

# Scikit-Learn Made Easy: API Fast Guide

Mohamad Yamen AL Mohamad  
`yamenmohamad@tabrizu.ac.ir`

May 29, 2025

## Abstract

This document provides a comprehensive reference guide to the core modules of `scikit-learn`, the premier machine learning library for Python. Each section introduces a module, explains its purpose, and details key APIs with itemizes, use cases, and features. Visual diagrams illustrate data flow and relationships between components, while practical code examples demonstrate real-world applications. The guide covers major functionalities including supervised learning (classification, regression), unsupervised learning (clustering, dimensionality reduction), model selection, preprocessing, and pipeline construction. It is intended for data scientists, machine learning practitioners, and developers who need a structured overview of scikit-learn's capabilities. By combining clear explanations with practical examples, this document helps users understand and implement effective machine learning workflows using scikit-learn's consistent API design. **Keywords:** Scikit-learn, Machine Learning, Python, Classification, Regression, Clustering, Dimensionality Reduction, Model Selection, Preprocessing, Pipelines, Feature Extraction, Cross-Validation, Hyperparameter Tuning, Supervised Learning, Unsupervised Learning

# 1 Overview

**Scikit-learn** [1, 2] is a comprehensive Python library for machine learning that provides simple and efficient tools for data mining and data analysis. Built on NumPy, SciPy, and matplotlib, it features various classification, regression, clustering algorithms, and includes utilities for model evaluation, data preprocessing, and pipeline construction.

This document serves as a structured reference to scikit-learn’s core modules:

- Clear definitions of each module’s purpose
- Detailed explanations of key APIs and functions
- Visual diagrams illustrating concepts and workflows
- Practical code examples demonstrating real-world usage

Scikit-learn’s consistent API design enables practitioners to:

- Quickly implement and compare different algorithms
- Build end-to-end machine learning pipelines
- Evaluate models using robust validation techniques
- Preprocess data efficiently

## 2 Supervised Learning

**Definition 2.1.** *The `sklearn` package provides numerous algorithms for supervised learning, where the goal is to predict target variables based on input features. These include both classification (predicting discrete labels) and regression (predicting continuous values) tasks.*

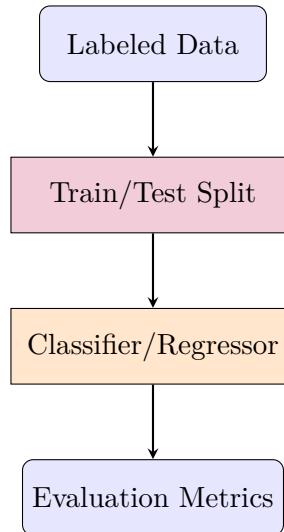


Figure 1: Supervised Learning Workflow

## 2.1 Key APIs

### 2.1.1 Classification

- `LogisticRegression`:

- Purpose: Linear model for binary and multiclass classification
- Use Case: When probabilities of class membership are needed
- Features: L1/L2 regularization, multiclass support

- `SVC` (Support Vector Classification):

- Purpose: Kernel-based classification
- Use Case: Complex decision boundaries in high-dimensional spaces
- Features: Multiple kernel options (linear, poly, rbf, sigmoid)

- `RandomForestClassifier`:

- Purpose: Ensemble of decision trees
- Use Case: Robust classification with feature importance
- Features: Handles missing values, parallel training

#### Example 2.1.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import
    train_test_split
from sklearn.metrics import accuracy_score

# Load dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split into train/test sets
X_train, X_test, y_train, y_test = train_test_split
(X, y, test_size=0.2)

# Train model
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train, y_train)

# Evaluate
y_pred = clf.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred)
      :.2f}")
```

### 2.1.2 Regression

- `LinearRegression`:

- Purpose: Ordinary least squares regression
- Use Case: Linear relationships between features and target
- Features: Fast training, interpretable coefficients

- Ridge:

- Purpose: L2-regularized linear regression
- Use Case: When features are correlated
- Features: Reduces overfitting through regularization

- SVR (Support Vector Regression):

- Purpose: Kernel-based regression
- Use Case: Non-linear relationships
- Features: Robust to outliers with proper kernel choice

### Example 2.2.

