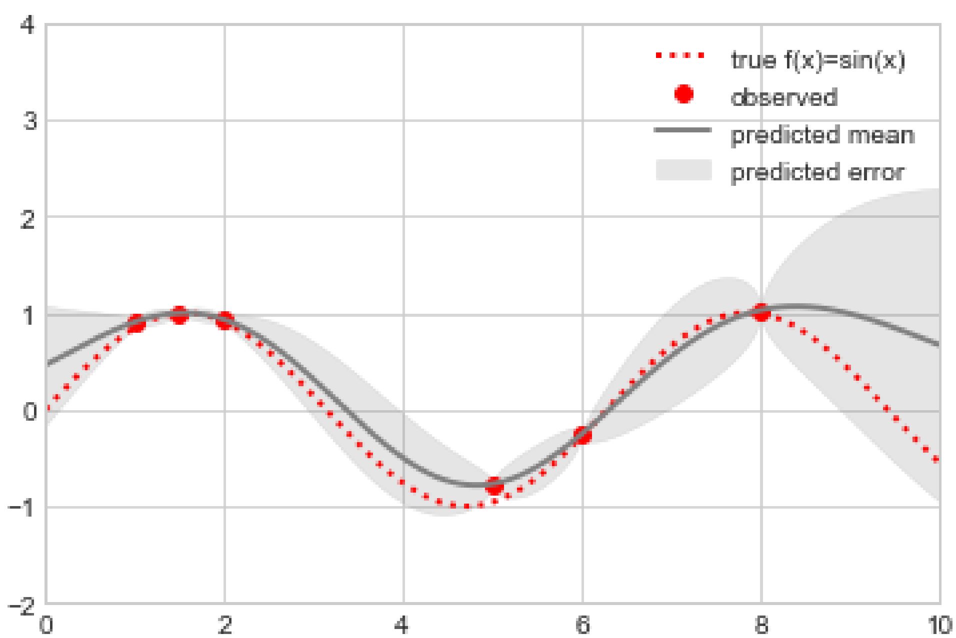
We will look at an example of Gaussian Process Regression
Taken from Scikit Learn Tutorials

In [509]:
```
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF

def f(x):
    """The function to predict."""
    return np.sin(x)

xdata = np.array([[1, 1.5, 2, 5, 6, 8]]).T
ydata = f(xdata)
ydata_with_noise = ydata + .2 * np.random.random(ydata.shape)

xfit = np.linspace(0, 10, 1000)
plt.plot(xfit, f(xfit), 'r:', label='true f(x)=sin(x)')
plt.plot(xdata, ydata_with_noise, 'or', label='observed')

plt.xlim(0, 10);
plt.ylim(-2, 3);
plt.legend();
```



In [510]:
```
# Instantiate a Gaussian Process model
kernel = RBF(.5, (1e-2, 1e1))
```

```
gp = GaussianProcessRegressor(kernel=kernel)

# Fit to data using Maximum Likelihood Estimation of the parameters
gp.fit(xdata, ydata_with_noise)

yfit, MSE = gp.predict(xfit[:, np.newaxis], return_std=True)
dyfit = 2 * np.sqrt(MSE)  # 2*sigma ~ 95% confidence region

plt.plot(xfit, f(xfit), 'r:', label='true f(x)=sin(x)')
plt.plot(xdata, ydata_with_noise, 'or', label='observed')
plt.plot(xfit, yfit, '-', color='gray', label='predicted mean')

plt.fill_between(xfit, yfit.flatten() - dyfit, yfit.flatten() + dyfit,
                 color='gray', alpha=0.2, label='predicted error')
plt.ylim(-2, 4);
plt.xlim(0, 10);
plt.legend();
```



7.9 Examples with real data

GapMinder

```
In [18]: import pandas as pd
         df = pd.read_csv("data/gapminder.csv")
         df.head()
```

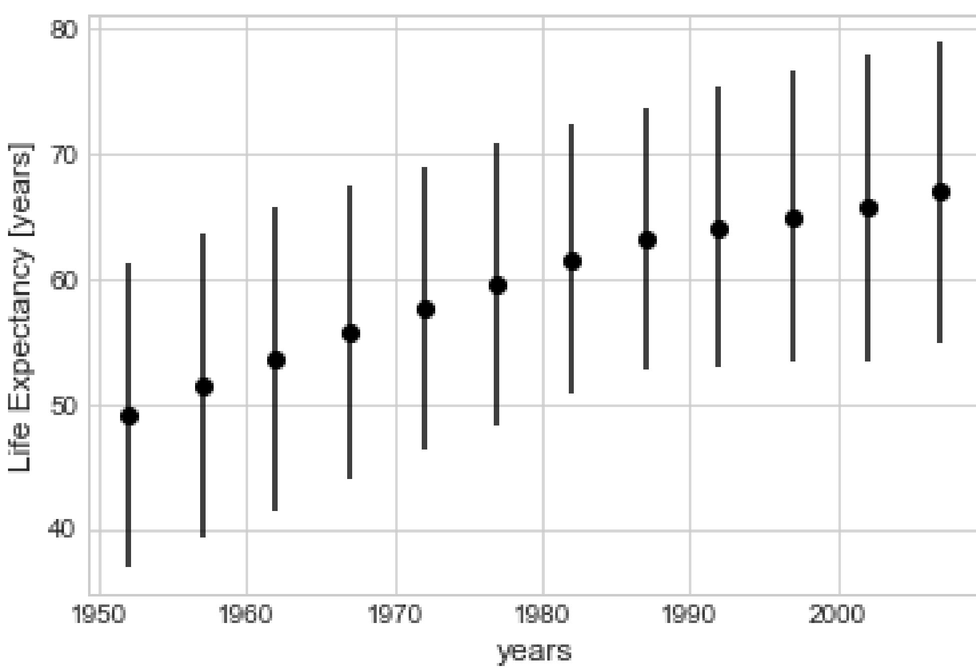
```
Out[18]:        country  year         pop continent  lifeExp   gdpPercap
         0  Afghanistan  1952   8425333.0      Asia   28.801  779.445314
         1  Afghanistan  1957   9240934.0      Asia   30.332  820.853030
         2  Afghanistan  1962  10267083.0      Asia   31.997  853.100710
         3  Afghanistan  1967  11537966.0      Asia   34.020  836.197138
         4  Afghanistan  1972  13079460.0      Asia   36.088  739.981106
```

7.9.1 Plot Life Expectancy and Gross Domestic Product (GDP)

7.9.2 Mean and Standard Deviations as Errorbars

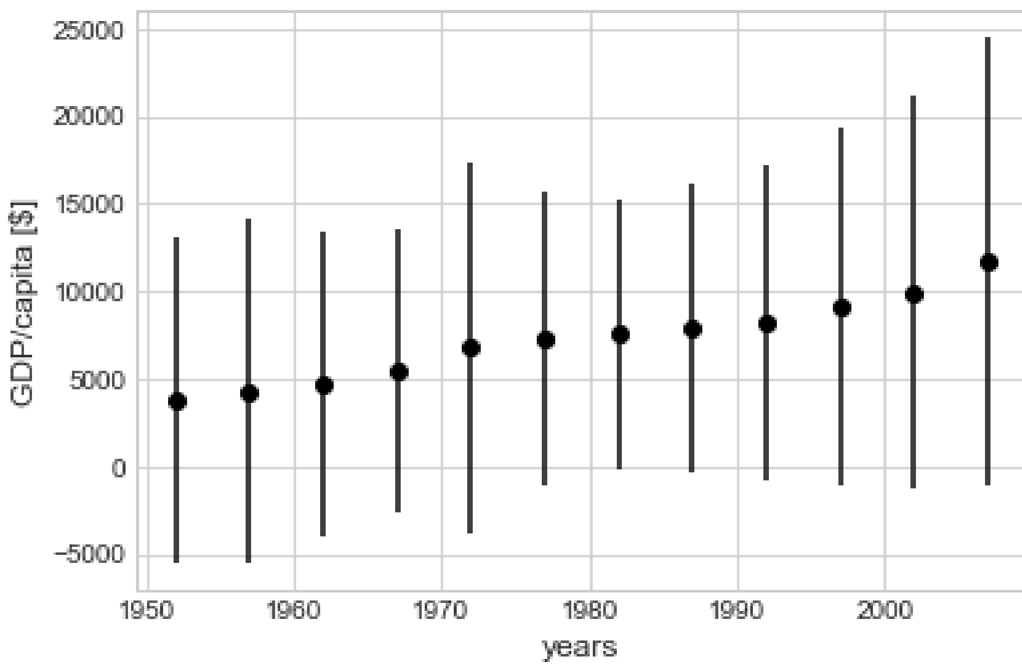
Mean and Standard Deviation are often not as informative/robust as percentiles!

```
In [21]: gdp_vs_life = df.groupby('year')\
             .agg({'gdpPercap':['mean', 'std'],'lifeExp':['mean','std']})

         plt.errorbar(gdp_vs_life.index,
                      gdp_vs_life['lifeExp']['mean'],
                      yerr=gdp_vs_life['lifeExp']['std'],
                      fmt='o', color='black')

         plt.xlabel("years")
         plt.ylabel("Life Expectancy [years]");
```




```
In [22]: plt.errorbar(gdp_vs_life.index,
                      gdp_vs_life['gdpPercap']['mean'],
```

```
                yerr=gdp_vs_life['gdpPercap']['std'],
                fmt='o', color='black')

plt.xlabel("years")
plt.ylabel("GDP/capita [$]");
```

