

Python For Data Science Cheat Sheet

NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com



NumPy

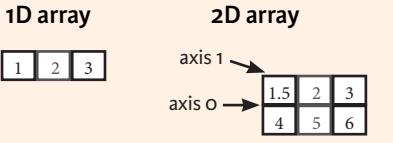
The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```



NumPy Arrays



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))
>>> np.ones((2,3,4),dtype=np.int16)
>>> d = np.arange(10,25,5)

>>> np.linspace(0,2,9)

>>> e = np.full((2,2),7)
>>> f = np.eye(2)
>>> np.random.random((2,2))
>>> np.empty((3,2))
```

Create an array of zeros
Create an array of ones
Create an array of evenly spaced values (step value)
Create an array of evenly spaced values (number of samples)
Create a constant array
Create a 2x2 identity matrix
Create an array with random values
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

<code>>>> np.int64</code>	Signed 64-bit integer types
<code>>>> np.float32</code>	Standard double-precision floating point
<code>>>> np.complex</code>	Complex numbers represented by 128 floats
<code>>>> np.bool</code>	Boolean type storing TRUE and FALSE values
<code>>>> np.object</code>	Python object type
<code>>>> np.string_</code>	Fixed-length string type
<code>>>> np_unicode_</code>	Fixed-length unicode type

Inspecting Your Array

```
>>> a.shape
>>> len(a)
>>> a.ndim
>>> a.size
>>> a.dtype
>>> a.dtype.name
>>> a.astype(int)
```

Array dimensions
Length of array
Number of array dimensions
Number of array elements
Data type of array elements
Name of data type
Convert an array to a different type

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b
      array([[-0.5,  0. ,  0. ],
             [-3. , -3. , -3. ]])
>>> np.subtract(a,b)
>>> b + a
      array([[ 2.5,  4. ,  6. ],
             [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)
>>> a / b
      array([[ 0.66666667,  1.        ,  1.        ],
             [ 0.25,  0.4,  0.5        ]])
>>> np.divide(a,b)
>>> a * b
      array([[ 1.5,  4. ,  9. ],
             [ 4. , 10. , 18. ]])
>>> np.multiply(a,b)
>>> np.exp(b)
>>> np.sqrt(b)
>>> np.sin(a)
>>> np.cos(b)
>>> np.log(a)
>>> e.dot(f)
      array([[ 7.,  7.],
             [ 7.,  7.]])
```

Subtraction
Addition
Addition
Division
Division
Multiplication
Multiplication
Exponentiation
Square root
Print sines of an array
Element-wise cosine
Element-wise natural logarithm
Dot product

Comparison

```
>>> a == b
      array([[False,  True,  True],
             [False, False, False]], dtype=bool)
>>> a < 2
      array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)
```

Element-wise comparison
Element-wise comparison
Array-wise comparison

Aggregate Functions

```
>>> a.sum()
>>> a.min()
>>> b.max(axis=0)
>>> b.cumsum(axis=1)
>>> a.mean()
>>> b.median()
>>> a.correlcoef()
>>> np.std(b)
```

Array-wise sum
Array-wise minimum value
Maximum value of an array row
Cumulative sum of the elements
Mean
Median
Correlation coefficient
Standard deviation

Copying Arrays

```
>>> h = a.view()
>>> np.copy(a)
>>> h = a.copy()
```

Create a view of the array with the same data
Create a copy of the array
Create a deep copy of the array

Sorting Arrays

```
>>> a.sort()
>>> c.sort(axis=0)
```

Sort an array
Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Subsetting

```
>>> a[2]
      3
>>> b[1,2]
      6.0
```

Select the element at the 2nd index
Select the element at row 0 column 2 (equivalent to `b[1][2]`)

Slicing

```
>>> a[0:2]
      array([1, 2])
>>> b[0:2,1]
      array([ 2.,  5.])
>>> b[:1]
      array([[1.5, 2., 3.]])
```

Select items at index 0 and 1
Select items at rows 0 and 1 in column 1
Select all items at row 0 (equivalent to `b[0:1, :]`)
Same as `[1, :, :]`

```
>>> c[1,:]
      array([[ 3.,  2.,  1.],
             [ 4.,  5.,  6.]])
>>> a[ : :-1]
      array([3, 2, 1])
```

Reversed array `a`

Boolean Indexing

```
>>> a[a<2]
      array([1])
```

Select elements from `a` less than 2

Fancy Indexing

```
>>> b[[1, 0, 1, 0], [0, 1, 2, 0]]
      array([ 4.,  2.,  6., 1.5])
>>> b[[1, 0, 1, 0], :, [0,1,2,0]]
      array([[ 4.,  5.,  6.,  4.],
             [ 1.5,  2.,  3.,  1.5],
             [ 4.,  5.,  6.,  4.],
             [ 1.5,  2.,  3.,  1.5]])
```

Select elements `(1,0),(0,1),(1,2)` and `(0,0)`
Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)
>>> i.T
```

Permute array dimensions
Permute array dimensions

Changing Array Shape

```
>>> b.ravel()
>>> g.reshape(3,-2)
```

Flatten the array
Reshape, but don't change data

Adding/Removing Elements

```
>>> h.resize((2,6))
>>> np.append(h,g)
>>> np.insert(a, 1, 5)
>>> np.delete(a, [1])
```

Return a new array with shape `(2,6)`
Append items to an array
Insert items in an array
Delete items from an array

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)
      array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))
      array([[ 1.,  2.,  3.],
             [ 1.5,  2.,  3.],
             [ 4.,  5.,  6.]])
>>> np.r_[e,f]
>>> np.hstack((e,f))
      array([[ 7.,  7.,  1.,  0.],
             [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))
      array([[ 1, 10],
             [ 2, 15],
             [ 3, 20]])
>>> np.c_[a,d]
```

Concatenate arrays
Stack arrays vertically (row-wise)
Stack arrays vertically (row-wise)
Stack arrays horizontally (column-wise)
Create stacked column-wise arrays
Create stacked column-wise arrays
Create stacked column-wise arrays

Splitting Arrays

```
>>> np.hsplit(a,3)
      [array([1]), array([2]), array([3])]
>>> np.vsplit(c,2)
      [array([[ 1.5,  2.,  1.],
             [ 4.,  5.,  6.]]),
       array([[ 3.,  2.,  1.],
             [ 4.,  5.,  6.]])]
```

Split the array horizontally at the 3rd index
Split the array vertically at the 2nd index



Data Wrangling

with pandas

Cheat Sheet

<http://pandas.pydata.org>

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
e	2	5	8	11
		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

