# Python For Data Science *Cheat Sheet*

## SciPy - Linear Algebra

Learn More Python for Data Science Interactively at www.datacamp.com

### SciPy

The **SciPy** library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.

### Interacting With NumPy

Also see NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]])
```

#### Index Tricks

```
>>> np.mgrid[0:5,0:5]            Create a dense meshgrid
>>> np.ogrid[0:2,0:2]            Create an open meshgrid
>>> np.r_[3,[0]*5,-1:1:10j]      Stack arrays vertically (row-wise)
>>> np.c_[b,c]                   Create stacked column-wise arrays
```

#### Shape Manipulation

```
>>> np.transpose(b)              Permute array dimensions
>>> b.flatten()                  Flatten the array
>>> np.hstack((b,c))             Stack arrays horizontally (column-wise)
>>> np.vstack((a,b))             Stack arrays vertically (row-wise)
>>> np.hsplit(c,2)               Split the array horizontally at the 2nd index
>>> np.vsplit(d,2)               Split the array vertically at the 2nd index
```

#### Polynomials

```
>>> from numpy import polyld
>>> p = polyld([3,4,5])          Create a polynomial object
```

#### Vectorizing Functions

```
>>> def myfunc(a):
        if a < 0:
          return a*2
        else:
          return a/2
>>> np.vectorize(myfunc)         Vectorize functions
```

#### Type Handling

```
>>> np.real(b)                   Return the real part of the array elements
>>> np.imag(b)                   Return the imaginary part of the array elements
>>> np.real_if_close(c,tol=1000) Return a real array if complex parts close to 0
>>> np.cast['f'](np.pi)          Cast object to a data type
```

#### Other Useful Functions

```
>>> np.angle(b,deg=True)         Return the angle of the complex argument
>>> g = np.linspace(0,np.pi,num=5) Create an array of evenly spaced values
                                   (number of samples)
>>> g [3:] += np.pi
>>> np.unwrap(g)                 Unwrap
>>> np.logspace(0,10,3)          Create an array of evenly spaced values (log scale)
>>> np.select([c<4],[c*2])       Return values from a list of arrays depending on
                                   conditions
>>> misc.factorial(a)            Factorial
>>> misc.comb(10,3,exact=True)   Combine N things taken at k time
>>> misc.central_diff_weights(3) Weights for Np-point central derivative
>>> misc.derivative(myfunc,1.0)  Find the n-th derivative of a function at a point
```

## Linear Algebra

Also see NumPy

You'll use the `linalg` and `sparse` modules. Note that `scipy.linalg` contains and expands on `numpy.linalg`.

```
>>> from scipy import linalg, sparse
```

### Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

### Basic Matrix Routines

#### Inverse

```
>>> A.I                          Inverse
>>> linalg.inv(A)                Inverse
```

#### Transposition

```
>>> A.T                          Tranpose matrix
>>> A.H                          Conjugate transposition
```

#### Trace

```
>>> np.trace(A)                  Trace
```

#### Norm

```
>>> linalg.norm(A)               Frobenius norm
>>> linalg.norm(A,1)             L1 norm (max column sum)
>>> linalg.norm(A,np.inf)        L inf norm (max row sum)
```

#### Rank

```
>>> np.linalg.matrix_rank(C)     Matrix rank
```

#### Determinant

```
>>> linalg.det(A)                Determinant
```

#### Solving linear problems

```
>>> linalg.solve(A,b)            Solver for dense matrices
>>> E = np.mat(a).T              Solver for dense matrices
>>> linalg.lstsq(F,E)            Least-squares solution to linear matrix
                                   equation
```

#### Generalized inverse

```
>>> linalg.pinv(C)               Compute the pseudo-inverse of a matrix
                                   (least-squares solver)
>>> linalg.pinv2(C)              Compute the pseudo-inverse of a matrix
                                   (SVD)
```

### Creating Sparse Matrices

