

Python For Data Science

NumPy Cheat Sheet

Learn NumPy online at www.DataCamp.com

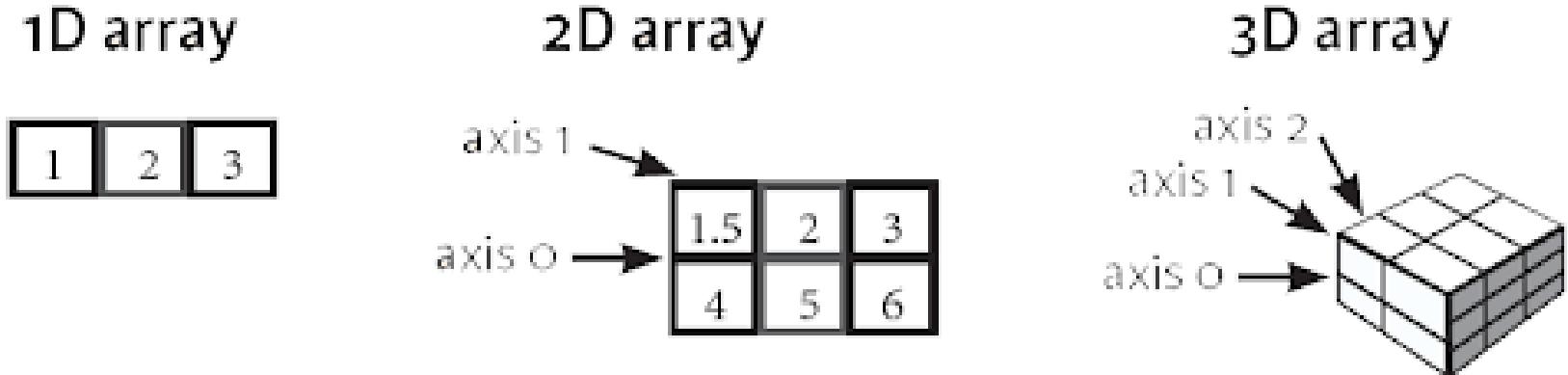
Numpy

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

Use the following import convention:

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

NumPy Arrays



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([(1.5,2,3), (4,5,6)],[(3,2,1), (4,5,6)]), dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4)) #Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16) #Create an array of ones
>>> d = np.arange(10,25,5) #Create an array of evenly spaced values (step value)
>>> np.linspace(0,2,9) #Create an array of evenly spaced values (number of samples)
>>> e = np.full((2,2),7) #Create a constant array
>>> f = np.eye(2) #Create a 2x2 identity matrix
>>> np.random.random((2,2)) #Create an array with random values
>>> np.empty((3,2)) #Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savetxt('array.npy', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Asking For Help

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

Inspecting Your Array

```
>>> a.shape #Array dimensions
>>> len(a) #Length of array
>>> b.ndim #Number of array dimensions
>>> e.size #Number of array elements
>>> b.dtype #Data type of array elements
>>> b.dtype.name #Name of data type
>>> b.astype(int) #Convert an array to a different type
```

Data Types

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

Array Mathematics

Arithmetic Operations

```
>>> g = a - b #Subtraction
array([[-0.5, 0. , 0. ],
       [-3. , -3. , -3. ]])
>>> np.subtract(a,b) #Subtraction
>>> b + a #Addition
array([[ 2.5, 4. , 6. ],
       [ 5. , 7. , 9. ]])
>>> np.add(b,a) Addition
>>> a / b #Division
array([[ 0.66666667, 1. , 1. ],
       [ 0.25 , 0.4 , 0.5 ]])
>>> np.divide(a,b) #Division
>>> a * b #Multiplication
array([[ 1.5, 4. , 9. ],
       [ 4. , 10. , 18. ]])
>>> np.multiply(a,b) #Multiplication
>>> np.exp(b) #Exponentiation
>>> np.sqrt(b) #Square root
>>> np.sin(a) #Print sines of an array
>>> np.cos(b) #Element-wise cosine
>>> np.log(a) #Element-wise natural logarithm
>>> e.dot(f) #Dot product
array([[ 7., 7.],
       [ 7., 7.]])
```

Comparison

```
>>> a == b #Element-wise comparison
array([[False, True, True],
       [False, False, False]], dtype=bool)
>>> a < 2 #Element-wise comparison
array[[True, False, False], dtype=bool]
>>> np.array_equal(a, b) #Array-wise comparison
```

Aggregate Functions

```
>>> a.sum() #Array-wise sum
>>> a.min() #Array-wise minimum value
>>> b.max(axis=0) #Maximum value of an array row
>>> b.cumsum(axis=1) #Cumulative sum of the elements
>>> a.mean() #Mean
>>> np.median(b) #Median
>>> np.correlcoef(a) #Correlation coefficient
>>> np.std(b) #Standard deviation
```

Copying Arrays

```
>>> h = a.view() #Create a view of the array with the same data
>>> np.copy(a) #Create a copy of the array
>>> h = a.copy() #Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort() #Sort an array
>>> c.sort(axis=0) #Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Subsetting

```
>>> a[2] #Select the element at the 2nd index
3
>>> b[1,2] #Select the element at row 1 column 2 (equivalent to b[1][2])
6.0
```

1	2	3
1.5	2	3
4	5	6

Slicing

```
>>> a[0:2] #Select items at index 0 and 1
array([1, 2])
>>> b[0:2,1] #Select items at rows 0 and 1 in column 1
array([ 2., 2.])
>>> b[:,1] #Select all items at row 0 (equivalent to b[0:, 1])
array([[1.5, 2., 3.]])
>>> c[1,...] #Same as [1,:,:]
array([[ 3., 2., 1.],
       [ 4., 5., 6.]])
>>> a[ : :-1] #Reversed array a array([3, 2, 1])
```

1	2	3
1.5	2	3
4	5	6

Boolean Indexing

