



Stefan Kojouharov [Follow](#)

Founder of Chatbots Life. I help Companies Create Great Chatbots & AI Systems and share my Insights along the way.

Jul 9, 2017 · 7 min read

Cheat Sheets for AI, Neural Networks, Machine Learning, Deep Learning & Big Data

The Most Complete List of Best AI Cheat Sheets

Over the past few months, I have been collecting AI cheat sheets. From time to time I share them with friends and colleagues and recently I have been getting asked a lot, so I decided to organize and share the entire collection. To make things more interesting and give context, I added descriptions and/or excerpts for each major topic.

This is the most complete list and the Big-O is at the very end, enjoy...

If you like this list, you can let me know [here](#).

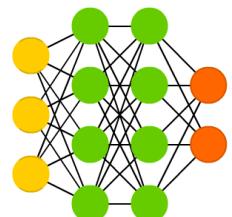
Neural Networks

- 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

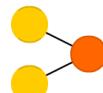
A mostly complete chart of Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

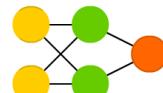
Deep Feed Forward (DFF)



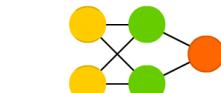
Perceptron (P)



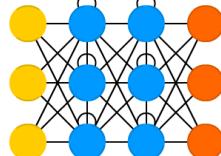
Feed Forward (FF)



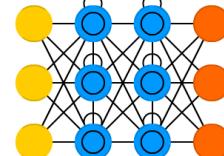
Radial Basis Network (RBF)



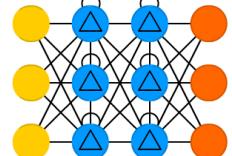
Recurrent Neural Network (RNN)



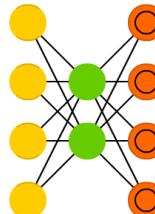
Long / Short Term Memory (LSTM)



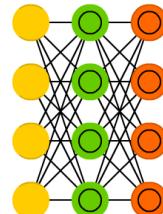
Gated Recurrent Unit (GRU)



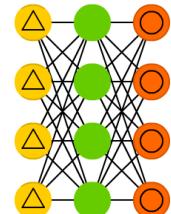
Auto Encoder (AE)



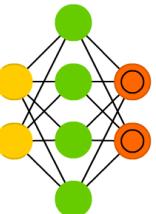
Variational AE (VAE)



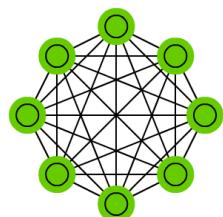
Denoising AE (DAE)



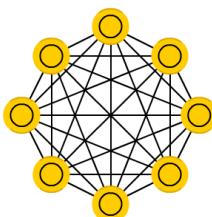
Sparse AE (SAE)



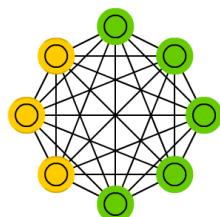
Markov Chain (MC)



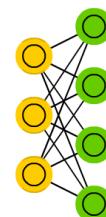
Hopfield Network (HN)



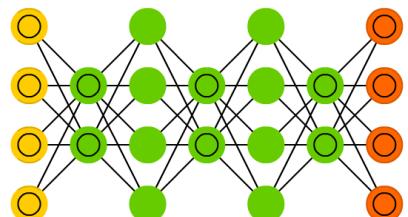
Boltzmann Machine (BM)



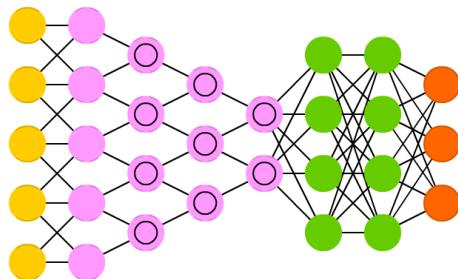
Restricted BM (RBM)



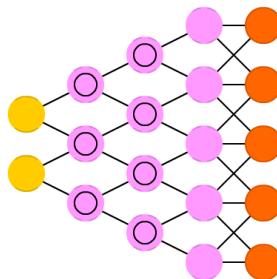
Deep Belief Network (DBN)



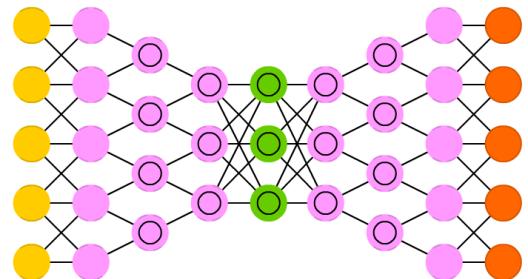
Deep Convolutional Network (DCN)



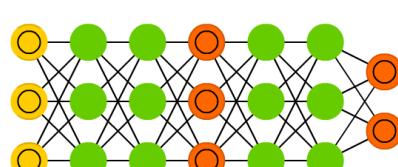
Deconvolutional Network (DN)



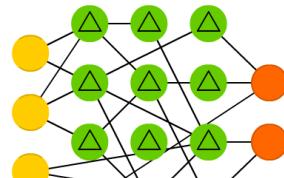
Deep Convolutional Inverse Graphics Network (DCIGN)



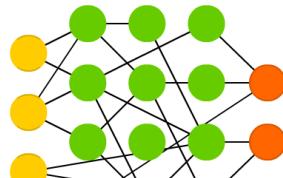
Generative Adversarial Network (GAN)



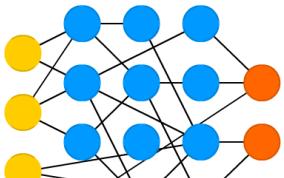
Liquid State Machine (LSM)

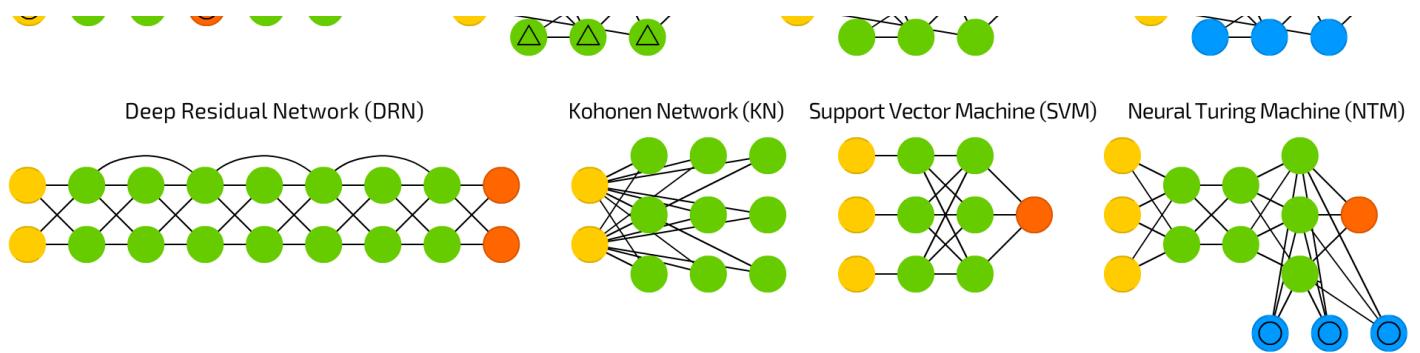


Extreme Learning Machine (ELM)



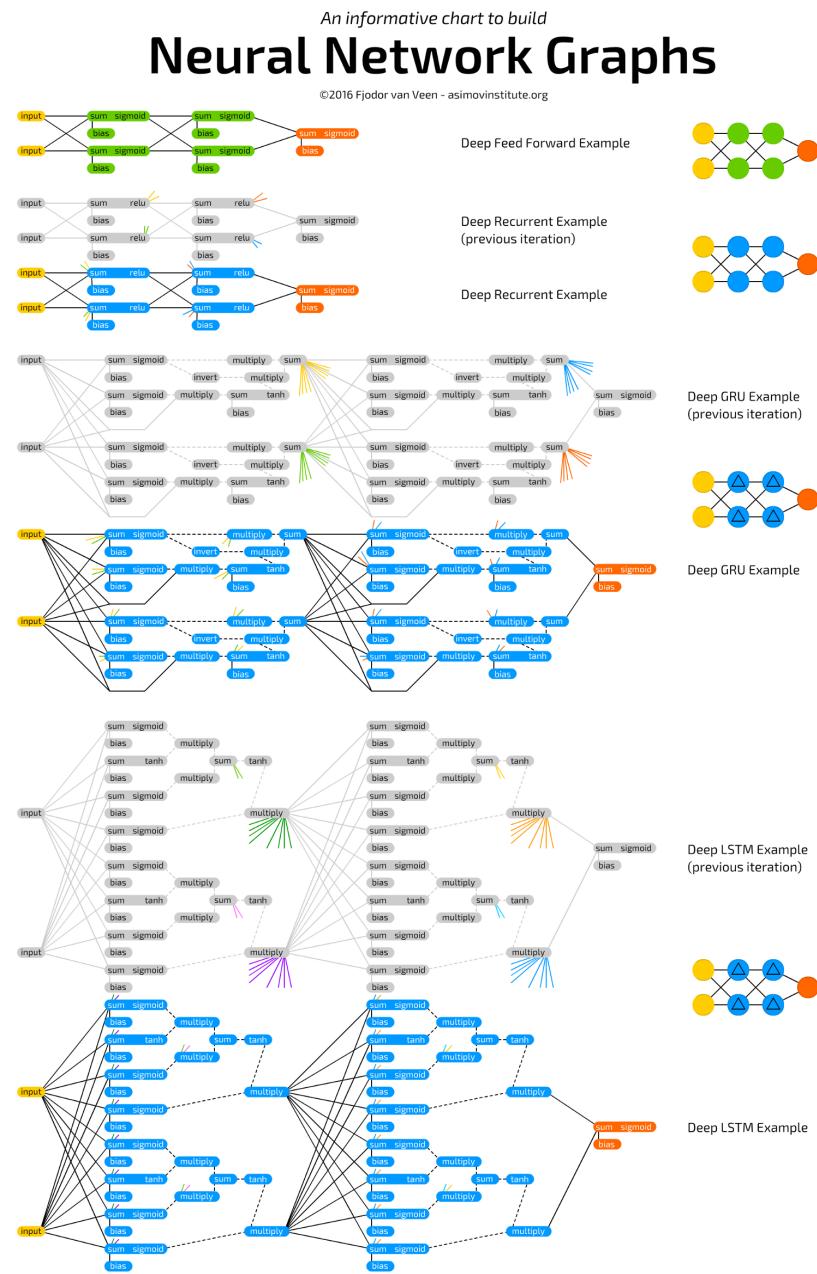
Echo State Network (ESN)





