

Data Visualization With Python

June 14, 2018



Course Outline

<https://www.datacamp.com/courses/introduction-to-data-visualization-with-python>

- **Python tools**
 - **Matplotlib**
 - **Seaborn**
- **Exercises**
 - **Plotting skills: common axes, axes in figure, subplots**
 - **Customization of plots**
 - **Regressions / residuals / grouping**
 - **Distributions: strip, swarm and violin plots**
- **Bonus**
 - **2D plots**
 - **Turtle**

<https://matplotlib.org/tutorials/index.html#introductory>

Tutorials — Matplotlib x scatter plot python x #40 Basic scatterplot x python read csv into x Using the CSV module x python - How to read x matplotlib built into x

Secure | <https://matplotlib.org/tutorials/index.html#introductory>

The 2018 SciPy John Hunter Excellence in Plotting Contest is accepting submissions until June 8th!

matplotlib

Version 2.2.2

home | examples | tutorials | pyplot | docs » User's Guide »

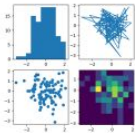
Tutorials

This page contains more in-depth guides for using Matplotlib. It is broken up into beginner, intermediate, and advanced sections, as well as sections covering specific topics.

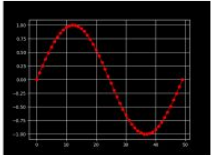
For shorter examples, see our [examples page](#). You can also find [external resources](#) and a [FAQ](#) in our [user guide](#).

Introductory

These tutorials cover the basics of creating visualizations with Matplotlib, as well as some best-practices in using the package effectively.



Sample plots in Matplotlib



Customizing matplotlib


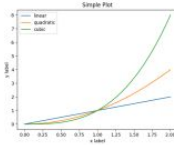


Image tutorial



Usage Guide

Quick search

Go

Table Of Contents

Tutorials

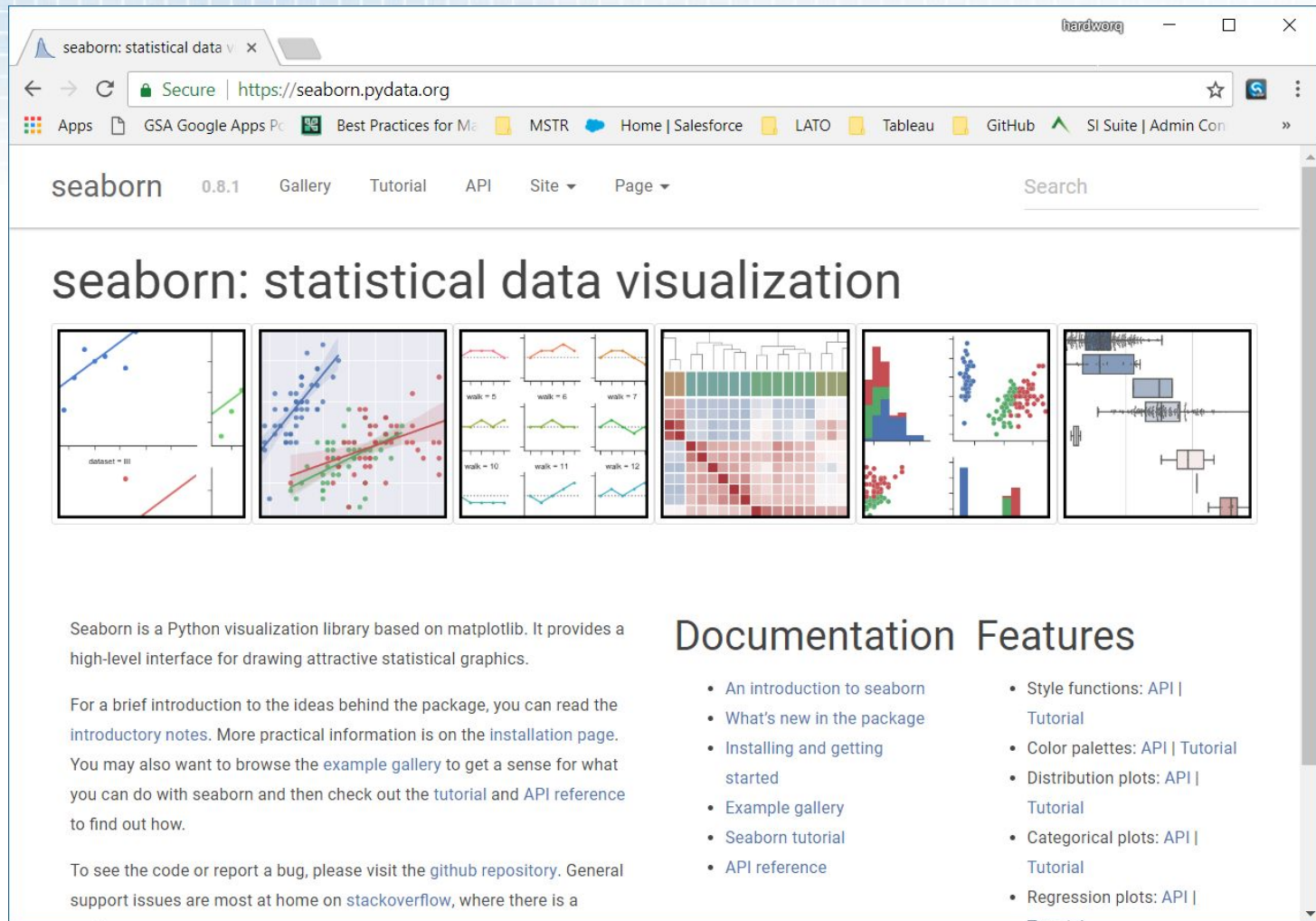
- [Introductory](#)
- [Intermediate](#)
- [Advanced](#)
- [Colors](#)
- [Text](#)
- [Toolkits](#)

Related Topics

Documentation overview

- [User's Guide](#)
 - Previous: [Installing](#)
 - Next: [Sample plots in Matplotlib](#)

<https://seaborn.pydata.org/>



The screenshot shows the seaborn website in a web browser. The browser's address bar displays the URL <https://seaborn.pydata.org/>. The website's navigation bar includes links for "seaborn", "0.8.1", "Gallery", "Tutorial", "API", "Site", and "Page". A search bar is located on the right side of the navigation bar. The main heading on the page is "seaborn: statistical data visualization". Below this heading is a grid of six thumbnail images showcasing different types of statistical plots created with seaborn, including scatter plots with regression lines, faceted line plots, a heatmap, a histogram, and a box plot. The text on the page describes seaborn as a Python visualization library based on matplotlib, providing a high-level interface for drawing attractive statistical graphics. It also provides links to introductory notes, installation page, example gallery, tutorial, and API reference. A section titled "Documentation Features" lists various features and links to their respective documentation pages.

seaborn 0.8.1 Gallery Tutorial API Site Page Search

seaborn: statistical data visualization

Seaborn is a Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics.

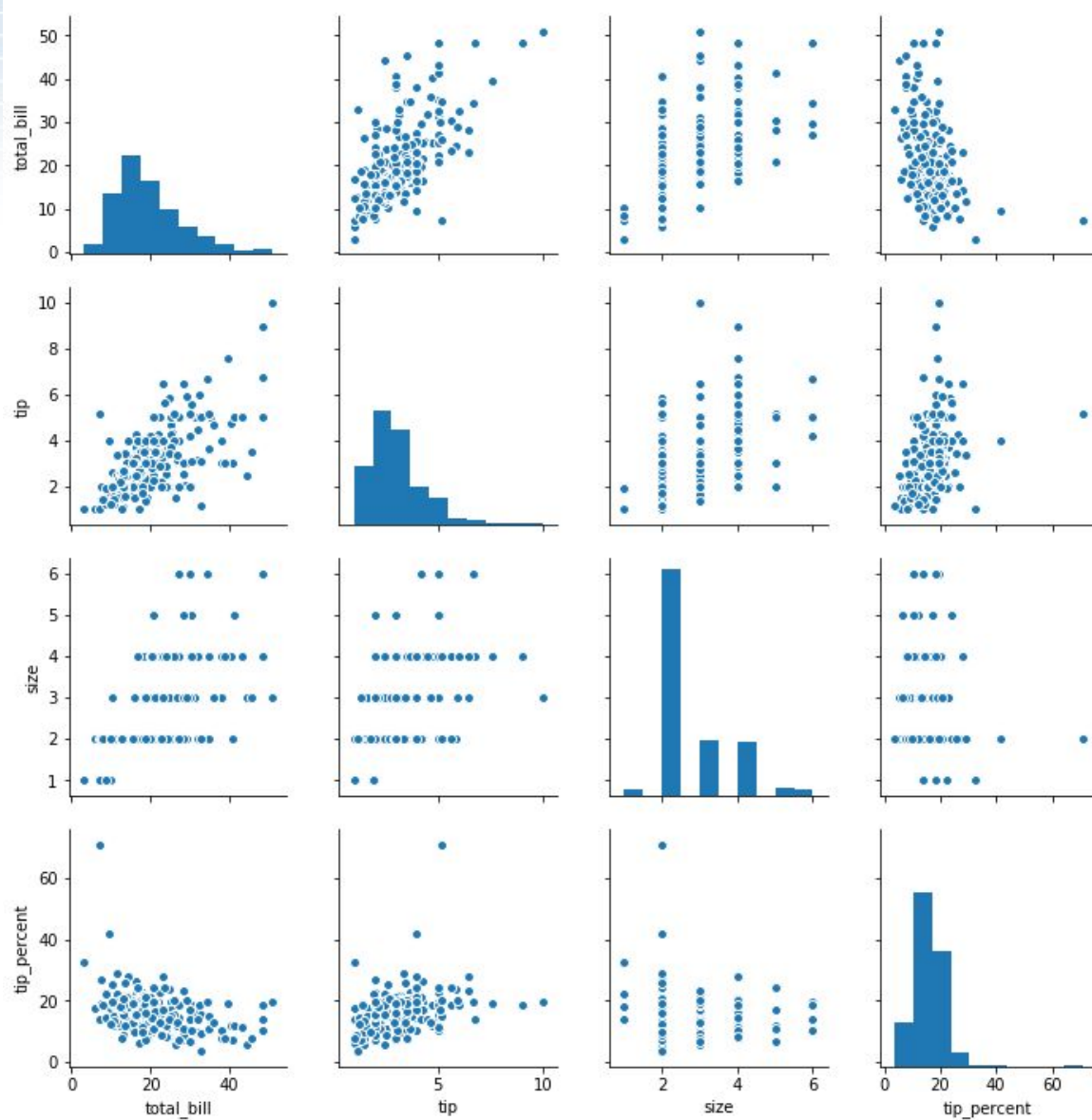
For a brief introduction to the ideas behind the package, you can read the [introductory notes](#). More practical information is on the [installation page](#). You may also want to browse the [example gallery](#) to get a sense for what you can do with seaborn and then check out the [tutorial](#) and [API reference](#) to find out how.