```
>>> F = np.eye(3, k=1)           Create a 2X2 identity matrix
>>> G = np.mat(np.identity(2))   Create a 2x2 identity matrix
>>> C[C > 0.5] = 0
>>> H = sparse.csr_matrix(C)     Compressed Sparse Row matrix
>>> I = sparse.csc_matrix(D)     Compressed Sparse Column matrix
>>> J = sparse.dok_matrix(A)     Dictionary Of Keys matrix
>>> E.todense()                  Sparse matrix to full matrix
>>> sparse.issparse(c)           Identify sparse matrix
```

### Sparse Matrix Routines

#### Inverse

```
>>> sparse.linalg.inv(I)         Inverse
```

#### Norm

```
>>> sparse.linalg.norm(I)        Norm
```

#### Solving linear problems

```
>>> sparse.linalg.spsolve(H,I)   Solver for sparse matrices
```

### Sparse Matrix Functions

```
>>> sparse.linalg.expm(I)        Sparse matrix exponential
```

### Matrix Functions

#### Addition

```
>>> np.add(A,D)                  Addition
```

#### Subtraction

```
>>> np.subtract(A,D)             Subtraction
```

#### Division

```
>>> np.divide(A,D)               Division
```

#### Multiplication

```
>>> A @ D                        Multiplication operator (Python 3)
>>> np.multiply(D,A)             Multiplication
>>> np.dot(A,D)                  Dot product
>>> np.vdot(A,D)                 Vector dot product
>>> np.inner(A,D)                Inner product
>>> np.outer(A,D)                Outer product
>>> np.tensordot(A,D)            Tensor dot product
>>> np.kron(A,D)                 Kronecker product
```

#### Exponential Functions

```
>>> linalg.expm(A)               Matrix exponential
>>> linalg.expm2(A)              Matrix exponential (Taylor Series)
>>> linalg.expm3(D)              Matrix exponential (eigenvalue
                                   decomposition)
```

#### Logarithm Function

```
>>> linalg.logm(A)               Matrix logarithm
```

#### Trigonometric Functions

```
>>> linalg.sinm(D)               Matrix sine
>>> linalg.cosm(D)               Matrix cosine
>>> linalg.tanm(A)               Matrix tangent
```

#### Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(D)              Hyperbolic matrix sine
>>> linalg.coshm(D)              Hyperbolic matrix cosine
>>> linalg.tanhm(A)              Hyperbolic matrix tangent
```

#### Matrix Sign Function

```
>>> np.signm(A)                  Matrix sign function
```

#### Matrix Square Root

```
>>> linalg.sqrtm(A)              Matrix square root
```

#### Arbitrary Functions

```
>>> linalg.funm(A, lambda x: x*x) Evaluate matrix function
```

### Decompositions

#### Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(A)        Solve ordinary or generalized
                                   eigenvalue problem for square matrix
>>> l1, l2 = la                  Unpack eigenvalues
>>> v[:,0]                       First eigenvector
>>> v[:,1]                       Second eigenvector
>>> linalg.eigvals(A)            Unpack eigenvalues
```

#### Singular Value Decomposition

```
>>> U,s,Vh = linalg.svd(B)       Singular Value Decomposition (SVD)
>>> M,N = B.shape
>>> Sig = linalg.diagsvd(s,M,N)  Construct sigma matrix in SVD
```

#### LU Decomposition

```
>>> P,L,U = linalg.lu(C)         LU Decomposition
```

### Sparse Matrix Decompositions

```
>>> la, v = sparse.linalg.eigs(F,1)  Eigenvalues and eigenvectors
>>> sparse.linalg.svds(H, 2)         SVD
```

### Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

DataCamp
Learn Python for Data Science Interactively

# Python For Data Science *Cheat Sheet*

## Matplotlib

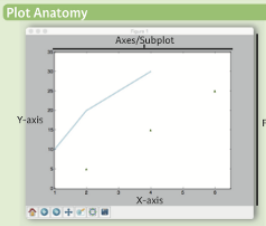
Learn Python Interactively at www.DataCamp.com

### Matplotlib

**Matplotlib** is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

### Plot Anatomy & Workflow

#### Plot Anatomy




#### Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3) Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

### 1 Prepare The Data

Also see Lists & NumPy

#### 1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

#### 2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

### 2 Create Plot

