

# Concise Cheat Sheets of Machine Learning with Python (and Maths)

Machine learning is difficult for beginners. As well as libraries for Machine Learning in python are difficult to understand.

Over the past few weeks, I have been collecting Machine Learning cheat sheets from different sources and to make things more interesting and give context, I added excerpts for each major topic.

If you are just getting started with Machine Learning or Data Science, you'll richly benefit from resources compiled from our recent publications;

The [Best Machine Learning Courses](#) on the Internet

Learn [Math for Machine Learning](#) from the World-Class Educators.

[Machine Learning with Python](#)

## The Best Machine Learning Cheat Sheets

Here's a curated a list of Machine Learning Cheat Sheets and most commonly used MLear Libraries in Python. You'll be able to download them with ease and grasp the fundamentals for long-term benefits.

---

### 1. [Scikit-Learn Cheat Sheet: Python Machine Learning](#)

This scikit-learn cheat sheet will introduce you to the basic steps that you need to go through to implement machine learning algorithms successfully.

# Python For Data Science Cheat Sheet

## Scikit-Learn

Learn Python for data science interactively at [www.DataCamp.com](http://www.DataCamp.com)



### Scikit-learn

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



#### A Basic Example

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

#### Loading The Data

Also see NumPy & Pandas

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

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

#### Training And Test Data

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

### Preprocessing The Data

#### Standardization

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

#### Normalization

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

#### Binarization

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

#### 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)
```

### Create Your Model

#### Supervised Learning Estimators

##### Linear Regression

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

##### Support Vector Machines (SVM)

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

##### Naive Bayes

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

##### KNN

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

#### Unsupervised Learning Estimators

##### Principal Component Analysis (PCA)

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

##### K Means

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

### Model Fitting

#### Supervised learning

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

Fit the model to the data

#### Unsupervised Learning

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

Fit the model to the data

Fit to data, then transform it

### Prediction

#### Supervised Estimators

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

Predict labels

Predict labels

Estimate probability of a label

#### Unsupervised Estimators

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

Predict labels in clustering algos

### Evaluate Your Model's Performance

#### Classification Metrics

##### Accuracy Score

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

Estimator score method  
Metric scoring functions

##### Classification Report

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

Precision, recall, f1-score  
and support

##### Confusion Matrix

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

#### Regression Metrics

##### Mean Absolute Error

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

##### Mean Squared Error

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

##### R<sup>2</sup> 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



You can [Download Pdf](#) here

## 2. Python Cheat Sheet for Scikit-learn

This scikit-learn cheat sheet is designed for the one who has already started learning about the Python package but wants a handy reference sheet.

# PYTHON FOR DATA SCIENCE CHEAT SHEET

Learn Python for Data Science at [www.edureka.co](http://www.edureka.co)

## 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, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
>>> 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)
```



PYTHON FOR DATA SCIENCE

## Scikit-learn

### Standardization

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

### Normalization

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

### Binarization

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

## Create Your Model

### Supervised Learning Estimators

```
Linear Regression
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
Support Vector Machines (SVM)
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
Naive Bayes
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
KNN
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

### Unsupervised Learning Estimators

```
Principal Component Analysis (PCA)
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
K Means
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

## Model Fitting

```
Supervised learning
>>> lr.fit(X, y) #Fit the model to the data
>>> knn.fit(X_train, y_train)
>>> svc.fit(X_train, y_train)
Unsupervised Learning
>>> k_means.fit(X_train) #Fit the model to the data
>>> pca_model = pca.fit_transform(X_train) #Fit to data, then transform it
```

## Prediction

```
Supervised Estimators
>>> y_pred = svc.predict(np.random.random((2,5))) Predict labels
>>> y_pred = lr.predict(X_test) Predict labels
>>> y_pred = knn.predict_proba(X_test) Estimate probability of a label
Unsupervised Estimators
>>> y_pred = k_means.predict(X_test) Predict labels in clustering algo
```

## Evaluate Your Model's Performance

### Classification Metrics

```
Accuracy Score
>>> knn.score(X_test, y_test) Estimator score method
>>> from sklearn.metrics import accuracy_score Metric scoring functions
>>> accuracy_score(y_test, y_pred)
Classification Report
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
Confusion Matrix
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
Precision, recall, f1-score and support
```

### Regression Metrics

```
Mean Absolute Error
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
Mean Squared Error
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
R2 Score
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

### Clustering Metrics

```
Adjusted Rand Index
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
Homogeneity
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
V-measure
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

### Cross-Validation

```
Adjusted Rand Index
>>> 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(1,3), '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_)
```

You can [Download Pdf here](#)

## 3. Keras Cheat Sheet: Neural Networks in Python

This Keras Cheat Sheet will boost your journey with deep learning in Python: you'll have pre-processed, created, validated and tuned your deep learning models in no time thanks to the code examples!