Neural Networks Cheat Sheet

Neural Networks Graphs



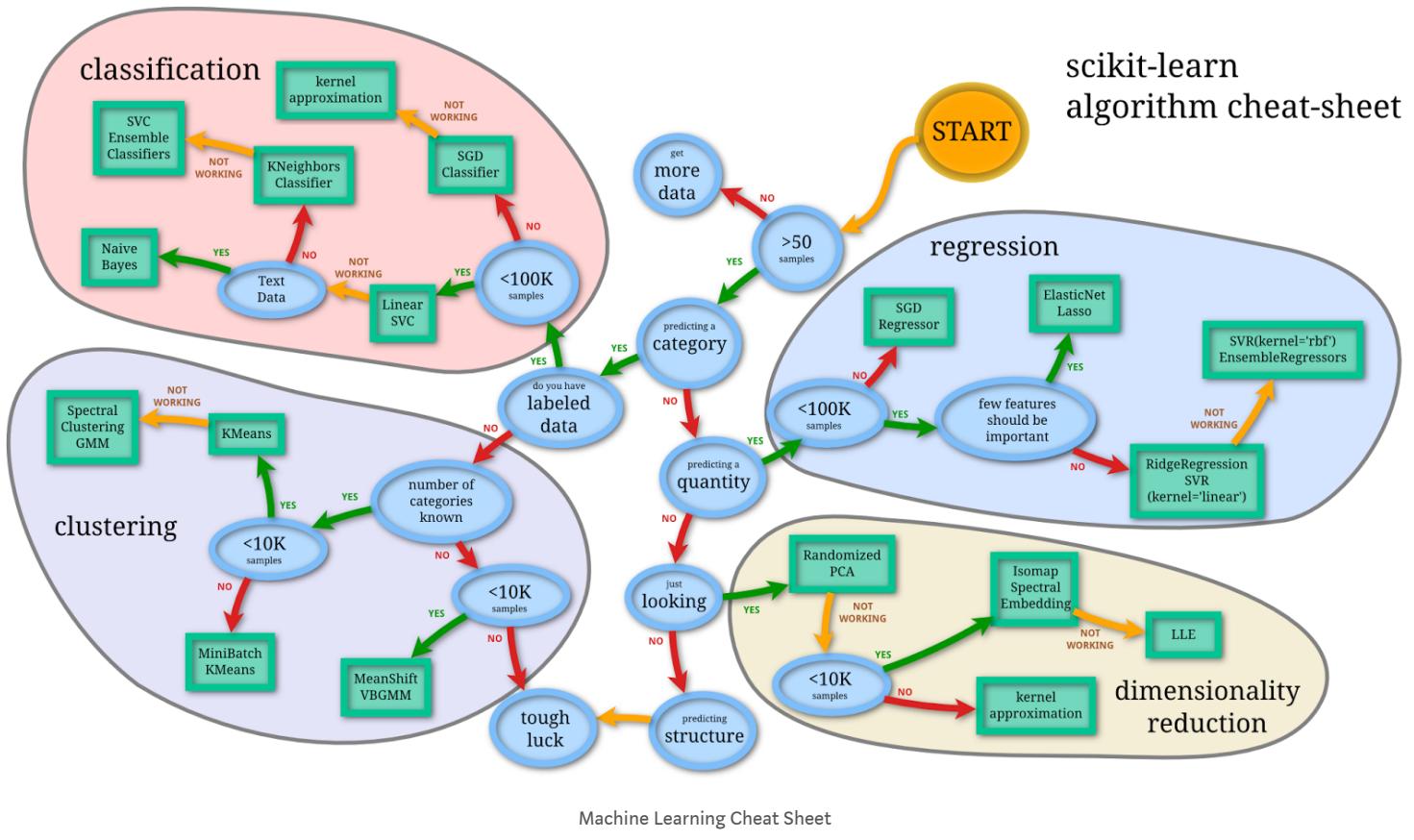
<p>Linear Vector Spaces: Definition: A linear vector space, X, is a set of elements (vectors) defined over a scalar field, F, that satisfies the following conditions: 1) If $x \in X$ and $y \in X$, then $x+y \in X$. 2) $\lambda x \in X$ for all $\lambda \in F$. 3) For each vector $x \in X$, there is a unique vector $-x$ in X, such that $x+(-x)=0$. 4) For any two scalars $a, b \in F$ and any $x \in X$, $a(bx)=(ab)x$. 5) For any two scalars $a, b \in F$ and any $x \in X$, $a(x+y)=a(x)+b(y)$.</p> <p>Linear Independence: Consider n vectors $\{x_1, x_2, \dots, x_n\}$. If there exists n linearly independent vectors $\{a_1, a_2, \dots, a_n\}$, at least one of which is nonzero, such that $a_1x_1+a_2x_2+\dots+a_nx_n=0$, then the $\{x_i\}$ are linearly independent.</p> <p>Spanning a Space: Let S be a linear vector space and let $\{x_1, x_2, \dots, x_n\}$ be a subset of vectors in X. This subset $\{x_1, x_2, \dots, x_n\}$ is said to span S if for every vector $x \in S$, there exist scalars a_1, a_2, \dots, a_n such that $x = a_1x_1 + a_2x_2 + \dots + a_nx_n$.</p> <p>Inner Product: $\langle x, y \rangle$ for any scalar function of x and y. 1. $\langle xy \rangle = \langle yx \rangle$. 2. $\langle x(y+z) \rangle = \langle xy \rangle + \langle xz \rangle$. 3. $\langle xz \rangle \geq 0$, where equality holds if x is the zero vector.</p> <p>Norm: A scalar function $\ x\$ is called a norm if it satisfies: 1. $\ x\ \geq 0$. 2. $\ x\ = 0$ if and only if $x = 0$. 3. $\ cx\ = c \ x\$. 4. $\ x+y\ \leq \ x\ + \ y\$.</p> <p>Angle: The angle θ between two vectors x and y is defined by $\cos \theta = \frac{\langle x, y \rangle}{\ x\ \ y\ }$.</p> <p>Orthogonality: Two vectors $x, y \in X$ are said to be orthogonal if $\langle x, y \rangle = 0$.</p> <p>Gram Schmidt Orthogonalization: Assume that we have n independent vectors $\{y_1, y_2, \dots, y_n\}$. From these vectors we will obtain n orthogonal vectors $\{v_1, v_2, \dots, v_n\}$, $v_1 = y_1, \quad v_2 = y_2 - \sum_{i=1}^{i=1} \frac{\langle y_i, y_1 \rangle}{\langle y_i, y_1 \rangle} v_1, \quad \dots$ where $\frac{\langle y_i, y_k \rangle}{\langle y_i, y_1 \rangle} v_1$ is the projection of y_k on v_1.</p> <p>Vector Expansions:</p> $\begin{aligned} x &= \sum_{i=1}^n x_i v_i = x_1 v_1 + x_2 v_2 + \dots + x_n v_n, \\ \text{for orthogonal vectors, } x_i &= \frac{\langle v_i, x \rangle}{\langle v_i, v_i \rangle} v_i \end{aligned}$ <p>Reciprocal Basis Vectors:</p> $(r_i, v_j) = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}, \quad r_j = (r_j, x)$ <p>To compute the reciprocal basis vectors: set $B^{-1} = [v_1 \ v_2 \ \dots \ v_n]$, $R = [r_1 \ r_2 \ \dots \ r_n]$, $R^T = B^{-1}$. In matrix form: $x^R = B^{-1} x^r$.</p> <p>Transformations: A transformation consists of three parts: domain: $X = \{x\}$, range: $Y = \{y\}$, and a rule relating each x, y to an element $y \in Y$.</p> <p>Linear Transformations: transformation A is linear if: 1. for all $x, y \in X$, $A(x+y) = A(x) + A(y)$ 2. for all $x \in X$, $A(ax) = aA(x)$</p> <p>Matrix Representations: Let $\{v_1, v_2, \dots, v_n\}$ be a basis for vector space X, and let $\{u_1, u_2, \dots, u_n\}$ be a basis for vector space Y. Let A be a linear transformation with domain X and range Y: $A(x) = y$ The coefficients of the matrix representation are obtained from $A(v_j) = \sum_{i=1}^m a_{ij} u_i$ <p>Change of Basis: $B_t = [t_1 \ t_2 \ \dots \ t_n]$, $B_w = [w_1 \ w_2 \ \dots \ w_n]$ $A' = [B_w^T \ A B_t]$</p> <p>Eigenvalues & Eigenvectors: $Ax = \lambda x$, $[\ A - \lambda I\ = 0]$</p> <p>Diagonalization: $B = [z_1 \ z_2 \ \dots \ z_n]$, where $\{z_1, z_2, \dots, z_n\}$ are the eigenvectors of a square matrix A, $[B^{-1}AB] = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$</p> </p>	<p>Perceptron Architecture: $\mathbf{a} = \text{hardlim}(\mathbf{Wp} + \mathbf{b}), \mathbf{W} = [\mathbf{w}^T \ \mathbf{w}^T \ \dots \ \mathbf{w}^T]^T$ $a_i = \text{hardlim}_i(a_i) = \text{hardlim}_i(\mathbf{W}^T \mathbf{p} + b_i)$</p> <p>Decision Boundary: $\mathbf{w}^T \mathbf{p} + b = 0$ The decision boundary is always orthogonal to the weight vector.</p> <p>Perceptron Learning Rule: $\mathbf{W}^{\text{new}} = \mathbf{W}^{\text{old}} + \alpha \mathbf{x} \mathbf{e}^T, \mathbf{b}^{\text{new}} = \mathbf{b}^{\text{old}} + \alpha e, \quad \text{where } e = t - a$</p> <p>Hebb's Postulate: When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in exciting it, some growth or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing, is increased.</p> <p>Linear Associate: $a = \text{purelin}(\mathbf{Wp})$</p> <p>The Hebb Rule: Supervised Form: $w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + t_{ij} p_{iq}$ $\mathbf{W} = \mathbf{t}_1 \mathbf{P}_1^T + \mathbf{t}_2 \mathbf{P}_2^T + \dots + \mathbf{t}_Q \mathbf{P}_Q^T$</p> <p>Pseudoinverse Rule: $\mathbf{W} = \mathbf{T}^{\text{P}}$ When the number, R, of rows of \mathbf{P} is greater than the number of columns, Q, of \mathbf{P} and the columns of \mathbf{P} are independent, then the pseudoinverse can be computed by $\mathbf{P}^{\text{P}} = (\mathbf{P}^T \mathbf{P})^{-1} \mathbf{P}^T$</p> <p>Variations of Hebbian Learning:</p> <p>Filtered Learning (Ch.10): $\mathbf{W}^{\text{new}} = \mathbf{W}^{\text{old}} + \alpha(t_q - a_q) \mathbf{p}_q^T$</p> <p>Delta Rule (Ch.10): $\mathbf{W}^{\text{new}} = \mathbf{W}^{\text{old}} + \alpha(t_q - a_q) \mathbf{p}_q^T$</p> <p>Unsupervised Hebb (Ch.10): $\mathbf{W}^{\text{new}} = \mathbf{W}^{\text{old}} + \alpha a_q \mathbf{p}_q^T$</p> <p>Taylor's F(x) = F(x)^T + $\frac{1}{2} (x - x^*)^T \nabla^2 F(x) _{x=x^*} (x - x^*) + \dots$</p> <p>Grad F(x) = $\left[\begin{array}{c} \frac{\partial}{\partial x_1} F(x) \$</p>
--	---