```
from sklearn.linear_model import Ridge
from sklearn.datasets import make_regression

# Generate synthetic regression data
X, y = make_regression(n_features=10, noise=0.1)

# Train model
ridge = Ridge(alpha=1.0)
ridge.fit(X, y)

# Predict and evaluate
score = ridge.score(X, y)
print(f"R^2 Score: {score:.2f}")
```

## 3 Unsupervised Learning

**Definition 3.1.** The `sklearn` package provides algorithms for unsupervised learning tasks where the data has no labels. These include clustering (grouping similar data points) and dimensionality reduction (reducing the number of features while preserving structure).

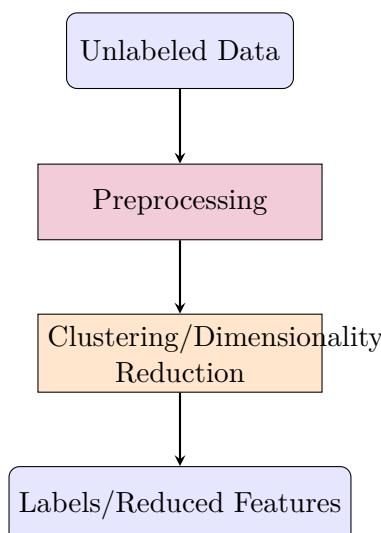


Figure 2: Unsupervised Learning Workflow

## 3.1 Key APIs

### 3.1.1 Clustering

- KMeans:

- Purpose: Partition data into k clusters
- Use Case: When number of clusters is known
- Features: Scalable, simple interpretation

- DBSCAN:

- Purpose: Density-based clustering
- Use Case: Clusters of arbitrary shape
- Features: Handles noise, no need to specify cluster count

- AgglomerativeClustering:

- Purpose: Hierarchical clustering
- Use Case: When hierarchy is important
- Features: Dendrogram visualization possible

#### Example 3.1.

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Generate synthetic data
X, _ = make_blobs(n_samples=300, centers=4,
random_state=42)

# Cluster data
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
labels = kmeans.labels_

# Visualize
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.title("KMeans Clustering")
plt.show()
```

### 3.1.2 Dimensionality Reduction

- PCA (Principal Component Analysis):

- Purpose: Linear dimensionality reduction
- Use Case: Feature extraction, visualization
- Features: Preserves variance, orthogonal components

- t-SNE:

- Purpose: Non-linear dimensionality reduction
- Use Case: Visualization of high-dimensional data
- Features: Preserves local structure

- TruncatedSVD:

- Purpose: Dimensionality reduction for sparse matrices
- Use Case: Text data, recommendation systems
- Features: Works with sparse input

#### Example 3.2.

```
from sklearn.decomposition import PCA
from sklearn.datasets import load_digits

# Load digit images
digits = load_digits()
X = digits.data

# Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Plot
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=digits.target,
            cmap='tab10', alpha=0.6)
plt.title("Digits Projected onto First 2 Principal Components")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.colorbar(label="Digit Class")
plt.show()
```

## 4 Model Selection

**Definition 4.1.** The `sklearn.model_selection` module provides tools for evaluating models and selecting hyperparameters, including cross-validation strategies and hyperparameter search methods.

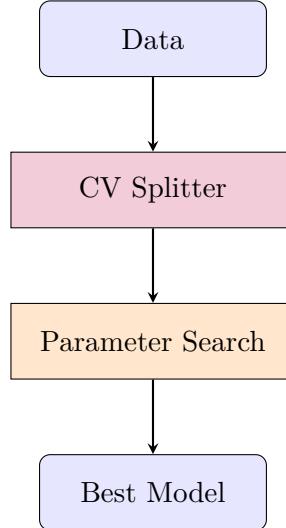


Figure 3: Model Selection Workflow

### 4.1 Key APIs

#### - `train_test_split`:

- Purpose: Split data into train and test sets
- Use Case: Simple model evaluation
- Features: Stratification option for classification

#### - `KFold`:

- Purpose: K-fold cross-validation
- Use Case: Robust model evaluation
- Features: Shuffle option, stratification

#### - `GridSearchCV`:

- Purpose: Exhaustive hyperparameter search
- Use Case: When parameter space is small
- Features: Parallel computation, refitting

#### - `RandomizedSearchCV`:

- Purpose: Randomized hyperparameter search
- Use Case: Large parameter spaces
- Features: More efficient than grid search

**Example 4.1.**

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.datasets import load_iris

# Load data
iris = load_iris()
X, y = iris.data, iris.target

# Define parameter grid
param_grid = {'C': [0.1, 1, 10], 'kernel': ['linear',
                                              'rbf']}

