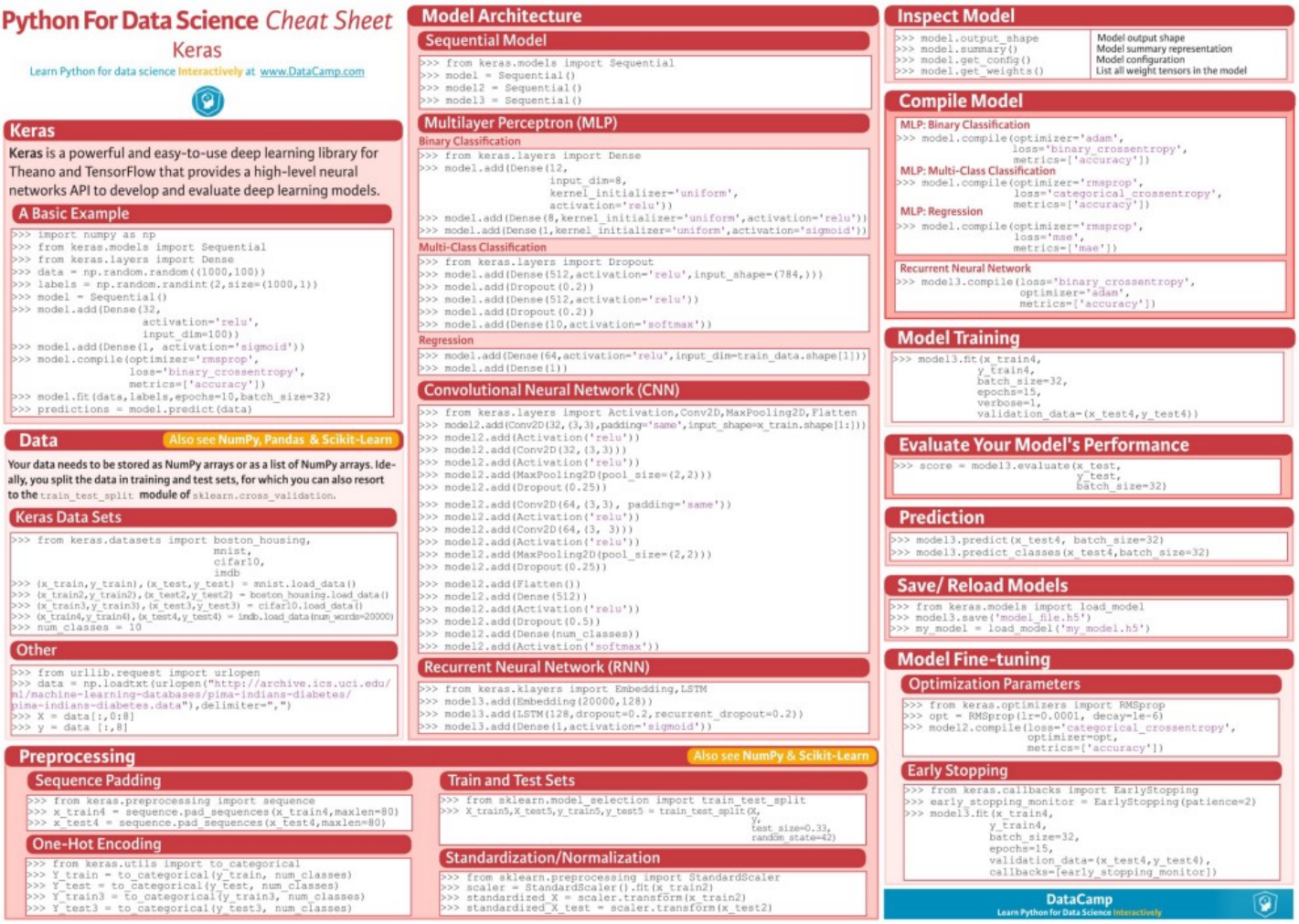
Kailash Ahirwar Follow
Co-founder @ Mate Labs | Democratizing Artificial Intelligence
May 28 · 2 min read

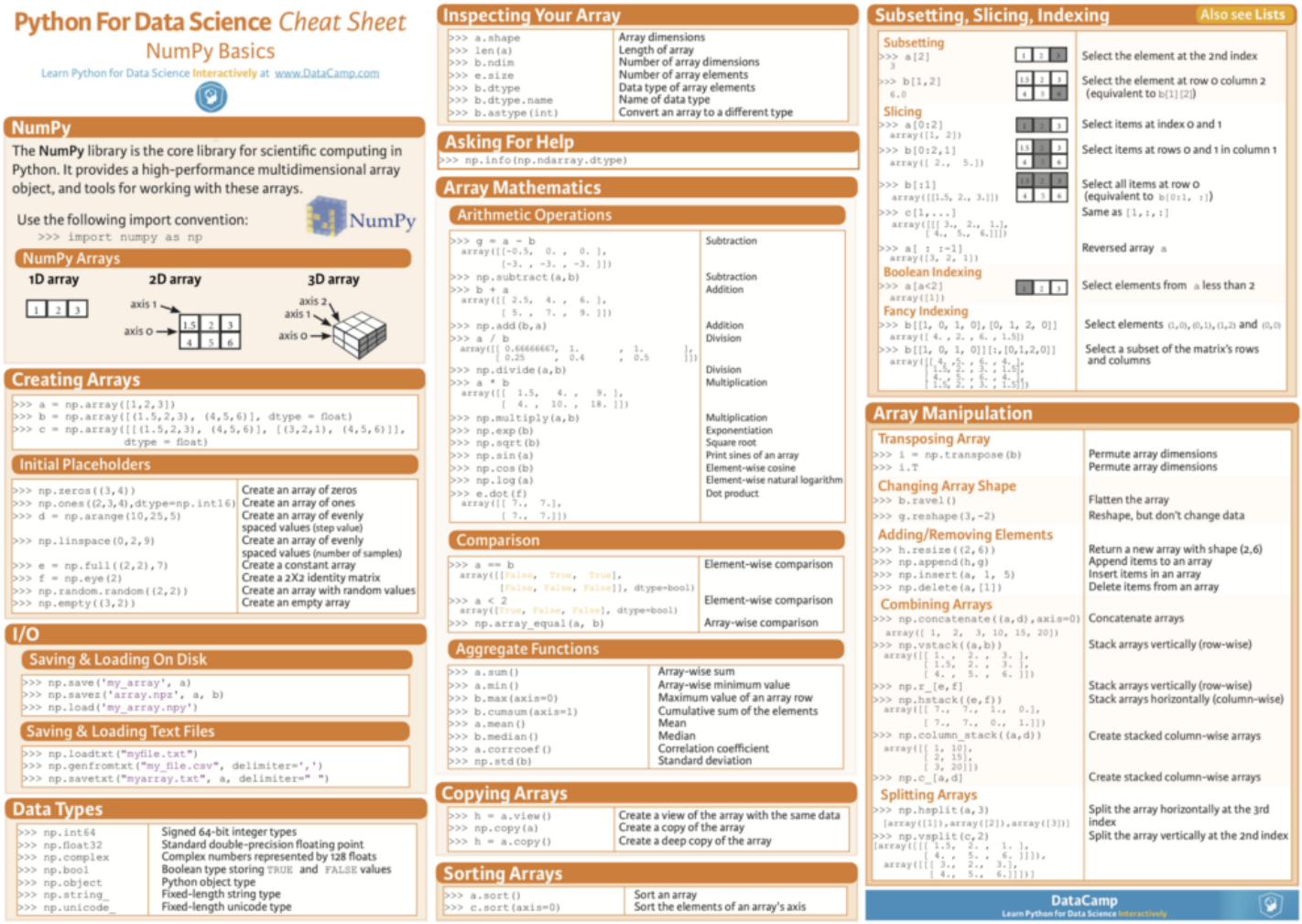
Essential Cheat Sheets for Machine Learning and Deep Learning Engineers