Machine Learning Cheat Sheet

Machine Learning: Scikit-learn algorithm

This machine learning cheat sheet will help you find the right estimator for the job which is the most difficult part. The flowchart will help you check the documentation and rough guide of each estimator that will help you to know more about the problems and how to solve it.



Scikit-Learn

Scikit-learn (formerly `scikits.learn`) is a [free software machine learning library](#) for the [Python](#) programming language. It features various [classification](#), [regression](#) and [clustering](#) algorithms including [support vector machines](#), [random forests](#), [gradient boosting](#), [k-means](#) and [DBSCAN](#), and is designed to interoperate with the Python numerical and scientific libraries [NumPy](#) and [SciPy](#).

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.cross_validation import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X = iris.data[:, 2:4]
>>> y = iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
>>> 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','M','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

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

Model Fitting

Supervised learning

```
>>> lr.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)
```

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

Estimator score method
Metric scoring functions
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_true, 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_)
```

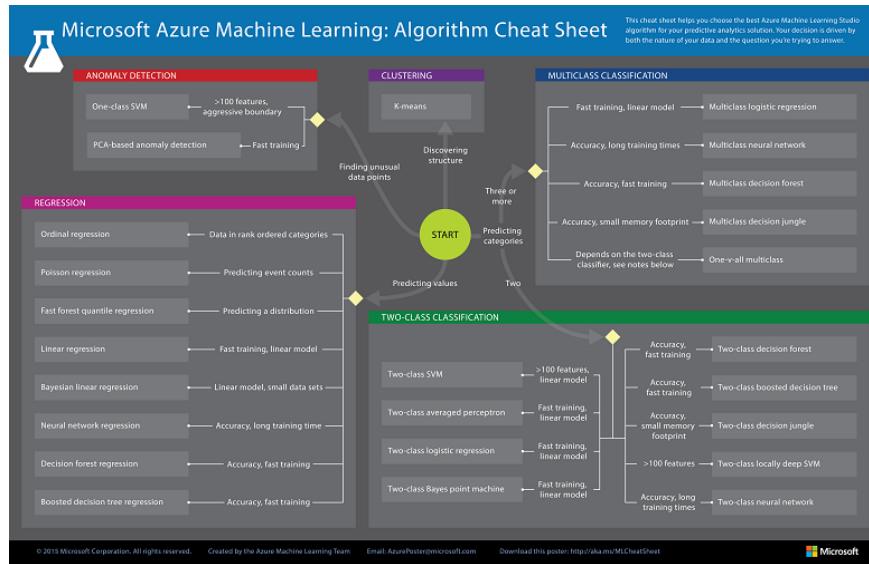
DataCamp
Learn Python for Data Science interactively



Scikit-Learn Cheat Sheet

MACHINE LEARNING : ALGORITHM CHEAT SHEET

This machine learning cheat sheet from Microsoft Azure will help you choose the appropriate machine learning algorithms for your predictive analytics solution. First, the cheat sheet will ask you about the data nature and then suggests the best algorithm for the job.



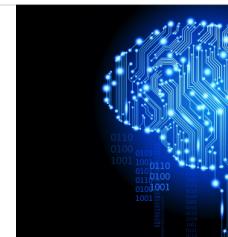
MACHINE LEARNING ALGORITHM CHEAT SHEET

>>> If you like this list, you can let me know here. <<<

Ultimate Guide to Leveraging NLP & Machine Learning for your Chatbot

Code Snippets and Github Included

chatbotslife.com



Python for Data Science

This cheat sheet provides a quick reference for Python Data Science, organized into several sections:

- Python Basics**: Includes code examples for variable assignment (e.g., `x = 5`), calculations with variables (e.g., `x*2`), and type conversion (e.g., `str(5)`).
- Variables and Data Types**: Shows how to work with lists, strings, integers, floats, and booleans.
- Asking For Help**: How to use `help(str)` to get documentation for built-in functions.
- Lists**: Examples of list operations like `my_list[1]`, `my_list[-3]`, and `my_list[1:3]`.
- Selecting List Elements**: How to select items at index 1, 2, or 3, or before index 3.
- List Operations**: Examples of list methods like `my_list.append()`, `my_list.extend()`, and `my_list.insert()`.
- List Methods**: Detailed descriptions of list methods such as `index()`, `count()`, `remove()`, `reverse()`, `extend()`, `insert()`, and `sort()`.
- String Operations**: Examples of string concatenation and repetition (e.g., `my_string * 2`).
- String Methods**: Descriptions of string methods like `upper()`, `lower()`, `lstrip()`, `rstrip()`, `replace()`, and `strip()`.
- String Operations**: Examples of string operations like `my_string[3]` and `my_string[4:9]`.
- String Methods**: Descriptions of string methods like `lstrip()`, `lower()`, `count()`, and `replace()`.
- Libraries**: A section on libraries including NumPy, pandas, and matplotlib.
- Install Python**: Information on installing Python and using Anaconda.
- Numpy Arrays**: Examples of NumPy array creation and selection.
- Selecting Numpy Array Elements**: How to select items at index 0 and 1.
- Numpy Array Operations**: Examples of NumPy array operations like `my_array > 3` and `np.array([1, 2, 3], [4, 5, 6])`.
- Numpy Array Functions**: Descriptions of NumPy array functions like `array().shape`, `append()`, `insert()`, `delete()`, `mean()`, `median()`, `corrcoef()`, and `std()`.

Python Data Science Cheat Sheet

Python For Data Science Cheat Sheet

Bokeh

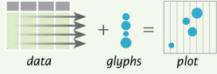
Learn Bokeh Interactively at www.DataCamp.com, taught by Bryan Van de Ven, core contributor



Plotting With Bokeh

The Python interactive visualization library **Bokeh** enables high-performance visual presentation of large datasets in modern web browsers.

Bokeh's mid-level general purpose `bokeh.plotting` interface is centered around two main components: data and glyphs.



The basic steps to creating plots with the `bokeh.plotting` interface are:

1. Prepare some data:
Python lists, NumPy arrays, Pandas DataFrames and other sequences of values
2. Create a new plot
3. Add renderers for your data, with visual customizations
4. Specify where to generate the output
5. Show or save the results

```
>>> from bokeh.plotting import figure
>>> from bokeh.io import output_file, show
>>> x = [1, 2, 3, 4, 5]          # Step 1
>>> y = [6, 7, 2, 4, 5]
>>> p = figure(title="simple line example",
              x_axis_label='x',
              y_axis_label='y')
>>> p.line(x, y, legend="Temp.", line_width=2) # Step 2
>>> output_file("lines.html")      # Step 3
>>> show(p)                      # Step 4
```

1 Data

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

Under the hood, your data is converted to Column Data Sources. You can also do this manually:

```
>>> import numpy as np
>>> import pandas as pd
>>> df = pd.DataFrame(np.array([[33.9, 4, 65, 'US'],
                                [32.4, 4, 66, 'Asia'],
                                [21.4, 4, 109, 'Europe'],
                                [columns=['mpg', 'cyl', 'hp', 'origin'],
                                index=['Toyota', 'Fiat', 'Volvo']])]
```

2 Plotting

```
>>> from bokeh.plotting import figure
>>> p1 = figure(plot_width=300, tools='pan,box_zoom')
>>> p2 = figure(plot_width=300, plot_height=300,
               x_range=(0, 8), y_range=(0, 8))
>>> p3 = figure()
```

3 Renderers & Visual Customizations

Glyphs

Scatter Markers