# Python For Data Science Cheat Sheet

## Keras

Learn Python for data science interactively at [www.DataCamp.com](http://www.DataCamp.com)



### Keras

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

#### A Basic Example

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

#### Data

Also see NumPy, Pandas & Scikit-Learn

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

#### Keras Data Sets

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

#### Other

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

### Preprocessing

#### Sequence Padding

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

#### One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

### Model Architecture

#### Sequential Model

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

#### Multilayer Perceptron (MLP)

##### Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
>>>                 input_dim=8,
>>>                 kernel_initializer='uniform',
>>>                 activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(512,kernel_initializer='uniform',activation='sigmoid'))
```

##### Multi-Class Classification

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

##### Regression

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

#### Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Conv2D(64,(3,3),padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dense(1000))
>>> model2.add(Activation('relu'))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

#### Recurrent Neural Network (RNN)

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

#### Train and Test Sets

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

#### Standardization/Normalization

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

### Inspect Model

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

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

### Compile Model

#### MLP: Binary Classification

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

#### MLP: Multi-Class Classification

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

#### MLP: Regression

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

#### Recurrent Neural Network

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

### Model Training

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

### Evaluate Your Model's Performance

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

### Prediction

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

### Save/Reload Models

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

### Model Fine-tuning

#### Optimization Parameters

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

#### Early Stopping

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

DataCamp

Learn Python for Data Science interactively



You can [Download Pdf here](#)

## 4. Python SciPy Cheat Sheet

This SciPy cheat sheet covers the basics of linear algebra that you need to get started and learn how you can use it to interact with NumPy, and goes on to summarize important topics in linear algebra for Machine Learning and Data Science.

# Python For Data Science Cheat Sheet

## SciPy - Linear Algebra

Learn More Python for Data Science [interactively at www.datacamp.com](https://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,-1:1:10]    Stack arrays vertically (row-wise)
>>> np.c_[b,c]                Create stacked column-wise arrays
```

### Shape Manipulation

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

### Polynomials

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

### Vectorizing Functions

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

### Type Handling

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

### Other Useful Functions

```
>>> np.angle(b,deg=True)      Return the angle of the complex argument
>>> g = np.linspace(0,np.pi,num=5) Create an array of evenly spaced values
>>>                                     (number of samples)
>>> g[3:] += np.pi
>>> np.unwrap(g)              Unwrap
>>> np.linspace(0,10,3)        Create an array of evenly spaced values (log scale)
>>> np.select([c<4],[c*2])    Return values from a list of arrays depending on
>>>                                     conditions
>>> misc.factorial(a)          Factorial
>>> misc.comb(10,3,exact=True) Combinatorial
>>> misc.central_diff_weights(3) Weights for Np-point central derivative
>>> misc.derivative(myfunc,1,0) Find the n-th derivative of a function at a point
```

## Linear Algebra

Also see NumPy

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

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

### Creating Matrices

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

### Basic Matrix Routines

<b>Inverse</b> >>> A.I >>> linalg.inv(A)	Inverse Inverse
<b>Transposition</b> >>> A.T >>> A.H	Transpose matrix Conjugate transposition
<b>Trace</b> >>> np.trace(A)	Trace
<b>Norm</b> >>> linalg.norm(A) >>> linalg.norm(A,1) >>> linalg.norm(A,np.inf)	Frobenius norm L1 norm (max column sum) Linf norm (max row sum)
<b>Rank</b> >>> np.linalg.matrix_rank(C)	Matrix rank
<b>Determinant</b> >>> linalg.det(A)	Determinant
<b>Solving linear problems</b> >>> linalg.solve(A,b) >>> E = np.mat(a).T >>> linalg.lstsq(F,E)	Solver for dense matrices Solver for dense matrices Least-squares solution to linear matrix equation
<b>Generalized inverse</b> >>> linalg.pinv(C) >>> linalg.pinv2(C)	Compute the pseudo-inverse of a matrix (least-squares solver) Compute the pseudo-inverse of a matrix (SVD)

### Creating Sparse Matrices

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

### Sparse Matrix Routines

<b>Inverse</b> >>> sparse.linalg.inv(I)	Inverse
<b>Norm</b> >>> sparse.linalg.norm(I)	Norm
<b>Solving linear problems</b> >>> sparse.linalg.spsolve(H,I)	Solver for sparse matrices