Learning machine learning and deep learning is difficult for newbies. As well as deep learning libraries are difficult to understand. I am creating a repository on Github(cheatsheets-ai) with cheat sheets which I collected from different sources. Do give it a visit and contribute cheatsheets if you have any. Thanks

1. Keras



2. Numpy


Python For Data Science Cheat Sheet
NumPy Basics
Learn Python for Data Science Interactively at www.DataCamp.com
NumPy
The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.
Use the following import convention:
>>> import numpy as np
NumPy Arrays
1D array
2D array
3D array
axis 1
axis 0
axis 2
Creating Arrays
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]], dtype = float)
Initial Placeholders
>>> np.zeros((3,4)) Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16) Create an array of ones
>>> d = np.arange(10,25,5) Create an array of evenly spaced values (step value)
>>> np.linspace(0,2,9) Create an array of evenly spaced values (number of samples)
>>> e = np.full((2,2),7) Create a constant array
>>> f = np.eye(2) Create a 2X2 identity matrix
>>> np.random.random((2,2)) Create an array with random values
>>> np.empty((3,2)) Create an empty array
I/O
Saving & Loading On Disk
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
Saving & Loading Text Files
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
Data Types
>>> np.int64 Signed 64-bit integer types
>>> np.float32 Standard double-precision floating point
>>> np.complex Complex numbers represented by 128 floats
>>> np.bool Boolean type storing TRUE and FALSE values
>>> np.object Python object type
>>> np.string_ Fixed-length string type
>>> np.unicode_ Fixed-length unicode type
Inspecting Your Array
>>> a.shape Array dimensions
>>> len(a) Length of array
>>> b.ndim Number of array dimensions
>>> e.size Number of array elements
>>> b.dtype Data type of array elements
>>> b.dtype.name Name of data type
>>> b.astype(int) Convert an array to a different type
Asking For Help
>>> np.info(np.ndarray.dtype)
Array Mathematics
Arithmetic Operations
>>> g = a - b Subtraction
>>> np.subtract(a,b) Subtraction
>>> b + a Addition
>>> np.add(b,a) Addition
>>> a / b Division
>>> np.divide(a,b) Division
>>> a * b Multiplication
>>> np.multiply(a,b) Multiplication
>>> np.exp(b) Exponentiation
>>> np.sqrt(b) Square root
>>> np.sin(a) Print sines of an array
>>> np.cos(b) Element-wise cosine
>>> np.log(a) Element-wise natural logarithm
>>> e.dot(f) Dot product
Comparison
>>> a == b Element-wise comparison
>>> a < 2 Element-wise comparison
>>> np.array_equal(a, b) Array-wise comparison
Aggregate Functions
>>> a.sum() Array-wise sum
>>> a.min() Array-wise minimum value
>>> b.max(axis=0) Maximum value of an array row
>>> b.cumsum(axis=1) Cumulative sum of the elements
>>> a.mean() Mean
>>> b.median() Median
>>> a.corrcoef() Correlation coefficient
>>> np.std(b) Standard deviation
Copying Arrays
>>> h = a.view() Create a view of the array with the same data
>>> np.copy(a) Create a copy of the array
>>> h = a.copy() Create a deep copy of the array
Sorting Arrays
>>> a.sort() Sort an array
>>> c.sort(axis=0) Sort the elements of an array's axis
Subsetting, Slicing, Indexing
Also see Lists
Subsetting
>>> a[2] Select the element at the 2nd index
>>> b[1,2] Select the element at row 0 column 2 (equivalent to b[1][2])
Slicing
>>> a[0:2] Select items at index 0 and 1
>>> b[0:2,1] Select items at rows 0 and 1 in column 1
>>> b[:1] Select all items at row 0 (equivalent to b[0:1, :])
>>> c[1,...] Same as [1,:,:]
>>> a[: :-1] Reversed array a
Boolean Indexing
>>> a[a<2] Select elements from a less than 2
Fancy Indexing
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]] Select elements (1,0),(0,1),(1,2) and (0,0)
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]] Select a subset of the matrix's rows and columns
Array Manipulation
Transposing Array
>>> i = np.transpose(b) Permute array dimensions
>>> i.T Permute array dimensions
Changing Array Shape
>>> b.ravel() Flatten the array
>>> g.reshape(3,-2) Reshape, but don't change data
Adding/Removing Elements
>>> h.resize((2,6)) Return a new array with shape (2,6)
>>> np.append(h,g) Append items to an array
>>> np.insert(a, 1, 5) Insert items in an array
>>> np.delete(a,[1]) Delete items from an array
Combining Arrays
>>> np.concatenate((a,d),axis=0) Concatenate arrays
>>> np.vstack((a,b)) Stack arrays vertically (row-wise)
>>> np.r_[e,f] Stack arrays vertically (row-wise)
>>> np.hstack((e,f)) Stack arrays horizontally (column-wise)
>>> np.column_stack((a,d)) Create stacked column-wise arrays
>>> np.c_[a,d] Create stacked column-wise arrays
Splitting Arrays
>>> np.hsplit(a,3) Split the array horizontally at the 3rd index
>>> np.vsplit(c,2) Split the array vertically at the 2nd index
DataCamp
Learn Python for Data Science Interactively

3. Pandas

Data Wrangling with pandas Cheat Sheet

http://pandas.pydata.org

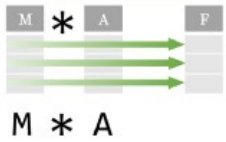
Tidy Data – A foundation for wrangling in pandas

In a tidy data set:

Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.



Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
        index = [1, 2, 3])
```
Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```
Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
          names=['n','v'])))
```
Create DataFrame with a MultiIndex

Method Chaining

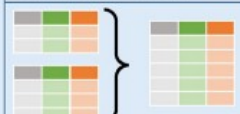
Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
    )
```

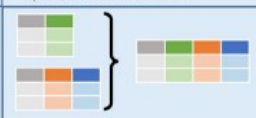
Reshaping Data – Change the layout of a data set

`pd.melt(df)`
Gather columns into rows.

`df.pivot(columns='var', values='val')`
Spread rows into columns.



`pd.concat([df1,df2])`
Append rows of DataFrames



`pd.concat([df1,df2], axis=1)`
Append columns of DataFrames

`df.sort_values('mpg')`
Order rows by values of a column (low to high).

`df.sort_values('mpg',ascending=False)`
Order rows by values of a column (high to low).

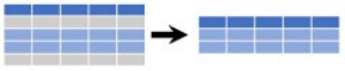
`df.rename(columns = {'y':'year'})`
Rename the columns of a DataFrame

`df.sort_index()`
Sort the index of a DataFrame

`df.reset_index()`
Reset index of DataFrame to row numbers, moving index to columns.

`df.drop(['Length','Height'], axis=1)`
Drop columns from DataFrame

Subset Observations (Rows)



`df[df.Length > 7]`
Extract rows that meet logical criteria.

`df.drop_duplicates()`
Remove duplicate rows (only considers columns).

`df.head(n)`
Select first n rows.

`df.tail(n)`
Select last n rows.

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)`
Randomly select n rows.

`df.iloc[10:20]`
Select rows by position.

`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

Logic in Python (and pandas)			
<	Less than	!=	Not equal to
>	Greater than	`df.column.isin(values)`	Group membership
==	Equals	`pd.isnull(obj)`	Is NaN
<=	Less than or equals	`pd.notnull(obj)`	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

Subset Variables (Columns)

`df[['width','length','species']]`
Select multiple columns with specific names.

`df['width']` *or* `df.width`
Select single column with specific name.

`df.filter(regex='regex')`
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples	
`'\.'`	Matches strings containing a period '.'
`'Length$'`	Matches strings ending with word 'Length'
`'^Sepal'`	Matches strings beginning with the word 'Sepal'
`'^x[1-5]$'`	Matches strings beginning with 'x' and ending with 1,2,3,4,5
`''^(?!Species$).*'`	Matches strings except the string 'Species'

`df.loc[:,'x2':'x4']`
Select all columns between x2 and x4 (inclusive).

`df.iloc[:,[1,2,5]]`
Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[df['a'] > 10, ['a','c']]`
Select rows meeting logical condition, and only the specific columns .

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

Summarize Data

`df['w'].value_counts()`