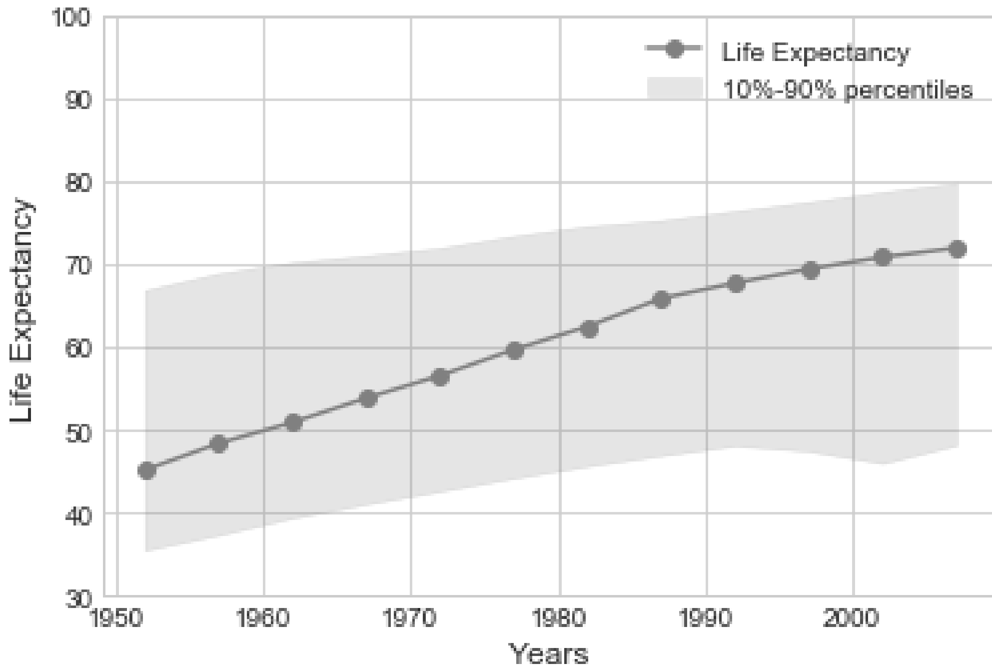


7.9.3 Median and Percentiles as Errortubes

Mean and Standard Deviation are often not as informative/robust as percentiles!

```
In [23]: life_exp = df.groupby('year')['lifeExp'].quantile(q=[.1,.5,.9]).reset_index()

         years = life_exp['year'].unique()

         plt.plot(years,
                  life_exp.loc[life_exp['level_1']==.5, 'lifeExp'],
                  '-o', color='gray', label='Life Expectancy')

         plt.fill_between(years,
                          life_exp.loc[life_exp['level_1']==.1, 'lifeExp'],
                          life_exp.loc[life_exp['level_1']==.9, 'lifeExp'],
                          color='gray', alpha=0.2, label='10%-90% percentiles')
         plt.legend()
         plt.ylim(30,100)
         plt.xlabel('Years')
         plt.ylabel('Life Expectancy')
```

Out[23]: Text(0,0.5,'Life Expectancy')



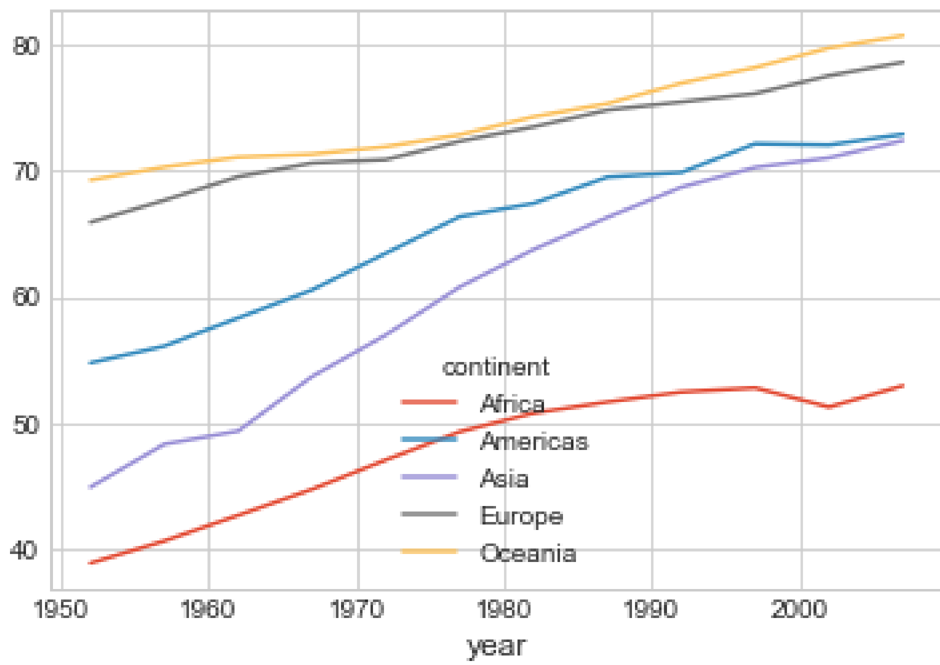
8 Plotting from pandas

- It's important to understand the plotting basics
- for modifying details
- for simple plots
- But with complex data, interactive visualization with `matplotlib` is tedious
- `pandas` allows to call `matplotlib` conveniently on DataFrames

In [ ]:
```
df = pd.read_csv("data/gapminder.csv")
life_exp_per_continent = df[['year','continent','lifeExp']].groupby(['year','continent']
life_exp_per_continent.columns = life_exp_per_continent.columns.droplevel(0)
life_exp_per_continent.head()
```

8.1 Simple Line Plots

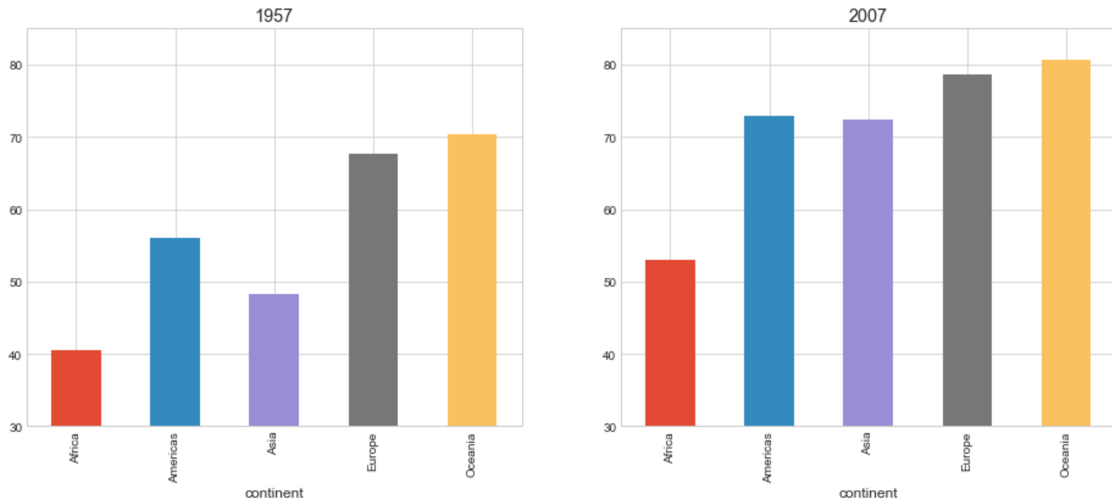
In [25]:
```
life_exp_per_continent.plot();
```