To see the code or report a bug, please visit the [github repository](#). General support issues are most at home on [stackoverflow](#), where there is a

Documentation Features

- [An introduction to seaborn](#)
- [What's new in the package](#)
- [Installing and getting started](#)
- [Example gallery](#)
- [Seaborn tutorial](#)
- [API reference](#)
- [Style functions: API | Tutorial](#)
- [Color palettes: API | Tutorial](#)
- [Distribution plots: API | Tutorial](#)
- [Categorical plots: API | Tutorial](#)
- [Regression plots: API | Tutorial](#)

Copy / save plot image w/ right click

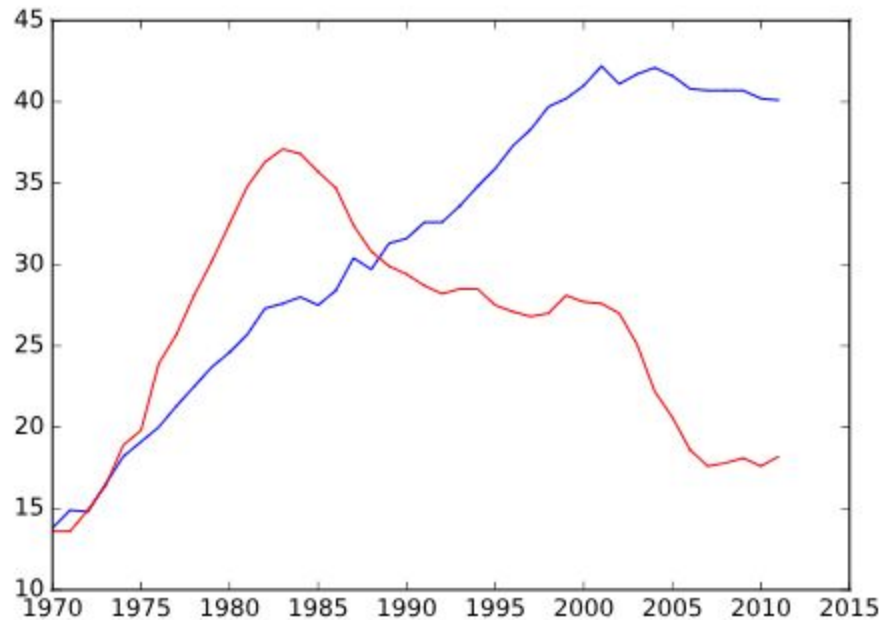


Customizing plots



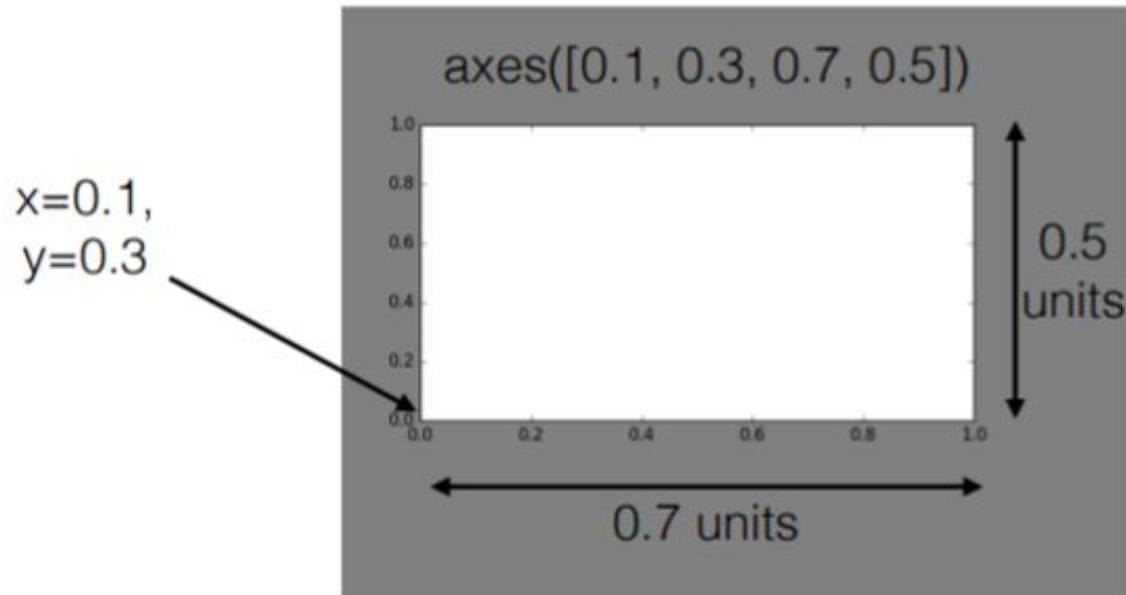
Multiple plots on single axis

```
1 # Import matplotlib.pyplot
2 import matplotlib.pyplot as plt
3
4 # Plot in blue the % of degrees awarded to
  women in the Physical Sciences
5 plt.plot(year, physical_sciences, color
  ='blue')
6
7 # Plot in red the % of degrees awarded to
  women in Computer Science
8 plt.plot(year, computer_science, color
  ='red')
9
10 # Display the plot
11 plt.show()
12
```



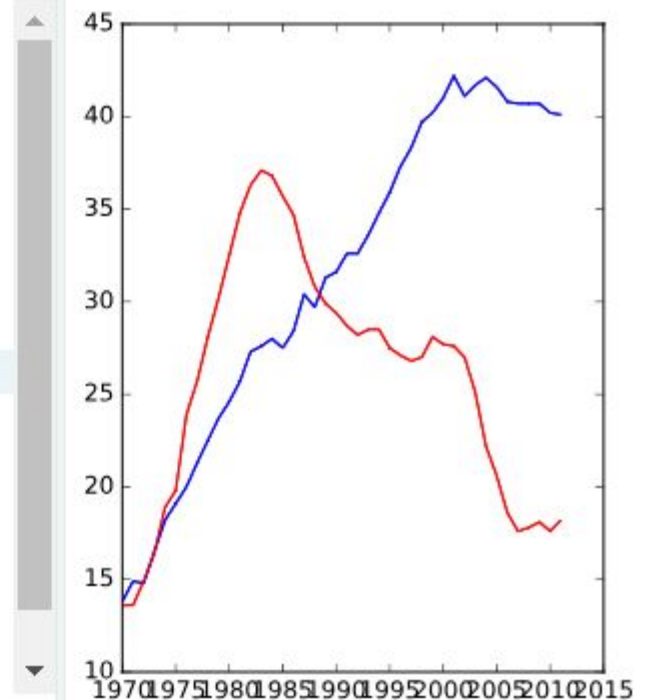
axes() command

- Syntax: `axes([x_lo, y_lo, width, height])`
- Units between 0 and 1 (figure dimensions)



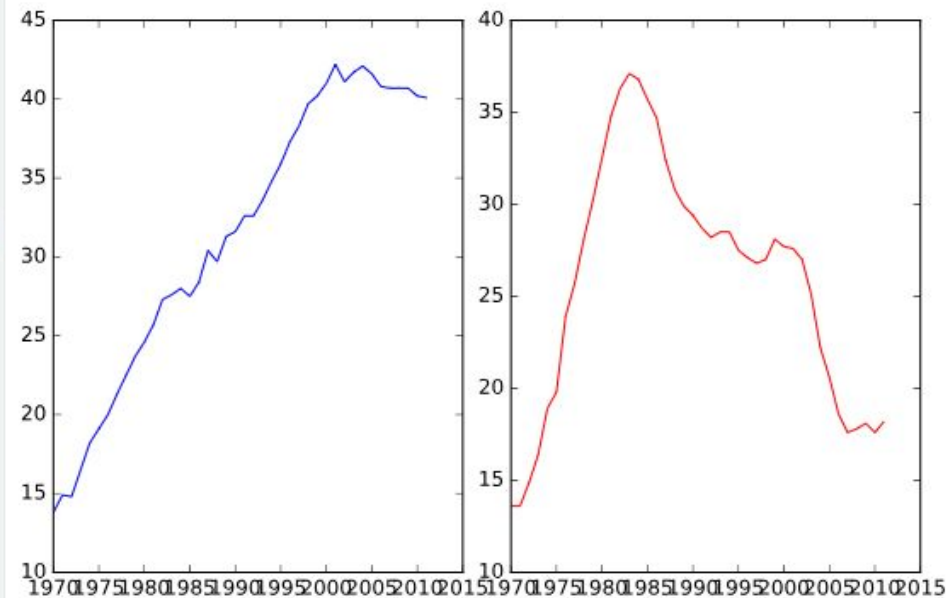
Using axes()

```
1 # Create plot axes for the first line plot
2 plt.axes([0.05, 0.05, 0.425, 0.9])
3
4 # Plot in blue the % of degrees awarded to women in
  the Physical Sciences
5 plt.plot(year, physical_sciences, color='blue')
6
7 # Create plot axes for the second line plot
8 plt.axes([0.05, 0.05, 0.425, 0.9])
9
10 # Plot in red the % of degrees awarded to women in
   Computer Science
11 plt.plot(year, computer_science, color='red')
12
13 # Display the plot
14 plt.show()
```



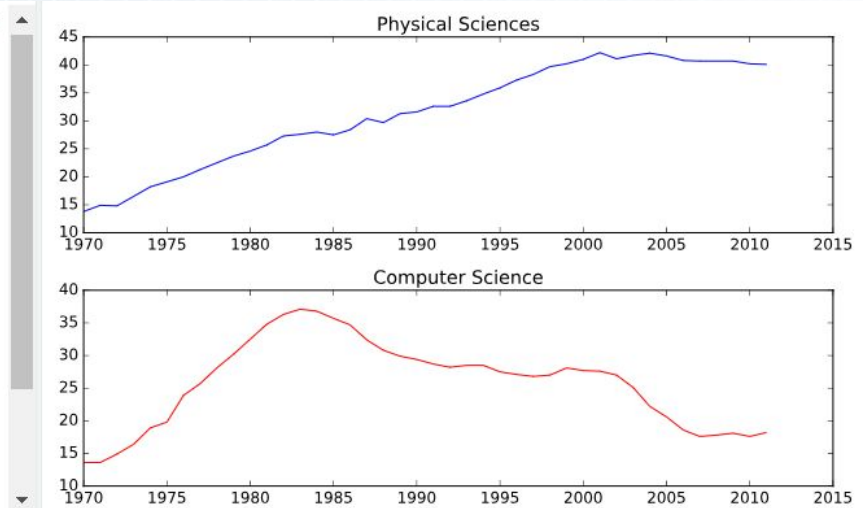
Using axes() (2)

```
1 # Create plot axes for the first line plot
2 plt.axes([0.05, 0.05, 0.425, 0.9])
3
4 # Plot in blue the % of degrees awarded to women in
  the Physical Sciences
5 plt.plot(year, physical_sciences, color='blue')
6
7 # Create plot axes for the second line plot
8 plt.axes([0.525, 0.05, 0.425, 0.9])
9
10 # Plot in red the % of degrees awarded to women in
   Computer Science
11 plt.plot(year, computer_sciences, color='red')
12
13 # Display the plot
14 plt.show()
15
```

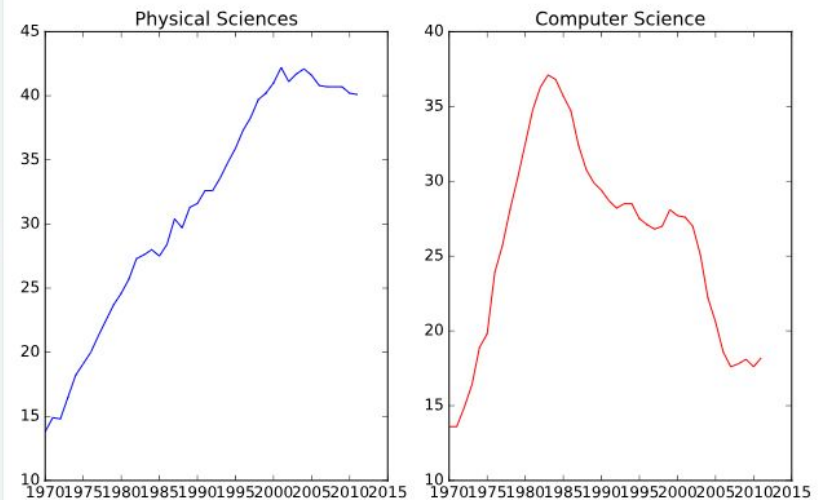


Using subplot() (1)

```
1 # Create a figure with 1x2 subplot and make the left subplot active
2 plt.subplot(2,1,1)
3
4 # Plot in blue the % of degrees awarded to women in the Physical Sciences
5 plt.plot(year, physical_sciences, color='blue')
6 plt.title('Physical Sciences')
7
8 # Make the right subplot active in the current 1x2 subplot grid
9 plt.subplot(2,1,2)
10
11 # Plot in red the % of degrees awarded to women in Computer Science
12 plt.plot(year, computer_science, color='red')
13 plt.title('Computer Science')
14
```