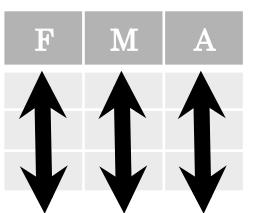
Method Chaining

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'))
```

Tidy Data – A foundation for wrangling in pandas

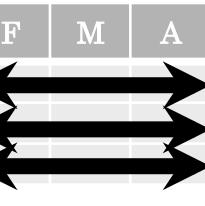
In a tidy data set:



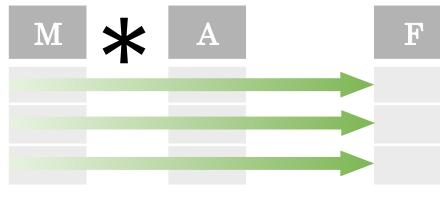
Each variable is saved in its own column

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.

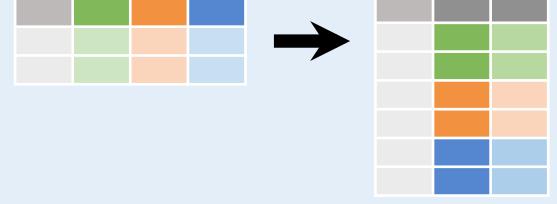
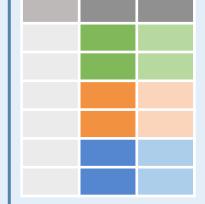
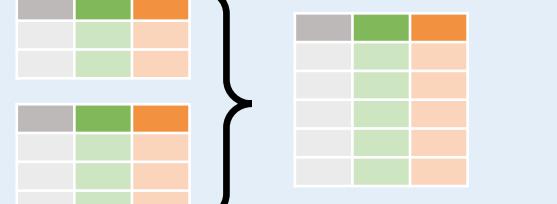
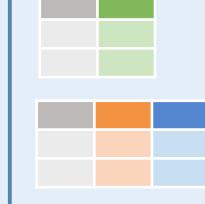


M * A



M * A

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.

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 .

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

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.

Windows

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

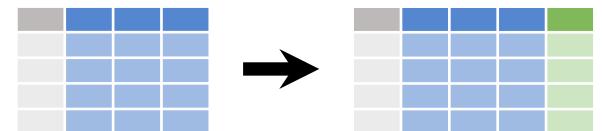
df.rolling(n)
Return a Rolling object allowing summary functions to be applied to windows of length n.

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:

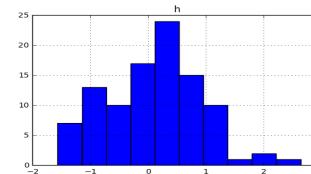
max(axis=1)	Element-wise max.
clip(lower=-10,upper=10)	Trim values at input thresholds
abs()	Absolute value.
min(axis=1)	Element-wise min.

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.

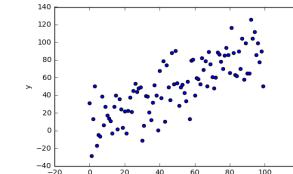
shift(1)	Copy with values shifted by 1.
rank(method='dense')	Ranks with no gaps.
rank(method='min')	Ranks. Ties get min rank.
rank(pct=True)	Ranks rescaled to interval [0, 1].
rank(method='first')	Ranks. Ties go to first value.
shift(-1)	Copy with values lagged by 1.
cumsum()	Cumulative sum.
cummax()	Cumulative max.
cummin()	Cumulative min.
cumprod()	Cumulative product.

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

x1	x2
A	1
B	2

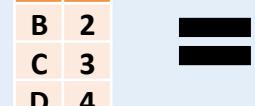
adf[adf.x1.isin(bdf.x1)]
All rows in adf that have a match in bdf.

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

zdf	
x1	x2
B	2
C	3
D	4



x1	x2
B	2
C	3
D	4

x1	x2
A	1

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

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

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).

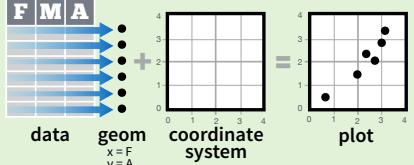
Data Visualization with ggplot2

Cheat Sheet

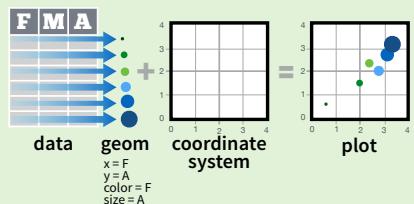


Basics

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()**

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** `a <- ggplot(mpg, aes(hwy))`
a + geom_area(stat = "bin")
`x, y, alpha, color, fill, linetype, size`
a + geom_density(kernel = "gaussian")
`x, y, alpha, color, fill, linetype, size, weight`
a + geom_dotplot()
`x, y, alpha, color, fill`
a + geom_freqpoly()
`x, y, alpha, color, linetype, size`
a + geom_histogram(binwidth = 5)
`x, y, alpha, color, fill, linetype, size, weight`
`b + geom_histogram(aes(y = ..density..))`

Discrete

- b** `b <- ggplot(mpg, aes(fl))`
b + geom_bar()
`x, alpha, color, fill, linetype, size, weight`

Graphical Primitives

- c** `c <- ggplot(map, aes(long, lat))`
c + geom_polygon(aes(group = group))
`x, y, alpha, color, fill, linetype, size`

- d** `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** `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** `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** `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** `h <- ggplot(diamonds, aes(cut, color))`
h + geom_jitter()
`x, y, alpha, color, fill, shape, size`

Three Variables

- seals\$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))**
m <- ggplot(seals, aes(long, lat))
m + geom_raster(aes(fill = z), hjust = 0.5, vjust = 0.5, interpolate = FALSE)
`x, y, alpha, fill`
m + geom_contour(aes(z = z))
`x, y, z, alpha, colour, linetype, size, weight`

Continuous Bivariate Distribution

- i** `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** `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))

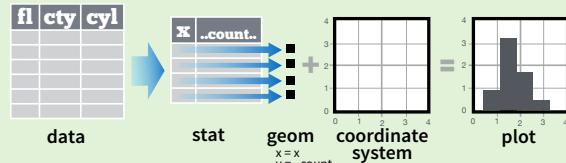
- k + geom_crossbar(fatten = 2)**
`x, y, ymax, ymin, alpha, color, fill, linetype, size`
k + geom_errorbar()
`x, ymax, ymin, alpha, color, linetype, size, width (also geom_errorbarh())`
k + geom_linerange()
`x, ymin, ymax, alpha, color, linetype, size`
k + geom_pointrange()
`x, y, ymin, ymax, alpha, 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`

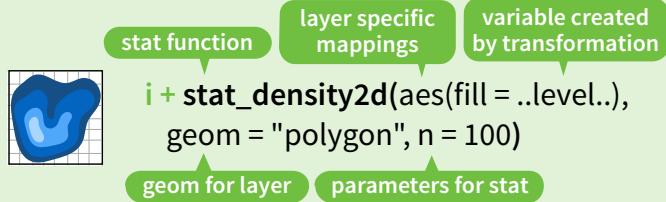
Stats - An alternative way to build a layer

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")`



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")`



`a + stat_bin(binwidth = 1, origin = 10)` 1D distributions
`x, y | ..count., ..ncount., ..density., ..ndensity..`
`a + stat_bindot(binwidth = 1, binaxis = "x")`
`x, y, | ..count., ..ncount..`
`a + stat_density(adjust = 1, kernel = "gaussian")`
`x, y, | ..count., ..density., ..scaled..`

`f + stat_bin2d(bins = 30, drop = TRUE)` 2D distributions
`x, y, fill | ..count., ..density..`
`f + stat_binhex(bins = 30)`
`x, y, fill | ..count., ..density..`
`f + stat_density2d(contour = TRUE, n = 100)`
`x, y, color, size | ..level..`

`m + stat_contour(aes(z = z))` 3 Variables
`x, y, z, order | ..level..`
`m + stat_spoke(aes(radius = z, angle = z))`
`angle, radius, x, xend, y, yend | ..x., ..xend., ..y., ..yend..`
`m + stat_summary_hex(aes(z = z), bins = 30, fun = mean)`
`x, y, z, fill | ..value..`
`m + stat_summary2d(aes(z = z), bins = 30, fun = mean)`
`x, y, z, fill | ..value..`

`g + stat_boxplot(coef = 1.5)` Comparisons
`x, y | ..lower.,..middle.,..upper.,..outliers..`
`g + stat_ydensity(adjust = 1, kernel = "gaussian", scale = "area")`
`x, y | ..density.,..scaled.,..count.,..n.,..violinwidth.,..width..`

`f + stat_ecdf(n = 40)` Functions
`x, y | ..x.,..y..`
`f + stat_quantile(quantiles = c(0.25, 0.5, 0.75), formula = y ~ log(x), method = "rq")`
`x, y | ..quantile.,..x.,..y..`
`f + stat_smooth(method = "auto", formula = y ~ x, se = TRUE, n = 80, fullrange = FALSE, level = 0.95)`
`x, y | ..se.,..x.,..y.,..ymin.,..ymax..`

`ggplot() + stat_function(aes(x = -3:3), fun = dnorm, n = 101, args = list(sd = 0.5))` General Purpose
`x | ..y..`
`f + stat_identity()`
`ggplot() + stat_qq(aes(sample = 1:100), distribution = qt, dparams = list(df = 5))`
`sample, x, y | ..x.,..y..`
`f + stat_sum()`
`x, y, size | ..size..`
`f + stat_summary(fun.data = "mean_cl_boot")`
`f + stat_unique()`

Scales

Scales control how a plot maps data values to the visual values of an aesthetic. To change the mapping, add a custom scale.



General Purpose scales

Use with any aesthetic:
alpha, color, fill, linetype, shape, size

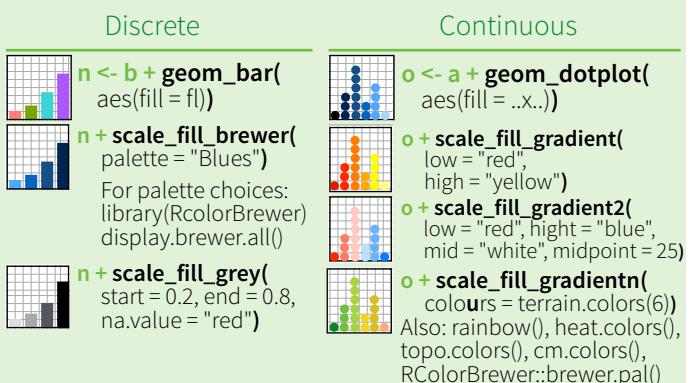
`scale_*_continuous()` - map cont' values to visual values
`scale_*_discrete()` - map discrete values to visual values
`scale_*_identity()` - use data values **as** visual values
`scale_*_manual(values = c())` - map discrete values to manually chosen visual values

X and Y location scales

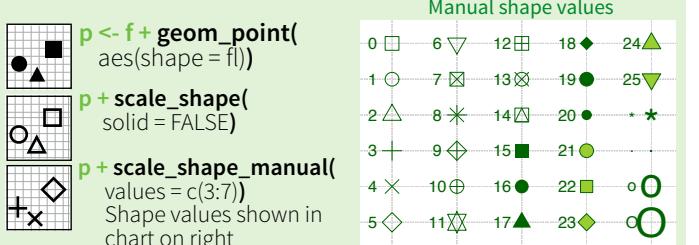
Use with x or y aesthetics (x shown here)

`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

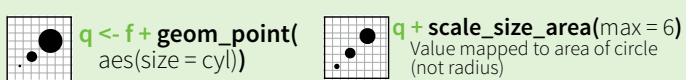
Color and fill scales



Shape scales



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

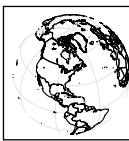
Flipped Cartesian coordinates

`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_grey()`
Grey background (default theme)

`r + theme_classic()`
White background no gridlines

`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 `labeler` to adjust facet labels

`t + facet_grid(. ~ fl, labeler = label_both)`
fl: c fl: d fl: e fl: p fl: r

`t + facet_grid(. ~ fl, labeler = label_bquote(alpha ^ .(x)))`
 $\alpha^c \quad \alpha^d \quad \alpha^e \quad \alpha^p \quad \alpha^r$

`t + facet_grid(. ~ fl, labeler = label_parsed)`
c d e p r

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))`

Python For Data Science Cheat Sheet

Seaborn

Learn Data Science **interactively** at www.DataCamp.com



Statistical Data Visualization With Seaborn

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

Make use of the following aliases to import the libraries:

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

The basic steps to creating plots with Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")
>>> sns.set_style("whitegrid")           Step 1
>>> g = sns.lmplot(x="tip",
>                  y="total_bill",
>                  data=tips,
>                  aspect=2)                         Step 2
>>> g.set_axis_labels("Tip", "Total bill(USD)") . Step 3
>>> set(xlim=(0,10), ylim=(0,100))
>>> plt.title("title")                  Step 4
>>> plt.show(g)                         Step 5
```