```
>>> p1.circle(np.array([1,2,3]), np.array([3,2,1]),
             fill_color='white')
>>> p2.square(np.array([1,5,3,5,5]), [1,4,3],
             color='blue', size=1)
```

Line Glyphs

```
>>> p1.line([1,2,3,4], [3,4,5,6], line_width=2)
>>> p2.multi_line(pd.DataFrame([[1,2,3],[5,6,7]]),
                  pd.DataFrame([[3,4,5],[3,2,1]]),
                  color="blue")
```

Rows & Columns Layout

Rows	Columns
>>> from bokeh.layouts import row	>>> from bokeh.layouts import column
>>> layout = row(p1,p2, p3)	>>> layout = column(p1,p2,p3)
Nesting Rows & Columns	
>>> layout = row(column(p1,p2), p3)	

Grid Layout

```
>>> from bokeh.layouts import gridplot
>>> row1 = [p1,p2]
>>> row2 = [p3]
>>> layout = gridplot([[p1,p2],[p3]])
```

Tabbed Layout

```
>>> from bokeh.models.widgets import Panel, Tabs
>>> tab1 = Panel(child=p1, title="tab1")
>>> tab2 = Panel(child=p2, title="tab2")
>>> layout = Tabs(tabs=[tab1, tab2])
```

Legends

Legend Location

Inside Plot Area	p.legend.location = 'bottom_left'
Outside Plot Area	>>> r1 = p2.asterisk(np.array([1,2,3]), np.array([3,2,1])) >>> r2 = p2.line([1,2,3,4], [3,4,5,6]) >>> legend = Legend(items=[None, (p1, r1), ("Two", [r2])], location=(0, -30)) >>> p.add_layout(legend, 'right')

Output

Output to HTML File

```
>>> from bokeh.io import output_file, show
>>> output_file('my_bar_chart.html', mode='cdn')
```

Notebook Output

```
>>> from bokeh.io import output_notebook, show
>>> output_notebook()
```

Embedding

Standalone HTML

```
>>> from bokeh.embed import file_html
>>> html = file_html(p, CDN, "my_plot")
>>> Components
>>> from bokeh.embed import components
>>> script, div = components(p)
```

5 Show or Save Your Plots

>>> show(p1)	>>> save(p1)
>>> show(layout)	>>> save(layout)

Customized Glyphs

[Also see Data](#)

```
>>> p.circle('mpg', 'cyl', source=cds_df,
             selection_color='red',
             nonselection_alpha=0.1)
```

Hover Glyphs

```
>>> hover = HoverTool(tooltips=None, mode='vline')
>>> p.add_tools(hover)
```

Colormapping

```
>>> color_mapper = CategoricalColorMapper(
    factors=['Europe', 'Asia', 'US'],
    palette=['red', 'green', 'blue'])
>>> p.circle('mpg', 'cyl', source=cds_df,
             color=dict(field='origin',
                         transform=color_mapper),
             legend='origin'))
```

[Also see Data](#)

Linked Plots

Linked Axes

```
>>> p1.x_range = p1.x_range
>>> p2.y_range = p1.y_range
```

Linked Brushing

```
>>> p4 = figure(plot_width = 100, tools='box_select,lasso_select')
>>> p4.circle('mpg', 'cyl', source=cds_df)
>>> p5 = figure(plot_width = 200, tools='box_select,lasso_select')
>>> p5.circle('mpg', 'hp', source=cds_df)
>>> layout = row(p4,p5)
```

Legend Orientation

[Also see Data](#)

```
>>> p.legend.orientation = "horizontal"
>>> p.legend.orientation = "vertical"
```

Legend Background & Border

```
>>> p.legend.border_line_color = "navy"
>>> p.legend.background_fill_color = "white"
```

[Also see Data](#)

Statistical Charts With Bokeh

Bokeh's high-level `bokeh.charts` interface is ideal for quickly creating statistical charts

Bar Chart

```
>>> from bokeh.charts import Bar
>>> p = Bar(df, stacked=True, palette=['red','blue'])
```

Box Plot

```
>>> from bokeh.charts import BoxPlot
>>> p = BoxPlot(df, values='vals', label='cyl',
                legend='bottom_right')
```

Histogram

```
>>> from bokeh.charts import Histogram
>>> p = Histogram(df, title='Histogram')
```

Scatter Plot

```
>>> from bokeh.charts import Scatter
>>> p = Scatter(df, x='mpg', y='hp', marker='square',
                xlabel='Miles Per Gallon',
                ylabel='Horsepower')
```

DataCamp
Learn Python for Data Science Interactively



Big Data Cheat Sheet

TensorFlow

In May 2017 Google announced the second-generation of the TPU, as well as the availability of the TPUs in [Google Compute Engine](#).^[12] The second-generation TPUs deliver up to 180 teraflops of performance, and when organized into clusters of 64 TPUs provide up to 11.5 petaflops.



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

Scikit Flow provides a set of high level model classes that you can use to easily integrate with your existing Scikit-learn pipeline code. Scikit Flow is a simplified interface for TensorFlow, to get people started on predictive analytics and data mining. Scikit Flow 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 documentations about the object

`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
```

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,`

globals()

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

locals()

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

TensorFlowDNNClassifier

TensorFlowDNNRegressor

TensorFlowLinearClassifier

TensorFlowLinearRegressor

TensorFlowRNNClassifier

TensorFlowRNNRegressor

TesorFlow Cheat Sheet

Keras

In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library. Chollet explained that Keras was conceived to be an interface rather than an end-to-end machine-learning framework. It presents a higher-level, more intuitive set of abstractions that make it easy to configure neural networks regardless of the backend scientific computing library.

Python For Data Science Cheat Sheet

Pandas Basics

Learn Python for Data Science [Interactively](#) at [www.DataCamp.com](#)



Pandas

The Pandas library is built on NumPy and provides easy-to-use data structures and data analysis tools for the Python programming language.



Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

Series

A one-dimensional labeled array capable of holding any data type

Index	A 3 B -5 C 7 D 4
--------------	---------------------------

```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame

Columns	Country	Capital	Population
Index	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasilia	207847528

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
   'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
   'Population': [11190846, 1303171035, 207847528]}
```

```
>>> df = pd.DataFrame(data,
   columns=['Country', 'Capital', 'Population'])
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> pd.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
Read multiple sheets from the same file
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

[Also see NumPy Arrays](#)

Getting

```
>>> s['b']
-5
>>> df[1:]
   Country    Capital  Population
1  India      New Delhi     1303171035
2  Brazil     Brasilia     207847528
```

Get one element

Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0], [0]]
   Belgium
0  Belgium
```

Select single value by row & column

By Label

```
>>> df.loc[[0], ['Country']]
   Belgium
0  Belgium
```

Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
   Country    Brazil
   Capital    Brasilia
   Population 207847528
>>> df.ix[:, 'Capital']
0  Brussels
1  New Delhi
2  Brasilia
```

Select single row of subset of rows

```
>>> df.ix[1, 'Capital']
'New Delhi'
```

Select a single column of subset of columns

Setting

```
>>> s['a'] = 6
```

Set index a of Series s to 6

Boolean Indexing

```
>>> s[(s > 1)]
>>> s[(s < -1) | (s > 2)]
>>> df[df['Population']>1200000000]
```

Series s where value is not >1
s where value is <-1 or >2
Use filter to adjust DataFrame

Setting

```
>>> s['a'] = 6
```

Set index a of Series s to 6

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
read_sql() is a convenience wrapper around read_sql_table() and read_sql_query()

>>> pd.to_sql('myDF', engine)
```

Dropping

```
>>> s.drop(['a', 'c'])
Drop values from rows (axis=0)
>>> df.drop('Country', axis=1)
Drop values from columns(axis=1)
```

Drop values from rows (axis=0)

Drop values from columns(axis=1)

Sort & Rank

```
>>> df.sort_index(by='Country')
>>> s.order()
>>> df.rank()
```

Sort by row or column index

Sort a series by its values

Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape
(rows,columns)
>>> df.index
Describe index
>>> df.columns
Describe DataFrame columns
>>> df.info()
Info on DataFrame
>>> df.count()
Number of non-NA values
```

Describe index

Describe DataFrame columns

Info on DataFrame

Number of non-NA values

Summary

```
>>> df.sum()
Sum of values
>>> df.cumsum()
Cumulative sum of values
>>> df.min()/df.max()
Minimum/maximum values
>>> df.idmin()/df.idmax()
Minimum/Maximum index value
>>> df.describe()
Summary statistics
>>> df.mean()
Mean of values
>>> df.median()
Median of values
```

Sum of values

Cumulative sum of values

Minimum/maximum values

Minimum/Maximum index value

Summary statistics

Mean of values

Median of values

Applying Functions

```
>>> f = lambda x: x*x
>>> df.apply(f)
>>> df.applymap(f)
```

Apply function

Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
   a    10.0
   b    NaN
   c     5.0
   d     7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
   a    10.0
   b    -5.0
   c     5.0
   d     7.0
>>> s.sub(s3, fill_value=2)
   a     8.0
   b    -7.0
   c     3.0
   d     5.0
>>> s.div(s3, fill_value=4)
   a    2.5
   b    -1.0
   c     1.0
   d     1.75
>>> s.mul(s3, fill_value=3)
   a    30.0
   b    -15.0
   c     15.0
   d     21.0
```

DataCamp

Learn Python for Data Science [Interactively](#)



Pandas Cheat Sheet

Data Wrangling

The term “data wrangler” is starting to infiltrate pop culture. In the 2017 movie Kong: Skull Island, one of the characters, played by actor Marc Evan Jackson is introduced as “Steve Woodward, our data wrangler”.