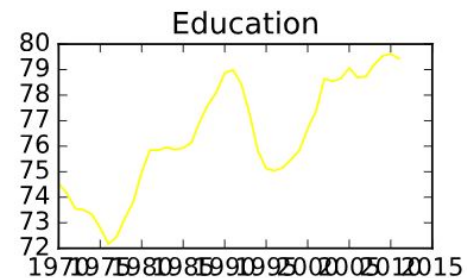
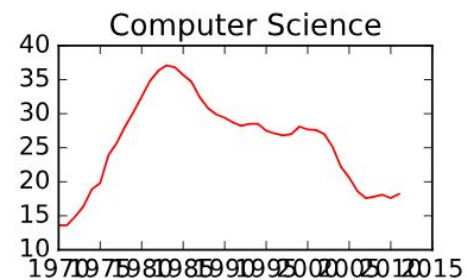
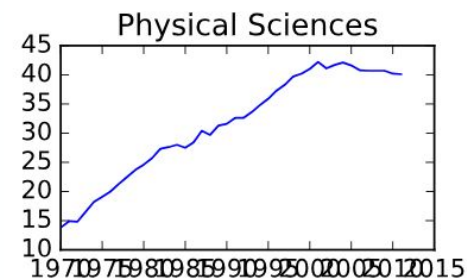
```
1 # Create a figure with 1x2 subplot and make the left subplot active
2 plt.subplot(1,2,1)
3
4 # Plot in blue the % of degrees awarded to women in the Physical Sciences
5 plt.plot(year, physical_sciences, color='blue')
6 plt.title('Physical Sciences')
7
8 # Make the right subplot active in the current 1x2 subplot grid
9 plt.subplot(1,2,2)
10
11 # Plot in red the % of degrees awarded to women in Computer Science
12 plt.plot(year, computer_science, color='red')
13 plt.title('Computer Science')
14
15 # Use plt.tight_layout() to improve the spacing between subplots
16 plt.tight_layout()
17 plt.show()
18
```



Using subplot() (2)

- Syntax: subplot(nrows, ncols, subplot)
- Subplot ordering:
 - Row-wise from top left
 - Indexed from 1

```
1 # Create a figure with 2x2 subplot layout and make the top left subplot active
2 plt.subplot(2, 2, 1)
3 # Plot in blue the % of degrees awarded to women in the Physical Sciences
4 plt.plot(year, physical_sciences, color='blue')
5 plt.title('Physical Sciences')
6 # Make the top right subplot active in the current 2x2 subplot grid
7 plt.subplot(2, 2, 2)
8 # Plot in red the % of degrees awarded to women in Computer Science
9 plt.plot(year, computer_science, color='red')
10 plt.title('Computer Science')
11 # Make the bottom left subplot active in the current 2x2 subplot grid
12 plt.subplot(2, 2, 3)
13 # Plot in green the % of degrees awarded to women in Health Professions
14 plt.plot(year, health, color='green')
15 plt.title('Health Professions')
16 # Make the bottom right subplot active in the current 2x2 subplot grid
17 plt.subplot(2, 2, 4)
18 # Plot in yellow the % of degrees awarded to women in Education
19 plt.plot(year, education, color='yellow')
20 plt.title('Education')
21 # Improve the spacing between subplots and display them
22 plt.tight_layout()
23 plt.show()
24
```



Customizing axes

Controlling axis extents

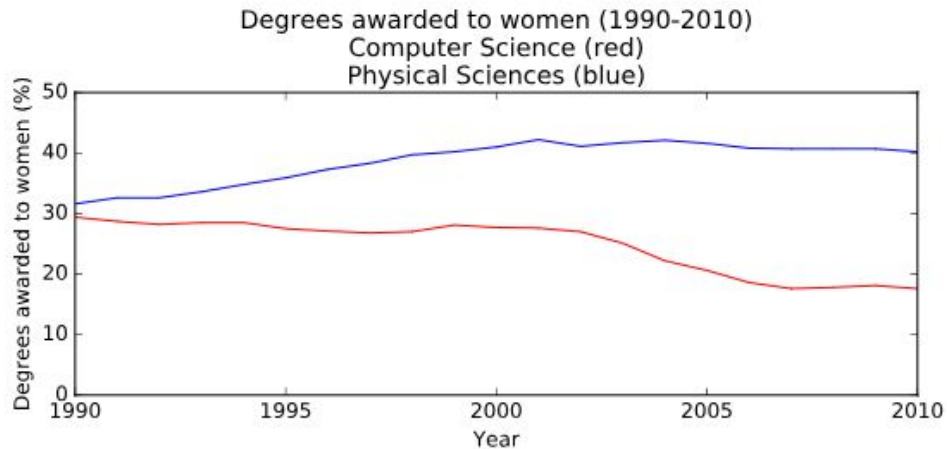
- `axis([xmin, xmax, ymin, ymax])` sets axis extents
- Control over individual axis extents
 - `xlim([xmin, xmax])`
 - `ylim([ymin, ymax])`
- Can use tuples, lists for extents
 - e.g., `xlim((-2, 3))` works
 - e.g., `xlim([-2, 3])` works also

Other axis() options

Invocation	Result
<code>axis('off')</code>	turns off axis lines, labels
<code>axis('equal')</code>	equal scaling on x, y axes
<code>axis('square')</code>	forces square plot
<code>axis('tight')</code>	sets <code>xlim()</code> , <code>ylim()</code> to show all data

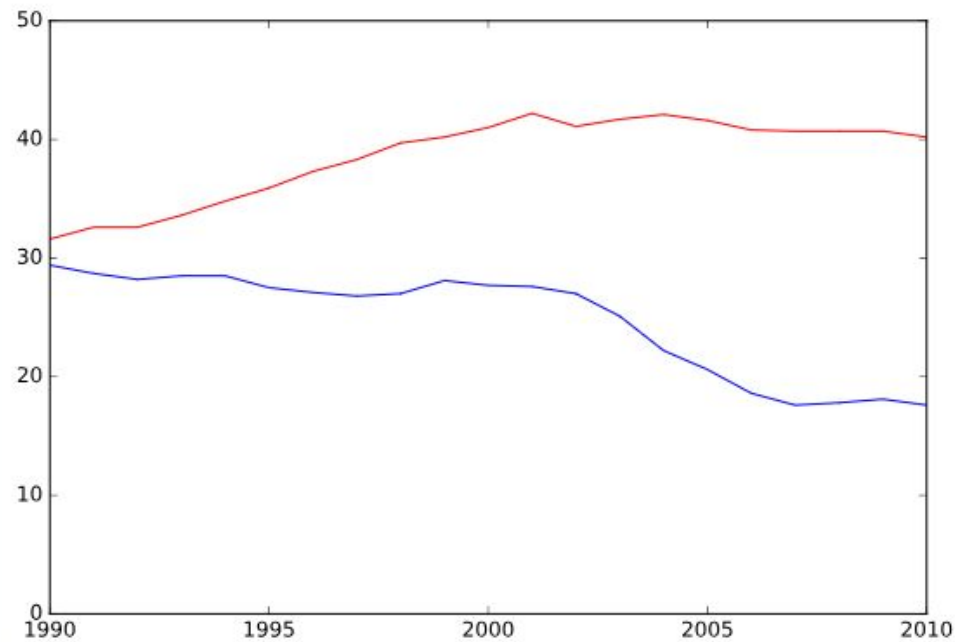
Using xlim(), ylim()

```
1 # Plot the % of degrees awarded to women in Computer
  Science and the Physical Sciences
2 plt.plot(year, computer_science, color='red')
3 plt.plot(year, physical_sciences, color='blue')
4
5 # Add the axis labels
6 plt.xlabel('Year')
7 plt.ylabel('Degrees awarded to women (%)')
8
9 # Set the x-axis range
10 plt.xlim(1990, 2010)
11
12 # Set the y-axis range
13 plt.ylim(0, 50)
14
15 # Add a title and display the plot
16 plt.title('Degrees awarded to women (1990-2010)
17           \nComputer Science (red)\nPhysical Sciences (blue)')
18 plt.show()
19
20 # Save the image as 'xlim_and_ylim.png'
21 plt.savefig('xlim_and_ylim.png')
```



Use plt.axis()

```
1 # Plot in blue the % of degrees awarded to women in
  Computer Science
2 plt.plot(year, computer_science, color='blue')
3
4 # Plot in red the % of degrees awarded to women in the
  Physical Sciences
5 plt.plot(year, physical_sciences, color='red')
6
7 # Set the x-axis and y-axis limits
8 plt.axis((1990, 2010, 0, 50))
9
10 # Show the figure
11 plt.show()
12
13 # Save the figure as 'axis_limits.png'
14 plt.savefig('axis_limits.png')
15
```



Legend

Legend locations

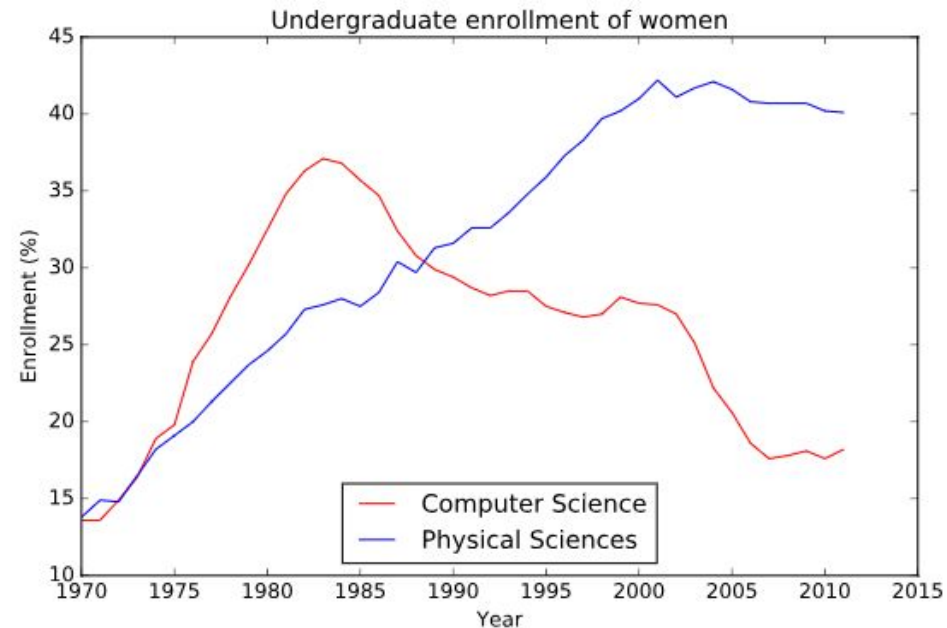
string	code	string	code	string	code
'upper left'	2	'upper center'	9	'upper right'	1
'center left'	6	'center'	10	'center right'	7
'lower left'	3	'lower center'	8	'lower right'	4
'best'	0			'right'	5

Options for annotate()

option	description
s	text of label
xy	coordinates to annotate
xytext	coordinates of label
arrowprops	controls drawing of arrow

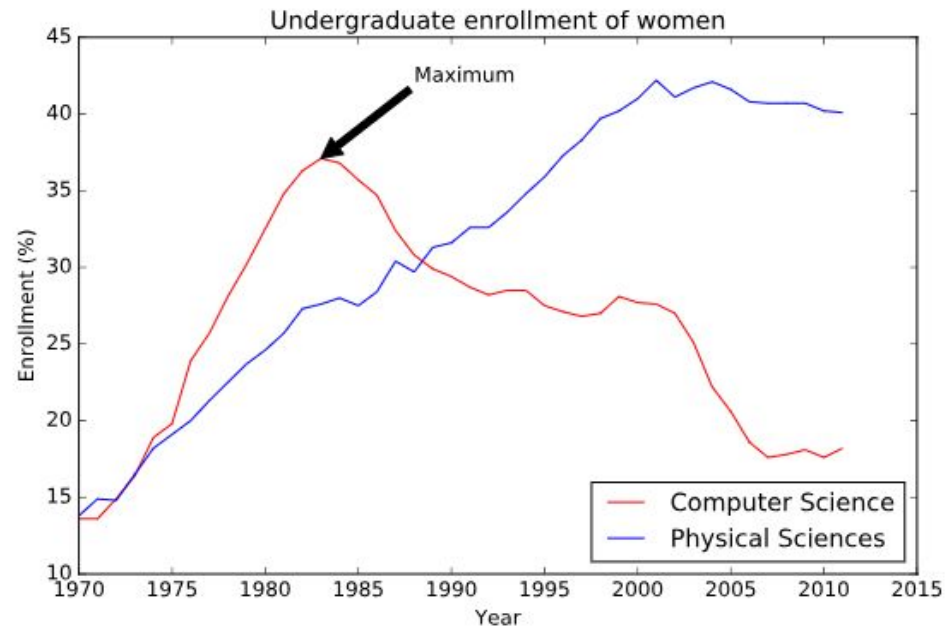
Legend (Cont. 2)

```
1 # Specify the label 'Computer Science'
2 plt.plot(year, computer_science, color='red', label
3         ='Computer Science')
4 # Specify the label 'Physical Sciences'
5 plt.plot(year, physical_sciences, color='blue', label
6         ='Physical Sciences')
7 # Add a legend at the lower center
8 plt.legend()
9
10 # Add axis labels and title
11 plt.xlabel('Year')
12 plt.ylabel('Enrollment (%)')
13 plt.title('Undergraduate enrollment of women')
14 plt.legend(loc = 'lower center')
15 plt.show()
```