1 Data

Also see [Lists, NumPy & Pandas](#)

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({'x':np.arange(1,101),
>                      'y':np.random.normal(0,4,100)})
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

2 Figure Aesthetics

Seaborn styles

```
>>> sns.set()
>>> sns.set_style("whitegrid")
>>> sns.set_style("ticks",
>                 {"xtick.major.size":8,
>                  "ytick.major.size":8})
>>> sns.axes_style("whitegrid")
```

(Re)set the seaborn default
Set the matplotlib parameters
Set the matplotlib parameters

Return a dict of params or use with
with to temporarily set the style

3 Plotting With Seaborn

Axis Grids

```
>>> g = sns.FacetGrid(titanic,
>                     col="survived",
>                     row="sex")
>>> g.map(plt.hist, "age")
>>> sns.factorplot(x="pclass",
>                  y="survived",
>                  hue="sex",
>                  data=titanic)
>>> sns.lmplot(x="sepal_width",
>                 y="sepal_length",
>                 hue="species",
>                 data=iris)
```

Subplot grid for plotting conditional relationships

Draw a categorical plot onto a Facetgrid

Plot data and regression model fits across a FacetGrid

```
>>> h = sns.PairGrid(iris)
>>> h = h.map(plt.scatter)
>>> sns.pairplot(iris)
>>> i = sns.JointGrid(x="x",
>                     y="y",
>                     data=data)
>>> i = i.plot(sns.regplot,
>                      sns.distplot)
>>> sns.jointplot("sepal_length",
>                  "sepal_width",
>                  data=iris,
>                  kind='kde')
```

Subplot grid for plotting pairwise relationships
Plot pairwise bivariate distributions
Grid for bivariate plot with marginal univariate plots

Plot bivariate distribution

Categorical Plots

Scatterplot
>>> sns.stripplot(x="species",
> y="petal_length",
> data=iris)
>>> sns.swarmplot(x="species",
> y="petal_length",
> data=iris)

Bar Chart

```
>>> sns.barplot(x="sex",
>                 y="survived",
>                 hue="class",
>                 data=titanic)
```

Count Plot

```
>>> sns.countplot(x="deck",
>                  data=titanic,
>                  palette="Greens_d")
```

Point Plot

```
>>> sns.pointplot(x="class",
>                  y="survived",
>                  hue="sex",
>                  data=titanic,
>                  palette={"male":"g",
>                           "female":"m"},
>                  markers=["^", "o"],
>                  linestyles=["-", "--"])
```

Boxplot

```
>>> sns.boxplot(x="alive",
>                 y="age",
>                 hue="adult_male",
>                 data=titanic)
```

Violinplot

```
>>> sns.violinplot(x="age",
>                  y="sex",
>                  hue="survived",
>                  data=titanic)
```

Scatterplot with one categorical variable

Categorical scatterplot with non-overlapping points

Show point estimates and confidence intervals with scatterplot glyphs

Show count of observations

Show point estimates and confidence intervals as rectangular bars

Boxplot

Boxplot with wide-form data

Violin plot

Subplot grid for plotting pairwise relationships
Plot pairwise bivariate distributions
Grid for bivariate plot with marginal univariate plots

Plot bivariate distribution

Regression Plots

```
>>> sns.regplot(x="sepal_width",
>                 y="sepal_length",
>                 data=iris,
>                 ax=ax)
```

Plot data and a linear regression model fit

Distribution Plots

```
>>> plot = sns.distplot(data.y,
>                       kde=False,
>                       color="b")
```

Plot univariate distribution

Matrix Plots

```
>>> sns.heatmap(uniform_data, vmin=0, vmax=1)
```

Heatmap

4 Further Customizations

Also see [Matplotlib](#)

Axisgrid Objects

```
>>> g.despine(left=True)
>>> g.set_ylabels("Survived")
>>> g.set_xticklabels(rotation=45)
>>> g.set_axis_labels("Survived",
>                     "Sex")
```

Remove left spine
Set the labels of the y-axis
Set the tick labels for x
Set the axis labels

Set the limit and ticks of the x-and y-axis

Plot

```
>>> plt.title("A Title")
>>> plt.ylabel("Survived")
>>> plt.xlabel("Sex")
>>> plt.ylim(0,100)
>>> plt.xlim(0,10)
>>> plt.setp(ax, yticks=[0,5])
>>> plt.tight_layout()
```

Add plot title
Adjust the label of the y-axis
Adjust the label of the x-axis
Adjust the limits of the y-axis
Adjust the limits of the x-axis
Adjust a plot property
Adjust subplot params

5 Show or Save Plot

Also see [Matplotlib](#)

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

Show the plot
Save the plot as a figure
Save transparent figure

Close & Clear

```
>>> plt.cla()
>>> plt.clf()
>>> plt.close()
```

Clear an axis
Clear an entire figure
Close a window



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.



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)
```

3 Plotting Routines

1D Data

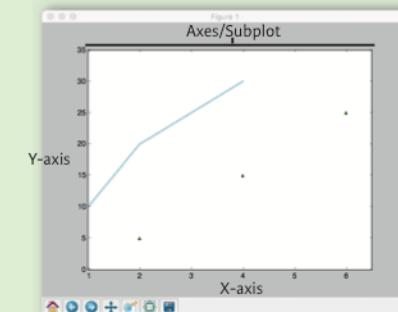
```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x, y)
>>> ax.scatter(x, y)
>>> axes[0,0].bar([1,2,3],[3,4,5])
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])
>>> axes[1,1].axhline(0.45)
>>> axes[0,1].axvline(0.65)
>>> ax.fill(x,y,color='blue')
>>> ax.fill_between(x,y,color='yellow')
```

2D Data or Images

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

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]
>>> 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>
```

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")
```

Line Styles

```
>>> 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"),)
```

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)
>>> axes[1,1].quiver(y,z)
>>> axes[0,1].streamplot(X,Y,U,V)
```

Mathtext

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

Limits, Legends & Layouts

```
>>> ax.margins(x=0.0,y=0.1)
>>> ax.axis('equal')
>>> ax.set(xlim=[0,10.5], ylim=[-1.5,1.5])
>>> ax.set_xlim(0,10.5)
```

Legends

```
>>> ax.set(title='An Example Axes',
>>>         ylabel='Y-Axis',
>>>         xlabel='X-Axis')
>>> ax.legend(loc='best')
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),
>>>               ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',
>>>                 direction='inout',
>>>                 length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,
>>>                       hspace=0.3,
>>>                       left=0.125,
>>>                       right=0.9,
>>>                       top=0.9,
>>>                       bottom=0.1)
>>> fig.tight_layout()
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)
>>> ax1.spines['bottom'].set_position(('outward',10))
```

Add padding to a plot
Set the aspect ratio of the plot to 1
Set limits for x-and y-axis
Set limits for x-axis

Set a title and x-and y-axis labels

No overlapping plot elements

Manually set x-ticks

Make y-ticks longer and go in and out

Adjust the spacing between subplots

Fit subplot(s) in to the figure area

Make the top axis line for a plot invisible

Move the bottom axis line outward

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()
```

```
>>> plt.clf()
```

```
>>> plt.close()
```

Clear an axis
Clear the entire figure
Close a window



TensorFlow Cheat Sheet



About

TensorFlow

TensorFlow™ is an open source software library for numerical computation using data flow graphs. TensorFlow was originally developed for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

Skflow

Skflow provides a set of high level model classes that you can use to easily integrate with your existing Sklearn pipeline code. Skflow is a simplified interface for TensorFlow, to get people started on predictive analytics and data mining. Skflow has been merged into TensorFlow since version 0.8 and now called TensorFlow Learn.

Keras

Keras is a minimalist, highly modular neural networks library, written in Python and capable of running on top of either TensorFlow or Theano

Installation

How to install new package in Python:

```
pip install <package-name>
```

Example: pip install requests

How to install tensorflow?