Count number of rows with each unique value of variable

`len(df)`
# of rows in DataFrame.

`df['w'].nunique()`
# of distinct values in a column.

`df.describe()`
Basic descriptive statistics for each column (or GroupBy)

pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()`
Sum values of each object.

`count()`
Count non-NA/null values of each object.

`median()`
Median value of each object.

`quantile([0.25,0.75])`
Quantiles of each object.

`apply(function)`
Apply function to each object.

`min()`
Minimum value in each object.

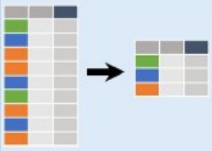
`max()`
Maximum value in each object.

`mean()`
Mean value of each object.

`var()`
Variance of each object.

`std()`
Standard deviation of each object.

Group Data



`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

`size()`
Size of each group.

`agg(function)`
Aggregate group using function.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.

`df.fillna(value)`
Replace all NA/null data with value.

Make New Columns

`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.

`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.

`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.

pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

`max(axis=1)`
Element-wise max.

`clip(lower=-10,upper=10)`
Trim values at input thresholds

`min(axis=1)`
Element-wise min.

`abs()`
Absolute value.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

`shift(1)`
Copy with values shifted by 1.

`rank(method='dense')`
Ranks with no gaps.

`rank(method='min')`
Ranks. Ties get min rank.

`rank(pct=True)`
Ranks rescaled to interval [0, 1].

`rank(method='first')`
Ranks. Ties go to first value.

`shift(-1)`
Copy with values lagged by 1.

`cumsum()`
Cumulative sum.

`cummax()`
Cumulative max.

`cummin()`
Cumulative min.

`cumprod()`
Cumulative product.

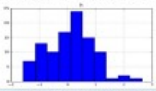
Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.

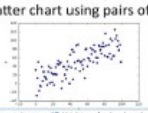
`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Plotting

`df.plot.hist()`
Histogram for each column



`df.plot.scatter(x='w',y='h')`
Scatter chart using pairs of points



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

bdf

x1	x3
A	T
B	F
D	T

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

x1	x2
C	3

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3

zdf

x1	x2
B	2
C	3
D	4

Set-like Operations

x1	x2
B	2
C	3

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

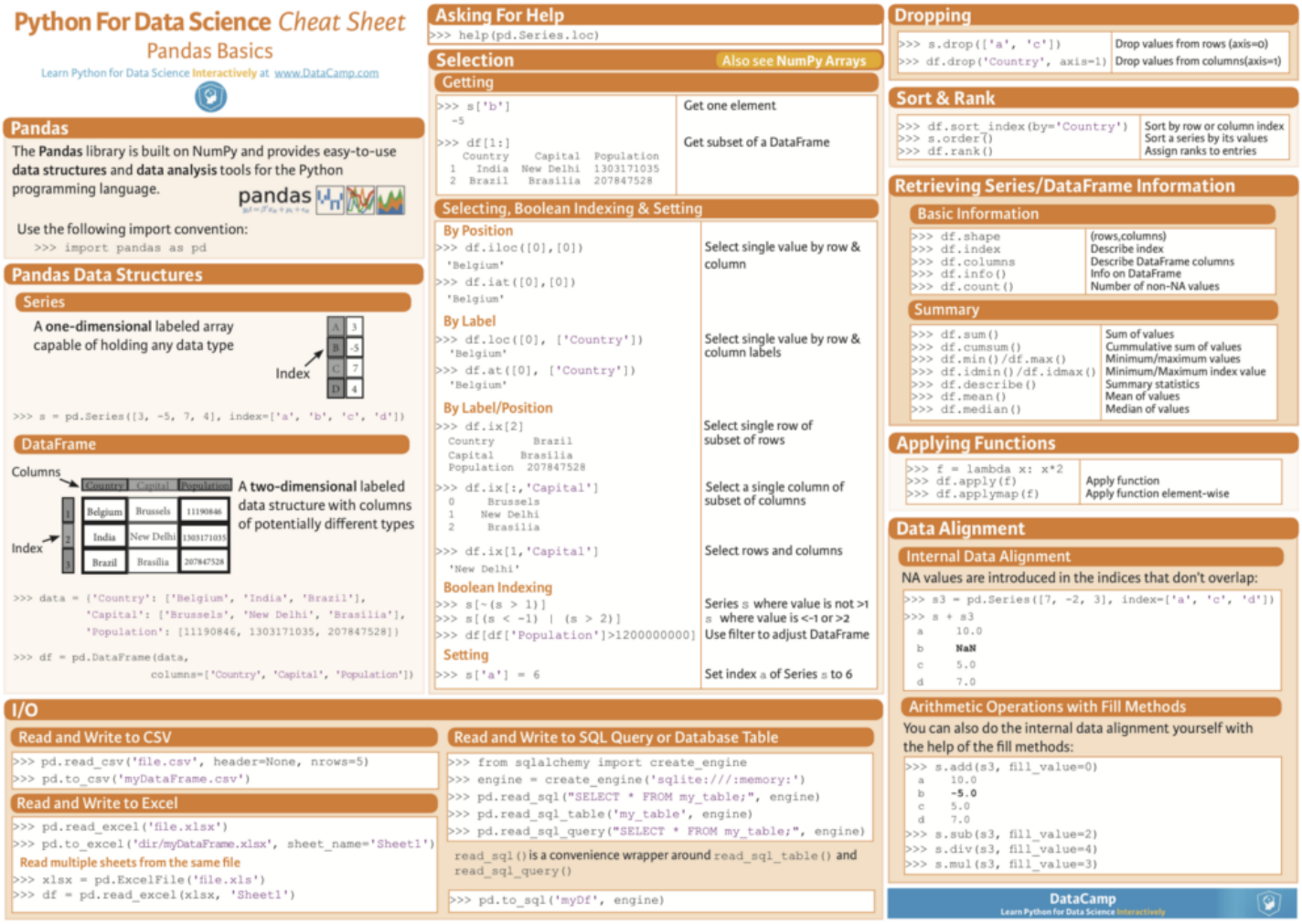
x1	x2
A	1
B	2
C	3
D	4

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

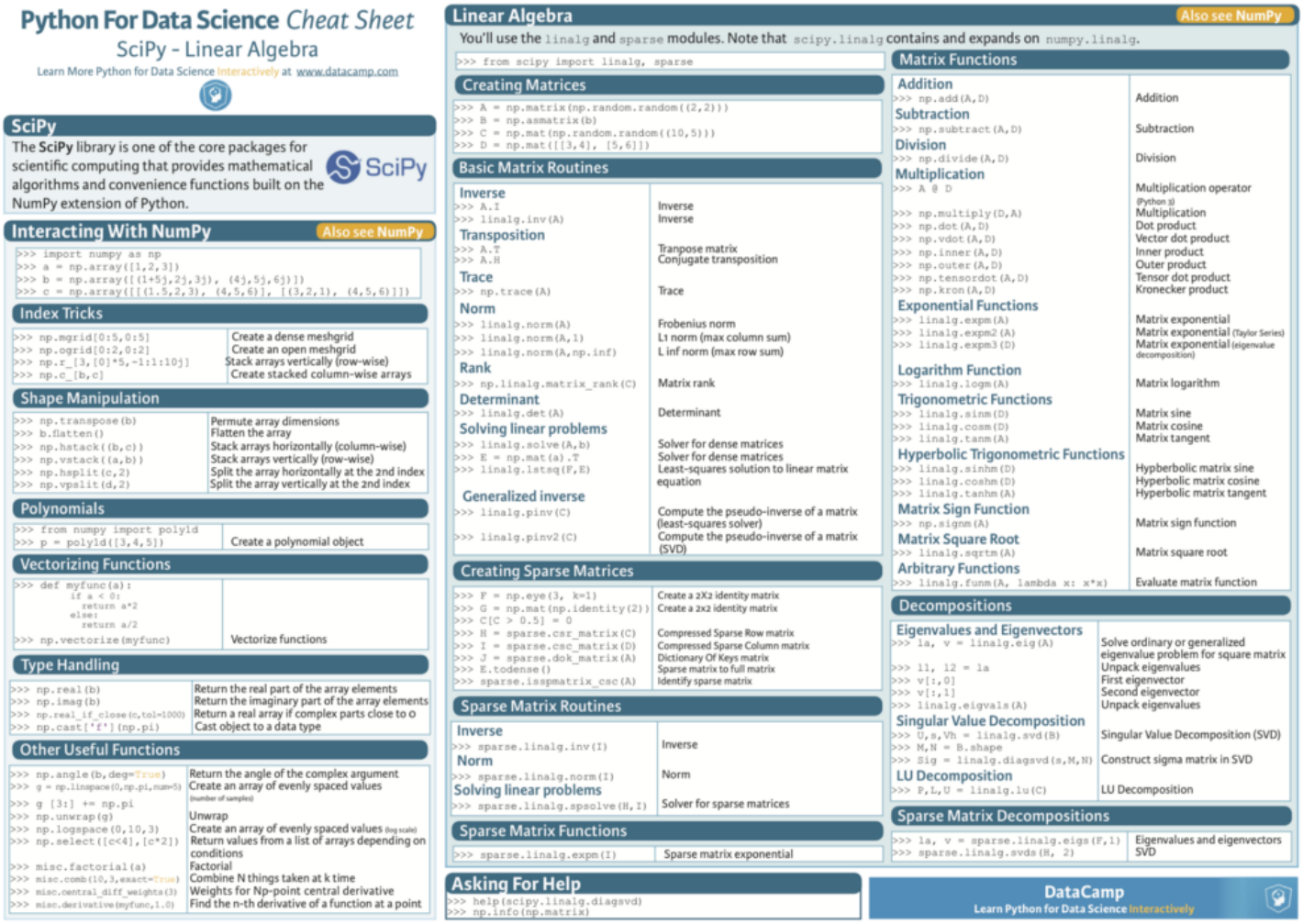
x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(['_merge'],axis=1)
```
Rows that appear in ydf but not zdf (Setdiff).