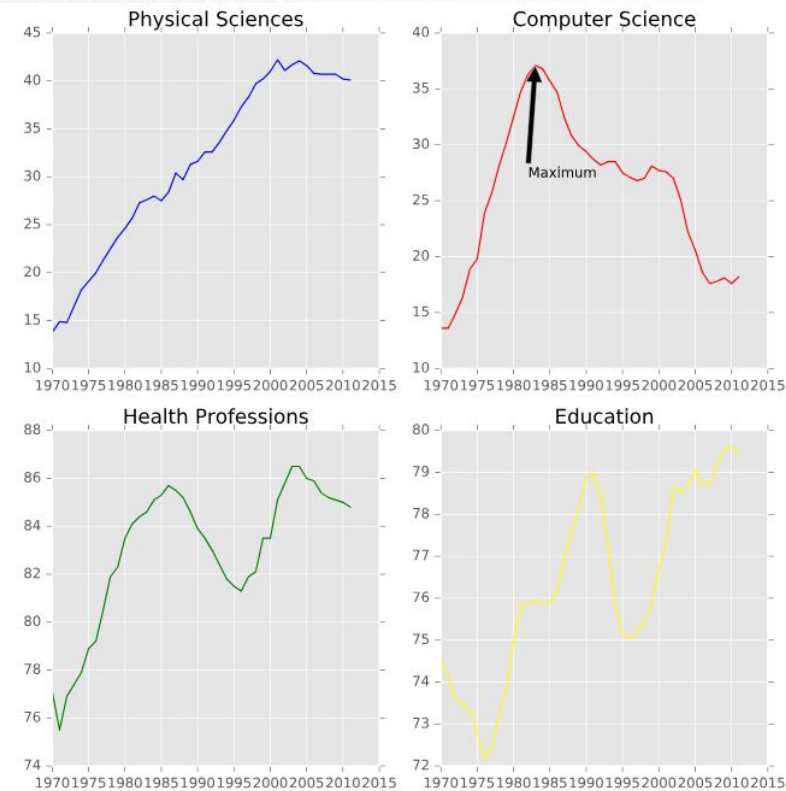
Using annotate()

```
1 # Plot with legend as before
2 plt.plot(year, computer_science, color='red', label
3          ='Computer Science')
4 plt.plot(year, physical_sciences, color='blue', label
5          ='Physical Sciences')
6 plt.legend(loc='lower right')
7
8 # Compute the maximum enrollment of women in Computer
9 # Science: cs_max
10 cs_max = computer_science.max()
11
12 # Calculate the year in which there was maximum
13 # enrollment of women in Computer Science: yr_max
14 yr_max = year[computer_science.argmax()]
15
16 # Add a black arrow annotation
17 plt.annotate('Maximum', xy=(yr_max, cs_max), xytext
18              =(yr_max+5, cs_max+5), arrowprops=dict(facecolor
19              ='black'))
20
21 # Add axis labels and title
22 plt.xlabel('Year')
23 plt.ylabel('Enrollment (%)')
24 plt.title('Undergraduate enrollment of women')
25 plt.show()
```



Modifying styles

```
1 # Import matplotlib.pyplot
2 import matplotlib.pyplot as plt
3 # Set the style to 'ggplot'
4 plt.style.use('ggplot')
5 # Create a figure with 2x2 subplot layout
6 plt.subplot(2, 2, 1)
7 # Plot the enrollment % of women in the Physical Sciences
8 plt.plot(year, physical_sciences, color='blue')
9 plt.title('Physical Sciences')
10 # Plot the enrollment % of women in Computer Science
11 plt.subplot(2, 2, 2)
12 plt.plot(year, computer_science, color='red')
13 plt.title('Computer Science')
14 # Add annotation
15 cs_max = computer_science.max()
16 yr_max = year[computer_science.argmax()]
17 plt.annotate('Maximum', xy=(yr_max, cs_max), xytext=(yr_max-1, cs_max-10), arrowprops=dict(
18     facecolor='black'))
19 # Plot the enrollment % of women in Health professions
20 plt.subplot(2, 2, 3)
21 plt.plot(year, health, color='green')
22 plt.title('Health Professions')
23 # Plot the enrollment % of women in Education
24 plt.subplot(2, 2, 4)
25 plt.plot(year, education, color='yellow')
26 plt.title('Education')
27 # Improve spacing between subplots and display them
28 plt.tight_layout()
29 plt.show()
```



- Style sheets in Matplotlib
- Defaults for lines, points, backgrounds, etc.
- Switch styles globally with `plt.style.use()`
- `plt.style.available`: list of styles

Statistical Plotting

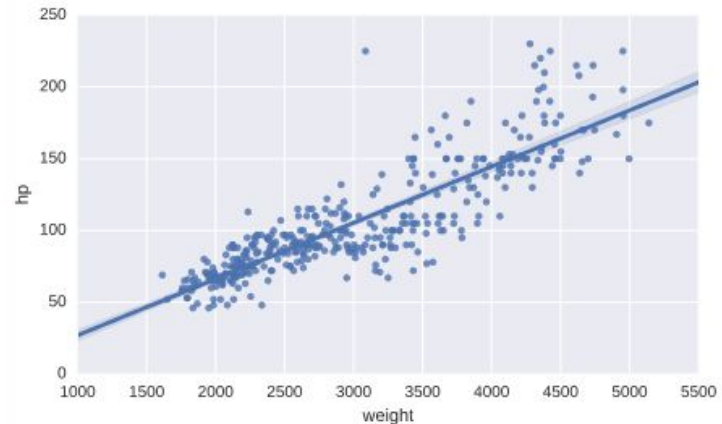


Seaborn

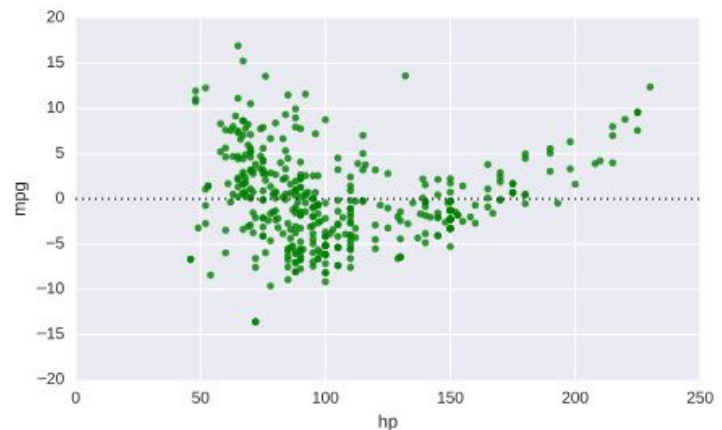
- Built on top of Matplotlib
- Purpose: simplify visualizing statistical data
- Works with Pandas – data frame package

Simple linear regressions and residuals

```
1 # Import plotting modules
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4
5 # Plot a linear regression between 'weight'
  and 'hp'
6 sns.lmplot(x='weight', y='hp', data =
  auto)
7
8 # Display the plot
9 plt.show()
10
```

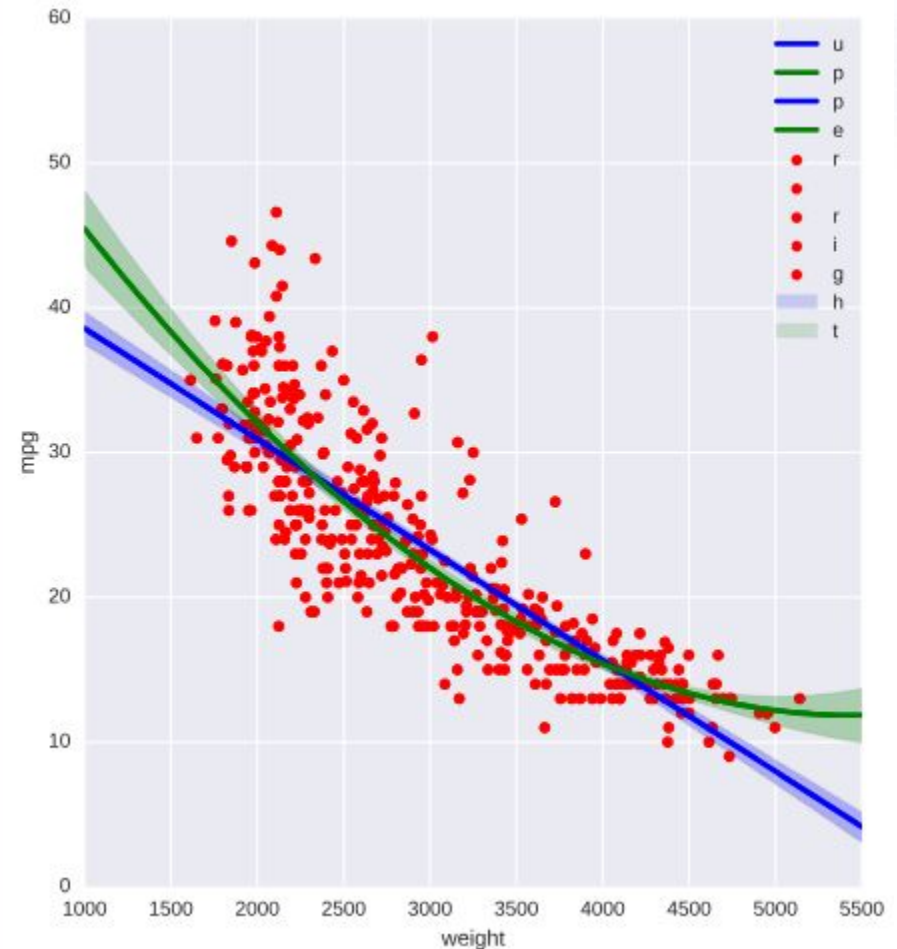


```
1 # Import plotting modules
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4
5 # Generate a green residual plot of the
  regression between 'hp' and 'mpg'
6 sns.residplot(x='hp', y='mpg', data =
  auto, color='green')
7
8 # Display the plot
9 plt.show()
10
```



Higher-order regressions

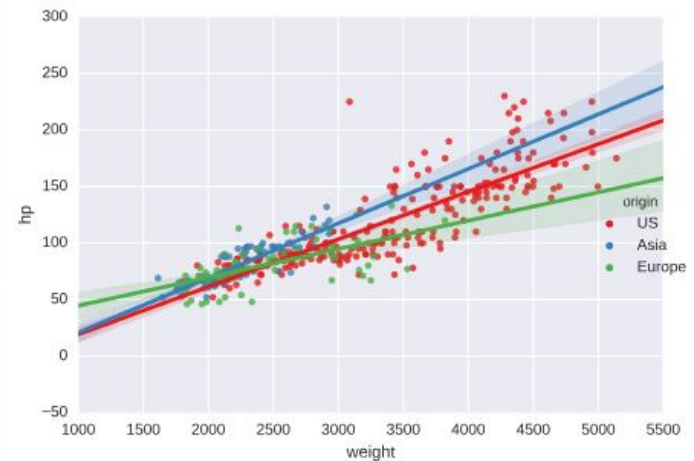
```
1 # Generate a scatter plot of 'weight' and
  # 'mpg' using red circles
2 plt.scatter(auto['weight'], auto['mpg'],
  label='data', color='red', marker='o')
3
4 # Plot in blue a linear regression of order
  1 between 'weight' and 'mpg'
5 sns.regplot(x='weight', y='mpg', data=auto,
  scatter=None, color='blue', label='order 1'
  )
6
7 # Plot in green a linear regression of
  order 2 between 'weight' and 'mpg'
8 sns.regplot(x='weight', y='mpg', data=auto,
  scatter=None, order=2, color='green', label
  ='order 2')
9
10 # Add a legend and display the plot
11 plt.legend(loc='upper right')
12 plt.show()
```