```
device = 'cpu/gpu'
```

```
python_version = 'cp27/cp34'
```

```
sudo pip install
```

```
https://storage.googleapis.com/tensorflow/linux/$device/tensorflow-0.8.0-$python_version-none-linux_x86_64.whl
```

How to install Skflow

```
pip install sklearn
```

How to install Keras

```
pip install keras
```

update `~/keras/keras.json` - replace "theano" by "tensorflow"

Helpers

Python helper

Important functions

`type(object)`

Get object type

`help(object)`

Get help for object (list of available methods, attributes, signatures and so on)

`dir(object)`

Get list of object attributes

(fields, functions)

`str(object)`

Transform an object to string

`object?`

Shows documentation about the object

`globals()`

Return the dictionary containing the current scope's global variables.

`locals()`

Update and return a dictionary containing the current scope's local variables.

`id(object)`

Return the identity of an object. This is guaranteed to be unique among simultaneously existing objects.

```
import __builtin__
```

```
dir(__builtin__)
```

Other built-in functions

TensorFlow

Main classes

```
tf.Graph()
```

```
tf.Operation()
```

```
tf.Tensor()
```

```
tf.Session()
```

Some useful functions

```
tf.get_default_session()
```

```
tf.get_default_graph()
```

```
tf.reset_default_graph()
```

```
ops.reset_default_graph()
```

```
tf.device('/cpu:0')
```

```
tf.name_scope(value)
```

```
tf.convert_to_tensor(value)
```

TensorFlow Optimizers

```
GradientDescentOptimizer
```

```
AdadeltaOptimizer
```

```
AdagradOptimizer
```

```
MomentumOptimizer
```

```
AdamOptimizer
```

```
FtrlOptimizer
```

```
RMSPropOptimizer
```

Reduction

```
reduce_sum
```

```
reduce_prod
```

```
reduce_min
```

```
reduce_max
```

```
reduce_mean
```

```
reduce_all
```

```
reduce_any
```

```
accumulate_n
```

Activation functions

```
tf.nn?
```

```
relu
```

```
relu6
```

```
elu
```

```
softplus
```

```
softsign
```

```
dropout
```

```
bias_add
```

```
sigmoid
```

```
tanh
```

```
sigmoid_cross_entropy_with_logits
```

```
softmax
```

```
log_softmax
```

```
softmax_cross_entropy_with_logits
```

```
sparse_softmax_cross_entropy_with_logits
```

```
weighted_cross_entropy_with_logits
```

etc.

Skflow

Main classes

```
TensorFlowClassifier
```

```
TensorFlowRegressor
```

```
TensorFlowDNNClassifier
```

```
TensorFlowDNNRegressor
```

```
TensorFlowLinearClassifier
```

```
TensorFlowLinearRegressor
```

```
TensorFlowRNNClassifier
```

```
TensorFlowRNNRegressor
```

TensorFlowEstimator

Each classifier and regressor have following fields

`n_classes=0` (Regressor), `n_classes` are expected to be input (Classifiers)

`batch_size=32`,

`steps=200`, // except

`TensorFlowRNNClassifier` - there is 50

`optimizer='Adagrad'`,

`learning_rate=0.1`,

`class_weight=None`,

`clip_gradients=5.0`,

`continue_training=False`,

`config=None`,

`verbose=1`.

Each class has a method fit

```
fit(X, y, monitor=None, logdir=None)
```

`X`: matrix or tensor of shape `[n_samples, n_features...]`. Can be iterator that returns arrays of features. The training input samples for fitting the model.

`y`: vector or matrix `[n_samples]` or

`[n_samples, n_outputs]`. Can be iterator that returns array of targets. The training target values (class labels in classification, real numbers in regression).

`monitor`: Monitor object to print training progress and invoke early stopping

`logdir`: the directory to save the log file that can be used for optional visualization.

`predict` (`X, axis=1, batch_size=None`)

Args:

`X`: array-like matrix, `[n_samples, n_features...]` or iterator.

`axis`: Which axis to argmax for classification.

By default axis 1 (next after batch) is used. Use 2 for sequence predictions.

`batch_size`: If test set is too big, use batch size to split it into mini batches. By default the `batch_size` member variable is used. Returns:

`y`: array of shape `[n_samples]`. The predicted classes or predicted value.

Useful links

TensorFlow API

https://www.tensorflow.org/versions/r0.9/api_docs/index.html

TensorFlow How-Tos

https://www.tensorflow.org/versions/r0.9/how_tos/index.html

Skflow github

<https://github.com/tensorflow/skflow>

Skflow API

<http://skflow.learn.org/stable/modules/classes.html#module-skflow.cluster>

Keras

<http://keras.io/>

Our github

<https://github.com/Altoros/TensorFlow-training>

TensorFlow github

<https://github.com/tensorflow/tensorflow>

Python For Data Science Cheat Sheet

Scikit-Learn

Learn Python for data science interactively at www.DataCamp.com



Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.



A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> x, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10, 5))
>>> y = np.array(['M', 'M', 'F', 'F', 'M', 'F', 'M', 'F', 'F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
...                                                    y,
...                                                    random_state=0)
```

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning

```
>>> lr.fit(X, y)
>>> knn.fit(X_train, y_train)
>>> svc.fit(X_train, y_train)
```

Unsupervised Learning

```
>>> k_means.fit(X_train)
>>> pca_model = pca.fit_transform(X_train)
```

Fit the model to the data

Fit the model to the data
Fit to data, then transform it

Prediction

Supervised Estimators

```
>>> y_pred = svc.predict(np.random.random((2,5)))
>>> y_pred = lr.predict(X_test)
>>> y_pred = knn.predict_proba(X_test)
```

Unsupervised Estimators

```
>>> y_pred = k_means.predict(X_test)
```

Predict labels
Predict labels
Estimate probability of a label
Predict labels in clustering algos

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```

Estimator score method

Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```

Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R² Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
...            "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
...                      param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
...            "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
...                               param_distributions=params,
...                               cv=4,
...                               n_iter=8,
...                               random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```



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 poly1d
>>> p = poly1d([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	Unwrap
>>> np.unwrap(g)	Create an array of evenly spaced values (log scale)
>>> np.logspace(0,10,3)	Return values from a list of arrays depending on conditions
>>> np.select([c<4],[c*2])	Factorial
>>> misc.factorial(a)	Combine N things taken at k time
>>> misc.comb(10,3,exact=True)	Weights for Np-point central derivative
>>> misc.central_diff_weights(3)	Find the n-th derivative of a function at a point
>>> misc.derivative(myfunc,1.0)	

Linear Algebra

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

Also see NumPy

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

Transposition

```
>>> A.T
>>> A.H
```

Trace

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

Norm

```
>>> linalg.norm(A)
>>> linalg.norm(A,1)
>>> linalg.norm(A,np.inf)
```

Rank

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

Determinant

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

Solving linear problems

```
>>> linalg.solve(A,b)
>>> E = np.mat(a).T
>>> linalg.lstsq(F,E)
```

Generalized inverse

```
>>> linalg.pinv(C)
```

```
>>> linalg.pinv2(C)
```

Creating Sparse Matrices

```
>>> F = np.eye(3, k=1)
>>> G = np.mat(np.identity(2))
>>> C[C > 0.5] = 0
>>> H = sparse.csr_matrix(C)
>>> I = sparse.csc_matrix(D)
>>> J = sparse.dok_matrix(A)
>>> E.todense()
>>> sparse.isspmatrix_csc(A)
```

Inverse

Transpose

Transpose matrix
Conjugate transposition

Trace

Frobenius norm
L1 norm (max column sum)
Linf norm (max row sum)

Matrix rank

Determinant

Solver for dense matrices
Solver for dense matrices
Least-squares solution to linear matrix equation

Compute the pseudo-inverse of a matrix (least-squares solver)
Compute the pseudo-inverse of a matrix (SVD)

Matrix Functions

Addition

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

Subtraction

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

Division

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

Multiplication

```
>>> A @ D
```

```
>>> np.multiply(D,A)
>>> np.dot(A,D)
>>> np.vdot(A,D)
>>> np.inner(A,D)
>>> np.outer(A,D)
>>> np.tensordot(A,D)
>>> np.kron(A,D)
```

Exponential Functions

```
>>> linalg.expm(A)
>>> linalg.expm2(A)
>>> linalg.expm3(D)
```

Logarithm Function

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

Trigonometric Functions

```
>>> linalg.sinm(D)
>>> linalg.cosm(D)
>>> linalg.tanm(A)
```

Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(D)
>>> linalg.coshm(D)
>>> linalg.tanhm(A)
```

Matrix Sign Function

```
>>> np.signm(A)
```

Matrix Square Root

```
>>> linalg.sqrtm(A)
```

Arbitrary Functions

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

Decompositions

Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(A)
>>> 11, 12 = la
>>> v[:,0]
>>> v[:,1]
>>> linalg.eigvals(A)
```

Singular Value Decomposition

```
>>> U,s,Vh = linalg.svd(B)
>>> M,N = B.shape
>>> Sig = linalg.diagsvd(s,M,N)
```

LU Decomposition

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

Sparse Matrix Decompositions

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

Asking For Help

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

DataCamp

Learn Python for Data Science [Interactively](#)



Python For Data Science Cheat Sheet

PySpark Basics

Learn Python for data science **Interactively** at www.DataCamp.com



Spark

PySpark is the Spark Python API that exposes the Spark programming model to Python



Initializing Spark

SparkContext

```
>>> from pyspark import SparkContext  
>>> sc = SparkContext(master = 'local[2]')
```

Inspect SparkContext

>>> sc.version	Retrieve SparkContext version
>>> sc.pythonVer	Retrieve Python version
>>> sc.master	Master URL to connect to
>>> str(sc.sparkHome)	Path where Spark is installed on worker nodes
>>> str(sc.sparkUser())	Retrieve name of the Spark User running SparkContext
>>> sc.appName	Return application name
>>> sc.applicationId	Retrieve application ID
>>> sc.defaultParallelism	Return default level of parallelism
>>> sc.defaultMinPartitions	Default minimum number of partitions for RDDs

Configuration

```
>>> from pyspark import SparkConf, SparkContext  
>>> conf = (SparkConf()  
          .setMaster("local")  
          .setAppName("My app")  
          .set("spark.executor.memory", "1g"))  
>>> sc = SparkContext(conf = conf)
```

Using The Shell

In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.

```
$ ./bin/spark-shell --master local[2]  
$ ./bin/pyspark --master local[4] --py-files code.py
```

Set which master the context connects to with the `--master` argument, and add Python .zip, .egg or .py files to the runtime path by passing a comma-separated list to `--py-files`.