# Perform grid search
grid_search = GridSearchCV(SVC(), param_grid, cv=5)
grid_search.fit(X, y)

# Output results
print(f"Best parameters: {grid_search.best_params_}")
print(f"Best score: {grid_search.best_score_.2f}")
```

## 5 Preprocessing

**Definition 5.1.** *The `sklearn.preprocessing` module provides tools for transforming raw data into formats suitable for machine learning, including scaling, normalization, encoding categorical variables, and feature extraction.*

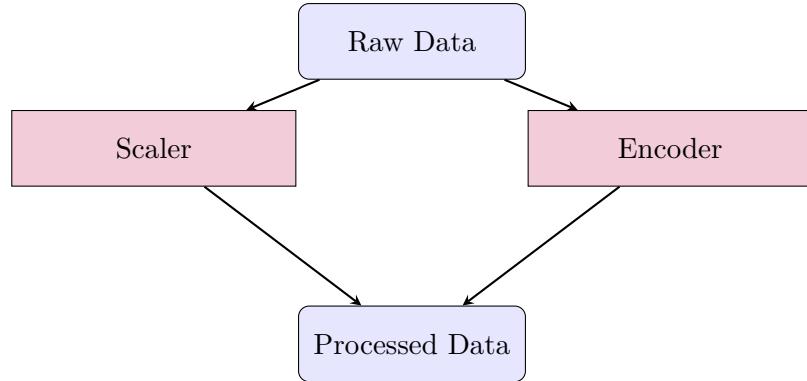


Figure 4: Preprocessing Workflow

### 5.1 Key APIs

#### - StandardScaler:

- Purpose: Standardize features by removing mean and scaling to unit variance
- Use Case: When features have different scales
- Features: Preserves outliers

#### - MinMaxScaler:

- Purpose: Scale features to a given range (default [0, 1])
- Use Case: When bounded features are required
- Features: Sensitive to outliers

#### - OneHotEncoder:

- Purpose: Convert categorical features to binary indicators
- Use Case: When categories have no ordinal relationship
- Features: Handles unknown categories

#### - LabelEncoder:

- Purpose: Encode target labels with value between 0 and n\_classes-1
- Use Case: Preparing classification targets
- Features: Simple transformation

### Example 5.1.

```
from sklearn.preprocessing import StandardScaler,
    OneHotEncoder
from sklearn.compose import ColumnTransformer
import pandas as pd

# Create sample data
data = pd.DataFrame({
    'age': [25, 30, 35],
    'income': [50000, 60000, 70000],
    'gender': ['M', 'F', 'M']
})

# Define preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['age', 'income']),
        ('cat', OneHotEncoder(), ['gender'])
    ])

# Apply transformations
processed = preprocessor.fit_transform(data)
print(processed)
```

## 6 Pipelines

**Definition 6.1.** The `sklearn.pipeline` module provides utilities to chain multiple processing steps together, ensuring proper data flow and preventing data leakage during cross-validation.

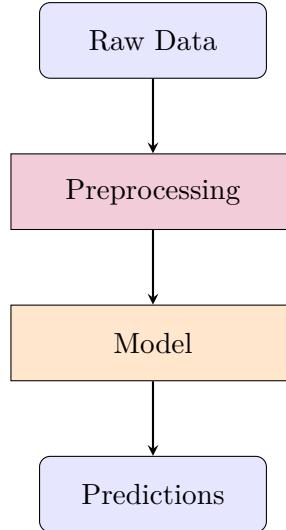


Figure 5: Pipeline Workflow

### 6.1 Key APIs

- Pipeline:

- Purpose: Chain multiple estimators into one
- Use Case: Ensuring proper data flow
- Features: Single interface for all steps

**Example 6.1.**

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import
    train_test_split
from sklearn.datasets import load_breast_cancer

# Load data
X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split
    (X, y, test_size=0.2)

# Create pipeline
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', LogisticRegression())
])

# Fit and evaluate
pipe.fit(X_train, y_train)
print(f"Test Accuracy: {pipe.score(X_test, y_test)
    :.2f}")
```

## References

- [1] Scikit-learn Development Team. *Scikit-learn Documentation*. <https://scikit-learn.org/stable/>, 2025.
- [2] Scikit-learn Contributors. *Scikit-learn GitHub Repository*. <https://github.com/scikit-learn/scikit-learn>, 2025.