Grouping linear regressions

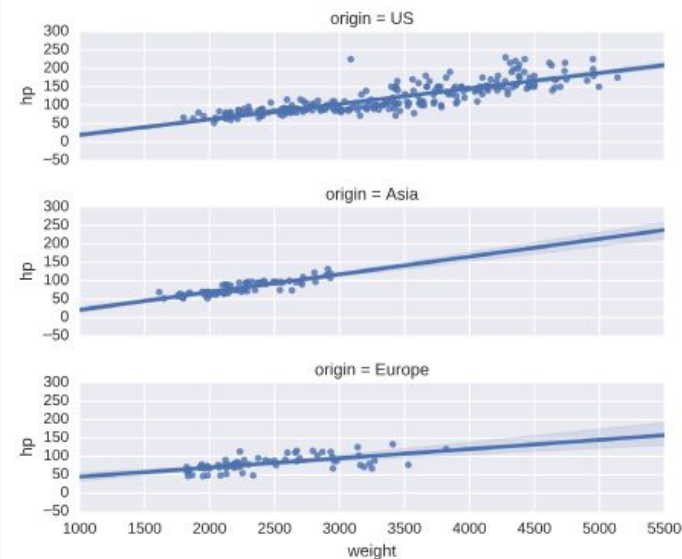
by hue

```
1 # Plot a linear regression between 'weight'
  and 'hp', with a hue of 'origin' and
  palette of 'Set1'
2 sns.lmplot(x='weight', y='hp', data=auto,
  hue='origin', palette='Set1')
3
4 # Display the plot
5 plt.show()
6
```



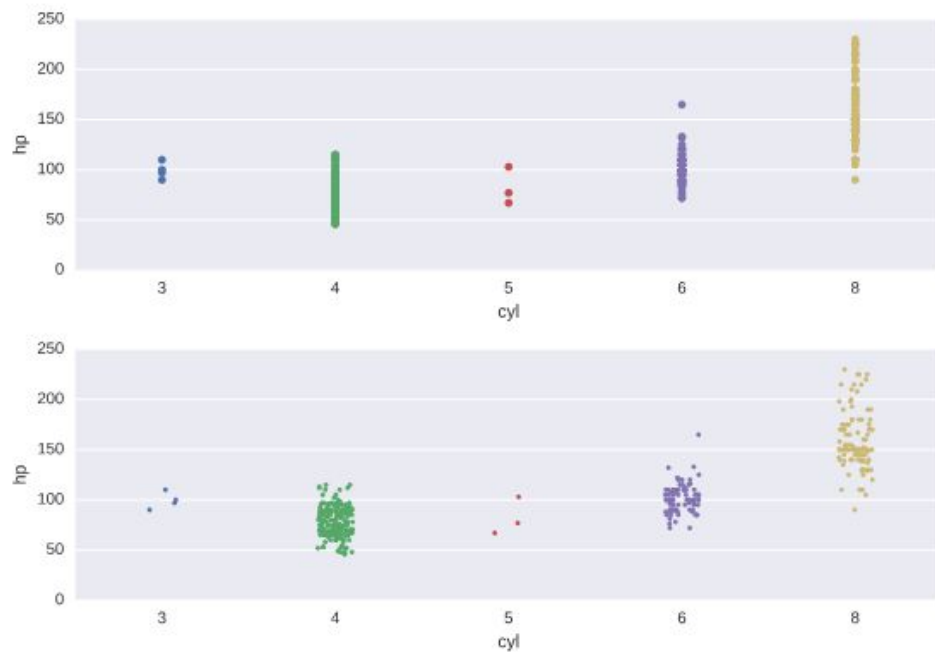
by row
/
column

```
1 # Plot linear regressions between 'weight'
  and 'hp' grouped row-wise by 'origin'
2 sns.lmplot(x='weight', y='hp', data=auto,
  row='origin', palette='Set1')
3
4 # Display the plot
5 plt.show()
6
```



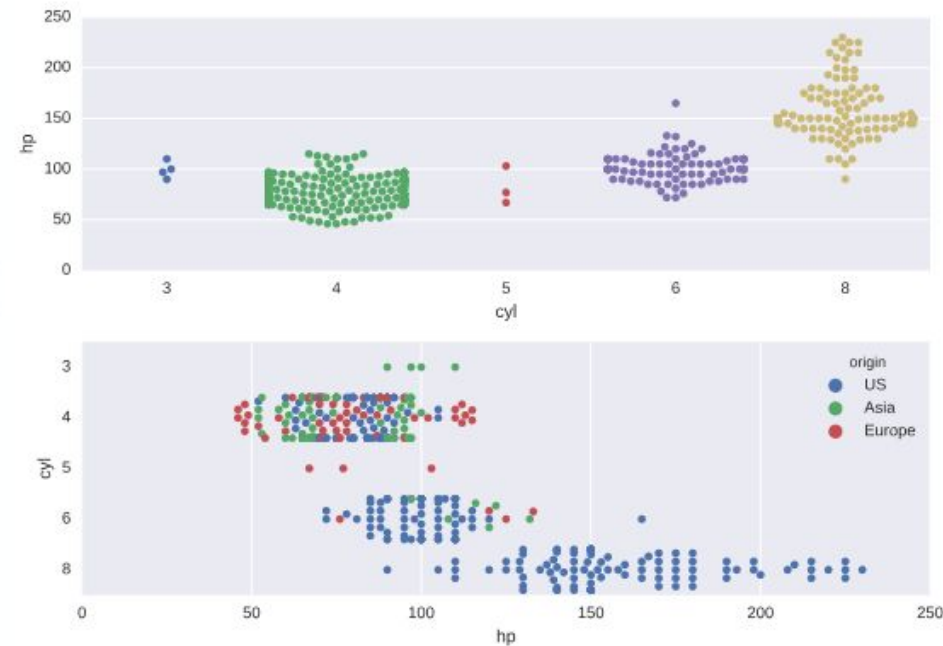
Constructing strip plots

```
1 # Make a strip plot of 'hp' grouped by 'cyl'
2 plt.subplot(2,1,1)
3 sns.stripplot(x='cyl', y='hp', data=auto)
4
5 # Make the strip plot again using jitter and a smaller
  point size
6 plt.subplot(2,1,2)
7 sns.stripplot(x='cyl', y='hp', data=auto, jitter = True,
  size = 3)
8
9 # Display the plot
10 plt.show()
11
```



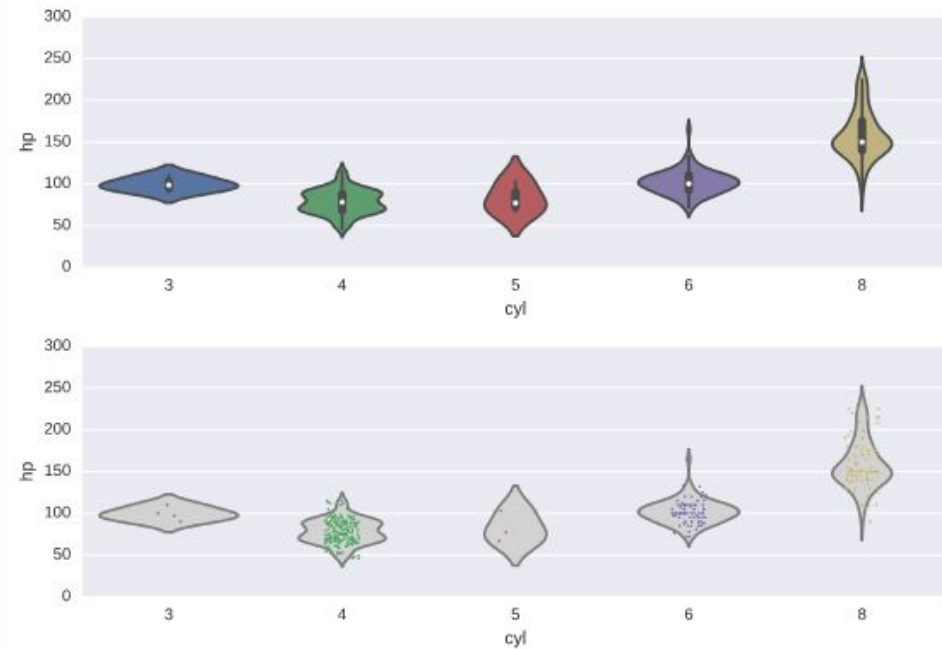
Constructing swarm plots

```
1 # Generate a swarm plot of 'hp' grouped horizontally by  
  'cyl'  
2 plt.subplot(2,1,1)  
3 sns.swarmplot(x = 'cyl', y = 'hp', data = auto )  
4  
5 # Generate a swarm plot of 'hp' grouped vertically by 'cyl'  
  with a hue of 'origin'  
6 plt.subplot(2,1,2)  
7 sns.swarmplot(x = 'hp', y = 'cyl', data = auto, orient =  
  'h', hue = 'origin')  
8  
9 # Display the plot  
10 plt.show()  
11
```



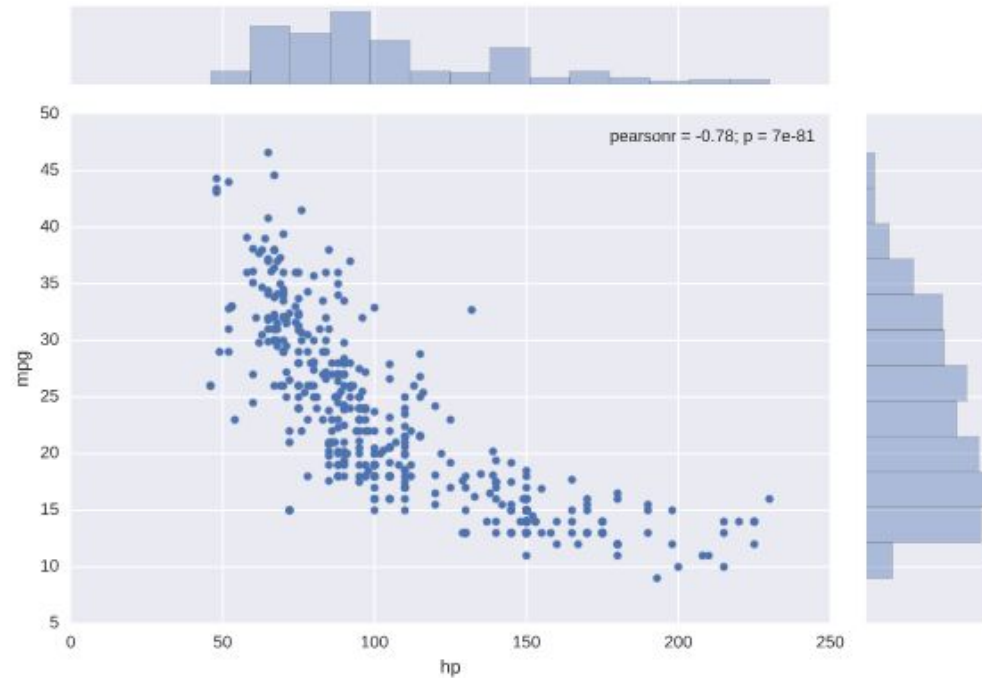
Constructing violin plots

```
1 # Generate a violin plot of 'hp' grouped horizontally by
  'cyl'
2 plt.subplot(2,1,1)
3 sns.violinplot(x='cyl', y='hp', data = auto)
4
5 # Generate the same violin plot again with a color of
  'lightgray' and without inner annotations
6 plt.subplot(2,1,2)
7 sns.violinplot(x='cyl', y='hp', data = auto, inner = None,
  color = 'lightgray')
8
9 # Overlay a strip plot on the violin plot
10 plt.subplot(2,1,2)
11 sns.stripplot(x='cyl', y='hp', data=auto, jitter = True,
  size = 1.5)
12
13 # Display the plot
14 plt.show()
```