```
>>> import matplotlib.pyplot as plt
```

#### Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

#### Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

### 4 Customize Plot

#### Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

#### Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

#### Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

#### Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

#### Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

#### Limits, Legends & Layouts

##### Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)              Add padding to a plot
>>> ax.axis('equal')                     Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5]) Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5)                  Set limits for x-axis
```

##### Legends

```
>>> ax.set(title='An Example Axes',      Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                No overlapping plot elements
```

##### Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),       Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',             Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

##### Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,     Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()                   Fit subplot(s) in to the figure area
```

##### Axis Spines

```
>>> ax1.spines['top'].set_visible(False) Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position(('outward',10)) Move the bottom axis line outward
```

### 3 Plotting Routines

#### 1D Data

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y)                 Draw points with lines or markers connecting them
>>> ax.scatter(x,y)                      Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5])       Plot vertical rectangles (constant width)
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])  Plot horiontal rectangles (constant height)
>>> axes[1,1].axhline(0.45)              Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65)              Draw a vertical line across axes
>>> ax.fill(x,y,color='blue')            Draw filled polygons
>>> ax.fill_between(x,y,color='yellow')  Fill between y-values and 0
```

#### Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)         Add an arrow to the axes
>>> axes[1,1].quiver(y,z)                Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V)        Plot a 2D field of arrows
```

#### Data Distributions

```
>>> ax1.hist(y)                          Plot a histogram
>>> ax3.boxplot(y)                       Make a box and whisker plot
>>> ax3.violinplot(z)                    Make a violin plot
```

#### 2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,                  Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> axes2[0].pcolor(data2)               Pseudocolor plot of 2D array
>>> axes2[0].pcolormesh(data)            Pseudocolor plot of 2D array
>>> CS = plt.contour(Y,X,U)              Plot contours
>>> axes2[2].contourf(data1)             Plot filled contours
>>> axes2[2]= ax.clabel(CS)              Label a contour plot
```

### 5 Save Plot

#### Save figures

```
>>> plt.savefig('foo.png')
```

#### Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

### 6 Show Plot

```
>>> plt.show()
```

### Close & Clear

```
>>> plt.cla()                            Clear an axis
>>> plt.clf()                            Clear the entire figure
>>> plt.close()                          Close a window
```

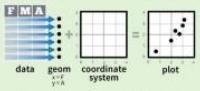
DataCamp
Learn Python for Data Science Interactively

# Data Visualization with ggplot2 Cheat Sheet

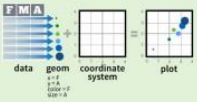
RStudio

## Basics

**ggplot2** is based on the **grammar of graphics**, the idea that you can build every graph from the same few components: a **data** set, a set of **geoms**—visual marks that represent data points, and a **coordinate system.**




To display data values, map variables in the data set to aesthetic properties of the geom like **size**, **color**, and **x** and **y** locations.




Build a graph with **qplot()** or **ggplot()**

aesthetic mappings | data | geom

**qplot**(x = cty, y = hwy, color = cyl, data = mpg, geom = "point")
Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

**ggplot**(data = mpg, **aes**(x = cty, y = hwy))
Begins a plot that you finish by adding layers to. No defaults, but provides more control than qplot().

data

```
ggplot(mpg, aes(hwy, cty)) +
 geom_point(aes(color = cyl)) +
 geom_smooth(method ="lm") +
 coord_cartesian() +
 scale_color_gradient() +
 theme_bw()
```

add layers, elements with +

layer = geom + default stat + layer specific mappings

additional elements

Add a new layer to a plot with a **geom_*()** or **stat_*()** function. Each provides a geom, a set of aesthetic mappings, and a default stat and position adjustment.