Loading Data

Parallelized Collections

```
>>> rdd = sc.parallelize([('a',7),('a',2),('b',2)])  
>>> rdd2 = sc.parallelize([('a',2),('d',1),('b',1)])  
>>> rdd3 = sc.parallelize(range(100))  
>>> rdd4 = sc.parallelize([('a',[ "x", "y", "z" ]),  
                         ("b",["p", "r"])] )
```

External Data

Read either one text file from HDFS, a local file system or or any Hadoop-supported file system URI with `textFile()`, or read in a directory of text files with `wholeTextFiles()`.

```
>>> textFile = sc.textFile("./my/directory/*.txt")  
>>> textFile2 = sc.wholeTextFiles("./my/directory/")
```

Retrieving RDD Information

Basic Information

```
>>> rdd.getNumPartitions()  
>>> rdd.count()  
3  
>>> rdd.countByKey()  
defaultdict(<type 'int'>,{'a':2,'b':1})  
>>> rdd.countByValue()  
defaultdict(<type 'int'>,{'b':2,('a',2):1,('a',7):1})  
>>> rdd.collectAsMap()  
{'a': 2, 'b': 2}  
>>> rdd3.sum()  
4950  
>>> sc.parallelize([]).isEmpty()  
True
```

List the number of partitions
Count RDD instances
Count RDD instances by key
Count RDD instances by value
Return (key,value) pairs as a dictionary
Sum of RDD elements
Check whether RDD is empty

Summary

```
>>> rdd3.max()  
99  
>>> rdd3.min()  
0  
>>> rdd3.mean()  
49.5  
>>> rdd3.stdev()  
28.86607004772218  
>>> rdd3.variance()  
833.25  
>>> rdd3.histogram(3)  
([0,33,66,99],[33,33,34])  
>>> rdd3.stats()
```

Maximum value of RDD elements
Minimum value of RDD elements
Mean value of RDD elements
Standard deviation of RDD elements
Compute variance of RDD elements
Compute histogram by bins
Summary statistics (count, mean, stdev, max & min)

Applying Functions

```
>>> rdd.map(lambda x: x+(x[1],x[0]))  
     .collect()  
[('a',7,7,'a'), ('a',2,2,'a'), ('b',2,2,'b')]  
>>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))  
  
>>> rdd5.collect()  
[('a',7,7,'a','a',2,2,'a','b',2,2,'b')]  
>>> rdd4.flatMapValues(lambda x: x)  
     .collect()  
[('a','x'),('a','y'),('a','z'),('b','p'),('b','r')]
```

Apply a function to each RDD element
Apply a function to each RDD element and flatten the result
Apply a flatMap function to each (key,value) pair of `rdd4` without changing the keys

Selecting Data

Getting

```
>>> rdd.collect()  
[('a', 7), ('a', 2), ('b', 2)]  
>>> rdd.take(2)  
[('a', 7), ('a', 2)]  
>>> rdd.first()  
('a', 7)  
>>> rdd.top(2)  
[('b', 2), ('a', 7)]
```

Sampling

```
>>> rdd3.sample(False, 0.15, 81).collect()  
[3,4,27,31,40,41,42,43,60,76,79,80,86,97]
```

Filtering

```
>>> rdd.filter(lambda x: "a" in x)  
     .collect()  
[('a',7),('a',2)]  
>>> rdd5.distinct().collect()  
[('a',2),('b',7)]  
>>> rdd.keys().collect()  
['a', 'a', 'b']
```

Return a list with all RDD elements
Take first 2 RDD elements
Take first RDD element
Take top 2 RDD elements
Return sampled subset of `rdd3`
Filter the RDD
Return distinct RDD values
Return (key,value) RDD's keys

Iterating

```
>>> def g(x): print(x)  
>>> rdd.foreach(g)  
('a', 7)  
('b', 2)  
('a', 2)
```

Apply a function to all RDD elements

Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x,y : x+y)  
     .collect()  
[('a',9),('b',2)]  
>>> rdd.reduce(lambda a, b: a + b)  
('a',7,'a',2,'b',2)
```

Merge the `rdd` values for each key

Merge the `rdd` values

Return RDD of grouped values

Group `rdd` by key

Grouping by

```
>>> rdd3.groupBy(lambda x: x % 2)  
     .mapValues(list)  
     .collect()  
>>> rdd.groupByKey()  
     .mapValues(list)  
     .collect()  
[('a',[7,2]),('b',[2])]
```

Aggregate RDD elements of each partition and then the results
Aggregate values of each RDD key

Aggregating

```
>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))  
>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))  
>>> rdd3.aggregate((0,0),seqOp,combOp)  
(4950,100)  
>>> rdd.aggregateByKey((0,0),seqOp,combOp)  
     .collect()  
[('a',(9,2)), ('b',(2,1))]  
>>> rdd3.fold(0,add)  
4950  
>>> rdd.foldByKey(0, add)  
     .collect()  
[('a',(9,2))]  
>>> rdd3.keyBy(lambda x: x+x)  
     .collect()
```

Aggregate the elements of each partition, and then the results
Merge the values for each key

Create tuples of RDD elements by applying a function

Mathematical Operations

```
>>> rdd.subtract(rdd2)  
     .collect()  
[('b',2),('a',7)]  
>>> rdd2.subtractByKey(rdd)  
     .collect()  
[('d',1)]  
>>> rdd.cartesian(rdd2).collect()
```

Return each `rdd` value not contained in `rdd2`

Return each (key,value) pair of `rdd2` with no matching key in `rdd`

Return the Cartesian product of `rdd` and `rdd2`

Sort

```
>>> rdd2.sortBy(lambda x: x[1])  
     .collect()  
[('d',1),('b',1),('a',2)]  
>>> rdd2.sortByKey()  
     .collect()  
[('a',2),('b',1),('d',1)]
```

Sort RDD by given function

Sort (key, value) RDD by key

Repartitioning

```
>>> rdd.repartition(4)  
>>> rdd.coalesce(1)
```

New RDD with 4 partitions
Decrease the number of partitions in the RDD to 1

Saving

```
>>> rdd.saveAsTextFile("rdd.txt")  
>>> rdd.saveAsHadoopFile("hdfs://namenodehost/parent/child",  
                           'org.apache.hadoop.mapred.TextOutputFormat')
```

Stopping SparkContext

```
>>> sc.stop()
```

Execution

```
$ ./bin/spark-submit examples/src/main/python/pi.py
```



Python For Data Science Cheat Sheet

PySpark - SQL Basics

Learn Python for data science interactively at www.DataCamp.com



PySpark & Spark SQL

Spark SQL is Apache Spark's module for working with structured data.



Initializing SparkSession

A SparkSession can be used to create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files.

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

Creating DataFrames

From RDDs

```
>>> from pyspark.sql.types import *
Infer Schema
>>> sc = spark.sparkContext
>>> lines = sc.textFile("people.txt")
>>> parts = lines.map(lambda l: l.split(","))
>>> people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))
>>> peopledf = spark.createDataFrame(people)
Specify Schema
>>> people = parts.map(lambda p: Row(name=p[0],
                                     age=int(p[1].strip())))
>>> schemaString = "name age"
>>> fields = [StructField(field_name, StringType(), True) for
field_name in schemaString.split()]
>>> schema = StructType(fields)
>>> spark.createDataFrame(people, schema).show()
+-----+
| name|age|
+-----+
|  Mine| 28|
|  Filip| 29|
| Jonathan| 30|
+-----+
```

From Spark Data Sources

JSON

```
>>> df = spark.read.json("customer.json")
>>> df.show()
+-----+-----+-----+-----+
| address|age|firstName|lastName|  phoneNumber|
+-----+-----+-----+-----+
|[New York,10021,N...| 25|  John|  Smith|[212 555-1234,ho...
|[New York,10021,N...| 21|  Jane|  Doe|[322 888-1234,ho...
+-----+-----+-----+-----+
>>> df2 = spark.read.load("people.json", format="json")
Parquet files
>>> df3 = spark.read.load("users.parquet")
TXT files
>>> df4 = spark.read.text("people.txt")
```

Inspect Data