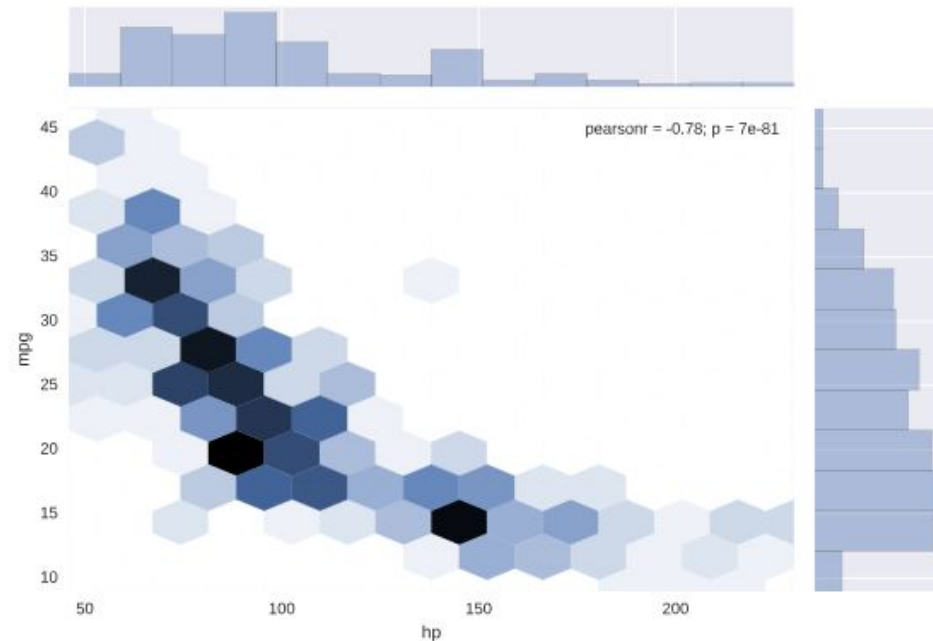
Plotting joint distributions (1)

```
1 # Generate a joint plot of 'hp' and 'mpg'
2 sns.jointplot(x = 'hp', y = 'mpg', data = auto)
3
4 # Display the plot
5 plt.show()
6
```



Plotting joint distributions (2)

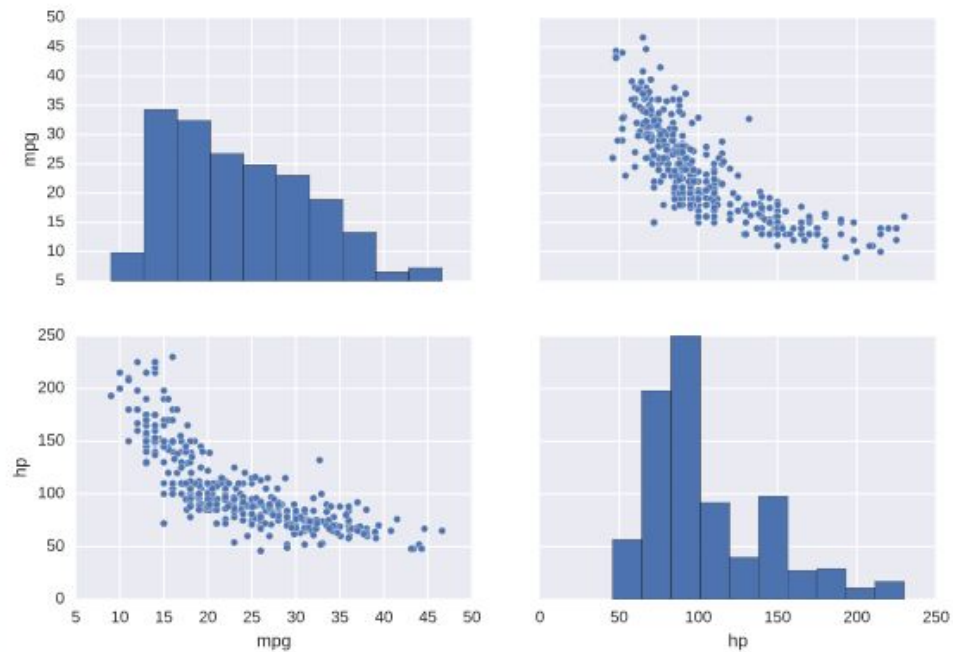
```
1 # Generate a joint plot of 'hp' and 'mpg' using a hexbin  
  plot  
2 sns.jointplot(x = 'hp', y = 'mpg', data = auto, kind='hex')  
3  
4 # Display the plot  
5 plt.show()  
6
```



- `kind='scatter'` uses a scatter plot of the data points
- `kind='reg'` uses a regression plot (default order 1)
- `kind='resid'` uses a residual plot
- `kind='kde'` uses a *kernel density estimate* of the joint distribution
- `kind='hex'` uses a hexbin plot of the joint distribution

Plotting distributions pairwise (1)

```
1 # Print the first 5 rows of the DataFrame
2 print(auto.head())
3
4 # Plot the pairwise joint distributions from the DataFrame
5 sns.pairplot(auto)
6
7 # Display the plot
8 plt.show()
9
```



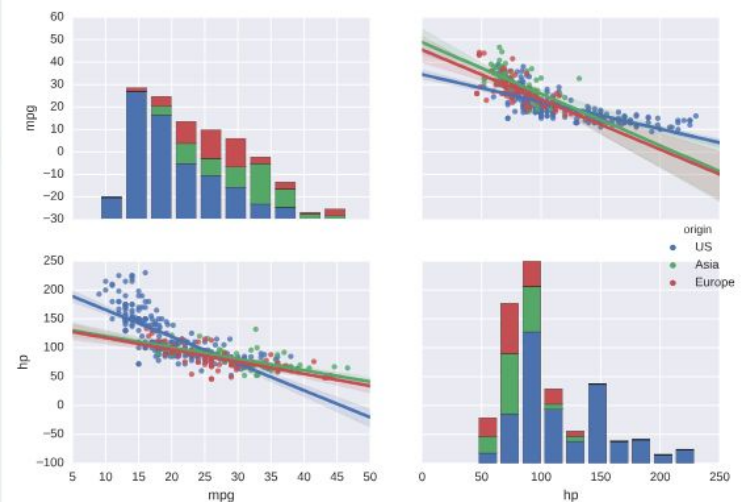
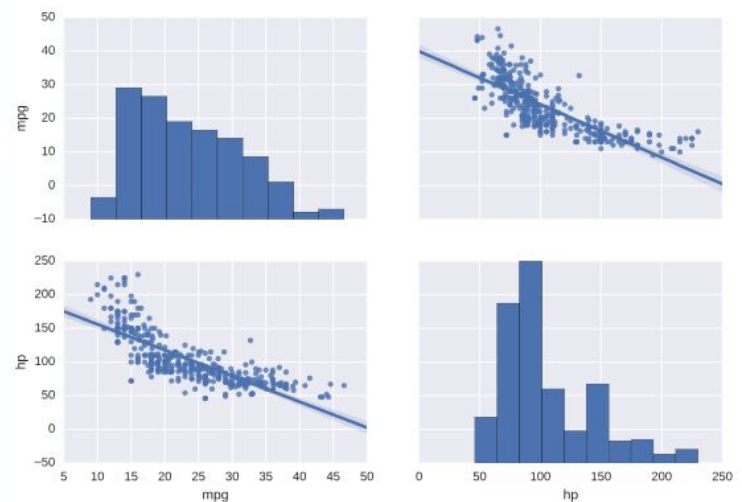
Plotting distributions pairwise (2)

```
1 # Print the first 5 rows of the DataFrame
2 print(auto.head())
3
4 # Plot the pairwise joint distributions grouped by 'origin'
  along with regression lines
5 sns.pairplot(auto, kind='reg')
6
7 # Display the plot
8 plt.show()
9
```

<script.py> output:

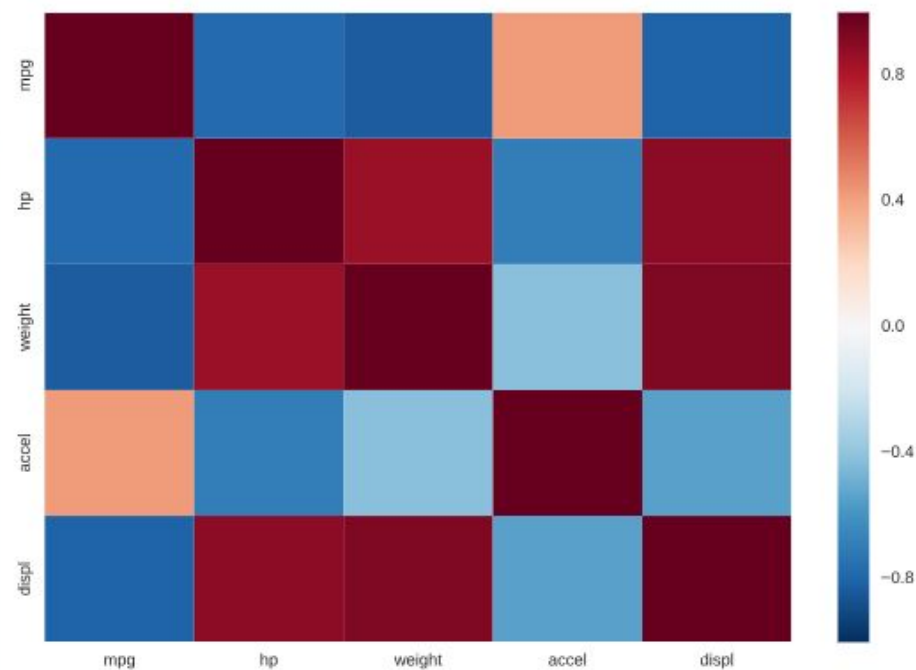
	mpg	hp	origin
0	18.0	88	US
1	9.0	193	US
2	36.1	60	Asia
3	18.5	98	US
4	34.3	78	Europe

```
1 # Print the first 5 rows of the DataFrame
2 print(auto.head())
3
4 # Plot the pairwise joint distributions grouped by 'origin'
  along with regression lines
5 sns.pairplot(auto, kind='reg', hue = 'origin')
6
7 # Display the plot
8 plt.show()
9
```



Visualizing correlations with a heatmap

```
1 # Print the covariance matrix
2 print(cov_matrix)
3
4 # Visualize the covariance matrix using a heatmap
5 sns.heatmap(cov_matrix)
6
7 # Display the heatmap
8 plt.show()
9
```



	mpg	hp	weight	accel	displ
mpg	1.000000	-0.778427	-0.832244	0.423329	-0.805127
hp	-0.778427	1.000000	0.864538	-0.689196	0.897257
weight	-0.832244	0.864538	1.000000	-0.416839	0.932994
accel	0.423329	-0.689196	-0.416839	1.000000	-0.543800
displ	-0.805127	0.897257	0.932994	-0.543800	1.000000