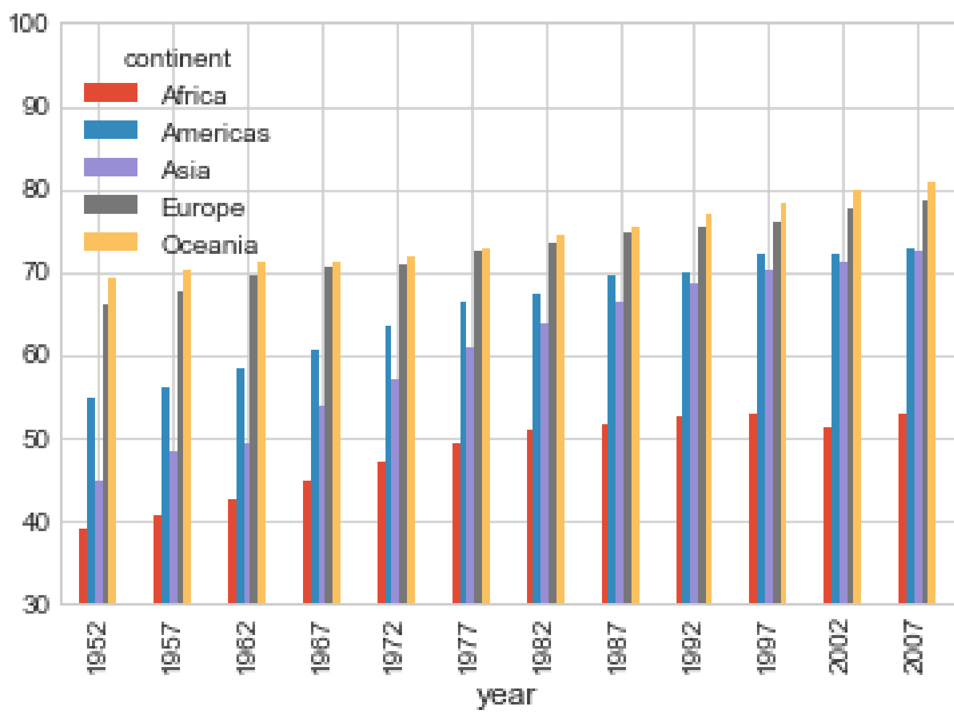
8.2 Bar Plots

8.2.1 Life Expectancy

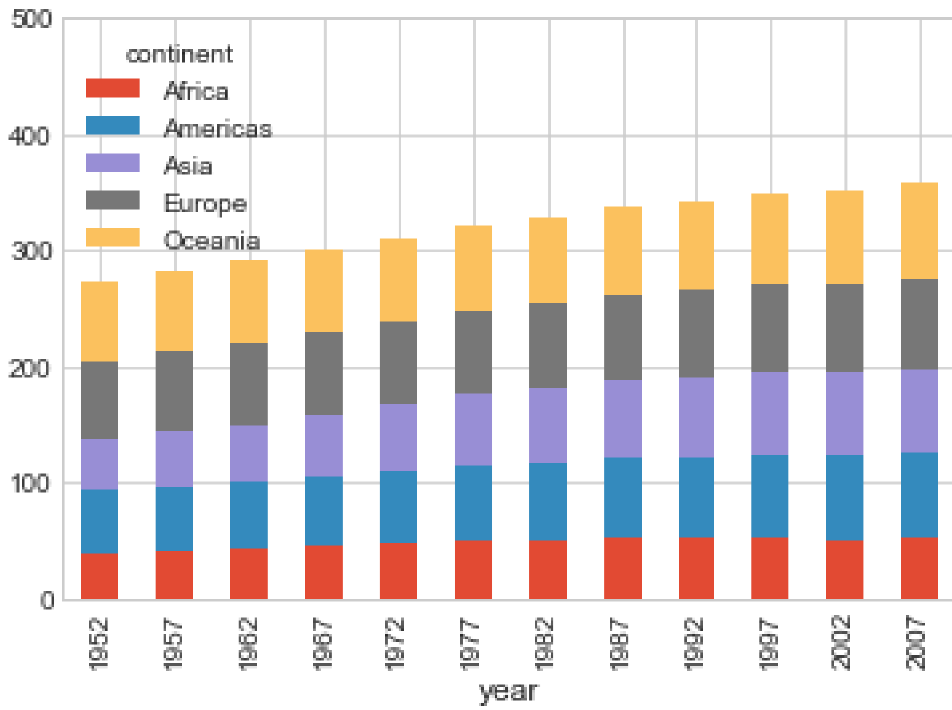
```
In [26]: plt.figure(figsize=[16,6])
         plt.subplot(1,2,1)
         life_exp_per_continent.loc[1957].plot(kind='bar');
         plt.title('1957');
         plt.ylim([30,85]);
         plt.subplot(1,2,2)
         life_exp_per_continent.loc[2007].plot(kind='bar');
         plt.title('2007');
         plt.ylim([30,85]);
```




```
In [27]: life_exp_per_continent.plot.bar();
         plt.ylim([30,100]);
```




```
In [30]: life_exp_per_continent.plot.bar(stacked=True);
         plt.ylim([0,500]);
```



8.3 Histograms

8.3.1 Overall GDP

```
In [ ]: df['gdpPercap'].hist();
        plt.ylabel("# Countries")
        plt.xlabel("GDP/Capita")
```

8.3.2 The differences in per capita GDP are huge

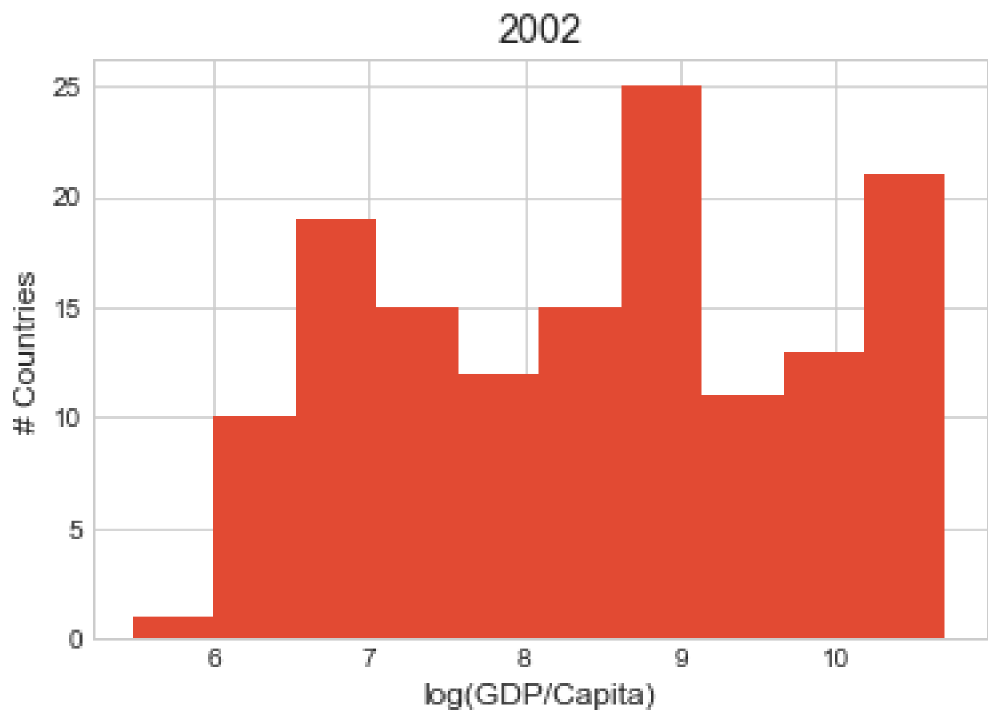
Few rich countries are much richer than the many poorer countries.

Plot is not very conclusive. How can we zoom in, without loosing the bigger picture?

8.3.3 Log-Scaling Prior to Histogram Computation

Leads to log-scaled bins and more 'normal' looking distribution

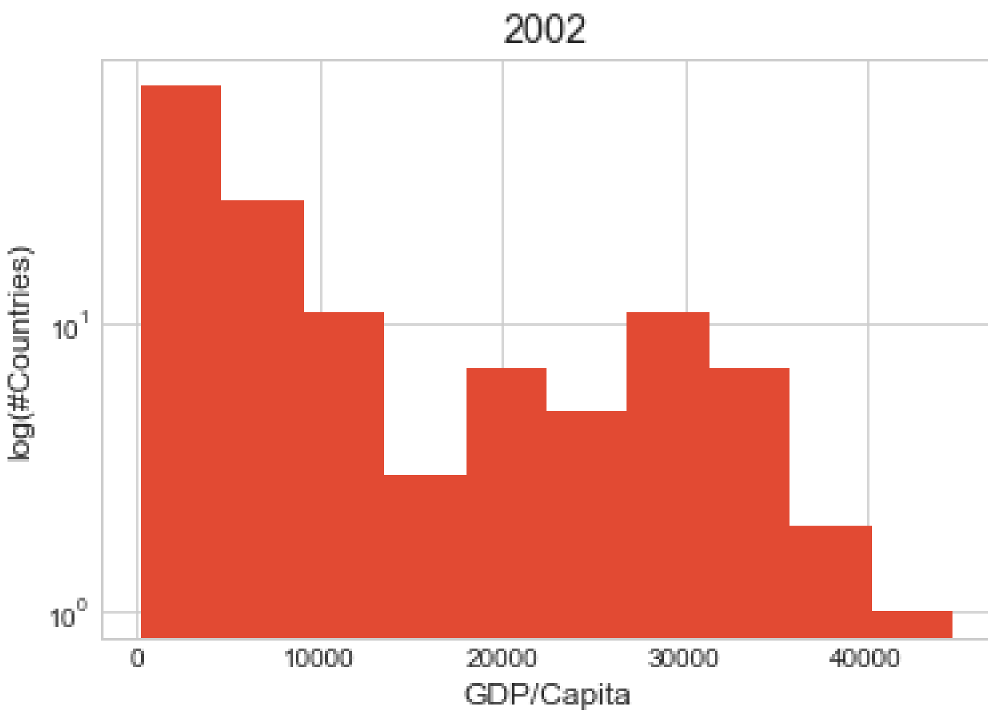
```
In [32]: df.loc[df.year==2002, 'gdpPercap'].apply(np.log).hist();
         plt.ylabel("# Countries")
         plt.xlabel("log(GDP/Capita)");
```