```
>>> df.dtypes
Return df column names and data types
>>> df.show()
Return the content of df
>>> df.head()
Return first n rows
>>> df.first()
Return first row
>>> df.take(2)
Return the first n rows
>>> df.schema
Return the schema of df
```

Duplicate Values

```
>>> df = df.dropDuplicates()
```

Queries

```
>>> from pyspark.sql import functions as F
Select
>>> df.select("firstName").show()
>>> df.select("firstName", "lastName") \
    .show()
>>> df.select("firstName",
             "age",
             explode("phoneNumber") \
             .alias("contactInfo")) \
    .select("contactInfo.type",
           "firstName",
           "age") \
    .show()
>>> df.select(df["firstName"], df["age"] + 1) \
    .show()
>>> df.select(df['age'] > 24).show()
When
>>> df.select("firstName",
             F.when(df.age > 30, 1) \
             .otherwise(0)) \
    .show()
>>> df[df.firstName.isin("Jane", "Boris")] \
    .collect()
Like
>>> df.select("firstName",
             df.lastName.like("Smith")) \
    .show()
Startswith - Endswith
>>> df.select("firstName",
             df.lastName \
             .startswith("Sm")) \
    .show()
>>> df.select(df.lastName.endswith("th")) \
    .show()
Substring
>>> df.select(df.firstName.substr(1, 3) \
             .alias("name")) \
    .collect()
Between
>>> df.select(df.age.between(22, 24)) \
    .show()
```

Show all entries in firstName column
Show all entries in firstName, age and type
Show all entries in firstName and age, add 1 to the entries of age
Show all entries where age >24
Show firstName and 0 or 1 depending on age >30
Show firstName if in the given options
Show firstName, and lastName is TRUE if lastName is like Smith
Show firstName, and TRUE if lastName starts with Sm
Show last names ending in th
Return substrings of firstName
Show age: values are TRUE if between 22 and 24

Add, Update & Remove Columns

Adding Columns

```
>>> df = df.withColumn('city', df.address.city) \
    .withColumn('postalCode', df.address.postalCode) \
    .withColumn('state', df.address.state) \
    .withColumn('streetAddress', df.address.streetAddress) \
    .withColumn('telephoneNumber',
               explode(df.phoneNumber.number)) \
    .withColumn('telephoneType',
               explode(df.phoneNumber.type))
```

Updating Columns

```
>>> df = df.withColumnRenamed('telephoneNumber', 'phoneNumber')
```

Removing Columns

```
>>> df = df.drop("address", "phoneNumber")
>>> df = df.drop(df.address).drop(df.phoneNumber)
```

GroupBy

```
>>> df.groupBy("age") \
    .count() \
    .show()
```

Group by age, count the members in the groups

Filter

```
>>> df.filter(df["age"] > 24).show()
```

Filter entries of age, only keep those records of which the values are >24

Sort

```
>>> peopledf.sort(peopledf.age.desc()).collect()
>>> df.sort("age", ascending=False).collect()
>>> df.orderBy(["age", "city"], ascending=[0, 1]) \
    .collect()
```

Missing & Replacing Values

```
>>> df.na.fill(50).show()
>>> df.na.drop().show()
>>> df.na \
    .replace(10, 20) \
    .show()
```

Replace null values
Return new df omitting rows with null values
Return new df replacing one value with another

Repartitioning

```
>>> df.repartition(10) \
    .rdd \
    .getNumPartitions()
>>> df.coalesce(1).rdd.getNumPartitions()
```

df with 10 partitions
df with 1 partition

Running SQL Queries Programmatically

Registering DataFrames as Views

```
>>> peopledf.createGlobalTempView("people")
>>> df.createTempView("customer")
>>> df.createOrReplaceTempView("customer")
```

Query Views

```
>>> df5 = spark.sql("SELECT * FROM customer").show()
>>> peopledf2 = spark.sql("SELECT * FROM global_temp.people") \
    .show()
```

Output

Data Structures

```
>>> rdd1 = df.rdd
>>> df.toJSON().first()
>>> df.toPandas()
```

Convert df into an RDD
Convert df into a RDD of string
Return the contents of df as Pandas DataFrame

Write & Save to Files

```
>>> df.select("firstName", "city") \
    .write \
    .save("nameAndCity.parquet")
>>> df.select("firstName", "age") \
    .write \
    .save("namesAndAges.json", format="json")
```

Stopping SparkSession

```
>>> spark.stop()
```



Python For Data Science Cheat Sheet

PySpark - RDD Basics

Learn Python for data science **Interactively** at www.DataCamp.com



Spark

PySpark is the Spark Python API that exposes the Spark programming model to Python.



Initializing Spark

SparkContext

```
>>> from pyspark import SparkContext  
>>> sc = SparkContext(master = 'local[2]')
```

Inspect SparkContext

>>> sc.version	Retrieve SparkContext version
>>> sc.pythonVer	Retrieve Python version
>>> sc.master	Master URL to connect to
>>> str(sc.sparkHome)	Path where Spark is installed on worker nodes
>>> str(sc.sparkUser())	Retrieve name of the Spark User running SparkContext
>>> sc.appName	Return application name
>>> sc.applicationId	Retrieve application ID
>>> sc.defaultParallelism	Return default level of parallelism
>>> sc.defaultMinPartitions	Default minimum number of partitions for RDDs

Configuration

```
>>> from pyspark import SparkConf, SparkContext  
>>> conf = (SparkConf()  
          .setMaster("local")  
          .setAppName("My app")  
          .set("spark.executor.memory", "1g"))  
>>> sc = SparkContext(conf = conf)
```

Using The Shell

In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.

```
$ ./bin/spark-shell --master local[2]  
$ ./bin/pyspark --master local[4] --py-files code.py
```

Set which master the context connects to with the `--master` argument, and add Python .zip, .egg or .py files to the runtime path by passing a comma-separated list to `--py-files`.

Loading Data

Parallelized Collections

```
>>> rdd = sc.parallelize([('a',7),('a',2),('b',2)])  
>>> rdd2 = sc.parallelize([('a',2),('d',1),('b',1)])  
>>> rdd3 = sc.parallelize(range(100))  
>>> rdd4 = sc.parallelize([('a',[ "x", "y", "z" ]),  
                         ("b",[ "p", "r" ])])
```

External Data

Read either one text file from HDFS, a local file system or or any Hadoop-supported file system URI with `textFile()`, or read in a directory of text files with `wholeTextFiles()`.

```
>>> textFile = sc.textFile("./my/directory/*.txt")  
>>> textFile2 = sc.wholeTextFiles("./my/directory/")
```

Retrieving RDD Information

Basic Information

```
>>> rdd.getNumPartitions()  
>>> rdd.count()  
3  
>>> rdd.countByKey()  
defaultdict(<type 'int'>, {'a':2, 'b':1})  
>>> rdd.countByValue()  
defaultdict(<type 'int'>, {'b':2, 'a':2, 'c':1})  
>>> rdd.collectAsMap()  
{'a': 2, 'b': 1}  
>>> rdd3.sum()  
4950  
>>> sc.parallelize([]).isEmpty()  
True
```

List the number of partitions
Count RDD instances
Count RDD instances by key
Count RDD instances by value
Return (key,value) pairs as a dictionary
Sum of RDD elements
Check whether RDD is empty

Summary

```
>>> rdd3.max()  
99  
>>> rdd3.min()  
0  
>>> rdd3.mean()  
49.5  
>>> rdd3.stdev()  
28.86607004772218  
>>> rdd3.variance()  
833.25  
>>> rdd3.histogram(3)  
([0,33,66,99],[33,33,34])  
>>> rdd3.stats()
```

Maximum value of RDD elements
Minimum value of RDD elements
Mean value of RDD elements
Standard deviation of RDD elements
Compute variance of RDD elements
Compute histogram by bins
Summary statistics (count, mean, stdev, max & min)

Applying Functions

```
>>> rdd.map(lambda x: x+(x[1],x[0]))  
     .collect()  
[('a',7,7,'a'), ('a',2,2,'a'), ('b',2,2,'b')]  
>>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))  
  
>>> rdd5.collect()  
[('a',7,7,'a','a',2,2,'a','b',2,2,'b')]  
>>> rdd4.flatMapValues(lambda x: x)  
     .collect()  
[('a','x'),('a','y'),('a','z'),('b','p'),('b','r')]
```

Apply a function to each RDD element
Apply a function to each RDD element and flatten the result
Apply a flatMap function to each (key,value) pair of `rdd4` without changing the keys

Selecting Data

Getting

```
>>> rdd.collect()  
[('a', 7), ('a', 2), ('b', 2)]
```

Return a list with all RDD elements

```
>>> rdd.take(2)
```

Take first 2 RDD elements

```
[('a', 7), ('a', 2)]
```

Take first RDD element

```
>>> rdd.first()
```

Take top 2 RDD elements

Sampling

```
>>> rdd3.sample(False, 0.15, 81).collect()  
[3,4,27,31,40,41,42,43,60,76,79,80,86,97]
```

Return sampled subset of `rdd3`

Filtering

```
>>> rdd.filter(lambda x: "a" in x)  
     .collect()  
[('a',7), ('a',2)]  
>>> rdd5.distinct().collect()  
[('a',2), ('b',7)]  
>>> rdd.keys().collect()  
[('a', 'a'), ('b', 'b')]
```

Filter the RDD

Return distinct RDD values

Return (key,value) RDD's keys

Iterating

