

# Python For Data Science Cheat Sheet

## Scikit-Learn

### Scikit-learn

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



#### A Basic Example

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

### Loading The Data

Also see NumPy & Pandas

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

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

### Training And Test Data

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

### Preprocessing The Data

#### Standardization

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

#### Normalization

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

#### Binarization

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

### Create Your Model

#### Supervised Learning Estimators

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

Support Vector Machines (SVM)
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')

Naïve 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

<b>Supervised learning</b> >>> lr.fit(X, y) >>> knn.fit(X_train, y_train) >>> svc.fit(X_train, y_train)	Fit the model to the data
<b>Unsupervised Learning</b> >>> 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

<b>Supervised Estimators</b> >>> 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
<b>Unsupervised Estimators</b> >>> y_pred = k_means.predict(X_test)	Predict labels in clustering algos

### Evaluate Your Model's Performance

#### Classification Metrics

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

#### Regression Metrics

<b>Mean Absolute Error</b> >>> from sklearn.metrics import mean_absolute_error >>> y_true = [3, -0.5, 2] >>> mean_absolute_error(y_true, y_pred)	
<b>Mean Squared Error</b> >>> from sklearn.metrics import mean_squared_error >>> mean_squared_error(y_true, y_pred)	
<b>R<sup>2</sup> Score</b> >>> from sklearn.metrics import r2_score >>> r2_score(y_true, y_pred)	

#### Clustering Metrics

<b>Adjusted Rand Index</b> >>> from sklearn.metrics import adjusted_rand_score >>> adjusted_rand_score(y_true, y_pred)	
<b>Homogeneity</b> >>> from sklearn.metrics import homogeneity_score >>> homogeneity_score(y_true, y_pred)	
<b>V-measure</b> >>> 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(1,5),
            "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_)
```