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			
1	4	7	10
d	2	5	8
e	2	6	9
			11
			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
```

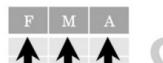
Method Chaining

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')
     )
```

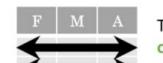
Tidy Data – A foundation for wrangling in pandas

In a tidy data set:



Each variable is saved in its own column

&



Each observation is saved in its own row

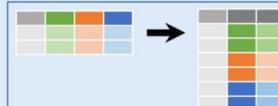


M * A

Reshaping Data – Change the layout of a data set

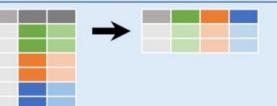
pd.melt(df)

Gather columns into rows.



pd.pivot(columns='var', values='val')

Spread rows into columns.



pd.concat([df1, df2])

Append rows of DataFrames

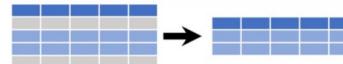


pd.concat([df1, df2], axis=1)

Append columns of DataFrames



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.

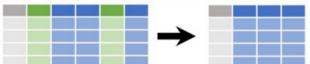
```
df.nlargest(n, 'value')
```

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	!=
>	Greater than	df.column.isin(values)
==	Equals	pd.isnull(obj)
<=	Less than or equals	pd.notnull(obj)
>=	Greater than or equals	&, , ~, ^, df.any(), df.all()

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

Data Wrangling Cheat Sheet

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.
```

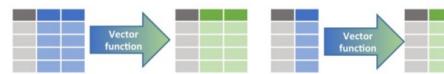
Make New Columns



```
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:

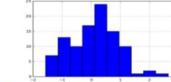
```
max(axis=1) min(axis=1)
Element-wise max. Element-wise min.
clip(lower=-10,upper=10) abs()
Trim values at input thresholds Absolute value.
```

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

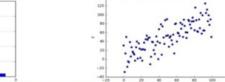
shift(1)	shift(-1)
Copy with values shifted by 1.	Copy with values lagged by 1.
rank(method='dense')	cumsum()
Ranks with no gaps.	Cumulative sum.
rank(method='min')	cummax()
Ranks. Ties get min rank.	Cumulative max.
rank(pct=True)	cummin()
Ranks rescaled to interval [0, 1].	Cumulative min.
rank(method='first')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

Plotting

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

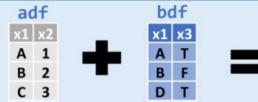


```
df.plot.scatter(x='w',y='h')
Scatter chart using pairs of points
```



Pandas Data Wrangling Cheat Sheet

Combine Data Sets



Standard Joins

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

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

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

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

Filtering Joins

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

```
x1 x2
A 1
B 2
C 3
adf[~adf.x1.isin(bdf.x1)]
All rows in adf that do not have a match in bdf.
```



Set-like Operations

```
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)
.indicator(_merge == "left_only")
.drop(['_merge'], axis=1)
```

```
Rows that appear in ydf but not zdf (Setdiff).
```

Ultimate Guide to Leveraging NLP & Machine Learning for your Chatbot

Code Snippets and Github Included

chatbotslife.com



Data Wrangling with dplyr and tidyr

Data Wrangling with dplyr and tidyverse

Cheat Sheet



Syntax - Helpful conventions for wrangling

`dplyr::tbl_df(iris)`

Converts data to `tbl` class. `tbl`'s are easier to examine than data frames. R displays only the data that fits onscreen:

```
Source: local data frame [150 x 5]
# ... omitted columns ...
Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4      0.2   setosa
2          4.9         3.0          1.4      0.2   setosa
3          4.7         3.2          1.3      0.2   setosa
4          4.6         3.1          1.5      0.2   setosa
5          5.0         3.6          1.4      0.2   setosa
..          ...         ...          ...      ...
Variables not shown: Petal.Width (dbl); ...
Species (fctr)
```

`dplyr::glimpse(iris)`

Information dense summary of `tbl` data.

`utils::View(iris)`

View data set in spreadsheet-like display (note capital V).

iris					
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa

`dplyr::%>%`

Passes object on left hand side as first argument (or . argument) of function on righthand side.

```
x %>% f(y) is the same as f(x, y)
y %>% f(x, ., z) is the same as f(x, y, z)
```

"Piping" with `%>%` makes code more readable, e.g.

```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com

Tidy Data - A foundation for wrangling in R

In a tidy data set:

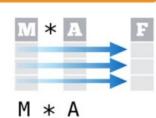


&



Each variable is saved in its own column

Each observation is saved in its own row



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

Reshaping Data - Change the layout of a data set



`tidy::gather(cases, "year", "n", 2:4)`

Gather columns into rows.



`tidy::spread(pollution, size, amount)`

Spread rows into columns.

`tidy::separate(storms, date, c("y", "m", "d"))`

Separate one column into several.



`tidy::unite(data, col, ..., sep)`

Unite several columns into one.

`dplyr::data_frame(a = 1:3, b = 4:6)`

Combine vectors into data frame (optimized).

`dplyr::arrange(mtcars, mpg)`

Order rows by values of a column (low to high).

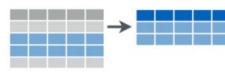
`dplyr::arrange(mtcars, desc(mpg))`

Order rows by values of a column (high to low).

`dplyr::rename(tb, y = year)`

Rename the columns of a data frame.

Subset Observations (Rows)



`dplyr::filter(iris, Sepal.Length > 7)`

Extract rows that meet logical criteria.

`dplyr::distinct(iris)`

Remove duplicate rows.

`dplyr::sample_frac(iris, 0.5, replace = TRUE)`

Randomly select fraction of rows.

`dplyr::sample_n(iris, 10, replace = TRUE)`

Randomly select n rows.

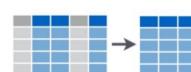
`dplyr::slice(iris, 10:15)`

Select rows by position.

`dplyr::top_n(storms, 2, date)`

Select and order top n entries (by group if grouped data).

Subset Variables (Columns)



`dplyr::select(iris, Sepal.Width, Petal.Length, Species)`

Select columns by name or helper function.

Helper functions for select - ?select

`select(iris, contains("x"))`

Select columns whose name contains a character string.

`select(iris, ends_with("Length"))`

Select columns whose name ends with a character string.

`select(iris, everything())`

Select every column.

`select(iris, matches("t.*"))`

Select columns whose name matches a regular expression.

`select(iris, num_range("x", 1:5))`

Select columns named x1, x2, x3, x4, x5.

`select(iris, one_of(c("Species", "Genus")))`

Select columns whose names are in a group of names.

`select(iris, starts_with("Sepal"))`

Select columns whose name starts with a character string.

`select(iris, Sepal.Length:Petal.Width)`

Select all columns between Sepal.Length and Petal.Width (inclusive).

`select(iris, -Species)`

Select all columns except Species.

Logic in R - ?Comparison, ?base::Logic

<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&, , !, xor, any, all	Boolean operators

devtools::install_github("rstudio/EDAWR") for data sets

Learn more with `browseVignettes(package = c("dplyr", "tidyverse"))` • dplyr 0.4.0 • tidyverse 0.2.0 • Updated: 1/15

Data Wrangling with dplyr and tidyverse Cheat Sheet

Summarise Data



`dplyr::summarise(iris, avg = mean(Sepal.Length))`

Summarise data into single row of values.

`dplyr::summarise_each(iris, funs(mean))`

Apply summary function to each column.

`dplyr::count(iris, Species, wt = Sepal.Length)`

Count number of rows with each unique value of variable (with or without weights).



Summarise uses **summary functions**, functions that take a vector of values and return a single value, such as:

`dplyr::first`

First value of a vector.

`dplyr::last`

Last value of a vector.

`dplyr::nth`

Nth value of a vector.

`dplyr::n`

of values in a vector.

`dplyr::n_distinct`

of distinct values in a vector.

`IQR`

IQR of a vector.

`min`

Minimum value in a vector.

`max`

Maximum value in a vector.

`mean`

Mean value of a vector.

`median`

Median value of a vector.

`var`

Variance of a vector.

`sd`

Standard deviation of a vector.

Make New Variables



`dplyr::mutate(iris, sepal = Sepal.Length + Sepal.Width)`

Compute and append one or more new columns.

`dplyr::mutate_each(iris, funs(min_rank))`

Apply window function to each column.

`dplyr::transmute(iris, sepal = Sepal.Length + Sepal.Width)`

Compute one or more new columns. Drop original columns.



Mutate uses **window functions**, functions that take a vector of values and return another vector of values, such as:

`dplyr::lead`

Copy with values shifted by 1.

`dplyr::lag`

Copy with values lagged by 1.

`dplyr::dense_rank`

Ranks with no gaps.

`dplyr::min_rank`

Ranks. Ties get min rank.

`dplyr::percent_rank`

Ranks rescaled to [0, 1].

`dplyr::row_number`

Ranks. Ties get to first value.

`dplyr::ntile`

Bin vector into n buckets.

`dplyr::between`

Are values between a and b?

`dplyr::cume_dist`

Cumulative distribution.

`dplyr::cumall`

Cumulative all

`dplyr::cumany`

Cumulative any

`dplyr::cummean`

Cumulative mean

`cumsum`

Cumulative sum

`cummax`

Cumulative max

`cummin`

Cumulative min

`cumprod`

Cumulative prod

`pmax`

Element-wise max

`pmin`

Element-wise min

`iris %>% group_by(Species) %>% summarise(...)`

Compute separate summary row for each group.



RStudio® is a trademark of RStudio, Inc. • [CC BY RStudio](#) • [info@rstudio.com](#) • 844-448-1212 • [rstudio.com](#)

`devtools::install_github("rstudio/EDAWR")` for data sets

Learn more with `browseVignettes(package = c("dplyr", "tidyverse"))` • dplyr 0.4.0 • tidyverse 0.2.0 • Updated: 1/15

Combine Data Sets

a	x1	x2	x3
A	1	T	
B	2	F	
C	3	NA	

b	x1	x3
A	T	
B	F	
D	T	

Mutating Joins

`dplyr::left_join(a, b, by = "x1")`

Join matching rows from b to a.

`dplyr::right_join(a, b, by = "x1")`

Join matching rows from a to b.

`dplyr::inner_join(a, b, by = "x1")`

Join data. Retain only rows in both sets.

`dplyr::full_join(a, b, by = "x1")`

Join data. Retain all values, all rows.

Filtering Joins

`dplyr::semi_join(a, b, by = "x1")`

All rows in a that have a match in b.

`dplyr::anti_join(a, b, by = "x1")`

All rows in a that do not have a match in b.

y	x1	x2
A	1	
B	2	
C	3	

z	x1	x2
B	2	
C	3	
D	4	

Set Operations

y	x1	x2
A	1	
B	2	

z	x1	x2
B	2	
C	3	

`dplyr::intersect(y, z)`

Rows that appear in both y and z.

`dplyr::union(y, z)`

Rows that appear in either or both y and z.

`dplyr::setdiff(y, z)`

Rows that appear in y but not z.

Binding
<code>dplyr::bind_rows(y, z)</code>
Append z to y as new rows.

Binding
<code>dplyr::bind_cols(y, z)</code>
Append z to y as new columns.

Caution: matches rows by position.

Data Wrangling with dplyr and tidyverse Cheat Sheet

Scipy

SciPy builds on the NumPy array object and is part of the NumPy stack which includes tools like Matplotlib, pandas and SymPy, and an expanding set of scientific computing libraries. This NumPy stack has similar users to other applications such as MATLAB, GNU Octave, and Scilab. The NumPy stack is also sometimes referred to as the SciPy stack.[3]

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](#)

Matrix Functions

Addition	>>> np.add(A,D)	Addition
Subtraction	>>> np.subtract(A,D)	Subtraction
Division	>>> np.divide(A,D)	Division
Multiplication	>>> A @ D	Multiplication operator (Python 3)
	>>> np.multiply(D,A)	Multiplication
	>>> np.dot(A,D)	Dot product
	>>> np.vdot(A,D)	Vector dot product
	>>> np.inner(A,D)	Inner product
	>>> np.outer(A,D)	Outer product
	>>> np.tensordot(A,D)	Tensor dot product
	>>> np.kron(A,D)	Kronecker product

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

Logarithm Function

	>>> linalg.logm(A)	Matrix sine
--	--------------------	-------------

Trigonometric Functions

	>>> linalg.sinm(D)	Matrix cosine
	>>> linalg.cosm(D)	Matrix tangent

Hyperbolic Trigonometric Functions

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

Matrix Sign Function

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

Matrix Square Root

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

Arbitrary Functions

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

Decompositions

Eigenvalues and Eigenvectors	>>> la, v = linalg.eig(A)	Solve ordinary or generalized eigenvalue problem for square matrix
------------------------------	---------------------------	--

	>>> l1, l2 = la	Unpack eigenvalues
	>>> v[:,0]	First eigenvector
	>>> v[:,1]	Second eigenvector

	>>> linalg.eigvals(A)	Unpack eigenvalues
--	-----------------------	--------------------

Singular Value Decomposition	>>> U,s,Vh = linalg.svd(B)	Singular Value Decomposition (SVD)
------------------------------	----------------------------	------------------------------------

	>>> M,N = B.shape	Construct sigma matrix in SVD
	>>> Sig = linalg.diagsvd(s,M,N)	

LU Decomposition

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

Sparse Matrix Decompositions

	>>> la, v = sparse.linalg.eigs(F,1)	Eigenvalues and eigenvectors
--	-------------------------------------	------------------------------

	>>> sparse.linalg.svds(H, 2)	SVD
--	------------------------------	-----

DataCamp

Learn Python for Data Science [Interactively](#)



Scipy Cheat Sheet

Matplotlib

matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural “pylab” interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged.[2] SciPy makes use of matplotlib.

pyplot is a matplotlib module which provides a MATLAB-like interface.

[6] matplotlib is designed to be as usable as MATLAB, with the ability to use Python, with the advantage that it is free.

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](#)