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants


Python For Data Science Cheat Sheet
Pandas Basics
Learn Python for Data Science Interactively at www.DataCamp.com
Pandas
The Pandas library is built on NumPy and provides easy-to-use data structures and data analysis tools for the Python programming language.
Use the following import convention:
>>> import pandas as pd
Pandas Data Structures
Series
A one-dimensional labeled array capable of holding any data type
Index
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
DataFrame
Columns
Index
A two-dimensional labeled data structure with columns of potentially different types
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
'Population': [11190846, 1303171035, 207847528]}
>>> df = pd.DataFrame(data,
columns=['Country', 'Capital', 'Population'])
Asking For Help
>>> help(pd.Series.loc)
Selection
Also see NumPy Arrays
Getting
>>> s['b']
-5
Get one element
>>> df[1:]
Country Capital Population
1 India New Delhi 1303171035
2 Brazil Brasilia 207847528
Get subset of a DataFrame
Selecting, Boolean Indexing & Setting
By Position
>>> df.iloc[[0],[0]]
'Belgium'
>>> df.iat[[0],[0]]
'Belgium'
Select single value by row & column
By Label
>>> df.loc[[0], ['Country']]
'Belgium'
>>> df.at[[0], ['Country']]
'Belgium'
Select single value by row & column labels
By Label/Position
>>> df.ix[2]
Country Brazil
Capital Brasilia
Population 207847528
Select single row of subset of rows
>>> df.ix[:,'Capital']
0 Brussels
1 New Delhi
2 Brasilia
Select a single column of subset of columns
>>> df.ix[1,'Capital']
'New Delhi'
Select rows and columns
Boolean Indexing
>>> s[~(s > 1)]
Series s where value is not >1
>>> s[(s < -1) | (s > 2)]
s where value is <-1 or >2
>>> df[df['Population']>1200000000]
Use filter to adjust DataFrame
Setting
>>> s['a'] = 6
Set index a of Series s to 6
Dropping
>>> s.drop(['a', 'c'])
Drop values from rows (axis=0)
>>> df.drop('Country', axis=1)
Drop values from columns(axis=1)
Sort & Rank
>>> df.sort_index(by='Country')
Sort by row or column index
>>> s.order()
Sort a series by its values
>>> df.rank()
Assign ranks to entries
Retrieving Series/DataFrame Information
Basic Information
>>> df.shape
(rows,columns)
>>> df.index
Describe index
>>> df.columns
Describe DataFrame columns
>>> df.info()
Info on DataFrame
>>> df.count()
Number of non-NA values
Summary
>>> df.sum()
Sum of values
>>> df.cumsum()
Cummulative sum of values
>>> df.min()/df.max()
Minimum/maximum values
>>> df.idmin()/df.idmax()
Minimum/Maximum index value
>>> df.describe()
Summary statistics
>>> df.mean()
Mean of values
>>> df.median()
Median of values
Applying Functions
>>> f = lambda x: x*2
>>> df.apply(f)
Apply function
>>> df.applymap(f)
Apply function element-wise
Data Alignment
Internal Data Alignment
NA values are introduced in the indices that don't overlap:
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
a 10.0
b NaN
c 5.0
d 7.0
Arithmetic Operations with Fill Methods
You can also do the internal data alignment yourself with the help of the fill methods:
>>> s.add(s3, fill_value=0)
a 10.0
b -5.0
c 5.0
d 7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
I/O
Read and Write to CSV
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> pd.to_csv('myDataFrame.csv')
Read and Write to Excel
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
Read multiple sheets from the same file
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
Read and Write to SQL Query or Database Table
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
read_sql() is a convenience wrapper around read_sql_table() and read_sql_query()
>>> pd.to_sql('myDf', engine)
DataCamp
Learn Python for Data Science Interactively

4. Scipy



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.vpslit(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

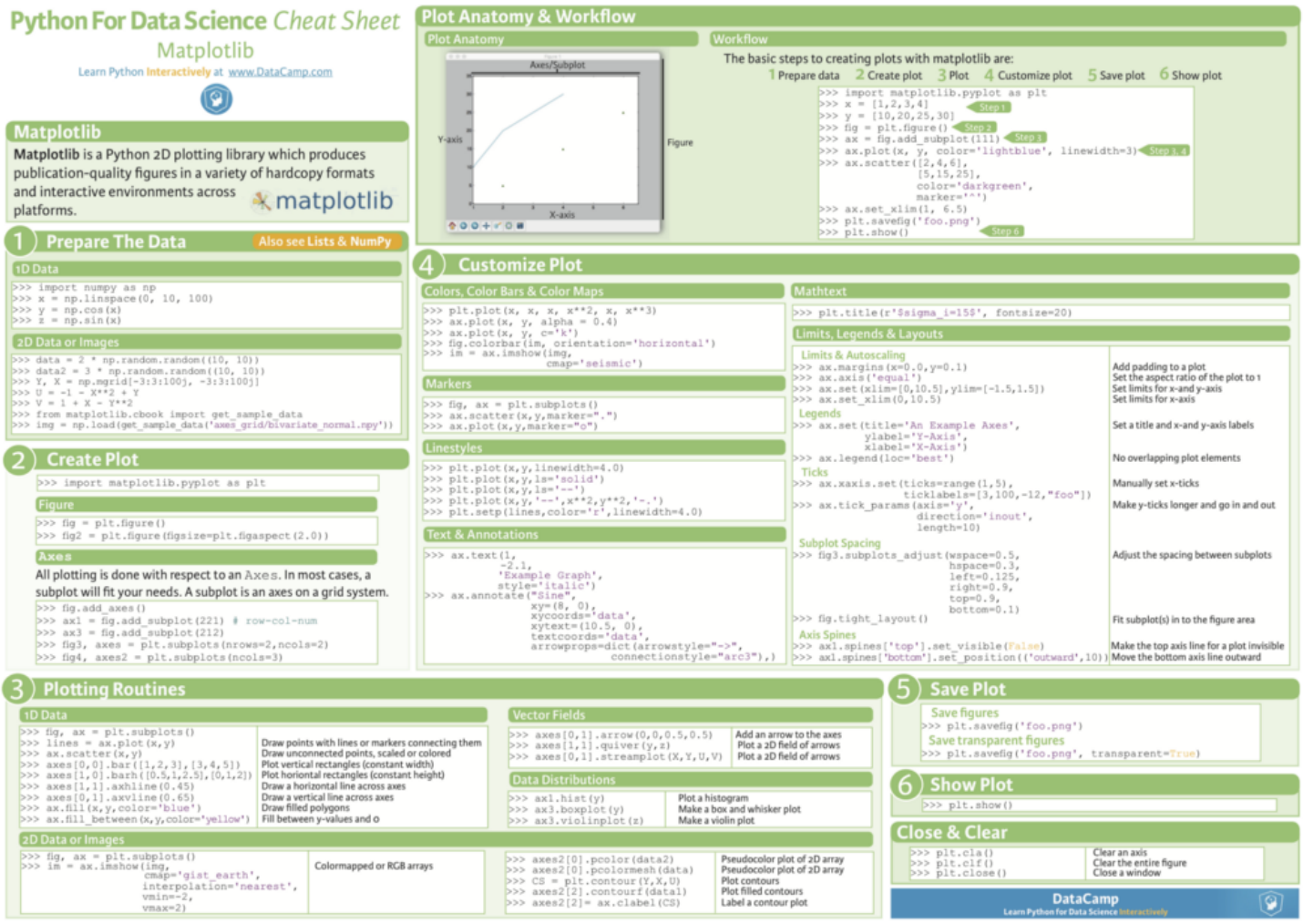
```
>>> la, v = sparse.linalg.eigs(F,1) Eigenvalues and eigenvectors
>>> sparse.linalg.svds(H, 2)        SVD
```