8.3.4 Log-Scaling Y-Axis

Linearly scaled bins but log-scaled histogram counts

```
In [33]: df.loc[df.year==2002, 'gdpPercap'].hist();
         plt.yscale('log');
         plt.ylabel('log(#Countries)');
         plt.xlabel("GDP/Capita");
```



8.3.5 GDP per Continent

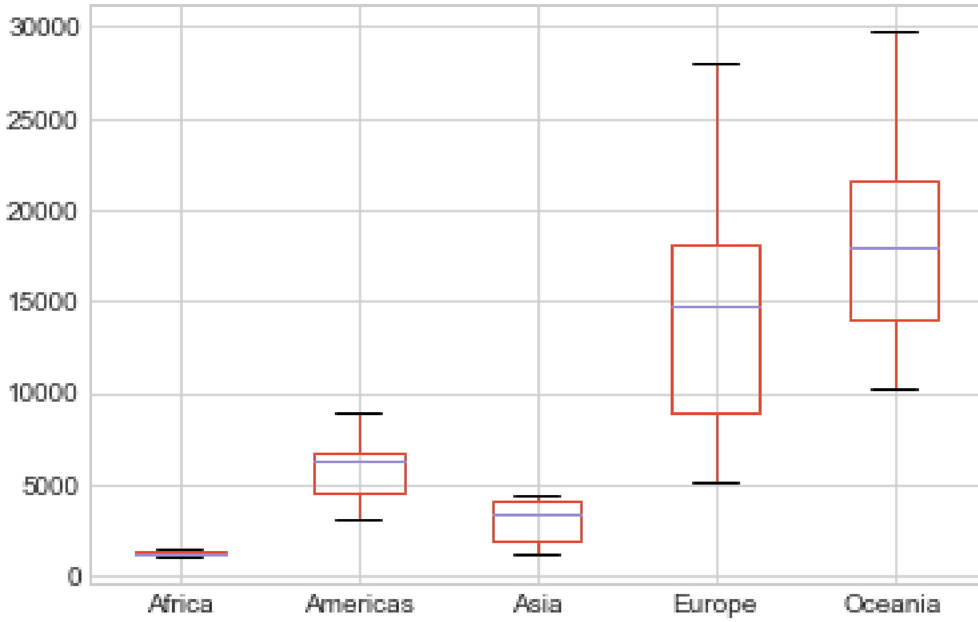
```
In [35]: gdp_per_continent = df[['year','continent','gdpPercap']]\
                .groupby(['year','continent']).median().unstack()
         gdp_per_continent.columns = gdp_per_continent.columns.droplevel(0)
         gdp_per_continent.head()
```

Out[35]:

continent	Africa	Americas	Asia	Europe	Oceania
year					
1952	987.025569	3048.302900	1206.947913	5142.469716	10298.085650
1957	1024.022987	3780.546651	1547.944844	6066.721495	11598.522455
1962	1133.783677	4086.114078	1649.552153	7515.733737	12696.452430
1967	1210.376379	4643.393534	2029.228142	9366.067033	14495.021790
1972	1443.372508	5305.445256	2571.423014	12326.379990	16417.333380

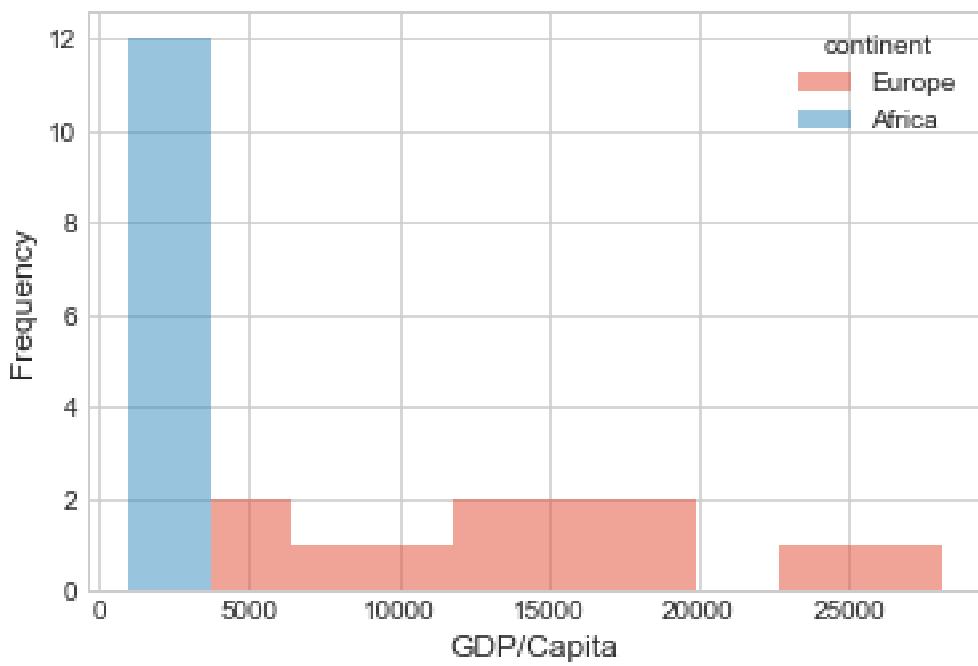
8.3.6 Box Plots

```
In [36]: gdp_per_continent.plot.box();
```

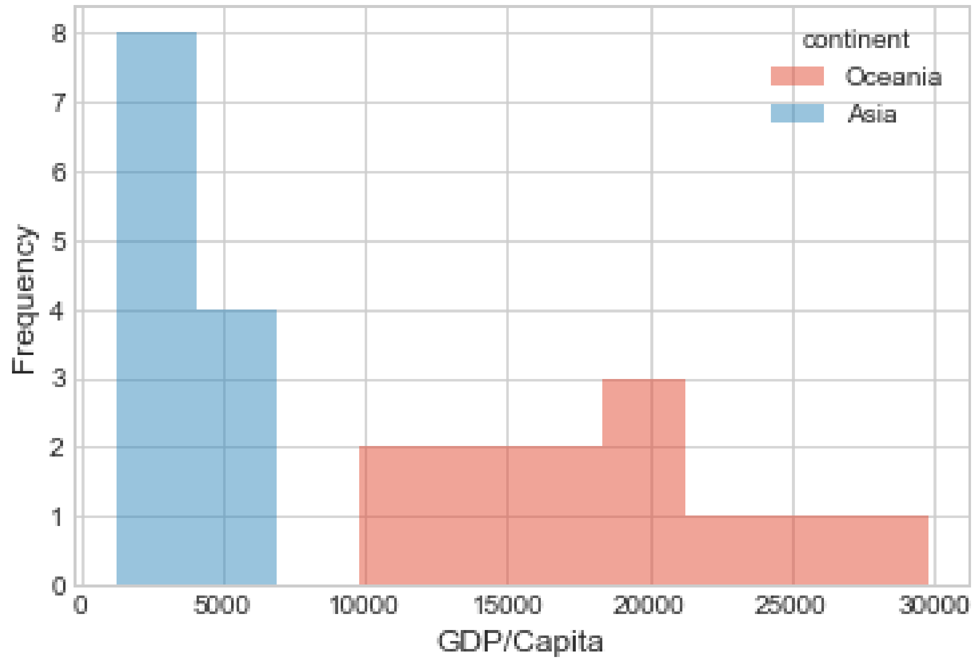


8.3.7 Histograms

```
In [37]: gdp_per_continent[['Europe','Africa']].plot.hist(alpha=0.5);
         plt.xlabel("GDP/Capita");
```