```
>>> 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
>>> ax2 = 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[0,1].bar([1,2,3],[5,1,2.5],[0,1,2])
>>> axes[1,1].xline(1.5)
>>> axes[0,1].axvline(0.65)
>>> ax.fill(x,y,color='blue')
>>> ax.fill_between(x,y,color='yellow')
```

Draw points with lines or markers connecting them
Draw unconnected points, scaled or colored
Not vertical rectangles (constant width)
Plot horizontal rectangles (constant height)
Draw a horizontal line across axes
Draw a vertical line across axes
Draw filled polygons
Fill between y-values and 0

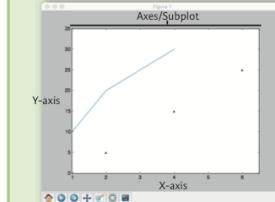
2D Data or Image

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

Colormapped or RGB arrays

Plot Anatomy & Workflow

Plot Anatomy



Figure

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()
>>> ax = fig.add_subplot(111)
>>> ax.plot(x, y, color='lightblue', linewidth=3)
>>> ax.scatter([5,15,25],
              color='darkgreen',
              marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()
```

Step 1
Step 2
Step 3
Step 4
Step 5
Step 6

4 Customize Plot

Colors, Color Bars & Color Maps

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

Markers

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

LineStyles

```
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x*x**2,'-')
>>> pit.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[0,0].arrow(0,0,0.5,0.5)
>>> axes[0,1].quiver(y,z)
>>> axes[0,1].streamplot(X,Y,U,V)
```

Add an arrow to the axes
Plot a 2D field of arrows
Plot a 2D field of arrows

Data Distributions

```
>>> ax1.hist(y)
>>> ax3.boxplot(y)
>>> ax3.violinplot(z)
```

Plot a histogram
Make a box and whisker plot
Make a violin plot

MathText

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

Limits, Legends & Layouts

```
>>> ax.margins(x=0,y=0.1)
>>> ax.axis('equal')
>>> ax.set(xlim=[0, 10.5], ylim=[-1.5,1.5])
>>> ax.set(x0,10.5)
```

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

Legends

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

Ticks

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

No overlapping plot elements
Manually set x-ticks
Make y-ticks longer and go in and out

Subplot Spacing

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

Axis Spines

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

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

Show Plot

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

Close & Clear

Close

Clear

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

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

DataCamp

Learn Python For Data Science Interactively



Matplotlib Cheat Sheet

>>> If you like this list, you can let me know [here](#). <<<

Data Visualization

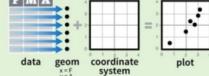
Data Visualization with ggplot2

Cheat Sheet

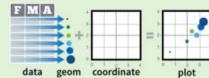


Basics

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 **plot()** or **ggplot()**

```
aesthetic mappings    data    geom
ggplot(x = cty, y = hwy, aes(x = cty, y = hwy))
```

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 **plot()**.

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

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.

RStudio® is a trademark of RStudio, Inc. • [CC BY RStudio](#) • [info@rstudio.com](#) • 844-448-1212 • [rstudio.com](#)

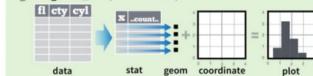
Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.		
One Variable <ul style="list-style-type: none"> Continuous <ul style="list-style-type: none"> a + geom_area(stat = "bin") x, y, alpha, color, fill, linetype, size b + geom_area(aes(y = ..density..), stat = "bin") a + geom_density(stat = "gaussian") x, y, alpha, color, fill, linetype, size, weight b + geom_density(aes(y = ..count..)) a + geom_dotplot() x, y, alpha, color, fill a + geom_freqpoly() x, y, alpha, color, linetype, size b + geom_freqpoly(aes(y = ..density..)) a + geom_histogram(binwidth = 5) x, y, alpha, color, fill, linetype, size, weight b + geom_histogram(aes(y = ..density..)) Discrete <ul style="list-style-type: none"> b + geom_bar() x, alpha, color, fill, linetype, size, weight 		Two Variables <ul style="list-style-type: none"> 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
Graphical Primitives <ul style="list-style-type: none"> c <- ggplot(map, aes(long, lat)) c + geom_polygon(aes(group = group)) x, y, alpha, color, fill, linetype, size d <- ggplot(economics, aes(date, unemploy)) d + geom_path(linewidth = "butt", linejoin = "round", linemetre = 1) x, y, alpha, color, linetype, size d + geom_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900)) x, y, alpha, color, fill, linetype, size e <- ggplot(seals, aes(x = long, y = lat)) e + geom_segment(aes(xend = long + delta_long, yend = lat + delta_lat)) x, end, y, end, 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 		Continuous X, Continuous Y <ul style="list-style-type: none"> f <- ggplot(mpg, aes(cty, hwy)) f + geom_hex() x, y, alpha, colour, fill size
		Continuous Bivariate Distribution <ul style="list-style-type: none"> 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
		Continuous Function <ul style="list-style-type: none"> 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 <ul style="list-style-type: none"> 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, y, max, ymin, alpha, color, linetype, size, width (also geom_errorbarh()) k + geom_linerange() x, y, min, max, alpha, color, linetype, size k + geom_pointrangle() x, y, ymin, ymax, alpha, color, fill, linetype, shape, size
		Maps <ul style="list-style-type: none"> data <- data.frame(murder = USArrests\$Murder, state = tolower(rownames(USArrests))) map <- map_data("state") l <- ggplot(data, aes(fill = murder)) l + geom_map(aes(map_id = state), map = map) + expand_limits(x = map\$long, y = map\$lat) map_id, alpha, color, fill, linetype, size
Three Variables <ul style="list-style-type: none"> seals\$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)) m <- ggplot(seals, aes(long, lat)) m + geom_contour(aes(z = z)) x, y, z, alpha, colour, linetype, size, weight m + geom_raster(aes(fill = z), hjust = 0.5, vjust = 0.5, interpolate = FALSE) x, y, alpha, fill m + geom_tile(aes(fill = z)) x, y, alpha, color, fill, linetype, size 		