```
>>> def g(x): print(x)  
>>> rdd.foreach(g)  
('a', 7)  
('b', 2)  
('a', 2)
```

Apply a function to all RDD elements

Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x,y : x+y)  
     .collect()  
[('a',9), ('b',2)]  
>>> rdd.reduce(lambda a, b: a + b)  
('a',7, 'a',2, 'b',2)
```

Merge the rdd values for each key
Merge the rdd values

Grouping by

```
>>> rdd3.groupBy(lambda x: x % 2)  
     .mapValues(list)  
     .collect()  
>>> rdd.groupByKey()  
     .mapValues(list)  
     .collect()  
[('a',[7,2]), ('b',[2])]
```

Return RDD of grouped values
Group rdd by key

Aggregating

```
>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))  
>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))  
>>> rdd3.aggregate((0,0),seqOp,combOp)  
(4950,100)  
>>> rdd.aggregateByKey((0,0),seqOp,combOp)  
     .collect()  
[('a',(9,2)), ('b',(2,1))]  
>>> rdd3.fold(0,add)  
4950  
>>> rdd.foldByKey(0, add)  
     .collect()  
[('a',(9,2))]  
>>> rdd3.keyBy(lambda x: x+x)  
     .collect()
```

Aggregate RDD elements of each partition and then the results
Aggregate values of each RDD key

Aggregate the elements of each partition, and then the results
Merge the values for each key
Create tuples of RDD elements by applying a function

Mathematical Operations

```
>>> rdd.subtract(rdd2)  
     .collect()  
[('b',2), ('a',7)]  
>>> rdd2.subtractByKey(rdd)  
     .collect()  
[('d',1)]  
>>> rdd.cartesian(rdd2).collect()
```

Return each rdd value not contained in rdd2
Return each (key,value) pair of rdd2 with no matching key in rdd
Return the Cartesian product of rdd and rdd2

Sort

```
>>> rdd2.sortBy(lambda x: x[1])  
     .collect()  
[('d',1), ('b',1), ('a',2)]  
>>> rdd2.sortByKey()  
     .collect()  
[('a',2), ('b',1), ('d',1)]
```

Sort RDD by given function
Sort (key, value) RDD by key

Repartitioning

```
>>> rdd.repartition(4)  
>>> rdd.coalesce(1)
```

New RDD with 4 partitions
Decrease the number of partitions in the RDD to 1

Saving

```
>>> rdd.saveAsTextFile("rdd.txt")  
>>> rdd.saveAsHadoopFile("hdfs://namenodehost/parent/child",  
                           'org.apache.hadoop.mapred.TextOutputFormat')
```

Stopping SparkContext

```
>>> sc.stop()
```

Execution

```
$ ./bin/spark-submit examples/src/main/python/pi.py
```



Python For Data Science Cheat Sheet

Keras

Learn Python for data science **Interactively** at www.DataCamp.com



Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
>                  activation='relu',
>                  input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
>                  loss='binary_crossentropy',
>                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
>                           mnist,
>                           cifar10,
>                           imdb
>>> (x_train,y_train), (x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2), (x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3), (x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4), (x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data"), delimiter=",")
>>> X = data[:,0:8]
>>> y = data[:,8]
```

Preprocessing

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> y_train = to_categorical(y_train, num_classes)
>>> y_test = to_categorical(y_test, num_classes)
>>> y_train3 = to_categorical(y_train3, num_classes)
>>> y_test3 = to_categorical(y_test3, num_classes)
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
>                  input_dim=8,
>                  kernel_initializer='uniform',
>                  activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.layers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Also see NumPy & Scikit-Learn

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(x,
>                                         y,
>                                         test_size=0.33,
>                                         random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

Inspect Model

```
>>> model.output_shape
>>> model.summary()
>>> model.get_config()
>>> model.get_weights()
```

Model output shape
Model summary representation
Model configuration
List all weight tensors in the model

Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
>                  loss='binary_crossentropy',
>                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
>                  loss='categorical_crossentropy',
>                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
>                  loss='mse',
>                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
>                  optimizer='adam',
>                  metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
>               y_train4,
>               batch_size=32,
>               epochs=15,
>               verbose=1,
>               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
>                           y_test,
>                           batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4, batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
>                  optimizer=opt,
>                  metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
>               y_train4,
>               batch_size=32,
>               epochs=15,
>               validation_data=(x_test4,y_test4),
>               callbacks=[early_stopping_monitor])
```

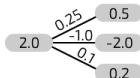


An informative chart to build
Neural Network Cells

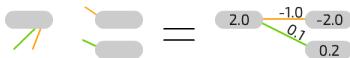
©2016 Fjodor van Veen - asimovinstitute.org

0.6 ----- 0.6

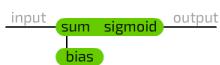
Fixed Weight
(fixed at 1)



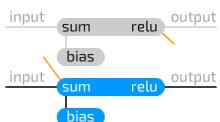
Weights



Recurrent Weights
(grouped by colour)



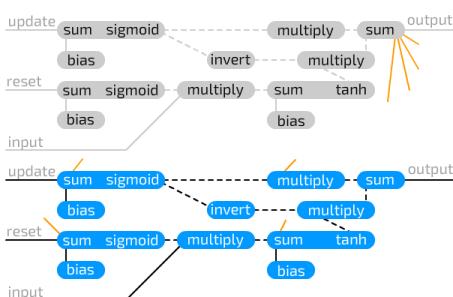
Feed Forward Cell
(basic cell)



Recurrent Cell
(previous iteration)



Recurrent Cell



GRU Cell
(previous iteration)



GRU Cell



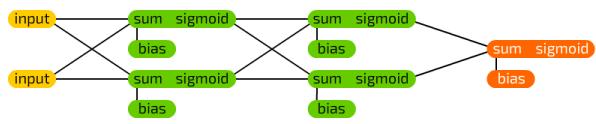
LSTM Cell
(previous iteration)



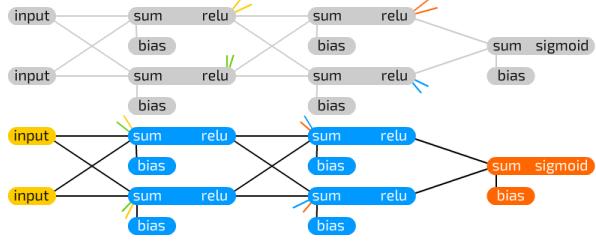
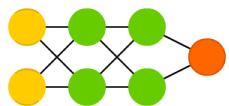
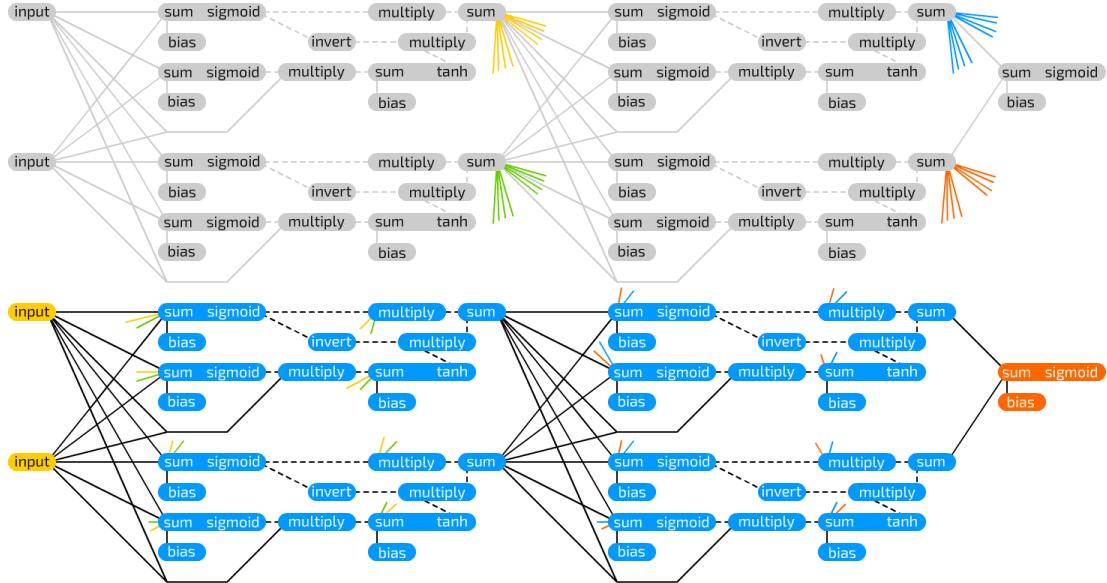
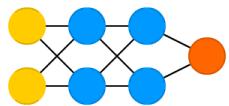
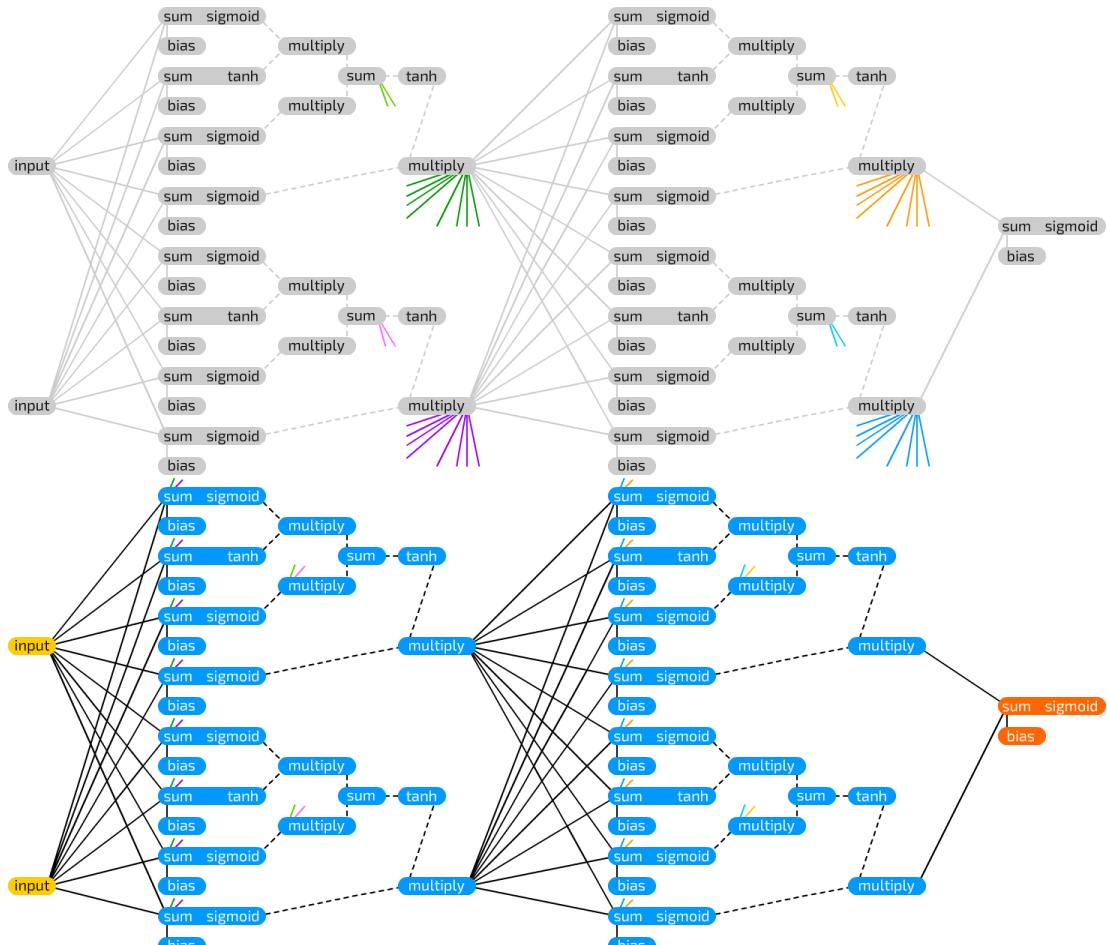
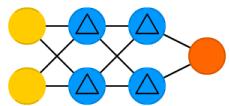
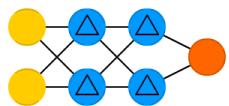
LSTM Cell