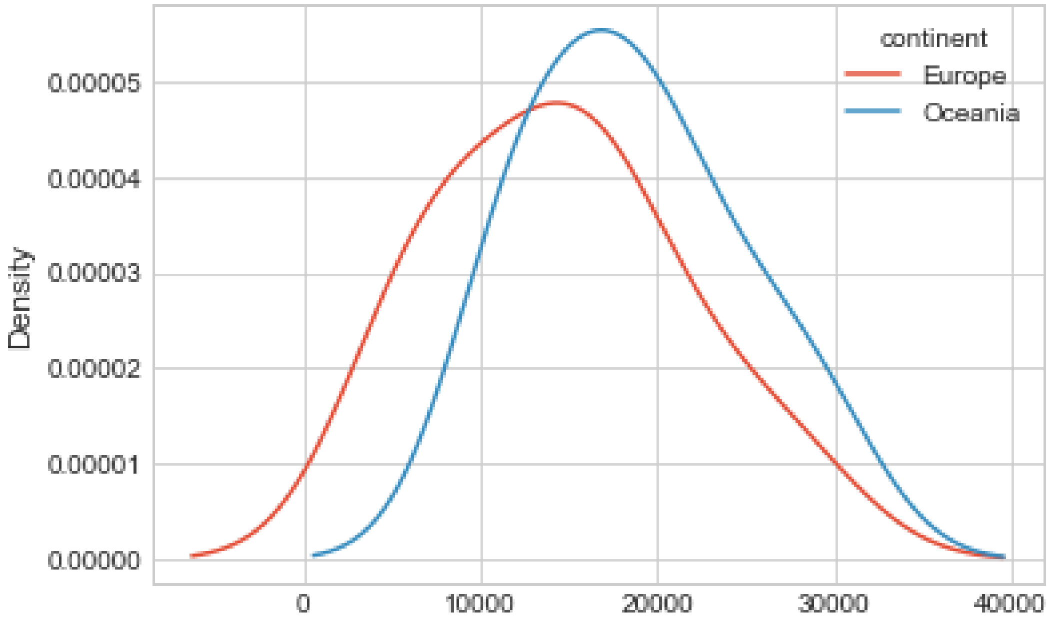
```
In [38]: gdp_per_continent[['Oceania','Asia']].plot.hist(alpha=0.5);
         plt.xlabel("GDP/Capita");
```



8.3.8 Kernel Density Estimate Plots

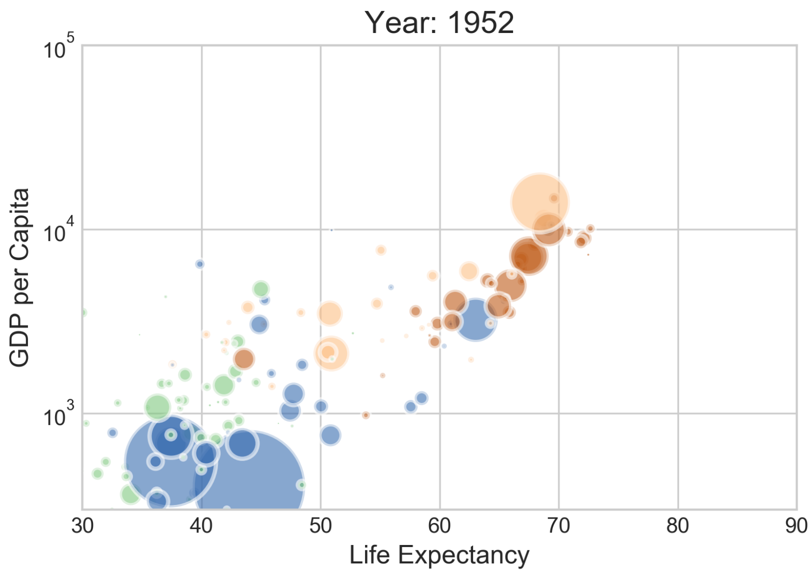
Smoothed histograms

```
In [44]: gdp_per_continent[['Europe','Oceania']].plot.kde();
```



9 GapMinder Demo

```
In [42]: data = pd.read_csv("data/gapminder.csv")

         # Transform Continent into numerical values group1->1, group2->2...
         data['continent']=pd.Categorical(data['continent'])

         def plot_gapminder(year, filename=''):
             fig = plt.figure()
             ax = plt.axes()
             year = data.loc[abs(data.year.unique() - year).argmin(),'year']
             # Change color for the x-axis values
             tmp=data[ data.year == year ]
             ax.scatter(tmp['lifeExp'], tmp['gdpPercap'] , s=tmp['pop']/200000 ,
                        c=tmp['continent'].cat.codes, cmap="Accent", alpha=0.6,
                        edgecolors="white")

             # Add titles (main and on axis)
             ax.set_yscale('log')
             ax.set_xlabel("Life Expectancy")
             ax.set_ylabel("GDP per Capita")
             ax.set_title("Year: "+str(year) )
             ax.set_ylim(0,100000)
             ax.set_xlim(30, 90)
             if filename:
```



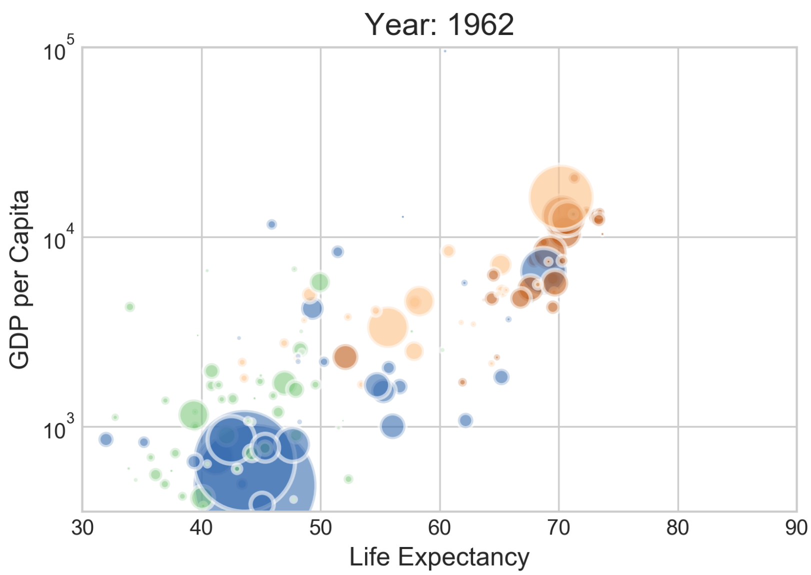
1952

```
        plt.savefig(filename)
    else:
        plt.show()
```

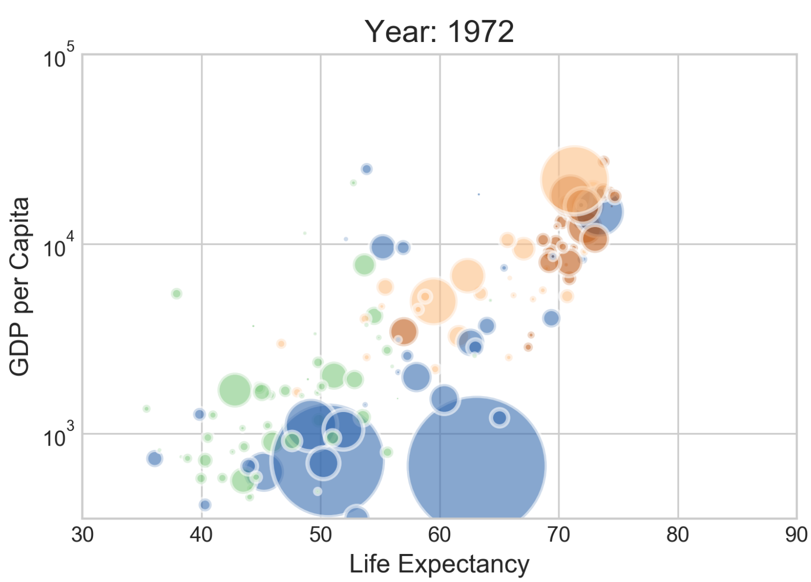
```
In [578]: from ipywidgets import interact
          import ipywidgets as widgets

          def plot_gapminder_no_savefig(year):
              plot_gapminder(year, "")

          interact(plot_gapminder_no_savefig, year=widgets.IntSlider(min=1952,max=2007,step=1,va
```

```
interactive(children=(IntSlider(value=1952, description='year', max=2007, min=1952), Output()),
```

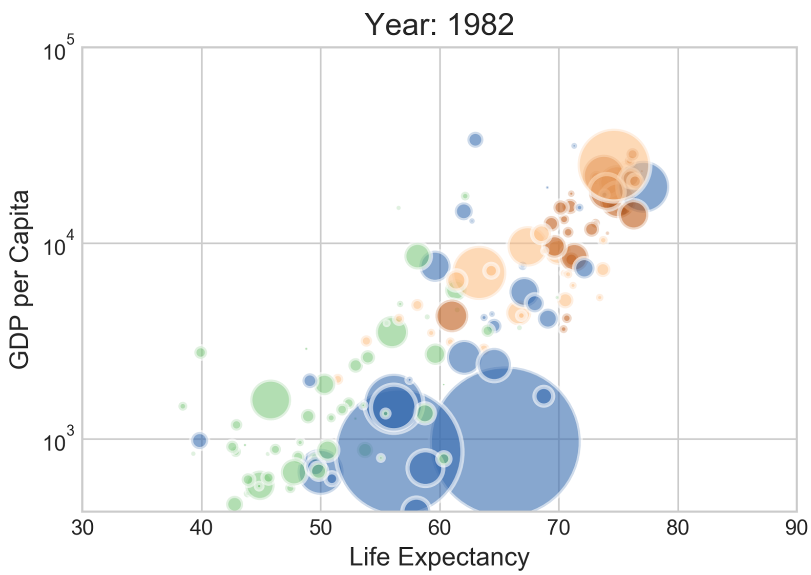
```
In [ ]: for year in data.year.unique():
            filename='plots/Gapminder_step'+str(year)+'.png'
            plot_gapminder(year, filename=filename)
```