Learn more at [docs.ggplot2.org](#) • ggplot2 0.9.3.1 • Updated: 3/15

Data Visualization Cheat Sheet

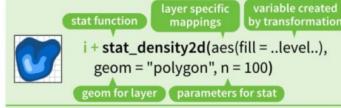
Stats - An alternative way to build a layer

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, ..count, ..density, ..ndensity.
a + stat_bindot(binwidth = 1, binaxis = "x")
x, y | ..count, ..count...
a + stat_binhex(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 | ..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(quartiles = 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),           General Purpose
fun = dnorm, n = 10), args = list(sd = 0.5))
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()
```

RStudio® is a trademark of RStudio, Inc. • [CC BY RStudio](#) • [info@rstudio.com](#) • 844-448-1212 • [rstudio.com](#)

Scales

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



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(l))` - map discrete values to manually chosen visual values

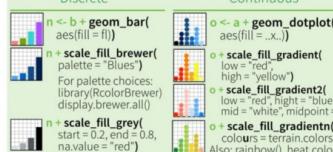
X and Y location scales

Use with x or y aesthetics (x shown here)

`scale_x_date(labels = date_format("%m/%d/%Y"),
breaks = date_breaks("2 weeks"))` - treat x values as dates. See `strptime` for label formats.
`scale_x_datetime()` - treat x values as date times. Use same arguments as `scale_x_date`.
`scale_x_log10()` - Plot x on log10 scale
`scale_x_reverse()` - Reverse direction of x axis
`scale_x_sqrt()` - Plot x on square root scale

Color and fill scales

Discrete Continuous



Shape scales

Manual shape values



Size scales

Size mapped to area of circle (not radius)

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(xtrans = "sqrt")`

xtrans, ytrans, xlim, ylim
Transformed cartesian coordinates. Set extras and strains to the name of a window function.

`r + coord_map(projection = "ortho",
orientation = c(41, -74, 0))`
projection, orientation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)



Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

`s <- ggplot(mpg, aes(fl, fill = drv))`

`s + geom_bar(position = "dodge")`
Arrange elements side by side

`s + geom_bar(position = "fill")`
Stack elements on top of one another, normalize height

`s + geom_bar(position = "stack")`
Stack elements on top of one another

`f + geom_point(position = "jitter")`
Add random noise to X and Y position of each element to avoid overplotting

Each position adjustment can be recast as a function with manual `width` and `height` arguments

`s + geom_bar(position = position_dodge(width = 1))`

Themes

`r + theme_bw()`
White background with grid lines

`r + theme_classic()`
White background no gridlines

`r + theme_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()`



Set `scales` to let axis limits vary across facets

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

Use scale functions to update legend labels

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

Labels

`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.

Legends

`t + theme_bw()`
Without clipping (preferred)
`t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20))`
`t + scale_x_continuous(limits = c(0, 100)) + scale_y_continuous(limits = c(0, 100))`
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))`

Learn more at [docs.ggplot2.org](#) • ggplot2 0.9.3.1 • Updated: 3/15

ggplot cheat sheet

PySpark

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
>>> sc.appName          Return application name
>>> sc.applicationId   Return 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
```

Select 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

<pre>>>> 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',2):1,('a',7):1}) 4950 >>> sc.parallelize([]).isEmpty() True</pre>	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

<pre>>>> 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()</pre>	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

<pre>>>> rdd.map(lambda x: x+[x[1],x[0]]) >>> rdd1 = rdd.map(lambda x: x[1]+x[0]) [('a',7,('a',1)), ('a',2,2,('a',1)), ('b',2,2,('b',1))] >>> 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 = rdd.flatMapValues(lambda x: x) >>> rdd4.collect() [('a','x'), ('a','y'), ('a','z'), ('b','p'), ('b','r')]</pre>	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 <code>rdd4</code> without changing the keys
---	---

Selecting Data

<pre>>>> rdd.collect() [('a',7,('a',1)), ('a',2,2,('a',1)), ('b',2,2,('b',1))] >>> rdd.take(2) [('a',7,('a',1)), ('a',2,2,('a',1))] >>> rdd.first() ('a',7) >>> rdd.top(2) [('b', 2), ('a', 7)] >>> rdd.sample(False, 0.15, 81).collect() [3,4,27,31,40,41,42,43,60,76,79,80,86,97]</pre>	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 <code>rdd3</code>
<pre>>>> rdd.filter(lambda x: "a" in x) >>> rdd.collect() [('a',7), ('a',2)] >>> rdd.distinct().collect() [('a',2), ('b',7)] >>> rdd.keys().collect() ['a', 'a', 'b']</pre>	Filter the RDD Return distinct RDD values Return (key,value) RDD's keys

Iterating

```
>>> def g(x): print(x)
>>> rdd.foreach(g)
```

Apply a function to all RDD elements

Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x,y : x+y)
[('a',3), ('b',2)]
>>> rdd.reduceByKey(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()
[('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]))
(4950, 100)
>>> rdd3.aggregate((0,0),seqOp,combOp)
.collect()
[('a', (9, 2)), ('b', (2, 1))]
>>> rdd3.foldByKey(0,add)
4950
>>> rdd3.foldByKey(0, add)
.collect()
[('a',9), ('b',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',7), ('a',7)]
```

Return each RDD value not contained in `rdd2`

```
>>> rdd2.subtractByKey(rdd)
.collect()
[('d', 1)]
```

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

```
>>> rdd.cartesian(rdd2).collect()
```

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

Sort

```
>>> rdd2.sortBy(lambda x: x[1])
.collect()
```

Sort RDD by given function

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

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
```