Asking For Help

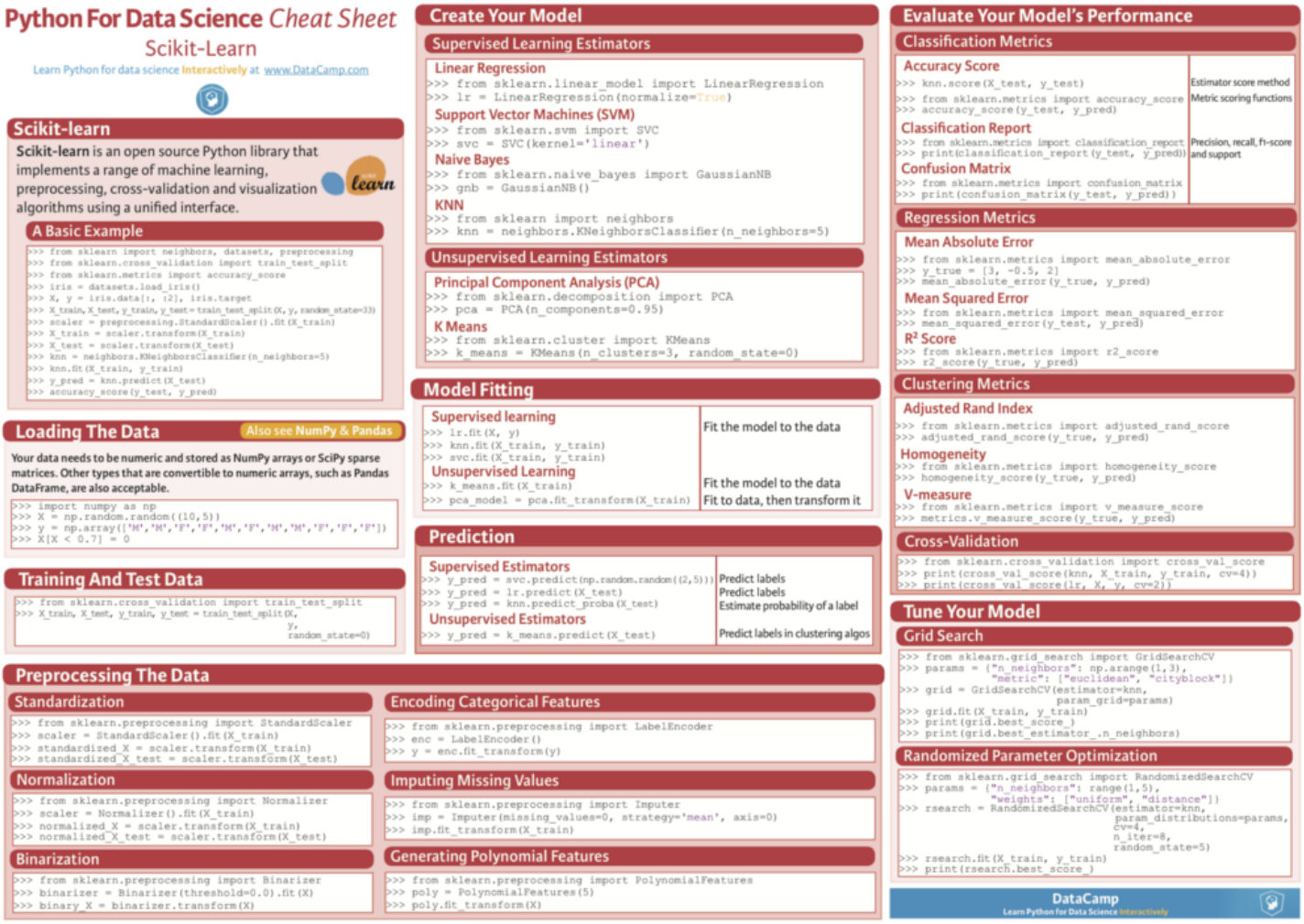
```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

DataCamp
Learn Python for Data Science Interactively

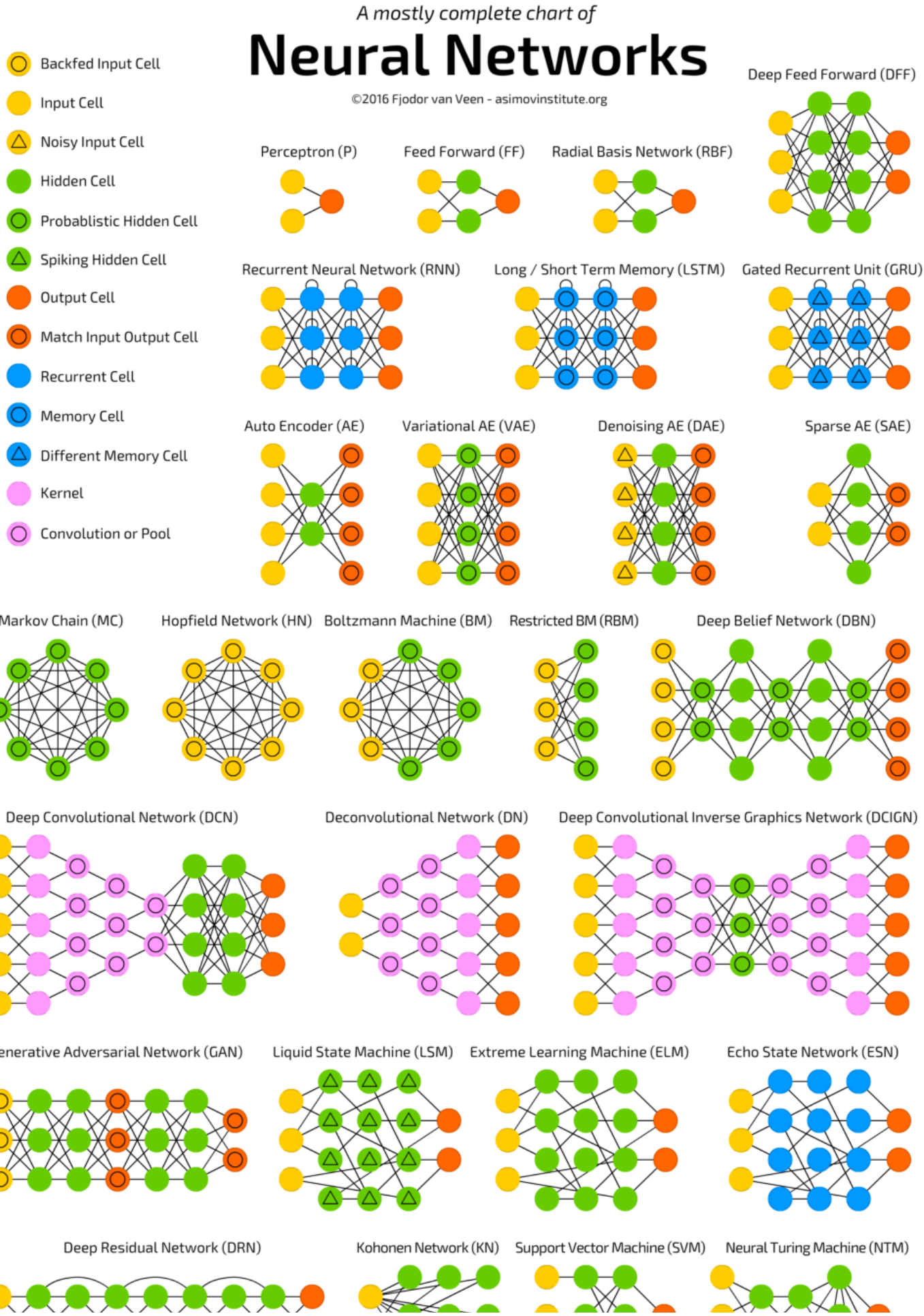
5. Matplotlib

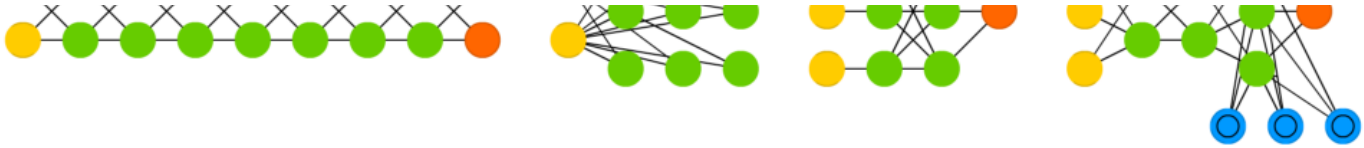


6. Scikit-learn



7. Neural Networks Zoo


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)



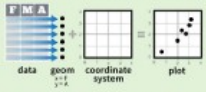
8. ggplot2

Data Visualization with ggplot2 Cheat Sheet

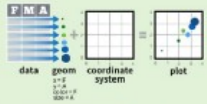
RStudio

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

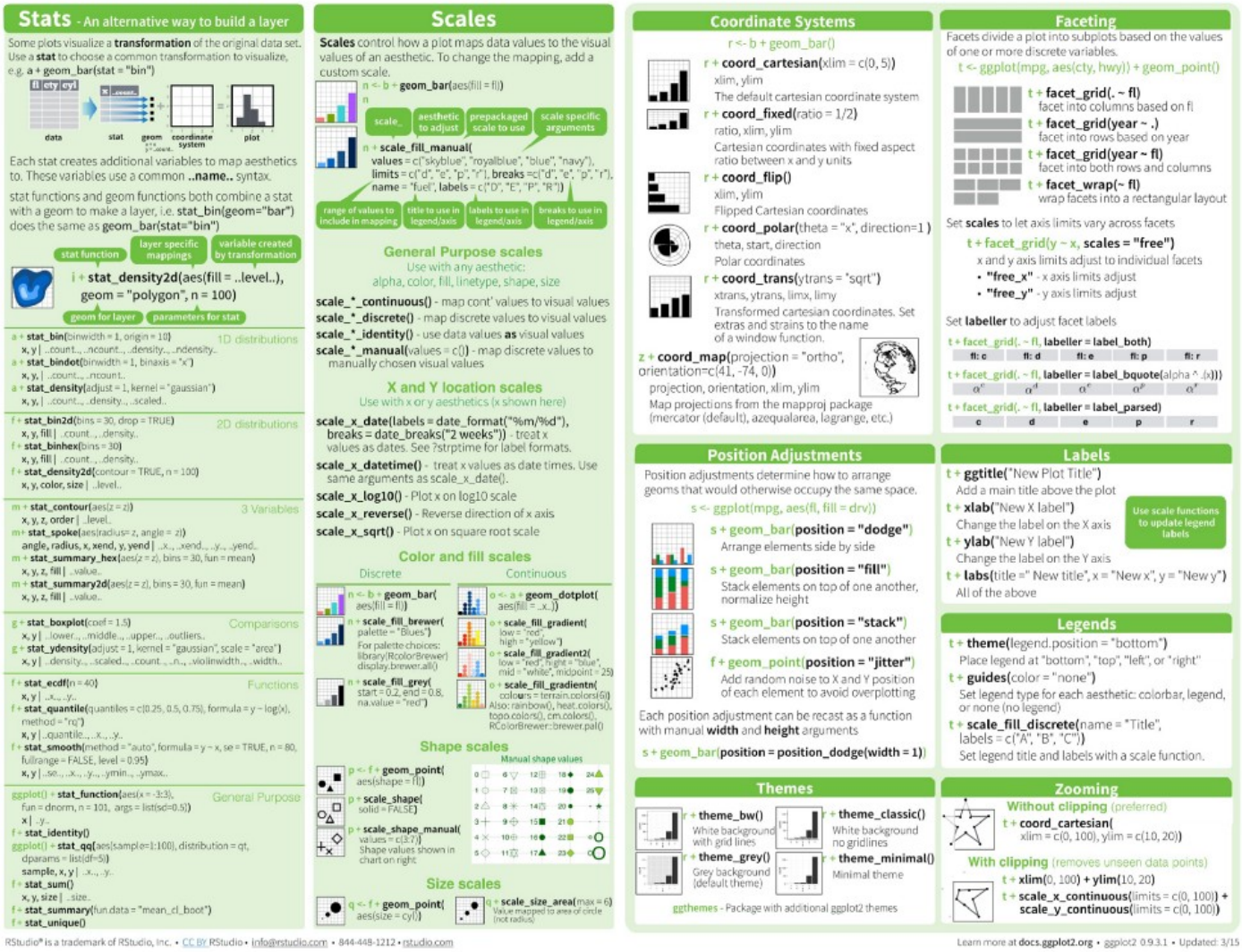
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))

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


Stats - An alternative way to build a layer
Some plots visualize a transformation of the original data set. Use a stat to choose a common transformation to visualize, e.g. a + geom_bar(stat = "bin")
data
stat
geom
coordinate system
plot
Each stat creates additional variables to map aesthetics to. These variables use a common ..name.. syntax.
stat functions and geom functions both combine a stat with a geom to make a layer, i.e. stat_bin(geom="bar") does the same as geom_bar(stat="bin")
stat function
layer specific mappings
variable created by transformation
i + stat_density2d(aes(fill = ..level..), geom = "polygon", n = 100)
geom for layer
parameters for stat
1D distributions
2D distributions
3 Variables
Comparisons
Functions
General Purpose
Scales
Scales control how a plot maps data values to the visual values of an aesthetic. To change the mapping, add a custom scale.
n <- b + geom_bar(aes(fill = fl))
scale_
aesthetic to adjust
prepackaged scale to use
scale specific arguments
n + scale_fill_manual(values = c("skyblue", "royalblue", "blue", "navy"), limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"), name = "fuel", labels = c("D", "E", "P", "R"))
range of values to include in mapping
title to use in legend/axis