Year: 1962
GDP per Capita
Life Expectancy
30
40
50
60
70
80
90

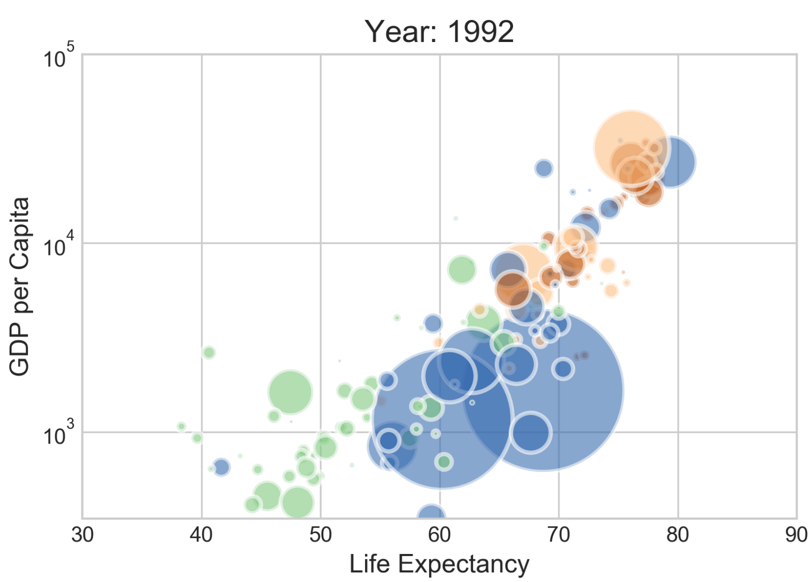
1962


Year: 1972
GDP per Capita
Life Expectancy
30
40
50
60
70
80
90

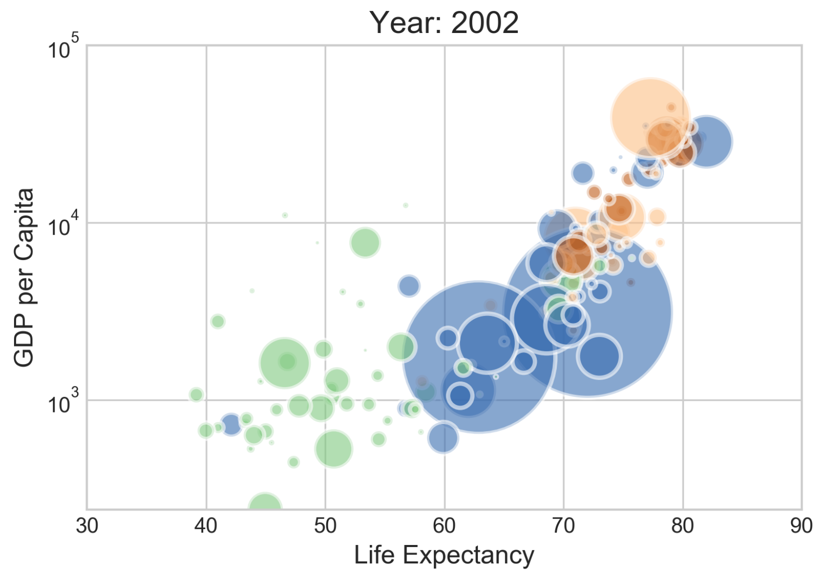
1972


Year: 1982
GDP per Capita
Life Expectancy
10^5
10^4
10^3
30
40
50
60
70
80
90

1982


Year: 1992
GDP per Capita
Life Expectancy
10^5
10^4
10^3
30
40
50
60
70
80
90

1992



2002

9.1 Life Expectancy vs GDP vs Time

9.2 Life Expectancy vs GDP vs Time

9.3 Life Expectancy vs GDP vs Time

9.4 Life Expectancy vs GDP vs Time

9.5 Life Expectancy vs GDP vs Time

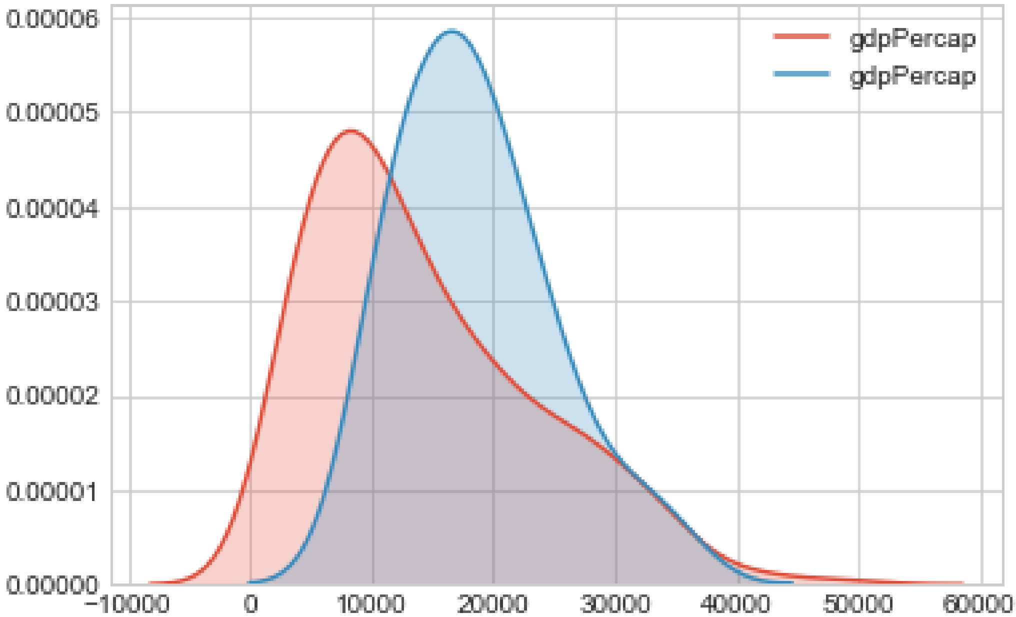
9.6 Life Expectancy vs GDP vs Time

10 Plotting with `seaborn`

- Matplotlib is too low level for complex interactive visualization
- pandas allows for simpler API
- Many great plots you'll find use abstractions on matplotlib

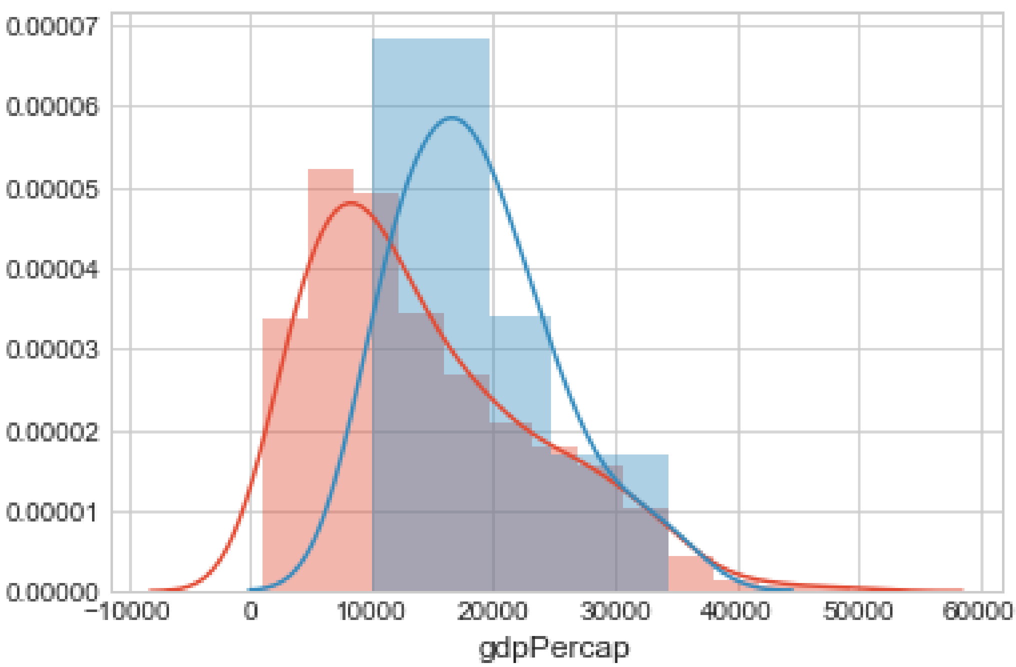
10.1 KDE Plots

```
In [74]: import seaborn as sns
         data = pd.read_csv("data/gapminder.csv")
         sns.kdeplot(data.loc[data.continent=='Europe','gdpPercap'], shade=True);
         sns.kdeplot(data.loc[data.continent=='Oceania','gdpPercap'], shade=True);
```