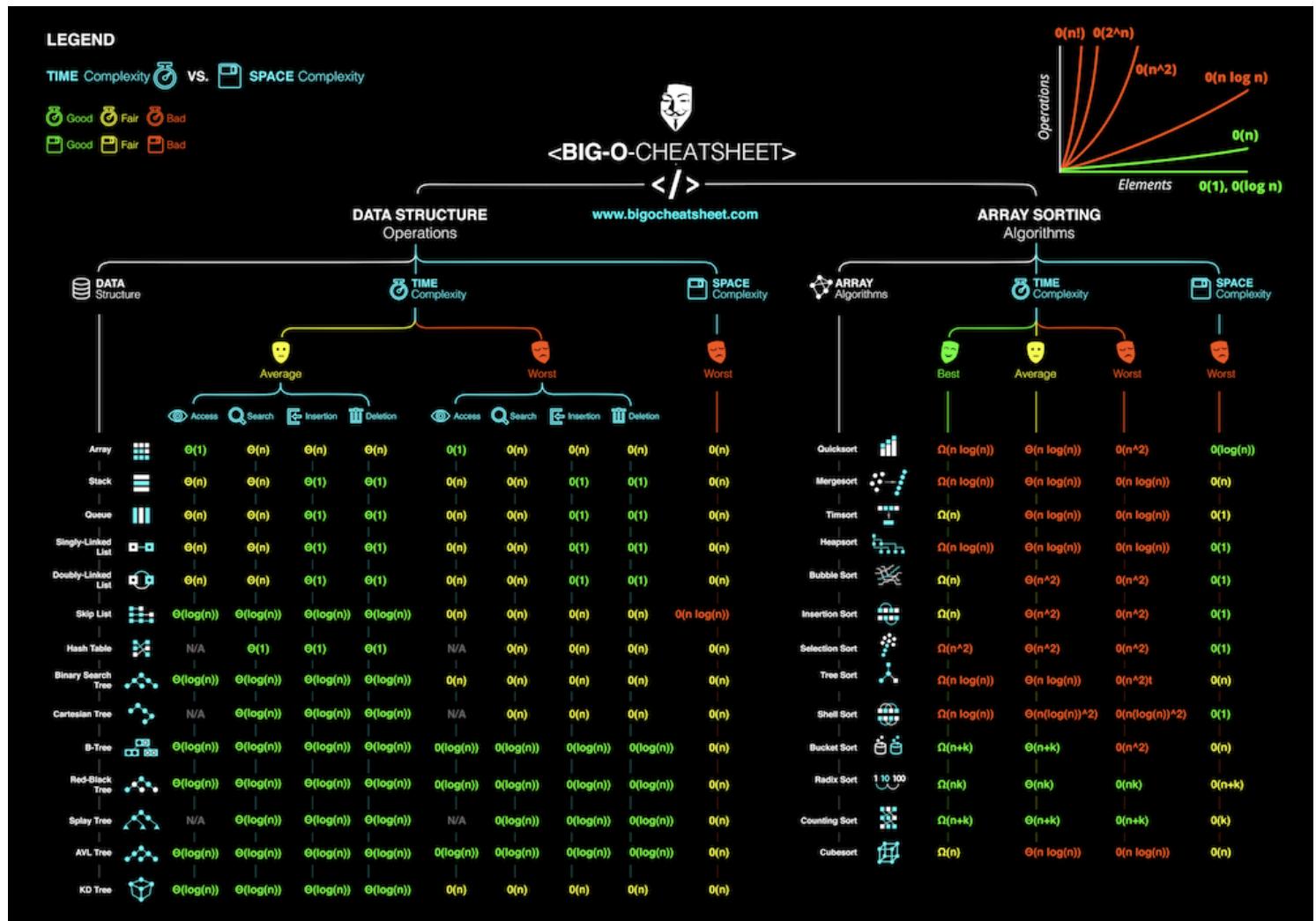
DataCamp

Learn Python for Data Science interactively

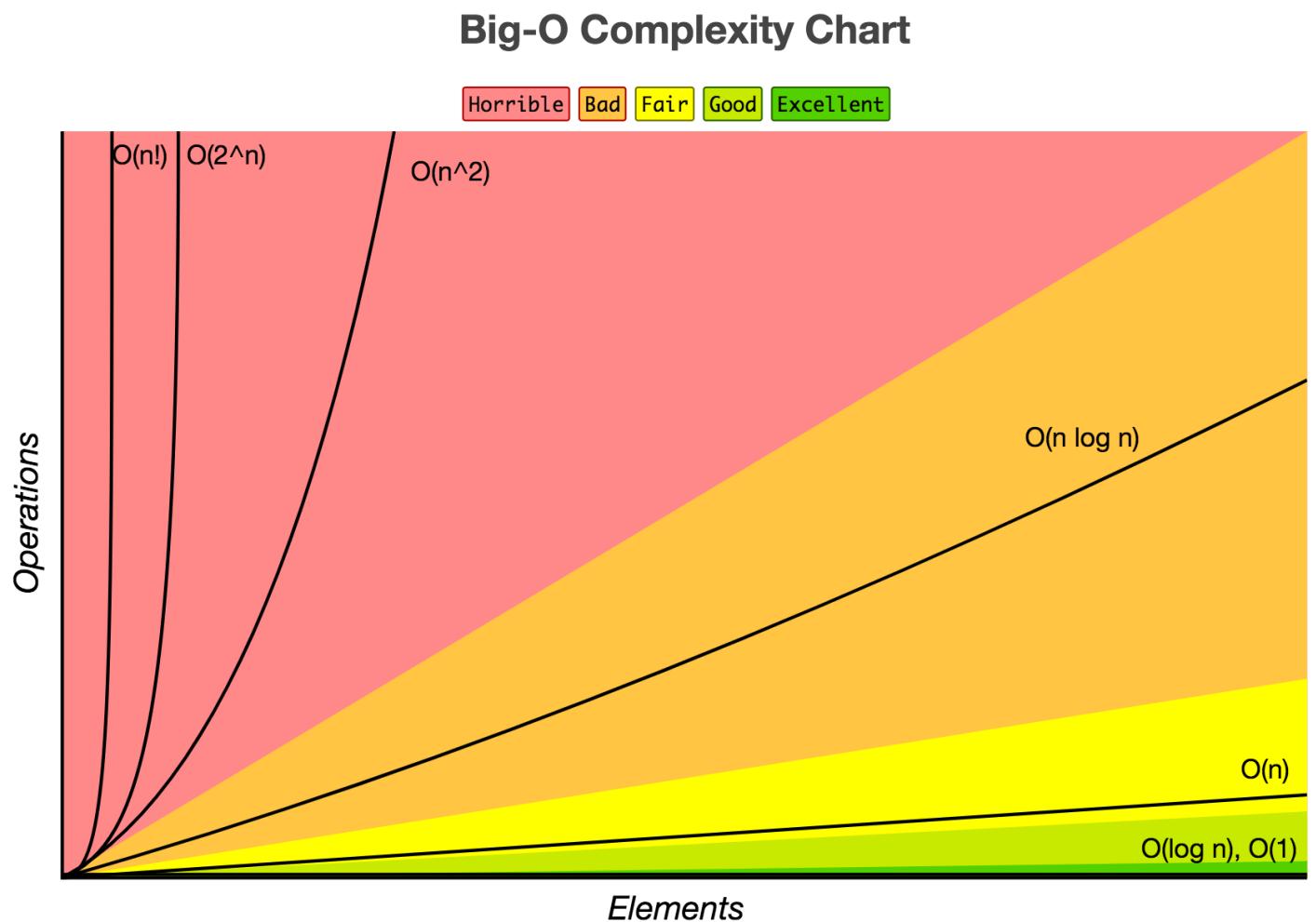


Pyspark Cheat Sheet

Big-O



Big-O Algorithm Cheat Sheet



Big-O Algorithm Complexity Chart

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity	
	Average				Worst					
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion		
Array	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	
Stack	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	
Queue	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	
Singly-Linked List	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	
Doubly-Linked List	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	
Skip List	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n \log(n))$	
Hash Table	N/A	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	N/A	$O(n)$	$O(n)$	$O(n)$	$O(n)$	
Binary Search Tree	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	
Cartesian Tree	N/A	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	N/A	$O(n)$	$O(n)$	$O(n)$	$O(n)$	
B-Tree	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$	
Red-Black Tree	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$	
Splay Tree	N/A	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	N/A	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$	
AVL Tree	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$	
KD Tree	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	

BIG-O Algorithm Data Structure Operations

Array Sorting Algorithms

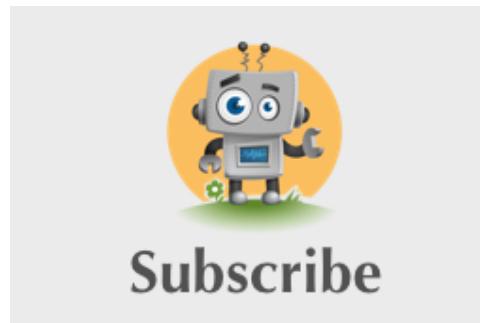
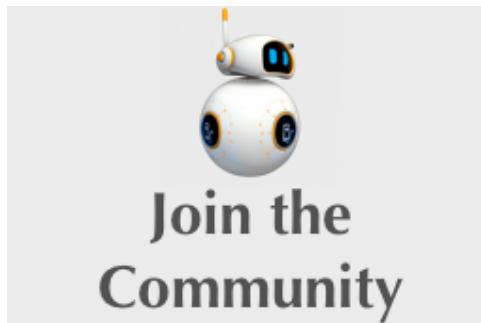
Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	
<u>Quicksort</u>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$	$O(\log(n))$
<u>Mergesort</u>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$
<u>Timsort</u>	$\Omega(n)$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$
<u>Heapsort</u>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(1)$
<u>Bubble Sort</u>	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
<u>Insertion Sort</u>	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
<u>Selection Sort</u>	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
<u>Tree Sort</u>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$	$O(n)$
<u>Shell Sort</u>	$\Omega(n \log(n))$	$\Theta(n(\log(n))^2)$	$O(n(\log(n))^2)$	$O(1)$
<u>Bucket Sort</u>	$\Omega(n+k)$	$\Theta(n+k)$	$O(n^2)$	$O(n)$
<u>Radix Sort</u>	$\Omega(nk)$	$\Theta(nk)$	$O(nk)$	$O(n+k)$
<u>Counting Sort</u>	$\Omega(n+k)$	$\Theta(n+k)$	$O(n+k)$	$O(k)$
<u>Cubesort</u>	$\Omega(n)$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$

Big-O Array Sorting Algorithms

About Stefan

Stefan is the founder of [Chatbot's Life](#), a Chatbot media and consulting firm. Chatbot's Life has grown to over 150k views per month and has become the premium place to learn about Bots & AI online. Chatbot's Life has also consulted many of the top Bot companies like Swelly, Instavest, OutBrain, NearGroup and a number of Enterprises.

176 29



Resources

Big-O Algorithm Cheat Sheet: <http://bigocheatsheet.com/>

Bokeh Cheat Sheet:

https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_Bokeh_Cheat_Sheet.pdf

Data Science Cheat Sheet:

<https://www.datacamp.com/community/tutorials/python-data-science-cheat-sheet-basics>

Data Wrangling Cheat Sheet: <https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf>

Data Wrangling: https://en.wikipedia.org/wiki/Data_wrangling

Ggplot Cheat Sheet: <https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf>

Keras Cheat Sheet:

<https://www.datacamp.com/community/blog/keras-cheat-sheet#gs.DRKeNMs>

Keras: <https://en.wikipedia.org/wiki/Keras>

Machine Learning Cheat Sheet: <https://ai.icymi.email/new-machinelearning-cheat-sheet-by-emily-barry-abdsc/>

Machine Learning Cheat Sheet: <https://docs.microsoft.com/en-in/azure/machine-learning/machine-learning-algorithm-cheat-sheet>

ML Cheat Sheet:: <http://peekaboo-vision.blogspot.com/2013/01/machine-learning-cheat-sheet-for-scikit.html>

Matplotlib Cheat Sheet:

<https://www.datacamp.com/community/blog/python-matplotlib-cheat-sheet#gs.uEKySpY>

Matplotlib: <https://en.wikipedia.org/wiki/Matplotlib>

Neural Networks Cheat Sheet: <http://www.asimovinstitute.org/neural-network-zoo/>

Neural Networks Graph Cheat Sheet:

<http://www.asimovinstitute.org/blog/>

Neural Networks: <https://www.quora.com/Where-can-find-a-cheat-sheet-for-neural-network>

Numpy Cheat Sheet:

<https://www.datacamp.com/community/blog/python-numpy-cheat-sheet#gs.AK5ZBgE>

NumPy: <https://en.wikipedia.org/wiki/NumPy>

Pandas Cheat Sheet:

<https://www.datacamp.com/community/blog/python-pandas-cheat-sheet#gs.oundfxM>

Pandas: [https://en.wikipedia.org/wiki/Pandas_\(software\)](https://en.wikipedia.org/wiki/Pandas_(software))

Pandas Cheat Sheet:

<https://www.datacamp.com/community/blog/pandas-cheat-sheet-python#gs.HPFoRIc>

Pyspark Cheat Sheet:

<https://www.datacamp.com/community/blog/pyspark-cheat-sheet-python#gs.L=J1zxQ>

Scikit Cheat Sheet:

<https://www.datacamp.com/community/blog/scikit-learn-cheat-sheet>

Scikit-learn: <https://en.wikipedia.org/wiki/Scikit-learn>

Scikit-learn Cheat Sheet: <http://peekaboo-vision.blogspot.com/2013/01/machine-learning-cheat-sheet-for-scikit.html>

Scipy Cheat Sheet:

<https://www.datacamp.com/community/blog/python-scipy-cheat-sheet#gs.JDSg3OI>

SciPy: <https://en.wikipedia.org/wiki/SciPy>

TesorFlow Cheat Sheet: <https://www.altoros.com/tensorflow-cheat-sheet.html>

Tensor Flow: <https://en.wikipedia.org/wiki/TensorFlow>