labels to use in legend/axis
breaks to use in legend/axis
General Purpose scales
Use with any aesthetic: alpha, color, fill, linetype, shape, size
scale_*_continuous() - map cont' values to visual values
scale_*_discrete() - map discrete values to visual values
scale_*_identity() - use data values as visual values
scale_*_manual(values = c()) - map discrete values to manually chosen visual values
X and Y location scales
Use with x or y aesthetics (x shown here)
scale_x_date(labels = date_format("%m/%d"), breaks = date_breaks("2 weeks")) - treat x values as dates. See ?strptime for label formats.
scale_x_datetime() - treat x values as date times. Use same arguments as scale_x_date().
scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale
Color and fill scales
Discrete
Continuous
Shape scales
Manual shape values
Size scales
Coordinate Systems
r <- b + geom_bar()
r + coord_cartesian(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system
r + coord_fixed(ratio = 1/2)
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units
r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates
r + coord_polar(theta = "x", direction=1)
theta, start, direction
Polar coordinates
r + coord_trans(ytrans = "sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set extras and strains to the name of a window function.
z + coord_map(projection = "ortho", orientation=c(41, -74, 0))
projection, orientation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)
Position Adjustments
Position adjustments determine how to arrange geoms that would otherwise occupy the same space.
s <- ggplot(mpg, aes(fl, fill = drv))
s + geom_bar(position = "dodge")
Arrange elements side by side
s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height
s + geom_bar(position = "stack")
Stack elements on top of one another
f + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting
Each position adjustment can be recast as a function with manual width and height arguments