Plotting 2D arrays



Reminder: NumPy arrays

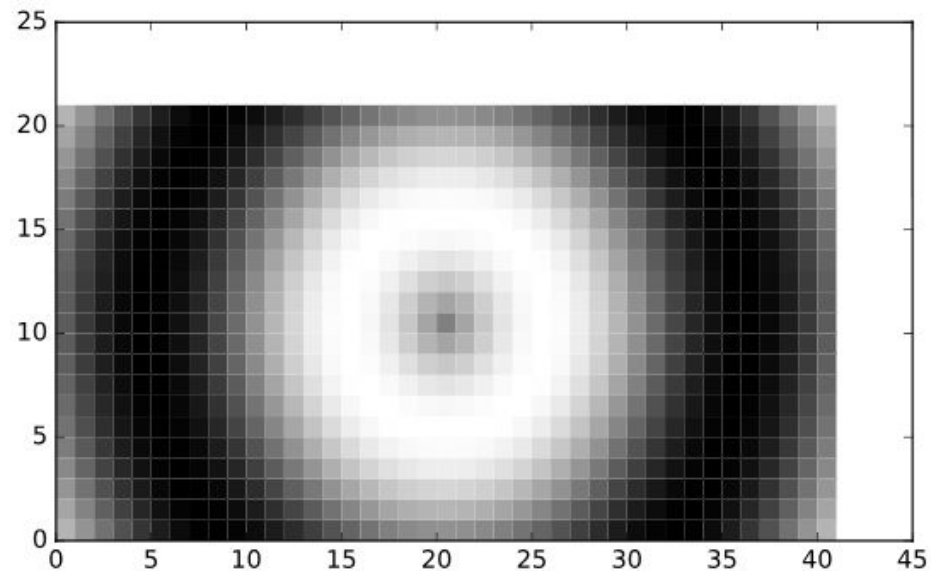
- Homogeneous in type
- Calculations all at once
- Indexing with brackets:
 - $A[\text{index}]$ for 1D array
 - $A[\text{index0}, \text{index1}]$ for 2D array

Reminder: slicing arrays

- Slicing: 1D arrays: $A[\text{slice}]$, 2D arrays: $A[\text{slice0}, \text{slice1}]$
- Slicing: $\text{slice} = \text{start}:\text{stop}:\text{stride}$
 - Indexes from *start* to *stop-1* in steps of *stride*
 - Missing *start*: implicitly at *beginning* of array
 - Missing *stop*: implicitly at *end* of array
 - Missing *stride*: implicitly *stride 1*
- Negative indexes/slices: count from *end of array*

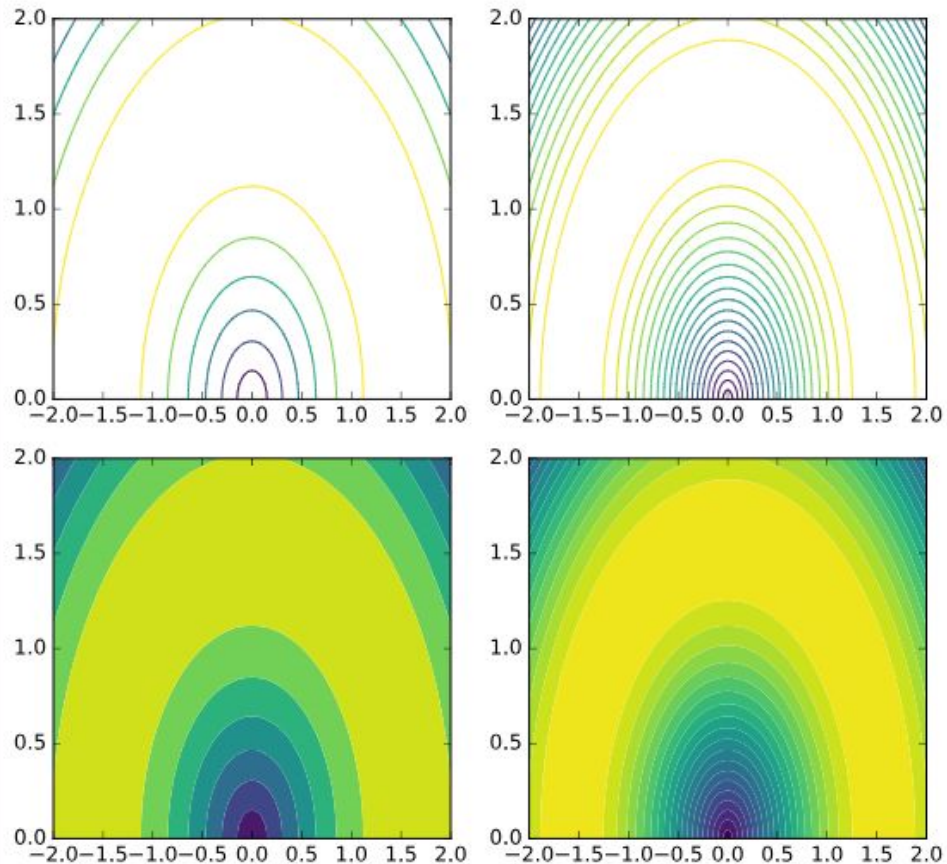
Generating meshes

```
1 # Import numpy and matplotlib.pyplot
2 import numpy as np
3 import matplotlib.pyplot as plt
4 # Generate two 1-D arrays: u, v
5 u = np.linspace(-2, + 2, num = 41)
6 v = np.linspace(-1, + 1, num = 21)
7 # Generate 2-D arrays from u and v: X, Y
8 X,Y = np.meshgrid(u, v)
9 # Compute Z based on X and Y
10 Z = np.sin(3*np.sqrt(X**2 + Y**2))
11 # Display the resulting image with pcolor()
12 print ('Z:\n', Z)
13 plt.set_cmap('gray')
14 plt.pcolor(Z)
15 plt.show()
16 # Save the figure to 'sine_mesh.png'
17 plt.savefig('sine_mesh.png')
18
```



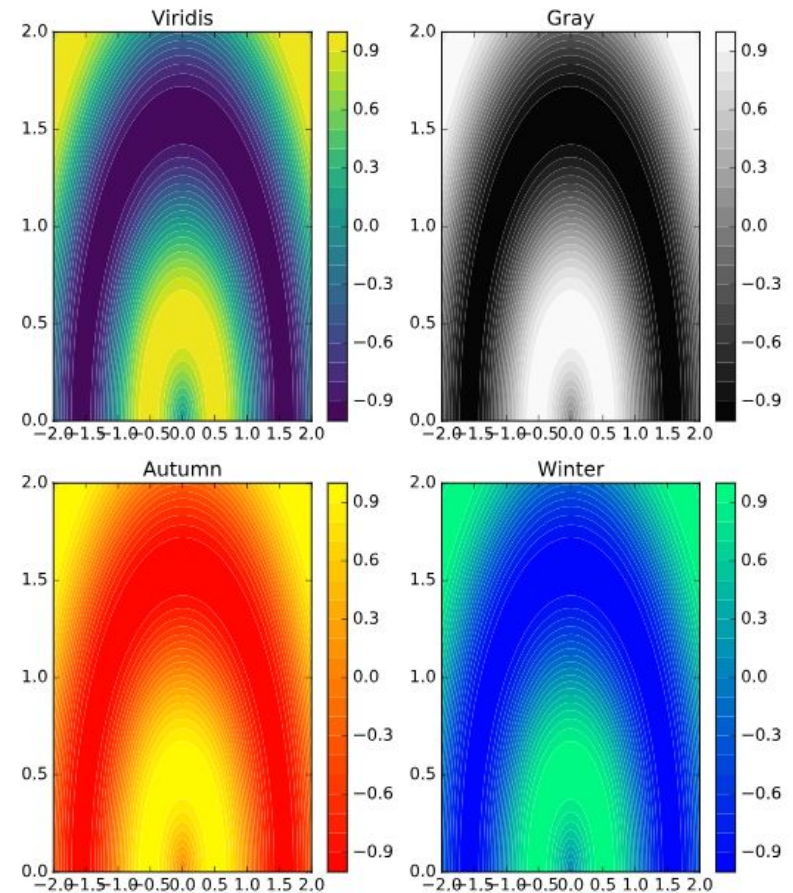
Contour & filled contour plots

```
1 # Generate a default contour map of the array Z
2 plt.subplot(2,2,1)
3 plt.contour(X, Y, Z)
4
5 # Generate a contour map with 20 contours
6 plt.subplot(2,2,2)
7 plt.contour(X, Y, Z, 20)
8
9 # Generate a default filled contour map of the array Z
10 plt.subplot(2,2,3)
11 plt.contourf(X, Y, Z)
12
13 # Generate a default filled contour map with 20
   contours
14 plt.subplot(2,2,4)
15 plt.contourf(X, Y, Z, 20)
16
17 # Improve the spacing between subplots
18 plt.tight_layout()
19
20 # Display the figure
21 plt.show()
22
23
```



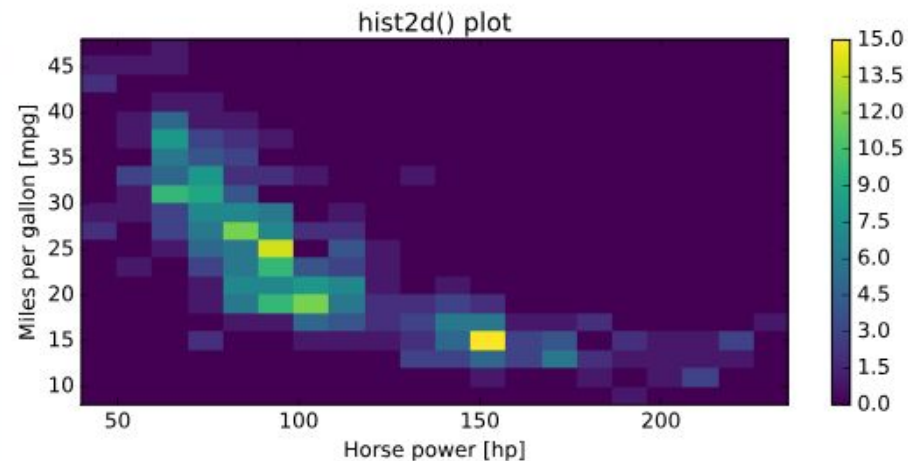
Modifying colormaps

```
1 # Create a filled contour plot with a color map of 'viridis'
2 plt.subplot(2,2,1)
3 plt.contourf(X,Y,Z,20, cmap='viridis')
4 plt.colorbar()
5 plt.title('Viridis')
6
7 # Create a filled contour plot with a color map of 'gray'
8 plt.subplot(2,2,2)
9 plt.contourf(X,Y,Z,20, cmap='gray')
10 plt.colorbar()
11 plt.title('Gray')
12
13 # Create a filled contour plot with a color map of 'autumn'
14 plt.subplot(2,2,3)
15 plt.contourf(X,Y,Z,20, cmap='autumn')
16 plt.colorbar()
17 plt.title('Autumn')
18
19 # Create a filled contour plot with a color map of 'winter'
20 plt.subplot(2,2,4)
21 plt.contourf(X,Y,Z,20, cmap='winter')
22 plt.colorbar()
23 plt.title('Winter')
24
25 # Improve the spacing between subplots and display them
26 plt.tight_layout()
27 plt.show()
28
```

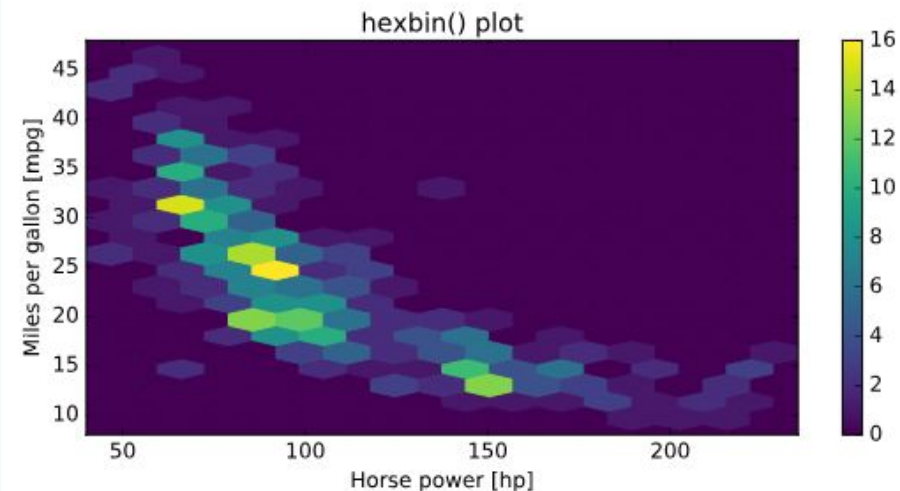


Using hist2d() and hexbin()

```
1 # Generate a 2-D histogram
2 plt.hist2d(hp, mpg, bins = (20, 20), range = ((40, 235
3 ), (8, 48)))
4 # Add a color bar to the histogram
5 plt.colorbar()
6
7 # Add labels, title, and display the plot
8 plt.xlabel('Horse power [hp]')
9 plt.ylabel('Miles per gallon [mpg]')
10 plt.title('hist2d() plot')
11 plt.show()
12
```



```
1 # Generate a 2d histogram with hexagonal bins
2 plt.hexbin(hp, mpg, gridsize=(15,12),
3            extent=(40,235,8, 48))
4
5 # Add a color bar to the histogram
6 plt.colorbar()
7
8 # Add labels, title, and display the plot
9 plt.xlabel('Horse power [hp]')
10 plt.ylabel('Miles per gallon [mpg]')
11 plt.title('hexbin() plot')
12 plt.show()
13
```