**last_plot()**
Returns the last plot

**ggsave("plot.png", width = 5, height = 5)**
Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

## Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

### One Variable

#### Continuous

a <- ggplot(mpg, aes(hwy))

**a + geom_area(stat = "bin")**
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")

**a + geom_density(**kernel = "gaussian"**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

**a + geom_dotplot()**
x, y, alpha, color, fill

**a + geom_freqpoly()**
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y = ..density..))

**a + geom_histogram(**binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(y = ..density..))

#### Discrete

b <- ggplot(mpg, aes(fl))

**b + geom_bar()**
x, alpha, color, fill, linetype, size, weight

### Graphical Primitives

c <- ggplot(map, aes(long, lat))

**c + geom_polygon(**aes(group = group)**)**
x, y, alpha, color, fill, linetype, size

d <- ggplot(economics, aes(date, unemploy))

**d + geom_path(**lineend="butt", linejoin="round', linemitre=1**)**
x, y, alpha, color, linetype, size

**d + geom_ribbon(**aes(ymin=unemploy - 900, ymax=unemploy + 900)**)**
x, ymax, ymin, alpha, color, fill, linetype, size

e <- ggplot(seals, aes(x = long, y = lat))

**e + geom_segment(**aes(xend = long + delta_long, yend = lat + delta_lat)**)**
x, xend, y, yend, alpha, color, linetype, size

**e + geom_rect(**aes(xmin = long, ymin = lat, xmax= long + delta_long, ymax = lat + delta_lat)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

### Two Variables

#### Continuous X, Continuous Y

f <- ggplot(mpg, aes(cty, hwy))

**f + geom_blank()**

**f + geom_jitter()**
x, y, alpha, color, fill, shape, size

**f + geom_point()**
x, y, alpha, color, fill, shape, size

**f + geom_quantile()**
x, y, alpha, color, linetype, size, weight

**f + geom_rug(**sides = "bl"**)**
alpha, color, linetype, size

**f + geom_smooth(**model = lm**)**
x, y, alpha, color, fill, linetype, size, weight

**f + geom_text(**aes(label = cty)**)**
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

#### Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))

**g + geom_bar(stat = "identity")**
x, y, alpha, color, fill, linetype, size, weight

**g + geom_boxplot()**
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight

**g + geom_dotplot(**binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill

**g + geom_violin(**scale = "area"**)**
x, y, alpha, color, fill, linetype, size, weight

#### Discrete X, Discrete Y

h <- ggplot(diamonds, aes(cut, color))

**h + geom_jitter()**
x, y, alpha, color, fill, shape, size

#### Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

**i + geom_bin2d(**binwidth = c(5, 0.5)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight

**i + geom_density2d()**
x, y, alpha, colour, linetype, size

**i + geom_hex()**
x, y, alpha, colour, fill size

#### Continuous Function

j <- ggplot(economics, aes(date, unemploy))

**j + geom_area()**
x, y, alpha, color, fill, linetype, size

**j + geom_line()**
x, y, alpha, color, linetype, size

**j + geom_step(**direction = "hv"**)**
x, y, alpha, color, linetype, size

#### Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

**k + geom_crossbar(**fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, linetype, size

**k + geom_errorbar()**
x, ymax, ymin, alpha, color, linetype, size, width (also **geom_errorbarh()**)

**k + geom_linerange()**
x, ymin, ymax, alpha, color, linetype, size

**k + geom_pointrange()**
x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

#### Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

**l + geom_map(**aes(map_id = state), map = map**) +**
**expand_limits(**x = map$long, y = map$lat**)**
map_id, alpha, color, fill, linetype, size

### Three Variables

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))

**m + geom_contour(**aes(z = z)**)**
x, y, z, alpha, colour, linetype, size, weight

**m + geom_raster(**aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE**)**
x, y, alpha, fill

**m + geom_tile(**aes(fill = z)**)**
x, y, alpha, color, fill, linetype, size



Learn more at **docs.ggplot2.org** • ggplot2 0.9.3.1 • Updated: 3/15