### Sparse Matrix Functions

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

### Asking For Help

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

### Matrix Functions

<b>Addition</b> >>> np.add(A,D)	Addition
<b>Subtraction</b> >>> np.subtract(A,D)	Subtraction
<b>Division</b> >>> np.divide(A,D)	Division
<b>Multiplication</b> >>> A @ D >>> np.multiply(D,A) >>> np.dot(A,D) >>> np.vdot(A,D) >>> np.inner(A,D) >>> np.outer(A,D) >>> np.tensordot(A,D) >>> np.kron(A,D)	Multiplication operator (Python 3) Multiplication Dot product Vector dot product Inner product Outer product Tensor dot product Kronecker product
<b>Exponential Functions</b> >>> linalg.expm(A) >>> linalg.expm2(A) >>> linalg.expm3(D)	Matrix exponential Matrix exponential (Taylor Series) Matrix exponential (eigenvalue decomposition)
<b>Logarithm Function</b> >>> linalg.logm(A)	Matrix logarithm
<b>Trigonometric Functions</b> >>> linalg.sirm(D) >>> linalg.cosm(D) >>> linalg.tanm(A)	Matrix sine Matrix cosine Matrix tangent
<b>Hyperbolic Trigonometric Functions</b> >>> linalg.coshm(D) >>> linalg.tanhm(A)	Hyperbolic matrix sine Hyperbolic matrix cosine Hyperbolic matrix tangent
<b>Matrix Sign Function</b> >>> np.signm(A)	Matrix sign function
<b>Matrix Square Root</b> >>> linalg.sqrtm(A)	Matrix square root
<b>Arbitrary Functions</b> >>> linalg.fnum(A, lambda x: x*x)	Evaluate matrix function

### Decompositions

<b>Eigenvalues and Eigenvectors</b> >>> la, v = linalg.eig(A) >>> l1, l2 = la >>> v[:,0] >>> v[:,1] >>> linalg.eigvals(A)	Solve ordinary or generalized eigenvalue problem for square matrix Unpack eigenvalues First eigenvector Second eigenvector Unpack eigenvalues
<b>Singular Value Decomposition</b> >>> U,s,Vh = linalg.svd(B) >>> M,N = B.shape >>> Sig = linalg.diagsvd(s,M,N)	Singular Value Decomposition (SVD) Construct sigma matrix in SVD
<b>LU Decomposition</b> >>> 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
```

You can [Download Pdf here](#)

## 5. Theano Cheat Sheet

Theano is the powerful deep learning library in python and this Cheat Sheet includes the most common ways to implement high-level neural networks API to develop and evaluate machine learning models. [Download PDF Version using the link below for the complete set of Theano Cheat Sheet.](#)

## Theano cheatsheet

(for detailed information on Theano see: <http://deeplearning.net/software/theano>)

### Imports:

- `import theano #main module`
- `from theano import tensor as T #variables & vector/matrix/tensor operations`

### Symbolic variables

- `X = T.matrix()`
- scalar, vector, matrix, tensor, etc.
- Type can be specified by using i/f/d for integer/float32/double: `X = T.ivector()`

### Operations with symbolic variables

- Basic elementwise operations: `+`, `-`, `*`, `/`, `**`, etc.
- Dot product: `T.dot(A, B)` or `A.dot(B)`
- Aggregations: `T.min()`, `T.max()`, `T.mean()`, `T.sum()`, etc.
  - axis: only aggregate along the specified dimension
    - Maximum per column of matrix A: `T.max(A, axis=0)`
    - Sum of rows of matrix A: `T.sum(A, axis=1)`
- Reshaping
  - `A.reshape((10, 1))`, `A.reshape((10))`, `A.reshape((2, 5))`, etc.
  - `A.dimshuffle((1,0))`, `A.dimshuffle(('x', 0))`, etc.
    - Use already existing dimension indices and use 'x' for a new broadcastable dimension (e.g. ('x', 0) creates a matrix row (2D) from a 1D vector)
  - `A.T` #transpose
- Subtensor operators
  - Indexing as in numpy (Might be slow on GPU! Indexing 2D matrices by row is fast.)
  - Slicing similar to numpy
  - `T.set_subtensor(subtensor, value)`
  - `T.inc_subtensor(subtensor, inc)`

### Shared variables

- Variables with persistent value
- Ideal for model parameters
- `w = theano.shared(np.random.rand(100).astype('float32'))`
  - Initialized with numpy arrays (determines shape, size, type and starting values)
    - In the example above: a vector of 100 float32 values
  - Optional name parameter
- Can be used as a variable in symbolic expressions
- Any operation with shared variables results in a symbolic expression/variable
- `W.get_value()`, `W.set_value(new_value)`
  - Getting/setting the value of a symbolic variable
  - Use only when necessary
  - borrow parameter

## You can [Download Pdf here](#)

---

*Thanks for making it to the end* ☐

Share & Learn! If you liked this article, i've got a few practical reads for you. One about the Best [TensorFlow Courses](#) and One about the Best [Deep Learning Courses](#) on the Internet.

Also, don't miss out on our collection of best available data-centric [Python Cheat Sheets](#) or give these [Python Podcasts](#) a listen to build functional knowledge in Python.

I've also got this [Data Science newsletter](#) that you might be into. I send a tiny email once or twice every quarter with some useful resource I've found.

Don't worry, I hate spam as much as you. Feel free to subscribe.