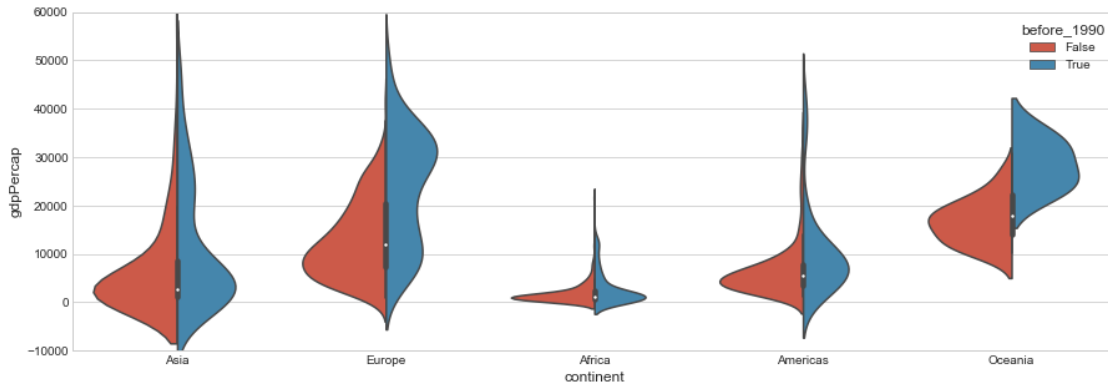
10.2 Histograms and Density Plots

```
In [75]: sns.distplot(data.loc[data.continent=='Europe','gdpPercap']);
         sns.distplot(data.loc[data.continent=='Oceania','gdpPercap']);
```

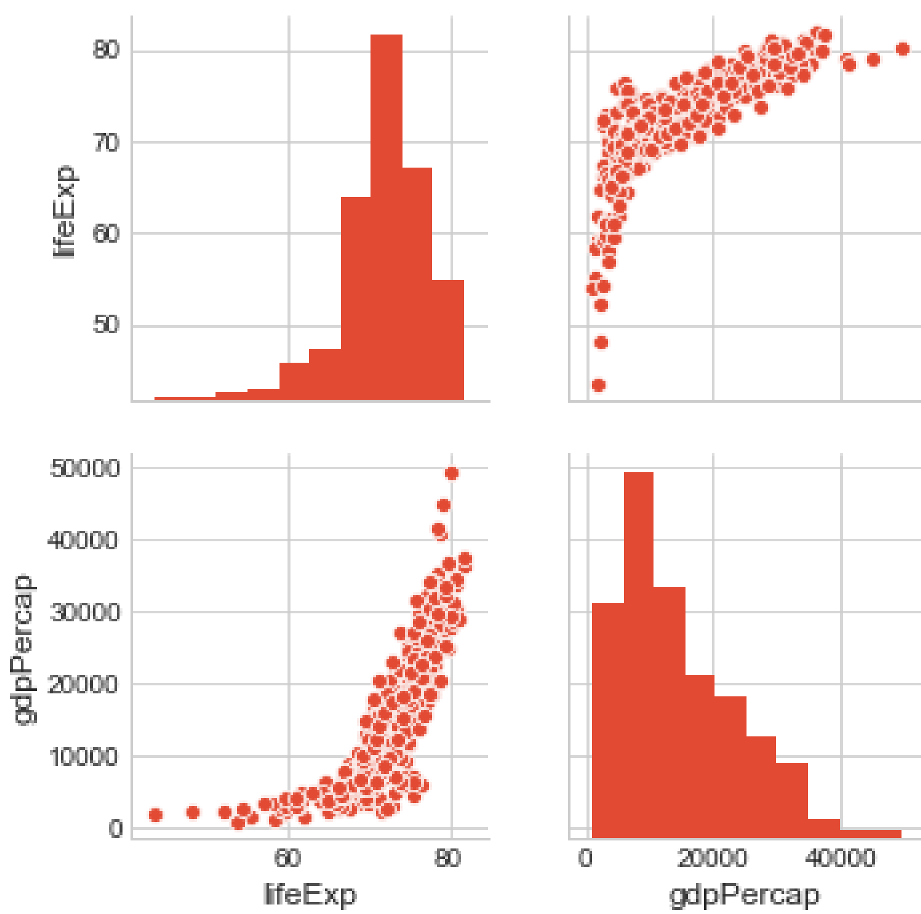


10.3 Violin Plots

```
In [109]: plt.figure(figsize=[15,5])
          data['before_1990'] = data.year>2000
          sns.violinplot( "continent", "gdpPercap", hue='before_1990', data=data, split=True);
          plt.ylim([-10000, 60000]);
```




```
In [112]: sns.pairplot(data.loc[data.continent=='Europe',['lifeExp','gdpPercap']]);
```



11 Plotting Maps with `folium`

- `leaflet.js` is a great javascript library for web based map visualizations
- `folium` let's you leverage python's flexibility with `leaflet`'s visualization strengths

```
In [ ]: !pip install folium
```

```
In [204]: import folium
          m = folium.Map(location=[52.545195, 13.354670], zoom_start=16)
          m
```

```
Out[204]: <folium.folium.Map at 0x1a407be4a8>
```

```
In [205]: folium.Marker(
              location=[52.545195, 13.354670], popup='Where we are.',
              icon=folium.Icon(icon='cloud')
          ).add_to(m)

          folium.Marker(
              location=[52.543941, 13.355539], popup='Where the ponies live.',
              icon=folium.Icon(icon='cloud')
          ).add_to(m)
          m
```

```
Out[205]: <folium.folium.Map at 0x1a407be4a8>
```

12 Exercises

- In each exercise you will plot some data.
- The resulting plots should be saved to a file and handed in.
- Solutions will be evaluated based on the figures and the code.

12.1 Assignment 01

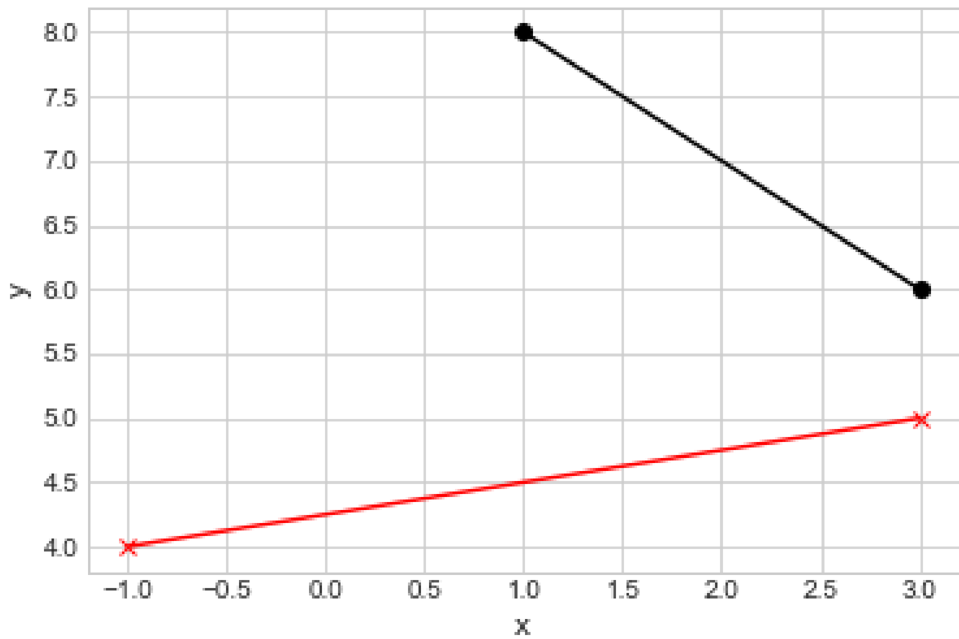
Plot some lines

Write a function `assignment_05_01` that draws a line from the point [-1,4] to the point [3,5] with line style `'r-x'` and another line from [3,6] to [1,8] with line style `'k-o'`.

```
def assignment_05_01():
    x = [-1,3]
    y = [4,5]
    ...
    plt.xlabel('x')
    plt.ylabel('y')
    plt.savefig("assignment_05_01.png")
```

```
In [118]: from IPython.display import Image
          Image("assignment_05_01.png")
```

Out[118]:



12.2 Assignment 02

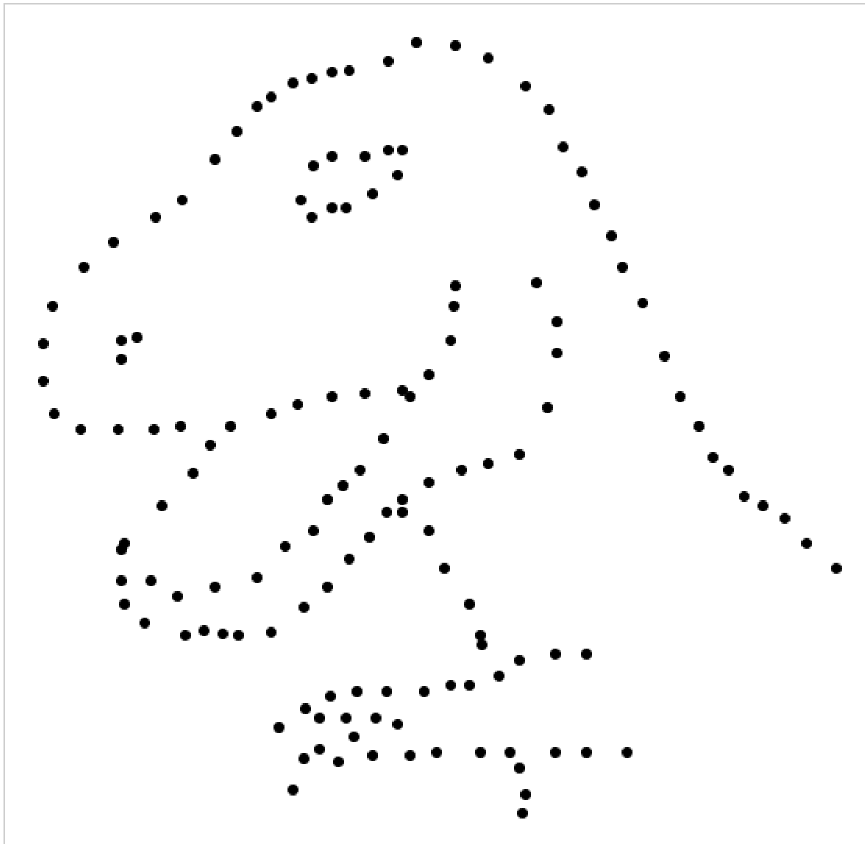
Plot a dinosaur

Write a function `assignment_05_02` that draws the data set 'dino' from the tab-separated file `data/DatasaurusDozen.tsv`.

```
def assignment_05_02():
    ds = pd.read_csv('data/DatasaurusDozen.tsv',sep='\t')
    plt.figure(figsize=[10,10])
    ...
    plt.xticks([])
    plt.yticks([])
    plt.savefig("assignment_05_02.png")
```

```
In [131]: from IPython.display import Image
          Image("assignment_05_02.png")
```

Out[131]:



12.3 Assignment 03

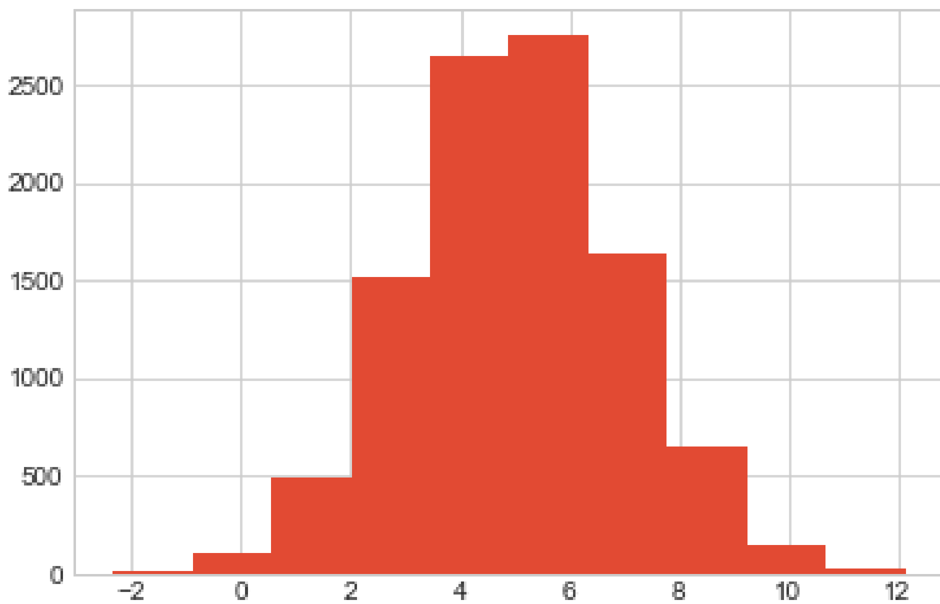
Plot some histograms

Write a function `assignment_05_03` creates a normally distributed variable with mean 5 and standard deviation 2. Plot the histogram and save the file as `assignment_05_03.png`.

```
def assignment_05_03():
    ...
    plt.savefig("assignment_05_03.png")
```

```
In [186]: from IPython.display import Image
          Image("assignment_05_03.png")
```

Out[186]:



12.4 Assignment 04

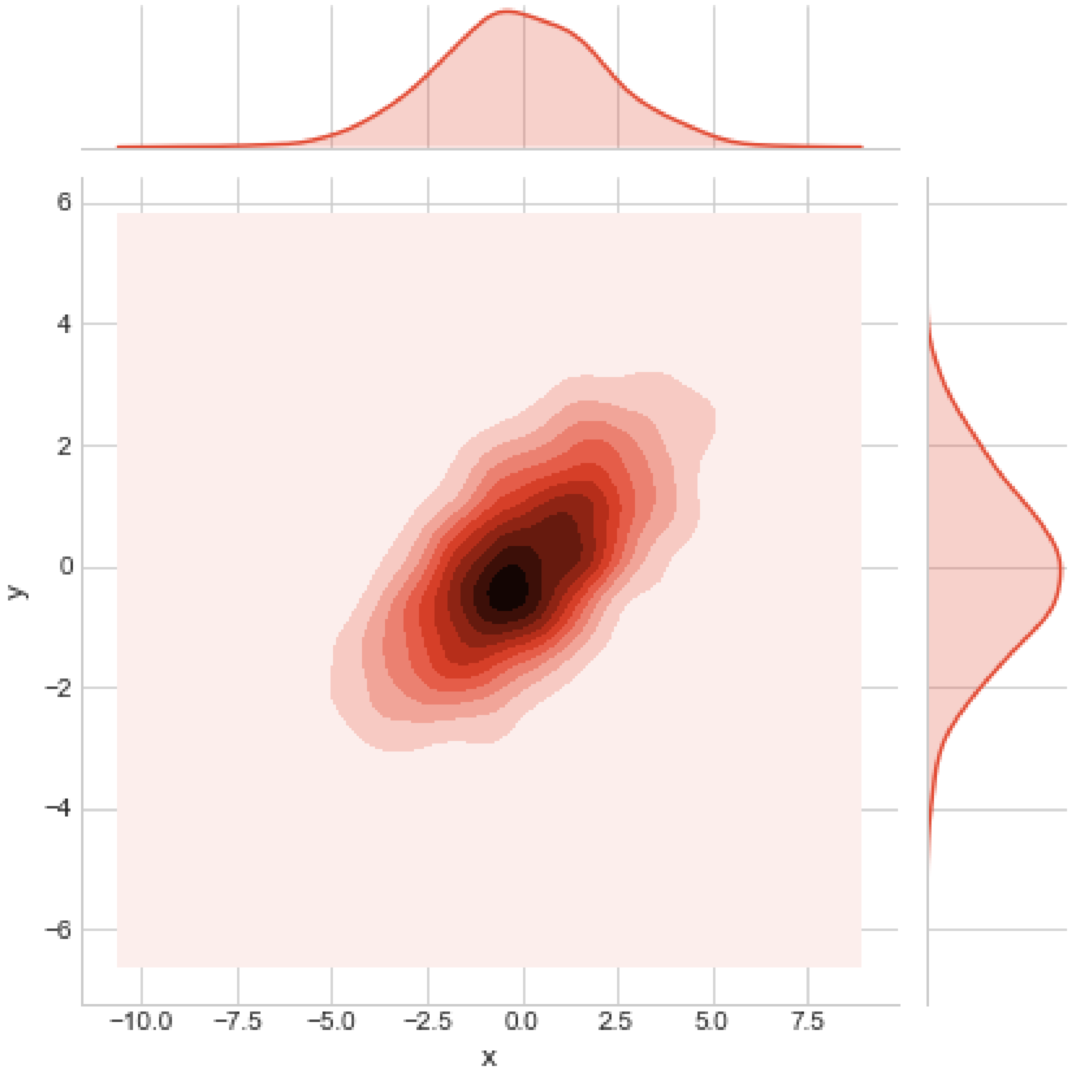
Plot some 2D Kernel Density Estimates

Write a function `assignment_05_04` creates a 2D normally distributed variable with mean [0,0] and covariance matrix [[5, 2], [2, 2]]. Plot the joint kernel density estimate and save the file as `assignment_05_04.png`.

```
def assignment_05_04():
    import seaborn as sns
    ...
    normal_data = pd.DataFrame(data, columns=['x', 'y'])
    sns.jointplot("x", "y", normal_data, kind='kde');
    plt.savefig("assignment_05_04.png")
```

```
In [197]: from IPython.display import Image
          Image("assignment_05_04.png")
```

Out[197]:



6
4
2
0
−2
−4
−6
y
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
x