Seaborn color palettes

- https://seaborn.pydata.org/tutorial/color_palettes.html

➤ Qualitative (categorical) palettes



➤ Sequential palettes



➤ Diverging palettes



- <http://colorbrewer2.org/#type=sequential&scheme=BuGn&n=3>

➤ Provide guidance for color blind safe design

- XKCD colors



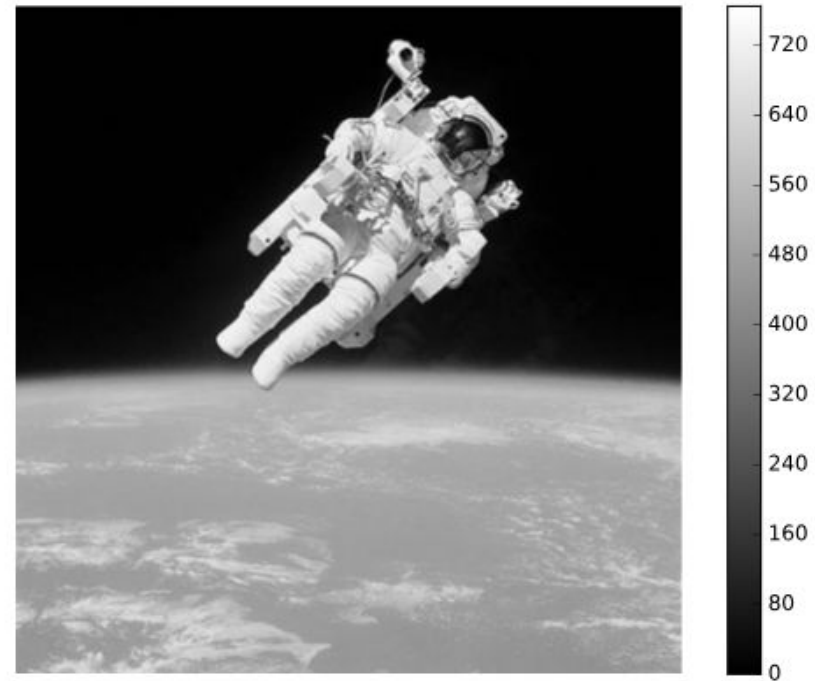
Loading, examining images

```
1 # Load the image into an array: img
2 img = plt.imread('480px-Astronaut-EVA.jpg')
3
4 # Print the shape of the image
5 print(img.shape)
6 (480, 480, 3)
7
8 # Display the image
9 plt.imshow(img)
10
11 # Hide the axes
12 plt.axis('off')
13 plt.show()
14
```



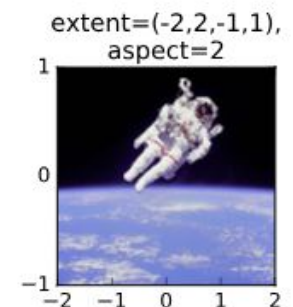
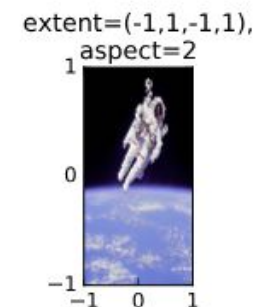
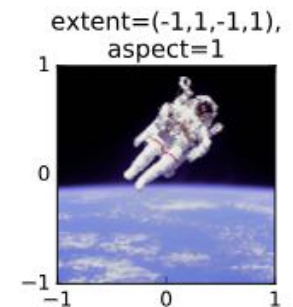
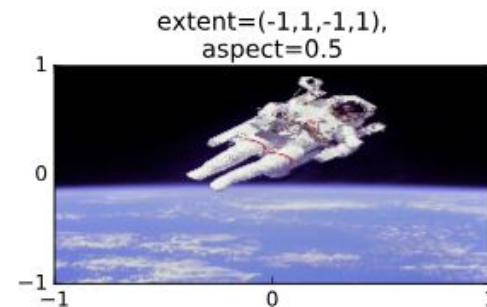
Pseudocolor plot from image data

```
1 # Load the image into an array: img
2 img = plt.imread('480px-Astronaut-EVA.jpg')
3
4 # Print the shape of the image
5 print(img.shape)
6
7 # Compute the sum of the red, green and blue channels:
  intensity
8 intensity = img.sum(axis = 2)
9
10 # Print the shape of the intensity
11 print(intensity)
12 (480, 480, 3)
13
14 # Display the intensity with a colormap of 'gray'
15 plt.imshow(intensity, cmap = 'gray')
16
17
18 # Add a colorbar
19 plt.colorbar()
20
21 # Hide the axes and show the figure
22 plt.axis('off')
23 plt.show()
```



Extent and aspect

```
1 # Load the image into an array: img
2 img = plt.imread('480px-Astronaut-EVA.jpg')
3 # Specify the extent and aspect ratio of the top left
  subplot
4 plt.subplot(2,2,1)
5 plt.title('extent=(-1,1,-1,1), \naspect=0.5')
6 plt.xticks([-1,0,1])
7 plt.yticks([-1,0,1])
8 plt.imshow(img, extent = (-1,1,-1,1), aspect = 0.5)
9 # Specify the extent and aspect ratio of the top right
  subplot
10 plt.subplot(2,2,2)
11 plt.title('extent=(-1,1,-1,1), \naspect=1')
12 plt.xticks([-1,0,1])
13 plt.yticks([-1,0,1])
14 plt.imshow(img, extent = (-1,1,-1,1), aspect = 1)
15 # Specify the extent and aspect ratio of the bottom left
  subplot
16 plt.subplot(2,2,3)
17 plt.title('extent=(-1,1,-1,1), \naspect=2')
18 plt.xticks([-1,0,1])
19 plt.yticks([-1,0,1])
20 plt.imshow(img, extent = (-1,1,-1,1), aspect = 2)
21 # Specify the extent and aspect ratio of the bottom right
  subplot
22 plt.subplot(2,2,4)
23 plt.title('extent=(-2,2,-1,1), \naspect=2')
24 plt.xticks([-2,-1,0,1,2])
25 plt.yticks([-1,0,1])
26 plt.imshow(img, extent = (-2,2,-1,1), aspect = 2)
27 # Improve spacing and display the figure
28 plt.tight_layout()
29 plt.show()
```



Rescaling pixel intensities

```
1 # Load the image into an array: image
2 image = plt.imread('640px-Unequalized_Hawkes_Bay_NZ.jpg')
3
4 # Extract minimum and maximum values from the image: pmin,
  pmax
5 pmin, pmax = image.min(), image.max()
6 print("The smallest & largest pixel intensities are %d & %d
  ." % (pmin, pmax))
7
8 # Rescale the pixels: rescaled_image
9 rescaled_image = 256*(image - pmin) / (pmax - pmin)
10 print("The rescaled smallest & largest pixel intensities
  are %.1f & %.1f." %
11       (rescaled_image.min(), rescaled_image.max()))
12
13 # Display the original image in the top subplot
14 plt.subplot(2,1,1)
15 plt.title('original image')
16 plt.axis('off')
17 plt.imshow(image)
18
19 # Display the rescaled image in the bottom subplot
20 plt.subplot(2,1,2)
21 plt.title('rescaled image')
22 plt.axis('off')
23 plt.imshow(rescaled_image)
24
25 plt.show()
26
```

original image



rescaled image



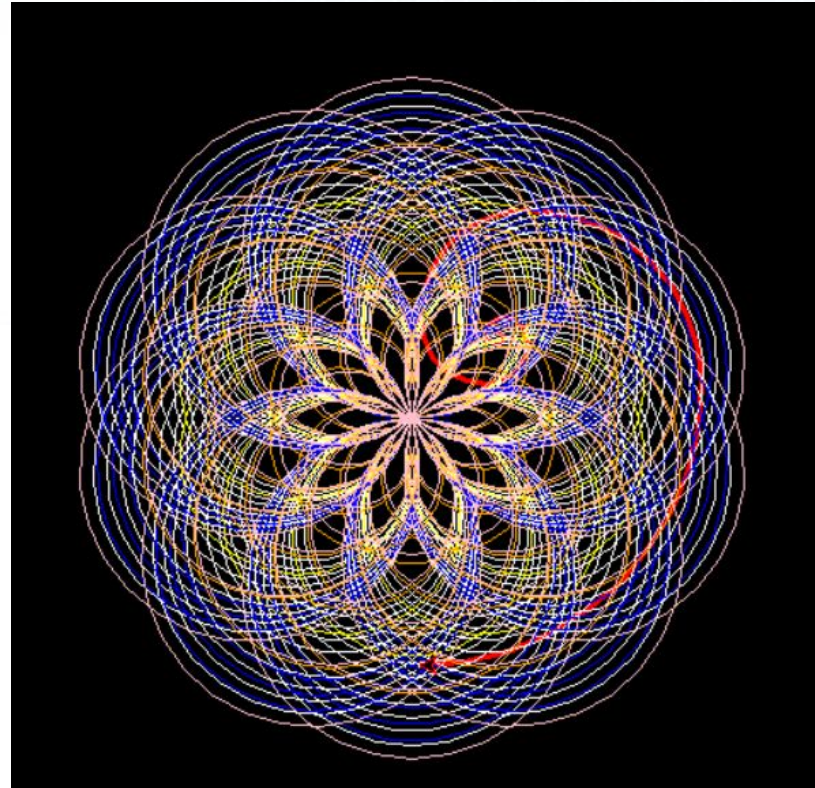
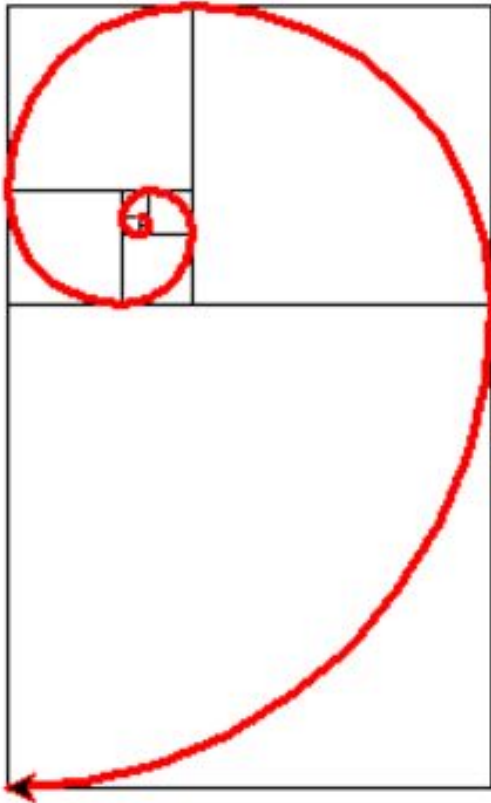
Turtle

<https://docs.python.org/2/library/turtle.html>

<http://www.cs.middlebury.edu/~mlinderman/courses/middsip/sm17/lectures/lecture4-notes.html>

- Turtle graphics is a popular way for introducing programming to kids
- It was part of the original Logo programming language developed by Wally Feurzig and Seymour Papert in 1966
- Imagine a robotic turtle starting at (0, 0) in the x-y plane
 - Give it the command `turtle.forward(15)`, and it moves (on-screen!) 15 pixels in the direction it is facing, drawing a line as it moves
 - Give it the command `turtle.right(25)`, and it rotates in-place 25 degrees clockwise.
- By combining together these and similar commands, intricate shapes and pictures can easily be drawn.

Mathematical drawing with Turtle



Q & A