Neural Network Graphs

©2016 Fjodor van Veen - asimovinstitute.org



Deep Feed Forward Example

Deep Recurrent Example
(previous iteration)Deep GRU Example
(previous iteration)Deep LSTM Example
(previous iteration)

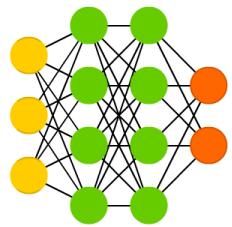
Deep LSTM Example

A mostly complete chart of
Neural Networks

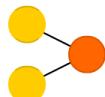
©2016 Fjodor van Veen - asimovinstitute.org

- Backfed Input Cell
- Input Cell
- △ Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- △ Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- △ Different Memory Cell
- Kernel
- Convolution or Pool

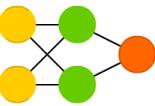
Deep Feed Forward (DFF)



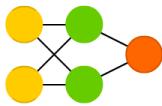
Perceptron (P)



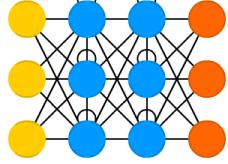
Feed Forward (FF)



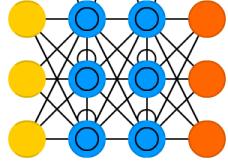
Radial Basis Network (RBF)



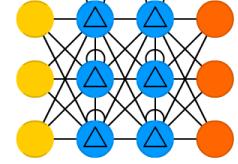
Recurrent Neural Network (RNN)



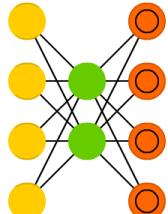
Long / Short Term Memory (LSTM)



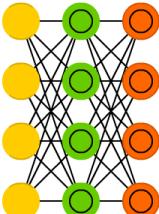
Gated Recurrent Unit (GRU)



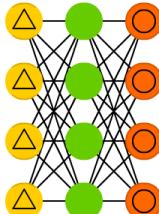
Auto Encoder (AE)



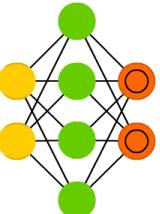
Variational AE (VAE)



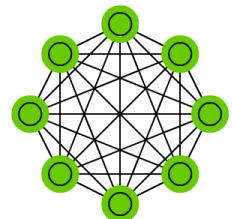
Denoising AE (DAE)



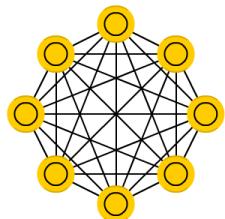
Sparse AE (SAE)



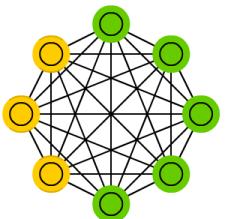
Markov Chain (MC)



Hopfield Network (HN)



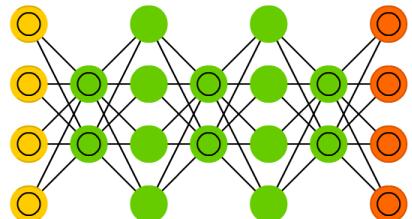
Boltzmann Machine (BM)



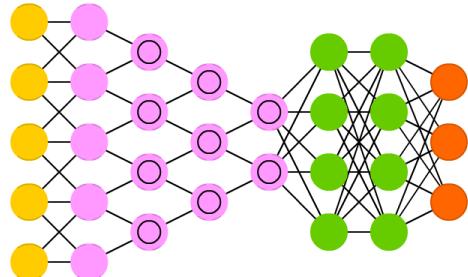
Restricted BM (RBM)



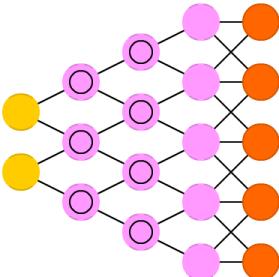
Deep Belief Network (DBN)



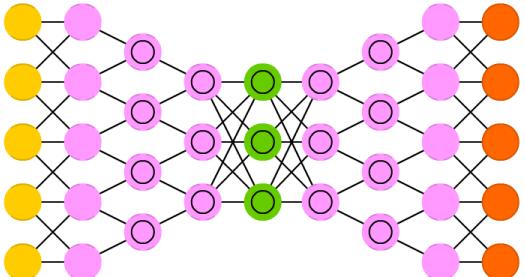
Deep Convolutional Network (DCN)



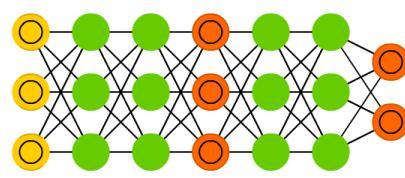
Deconvolutional Network (DN)



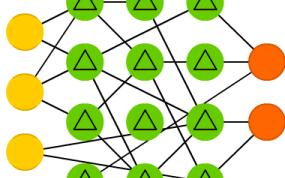
Deep Convolutional Inverse Graphics Network (DCIGN)



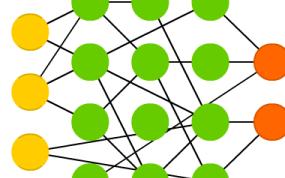
Generative Adversarial Network (GAN)



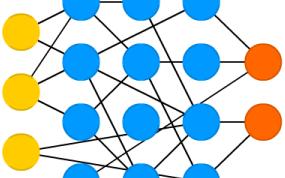
Liquid State Machine (LSM)



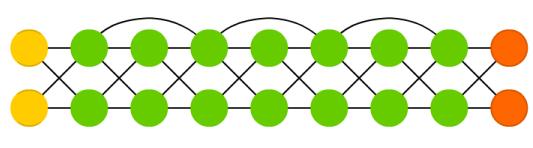
Extreme Learning Machine (ELM)



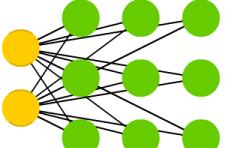
Echo State Network (ESN)



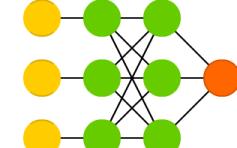
Deep Residual Network (DRN)



Kohonen Network (KN)



Support Vector Machine (SVM)



Neural Turing Machine (NTM)