```
>>> a[a<2] #Select elements from a less than 2
array([1])
```

1	2	3
---	---	---

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]] #Select elements (1,0),(0,1),(1,2) and (0,0)
array([ 4. , 2. , 6. , 1.5])
>>> b[[1, 0, 1, 0]][:, [0,1,2,0]] #Select a subset of the matrix's rows and columns
array([[ 4. , 5. , 6. , 4. ],
       [ 1.5, 2. , 3. , 1.5],
       [ 4. , 5. , 6. , 4. ],
       [ 1.5, 2. , 3. , 1.5]])
```

1	2	3
---	---	---

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b) #Permute array dimensions
>>> i.T #Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel() #Flatten the array
>>> g.reshape(3,-2) #Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6)) #Return a new array with shape (2,6)
>>> np.append(h,g) #Append items to an array
>>> np.insert(a, 1, 5) #Insert items in an array
>>> np.delete(a,[1]) #Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0) #Concatenate arrays
array([[ 1., 2., 3., 10.],
       [ 2., 15.],
       [ 3., 20.]])
>>> np.vstack((a,b)) #Stack arrays vertically (row-wise)
array([[ 1., 2., 3.],
       [ 1.5, 2., 3.],
       [ 4., 5., 6.]]))
>>> np.r_[e,f] #Stack arrays vertically (row-wise)
>>> np.hstack((e,f)) #Stack arrays horizontally (column-wise)
array([[ 7., 7., 1., 0.],
       [ 7., 7., 0., 1.]])
>>> np.column_stack((a,d)) #Create stacked column-wise arrays
array([[ 1, 10],
       [ 2, 15],
       [ 3, 20]])
>>> np.c_[a,d] #Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3) #Split the array horizontally at the 3rd index
[array([1]),array([2]),array([3])]
>>> np.vsplit(c,2) #Split the array vertically at the 2nd index
[array([[ 1.5, 2. , 1. ]]),
 [ 4. , 5. , 6. ]])
array([[[ 3., 2., 3.]],
 [ 4., 5., 6. ]])
```



Python For Data Science

Pandas Basics Cheat Sheet

Learn Pandas Basics online 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

a	3
b	-5
c	7
d	4

Index →

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

Dataframe

A **two-dimensional** labeled data structure with columns of potentially different types

	Country	Capital	Population
Index →	Belgium	Brussels	11190846
0	India	New Delhi	1303171035
1	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'])
```

Dropping

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

Asking For Help

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

Sort & Rank

```
>>> df.sort_index() #Sort by labels along an axis
>>> df.sort_values(by='Country') #Sort by the values along an axis
>>> df.rank() #Assign ranks to entries
```

> I/O

Read and Write to CSV

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

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> df.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')

Read multiple sheets from the same file
>>> xlsx = pd.ExcelFile('file.xlsx')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

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

read_sql() is a convenience wrapper around read_sql_table() and read_sql_query()
>>> df.to_sql('myDF', engine)
```

> Selection

Also see NumPy Arrays

Getting

```
>>> s['b'] #Get one element
-5
>>> df[1:] #Get subset of a DataFrame
   Country Capital Population
1 India New Delhi 1303171035
2 Brazil Brasilia 207847528
```

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]] #Select single value by row & column
'Belgium'
>>> df.iat[[0],[0]]
'Belgium'
```

By Label

```
>>> df.loc[[0], ['Country']] #Select single value by row & column labels
'Belgium'
>>> df.at[[0], ['Country']]
'Belgium'
```

By Label/Position

```
>>> df.ix[2] #Select single row of subset of rows
Country Brazil
Capital Brasilia
Population 207847528
>>> df.ix[:, 'Capital'] #Select a single column of subset of columns
0 Brussels
1 New Delhi
2 Brasilia
>>> df.ix[1, 'Capital'] #Select rows and columns
'New Delhi'
```

Boolean Indexing

```
>>> s[~(s > 1)] #Series s where value is not >1
>>> s[(s < -1) | (s > 2)] #s where value is <-1 or >2
>>> df[df['Population']>1200000000] #Use filter to adjust DataFrame
```

Setting

```
>>> s['a'] = 6 #Set index a of Series s to 6
```

> Retrieving Series/DataFrame Information

Basic Information

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

Summary

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

> Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f) #Apply function
>>> df.applymap(f) #Apply function element-wise
```

> Data Alignment

Internal Data Alignment

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

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

Arithmetic Operations with Fill Methods

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

```
>>> s.add(s3, fill_value=0)
a 10.0
b -5.0
c 5.0
d 7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

Learn Data Skills Online at
www.DataCamp.com

Python For Data Science

Scikit-Learn Cheat Sheet

Learn Scikit-Learn online at www.DataCamp.com

Scikit-learn

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



A Basic Example

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

> Loading The Data

Also see NumPy & Pandas

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

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

> 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

```
>>> kmeans.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 = kmeans.predict(X_test) #Predict labels in clustering algos
```

> 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
>>> kmeans = KMeans(n_clusters=3, random_state=0)
```

> 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 #Precision, recall, f1-score and support
>>> print(classification_report(y_test, y_pred))
```

Confusion Matrix

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

Regression Metrics

Mean Absolute Error

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

Mean Squared Error

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

R² Score

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

Clustering Metrics

Adjusted Rand Index

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

Homogeneity

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

V-measure

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

Cross-Validation

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

> Tune Your Model

Grid Search

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

Randomized Parameter Optimization

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



Learn Data Skills Online at www.DataCamp.com