s + geom_bar(position = position_dodge(width = 1))
Themes
r + theme_bw()
White background with grid lines
r + theme_classic()
White background no gridlines
r + theme_grey()
Grey background (default theme)
r + theme_minimal()
Minimal theme
ggthemes - Package with additional ggplot2 themes
Faceting
Facets divide a plot into subplots based on the values of one or more discrete variables.
t <- ggplot(mpg, aes(cty, hwy)) + geom_point()
t + facet_grid(. ~ fl)
facet into columns based on fl
t + facet_grid(year ~ .)
facet into rows based on year
t + facet_grid(year ~ fl)
facet into both rows and columns
t + facet_wrap(~ fl)
wrap facets into a rectangular layout
Set scales to let axis limits vary across facets
t + facet_grid(y ~ x, scales = "free")
x and y axis limits adjust to individual facets
"free_x" - x axis limits adjust
"free_y" - y axis limits adjust
Set labeller to adjust facet labels
t + facet_grid(. ~ fl, labeller = label_both)
t + facet_grid(. ~ fl, labeller = label_bquote(alpha ^ .(x)))
t + facet_grid(. ~ fl, labeller = label_parsed)
Labels
t + ggtitle("New Plot Title")
Add a main title above the plot
t + xlab("New X label")
Change the label on the X axis
t + ylab("New Y label")
Change the label on the Y axis
t + labs(title ="New title", x = "New x", y = "New y")
All of the above
Use scale functions to update legend labels
Legends
t + theme(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right"
t + guides(color = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)
t + scale_fill_discrete(name = "Title", labels = c("A", "B", "C"))
Set legend title and labels with a scale function.
Zooming
Without clipping (preferred)
t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20))
With clipping (removes unseen data points)
t + xlim(0, 100) + ylim(10, 20)
t + scale_x_continuous(limits = c(0, 100)) + scale_y_continuous(limits = c(0, 100))
RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com
Learn more at docs.ggplot2.org • ggplot2 0.9.3.1 • Updated: 3/15